Answer Set Solving in Practice

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Potassco Slide Packages are licensed under a Creative Commons Attribution 3.0 Unported License.
Rough Roadmap

1. Motivation
2. Introduction
3. Modeling
4. Language
5. Grounding
6. Foundations
7. Solving
8. Multi-shot solving
9. Theory solving
10. Heuristic programming
11. Systems
12. Advanced modeling
13. Preferences and Optimization
14. Applications
15. Summary
   Bibliography
Resources

- Course material
  - [http://potassco.org/teaching](http://potassco.org/teaching)

- Systems
  - `clasp` [http://potassco.org](http://potassco.org)
  - `clingo` [http://potassco.org](http://potassco.org)
  - `dlv` [http://www.dlvsystem.com](http://www.dlvsystem.com)
  - `wasp` [https://www.mat.unical.it/ricca/wasp](https://www.mat.unical.it/ricca/wasp)
  - `gringo` [http://potassco.org](http://potassco.org)
  - `asparagus` [http://asparagusc.cs.uni-potsdam.de](http://asparagusc.cs.uni-potsdam.de)
The Potassco Book

1. Motivation
2. Introduction
3. Basic modeling
4. Grounding
5. Characterizations
6. Solving
7. Systems
8. Advanced modeling
9. Conclusions

Resources

- http://potassco.org/teaching
The Potassco Book and Guide

1. Motivation
2. Introduction
3. Basic modeling
4. Grounding
5. Characterizations
6. Solving
7. Systems
8. Advanced modeling
9. Conclusions

Resources

- http://potassco.org/teaching
The Potassco Book and Guide

1. Motivation
2. Introduction
3. Basic modeling
4. Grounding
5. Characterizations
6. Solving
7. Systems
8. Advanced modeling
9. Conclusions

Resources

- http://potassco.org/teaching
Literature

Books [?], [?], [?], [?]
Surveys [?], [?], [?], [?]
Articles [?], [?], [?], [?], [?], [?], [?], etc.
Motivation: Overview

1. Motivation
2. Nutshell
3. Evolution
4. Foundation
5. Workflow
6. Engine
7. Usage
8. Summary
1 Motivation
2 Nutshell
3 Evolution
4 Foundation
5 Workflow
6 Engine
7 Usage
8 Summary
“What is the problem?” versus “How to solve the problem?”
Motivation

Informatics

“What is the problem?” versus “How to solve the problem?”

Problem → Computer → Solution → Output
Traditional programming

“What is the problem?” versus “How to solve the problem?”

Problem

Solution

Computer

Output
Traditional programming

“What is the problem?” versus “How to solve the problem?”
Motivation

Declarative problem solving

“What is the problem?” versus “How to solve the problem?”

Problem 

Solution

Computer 

Output

Interpreting

Torsten Schaub (KRR@UP)  
Answer Set Solving in Practice  
February 18, 2019  
8 / 653
Declarative problem solving

“What is the problem?” versus “How to solve the problem?”

- Problem
  - Modeling
    - Representation
  - Solving
- Solution
  - Interpreting
    - Output
Declarative problem solving

“What is the problem?” versus “How to solve the problem?”

Problem

Modeling

Representation

Solving

Solution

Interpreting

Output
Motivation

Traditional Software

User

Program

Problem Solving

Computer

User

Knowledge

Solver
Motivation

Traditional Software

User

Program

Computer

User

Knowledge

Solver

Problem Solving
Motivation

Traditional Software

User

Program

Problem Solving

Computer

User

Knowledge

Solver

Torsten Schaub (KRR@UP)
Traditional Software

Motivation

Programmer

User

Program

Problem Solving

Computer

User

Knowledge

Solver
Motivation

Traditional Software

Programmer

How?

Computer

User

Problem Solving

Knowledge

Solver
Knowledge-driven Software

User

Program

Problem Solving

Solver

Knowledge

Computer
Knowledge-driven Software

User -> Program

Problem Solving

Computer

User

Knowledge

Solver
Knowledge-driven Software

User

Problem Solving

How?

Computer

Knowledge

Solver

User

Torsten Schaub (KRR@UP)
Knowledge-driven Software

User

How?

Problem Solving

Computer

User

What?

How!
Knowledge-driven Software

Motivation

How?

Problem Solving

What?

How!

User

User

How!

How?
Knowledge-driven Software

User (Programmer) → Program → Problem Solving → Knowledge → Solver → Expert (User) → Computer
Motivation

What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability

Knowledge

Solver

Expert
Motivation

What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability

Knowledge

Solver

Expert
Motivation

What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability
What is the benefit?

- Transparency
- Flexibility
- Maintainability
- Reliability

- Generality
- Efficiency
- Optimality
- Availability

Knowledge

Solver

Expert
What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability
Outline

1. Motivation
2. Nutshell
3. Evolution
4. Foundation
5. Workflow
6. Engine
7. Usage
8. Summary
Answer Set Programming (ASP)

What is ASP?

ASP is an approach for declarative problem solving.
Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

- Where is ASP from?
  - Databases
  - Logic programming
  - Knowledge representation and reasoning
  - Satisfiability solving
Answer Set Programming (ASP)

- What is ASP?  
  ASP = DB + LP + KR + SAT!
  ASP is an approach for declarative problem solving

- Where is ASP from?
  - Databases
  - Logic programming
  - Knowledge representation and reasoning
  - Satisfiability solving
Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems

- What problems are this?
  Problems consisting of (many) decisions and constraints
What is ASP?
ASP is an approach for declarative problem solving
What is ASP good for?
Solving knowledge-intense combinatorial (optimization) problems
What problems are this?
Problems consisting of (many) decisions and constraints
Examples Sudoku, Configuration, Diagnosis, Music composition, Planning, System design, Time tabling, etc.
Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems

- What problems are this? — And industrial ones?
  Problems consisting of (many) decisions and constraints
  Examples Sudoku, Configuration, Diagnosis, Music composition, Planning, System design, Time tabling, etc.
Nutshell

Answer Set Programming (ASP)

- **What is ASP?**
  ASP is an approach for declarative problem solving

- **What is ASP good for?**
  Solving knowledge-intense combinatorial (optimization) problems

- **What problems are this? — And industrial ones?**
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - Fcc: Radio frequency auction
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration
Answer Set Programming (ASP)

- **What is ASP?**
  ASP is an approach for declarative problem solving

- **What is ASP good for?**
  Solving knowledge-intense combinatorial (optimization) problems

- **What problems are this? — And industrial ones?**
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - **Fcc: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration
Answer Set Programming (ASP)

- **What is ASP?**
  ASP is an approach for declarative problem solving.

- **What is ASP good for?**
  Solving knowledge-intense combinatorial problems.

- **What problems are this?** — And industrial ones:
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - **Fcc: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration

Over 13 months in 2016–17 the [US Federal Communications Commission](https://www.fcc.gov/) conducted an “incentive auction” to repurpose radio spectrum from broadcast television to wireless internet. In the end, the auction yielded $19.8 billion, $10.05 billion of which was paid to 175 broadcasters for voluntarily relinquishing their licenses across 14 UHF channels. Stations that continued broadcasting were assigned potentially new channels to fit as densely as possible into the channels that remained. The government netted more than $7 billion (used to pay down the national debt) after covering costs. A crucial element of the auction design was the construction of a solver, dubbed SATFC, that determined whether sets of stations could be “repacked” in this way; it needed to run every time a station was given a price quote. This
Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems

- What problems are this?
  Problems consisting of (many) decisions and constraints

- What are ASP’s distinguishing features?
  - High level, versatile modeling language
  - High performance solvers
  - Qualitative and quantitative optimization
Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints
- What are ASP’s distinguishing features?
  - High level, versatile modeling language
  - High performance solvers
  - Qualitative and quantitative optimization
- Any industrial impact?
  - ASP Tech companies: DLV Systems and Potassco Solutions
  - Increasing interest in (large) companies
Outline

1. Motivation
2. Nutshell
3. Evolution
4. Foundation
5. Workflow
6. Engine
7. Usage
8. Summary
Some biased moments in time

- '70/'80 Capturing incomplete information
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
    - Axiomatic characterization
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    - Auto-epistemic and Default logics, Circumscription
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
    - Axiomatic characterization
  - Logic programming  Negation as failure
    - Herbrand interpretations
    - Fix-point characterizations
- Non-monotonic reasoning
  Auto-epistemic and Default logics, Circumscription
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
    - Axiomatic characterization
  - Logic programming  Negation as failure
    - Herbrand interpretations
    - Fix-point characterizations
- Non-monotonic reasoning
  Auto-epistemic and Default logics, Circumscription
    - Extensions of first-order logic
    - Modalities, fix-points, second-order logic
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    - Auto-epistemic and Default logics, Circumscription

- '90 Amalgamation and computation
Some biased moments in time

- ’70/’80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

- ’90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    “Stable models = Well-founded semantics + Branch”
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
    - Stable models semantics derived from non-monotonic logics
    - Alternating fix-point theory
  - ASP solving
    “Stable models = Well-founded semantics + Branch”
Some biased moments in time

'70/'80 Capturing incomplete information
- Databases  Closed world assumption
- Logic programming  Negation as failure
- Non-monotonic reasoning
  Auto-epistemic and Default logics, Circumscription

'90 Amalgamation and computation
- Logic programming semantics
  Well-founded and stable models semantics
  - Stable models semantics derived from non-monotonic logics
  - Alternating fix-point theory
- ASP solving
  “Stable models = Well-founded semantics + Branch”
- Modeling — Grounding — Solving
- Icebreakers: lparse and smodels
Some biased moments in time

- **'70/'80 Capturing incomplete information**
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

- **'90 Amalgamation and computation**
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    “Stable models = Well-founded semantics + Branch”

- **'00 Applications and semantic rediscoveries**
Some biased moments in time

- **'70/'80 Capturing incomplete information**
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning  Auto-epistemic and Default logics, Circumscription

- **'90 Amalgamation and computation**
  - Logic programming semantics  Well-founded and stable models semantics
  - ASP solving  “Stable models = Well-founded semantics + Branch”

- **'00 Applications and semantic rediscoveries**
  - Growing dissemination  Decision Support for Space Shuttle
  - Constructive logics  Equilibrium Logic
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning  Auto-epistemic and Default logics, Circumscription

- '90 Amalgamation and computation
  - Logic programming semantics
    - Well-founded and stable models semantics
  - ASP solving
    - “Stable models = Well-founded semantics + Branch”

- '00 Applications and semantic rediscoveries
  - Growing dissemination  Decision Support for Space Shuttle
    - Bio-informatics, Linux Package Configuration, Music composition, Robotics, System Design, etc
  - Constructive logics  Equilibrium Logic
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning Auto-epistemic and Default logics, Circumscription

- '90 Amalgamation and computation
  - Logic programming semantics
    - Well-founded and stable models semantics
  - ASP solving
    - “Stable models = Well-founded semantics + Branch”

- '00 Applications and semantic rediscoveries
  - Growing dissemination  Decision Support for Space Shuttle
  - Constructive logics  Equilibrium Logic
    - Roots: Logic of Here-and-There , G3
Some biased moments in time

- **'70/'80** Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    - Auto-epistemic and Default logics, Circumscription

- **'90** Amalgamation and computation
  - Logic programming semantics
    - Well-founded and stable models semantics
  - ASP solving
    - “Stable models = Well-founded semantics + Branch”

- **'00** Applications and semantic rediscoveries
  - Growing dissemination  Decision Support for Space Shuttle
  - Constructive logics  Equilibrium Logic

- **'10** Integration
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning  Auto-epistemic and Default logics, Circumscription

- '90 Amalgamation and computation
  - Logic programming semantics  Well-founded and stable models semantics
  - ASP solving  “Stable models = Well-founded semantics + Branch”

- '00 Applications and semantic rediscoveries
  - Growing dissemination  Decision Support for Space Shuttle
  - Constructive logics  Equilibrium Logic

- '10 Integration  —  let’s see . . .
Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases  Closed world assumption
  - Logic programming  Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    “Stable models = Well-founded semantics + Branch”

- '00 Applications and semantic rediscoveries
  - Growing dissemination  Decision Support for Space Shuttle
  - Constructive logics  Equilibrium Logic

- '10 Integration
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1 Provide a representation of the problem
2 A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1 Provide a representation of the problem
2 A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions
Paradigm shift

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI’92)

Represent planning problems as propositional theories so that models not proofs describe solutions
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI’92)

Represent planning problems as propositional theories so that models not proofs describe solutions
Paradigm shift

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI’92)

Represent planning problems as propositional theories so that models not proofs describe solutions
### Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

---

Torsten Schaub (KRR@UP)  
Answer Set Solving in Practice  
February 18, 2019  
16 / 653
## Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Torsten Schaub (KRR@UP)
## Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
<th>( \text{SAT} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
<td></td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
<td></td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
<td></td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
<td></td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
<td></td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
<td></td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
<td></td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
<td></td>
</tr>
<tr>
<td>first-order theories</td>
<td>expansions</td>
<td></td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>extensions</td>
<td></td>
</tr>
<tr>
<td>default theories</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
LP-style playing with blocks

Prolog program

\[
\begin{align*}
on(a,b). \\
on(b,c). \\
above(X,Y) & :\ - \ on(X,Y). \\
above(X,Y) & :\ - \ on(X,Z), \ above(Z,Y).
\end{align*}
\]

Prolog queries

?- above(a,c).
true.

?- above(c,a).
nomark.
LP-style playing with blocks

Prolog program

on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).

Prolog queries

?- above(a,c).
true.

?- above(c,a).
no.
LP-style playing with blocks

Prolog program

\[
\begin{align*}
on(a,b). \\
on(b,c).
\end{align*}
\]

\[
\begin{align*}
\text{above}(X,Y) & :- \text{on}(X,Y). \\
\text{above}(X,Y) & :- \text{on}(X,Z), \text{above}(Z,Y).
\end{align*}
\]

Prolog queries

?- above(a,c).
true.

?- above(c,a).
no.
LP-style playing with blocks

Prolog program

on(a,b).
on(b,c).

above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).

Prolog queries (testing entailment)

?- above(a,c).
true.

?- above(c,a).
no.
**LP-style playing with blocks**

**Shuffled Prolog program**

```prolog
on(a,b).
on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y).
above(X,Y) :- on(X,Y).
```

**Prolog queries**

```prolog
?- above(a,c).
```

Fatal Error: local stack overflow.
LP-style playing with blocks

Shuffled Prolog program

\begin{verbatim}
on(a,b).
on(b,c).

above(X,Y) :- above(X,Z), on(Z,Y).
above(X,Y) :- on(X,Y).
\end{verbatim}

Prolog queries

?– above(a,c).

Fatal Error: local stack overflow.
LP-style playing with blocks

Shuffled Prolog program

\begin{align*}
on(a,b). \\
on(b,c).
\end{align*}

\begin{align*}
\text{above}(X,Y) &:\equiv \text{above}(X,Z), \text{on}(Z,Y). \\
\text{above}(X,Y) &:\equiv \text{on}(X,Y).
\end{align*}

Prolog queries (answered via fixed execution)

?- \text{above}(a,c).

Fatal Error: local stack overflow.
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
SAT-style playing with blocks

**Formula**

\[
\begin{align*}
on(a, b) \\
\land on(b, c) \\
\land (on(X, Y) \rightarrow above(X, Y)) \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\end{align*}
\]

**Herbrand model**

\[
\begin{align*}
\{on(a, b), & \quad on(b, c), \quad on(a, c), \quad on(b, b), \\
above(a, b), & \quad above(b, c), \quad above(a, c), \quad above(b, b), \quad above(c, b)\}
\end{align*}
\]
SAT-style playing with blocks

Formula

\[\begin{align*}
on(a, b) \\
\land on(b, c) \\
\land (on(X, Y) \rightarrow above(X, Y)) \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\end{align*}\]

Herbrand model

\[\{\text{on}(a, b), \text{on}(b, c), \text{on}(a, c), \text{on}(b, b), \text{above}(a, b), \text{above}(b, c), \text{above}(a, c), \text{above}(b, b), \text{above}(c, b)\}\]
SAT-style playing with blocks

Formula

\[
on(a, b) \land on(b, c) \land (on(X, Y) \rightarrow above(X, Y)) \land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\]

Herbrand model

\[
\{ on(a, b), on(b, c), on(a, c), on(b, b), above(a, b), above(b, c), above(a, c), above(b, b), above(c, b) \}
\]
SAT-style playing with blocks

Formula

\[
on(a, b) \\
\land on(b, c) \\
\land (on(X, Y) \rightarrow above(X, Y)) \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\]

Herbrand model

\[
\{ on(a, b), on(b, c), on(a, c), on(b, b), above(a, b), above(b, c), above(a, c), above(b, b), above(c, b) \}
\]
SAT-style playing with blocks

Formula

\[\begin{align*}
on(a, b) \\
\land on(b, c) \\
\land (on(X, Y) \rightarrow above(X, Y)) \\
\land (on(X, Z) \land above(Z, Y) \rightarrow above(X, Y))
\end{align*}\]

Herbrand model (among 426!)

\[\{\begin{align*}
on(a, b), & \quad on(b, c), & \quad on(a, c), & \quad on(b, b), \\
above(a, b), & \quad above(b, c), & \quad above(a, c), & \quad above(b, b), & \quad above(c, b)
\end{align*}\}\]
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation
Paradigm shift

Theorem Proving based approach (eg. Prolog)
1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)
1. Provide a representation of the problem
2. A solution is given by a model of the representation

Answer Set Programming (ASP)
### Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
## Answer Set Programming *at large*

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>expansions</td>
</tr>
<tr>
<td>first-order theories</td>
<td>extensions</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td></td>
</tr>
<tr>
<td>default theories</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
**Answer Set Programming** commonly

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Answer Set Programming *in practice*

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td><strong>propositional programs</strong></td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
</tbody>
</table>

...
## Answer Set Programming in practice

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>expansions</td>
</tr>
<tr>
<td>first-order programs</td>
<td>extensions</td>
</tr>
<tr>
<td>first-order programs</td>
<td>stable Herbrand models</td>
</tr>
</tbody>
</table>
ASP-style playing with blocks

Logic program

\[
\begin{align*}
on(a,b). \\
on(b,c).
\end{align*}
\]

\[
\begin{align*}
\text{above}(X,Y) & : - \text{on}(X,Y). \\
\text{above}(X,Y) & : - \text{on}(X,Z), \text{above}(Z,Y).
\end{align*}
\]

Stable Herbrand model

\[
\{ \text{on}(a,b), \text{on}(b,c), \text{above}(b,c), \text{above}(a,b), \text{above}(a,c) \}
\]
ASP-style playing with blocks

Logic program

on(a, b).
on(b, c).

above(X, Y) :- on(X, Y).
above(X, Y) :- on(X, Z), above(Z, Y).

Stable Herbrand model

\{ on(a, b), on(b, c), above(b, c), above(a, b), above(a, c) \}
ASP-style playing with blocks

Logic program

\[
\begin{align*}
on(a,b) & . \\
on(b,c) & . \\
above(X,Y) & : - on(X,Y) . \\
above(X,Y) & : - on(X,Z), above(Z,Y). \\
\end{align*}
\]

Stable Herbrand model (and no others)

\{
\begin{align*}
on(a,b), & \ on(b,c), \ above(b,c), \ above(a,b), \ above(a,c) \}
\end{align*}
\}
ASP-style playing with blocks

Logic program

\[
\begin{align*}
on(a,b). \\
on(b,c).
\end{align*}
\]

\[
\begin{align*}
\text{above}(X,Y) & :\neg \text{above}(Z,Y), \text{on}(X,Z). \\
\text{above}(X,Y) & :\neg \text{on}(X,Y).
\end{align*}
\]

Stable Herbrand model (and no others)

\{
on(a, b), \text{on}(b, c), \text{above}(b, c), \text{above}(a, b), \text{above}(a, c)\}
### ASP versus LP

<table>
<thead>
<tr>
<th></th>
<th>ASP</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model generation</td>
<td>Model generation</td>
<td>Query orientation</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>Bottom-up</td>
<td>Top-down</td>
</tr>
<tr>
<td>Modeling language</td>
<td>Modeling language</td>
<td>Programming language</td>
</tr>
</tbody>
</table>

#### Rule-based format

<table>
<thead>
<tr>
<th></th>
<th>Instantiation</th>
<th>Unification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat terms</td>
<td>Flat terms</td>
<td>Nested terms</td>
</tr>
<tr>
<td>(Turing +) (NP)</td>
<td>(Turing +) (NP)</td>
<td>Turing</td>
</tr>
</tbody>
</table>
## ASP versus SAT

<table>
<thead>
<tr>
<th>ASP</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model generation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Bottom-up</strong></td>
<td></td>
</tr>
<tr>
<td>Constructive Logic</td>
<td>Classical Logic</td>
</tr>
<tr>
<td>Closed (and open) world reasoning</td>
<td>Open world reasoning</td>
</tr>
<tr>
<td>Modeling language</td>
<td></td>
</tr>
<tr>
<td>Complex reasoning modes</td>
<td>Satisfiability testing</td>
</tr>
<tr>
<td>Satisfiability</td>
<td>Satisfiability</td>
</tr>
<tr>
<td>Enumeration/Projection</td>
<td></td>
</tr>
<tr>
<td>Intersection/Union</td>
<td></td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
</tr>
<tr>
<td>*( Turing + ) $NP(^{NP})$ *</td>
<td>$NP$</td>
</tr>
</tbody>
</table>
Propositional Normal Logic Programs

- A logic program $P$ is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- $a$ and all $b_i, c_j$ are atoms (propositional variables)
- $\leftarrow, \land, \neg$ denote if, and, and negation
- Intuitive reading: head must be true if body holds

- Semantics given by stable models, informally, models of $P$ justifying each true atom by some rule in $P$
A logic program $P$ is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- $a$ and all $b_i, c_j$ are atoms (propositional variables)
- $\leftarrow, \, \cdot, \, \neg$ denote if, and, and negation
- Intuitive reading: head must be true if body holds

Semantics given by stable models, informally, models of $P$ justifying each true atom by some rule in $P$
A logic program $P$ is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- $a$ and all $b_i, c_j$ are atoms (propositional variables)
- $\leftarrow, \land, \neg$ denote if, and, and negation
- Intuitive reading: head must be true if body holds

Semantics given by stable models, informally, models of $P$ justifying each true atom by some rule in $P$
A logic program $P$ is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n$$

- $a$ and all $b_i, c_j$ are atoms (propositional variables)
- $\leftarrow, \land, \neg$ denote if, and, and negation
- Intuitive reading: head must be true if body holds

Semantics given by stable models, informally, models of $P$ justifying each true atom by some rule in $P$

Disclaimer The following formalities apply to normal logic programs
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$(\neg F \rightarrow F) \land (F \rightarrow F)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$(\neg F \rightarrow F) \land (F \rightarrow T)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(\neg T \rightarrow F) \land (T \rightarrow F)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(\neg T \rightarrow F) \land (T \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$(\neg F \rightarrow T) \land (F \rightarrow F)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$(\neg F \rightarrow T) \land (F \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(\neg T \rightarrow T) \land (T \rightarrow F)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(\neg T \rightarrow T) \land (T \rightarrow T)$</td>
</tr>
</tbody>
</table>
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$(T \rightarrow F) \land (F \rightarrow F)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$(T \rightarrow F) \land (F \rightarrow T)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow F) \land (T \rightarrow F)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow F) \land (T \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$(T \rightarrow T) \land (F \rightarrow F)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$(T \rightarrow T) \land (F \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow T) \land (T \rightarrow F)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow T) \land (T \rightarrow T)$</td>
</tr>
</tbody>
</table>
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>(T → F) \land (F → F)</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>(T → F) \land (F → T)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>(F → F) \land (T → F)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>(F → F) \land (T → T)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>(T → T) \land (F → F)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>(T → T) \land (F → T)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>(F → T) \land (T → F)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>(F → T) \land (T → T)</td>
</tr>
</tbody>
</table>
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F $\land (F \rightarrow F)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F $\land (F \rightarrow T)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow F) \land F$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow F) \land (T \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$(T \rightarrow T) \land (F \rightarrow F)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$(T \rightarrow T) \land (F \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow T) \land F$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow T) \land (T \rightarrow T)$</td>
</tr>
</tbody>
</table>
### Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F $\land (F \rightarrow F)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F $\land (F \rightarrow T)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow F) \land F$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow F) \land (T \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$(T \rightarrow T) \land (F \rightarrow F)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$(T \rightarrow T) \land (F \rightarrow T)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow T) \land F$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow T) \land (T \rightarrow T)$</td>
</tr>
</tbody>
</table>
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F \land T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F \land T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T \land F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T \land T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T \land T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T \land T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T \land F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T \land T</td>
</tr>
</tbody>
</table>
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Torsten Schaub (KRR@UP)
Answer Set Solving in Practice
February 18, 2019 29 / 653
Some truth tabling, back to SAT

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(-b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

- We get four models: $\{b, c\}$, $\{a\}$, $\{a, c\}$, and $\{a, b, c\}$
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$(\neg F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$(\neg F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(\neg T \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(\neg T \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$(\neg F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$(\neg F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(\neg T \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(\neg T \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>((T \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>((T \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>((F \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>((F \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>((T \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>((T \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>((F \rightarrow a) \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>((F \rightarrow a) \land (b \rightarrow c))</td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(F \rightarrow a) \land (b \rightarrow c)$</td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$T \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$T \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$T \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$T \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$T \land (b \rightarrow c)$</td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>((b \rightarrow c))</td>
</tr>
</tbody>
</table>

**Reduct**
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>(b \rightarrow c)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>(b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>(b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>(b \rightarrow c)</td>
</tr>
</tbody>
</table>

Reduct
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>((b \rightarrow c)) (\models)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>((b \rightarrow c)) (\models) (a)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>((b \rightarrow c)) (\models) (a)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>((b \rightarrow c)) (\models)</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>((b \rightarrow c)) (\models)</td>
</tr>
</tbody>
</table>
| F | F | F | \((b \rightarrow c)\) \(\models\) \((\neg b \rightarrow a) \land (b \rightarrow c)\)

Reduct
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>((\neg b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>(a \land (b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>((b \rightarrow c))</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>((b \rightarrow c))</td>
</tr>
</tbody>
</table>

Reduct
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>$a \land (b \rightarrow c) \models a$</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>$a \land (b \rightarrow c) \models a$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>$(b \rightarrow c) \models$</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>$(b \rightarrow c) \models$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>$a \land (b \rightarrow c) \models a$</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>$a \land (b \rightarrow c) \models a$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>$(b \rightarrow c) \models$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>$(b \rightarrow c) \models$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Reduct</strong></td>
</tr>
</tbody>
</table>
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>((-b \rightarrow a) \land (b \rightarrow c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>(b \rightarrow c)</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>(b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>a \land (b \rightarrow c) \models a \quad \text{Stable model}</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>a \land (b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>(b \rightarrow c)</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>(b \rightarrow c)</td>
</tr>
</tbody>
</table>

- We get one stable model: \(\{a\}\)
Some truth tabling, and now ASP

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$(\neg b \rightarrow a) \land (b \rightarrow c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$T$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$F$</td>
<td>$(b \rightarrow c)$</td>
</tr>
<tr>
<td>$F$</td>
<td>$T$</td>
<td>$T$</td>
<td>$(b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$F$</td>
<td>$a \land (b \rightarrow c) \models a$ Stable model</td>
</tr>
<tr>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
<td>$a \land (b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$(b \rightarrow c)$</td>
</tr>
<tr>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$(b \rightarrow c)$</td>
</tr>
</tbody>
</table>

- We get one stable model: \{a\}
- Stable models = Smallest models of (respective) reducts
ASP modeling, grounding, and solving

**Workflow**

- **Problem**
- **Logic Program**
- **Grounder**
- **Solver**
- **Stable Models**

**Modeling**

**Solving**

**Interpreting**
SAT solving

Workflow

Problem

Formula (CNF)

Solver

Classical Models

Solution

Programming

Interpreting

Solving
Rooting ASP solving

Workflow:

- Problem
  - Modeling
  - Logic Program
  - Grounded Logic Program
  - Solver
    - Solution
      - Interpreting
  - Stable Models

Solving:
Rooting ASP solving

Workflow

Problem → Grounder → Solver → Solution

- Logic Program (LP)
- Grounder (DB)
- Solver (SAT)
- Stable Models (DB+KR+LP)

Modeling → KR

Interpreting
Outline

1 Motivation
2 Nutshell
3 Evolution
4 Foundation
5 Workflow
6 Engine
7 Usage
8 Summary
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
  - Program Builder

- **Coordination**
  - SharedContext
    - Propositional Variables
    - Atoms
    - Bodies
    - Static Nogoods
    - Short Nogoods

- **Solver 1...n**
  - Decision Heuristic
    - Assignment Atoms/Bodies
  - Conflict Resolution
    - Recorded Nogoods
  - Propagation
    - Unit Propagation
    - Post Propagation

- **ParallelContext**
  - Threads $S_1 S_2 ... S_n$
  - Counter $T W ... S$
  - Queue $P_1 P_2 ... P_n$
  - Shared Nogoods

- **Logic Program**
  - Preprocessing
    - Program Builder
  - Logic Program
Multi-threaded architecture of *clasp*

**Coordination**
- SharedContext
  - Propositional Variables
  - Atoms
  - Bodies
  - Static Nogoods
  - Short Nogoods
- Enumerator
  - ParallelContext
    - Threads: \(S_1, S_2, \ldots, S_n\)
    - Counter: \(T, W, \ldots, S\)
    - Queue: \(P_1, P_2, \ldots, P_n\)
- Nogood Distributor
  - Shared Nogoods

**Solver 1\ldots n**
- Conflict Resolution
- Recorded Nogoods
- Decision Heuristic
- Assignment Atoms/Bodies
- Propagation
  - Unit Propagation
  - Post Propagation

**Preprocessing**
- Preprocessor
  - Program Builder
  - Logic Program

**Answer Set Solving in Practice**

February 18, 2019
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
    - Program Builder
      - Logic Program

- **Coordination**
  - SharedContext
    - Propositional Variables
      - Atoms
      - Bodies
      - Static Nogoods
      - Short Nogoods
  - Enumerators
    - Threads $S_1 S_2 \ldots S_n$
    - Counter $T W \ldots S$
    - Queue $P_1 P_2 \ldots P_n$
  - Nogood Distributor
    - Shared Nogoods

- **Solver 1\ldots n**
  - Conflict Resolution
    - Decision Heuristic
      - Assignment Atoms/Bodies
  - Recorded Nogoods
    - Propagation
      - Unit Propagation
      - Post Propagation
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
  - Program Builder

- **Logic Program**

- **Solver 1...n**
  - Conflict Resolution
  - Decision Heuristic
  - Assignment Atoms/Bodies

- **Coordination**
  - Shared Context
    - Propositional Variables
    - Atoms → Bodies
    - Static Nogoods
    - Short Nogoods
  - Nogood Distributor

- **Parallel Context**
  - Threads $S_1, S_2, \ldots, S_n$
  - Counter $T, W, \ldots, S$
  - Queue $P_1, P_2, \ldots, P_n$

- **Shared Nogoods**

- **Propagations**
  - Recorded Nogoods
  - Propagation:
    - Unit Propagation
    - Post Propagation
Multi-threaded architecture of *clasp*

Preprocessing

*Preprocessor*

Program Builder

Logic Program

**Engine**

**Coordination**

*SharedContext*

Propositional Variables

Atoms → Bodies

Static Nogoods

Short Nogoods

**Conflict Resolution**

Decision Heuristic

Assignment Atoms/Bodies

**Solver 1...n**

Conflict Resolution

Recorded Nogoods

Propagation

Unit Propagation

Post Propagation

**ParallelContext**

Threads $S_1 S_2 ... S_n$

Counter $T W ... S$

Queue $P_1 P_2 ... P_n$

**Nmogood Distributor**

Shared Nogoods

**Logic Program**

*Preprocessor*

Program Builder

Logic Program
Multi-threaded architecture of clasp

Preprocessing
- Preprocessor
  - Program Builder

Logic Program

Coordination
- SharedContext
  - Propositional Variables
  - Atoms
  - Bodies
  - Static Nogoods
  - Short Nogoods

Decision Heuristic
- Assignment
  - Atoms/Bodies

Solver 1...n
- Conflict Resolution
- Recorded Nogoods

ParallelContext
- Threads: $S_1, S_2, \ldots, S_n$
- Counter: $T, W, \ldots, S$
- Queue: $P_1, P_2, \ldots, P_n$

Nogood Distributor

Shared Nogoods

Counter

Unit Propagation

Post Propagation

Preprocessing
- Preprocessor
  - Program Builder

Logic Program
Two sides of a coin

- **ASP as High-level Language**
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules

- **ASP as Low-level Language**
  - Compile a problem into a set of facts and rules
  - Solve the original problem by solving its compilation

- **ASP and Imperative language**
  Control continuously changing logic programs
Two sides of a coin

- **ASP as High-level Language**
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules

- **ASP as Low-level Language**
  - Compile a problem into a set of facts and rules
  - Solve the original problem by solving its compilation

- **ASP and Imperative language**
  Control continuously changing logic programs
Two sides of a coin

- **ASP as High-level Language**
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules

- **ASP as Low-level Language**
  - **Compile** a problem into a set of facts and rules
  - Solve the original problem by solving its compilation

- **ASP and Imperative language**
  - Control continuously changing logic programs
Two sides of a coin

- **ASP as High-level Language**
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules

- **ASP as “Low-level” Language**
  - Compile a problem instance into a set of facts
  - Encode problem class as a set of rules
  - Solve the original problem by solving its compilation

- **ASP and Imperative language**
  - Control continuously changing logic programs
Two and a half sides of a coin

- **ASP as High-level Language**
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules

- **ASP as “Low-level” Language**
  - Compile a problem instance into a set of facts
  - Encode problem class as a set of rules
  - Solve the original problem by solving its compilation

- **ASP and Imperative language**
  - Control continuously changing logic programs
Summary

Upcoming experience

- **ASP** is a viable tool for Knowledge Representation and Reasoning
  - Integration of DB, LP, KR, and SAT techniques
  - Combinatorial search problems in the realm of $NP$ and $NP^{NP}$
  - Succinct, elaboration-tolerant problem representations
    - rapid application development tool
  - Easy handling of knowledge-intensive applications
    - data, defaults, exceptions, frame axioms, reachability etc
- **ASP** offers efficient and versatile off-the-shelf solving technology
  - [http://potassco.org](http://potassco.org)
  - winning ASP, CASC, MISC, PB, and SAT competitions
- **ASP** has a growing range of applications, and it's good fun!
Upcoming experience

- ASP is a viable tool for Knowledge Representation and Reasoning
  - Integration of DB, LP, KR, and SAT techniques
  - Combinatorial search problems in the realm of $NP$ and $NP^{NP}$
  - Succinct, elaboration-tolerant problem representations
    - rapid application development tool
  - Easy handling of knowledge-intensive applications
    - data, defaults, exceptions, frame axioms, reachability etc
- ASP offers efficient and versatile off-the-shelf solving technology
  - http://potassco.org
  - winning ASP, CASC, MISC, PB, and SAT competitions
- ASP has a growing range of applications, and it's good fun!

\[
\text{ASP} = \text{DB} + \text{LP} + \text{KR} + \text{SAT}
\]
ASP is a viable tool for Knowledge Representation and Reasoning
- Integration of DB, LP, KR, and SAT techniques
- Combinatorial search problems in the realm of $NP$ and $NP^{NP}$
- Succinct, elaboration-tolerant problem representations
  - rapid application development tool
- Easy handling of knowledge-intensive applications
  - data, defaults, exceptions, frame axioms, reachability etc
ASP offers efficient and versatile off-the-shelf solving technology
- [http://potassco.org](http://potassco.org)
- winning ASP, CASC, MISC, PB, and SAT competitions
ASP has a growing range of applications, and it's good fun!

$$ASP = DB + LP + KR + SMT^n$$
Outline

9 Syntax
10 Semantics
11 Examples
12 Reasoning
13 Language
14 Variables
Normal logic programs

- A logic program, \( P \), over a set \( \mathcal{A} \) of atoms is a finite set of rules.
- A (normal) rule, \( r \), is of the form
  \[
  a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n
  \]
  where \( 0 \leq m \leq n \) and each \( a_i \in \mathcal{A} \) is an atom for \( 0 \leq i \leq n \).

Notation:

- \( h(r) = a_0 \)
- \( B(r) = \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \)
- \( B(r)^+ = \{ a_1, \ldots, a_m \} \)
- \( B(r)^- = \{ a_{m+1}, \ldots, a_n \} \)

A literal is an atom or a negated atom.
A program \( P \) is positive if \( B(r)^- = \emptyset \) for all \( r \in P \).
Normal logic programs

A logic program, \( P \), over a set \( \mathcal{A} \) of atoms is a finite set of rules.

A (normal) rule, \( r \), is of the form

\[
a_0 :\neg a_1, \ldots, \neg a_m, a_{m+1}, \ldots, \neg a_n.
\]

where \( 0 \leq m \leq n \) and each \( a_i \in \mathcal{A} \) is an atom for \( 0 \leq i \leq n \).

Notation

\[
\begin{align*}
h(r) & = a_0 \\
B(r) & = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\} \\
B(r)^+ & = \{a_1, \ldots, a_m\} \\
B(r)^- & = \{a_{m+1}, \ldots, a_n\}
\end{align*}
\]

A literal is an atom or a negated atom.

A program \( P \) is positive if \( B(r)^- = \emptyset \) for all \( r \in P \).
Normal logic programs

- A logic program, $P$, over a set $A$ of atoms is a finite set of rules.
- A (normal) rule, $r$, is of the form
  $$a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n$$
  where $0 \leq m \leq n$ and each $a_i \in A$ is an atom for $0 \leq i \leq n$.
- Notation
  $$h(r) = a_0$$
  $$B(r) = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$$
  $$B(r)^+ = \{a_1, \ldots, a_m\}$$
  $$B(r)^- = \{a_{m+1}, \ldots, a_n\}$$
- A literal is an atom or a negated atom.
- A program $P$ is positive if $B(r)^- = \emptyset$ for all $r \in P$. 

Torsten Schaub (KRR@UP)
Normal logic programs

- A logic program, $P$, over a set $A$ of atoms is a finite set of rules.
- A (normal) rule, $r$, is of the form
  \[ a_0 \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \]
  where $0 \leq m \leq n$ and each $a_i \in A$ is an atom for $0 \leq i \leq n$.

- Notation
  \[
  h(r) = a_0 \\
  B(r) = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\} \\
  B(r)^+ = \{a_1, \ldots, a_m\} \\
  B(r)^- = \{a_{m+1}, \ldots, a_n\} \\
  A(P) = \bigcup_{r \in P} (\{h(r)\} \cup B(r)^+ \cup B(r)^-) \\
  B(P) = \{B(r) \mid r \in P\} \\
  h(P) = \{h(r) \mid r \in P\}
  \]
Normal logic programs

- A logic program, \( P \), over a set \( A \) of atoms is a finite set of rules.
- A (normal) rule, \( r \), is of the form
  \[
  a_0 \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n
  \]
  where \( 0 \leq m \leq n \) and each \( a_i \in A \) is an atom for \( 0 \leq i \leq n \).
- Notation
  \[
  h(r) = a_0 \\
  B(r) = \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \\
  B(r)^+ = \{ a_1, \ldots, a_m \} \\
  B(r)^- = \{ a_{m+1}, \ldots, a_n \}
  \]
- A literal is an atom or a negated atom.
- A program \( P \) is positive if \( B(r)^- = \emptyset \) for all \( r \in P \).
Normal logic programs

- A logic program, $P$, over a set $\mathcal{A}$ of atoms is a finite set of rules.
- A (normal) rule, $r$, is of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$$

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$.

- Notation

$$h(r) = a_0$$
$$B(r) = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\}$$
$$B(r)^+ = \{a_1, \ldots, a_m\}$$
$$B(r)^- = \{a_{m+1}, \ldots, a_n\}$$

- A literal is an atom or a negated atom.
- A program $P$ is positive if $B(r)^- = \emptyset$ for all $r \in P$. 
- **Example rules**
  - \( a \leftarrow b, \neg c \)
  - \( a \leftarrow \neg c, b \)
  - \( a \leftarrow \)
  - \( a \leftarrow b \)
  - \( a \leftarrow \neg c \)
  - \( \text{bachelor}(joe) \leftarrow \text{male}(joe), \neg \text{married}(joe) \)

- **Example literals**
  - \( a, b, c, \text{bachelor}(joe), \text{male}(joe), \text{married}(joe) \)
  - \( \neg c, \neg \text{married}(joe) \)
Examples

- **Example rules**
  - \( a ← b, \sim c \)
  - \( a ← \sim c, b \)
  - \( a ← \)
  - \( a ← b \)
  - \( a ← \sim c \)
  - \( \text{bachelor}(joe) ← \text{male}(joe), \sim \text{married}(joe) \)

- **Example literals**
  - \( a, b, c, \text{bachelor}(joe), \text{male}(joe), \text{married}(joe) \)
  - \( \sim c, \sim \text{married}(joe) \)
Example rules

- $a \leftarrow b, \sim c$
- $a \leftarrow \sim c, b$
- $a \leftarrow$
- $a \leftarrow b$
- $a \leftarrow \sim c$
- $\text{bachelor}(joe) \leftarrow \text{male}(joe), \sim \text{married}(joe)$

Example literals

- $a, b, c, \text{bachelor}(joe), \text{male}(joe), \text{married}(joe)$
- $\sim c, \sim \text{married}(joe)$
Example rules

- $a \leftarrow b, \sim c$
- $a \leftarrow \sim c, b$
- $a \leftarrow$
- $a \leftarrow b$
- $a \leftarrow \sim c$
- $bachelor(joe) \leftarrow male(joe), \sim married(joe)$

Example literals

- $a, b, c, bachelor(joe), male(joe), married(joe)$
- $\sim c, \sim married(joe)$
We sometimes use the following notation interchangeably in order to stress the respective view:

<table>
<thead>
<tr>
<th>Source code logic program formula</th>
<th>true, false</th>
<th>if</th>
<th>and</th>
<th>or</th>
<th>iff</th>
<th>default negation</th>
<th>classical negation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:- , ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>not</td>
<td>−</td>
</tr>
<tr>
<td></td>
<td>← , ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~</td>
<td>¬</td>
</tr>
<tr>
<td>⊥, T</td>
<td>→ ∧ ∨ ↔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~</td>
<td>¬</td>
</tr>
</tbody>
</table>
Outline

- Syntax
- Semantics
- Examples
- Reasoning
- Language
- Variables
Semantics

Problem

Modeling

Logic Program

Solving

Solution

Interpreting

Stable Models
A set of atoms $X$ is closed under a positive program $P$ iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$.

- $X$ corresponds to a model of $P$ (seen as a formula).

The smallest set of atoms which is closed under a positive program $P$ is denoted by $Cn(P)$.

- $Cn(P)$ corresponds to the $\subseteq$-smallest model of $P$ (ditto).

The set $Cn(P)$ of atoms is the stable model of a positive program $P$. 
Semantics

Formal Definition

Stable models of positive programs

- A set of atoms $X$ is closed under a positive program $P$ iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)

- The smallest set of atoms which is closed under a positive program $P$ is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of $P$ (ditto)

- The set $Cn(P)$ of atoms is the stable model of a positive program $P$
Formal Definition
Stable models of positive programs

- A set of atoms $X$ is closed under a positive program $P$ iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)

- The smallest set of atoms which is closed under a positive program $P$ is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of $P$ (ditto)

- The set $Cn(P)$ of atoms is the stable model of a positive program $P$
A set of atoms $X$ is closed under a positive program $P$ iff for any $r \in P$, $h(r) \in X$ whenever $B(r)^+ \subseteq X$. 
- $X$ corresponds to a model of $P$ (seen as a formula).

The smallest set of atoms which is closed under a positive program $P$ is denoted by $Cn(P)$.
- $Cn(P)$ corresponds to the $\subseteq$-smallest model of $P$ (ditto).

The set $Cn(P)$ of atoms is the stable model of a positive program $P$. 
Some “logical” remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:
    \[ a_0 \lor \neg a_1 \lor \cdots \lor \neg a_m \]

- A set of definite clauses has a (unique) smallest model

- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none

- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program \( P \), \( Cn(P) \) corresponds to the smallest model of the set of definite clauses corresponding to \( P \)
Some “logical” remarks

- Positive rules are also referred to as **definite clauses**
  - Definite clauses are disjunctions with exactly one positive atom:
    \[ a_0 \lor \neg a_1 \lor \cdots \lor \neg a_m \]

- A set of definite clauses has a (unique) smallest model

- **Horn clauses** are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none

- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program \( P \), \( Cn(P) \) corresponds to the smallest model of the set of definite clauses corresponding to \( P \)
Some “logical” remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:
    \[ a_0 \lor \neg a_1 \lor \cdots \lor \neg a_m \]
  - A set of definite clauses has a (unique) smallest model

- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none

- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program \( P \), \( Cn(P) \) corresponds to the smallest model of the set of definite clauses corresponding to \( P \)
Consider the logical formula $\Phi$ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if $X$ is a (classical) model of $P$ and if all atoms in $X$ are justified by some rule in $P$.

$$\Phi = q \land (q \land \neg r \rightarrow p)$$

$$P_\Phi = \begin{align*}
q & \leftarrow \\
p & \leftarrow q, \sim r
\end{align*}$$
Consider the logical formula $\Phi$ and its three (classical) models:

\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}

Formula $\Phi$ has one stable model, often called answer set:

\{p, q\}

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if

- $X$ is a (classical) model of $P$ and
- if all atoms in $X$ are justified by some rule in $P$
Consider the logical formula $\Phi$ and its three (classical) models:

$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$

Formula $\Phi$ has one stable model, often called answer set:

$\{p, q\}$

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if

- $X$ is a (classical) model of $P$ and
- if all atoms in $X$ are justified by some rule in $P$
Consider the logical formula $\Phi$ and its three (classical) models:

\{p, q\}, \{q, r\}, and \{p, q, r\}

Formula $\Phi$ has one stable model, often called answer set:

\{p, q\}

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if $X$ is a (classical) model of $P$ and if all atoms in $X$ are justified by some rule in $P$.
Consider the logical formula $\Phi$ and its three (classical) models:

\[
\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}
\]

Formula $\Phi$ has one stable model, often called answer set:

\[
\{p, q\}
\]

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if

- $X$ is a (classical) model of $P$ and
- if all atoms in $X$ are justified by some rule in $P$
Consider the logical formula $\Phi$ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if $X$ is a (classical) model of $P$ and if all atoms in $X$ are justified by some rule in $P$. 

$$\Phi \quad q \land (q \land \neg r \rightarrow p)$$

$$P_\Phi \quad q \leftarrow$$

$$p \leftarrow q, \neg r$$
Consider the logical formula $\Phi$ and its three (classical) models:

\[
\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}
\]

Formula $\Phi$ has one stable model, often called answer set:

\[
\{p, q\}
\]

Informally, a set $X$ of atoms is a **stable model** of a logic program $P$:

- if $X$ is a (classical) model of $P$ and
- if all atoms in $X$ are **justified** by some rule in $P$
Consider the logical formula $\Phi$ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}$$

Formula $\Phi$ has one stable model, often called answer set:

$$\{p, q\}$$

Informally, a set $X$ of atoms is a stable model of a logic program $P$ if

- $X$ is a (classical) model of $P$ and
- if all atoms in $X$ are justified by some rule in $P$
Semantics

Formal definition

Stable models of normal programs

- The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

$$P^X = \{ h(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset \}$$

- A set $X$ of atoms is a stable model of a program $P$, if $Cn(P^X) = X$

Remarks

- $Cn(P^X)$ is the $\subseteq$-smallest (classical) model of $P^X$

Each atom in $X$ is justified by an "applying rule from $P$"
Set $X$ is stable under "applying rules from $P"
Formal definition

Stable models of normal programs

- The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

$$P^X = \{ h(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset \}$$

- A set $X$ of atoms is a stable model of a program $P$, if $Cn(P^X) = X$

Remarks

- $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of $P^X$
- Each atom in $X$ is justified by an “applying rule from $P$”
- Set $X$ is stable under “applying rules from $P$”
Formal definition
Stable models of normal programs

- The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

$$P^X = \{ h(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset \}$$

- A set $X$ of atoms is a stable model of a program $P$, if $Cn(P^X) = X$

Remarks
- $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of $P^X$
- Each atom in $X$ is justified by an “applying rule from $P$”
- Set $X$ is stable under “applying rules from $P$”
Formal definition
Stable models of normal programs

- The reduct, $P^X$, of a program $P$ relative to a set $X$ of atoms is defined by

$$P^X = \{ h(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset \}$$

- A set $X$ of atoms is a stable model of a program $P$, if $Cn(P^X) = X$

Remarks
- $Cn(P^X)$ is the $\subseteq$-smallest (classical) model of $P^X$
- Each atom in $X$ is justified by an “applying rule from $P$”
- Set $X$ is stable under “applying rules from $P$”
A closer look at $P^X$

- Alternatively, given a set $X$ of atoms from $P$,

$P^X$ is obtained from $P$ by deleting

1. each rule having $\sim a$ in its body with $a \in X$
   and then
2. all negative atoms of the form $\sim a$
   in the bodies of the remaining rules

- Note Only negative body literals are evaluated
A closer look at $P^X$

- Alternatively, given a set $X$ of atoms from $P$,

  $P^X$ is obtained from $P$ by deleting

  1. each rule having $\sim a$ in its body with $a \in X$
     and then

  2. all negative atoms of the form $\sim a$
     in the bodies of the remaining rules

- Note  Only negative body literals are evaluated
Example one

\[ P = \{ p \leftarrow p, \; q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p )</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(\text{Cn}(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>(p \leftarrow p)</td>
<td>({q})</td>
</tr>
<tr>
<td>{p}</td>
<td>(p \leftarrow p)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>{q}</td>
<td>(p \leftarrow p)</td>
<td>({q})</td>
</tr>
<tr>
<td>{p, q}</td>
<td>(p \leftarrow p)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( \mathcal{X} )</th>
<th>( P^\mathcal{X} )</th>
<th>( \text{Cn}(P^\mathcal{X}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p ) \qquad q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
**Example one**

\[ P = \{ p \leftarrow p, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{q} \text{ \texttimes}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow p )</td>
<td>{}</td>
</tr>
<tr>
<td>{q}</td>
<td>( p \leftarrow p )</td>
<td>{q} \text{ \textcheckmark}</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( p \leftarrow p )</td>
<td>{}</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )  ( q \leftarrow )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow p )  ( q \leftarrow )</td>
<td>{q}</td>
</tr>
<tr>
<td>{q}</td>
<td>( p \leftarrow p )  ( q \leftarrow )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p,q}</td>
<td>( p \leftarrow p )</td>
<td>{q}</td>
</tr>
</tbody>
</table>

\[ X \]
\[ P^X \]
\[ Cn(P^X) \]
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{ q }</td>
</tr>
<tr>
<td>\begin{align*} { &amp;p } \ \end{align*}</td>
<td>( p \leftarrow p )</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\begin{align*} { &amp;q } \ \end{align*}</td>
<td>( p \leftarrow p )</td>
<td>{ q }</td>
</tr>
<tr>
<td>\begin{align*} { &amp;p, q } \ \end{align*}</td>
<td>( p \leftarrow p )</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { } )</td>
<td>( p \leftarrow p )</td>
<td>( { q } )</td>
</tr>
<tr>
<td>( { p } )</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( { q } )</td>
<td>( p \leftarrow p )</td>
<td>( { q } )</td>
</tr>
<tr>
<td>( { p, q } )</td>
<td>( p \leftarrow p )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \; q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{ }</td>
<td>{ q } \times</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td>\emptyset \times</td>
</tr>
<tr>
<td>{ p }</td>
<td>{ p }</td>
<td>\emptyset \times</td>
</tr>
<tr>
<td>{ q }</td>
<td>{ p } { q }</td>
<td>{ q } \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>{ p }</td>
<td>\emptyset \times</td>
</tr>
</tbody>
</table>
Example one

\[ P = \{ p \leftarrow p, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow p )</td>
<td>{ q } \xmark</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow p )</td>
<td>\emptyset \checkmark</td>
</tr>
<tr>
<td>{ q }</td>
<td>( p \leftarrow p )</td>
<td>{ q } \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( p \leftarrow p )</td>
<td>\emptyset \checkmark</td>
</tr>
</tbody>
</table>
**Example two**

\[ P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{ p, q }</td>
</tr>
<tr>
<td></td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>{ p }</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow )</td>
<td>{ q }</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{p, q}</td>
<td></td>
<td>{ }</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \lnot q, \ q \leftarrow \lnot p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{  } {  }</td>
<td>( p \leftarrow ) {  } {  }</td>
<td>{ p, q } \times</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow ) { p } { p }</td>
<td>{ p } \checkmark</td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow ) { q } { q }</td>
<td>{ q } \checkmark</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( q \leftarrow ) {  } {  }</td>
<td>{  } \checkmark</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>( p \leftarrow )</td>
<td>( { p, q } )</td>
</tr>
<tr>
<td>{}</td>
<td>( q \leftarrow )</td>
<td></td>
</tr>
<tr>
<td>( { p } )</td>
<td>( p \leftarrow )</td>
<td>( { p } )</td>
</tr>
<tr>
<td>( { q } )</td>
<td>( q \leftarrow )</td>
<td>( { q } )</td>
</tr>
<tr>
<td>( { p, q } )</td>
<td>( )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p, q}</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{q}</td>
<td>( q \leftarrow )</td>
<td>{q}</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( q \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p, q} \xmark</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td>{p} \checkmark</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow )</td>
<td>{p} \checkmark</td>
</tr>
<tr>
<td>{q}</td>
<td>( q \leftarrow )</td>
<td>{q} \checkmark</td>
</tr>
<tr>
<td>{p, q}</td>
<td>( q \leftarrow )</td>
<td>{}</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>( { p, q } ) ( \times )</td>
</tr>
<tr>
<td>{ }</td>
<td>( q \leftarrow )</td>
<td>( { p, q } ) ( \times )</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>( { p } ) ( \checkmark )</td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow )</td>
<td>( { q } ) ( \checkmark )</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>(P^X)</th>
<th>(Cn(P^X))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>(p \leftarrow)</td>
<td>{p, q} X</td>
</tr>
<tr>
<td></td>
<td>(q \leftarrow)</td>
<td></td>
</tr>
<tr>
<td>{p}</td>
<td>(p \leftarrow)</td>
<td>{p} ✓</td>
</tr>
<tr>
<td></td>
<td>(q \leftarrow)</td>
<td>{q} ✓</td>
</tr>
<tr>
<td>{q}</td>
<td>(q \leftarrow)</td>
<td></td>
</tr>
<tr>
<td>{p, q}</td>
<td>(q \leftarrow)</td>
<td>(\emptyset) X</td>
</tr>
</tbody>
</table>
Example two

\[ P = \{ p \leftarrow \sim q, \ q \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ p, q }</td>
<td>{ p }</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ q }</td>
<td>{ p }</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ p }</td>
<td>{ q }</td>
<td>{ q }</td>
</tr>
<tr>
<td>{ }</td>
<td>{ p, q }</td>
<td>{ p, q }</td>
</tr>
</tbody>
</table>

\( p \leftarrow \sim q, \ q \leftarrow \sim p \)
Example two

\[ P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ } { }</td>
<td>( p \leftarrow ) ( q \leftarrow )</td>
<td>{ } { p, q } \text{ x}</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>{ p } \text{ ✓}</td>
</tr>
<tr>
<td>{ q }</td>
<td>( q \leftarrow )</td>
<td>{ q } \text{ ✓}</td>
</tr>
<tr>
<td>{ p, q }</td>
<td>( q \leftarrow )</td>
<td>{ } \text{ ✓}</td>
</tr>
</tbody>
</table>
Example three

\( P = \{ p \leftarrow \neg p \} \)

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ p }</td>
<td>( p \leftarrow )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>( {p} )</td>
</tr>
<tr>
<td>{p}</td>
<td>( )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>(X)</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{ p } (\times)</td>
</tr>
<tr>
<td>{ p }</td>
<td></td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \sim p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>{ p }</td>
<td>{ p }</td>
</tr>
<tr>
<td>{ p }</td>
<td>{ p }</td>
<td>{ p }</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

\[ X \]

\[ P^X \]

\[ Cn(P^X) \]
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( Cn(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ } { }</td>
<td>( p \leftarrow )</td>
<td>{p}</td>
</tr>
<tr>
<td>{p}</td>
<td>( \neg p )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>( p \leftarrow )</td>
<td>{p} ( \times )</td>
</tr>
<tr>
<td>{p}</td>
<td>( p \leftarrow )</td>
<td>( \emptyset ) ( \times )</td>
</tr>
</tbody>
</table>
Example three

\[ P = \{ p \leftarrow \neg p \} \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P^X )</th>
<th>( \text{Cn}(P^X) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ }</td>
<td>{p}</td>
<td>(\times)</td>
</tr>
<tr>
<td>{p}</td>
<td>(\emptyset)</td>
<td>(\checkmark)</td>
</tr>
</tbody>
</table>
Examples

Some properties

- A logic program may have zero, one, or multiple stable models

  - If \( X \) is a stable model of a logic program \( P \), then \( X \subseteq h(P) \)

  - If \( X \) is a stable model of a logic program \( P \), then \( X \) is a (classical) model of \( P \)

  - If \( X \) and \( Y \) are stable models of a normal program \( P \), then \( X \not\subset Y \)
Some properties

- A logic program may have zero, one, or multiple stable models

- If $X$ is a stable model of a logic program $P$, then $X \subseteq h(P)$

- If $X$ is a stable model of a logic program $P$, then $X$ is a (classical) model of $P$

- If $X$ and $Y$ are stable models of a normal program $P$, then $X \not\subset Y$
Some properties

- A logic program may have zero, one, or multiple stable models.

- If $X$ is a stable model of a logic program $P$, then $X \subseteq h(P)$.

- If $X$ is a stable model of a logic program $P$, then $X$ is a (classical) model of $P$.

- If $X$ and $Y$ are stable models of a normal program $P$, then $X \not\subset Y$. 

- Torsten Schaub (KRR@UP)
Some properties

- A logic program may have zero, one, or multiple stable models.

- If $X$ is a stable model of a logic program $P$, then $X \subseteq h(P)$.

- If $X$ is a stable model of a logic program $P$, then $X$ is a (classical) model of $P$.

- If $X$ and $Y$ are stable models of a normal program $P$, then $X \nsubseteq Y$.
### Exemplars

<table>
<thead>
<tr>
<th>Logic program</th>
<th>Answer sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>{a}</td>
</tr>
<tr>
<td>a :- b.</td>
<td>{}</td>
</tr>
<tr>
<td>a :- b. b.</td>
<td>{a,b}</td>
</tr>
<tr>
<td>a :- b. b :- a.</td>
<td>{}</td>
</tr>
<tr>
<td>a :- not c.</td>
<td>{a}</td>
</tr>
<tr>
<td>a :- not c. c.</td>
<td>{c}</td>
</tr>
<tr>
<td>a :- not c. c :- not a.</td>
<td>{a}, {c}</td>
</tr>
<tr>
<td>a :- not a.</td>
<td></td>
</tr>
</tbody>
</table>
Reasoning modes

- Problem
  - Modeling
  - Logic Program

- Solution
  - Interpreting
  - Stable Models

- Solving
Reasoning modes

- Satisfiability
- Enumeration\(^\dagger\)
- Projection\(^\dagger\)
- Intersection\(^\ddagger\)
- Union\(^\ddagger\)
- Optimization

and combinations of them

\(^\dagger\) without solution recording
\(^\ddagger\) without solution enumeration
Extended syntax

Problem  \rightarrow  Logic Program  \rightarrow  Stable Models

Modeling

Solution  \rightarrow  Stable Models

Interpreting

Solving
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  \[ \sim q(X), p(X,C) [C] \]
  \[ \#minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- **Variables**
  
- **Conditional literals**
  
- **Disjunction**
  
- **Integrity constraints**
  
- **Choice**
  
- **Aggregates**
  
- **Optimization**

```prolog
p(X) :- q(X)
p :- q(X) :- r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7
```

```prolog
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  \[ \sim q(X), p(X,C) [C] \]
  \[ \#\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
- Conditional literals
- Disjunction
- Integrity constraints
- Choice
- Aggregates
- Optimization

```
p(X) :- q(X)
p :- q(X) : r(X)
p(X) ; q(X) :- r(X)
:- q(X), p(X)
2 { p(X,Y) : q(X) } 7 :- r(Y)
s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7

:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```
Language constructs

- **Variables**
  \[ p(X) :- q(X) \]

- **Conditional literals**
  \[ p :- q(X) : r(X) \]

- **Disjunction**
  \[ p(X) ; q(X) :- r(X) \]

- **Integrity constraints**
  \[ :- q(X), p(X) \]

- **Choice**
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- **Aggregates**
  \[ s(Y) :- r(Y), 2 \sum\{ X : p(X,Y), q(X) \} 7 \]

- **Optimization**
  \[ ::= q(X), p(X,C) [C] \]
  \[ #\minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  \[ \sim q(X), p(X,C) [C] \]
  \[ \#minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  - Weak constraints
    \[ \sim q(X), p(X,C) [C] \]
    \[ #\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- Optimization
  - Weak constraints
    \[ :- q(X), p(X,C) [C] \]
  - Statements
    \[ #minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- **Variables**
  
  \[ p(X) :- q(X) \]

- **Conditional literals**
  
  \[ p :- q(X) : r(X) \]

- **Disjunction**
  
  \[ p(X) ; q(X) :- r(X) \]

- **Integrity constraints**
  
  \[ :- q(X), p(X) \]

- **Choice**
  
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- **Aggregates**
  
  \[ s(Y) :- r(Y), 2 \#sum\{ X : p(X,Y), q(X) \} 7 \]

- **Optimization**
  
  - **Weak constraints**
    
    \[ :- q(X), p(X,C) [C] \]

  - **Statements**
    
    \[ #\text{minimize} \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \#sum{ X : p(X,Y), q(X) } 7 \]

- Optimization
  - Weak constraints
    \[ :\sim q(X), p(X,C) [C] \]
  - Statements
    \[ \#minimize \{ C : q(X), p(X,C) \} \]
Language constructs

- Variables
  \[ p(X) :- q(X) \]

- Conditional literals
  \[ p :- q(X) : r(X) \]

- Disjunction
  \[ p(X) ; q(X) :- r(X) \]

- Integrity constraints
  \[ :- q(X), p(X) \]

- Choice
  \[ 2 \{ p(X,Y) : q(X) \} 7 :- r(Y) \]

- Aggregates
  \[ s(Y) :- r(Y), 2 \sum\{ X : p(X,Y), q(X) \} 7 \]

- Multi-objective optimization
  - Weak constraints
    \[ : \sim q(X), p(X,C) [C@42] \]
  - Statements
    \[ \#minimize \{ C@42 : q(X), p(X,C) \} \]
Outline

9 Syntax
10 Semantics
11 Examples
12 Reasoning
13 Language
14 Variables
Example

d(a)
d(c)
d(d)
p(a, b)
p(b, c)
p(c, d)
p(X, Z) ← p(X, Y), p(Y, Z)
q(a)
q(b)
q(X) ← ∼r(X), d(X)
r(X) ← ∼q(X), d(X)
s(X) ← ∼r(X), p(X, Y), q(Y)
Example

d(a)
d(c)
d(d)
p(a, b)
p(b, c)
p(c, d)
p(X, Z) ← p(X, Y), p(Y, Z)
q(a)
q(b)
q(X) ← ∼r(X), d(X)
r(X) ← ∼q(X), d(X)
s(X) ← ∼r(X), p(X, Y), q(Y)
Grounding instantiation

Let \( P \) be a logic program

- Let \( \mathcal{T} \) be a set of (variable-free) terms
- Let \( \mathcal{A} \) be a set of (variable-free) atoms constructible from \( \mathcal{T} \)
- A variable-free atom is also called ground

Ground instances of \( r \in P \): Set of variable-free rules obtained by replacing all variables in \( r \) by elements from \( \mathcal{T} \):

\[
ground(r) = \{ r\theta \mid \theta : \text{var}(r) \rightarrow \mathcal{T} \text{ and } \text{var}(r\theta) = \emptyset \}
\]

where \( \text{var}(r) \) stands for the set of all variables occurring in \( r \); \( \theta \) is a (ground) substitution

Ground instantiation of \( P \): \( \ground(P) = \bigcup_{r \in P} \ground(r) \)
Grounding instantiation

Let $P$ be a logic program

- Let $\mathcal{T}$ be a set of variable-free terms (also called Herbrand universe)
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$ (also called alphabet or Herbrand base)

- A variable-free atom is also called ground

- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in $r$ by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

where $var(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

- Ground instantiation of $P$: $ground(P) = \bigcup_{r \in P} ground(r)$
Grounding instantiation

Let $P$ be a logic program

- Let $T$ be a set of (variable-free) terms
- Let $A$ be a set of (variable-free) atoms constructible from $T$
- A variable-free atom is also called ground

- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in $r$ by elements from $T$:

$$\text{ground}(r) = \{ r\theta \mid \theta : \text{var}(r) \rightarrow T \text{ and } \text{var}(r\theta) = \emptyset \}$$

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

- Ground instantiation of $P$: $\text{ground}(P) = \bigcup_{r \in P} \text{ground}(r)$
Grounding instantiation

Let $P$ be a logic program

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground

- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in $r$ by elements from $\mathcal{T}$:

$$\text{ground}(r) = \{ r\theta \mid \theta : \text{var}(r) \rightarrow \mathcal{T} \text{ and } \text{var}(r\theta) = \emptyset \}$$

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

- Ground instantiation of $P$: $\text{ground}(P) = \bigcup_{r \in P} \text{ground}(r)$
Grounding instantiation

Let $P$ be a logic program

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of $r \in P$: Set of variable-free rules obtained by replacing all variables in $r$ by elements from $\mathcal{T}$:

$$ground(r) = \{ r\theta \mid \theta : \text{var}(r) \rightarrow \mathcal{T} \text{ and } \text{var}(r\theta) = \emptyset \}$$

where $\text{var}(r)$ stands for the set of all variables occurring in $r$; $\theta$ is a (ground) substitution

- Ground instantiation of $P$: $ground(P) = \bigcup_{r \in P} ground(r)$
An example

\[ P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \} \]

\[ T = \{ a, b, c \} \]

\[ A = \{ r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \} \]

\[ \text{ground}(P) = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), t(a, b) \leftarrow r(a, b), t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c) \} \]

Grounding aims at reducing the ground instantiation
An example

\[ P = \{ \, r(a, b) \leftarrow, \, r(b, c) \leftarrow, \, t(X, Y) \leftarrow r(X, Y) \, \} \]
\[ T = \{ a, b, c \} \]
\[ A = \{ \, r(a, a), \, r(a, b), \, r(a, c), \, r(b, a), \, r(b, b), \, r(b, c), \, r(c, a), \, r(c, b), \, r(c, c), \, t(a, a), \, t(a, b), \, t(a, c), \, t(b, a), \, t(b, b), \, t(b, c), \, t(c, a), \, t(c, b), \, t(c, c) \, \} \]

\[ \text{ground}(P) = \left\{ \begin{array}{l}
    r(a, b) \leftarrow , \\
    r(b, c) \leftarrow , \\
    t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), \\
    t(a, b) \leftarrow r(a, b), t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), \\
    t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c)
\end{array} \right\} \]

Grounding aims at reducing the ground instantiation
Variables

An example

\[ P = \{ r(a, b) \leftarrow, \ r(b, c) \leftarrow, \ t(X, Y) \leftarrow r(X, Y) \} \]

\[ T = \{ a, b, c \} \]

\[ A = \left\{ \begin{array}{l}
     r(a, a), \ r(a, b), \ r(a, c), \ r(b, a), \ r(b, b), \ r(b, c), \ r(c, a), \ r(c, b), \ r(c, c), \\
     t(a, a), \ t(a, b), \ t(a, c), \ t(b, a), \ t(b, b), \ t(b, c), \ t(c, a), \ t(c, b), \ t(c, c)
\end{array} \right\} \]

\[ \text{ground}(P) = \left\{ \begin{array}{l}
     r(a, b) \leftarrow, \\
     r(b, c) \leftarrow, \\
     t(a, a) \leftarrow r(a, a), \ t(b, a) \leftarrow r(b, a), \ t(c, a) \leftarrow r(c, a), \\
     t(a, b) \leftarrow r(a, b), \ t(b, b) \leftarrow r(b, b), \ t(c, b) \leftarrow r(c, b), \\
     t(a, c) \leftarrow r(a, c), \ t(b, c) \leftarrow r(b, c), \ t(c, c) \leftarrow r(c, c)
\end{array} \right\} \]

- Grounding aims at reducing the ground instantiation
An example

\[ P = \{ r(a, b) \leftarrow, r(b, c) \leftarrow, t(X, Y) \leftarrow r(X, Y) \} \]
\[ T = \{ a, b, c \} \]
\[ A = \{ r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \]
\[ t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \} \]

\[ \text{ground}(P) = \{ \]
\[ r(a, b) \leftarrow, \]
\[ r(b, c) \leftarrow, \]
\[ t(a, a) \leftarrow r(a, a), t(b, a) \leftarrow r(b, a), t(c, a) \leftarrow r(c, a), \]
\[ t(a, b) \leftarrow, t(b, b) \leftarrow r(b, b), t(c, b) \leftarrow r(c, b), \]
\[ t(a, c) \leftarrow r(a, c), t(b, c) \leftarrow r(b, c), t(c, c) \leftarrow r(c, c) \} \]

- Grounding aims at reducing the ground instantiation
An example

\[ P = \{ \text{r}(a, b) \leftarrow, \text{r}(b, c) \leftarrow, \text{t}(X, Y) \leftarrow \text{r}(X, Y) \} \]

\[ T = \{ a, b, c \} \]

\[ A = \{ \text{r}(a, a), \text{r}(a, b), \text{r}(a, c), \text{r}(b, a), \text{r}(b, b), \text{r}(b, c), \text{r}(c, a), \text{r}(c, b), \text{r}(c, c), \text{t}(a, a), \text{t}(a, b), \text{t}(a, c), \text{t}(b, a), \text{t}(b, b), \text{t}(b, c), \text{t}(c, a), \text{t}(c, b), \text{t}(c, c) \} \]

\[ \text{ground}(P) = \begin{cases} 
\text{r}(a, b) \leftarrow, \\
\text{r}(b, c) \leftarrow, \\
\text{t}(a, a) \leftarrow \text{r}(a, a), \text{t}(b, a) \leftarrow \text{r}(b, a), \text{t}(c, a) \leftarrow \text{r}(c, a), \\
\text{t}(a, b) \leftarrow, \text{t}(b, b) \leftarrow \text{r}(b, b), \text{t}(c, b) \leftarrow \text{r}(c, b), \\
\text{t}(a, c) \leftarrow \text{r}(a, c), \text{t}(b, c) \leftarrow \text{r}(b, c), \text{t}(c, c) \leftarrow \text{r}(c, c) 
\end{cases} \]

- **Grounding** aims at reducing the ground instantiation
A normal rule is **safe**, if each of its variables also occurs in some positive body literal.

A normal program is safe, if all of its rules are safe.
$d(a)$
$d(c)$
$d(d)$
$p(a, b)$
$p(b, c)$
$p(c, d)$
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$
$q(a)$
$q(b)$
$q(X) \leftarrow \sim r(X)$
$r(X) \leftarrow \sim q(X), d(X)$
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$
Example

Safe ?

d(a)
d(c)
d(d)
p(a, b)
p(b, c)
p(c, d)
p(X, Z) ← p(X, Y), p(Y, Z)

q(a)
q(b)
q(X) ← ¬r(X)

r(X) ← ¬q(X), d(X)
s(X) ← ¬r(X), p(X, Y), q(Y)
Example

Variables

\[
\begin{align*}
  d(a) & \quad \checkmark \\
  d(c) & \quad \checkmark \\
  d(d) & \quad \checkmark \\
  p(a, b) & \quad \checkmark \\
  p(b, c) & \quad \checkmark \\
  p(c, d) & \quad \checkmark \\
  p(X, Z) & \leftarrow p(X, Y), p(Y, Z) \quad \checkmark \\
  q(a) & \quad \checkmark \\
  q(b) & \quad \checkmark \\
  q(X) & \leftarrow \sim r(X) \quad \checkmark \\
  r(X) & \leftarrow \sim q(X), d(X) \\
  s(X) & \leftarrow \sim r(X), p(X, Y), q(Y) 
\end{align*}
\]
Example

\begin{align*}
  d(a) & \quad \checkmark \\
  d(c) & \quad \checkmark \\
  d(d) & \quad \checkmark \\
  p(a, b) & \quad \checkmark \\
  p(b, c) & \quad \checkmark \\
  p(c, d) & \quad \checkmark \\
  p(X, Z) & \leftarrow p(X, Y), p(Y, Z) \\
  q(a) & \quad \checkmark \\
  q(b) & \quad \checkmark \\
  q(X) & \leftarrow \neg r(X) \\
  r(X) & \leftarrow \neg q(X), d(X) \\
  s(X) & \leftarrow \neg r(X), p(X, Y), q(Y)
\end{align*}
Variables

Example

Safe ?

\[
d(a) \checkmark \\
d(c) \checkmark \\
d(d) \checkmark \\
p(a, b) \checkmark \\
p(b, c) \checkmark \\
p(c, d) \checkmark \\
p(X, Z) \leftarrow p(X, Y), p(Y, Z) \checkmark \\
q(a) \checkmark \\
q(b) \checkmark \\
q(X) \leftarrow \sim r(X) \checkmark \\
r(X) \leftarrow \sim q(X), d(X) \\
s(X) \leftarrow \sim r(X), p(X, Y), q(Y)
\]
Example

Safe ?

\[
\begin{align*}
&d(a) & \checkmark \\
&d(c) & \checkmark \\
&d(d) & \checkmark \\
&p(a, b) & \checkmark \\
&p(b, c) & \checkmark \\
&p(c, d) & \checkmark \\
&p(X, Z) & \leftarrow p(X, Y), p(Y, Z) & \checkmark \\
&q(a) & \checkmark \\
&q(b) & \checkmark \\
&q(X) & \leftarrow \sim r(X) & \times \\
&r(X) & \leftarrow \sim q(X), d(X) \\
&s(X) & \leftarrow \sim r(X), p(X, Y), q(Y) \\
\end{align*}
\]
Example

Safe ?

\[
\begin{align*}
d(a) & \quad \checkmark \\
d(c) & \quad \checkmark \\
d(d) & \quad \checkmark \\
p(a, b) & \quad \checkmark \\
p(b, c) & \quad \checkmark \\
p(c, d) & \quad \checkmark \\
p(X, Z) & \leftarrow p(X, Y), p(Y, Z) & \checkmark \\
q(a) & \quad \checkmark \\
q(b) & \quad \checkmark \\
q(X) & \leftarrow \sim r(X), d(X) & \checkmark \\
r(X) & \leftarrow \sim q(X), d(X) \\
s(X) & \leftarrow \sim r(X), p(X, Y), q(Y)
\end{align*}
\]
Variables

Example

Safe?

\[
\begin{align*}
  &d(a) \\
  &d(c) \\
  &d(d) \\
  &p(a, b) \\
  &p(b, c) \\
  &p(c, d) \\
  &p(X, Z) \leftarrow p(X, Y), p(Y, Z) \\
  &q(a) \\
  &q(b) \\
  &q(X) \leftarrow \neg r(X), d(X) \\
  &r(X) \leftarrow \neg q(X), d(X) \\
  &s(X) \leftarrow \neg r(X), p(X, Y), q(Y)
\end{align*}
\]
Example

Variables

Safe ?

d(a)  ✓
d(c)  ✓
d(d)  ✓
p(a, b)  ✓
p(b, c)  ✓
p(c, d)  ✓
p(X, Z) ← p(X, Y), p(Y, Z)  ✓
q(a)  ✓
q(b)  ✓
q(X) ← ¬r(X), d(X)  ✓
r(X) ← ¬q(X), d(X)  ✓
s(X) ← ¬r(X), p(X, Y), q(Y)  ✓
Example

<table>
<thead>
<tr>
<th>Safe?</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

```
Variables

\[ d(a) \]
\[ d(c) \]
\[ d(d) \]
\[ p(a, b) \]
\[ p(b, c) \]
\[ p(c, d) \]
\[ p(X, Z) \leftarrow p(X, Y), p(Y, Z) \]
\[ q(a) \]
\[ q(b) \]
\[ q(X) \leftarrow \sim r(X), d(X) \]
\[ r(X) \leftarrow \sim q(X), d(X) \]
\[ s(X) \leftarrow \sim r(X), p(X, Y), q(Y) \]
```
Example

Safe ?

\[
\begin{align*}
&d(a) &\checkmark \\
&d(c) &\checkmark \\
&d(d) &\checkmark \\
&p(a, b) &\checkmark \\
&p(b, c) &\checkmark \\
&p(c, d) &\checkmark \\
&p(X, Z) \leftarrow p(X, Y), p(Y, Z) &\checkmark \\
&q(a) &\checkmark \\
&q(b) &\checkmark \\
&q(X) \leftarrow \neg r(X), d(X) &\checkmark \\
&r(X) \leftarrow \neg q(X), d(X) &\checkmark \\
&s(X) \leftarrow \neg r(X), p(X, Y), q(Y) &\checkmark 
\end{align*}
\]
Stable models of programs with Variables

Let $P$ be a normal logic program with variables

- A set $X$ of (ground) atoms is a stable model of $P$, if $\text{Cn}(\text{ground}(P)^X) = X$
Stable models of programs with Variables

Let $P$ be a normal logic program with variables

- A set $X$ of (ground) atoms is a stable model of $P$, if $Cn(\text{ground}(P)^X) = X$
Basic Modeling: Overview

15 Elaboration tolerance
16 ASP solving process
17 Methodology
18 Case studies
Modeling and Interpreting Logic Program Solution

Stable Models

Modeling

Logic Program

Problem

Solving

Interpreting

Solution
Elaboration tolerance

Outline

15 Elaboration tolerance
16 ASP solving process
17 Methodology
18 Case studies
Elaboration tolerance

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

  "A formalism is elaboration tolerant [if] it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."

- Uniform problem representation

  For solving a problem instance $I$ of a problem class $C$,
  - $I$ is represented as a set of facts $P_I$,
  - $C$ is represented as a set of rules $P_C$, and
  - $P_C$ can be used to solve all problem instances in $C$
Elaboration Tolerance (McCarthy, 1998)

“A formalism is elaboration tolerant [if] it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances.”

Uniform problem representation

For solving a problem instance $I$ of a problem class $C$,
- $I$ is represented as a set of facts $P_I$,
- $C$ is represented as a set of rules $P_C$, and
- $P_C$ can be used to solve all problem instances in $C$
Outline

15 Elaboration tolerance
16 ASP solving process
17 Methodology
18 Case studies
ASP solving process

Problem

Logic Program

Grounder

Solver

Stable Models

Solution

Modeling

Solving

Interpreting
ASP solving process

Problem

Logic Program

Grounder

Solver

Stable Models

Solution

Modeling

Interpreting

Solving
A case-study: Graph coloring

Problem

Logic Program

Solving

Grounder

Solver

Stable Models

Solution

Modeling

Interpreting

Torsten Schaub (KRR@UP)

Answer Set Solving in Practice

February 18, 2019
Graph coloring

Problem instance  A graph consisting of nodes and edges
Graph coloring

- Problem instance: A graph consisting of nodes and edges
Graph coloring

- Problem instance: A graph consisting of nodes and edges
Graph coloring

- Problem instance: A graph consisting of nodes and edges
- facts formed by predicates \texttt{node/1} and \texttt{edge/2}
Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates node/1 and edge/2
  - facts formed by predicate color/1
Graph coloring

- Problem instance  A graph consisting of nodes and edges
  - facts formed by predicates node/1 and edge/2
  - facts formed by predicate color/1

- Problem class  Assign each node one color such that no two nodes connected by an edge have the same color
Graph coloring

- **Problem instance** A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`

- **Problem class** Assign each node one color such that no two nodes connected by an edge have the same color

  In other words,
  1. Each node has one color
  2. Two connected nodes must not have the same color
ASP solving process

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models

Modeling

Solving

Interpreting

Solution
Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

\{ \text{assign}(N,C) : \text{color}(C) \} = 1 :- \text{node}(N).

:- edge(N,M), \text{assign}(N,C), \text{assign}(M,C).
Graph coloring

node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).  color(b).  color(g).

\{ assign(N,C) : color(C) \} = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
ASP solving process

Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

\{ \text{assign}(N,C) : \text{color}(C) \} = 1 :- \text{node}(N).

:- \text{edge}(N,M), \text{assign}(N,C), \text{assign}(M,C).
Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

\{ assign(N,C) : color(C) \} = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
ASP solving process

Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
ASP solving process

Graph coloring

node(1..6).

dge(1,2). edge(1,3). edge(1,4).
dge(2,4). edge(2,5). edge(2,6).
dge(3,1). edge(3,4). edge(3,5).
dge(4,1). edge(4,2).
dge(5,3). edge(5,4). edge(5,6).
dge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

\{ assign(N,C) : color(C) \} = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
Graph coloring

node(1..6).
edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
Graph coloring

node(1..6).

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
ASP solving process

Problem

Logic Program

Grounder

Solver

Stable Models

Solution

Modeling

Interpreting

Solving
ASP solving process

Graph coloring: Grounding

$ gringo --text graph.lp color.lp

node(1). node(2). node(3). node(4). node(5). node(6).

dge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
dge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
dge(1,4). edge(2,6). edge(3,5). edge(5,6). edge(6,5).

color(r). color(b). color(g).

\{ assign(1,r), assign(1,b), assign(1,g) \} = 1. \{ assign(4,r), assign(4,b), assign(4,g) \} = 1.
\{ assign(2,r), assign(2,b), assign(2,g) \} = 1. \{ assign(5,r), assign(5,b), assign(5,g) \} = 1.
\{ assign(3,r), assign(3,b), assign(3,g) \} = 1. \{ assign(6,r), assign(6,b), assign(6,g) \} = 1.

\[\begin{align*}
\text{:- assign(1,r), assign(2,r).} \\
\text{:- assign(1,b), assign(2,b).} \\
\text{:- assign(1,g), assign(2,g).} \\
\text{:- assign(1,r), assign(3,r).} \\
\text{:- assign(1,b), assign(3,b).} \\
\text{:- assign(1,g), assign(3,g).} \\
\text{:- assign(1,r), assign(4,r).} \\
\text{:- assign(1,b), assign(4,b).} \\
\text{:- assign(1,g), assign(4,g).} \\
\text{:- assign(2,r), assign(4,r).} \\
\text{:- assign(2,b), assign(4,b).} \\
\text{:- assign(2,g), assign(4,g).} \\
\text{:- assign(2,r), assign(5,r).} \\
\text{:- assign(2,b), assign(5,b).} \\
\text{:- assign(2,g), assign(5,g).} \\
\text{:- assign(2,r), assign(6,r).} \\
\text{:- assign(2,b), assign(6,b).} \\
\text{:- assign(2,g), assign(6,g).} \\
\end{align*}\]
Graph coloring: Grounding

$ gringo --text graph.lp color.lp
	node(1). node(2). node(3). node(4). node(5). node(6).

type(1,red). type(1,blue). type(1,green).


type(2,red). type(2,blue). type(2,green).

type(5,red). type(5,blue). type(5,green).

type(3,red). type(3,blue). type(3,green).

type(6,red). type(6,blue). type(6,green).

:- type(1,red), type(2,red).
:- type(2,red), type(4,red).
:- type(6,red), type(2,red).
:- type(1,blue), type(2,blue).
:- type(2,blue), type(4,blue).
:- type(6,blue), type(2,blue).
:- type(1,green), type(2,green).
:- type(2,green), type(4,green).
:- type(6,green), type(2,green).
:- type(1,red), type(4,red).
:- type(2,red), type(6,red).
:- type(6,red), type(5,red).
:- type(1,blue), type(4,blue).
:- type(2,blue), type(6,blue).
:- type(6,blue), type(5,blue).
:- type(1,green), type(4,green).
:- type(2,green), type(6,green).
:- type(6,green), type(5,green).
ASP solving process

Graph coloring: Grounding

$ gringo --text graph.lp color.lp

node(1). node(2). node(3). node(4). node(5). node(6).

dge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5). edge(5,6). edge(6,5).

color(r). color(b). color(g).

{ assign(1,r), assign(1,b), assign(1,g) } = 1. { assign(4,r), assign(4,b), assign(4,g) } = 1.
{ assign(2,r), assign(2,b), assign(2,g) } = 1. { assign(5,r), assign(5,b), assign(5,g) } = 1.
{ assign(3,r), assign(3,b), assign(3,g) } = 1. { assign(6,r), assign(6,b), assign(6,g) } = 1.

:- assign(1,r), assign(2,r). :- assign(1,b), assign(2,b). ...
:- assign(1,r), assign(3,r). :- assign(1,b), assign(3,b). ...
:- assign(1,r), assign(4,r). :- assign(1,b), assign(4,b). ...
:- assign(1,r), assign(5,r). :- assign(1,b), assign(5,b). ...
:- assign(1,r), assign(6,r). :- assign(1,b), assign(6,b). ...
:- assign(1,r), assign(6,g). :- assign(1,b), assign(6,g). ...
Graph coloring: Grounding

$ gringo --text graph.lp color.lp
	node(1). node(2). node(3). node(4). node(5). node(6).

type(1, r). type(1, b). type(1, g).

type(2, r). type(2, b). type(2, g).

type(3, r). type(3, b). type(3, g).

type(4, r). type(4, b). type(4, g).

type(5, r). type(5, b). type(5, g).

type(6, r). type(6, b). type(6, g).

{ assign(1, r), assign(2, r), assign(3, r) } = 1.
{ assign(4, r), assign(5, r), assign(6, r) } = 1.
{ assign(1, b), assign(2, b), assign(3, b) } = 1.
{ assign(4, b), assign(5, b), assign(6, b) } = 1.
{ assign(1, g), assign(2, g), assign(3, g) } = 1.
{ assign(4, g), assign(5, g), assign(6, g) } = 1.

:- assign(1, r), assign(2, r). :- assign(2, r), assign(4, r).
:- assign(1, b), assign(2, b). :- assign(2, b), assign(4, b).
:- assign(1, g), assign(2, g). :- assign(2, g), assign(4, g).
:- assign(1, r), assign(3, r). :- assign(2, r), assign(5, r).
:- assign(1, b), assign(3, b). :- assign(2, b), assign(5, b).
:- assign(1, g), assign(3, g). :- assign(2, g), assign(5, g).
:- assign(1, r), assign(4, r). :- assign(2, r), assign(6, r).
:- assign(1, b), assign(4, b). :- assign(2, b), assign(6, b).
:- assign(1, g), assign(4, g). :- assign(2, g), assign(6, g).
Graph coloring: Grounding

$ clingo --text graph.lp color.lp
	node(1). node(2). node(3). node(4). node(5). node(6).
	edge(1,2). edge(2,4). edge(3,1). edge(4,1). edge(5,3). edge(6,2).
edge(1,3). edge(2,5). edge(3,4). edge(4,2). edge(5,4). edge(6,3).
edge(1,4). edge(2,6). edge(3,5). edge(5,6). edge(6,5).

color(r). color(b). color(g).

{ assign(1,r), assign(1,b), assign(1,g) } = 1. { assign(4,r), assign(4,b), assign(4,g) } = 1.
{ assign(2,r), assign(2,b), assign(2,g) } = 1. { assign(5,r), assign(5,b), assign(5,g) } = 1.
{ assign(3,r), assign(3,b), assign(3,g) } = 1. { assign(6,r), assign(6,b), assign(6,g) } = 1.

:- assign(1,r), assign(2,r). :- assign(2,r), assign(4,r). [...] :- assign(6,r), assign(2,r).
:- assign(1,b), assign(2,b). :- assign(2,b), assign(4,b). :- assign(6,b), assign(2,b).
:- assign(1,g), assign(2,g). :- assign(2,g), assign(4,g). :- assign(6,g), assign(2,g).
:- assign(1,r), assign(3,r). :- assign(2,r), assign(5,r). :- assign(6,r), assign(3,r).
:- assign(1,b), assign(3,b). :- assign(2,b), assign(5,b). :- assign(6,b), assign(3,b).
:- assign(1,g), assign(3,g). :- assign(2,g), assign(5,g). :- assign(6,g), assign(3,g).
:- assign(1,r), assign(4,r). :- assign(2,r), assign(6,r). :- assign(6,r), assign(5,r).
:- assign(1,b), assign(4,b). :- assign(2,b), assign(6,b). :- assign(6,b), assign(5,b).
:- assign(1,g), assign(4,g). :- assign(2,g), assign(6,g). :- assign(6,g), assign(5,g).
ASP solving process

Modeling

Problem

Logic Program

Grounder

Solver

Solving

Solution

Interpreting

Stable Models
Graph coloring: Solving

$ gringo graph.lp color.lp | clasp 0

clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models : 6
Time : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
ASP solving process

Graph coloring: Solving

$ gringo graph.lp color.lp | clasp 0

clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models : 6
Time : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
Graph coloring: Solving

$ clingo graph.lp color.lp 0

clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models : 6
Time : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
ASP solving process

Modeling

Logic Program

Grounder

Solver

Stable Models

Solving

Problem

Interpreting

Solution

- Modeling

- Interpreting
A coloring

Answer: 6
node(1) [...] \nassign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
A coloring

Answer: 6
node(1) [...] \
assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
Outline

15 Elaboration tolerance
16 ASP solving process
17 Methodology
18 Case studies
Generate and Test (or: Guess and Check)

Generator Generate potential stable model candidates (typically through non-deterministic constructs)

Tester Eliminate invalid candidates (typically through integrity constraints)

Nutshell

Logic program = Data + Generator + Tester (+ Optimizer)
Basic methodology

Methodology

Generate and Test (or: Guess and Check)

Generator  Generate potential stable model candidates
(typically through non-deterministic constructs)

Tester    Eliminate invalid candidates
(typically through integrity constraints)

Nutshell

Logic program  =  Data + Generator + Tester ( + Optimizer)
Methodology

Graph coloring

node(1..6).

define edge relationships:

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).  color(b).  color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).

Problem instance

Problem encoding
Graph coloring

node(1..6).

data

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).  color(b).  color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
Graph coloring

node(1..6).

data

edge(1,2). edge(1,3). edge(1,4).
edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5).
edge(4,1). edge(4,2).
edge(5,3). edge(5,4). edge(5,6).
edge(6,2). edge(6,3). edge(6,5).

color(r). color(b). color(g).

generator

\{ assign(N,C) : color(C) \} = 1 :- node(N).

\textit{Tester}

:- edge(N,M), assign(N,C), assign(M,C).
Outline

15 Elaboration tolerance
16 ASP solving process
17 Methodology
18 Case studies
Outline

15 Elaboration tolerance

16 ASP solving process

17 Methodology

18 Case studies

- Satisfiability
- Queens
- Traveling salesperson
- Reviewer Assignment
- Planning
Case studies
Satisfiability

Satisfiability testing

- Problem Instance A propositional formula $\phi$ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

Example: Consider formula

$$(a \lor \neg b) \land (\neg a \lor b)$$

Logic Program

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Stable models</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}$</td>
<td>$\neg a, b$</td>
<td>$X_1 = {a, b}$</td>
</tr>
<tr>
<td>${b}$</td>
<td>$a, \neg b$</td>
<td>$X_2 = {}$</td>
</tr>
</tbody>
</table>
Satisfiability testing

- **Problem Instance** A propositional formula $\phi$ in CNF
- **Problem Class** Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

**Example:** Consider formula

$$(a \lor \neg b) \land (\neg a \lor b)$$

- **Logic Program**

<table>
<thead>
<tr>
<th><strong>Generator</strong></th>
<th><strong>Tester</strong></th>
<th><strong>Stable models</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}$</td>
<td>$\sim a, b$</td>
<td>$X_1 = {a, b}$</td>
</tr>
<tr>
<td>${b}$</td>
<td>$a, \sim b$</td>
<td>$X_2 = {}$</td>
</tr>
</tbody>
</table>
Satisfiability testing

- **Problem Instance**: A propositional formula $\phi$ in CNF
- **Problem Class**: Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

**Example**: Consider formula

$$(a \lor \neg b) \land (\neg a \lor b)$$

- **Logic Program**

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Stable models</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}$</td>
<td>$\neg a, b$</td>
<td>$X_1 = {a, b}$</td>
</tr>
<tr>
<td>${b}$</td>
<td>$a, \neg b$</td>
<td>$X_2 = {}$</td>
</tr>
</tbody>
</table>
Satisfiability testing

- **Problem Instance**: A propositional formula $\phi$ in CNF
- **Problem Class**: Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

**Example**: Consider formula

$$(a \lor \neg b) \land (\neg a \lor b)$$

**Logic Program**

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Stable models</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}$</td>
<td>$\Leftarrow$</td>
<td>$X_1 = {a, b}$</td>
</tr>
<tr>
<td>${b}$</td>
<td>$\Leftarrow$</td>
<td>$X_2 = {}$</td>
</tr>
<tr>
<td>$\neg a, b$</td>
<td>$\Leftarrow$</td>
<td></td>
</tr>
<tr>
<td>$a, \neg b$</td>
<td>$\Leftarrow$</td>
<td></td>
</tr>
</tbody>
</table>
Satisfiability testing

- **Problem Instance** A propositional formula $\phi$ in CNF
- **Problem Class** Is there an assignment of propositional variables to true and false such that a given formula $\phi$ is true

- **Example:** Consider formula

\[(a \lor \neg b) \land (\neg a \lor b)\]

- **Logic Program**

<table>
<thead>
<tr>
<th>Generator</th>
<th>Tester</th>
<th>Stable models</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a}$</td>
<td>$\sim a, b$</td>
<td>$X_1 = {a, b}$</td>
</tr>
<tr>
<td>${b}$</td>
<td>$a, \sim b$</td>
<td>$X_2 = {}$</td>
</tr>
</tbody>
</table>
Outline

15 Elaboration tolerance

16 ASP solving process

17 Methodology

18 Case studies
  - Satisfiability
  - Queens
  - Traveling salesperson
  - Reviewer Assignment
  - Planning
The n-queens problem

- Place $n$ queens on an $n \times n$ chess board
- Queens must not attack one another
Defining the field

queens.lp

row(1..n).
col(1..n).

- Create file queens.lp
- Define the field
  - $n$ rows
  - $n$ columns
Running...

$ clingo queens.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
SATISFIABLE

Models : 1
Time   : 0.000
Placing some queens

queens.lp

row(1..n).
col(1..n).
\{ queen(I,J) : row(I), col(J) \}.

- Guess a solution candidate
  by placing some queens on the board
Placing some queens

Running ...

$ clingo queens.lp --const n=5 3
Answer: 1
row(1) row(2) row(3) row(4) row(5) \n  col(1) col(2) col(3) col(4) col(5)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \n  col(1) col(2) col(3) col(4) col(5) queen(1,1)
Answer: 3
row(1) row(2) row(3) row(4) row(5) \n  col(1) col(2) col(3) col(4) col(5) queen(2,1)
SATISFIABLE

Models : 3+
Placing some queens

Answer: 1

\[
\begin{array}{ccccc}
5 & \cdot & \cdot & \cdot & \cdot \\
4 & \cdot & \cdot & \cdot & \cdot \\
3 & \cdot & \cdot & \cdot & \cdot \\
2 & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & \cdot & \cdot \\
\end{array}
\]

Answer: 1
row(1) row(2) row(3) row(4) row(5) \ \col(1) \ col(2) \ col(3) \ col(4) \ col(5)
Placing some queens

Answer: 2

row(1) row(2) row(3) row(4) row(5) \ col(1) col(2) col(3) col(4) col(5) \ queen(1,1)
Placing some queens

Answer: 3

Answer: 3
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(2,1)
Placing $n$ queens

queens.lp

\[
\text{row}(1..n). \\
\text{col}(1..n). \\
\{ \text{queen}(I,J) : \text{row}(I), \text{col}(J) \}. \\
:- \{ \text{queen}(I,J) \} \neq n.
\]

- Place exactly $n$ queens on the board
Placing \( n \) queens

queens.lp

\[
\begin{align*}
\text{row}(1..n). \\
\text{col}(1..n). \\
\{ \text{queen}(I,J) : \text{row}(I), \text{col}(J) \}. \\
:- \text{not} \{ \text{queen}(I,J) \} = n.
\end{align*}
\]

Place exactly \( n \) queens on the board
Running ...

$ clingo queens.lp --const n=5 2
Answer: 1
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5) \nqueen(5,1) queen(4,1) queen(3,1) queen(2,1) queen(1,1)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5) \nqueen(1,2) queen(4,1) queen(3,1) queen(2,1) queen(1,1)
Placing $n$ queens

Answer: 1

row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,1) queen(4,1) queen(3,1) queen(2,1) 
queen(1,1)
Placing $n$ queens

Answer: 2

```plaintext
row(1) row(2) row(3) row(4) row(5) \ncol(1) col(2) col(3) col(4) col(5) \nqueen(1,2) queen(4,1) queen(3,1) queen(2,1) 
queen(1,1)
```
Horizontal and vertical attack

queens.lp

```prolog
row(1..n).
col(1..n).
\{ queen(I,J) : row(I), col(J) \}.
:- \{ queen(I,J) \} != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
```

- Forbid horizontal attacks
- Forbid vertical attacks
Horizontal and vertical attack

queens.lp

row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J’), J != J’.

- Forbid horizontal attacks
- Forbid vertical attacks
Horizontal and vertical attack

Running ...

$ clingo queens.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,5) queen(4,4) queen(3,3) queen(2,2) queen(1,1)
Horizontal and vertical attack

Answer: 1

row(1) row(2) row(3) row(4) row(5) \ col(1) col(2) col(3) col(4) col(5) \ queen(5,5) queen(4,4) queen(3,3) queen(2,2) queen(1,1)
Case studies

Queens

Diagonal attack

queens.lp

```prolog
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.
```

- Forbid diagonal attacks
$ clingo queens.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(4,5) queen(1,4) queen(3,3) queen(5,2) queen(2,1)
SATISFIABLE

Models : 1+
Time    : 0.000
Diagonal attack

**Answer:** 1

```
5 5 5 5 5
4 4 4 4 4
3 3 3 3 3
2 2 2 2 2
1 1 1 1 1
1 2 3 4 5
```

```
Answer: 1
row(1) row(2) row(3) row(4) row(5) \ 
col(1) col(2) col(3) col(4) col(5) \ 
queen(4,5) queen(1,4) queen(3,3) queen(5,2) 
queen(2,1)
```
Encoding can be optimized

- Much faster to solve
And sometimes it rocks

$ clingo -c n=5000 queens-opt-diag.lp --config=jumpy -q --stats=2

clingo version 4.1.0
Solving...
SATISFIABLE

Models : 1+
Time : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time : 3758.320s

Choices : 288594554
Conflicts : 3442 (Analyzed: 3442)
Restarts : 17 (Average: 202.47 Last: 3442)
Model-Level : 7594728.0
Problems : 1 (Average Length: 0.00 Splits: 0)
Lemmas : 3442 (Deleted: 0)
  Binary : 0 (Ratio: 0.00%)
  Ternary : 0 (Ratio: 0.00%)
Conflict : 3442 (Average Length: 229056.5 Ratio: 100.00%)
Loop : 0 (Average Length: 0.0 Ratio: 0.00%)
Other : 0 (Average Length: 0.0 Ratio: 0.00%)

Atoms : 75084857 (Original: 75069989 Auxiliary: 14868)
Constraints : 66664 (Binary: 35.6% Ternary: 0.0% Other: 64.4%)

Backjumps : 3442 (Average: 681.19 Max: 169512 Sum: 2344658)
Executed : 3442 (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
Bounded : 0 (Average: 0.00 Max: 0 Sum: 0 Ratio: 0.00%)
And sometimes it rocks

$ clingo -c n=5000 queens-opt-diag.lp --config=jumpy -q --stats=2

clingo version 4.1.0
Solving...
SATISFIABLE

Models : 1+
Time : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time : 3758.320s

Choices : 288594554
Conflicts : 3442 (Analyzed: 3442)
Restarts : 17 (Average: 202.47 Last: 3442)
Model-Level : 7594728.0
Problems : 1 (Average Length: 0.00 Splits: 0)
Lemmas : 3442 (Deleted: 0)
  Binary : 0 (Ratio: 0.00%)
  Ternary : 0 (Ratio: 0.00%)
Conflict : 3442 (Average Length: 229056.5 Ratio: 100.00%)
Loop : 0 (Average Length: 0.0 Ratio: 0.00%)
Other : 0 (Average Length: 0.0 Ratio: 0.00%)

Atoms : 75084857 (Original: 75069989 Auxiliary: 14868)
Rules : 100129956 (1: 50059992/100090100 2: 39990/298563: 10000/10000)
Bodies : 25090103
Equivalences : 125029999 (Atom=Atom: 50009999 Body=Body: 0 Other: 75020000)
Tight : Yes
Variables : 25024868 (Eliminated: 11781 Frozen: 25000000)
Constraints : 66664 (Binary: 35.6% Ternary: 0.0% Other: 64.4%)

Backjumps : 3442 (Average: 681.19 Max: 169512 Sum: 2344658)
Executed : 3442 (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
Bounded : 0 (Average: 0.00 Max: 0 Sum: 0 Ratio: 0.00%)
Outline

15 Elaboration tolerance

16 ASP solving process

17 Methodology

18 Case studies
   - Satisfiability
   - Queens
   - **Traveling salesperson**
   - Reviewer Assignment
   - Planning
The traveling salesperson problem (TSP)

Problem Instance: A set of cities and distances among them, or simply a weighted graph.

Problem Class: What is the shortest possible route visiting each city once and returning to the city of origin?

Note:
- TSP extends the Hamiltonian cycle problem: Is there a cycle in a graph visiting each node exactly once.
- TSP is relevant to applications in logistics, planning, chip design, and the core of the vehicle routing problem.
The traveling salesperson problem (TSP)

- **Problem Instance**: A set of cities and distances among them, or simply a weighted graph.

- **Problem Class**: What is the shortest possible route visiting each city once and returning to the city of origin?

- **Note**
  - TSP extends the Hamiltonian cycle problem: Is there a cycle in a graph visiting each node exactly once?
  - TSP is relevant to applications in logistics, planning, chip design, and the core of the vehicle routing problem.
Traveling salesperson

node(1..6).

edge(1, (2;3;4)).  edge(2, (4;5;6)).  edge(3, (1;4;5)).
edge(4, (1;2)).  edge(5, (3;4;6)).  edge(6, (2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).
Traveling salesperson

node(1..6).

decorate(a, 1, (2;3;4)). decorate(b, 2, (4;5;6)). decorate(c, 3, (1;4;5)).
decorate(d, 4, (1;2)). decorate(e, 5, (3;4;6)). decorate(f, 6, (2;3;5)).
cost(1,2,2). cost(1,3,3). cost(1,4,1).
cost(2,4,2). cost(2,5,2). cost(2,6,4).
cost(3,1,3). cost(3,4,2). cost(3,5,2).
cost(4,1,1). cost(4,2,2).
cost(5,3,2). cost(5,4,2). cost(5,6,1).
cost(6,2,4). cost(6,3,3). cost(6,5,1).
Traveling salesperson

node(1..6).
edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).  edge(5,(3;4;6)).  edge(6,(2;3;5)).
cost(1,2,2).    cost(1,3,3).    cost(1,4,1).
cost(2,4,2).    cost(2,5,2).    cost(2,6,4).
cost(3,1,3).    cost(3,4,2).    cost(3,5,2).
cost(4,1,1).    cost(4,2,2).
cost(5,3,2).    cost(5,4,2).    cost(5,6,1).
cost(6,2,4).    cost(6,3,3).    cost(6,5,1).
Traveling salesperson

node(1..6).

edge(1,(2;3;4)). edge(2,(4;5;6)). edge(3,(1;4;5)).
edge(4,(1;2)). edge(5,(3;4;6)). edge(6,(2;3;5)).

cost(1,2,2). cost(1,3,3). cost(1,4,1).
cost(2,4,2). cost(2,5,2). cost(2,6,4).
cost(3,1,3). cost(3,4,2). cost(3,5,2).
cost(4,1,1). cost(4,2,2).
cost(5,3,2). cost(5,4,2). cost(5,6,1).
cost(6,2,4). cost(6,3,3). cost(6,5,1).

drive(X,Y) :- cost(X,Y,_).
Traveling salesperson

node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).  edge(5,(3;4;6)).  edge(6,(2;3;5)).

cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
cost(4,1,1).  cost(4,2,2).
cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
cost(6,2,4).  cost(6,3,3).  cost(6,5,1).

defined(X,Y) :- cost(X,Y,_).
node(X) :- cost(X,_,_).  node(Y) :- cost(_,Y,_,).
Traveling salesperson

\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 \quad \text{:- node}(X).
\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 \quad \text{:- node}(Y).

\text{reached}(Y) \quad \text{:- cycle}(1,Y).
\text{reached}(Y) \quad \text{:- cycle}(X,Y), \text{reached}(X).

\text{:- node}(Y), \text{not reached}(Y).

\#\text{minimize} \{ C,X,Y : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}.
Traveling salesperson

\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 :- \text{node}(X).
\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 :- \text{node}(Y).

\text{reached}(Y) :- \text{cycle}(1,Y).
\text{reached}(Y) :- \text{cycle}(X,Y), \text{reached}(X).

:- \text{node}(Y), \text{not} \text{reached}(Y).

#\text{minimize} \{ C,X,Y : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}.
Traveling salesperson

\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 : \neg \text{node}(X).
\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 : \neg \text{node}(Y).

\text{reached}(Y) : \neg \text{cycle}(1,Y).
\text{reached}(Y) : \neg \text{cycle}(X,Y), \text{reached}(X).

\neg \text{node}(Y), \neg \text{reached}(Y).

#\text{minimize} \{ C,X,Y : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}. 
Traveling salesperson

\[
\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 \quad \text{:-} \quad \text{node}(X).
\]

\[
\{ \text{cycle}(X,Y) : \text{edge}(X,Y) \} = 1 \quad \text{:-} \quad \text{node}(Y).
\]

\[
\text{reached}(Y) : \text{cycle}(1,Y).
\]

\[
\text{reached}(Y) : \text{cycle}(X,Y), \text{reached}(X).
\]

\[
\text{:-} \quad \text{node}(Y), \neg \text{reached}(Y).
\]

\[
\#\text{minimize} \quad \{ C,X,Y : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}.\]
Outline

15 Elaboration tolerance

16 ASP solving process

17 Methodology

18 Case studies
   - Satisfiability
   - Queens
   - Traveling salesperson
   - Reviewer Assignment
   - Planning
Reviewer Assignment

- **Problem Instance** A set of papers and a set of reviewers along with their first and second choices of papers and conflict of interests

- **Problem Class** A nice assignment of three reviewers to each paper
Reviewer Assignment

- **Problem Instance** A set of papers and a set of reviewers along with their first and second choices of papers and conflict of interests
- **Problem Class** A “nice” assignment of three reviewers to each paper
Reviewer Assignment
by Ilkka Niemelä

```prolog
paper(p1).  reviewer(r1).  classA(r1,p1).  classB(r1,p2).  coi(r1,p3).
paper(p2).  reviewer(r2).  classA(r2,p3).  classB(r2,p4).  coi(r2,p6).

{ assigned(P,R) : reviewer(R) } = 3 :- paper(P).

:- assigned(P,R), coi(R,P).
:- assigned(P,R), not classA(R,P), not classB(R,P).
:- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :- classB(R,P), assigned(P,R).
    :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Torsten Schaub (KRR@UP)  Answer Set Solving in Practice  February 18, 2019  122 / 653
Reviewer Assignment
by Ilkka Niemelä

\[
\begin{align*}
\text{paper}(p1). & \quad \text{reviewer}(r1). \quad \text{classA}(r1,p1). \quad \text{classB}(r1,p2). \quad \text{coi}(r1,p3). \\
\text{paper}(p2). & \quad \text{reviewer}(r2). \quad \text{classA}(r2,p3). \quad \text{classB}(r2,p4). \quad \text{coi}(r2,p6).
\end{align*}
\]

\[
\{ \text{assigned}(P,R) : \text{reviewer}(R) \} = 3 : - \quad \text{paper}(P).
\]

\[
\begin{align*}
& - \quad \text{assigned}(P,R), \quad \text{coi}(R,P). \\
& - \quad \text{assigned}(P,R), \quad \text{not classA}(R,P), \quad \text{not classB}(R,P). \\
& - \quad \text{not} \quad 6 \quad \{ \quad \text{assigned}(P,R) : \quad \text{paper}(P) \} \quad 9, \quad \text{reviewer}(R).
\end{align*}
\]

\[
\text{assignedB}(P,R) : - \quad \text{classB}(R,P), \quad \text{assigned}(P,R).
\]

\[
\begin{align*}
& - \quad 3 \quad \{ \quad \text{assignedB}(P,R) : \quad \text{paper}(P) \} , \quad \text{reviewer}(R).
\end{align*}
\]

\[
\#\text{minimize} \quad \{ \quad 1,P,R : \quad \text{assignedB}(P,R), \quad \text{paper}(P), \quad \text{reviewer}(R) \} .
\]
Reviewer Assignment
by Ilkka Niemelä

paper(p1). reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2). reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).

{ assigned(P,R) : reviewer(R) } = 3 :- paper(P).

:- assigned(P,R), coi(R,P).
:- assigned(P,R), not classA(R,P), not classB(R,P).
:- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :- classB(R,P), assigned(P,R).
:- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.

Torsten Schaub (KRR@UP)
Reviewer Assignment
by Ilkka Niemelä

\[
\text{paper}(p1). \text{ reviewer}(r1). \text{ classA}(r1,p1). \text{ classB}(r1,p2). \text{ coi}(r1,p3).
\]
\[
\text{paper}(p2). \text{ reviewer}(r2). \text{ classA}(r2,p3). \text{ classB}(r2,p4). \text{ coi}(r2,p6).
\]

\[
\{ \text{assigned}(P,R) : \text{reviewer}(R) \} = 3 :- \text{paper}(P).
\]
\[
:- \text{assigned}(P,R), \text{coi}(R,P).
\]
\[
:- \text{assigned}(P,R), \text{not classA}(R,P), \text{not classB}(R,P).
\]
\[
:- \text{not 6 \{} \text{assigned}(P,R) : \text{paper}(P) \} 9, \text{reviewer}(R).
\]

\[
\text{assignedB}(P,R) :- \text{classB}(R,P), \text{assigned}(P,R).
\]
\[
:- 3 \{ \text{assignedB}(P,R) : \text{paper}(P) \}, \text{reviewer}(R).
\]

\#minimize \{ 1,P,R : \text{assignedB}(P,R), \text{paper}(P), \text{reviewer}(R) \}.\]
case studies

Reviewer Assignment

by Ilkka Niemelä

paper(p1). reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2). reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).

{ assigned(P,R) : reviewer(R) } = 3 :- paper(P).

:- assigned(P,R), coi(R,P).
:- assigned(P,R), not classA(R,P), not classB(R,P).
:- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :- classB(R,P), assigned(P,R).
:- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
reviewer(r1). paper(p1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
reviewer(r2). paper(p2). classA(r1,p3). classB(r1,p4). coi(r1,p6).

[...]

#count { P,R : assigned(P,R) : reviewer(R) } = 3 :- paper(P).

:- assigned(P,R), coi(R,P).
:- assigned(P,R), not classA(R,P), not classB(R,P).
:- not 6 <= #count { P,R : assigned(P,R), paper(P) } <= 9, reviewer(R).

assignedB(P,R) :- classB(R,P), assigned(P,R).
:- 3 <= #count { P,R : assignedB(P,R), paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
reviewer(r1). paper(p1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
reviewer(r2). paper(p2). classA(r1,p3). classB(r1,p4). coi(r1,p6).

\[
\text{#count} \{ P,R : \text{assigned}(P,R) : \text{reviewer}(R) \} = 3 :- \text{paper}(P).
\]

:- \text{assigned}(P,R), \text{coi}(R,P).
:- \text{assigned}(P,R), \text{not classA}(R,P), \text{not classB}(R,P).
:- \text{not} 6 \leq \text{#count} \{ P,R : \text{assigned}(P,R), \text{paper}(P) \} \leq 9, \text{reviewer}(R).

\text{assignedB}(P,R) :- \text{classB}(R,P), \text{assigned}(P,R).
:- 3 \leq \text{#count} \{ P,R : \text{assignedB}(P,R), \text{paper}(P) \}, \text{reviewer}(R).

\text{#minimize} \{ 1,P,R : \text{assignedB}(P,R), \text{paper}(P), \text{reviewer}(R) \}.
Outline

15  Elaboration tolerance
16  ASP solving process
17  Methodology
18  Case studies
   - Satisfiability
   - Queens
   - Traveling salesperson
   - Reviewer Assignment
   - Planning
Simplified STRIPS\(^1\) Planning

- **Problem Instance**
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number \(k\) of allowed actions

- **Problem Class** Find a plan, that is, a sequence of \(k\) actions leading from the initial state to the goal state

- **Example**
  - fluents \(\{p, q, r\}\)
  - initial state \(\{p\}\)
  - goal state \(\{r\}\)
  - actions \(a = (\{p\}, \{q, \neg p\})\) and \(b = (\{q\}, \{r, \neg q\})\)
  - length 2
  - plan \(\langle a, b \rangle\)

\(^1\)Stanford Research Institute Problem Solver, 1971

Torsten Schaub (KRR@UP)
Simplified STRIPS\(^1\) Planning

- **Problem Instance**
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number \(k\) of allowed actions

- **Problem Class** Find a plan, that is, a sequence of \(k\) actions leading from the initial state to the goal state

- **Example**
  - fluents \(\{p, q, r\}\)
  - initial state \(\{p\}\)
  - goal state \(\{r\}\)
  - actions \(a = (\{p\}, \{q, \neg p\})\) and \(b = (\{q\}, \{r, \neg q\})\)
  - length 2
  - plan \(\langle a, b \rangle\)

---

\(^1\)Stanford Research Institute Problem Solver, 1971
Simplified STRIPS\(^1\) Planning

- **Problem Instance**
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number \(k\) of allowed actions

- **Problem Class** Find a plan, that is, a sequence of \(k\) actions leading from the initial state to the goal state

- **Example**
  - fluents \(\{p, q, r\}\)
  - initial state \(\{p\}\)
  - goal state \(\{r\}\)
  - actions \(a = (\{p\}, \{q, \neg p\})\) and \(b = (\{q\}, \{r, \neg q\})\)
  - length 2
  - plan \(\langle a, b \rangle\)

\(^1\)Stanford Research Institute Problem Solver, 1971
Simplified STRIPS\(^1\) Planning

- **Problem Instance**
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number \(k\) of allowed actions

- **Problem Class** Find a plan, that is, a sequence of \(k\) actions leading from the initial state to the goal state

- **Example**
  - fluents \(\{p, q, r\}\)
  - initial state \(\{p\}\)
  - goal state \(\{r\}\)
  - actions \(a = (\{p\}, \{q, \neg p\})\) and \(b = (\{q\}, \{r, \neg q\})\)
  - length 2
  - plan \(\langle a, b \rangle\)

\(^1\)Stanford Research Institute Problem Solver, 1971
Simplistic STRIPS Planning

time(1..k).

fluent(p). action(a). action(b). init(p).
fluent(q). pre(a,p). pre(b,q).
fluent(r). add(a,q). add(b,r). query(r).
del(a,p). del(b,q).

holds(P,0) :- init(P).

\{ occ(A,T) : action(A) \} = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
Simplistic STRIPS Planning

time(1..k).

fluent(p).  action(a).  action(b).  init(p).
fluent(q).  pre(a,p).  pre(b,q).
fluent(r).  add(a,q).  add(b,r).  query(r).
  del(a,p).  del(b,q).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
Simplistic STRIPS Planning

time(1..k).

fluent(p). action(a). action(b). init(p).
fluent(q). pre(a,p). pre(b,q).
fluent(r). add(a,q). add(b,r). query(r).
del(a,p). del(b,q).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
Simplistic STRIPS Planning

time(1..k).

fluent(p). action(a). action(b). init(p).
fluent(q). pre(a,p). pre(b,q).
fluent(r). add(a,q). add(b,r). query(r).
del(a,p). del(b,q).

holds(P,0) :- init(P).

\{ occ(A,T) : action(A) \} = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
Language: Overview

19 Motivation
20 Core language
21 Extended language
22 Intermediate formats
Motivation

Outline

19 Motivation

20 Core language

21 Extended language

22 Intermediate formats
Basic language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?

- A way of providing semantics is to furnish a translation removing the new constructs, e.g., classical negation.
- This translation might also be used for implementing the language extension.
Basic language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, e.g., classical negation.
- This translation might also be used for implementing the language extension.
Basic language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, e.g., classical negation.
- This translation might also be used for implementing the language extension.
Outline

19 Motivation
20 Core language
21 Extended language
22 Intermediate formats
Outline

19 Motivation

20 Core language
  - Integrity constraint
  - Choice rule
  - Cardinality rule
  - Weight rule

21 Extended language
  - Conditional literal
  - Optimization statement

22 Intermediate formats
  - smodels format
  - aspif format
Integrity constraint

- Idea: Eliminate unwanted solution candidates
- Syntax: An integrity constraint is of the form
  \[ \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]
  where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\)

- Example
  \[ :- \text{edge}(3,7), \text{color}(3,\text{red}), \text{color}(7,\text{red}). \]

- Example programs
  \[
  \{ a \leftarrow \sim b, \ b \leftarrow \sim a \} \\
  \{ a \leftarrow \sim b, \ b \leftarrow \sim a \} \cup \{ \leftarrow a \} \\
  \{ a \leftarrow \sim b, \ b \leftarrow \sim a \} \cup \{ \leftarrow \sim a \}
  \]
Integrity constraint

- **Idea** Eliminate unwanted solution candidates
- **Syntax** An integrity constraint is of the form
  \[ \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \]
  where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\)
- **Example**
  \[ \leftarrow \text{edge}(3,7), \text{color}(3,\text{red}), \text{color}(7,\text{red}). \]
- **Example programs**
  \[
  \{a \leftarrow \neg b, \ b \leftarrow \neg a\} \\
  \{a \leftarrow \neg b, \ b \leftarrow \neg a\} \cup \{ \leftarrow a\} \\
  \{a \leftarrow \neg b, \ b \leftarrow \neg a\} \cup \{ \leftarrow \neg a\}
  \]
Integrity constraint

- **Idea** Eliminate unwanted solution candidates
- **Syntax** An integrity constraint is of the form

  \[ \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \)

- **Example**

  \[ :- \text{edge}(3,7), \text{color}(3,\text{red}), \text{color}(7,\text{red}). \]

- **Example programs**

  \[
  \{ a \leftarrow \sim b, \ b \leftarrow \sim a \} \\
  \{ a \leftarrow \sim b, \ b \leftarrow \sim a \} \cup \{ \leftarrow a \} \\
  \{ a \leftarrow \sim b, \ b \leftarrow \sim a \} \cup \{ \leftarrow \sim a \}
  \]
Embedding in normal rules

An integrity constraint of form

\[ \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]

can be translated into the normal rule

\[ x \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n, \sim x \]

where \( x \) is a new symbol
An integrity constraint of form

\[ \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]

can be translated into the normal rule

\[ x \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n, \sim x \]

where \( x \) is a new symbol
Outline

19 Motivation

20 Core language
- Integrity constraint
- **Choice rule**
- Cardinality rule
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
Choice rule

- **Idea**: Choices over subsets of literals
- **Syntax**: A choice rule is of the form

\[
\{ a_1, \ldots, a_m \} \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o
\]

where \( 0 \leq m \leq n \leq o \) and each \( a_i \) is an atom for \( 1 \leq i \leq o \)

- **Informal meaning**: If the body is satisfied by the stable model at hand, then any subset of \( \{ a_1, \ldots, a_m \} \) can be included in the stable model.

- **Example**

\[
\{ \text{buy(pizza)}; \text{buy(wine)}; \text{buy(corn)} \} \leftarrow \text{at(grocery)}.
\]

- **Example program**

\[
\{ \{ a \} \leftarrow b, \ b \leftarrow \}
\]
Choice rule

- **Idea**: Choices over subsets of literals
- **Syntax**: A choice rule is of the form

\[
\{a_1, \ldots, a_m\} \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o
\]

where \(0 \leq m \leq n \leq o\) and each \(a_i\) is an atom for \(1 \leq i \leq o\)

- **Informal meaning**: If the body is satisfied by the stable model at hand, then any subset of \(\{a_1, \ldots, a_m\}\) can be included in the stable model

- **Example**

\[
\{\text{buy(pizza)}; \text{buy(wine)}; \text{buy(corn)}\} \leftarrow \text{at(grocery)}.
\]

- **Example program**

\[
\{\{a\} \leftarrow b, \ b \leftarrow\}
\]
Choice rule

- **Idea** Choices over subsets of literals
- **Syntax** A *choice rule* is of the form

\[
\{a_1, \ldots, a_m\} \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o
\]

where \(0 \leq m \leq n \leq o\) and each \(a_i\) is an atom for \(1 \leq i \leq o\)

- **Informal meaning** If the body is satisfied by the stable model at hand, then any subset of \(\{a_1, \ldots, a_m\}\) can be included in the stable model

- **Example**

\[
\{ \text{buy(pizza); buy(wine); buy(corn)} \} \leftarrow \text{at(grocery)}.
\]

- **Example program**

\[
\{ \{a\} \leftarrow b, \ b \leftarrow \}
\]
**Choice rule**

- **Idea** Choices over subsets of literals
- **Syntax** A choice rule is of the form

\[
\{ a_1, \ldots, a_m \} \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o
\]

where \(0 \leq m \leq n \leq o\) and each \(a_i\) is an atom for \(1 \leq i \leq o\)

- **Informal meaning** If the body is satisfied by the stable model at hand, then any subset of \(\{a_1, \ldots, a_m\}\) can be included in the stable model.

- **Example**

\[
\{ \text{buy(pizza); buy(wine); buy(corn)} \} \leftarrow \text{at(grocery)}.\]

- **Example program**

\[
\{ \{a\} \leftarrow b, \ b \leftarrow \}
\]
A choice rule of form

\[
\{a_1, \ldots, a_m\} \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o
\]

can be translated into \(2m + 1\) normal rules

\[
b \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o
\]

\[
a_1 \leftarrow b, \neg a'_1, \ldots, a_m \leftarrow b, \neg a'_m
\]

\[
a'_1 \leftarrow \neg a_1, \ldots, a'_m \leftarrow \neg a_m
\]

by introducing new atoms \(b, a'_1, \ldots, a'_m\)
Embedding in normal rules

A choice rule of form

$$\{a_1, \ldots, a_m\} \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o$$

can be translated into $2m + 1$ normal rules

$$b \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o$$

$$a_1 \leftarrow b, \sim a'_1 \ldots a_m \leftarrow b, \sim a'_m$$

$$a'_1 \leftarrow \sim a_1 \ldots a'_m \leftarrow \sim a_m$$

by introducing new atoms $b, a'_1, \ldots, a'_m$
Embedding in normal rules

- A choice rule of form

\[ \{a_1, \ldots, a_m\} \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o \]

can be translated into \(2m + 1\) normal rules

\[
\begin{align*}
    b & \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o \\
    a_1 & \leftarrow b, \sim a'_1, \ldots, a_m \leftarrow b, \sim a'_m \\
    a'_1 & \leftarrow \sim a_1, \ldots, a'_m \leftarrow \sim a_m
\end{align*}
\]

by introducing new atoms \(b, a'_1, \ldots, a'_m\)
Outline

19 Motivation

20 Core language
- Integrity constraint
- Choice rule
- **Cardinality rule**
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
Cardinality rule

- **Idea**: Control (lower) cardinality of subsets of literals
- **Syntax**: A cardinality rule is the form

$$a_0 \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \}$$

where $0 \leq m \leq n$ and each $a_i$ is an atom for $1 \leq i \leq n$; $l$ is a non-negative integer (acting as a lower bound on the body)

- **Informal meaning**: The head atom belongs to the stable model, if at least $l$ positive/negative body literals are in/excluded in the stable model

- **Example**

$$\text{pass(c42)} : - 2 \{ \text{pass(a1)} ; \text{pass(a2)} ; \text{pass(a3)} \}.$$  

- **Example program**

$$\{ \ a \leftarrow 1 \{ b, c \}, \ b \leftarrow \}$$
Cardinality rule

- **Idea** Control (lower) cardinality of subsets of literals
- **Syntax** A cardinality rule is the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) is a non-negative integer (acting as a lower bound on the body)

- **Informal meaning** The head atom belongs to the stable model, if at least \( l \) positive/negative body literals are in/excluded in the stable model

- **Example**

\[ \text{pass(c42)} : \quad 2 \{ \text{pass(a1)} ; \text{pass(a2)} ; \text{pass(a3)} \} . \]

- **Example program**

\[ \{ a \leftarrow 1 \{ b, c \}, \quad b \leftarrow \} \]
Cardinality rule

- Idea  Control (lower) cardinality of subsets of literals
- Syntax  A cardinality rule is the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \);
\( l \) is a non-negative integer (acting as a lower bound on the body)

- Informal meaning  The head atom belongs to the stable model, if at least \( l \) positive/negative body literals are in/excluded in the stable model

- Example

\[ \text{pass(c42)} : - \ 2 \ \{ \ \text{pass(a1)}; \ \text{pass(a2)}; \ \text{pass(a3)} \ \} . \]

- Example program

\[ \{ \ a \leftarrow 1 \ \{ b, c \}, \ b \leftarrow \} \]
Cardinality rule

- **Idea** Control (lower) cardinality of subsets of literals
- **Syntax** A cardinality rule is the form
  \[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]
  where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \);
  \( l \) is a non-negative integer (acting as a lower bound on the body)
- **Informal meaning** The head atom belongs to the stable model, if at least \( l \) positive/negative body literals are in/excluded in the stable model
- **Example**
  \[
  \text{pass(c42)} : - 2 \{ \text{pass(a1)}; \text{pass(a2)}; \text{pass(a3)} \}.
  \]
- **Example program**
  \[
  \{ a \leftarrow 1 \{ b, c \}, \ b \leftarrow \}\]
Embedding in normal rules

- A cardinality rule of form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \]

is translated into the normal rule \( a_0 \leftarrow \text{ctr}(1, l) \) and

- The atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.
Embedding in normal rules

- A cardinality rule of form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

is translated into the normal rule \[ a_0 \leftarrow ctr(1, l) \] and

- The atom \( ctr(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.
Embedding in normal rules

- A cardinality rule of form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \]

is translated into the normal rule \( a_0 \leftarrow \text{ctr}(1, l) \) and

- The atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.
Embedding in normal rules

- A cardinality rule of form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

is translated into the normal rule \( a_0 \leftarrow ctr(1, l) \) and for \( 0 \leq k \leq l \) the rules

\[
\begin{align*}
ctr(i, k+1) & \leftarrow ctr(i + 1, k), a_i \\
ctr(i, k) & \leftarrow ctr(i + 1, k) \quad \text{for } 1 \leq i \leq m \\
ctr(j, k+1) & \leftarrow ctr(j + 1, k), \sim a_j \\
ctr(j, k) & \leftarrow ctr(j + 1, k) \quad \text{for } m + 1 \leq j \leq n \\
ctr(n + 1, 0) & \leftarrow
\end{align*}
\]

- The atom \( ctr(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.
Embedding in normal rules

- A cardinality rule of form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

is translated into the normal rule \( a_0 \leftarrow ctr(1, l) \) and for \( 0 \leq k \leq l \) the rules

\[
\begin{align*}
ctr(i, k+1) & \leftarrow ctr(i + 1, k), a_i \\
ctr(i, k) & \leftarrow ctr(i + 1, k) & \text{for } 1 \leq i \leq m \\
ctr(j, k+1) & \leftarrow ctr(j + 1, k), \sim a_j \\
ctr(j, k) & \leftarrow ctr(j + 1, k) & \text{for } m + 1 \leq j \leq n \\
ctr(n + 1, 0) & \leftarrow
\end{align*}
\]

- The atom \( ctr(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.
Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \}$$

is translated into the normal rule $a_0 \leftarrow \text{ctr}(1, l)$ and for $0 \leq k \leq l$ the rules

$$\text{ctr}(i, k+1) \leftarrow \text{ctr}(i + 1, k), a_i$$
$$\text{ctr}(i, k) \leftarrow \text{ctr}(i + 1, k) \quad \text{for } 1 \leq i \leq m$$

$$\text{ctr}(j, k+1) \leftarrow \text{ctr}(j + 1, k), \sim a_j$$
$$\text{ctr}(j, k) \leftarrow \text{ctr}(j + 1, k) \quad \text{for } m + 1 \leq j \leq n$$

- The atom $\text{ctr}(i,j)$ represents the fact that at least $j$ of the literals having an equal or greater index than $i$, are in a stable model.
Embedding in normal rules

- A cardinality rule of form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

is translated into the normal rule \( a_0 \leftarrow \text{ctr}(1, l) \) and for \( 0 \leq k \leq l \) the rules

\[
\begin{align*}
\text{ctr}(i, k+1) & \leftarrow \text{ctr}(i+1, k), a_i \\
\text{ctr}(i, k) & \leftarrow \text{ctr}(i+1, k) & \text{for } 1 \leq i \leq m \\
\text{ctr}(j, k+1) & \leftarrow \text{ctr}(j+1, k), \sim a_j \\
\text{ctr}(j, k) & \leftarrow \text{ctr}(j+1, k) & \text{for } m+1 \leq j \leq n \\
\text{ctr}(n+1, 0) & \leftarrow \\
\end{align*}
\]

- The atom \( \text{ctr}(i, j) \) represents the fact that at least \( j \) of the literals having an equal or greater index than \( i \), are in a stable model.
An example

Program \( \{ a \leftarrow, \ c \leftarrow 1 \ \{ a, b \} \} \) has the stable model \( \{ a, c \} \)

Translating the cardinality rule yields the rules

\[
\begin{align*}
  a & \leftarrow \\
  c & \leftarrow \ ctr(1, 1) \\
  ctr(1, 2) & \leftarrow ctr(2, 1), a \\
  ctr(1, 1) & \leftarrow ctr(2, 1) \\
  ctr(2, 2) & \leftarrow ctr(3, 1), b \\
  ctr(2, 1) & \leftarrow ctr(3, 1) \\
  ctr(1, 1) & \leftarrow ctr(2, 0), a \\
  ctr(1, 0) & \leftarrow ctr(2, 0) \\
  ctr(2, 1) & \leftarrow ctr(3, 0), b \\
  ctr(2, 0) & \leftarrow ctr(3, 0) \\
  ctr(3, 0) & \leftarrow \\
\end{align*}
\]

having stable model \( \{ a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c \} \)
An example

- Program \( \{ a \leftarrow, \ c \leftarrow 1 \ \{ a, b \} \} \) has the stable model \( \{ a, c \} \)
- Translating the cardinality rule yields the rules

\[
\begin{align*}
  a & \leftarrow \\
  c & \leftarrow ctr(1, 1) \\
  ctr(1, 2) & \leftarrow ctr(2, 1), a \\
  ctr(1, 1) & \leftarrow ctr(2, 1) \\
  ctr(2, 2) & \leftarrow ctr(3, 1), b \\
  ctr(2, 1) & \leftarrow ctr(3, 1) \\
  ctr(1, 1) & \leftarrow ctr(2, 0), a \\
  ctr(1, 0) & \leftarrow ctr(2, 0) \\
  ctr(2, 1) & \leftarrow ctr(3, 0), b \\
  ctr(2, 0) & \leftarrow ctr(3, 0) \\
  ctr(3, 0) & \leftarrow \\
\end{align*}
\]

having stable model \( \{ a, ctr(3, 0), ctr(2, 0), ctr(1, 0), ctr(1, 1), c \} \)
A normal rule

\[ a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \]

can be represented by the cardinality rule

\[ a_0 \leftarrow n \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]
Cardinality rules with upper bounds

- A rule of the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \quad u \]  

(1)

where 0 \leq m \leq n and each \( a_i \) is an atom for 1 \leq i \leq n; \( l \) and \( u \) are non-negative integers

stands for

\[ a_0 \leftarrow b, \neg c \]

\[ b \leftarrow l \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \]

\[ c \leftarrow u+1 \{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \]

where \( b \) and \( c \) are new symbols

- Note The expression in the body of the cardinality rule (1) is referred to as a cardinality constraint with lower and upper bound \( l \) and \( u \).
Cardinality rules with upper bounds

A rule of the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \ u \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) and \( u \) are non-negative integers

stands for

\[ a_0 \leftarrow b, \sim c \]
\[ b \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]
\[ c \leftarrow u+1 \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

where \( b \) and \( c \) are new symbols

Note The expression in the body of the cardinality rule (1) is referred to as a cardinality constraint with lower and upper bound \( l \) and \( u \).
Cardinality rules with upper bounds

- A rule of the form

\[ a_0 \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} u \]  

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) and \( u \) are non-negative integers

stands for

\[ a_0 \leftarrow b, \sim c \]
\[ b \leftarrow l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]
\[ c \leftarrow u + 1 \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} \]

where \( b \) and \( c \) are new symbols

- Note The expression in the body of the cardinality rule (1) is referred to as a cardinality constraint with lower and upper bound \( l \) and \( u \)
Cardinality constraints

- Syntax  A cardinality constraint is of the form

\[
\begin{align*}
\forall \ l \ {\{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \}} \ u
\end{align*}
\]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom for \( 1 \leq i \leq n \); \( l \) and \( u \) are non-negative integers.

- Informal meaning A cardinality constraint is satisfied by a stable model \( X \), if the number of its contained literals satisfied by \( X \) is between \( l \) and \( u \) (inclusive).

- In other words, if

\[
\begin{align*}
l \leq | (\{a_1, \ldots, a_m\} \cap X) \cup (\{a_{m+1}, \ldots, a_n\} \setminus X) | \leq u
\end{align*}
\]
Cardinality constraints

- **Syntax** A *cardinality constraint* is of the form

\[
\{ a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \} \ u
\]

where \(0 \leq m \leq n\) and each \(a_i\) is an atom for \(1 \leq i \leq n\); \(l\) and \(u\) are non-negative integers.

- **Informal meaning** A cardinality constraint is satisfied by a stable model \(X\), if the number of its contained literals satisfied by \(X\) is between \(l\) and \(u\) (inclusive).

- In other words, if

\[
l \leq | (\{a_1, \ldots, a_m\} \cap X) \cup (\{a_{m+1}, \ldots, a_n\} \setminus X) | \leq u
\]
Cardinality constraints

**Syntax**
A cardinality constraint is of the form

\[ l \{ a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n \} u \]

where 0 ≤ m ≤ n and each a_i is an atom for 1 ≤ i ≤ n; l and u are non-negative integers.

**Informal meaning**
A cardinality constraint is satisfied by a stable model X, if the number of its contained literals satisfied by X is between l and u (inclusive).

**In other words,** if

\[ l \leq |(\{a_1, \ldots, a_m\} \cap X) \cup (\{a_{m+1}, \ldots, a_n\} \setminus X)| \leq u \]
Cardinality constraints as heads

A rule of the form

\[ l \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \ u \leftarrow a_{n+1}, \ldots, a_o, \sim a_{o+1}, \ldots, \sim a_p \]

where \(0 \leq m \leq n \leq o \leq p\) and each \(a_i\) is an atom for \(1 \leq i \leq p\);
\(l\) and \(u\) are non-negative integers stands for

\[ b \leftarrow a_{n+1}, \ldots, a_o, \sim a_{o+1}, \ldots, \sim a_p \]
\[ \{a_1, \ldots, a_m\} \leftarrow b \]
\[ c \leftarrow l \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \ u \]
\[ \leftarrow b, \sim c \]

where \(b\) and \(c\) are new symbols

Example

1\{color(v42,red);color(v42,green);color(v42,blue)}1.
Cardinality constraints as heads

- A rule of the form

\[
\begin{align*}
  l \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} & \quad u \leftarrow a_{n+1}, \ldots, a_o, \sim a_{o+1}, \ldots, \sim a_p \\
\end{align*}
\]

where \(0 \leq m \leq n \leq o \leq p\) and each \(a_i\) is an atom for \(1 \leq i \leq p\);
\(l\) and \(u\) are non-negative integers stands for

\[
\begin{align*}
  b & \leftarrow a_{n+1}, \ldots, a_o, \sim a_{o+1}, \ldots, \sim a_p \\
  \{a_1, \ldots, a_m\} & \leftarrow b \\
  c & \leftarrow l \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \quad u \\
  & \leftarrow b, \sim c
\end{align*}
\]

where \(b\) and \(c\) are new symbols

- Example

\[
1\{\text{color(v42,red);color(v42,green);color(v42,blue)}\}1.
\]
Cardinality constraints as heads

- A rule of the form

\[
\begin{align*}
& l \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} & u \leftarrow a_{n+1}, \ldots, a_o, \sim a_{o+1}, \ldots, \sim a_p
\end{align*}
\]

where 0 ≤ m ≤ n ≤ o ≤ p and each \( a_i \) is an atom for 1 ≤ i ≤ p; \( l \) and \( u \) are non-negative integers stands for

\[
\begin{align*}
& b \leftarrow a_{n+1}, \ldots, a_o, \sim a_{o+1}, \ldots, \sim a_p \\
& \{a_1, \ldots, a_m\} \leftarrow b \\
& c \leftarrow l \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} & u \\
& \leftarrow b, \sim c
\end{align*}
\]

where \( b \) and \( c \) are new symbols

- Example

\[
1\{\text{color(v42,red);color(v42,green);color(v42,blue)}\}1.
\]
Full-fledged cardinality rules

A rule of the form

\[ l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \ldots, l_n \ S_n \ u_n \]

where each \( l_i \ S_i \ u_i \) is a cardinality constraint for \( 0 \leq i \leq n \) stands for

\[
\begin{align*}
\text{a} & \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n \\
\text{S}_0^+ & \leftarrow \text{a} \\
& \leftarrow \text{a}, \sim b_0 \\
& \leftarrow \text{a}, c_0 \\
\text{b}_i & \leftarrow l_i \ S_i \\
\text{c}_i & \leftarrow u_i + 1 \ S_i
\end{align*}
\]

where \( a, b_i, c_i \) are new symbols (and \( \cdot^+ \) is defined as on Slide 44)
Full-fledged cardinality rules

- A rule of the form

\[ l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \ldots, l_n \ S_n \ u_n \]

where each \( l_i \ S_i \ u_i \) is a cardinality constraint for \( 0 \leq i \leq n \)

stands for

\[ a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n \]

\[ S_0^+ \leftarrow a \]

\[ \leftarrow a, \sim b_0 \quad \quad b_i \leftarrow l_i \ S_i \]

\[ \leftarrow a, c_0 \quad \quad c_i \leftarrow u_i + 1 \ S_i \]

where \( a, b_i, c_i \) are new symbols (and \( \cdot^+ \) is defined as on Slide 44)
Full-fledged cardinality rules

- A rule of the form

\[ l_0 S_0 u_0 \leftarrow l_1 S_1 u_1, \ldots, l_n S_n u_n \]

where each \( l_i S_i u_i \) is a cardinality constraint for \( 0 \leq i \leq n \)

stands for

\[ a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n \]

\[ S_0^+ \leftarrow a \]

\[ \leftarrow a, \sim b_0 \]

\[ b_i \leftarrow l_i S_i \]

\[ \leftarrow a, c_0 \]

\[ c_i \leftarrow u_i + 1 S_i \]

where \( a, b_i, c_i \) are new symbols (and \( \cdot^+ \) is defined as on Slide 44)
Full-fledged cardinality rules

A rule of the form

\[ l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \ldots, l_n \ S_n \ u_n \]

where each \( l_i \ S_i \ u_i \) is a cardinality constraint for \( 0 \leq i \leq n \) stands for

\[ a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n \]

\[ S_0^+ \leftarrow a \]

\[ \leftarrow a, \sim b_0 \]

\[ b_i \leftarrow l_i \ S_i \]

\[ \leftarrow a, c_0 \]

\[ c_i \leftarrow u_i + 1 \ S_i \]

where \( a, b_i, c_i \) are new symbols (and \( \cdot^+ \) is defined as on Slide 44)
Full-fledged cardinality rules

- A rule of the form

\[ l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \ldots, l_n \ S_n \ u_n \]

where each \( l_i \ S_i \ u_i \) is a cardinality constraint for \( 0 \leq i \leq n \)

stands for

\[ a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n \]

\[ S_0^+ \leftarrow a \]

\[ \leftarrow a, \sim b_0 \]

\[ b_i \leftarrow l_i \ S_i \]

\[ \leftarrow a, c_0 \]

\[ c_i \leftarrow u_i + 1 \ S_i \]

where \( a, b_i, c_i \) are new symbols (and \( \cdot^+ \) is defined as on Slide 44)
Full-fledged cardinality rules

- A rule of the form

\[ l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \ldots, l_n \ S_n \ u_n \]

where each \( l_i \ S_i \ u_i \) is a cardinality constraint for \( 0 \leq i \leq n \)
stands for

\[
\begin{align*}
a & \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_n \\
S_0^+ & \leftarrow a \\
& \leftarrow a, \sim b_0 \\
& \leftarrow a, c_0 \\
& \leftarrow b_i \\
& \leftarrow l_i \ S_i \\
& \leftarrow u_i + 1 \ S_i \\
\end{align*}
\]

where \( a, b_i, c_i \) are new symbols (and \( \cdot^+ \) is defined as on Slide 44)
19 Motivation

20 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
Weight rule

- **Idea** Bound (lower) sum of subsets of literal weights
- **Syntax** A **weighted literal** \( w : k \) associates the weight \( w \) with literal \( k \)
- **Syntax** A **weight rule** is the form

\[
a_0 \leftarrow l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \ldots, w_n : \sim a_n \}
\]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
\( l \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- **Informal meaning** The head atom belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least \( l \)

- **Note** A cardinality rule is a weight rule where \( w_i = 1 \) for \( 0 \leq i \leq n \)
Weight rule

- **Idea** Bound (lower) sum of subsets of literal weights
- **Syntax** A weighted literal $w : k$ associates the weight $w$ with literal $k$
- **Syntax** A weight rule is the form

\[
a_0 \leftarrow l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \ldots, w_n : \sim a_n \}
\]

where $0 \leq m \leq n$ and each $a_i$ is an atom; $l$ and $w_i$ are integers for $1 \leq i \leq n$

- **Informal meaning** The head atom belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least $l$

- **Note** A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$
Weight rule

- **Idea** Bound (lower) sum of subsets of literal weights
- **Syntax** A weighted literal \( w : k \) associates the weight \( w \) with literal \( k \)
- **Syntax** A weight rule is the form

\[
a_0 \leftarrow l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \ldots, w_n : \neg a_n \}
\]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
\( l \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- **Informal meaning** The head atom belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least \( l \)

- **Note** A cardinality rule is a weight rule where \( w_i = 1 \) for \( 0 \leq i \leq n \)
Weight constraints

Syntax A weight constraint is of the form

\[ l \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \ldots, w_n : \sim a_n \} u \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom; \( l, u \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

Meaning A weight constraint is satisfied by a stable model \( X \), if

\[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \notin X} w_i \right) \leq u \]

Note (Cardinality and) weight constraints amount to constraints on (count and) sum aggregate functions

Example

\[ 5 \{ 4: \text{course(db)}; 6: \text{course(ai)}; 3: \text{course(xml)} \} 10 \]
Weight constraints

Syntax A weight constraint is of the form

\[ l \{ \; w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \ldots, w_n : \sim a_n \; \} \; u \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
\( l, u \) and \( w_i \) are integers for \( 1 \leq i \leq n \).

Meaning A weight constraint is satisfied by a stable model \( X \), if

\[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \notin X} w_i \right) \leq u \]

Note (Cardinality and) weight constraints amount to constraints on (count and) sum aggregate functions.

Example

\[ 5 \{ \; 4: \text{course}(db); 6: \text{course}(ai); 3: \text{course}(xml) \; \} \; 10 \]
Weight constraints

- **Syntax** A weight constraint is of the form

\[ l \{ \ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \ldots, w_n : \sim a_n \ \} \ u \]

where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
\( l, u \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- **Meaning** A weight constraint is satisfied by a stable model \( X \), if

\[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \notin X} w_i \right) \leq u \]

- **Note** (Cardinality and) weight constraints amount to constraints on (count and) sum aggregate functions

- **Example**

\[ 5 \{ 4:\text{course(db)}; 6:\text{course(ai)}; 3:\text{course(xml)} \} \ 10 \]
Weight constraints

- **Syntax** A weight constraint is of the form

  \[ l \{ \ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \ldots, w_n : \neg a_n \ \} \ u \]

  where \( 0 \leq m \leq n \) and each \( a_i \) is an atom;
  \( l, u \) and \( w_i \) are integers for \( 1 \leq i \leq n \)

- **Meaning** A weight constraint is satisfied by a stable model \( X \), if

  \[ l \leq \left( \sum_{1 \leq i \leq m, a_i \in X} w_i + \sum_{m < i \leq n, a_i \notin X} w_i \right) \leq u \]

- **Note** (Cardinality and) weight constraints amount to constraints on (count and) sum aggregate functions

- **Example**

  \[ 5 \{ 4:\text{course(db)}; 6:\text{course(ai)}; 3:\text{course(xml)} \ \} \ 10 \]
Outline

19 Motivation

20 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ l : l_1, \ldots, l_n \]

  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l \mid l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘ \( p(1..3). \ q(2). \)’

  \[
  r(X) : p(X), \text{not q(X)} :- r(X) : p(X), \text{not q(X)} ; 1 \{ r(X) : p(X), \text{not q(X)} \}.
  \]

  is instantiated to

  \[
  r(1); r(3) :- r(1), r(3), 1 \{ r(1); r(3) \}.
  \]
Conditional literals

- **Syntax** A conditional literal is of the form
  \[ l : l_1, \ldots, l_n \]
  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l \mid l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

\[
  r(X) : p(X), \text{not}\ q(X) \quad :- \quad r(X) : p(X), \text{not}\ q(X); \quad 1 \quad \{ \quad r(X) : p(X), \text{not}\ q(X) \quad \}.
\]

is instantiated to

\[
  r(1); \quad r(3) \quad :- \quad r(1), \quad r(3), \quad 1 \quad \{ \quad r(1); \quad r(3) \quad \}.
\]
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ l : l_1, \ldots, l_n \]

  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l \mid l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3) \cdot q(2) \)’

  \[ r(X) : p(X), \text{not} q(X) :- r(X) : p(X), \text{not} q(X) ; 1 \{ r(X) : p(X), \text{not} q(X) \} . \]

  is instantiated to

  \[ r(1); r(3) :- r(1), r(3), 1 \{ r(1); r(3) \} . \]
Conditional literals

- **Syntax** A *conditional literal* is of the form

  \[ l : l_1, \ldots, l_n \]

  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l \mid l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

  \[
  r(X) : p(X), \not q(X) :- r(X) : p(X), \not q(X); 1 \{ r(X) : p(X), \not q(X) \}.
  \]

  is instantiated to

  \[
  r(1); r(3) :- r(1), r(3), 1 \{ r(1); r(3) \}.
  \]
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ l : l_1, \ldots, l_n \]

  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l \mid l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). q(2). \)’

  \[ r(X) : p(X), \text{not} q(X) :- r(X) : p(X), \text{not} q(X) ; 1 \{ r(X) : p(X), \text{not} q(X) \}. \]

  is instantiated to

  \[ r(1); r(3) :- r(1), r(3), 1 \{ r(1); r(3) \}. \]
Conditional literals

- **Syntax** A conditional literal is of the form

  \[ l : l_1, \ldots, l_n \]

  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l \mid l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). \quad q(2). \)’

  \[
  r(X) : p(X), \text{not} \ q(X) : \quad r(X) : p(X), \text{not} \ q(X); \quad 1 \{ r(X) : p(X), \text{not} \ q(X) \}.
  \]

  is instantiated to

  \[
  r(1); \quad r(3) : \quad r(1), \quad r(3), \quad 1 \{ \quad r(1); \quad r(3) \}.
  \]
Conditional literals

- **Syntax** A conditional literal is of the form
  \[ l : l_1, \ldots, l_n \]
  where \( l \) and \( l_i \) are literals for \( 0 \leq i \leq n \)

- **Informal meaning** A conditional literal can be regarded as the list of elements in the set \( \{ l | l_1, \ldots, l_n \} \)

- **Note** The expansion of conditional literals is context dependent

- **Example** Given ‘\( p(1..3). q(2). \)’
  \[
  r(X) : p(X), \text{not} \ q(X) \ :- \ r(X) : p(X), \text{not} \ q(X) ; \ 1 \ \{ \ r(X) : p(X), \text{not} \ q(X) \ }.
  \]
  is instantiated to
  \[
  r(1); \ r(3) \ :- \ r(1), r(3), 1 \ \{ \ r(1); \ r(3) \ }.
  \]
Outline

19 Motivation

20 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
Optimization statement

- **Idea** Express (multiple) cost functions subject to minimization and/or maximization

- **Syntax** A minimize statement is of the form

\[
\text{minimize} \ \{ \ w_1 \circ p_1 : l_1, \ldots, l_{m_1} ; \ldots ; w_n \circ p_n : l_{1_n}, \ldots, l_{m_n} \}.
\]

where each \( l_j \) is a literal; and \( w_i \) and \( p_i \) are integers for \( 1 \leq i \leq n \)

- **Priority levels**, \( p_i \), allow for representing lexicographically ordered minimization objectives

- **Meaning** A minimize statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a weighted sum of elements
Optimization statement

- **Idea** Express (multiple) cost functions subject to minimization and/or maximization

- **Syntax** A **minimize** statement is of the form

  \[
  \text{minimize } \{ w_1 \circ p_1 : l_{11}, \ldots, l_{m_1}; \ldots; w_n \circ p_n : l_{1n}, \ldots, l_{mn} \}.
  \]

  where each \( l_{ji} \) is a literal; and \( w_i \) and \( p_i \) are integers for \( 1 \leq i \leq n \)

- **Priority levels**, \( p_i \), allow for representing lexicographically ordered minimization objectives

- **Meaning** A **minimize** statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a weighted sum of elements
Optimization statement

- **Idea**: Express (multiple) cost functions subject to minimization and/or maximization

- **Syntax**: A **minimize statement** is of the form

  \[
  \text{minimize } \{ \ w_1 \odot p_1 : l_{11}, \ldots, l_{m_1} ; \ldots ; w_n \odot p_n : l_{1n}, \ldots, l_{mn} \}. 
  \]

  where each \( l_{ji} \) is a literal; and \( w_i \) and \( p_i \) are integers for \( 1 \leq i \leq n \)

  Priority levels, \( p_i \), allow for representing lexicographically ordered minimization objectives

- **Meaning**: A minimize statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a weighted sum of elements
A maximize statement of the form

\[
\text{maximize } \{ \ w_1 \circ p_1 : l_1, \ldots, w_n \circ p_n : l_n \ \}\]

stands for \textit{minimize} \{ \ -w_1 \circ p_1 : l_1, \ldots, -w_n \circ p_n : l_n \ \}\n
Example: When configuring a computer, we may want to maximize hard disk capacity, while minimizing price

\#maximize \{ 250@1:hd(1); 500@1:hd(2); 750@1:hd(3); 1000@1:hd(4) \}.  
\#minimize \{ 30@2:hd(1); 40@2:hd(2); 60@2:hd(3); 80@2:hd(4) \}.  

The priority levels indicate that (minimizing) price is more important than (maximizing) capacity
A maximize statement of the form

\[
\text{maximize } \{ \, w_1@p_1 : l_1, \ldots, w_n@p_n : l_n \} \]

stands for \text{minimize } \{ \, -w_1@p_1 : l_1, \ldots, -w_n@p_n : l_n \}.

Example When configuring a computer, we may want to maximize hard disk capacity, while minimizing price

\[
\text{#maximize } \{ \, 250@1:hd(1); 500@1:hd(2); 750@1:hd(3); 1000@1:hd(4) \}.
\]

\[
\text{#minimize } \{ \, 30@2:hd(1); 40@2:hd(2); 60@2:hd(3); 80@2:hd(4) \}.
\]

The priority levels indicate that (minimizing) price is more important than (maximizing) capacity.
Optimization statement

- A maximize statement of the form

  \[
  \text{maximize } \{ \ p_1 : l_1, \ldots, p_n : l_n \} \]

  stands for \textit{minimize} \{ \ -p_1 : l_1, \ldots, -p_n : l_n \} \}

- Example When configuring a computer, we may want to maximize hard disk capacity, while minimizing price
  
  \#maximize \{ P@1:hd(I,P,C) \}.
  \#minimize \{ C@2:hd(I,P,C) \}.

  The priority levels indicate that (minimizing) price is more important than (maximizing) capacity
Weak constraints

- Weak constraints are an alternative to minimize statements

- Syntax: \( \sim l_1, \ldots, l_n [w@l] \)
  where each \( l_i \) is a literal for \( 1 \leq i \leq n \); and \( w \) and \( p \) are integers

- Example
  \[
  \sim \text{hd}(1). \ [30@2] \\
  \sim \text{hd}(2). \ [40@2] \\
  \sim \text{hd}(3). \ [60@2] \\
  \sim \text{hd}(4). \ [80@2]
  \]
Weak constraints

- Weak constraints are an alternative to minimize statements

- Syntax: \( \sim l_1, \ldots, l_n \ [w@l] \)
  where each \( l_i \) is a literal for \( 1 \leq i \leq n \); and \( w \) and \( p \) are integers

- Example:
  \[ \sim \text{hd}(1). \ [30@2] \]
  \[ \sim \text{hd}(2). \ [40@2] \]
  \[ \sim \text{hd}(3). \ [60@2] \]
  \[ \sim \text{hd}(4). \ [80@2] \]
Weak constraints

- Weak constraints are an alternative to minimize statements.

- Syntax: \( \sim l_1, \ldots, l_n [w@l] \)
  
  where each \( l_i \) is a literal for \( 1 \leq i \leq n \); and \( w \) and \( p \) are integers.

- Example

  \( \sim \text{hd}(1). \ [30@2] \)
  \( \sim \text{hd}(2). \ [40@2] \)
  \( \sim \text{hd}(3). \ [60@2] \)
  \( \sim \text{hd}(4). \ [80@2] \)
Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax: \( l_1, \ldots, l_n \ [w@l] \)
  where each \( l_i \) is a literal for \( 1 \leq i \leq n \); and \( w \) and \( p \) are integers
- Example:
  \( :\sim \text{hd}(I,P,C). \ [C@2] \)
Outline

19 Motivation

20 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
The \textit{smodels} format consists of

- normal rules
- choice rules
- cardinality rules
- weight rules
- minimization statements

Block-oriented format

\textbf{Note} Minimization statements are not part of the logic program
smodels format

- The *smodels* format consists of
  - normal rules
  - choice rules
  - cardinality rules
  - weight rules
  - minimization statements

- Block-oriented format

- Note Minimization statements are not part of the logic program
**smodels format in detail**

<table>
<thead>
<tr>
<th>Type/Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal rule Slide 158</td>
<td>[ \ldots ]</td>
</tr>
<tr>
<td>Cardinality rule Slide 400</td>
<td>[ \ldots ]</td>
</tr>
<tr>
<td>Choice rule Slide 392</td>
<td>[ \ldots ]</td>
</tr>
<tr>
<td>Weight rule Slide 430</td>
<td>[ \ldots ]</td>
</tr>
<tr>
<td>Minimize statement Slide 447</td>
<td>[ \ldots ]</td>
</tr>
<tr>
<td>Disjunctive rule Slide 502</td>
<td>[ \ldots ]</td>
</tr>
</tbody>
</table>

- The function \( \iota \) represents a mapping of atoms to numbers.
Outline

19 Motivation

20 Core language
- Integrity constraint
- Choice rule
- Cardinality rule
- Weight rule

21 Extended language
- Conditional literal
- Optimization statement

22 Intermediate formats
- smodels format
- aspif format
The *aspif* format consists of
- rule statements
- minimize statements
- projection statements
- output statements
- external statements
- assumption statements
- heuristic statements
- edge statements
- theory terms and atoms
- comments

**Line-oriented format**
Rule statements

Rule statements have the form $1_H B$

- Head $H$ has form $h(m;a_1...a_m)$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each $a_i$ is a positive literal

Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules.

- Body $B$ has one of two forms
  - normal bodies have form $0(n;l_1...l_n)$
    - $n \geq 0$ is the length of the rule body, and
    - each $l_i$ is a literal.
  - weight bodies have form $1(l;n;l_1w_1...l_nw_n)$
    - $l$ is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each $l_i$ and $w_i$ are a literal and a positive integer
Rule statements

Rule statements have the form $1_H B$

- **Head** $H$ has form
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each $a_i$ is a positive literal

Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules.

- **Body** $B$ has one of two forms
  - Normal bodies have form $0_n l_1 \ldots l_n$
    - $n \geq 0$ is the length of the rule body, and
    - each $l_i$ is a literal.
  - Weight bodies have form $1_l n l_1 w_1 \ldots l_n w_n$
    - $l$ is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each $l_i$ and $w_i$ are a literal and a positive integer.
Rule statements have the form $1H \cdot B$

**Head $H$ has form**
- $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
- $m \geq 0$ is the number of head elements, and
- each $a_i$ is a positive literal

Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules.

**Body $B$ has one of two forms**
- normal bodies have form $0n \cdot l_1 \ldots l_n$
  - $n \geq 0$ is the length of the rule body, and
  - each $l_i$ is a literal.
- weight bodies have form $1l \cdot n \cdot l_1 \cdot w_1 \ldots l_n \cdot w_n$
  - $l$ is a positive integer to denote the lower bound,
  - $n \geq 0$ is the number of literals in the rule body, and
  - each $l_i$ and $w_i$ are a literal and a positive integer.
Example

\{a\}.
b :- a.
c :- not a.

asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
### Intermediate formats aspif format

**Example**

```
{a}.
b :- a.
c :- not a.
```

```plaintext
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
```
Language Extensions: Overview

23 Two kinds of negation
24 Disjunctive logic programs
25 Propositional theories
26 Aggregates
27 Gringo language
Two kinds of negation

Outline

23 Two kinds of negation
24 Disjunctive logic programs
25 Propositional theories
26 Aggregates
27 Gringo language
Two kinds of negation

Motivation

Classical versus default negation

Symbol \( \neg \) and \( \sim \)

Idea

\[ \neg a \approx \neg a \in X \]

\[ \sim a \approx a \notin X \]

Example

\[ \text{cross } \leftarrow \neg \text{train} \]

\[ \text{cross } \leftarrow \sim \text{train} \]
Classical versus default negation

- Symbol $\neg$ and $\sim$
- Idea
  - $\neg a \approx \neg a \in X$
  - $\sim a \approx a \notin X$
- Example
  - $cross \leftarrow \neg train$
  - $cross \leftarrow \sim train$
Two kinds of negation

Motivation

Classical versus default negation

Symbol $\neg$ and $\sim$

Idea

$\neg a \approx \neg a \in X$

$\sim a \approx a \notin X$

Example

$cross \leftarrow \neg train$

$cross \leftarrow \sim train$
Two kinds of negation

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only

- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$

- Given a program $P$ over $\mathcal{A}$, classical negation is encoded by adding
  
  $$P^\neg = \{ a \leftarrow b, \neg b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

- A set $X$ of atoms is a stable model of a program $P$ over $\mathcal{A} \cup \overline{\mathcal{A}}$, if $X$ is a stable model of $P \cup P^\neg$
Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program $P$ over $\mathcal{A}$, classical negation is encoded by adding
  \[ P^\neg = \{ a \leftarrow b, \neg b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A} \} \]

- A set $X$ of atoms is a stable model of a program $P$ over $\mathcal{A} \cup \overline{\mathcal{A}}$, if $X$ is a stable model of $P \cup P^\neg$
Two kinds of negation

Classical negation

We consider logic programs in negation normal form
- That is, classical negation is applied to atoms only

Given an alphabet \( \mathcal{A} \) of atoms, let \( \overline{\mathcal{A}} = \{ \neg a \mid a \in \mathcal{A} \} \) such that
\( \mathcal{A} \cap \overline{\mathcal{A}} = \emptyset \)

Given a program \( P \) over \( \mathcal{A} \), classical negation is encoded by adding

\[
P^- = \{ a \leftarrow b, \neg b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A} \}\]

A set \( X \) of atoms is a stable model of a program \( P \) over \( \mathcal{A} \cup \overline{\mathcal{A}} \), if \( X \) is a stable model of \( P \cup P^- \)
Two kinds of negation

Classical negation

- We consider logic programs in negation normal form.
  - That is, classical negation is applied to atoms only.

- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$.

- Given a program $P$ over $\mathcal{A}$, classical negation is encoded by adding
  $$P^\neg = \{a \leftarrow b, \neg b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

- A set $X$ of atoms is a stable model of a program $P$ over $\mathcal{A} \cup \overline{\mathcal{A}}$, if $X$ is a stable model of $P \cup P^\neg$. 

Two kinds of negation

An example

- The program

\[ P = \{ a \leftarrow \neg b, \ b \leftarrow \neg a \} \cup \{ c \leftarrow b, \ \neg c \leftarrow b \} \]

induces

\[ P^- = \begin{cases} 
  a \leftarrow a, \neg a \\
  \neg a \leftarrow a, \neg a \\
  b \leftarrow a, \neg a \\
  \neg b \leftarrow a, \neg a \\
  c \leftarrow a, \neg a \\
  \neg c \leftarrow a, \neg a 
\end{cases} \cup \begin{cases} 
  a \leftarrow b, \neg b \\
  \neg a \leftarrow b, \neg b \\
  b \leftarrow b, \neg b \\
  \neg b \leftarrow b, \neg b \\
  c \leftarrow b, \neg b \\
  \neg c \leftarrow b, \neg b 
\end{cases} \cup \begin{cases} 
  a \leftarrow c, \neg c \\
  \neg a \leftarrow c, \neg c \\
  b \leftarrow c, \neg c \\
  \neg b \leftarrow c, \neg c \\
  c \leftarrow c, \neg c \\
  \neg c \leftarrow c, \neg c 
\end{cases} \]

- The stable models of \( P \) are given by the ones of \( P \cup P^- \), viz \( \{ a \} \)
Two kinds of negation

An example

- The program

\[ P = \{a \leftarrow \neg b, \ b \leftarrow \neg a\} \cup \{c \leftarrow b, \ \neg c \leftarrow b\} \]

induces

\[ P^\neg = \{\begin{array}{cccc}
a & \leftarrow & a, \neg a & \quad a & \leftarrow & b, \neg b & \quad a & \leftarrow & c, \neg c \\
\neg a & \leftarrow & a, \neg a & \quad \neg a & \leftarrow & b, \neg b & \quad \neg a & \leftarrow & c, \neg c \\
b & \leftarrow & a, \neg a & \quad b & \leftarrow & b, \neg b & \quad b & \leftarrow & c, \neg c \\
\neg b & \leftarrow & a, \neg a & \quad \neg b & \leftarrow & b, \neg b & \quad \neg b & \leftarrow & c, \neg c \\
c & \leftarrow & a, \neg a & \quad c & \leftarrow & b, \neg b & \quad c & \leftarrow & c, \neg c \\
\neg c & \leftarrow & a, \neg a & \quad \neg c & \leftarrow & b, \neg b & \quad \neg c & \leftarrow & c, \neg c
\end{array}\} \]

- The stable models of \( P \) are given by the ones of \( P \cup P^\neg \), viz \( \{a\} \)
Two kinds of negation

An example

The program

\[ P = \{ a \leftarrow \neg b, \ b \leftarrow \neg a \} \cup \{ c \leftarrow b, \ \neg c \leftarrow b \} \]

induces

\[ P^- = \begin{cases} a \leftarrow a, \neg a & a \leftarrow b, \neg b & a \leftarrow c, \neg c \\
\neg a \leftarrow a, \neg a & \neg a \leftarrow b, \neg b & \neg a \leftarrow c, \neg c \\
b \leftarrow a, \neg a & b \leftarrow b, \neg b & b \leftarrow c, \neg c \\
\neg b \leftarrow a, \neg a & \neg b \leftarrow b, \neg b & \neg b \leftarrow c, \neg c \\
c \leftarrow a, \neg a & c \leftarrow b, \neg b & c \leftarrow c, \neg c \\
\neg c \leftarrow a, \neg a & \neg c \leftarrow b, \neg b & \neg c \leftarrow c, \neg c \end{cases} \]

The stable models of \( P \) are given by the ones of \( P \cup P^- \), viz \( \{ a \} \)
Two kinds of negation

Properties

- The only inconsistent stable "model" is \( X = A \cup \overline{A} \)

  Strictly speaking, an inconsistent set like \( A \cup \overline{A} \) is not a model.

- For a logic program \( P \) over \( A \cup \overline{A} \), exactly one of the following two cases applies:
  1. All stable models of \( P \) are consistent or
  2. \( X = A \cup \overline{A} \) is the only stable model of \( P \)
The only inconsistent stable “model” is \( X = A \cup \overline{A} \).

Strictly speaking, an inconsistent set like \( A \cup \overline{A} \) is not a model.

For a logic program \( P \) over \( A \cup \overline{A} \), exactly one of the following two cases applies:

1. All stable models of \( P \) are consistent or
2. \( X = A \cup \overline{A} \) is the only stable model of \( P \).
Two kinds of negation

Properties

- The only inconsistent stable “model” is \( X = \mathcal{A} \cup \overline{\mathcal{A}} \)
  
  Strictly speaking, an inconsistency set like \( \mathcal{A} \cup \overline{\mathcal{A}} \) is not a model

- For a logic program \( P \) over \( \mathcal{A} \cup \overline{\mathcal{A}} \), exactly one of the following two cases applies:
  1. All stable models of \( P \) are consistent or
  2. \( X = \mathcal{A} \cup \overline{\mathcal{A}} \) is the only stable model of \( P \)
Two kinds of negation

Train spotting

- \( P_1 = \{ \text{cross} \leftarrow \sim \text{train} \} \)
  - stable model: \( \{ \text{cross} \} \)

- \( P_2 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  - stable model: \( \emptyset \)

- \( P_3 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \} \)
  - stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_4 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \} \)
  - stable model: \( \{ \text{cross, \neg cross, train, \neg train} \} \)

- \( P_5 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train} \} \)
  - stable model: \( \{ \text{cross, \neg train} \} \)

- \( P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow \} \)
  - no stable model
Two kinds of negation

Train spotting

- \( P_1 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  - stable model: \{cross\}

- \( P_2 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  - stable model: \(\emptyset\)

- \( P_3 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \} \)
  - stable model: \{cross, \neg \text{train}\}

- \( P_4 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \} \)
  - stable model: \{cross, \neg \text{cross}, \text{train}, \neg \text{train}\}

- \( P_5 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train} \} \)
  - stable model: \{cross, \neg \text{train}\}

- \( P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train}, \neg \text{cross} \leftarrow \} \)
  - no stable model
Two kinds of negation

Train spotting

- \( P_1 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  - stable model: \( \{ \text{cross} \} \)

- \( P_2 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  - stable model: \( \emptyset \)

- \( P_3 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \} \)
  - stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_4 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \} \)
  - stable model: \( \{ \text{cross}, \neg \text{cross}, \text{train}, \neg \text{train} \} \)

- \( P_5 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train} \} \)
  - stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train}, \neg \text{cross} \leftarrow \} \)
  - no stable model
Two kinds of negation

Train spotting

- \( P_1 = \{ \text{cross} \leftarrow \sim \text{train} \} \)
  
  stable model: \( \{ \text{cross} \} \)

- \( P_2 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  
  stable model: \( \emptyset \)

- \( P_3 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \} \)
  
  stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_4 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \} \)
  
  stable model: \( \{ \text{cross}, \neg \text{cross}, \text{train}, \neg \text{train} \} \)

- \( P_5 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train} \} \)
  
  stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow \} \)
  
  no stable model
Two kinds of negation

Train spotting

- $P_1 = \{\text{cross} \leftarrow \sim \text{train}\}$
  - stable model: $\{\text{cross}\}$

- $P_2 = \{\text{cross} \leftarrow \neg \text{train}\}$
  - stable model: $\emptyset$

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
  - stable model: $\{\text{cross}, \neg \text{cross}, \text{train}, \neg \text{train}\}$

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}\}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow\}$
  - no stable model
Two kinds of negation

Train spotting

- \( P_1 = \{ \text{cross} \leftarrow \sim \text{train} \} \)
  
  stable model: \{ cross \}

- \( P_2 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  
  stable model: \( \emptyset \)

- \( P_3 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \} \)
  
  stable model: \{ cross, \neg train \}

- \( P_4 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \} \)
  
  stable model: \{ cross, \neg cross, train, \neg train \}

- \( P_5 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train} \} \)
  
  stable model: \{ cross, \neg train \}

- \( P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow \} \)
  
  no stable model
Two kinds of negation

Train spotting

- $P_1 = \{\text{cross} \leftarrow \sim \text{train}\}$
  stable model: $\{\text{cross}\}$

- $P_2 = \{\text{cross} \leftarrow \neg \text{train}\}$
  stable model: $\emptyset$

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$
  stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
  stable model: $\{\text{cross}, \neg \text{cross}, \text{train}, \neg \text{train}\}$

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}\}$
  stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow\}$
  no stable model
Two kinds of negation

Train spotting

- $P_1 = \{cross \leftarrow \sim train\}$
  stable model: $\{cross\}$

- $P_2 = \{cross \leftarrow \neg train\}$
  stable model: $\emptyset$

- $P_3 = \{cross \leftarrow \neg train, \neg train \leftarrow\}$
  stable model: $\{cross, \neg train\}$

- $P_4 = \{cross \leftarrow \neg train, \neg train \leftarrow, \neg cross \leftarrow\}$
  stable model: $\{cross, \neg cross, train, \neg train\}$

- $P_5 = \{cross \leftarrow \neg train, \neg train \leftarrow \sim train\}$
  stable model: $\{cross, \neg train\}$

- $P_6 = \{cross \leftarrow \neg train, \neg train \leftarrow \sim train, \neg cross \leftarrow\}$
  no stable model
Train spotting

- $P_1 = \{\text{cross} \leftarrow \neg \text{train}\}$
  stable model: $\{\text{cross}\}$

- $P_2 = \{\text{cross} \leftarrow \neg \text{train}\}$
  stable model: $\emptyset$

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \}$
  stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \}$
  stable model: $\{\text{cross}, \neg \text{cross}, \text{train}, \neg \text{train}\}$

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train}\}$
  stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \neg \text{train}, \neg \text{cross} \leftarrow \}$
  no stable model
Two kinds of negation

Train spotting

- \( P_1 = \{ \text{cross} \leftarrow \sim \text{train} \} \)
  stable model: \( \{ \text{cross} \} \)

- \( P_2 = \{ \text{cross} \leftarrow \neg \text{train} \} \)
  stable model: \( \emptyset \)

- \( P_3 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \} \)
  stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_4 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow \} \)
  stable model: \( \{ \text{cross}, \neg \text{cross}, \text{train}, \neg \text{train} \} \)

- \( P_5 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train} \} \)
  stable model: \( \{ \text{cross}, \neg \text{train} \} \)

- \( P_6 = \{ \text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow \} \)
  no stable model
Two kinds of negation

Train spotting

- $P_1 = \{\text{cross} \leftarrow \sim \text{train}\}$
  - stable model: $\{\text{cross}\}$

- $P_2 = \{\text{cross} \leftarrow \neg \text{train}\}$
  - stable model: $\emptyset$

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
  - stable model: $\{\text{cross}, \neg \text{cross}, \text{train}, \neg \text{train}\}$

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}\}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow\}$
  - no stable model
Two kinds of negation

Train spotting

- $P_1 = \{\text{cross} \leftarrow \sim \text{train}\}$
  - stable model: $\{\text{cross}\}$

- $P_2 = \{\text{cross} \leftarrow \neg \text{train}\}$
  - stable model: $\emptyset$

- $P_3 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow\}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_4 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow, \neg \text{cross} \leftarrow\}$
  - stable model: $\{\text{cross}, \neg \text{cross}, \text{train}, \neg \text{train}\}$

- $P_5 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}\}$
  - stable model: $\{\text{cross}, \neg \text{train}\}$

- $P_6 = \{\text{cross} \leftarrow \neg \text{train}, \neg \text{train} \leftarrow \sim \text{train}, \neg \text{cross} \leftarrow\}$
  - no stable model
Two kinds of negation

Train spotting

- $P_1 = \{cross \leftarrow \sim train\}$
  - stable model: $\{cross\}$

- $P_2 = \{cross \leftarrow \neg train\}$
  - stable model: $\emptyset$

- $P_3 = \{cross \leftarrow \neg train, \neg train \leftarrow\}$
  - stable model: $\{cross, \neg train\}$

- $P_4 = \{cross \leftarrow \neg train, \neg train \leftarrow, \neg cross \leftarrow\}$
  - stable model: $\{cross, \neg cross, train, \neg train\}$

- $P_5 = \{cross \leftarrow \neg train, \neg train \leftarrow \sim train\}$
  - stable model: $\{cross, \neg train\}$

- $P_6 = \{cross \leftarrow \neg train, \neg train \leftarrow \sim train, \neg cross \leftarrow\}$
  - no stable model
Default negation in rule heads

- We consider logic programs with default negation in rule heads.
- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{\tilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$.
- Given a program $P$ over $\mathcal{A}$, consider the program $\tilde{\tilde{P}} = \{r \in P \mid h(r) \neq \sim a\} \cup \{\leftarrow B(r) \cup \{\sim \tilde{a}\} \mid r \in P \text{ and } h(r) = \sim a\} \cup \{\tilde{a} \leftarrow \sim a \mid r \in P \text{ and } h(r) = \sim a\}$

- A set $X$ of atoms is a stable model of a program $P$ (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model $Y$ of $\tilde{\tilde{P}}$ over $\mathcal{A} \cup \tilde{\mathcal{A}}$. 
Two kinds of negation

Default negation in rule heads

- We consider logic programs with default negation in rule heads.
- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{\tilde{a} | a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$.
- Given a program $P$ over $\mathcal{A}$, consider the program $\tilde{P} = \{r \in P | h(r) \neq \sim a\}$
  $\cup \{\leftarrow B(r) \cup \{\sim \tilde{a}\} | r \in P \text{ and } h(r) = \sim a\}$
  $\cup \{\tilde{a} \leftarrow \sim a | r \in P \text{ and } h(r) = \sim a\}$

- A set $X$ of atoms is a stable model of a program $P$ (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model $Y$ of $\tilde{P}$ over $\mathcal{A} \cup \tilde{\mathcal{A}}$. 
Two kinds of negation

Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{\tilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$
- Given a program $P$ over $\mathcal{A}$, consider the program
  \[
  \tilde{P} = \{r \in P \mid h(r) \neq \sim a\} \\
  \cup \{\leftarrow B(r) \cup \{\sim \tilde{a}\} \mid r \in P \text{ and } h(r) = \sim a\} \\
  \cup \{\tilde{a} \leftarrow \sim a \mid r \in P \text{ and } h(r) = \sim a\}
  \]

- A set $X$ of atoms is a stable model of a program $P$ (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model $Y$ of $\tilde{P}$ over $\mathcal{A} \cup \tilde{\mathcal{A}}$
Two kinds of negation

Default negation in rule heads

- We consider logic programs with default negation in rule heads.

- Given an alphabet $\mathcal{A}$ of atoms, let $\tilde{\mathcal{A}} = \{\tilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \tilde{\mathcal{A}} = \emptyset$.

- Given a program $P$ over $\mathcal{A}$, consider the program
  $$\tilde{P} = \{r \in P \mid h(r) \neq \sim a\} \cup \{\leftarrow B(r) \cup \{\sim \tilde{a}\} \mid r \in P \text{ and } h(r) = \sim a\} \cup \{\tilde{a} \leftarrow \sim a \mid r \in P \text{ and } h(r) = \sim a\}$$

- A set $X$ of atoms is a stable model of a program $P$ (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model $Y$ of $\tilde{P}$ over $\mathcal{A} \cup \tilde{\mathcal{A}}$.
Outline

23 Two kinds of negation
24 Disjunctive logic programs
25 Propositional theories
26 Aggregates
27 Gringo language
A disjunctive rule, $r$, is of the form

$$a_1 ; \ldots ; a_m \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o$$

where $0 \leq m \leq n \leq o$ and each $a_i$ is an atom for $0 \leq i \leq o$

A disjunctive logic program is a finite set of disjunctive rules

Notation

- $H(r) = \{a_1, \ldots, a_m\}$
- $B(r) = \{a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o\}$
- $B(r)^+ = \{a_{m+1}, \ldots, a_n\}$
- $B(r)^- = \{a_{n+1}, \ldots, a_o\}$
- $A(P) = \bigcup_{r \in P} (H(r) \cup B(r)^+ \cup B(r)^-)$
- $B(P) = \{B(r) \mid r \in P\}$

A program is called positive if $B(r)^- = \emptyset$ for all its rules
A disjunctive rule, \( r \), is of the form

\[
a_1 ; \ldots ; a_m \leftarrow a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o
\]

where \( 0 \leq m \leq n \leq o \) and each \( a_i \) is an atom for \( 0 \leq i \leq o \)

A disjunctive logic program is a finite set of disjunctive rules

Notation

\[
H(r) = \{a_1, \ldots, a_m\}
\]
\[
B(r) = \{a_{m+1}, \ldots, a_n, \sim a_{n+1}, \ldots, \sim a_o\}
\]
\[
B(r)^+ = \{a_{m+1}, \ldots, a_n\}
\]
\[
B(r)^- = \{a_{n+1}, \ldots, a_o\}
\]
\[
A(P) = \bigcup_{r \in P} (H(r) \cup B(r)^+ \cup B(r)^-)
\]
\[
B(P) = \{B(r) \mid r \in P\}
\]

A program is called positive if \( B(r)^- = \emptyset \) for all its rules
Disjunctive logic programs

- A disjunctive rule, \( r \), is of the form

\[
a_1 ; \ldots ; a_m \leftarrow a_{m+1}, \ldots , a_n, \sim a_{n+1}, \ldots , \sim a_o
\]

where \( 0 \leq m \leq n \leq o \) and each \( a_i \) is an atom for \( 0 \leq i \leq o \)

- A disjunctive logic program is a finite set of disjunctive rules

- Notation

\[
H(r) = \{a_1, \ldots , a_m\}
\]
\[
B(r) = \{a_{m+1}, \ldots , a_n, \sim a_{n+1}, \ldots , \sim a_o\}
\]
\[
B(r)^+ = \{a_{m+1}, \ldots , a_n\}
\]
\[
B(r)^- = \{a_{n+1}, \ldots , a_o\}
\]
\[
A(P) = \bigcup_{r \in P} (H(r) \cup B(r)^+ \cup B(r)^-)
\]
\[
B(P) = \{B(r) \mid r \in P\}
\]

- A program is called positive if \( B(r)^- = \emptyset \) for all its rules
Stable models

- **Positive programs**
  - A set $X$ of atoms is **closed under** a positive program $P$ iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program $P$ is denoted by $\text{min}_{\subseteq}(P)$
  - $\text{min}_{\subseteq}(P)$ corresponds to the $\subseteq$-minimal models of $P$ (ditto)

- **Disjunctive programs**
  - The reduct, $P^X$, of a disjunctive program $P$ relative to a set $X$ of atoms is defined by
    $$P^X = \{H(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset\}$$
  - A set $X$ of atoms is a stable model of a disjunctive program $P$, if $X \in \text{min}_{\subseteq}(P^X)$
Stable models

Positive programs

- A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
  
  - $X$ corresponds to a model of $P$ (seen as a formula)

- The set of all $\subseteq$-minimal sets of atoms being closed under a positive program $P$ is denoted by $\text{min}_\subseteq(P)$
  
  - $\text{min}_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of $P$ (ditto)

Disjunctive programs

- The reduct, $P^X$, of a disjunctive program $P$ relative to a set $X$ of atoms is defined by

$$P^X = \{H(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset\}$$

- A set $X$ of atoms is a stable model of a disjunctive program $P$, if $X \in \text{min}_\subseteq(P^X)$
Stable models

Positive programs

- A set $X$ of atoms is closed under a positive program $P$ iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
  - $X$ corresponds to a model of $P$ (seen as a formula)
- The set of all $\subseteq$-minimal sets of atoms being closed under a positive program $P$ is denoted by $\text{min}_{\subseteq}(P)$
  - $\text{min}_{\subseteq}(P)$ corresponds to the $\subseteq$-minimal models of $P$ (ditto)

Disjunctive programs

- The reduct, $P^X$, of a disjunctive program $P$ relative to a set $X$ of atoms is defined by
  \[
P^X = \{ H(r) \leftarrow B(r)^+ \mid r \in P \text{ and } B(r)^- \cap X = \emptyset \}\]
- A set $X$ of atoms is a stable model of a disjunctive program $P$, if $X \in \text{min}_{\subseteq}(P^X)$
A “positive” example

\[ P = \{ a \leftarrow b ; c \leftarrow a \} \]

- The sets \{a, b\}, \{a, c\}, and \{a, b, c\} are closed under \( P \)
- We have \( \text{min}_\subseteq(P) = \{\{a, b\}, \{a, c\}\} \)
A “positive” example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \neg b \\
b ; c \leftarrow a
\end{array} \right\} \]

- The sets \{a, b\}, \{a, c\}, and \{a, b, c\} are closed under \( P \)
- We have \( \text{min}_\subseteq(P) = \{\{a, b\}, \{a, c\}\} \)
Disjunctive logic programs

A “positive” example

\[ P = \left\{ \begin{array}{c}
    a \\
    b ; c \leftarrow a
\end{array} \right\} \]

- The sets \{a, b\}, \{a, c\}, and \{a, b, c\} are closed under \(P\)
- We have \(\text{min}_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}\)
Disjunctive logic programs

Graph coloring (reloaded)

node(1..6).

edge(1,(2;3;4)). edge(2,(4;5;6)). edge(3,(1;4;5)).
edge(4,(1;2)). edge(5,(3;4;6)). edge(6,(2;3;5)).

assign(X,r) ; assign(X,b) ; assign(X,g) :- node(X).

:- edge(X,Y), assign(X,C), assign(Y,C).
Graph coloring (reloaded)

node(1..6).

edge(1,(2;3;4)). edge(2,(4;5;6)). edge(3,(1;4;5)).
edge(4,(1;2)). edge(5,(3;4;6)). edge(6,(2;3;5)).

color(r). color(b). color(g).

assign(X,C) : color(C) :- node(X).

:- edge(X,Y), assign(X,C), assign(Y,C).
More Examples

- \( P_1 = \{ a ; b ; c \leftarrow \} \)
  - stable models \( \{ a \} \), \( \{ b \} \), and \( \{ c \} \)

- \( P_2 = \{ a ; b ; c \leftarrow , \leftarrow a \} \)
  - stable models \( \{ b \} \) and \( \{ c \} \)

- \( P_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \} \)
  - stable model \( \{ b, c \} \)

- \( P_4 = \{ a ; b \leftarrow c , b \leftarrow \sim a , \sim c , a ; c \leftarrow \sim b \} \)
  - stable models \( \{ a \} \) and \( \{ b \} \)
More Examples

- \( P_1 = \{ a \land b \land c \leftarrow \} \)
  - stable models \( \{a\} \), \( \{b\} \), and \( \{c\} \)

- \( P_2 = \{ a \land b \land c \leftarrow \neg a \} \)
  - stable models \( \{b\} \) and \( \{c\} \)

- \( P_3 = \{ a \land b \land c \leftarrow \neg a \land b \leftarrow c \land c \leftarrow b \} \)
  - stable model \( \{b, c\} \)

- \( P_4 = \{ a \land b \leftarrow c \land b \leftarrow \neg a \land \neg c \land a \land c \leftarrow \neg b \} \)
  - stable models \( \{a\} \) and \( \{b\} \)
More Examples

- $P_1 = \{a; b; c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$

- $P_2 = \{a; b; c \leftarrow, \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$

- $P_3 = \{a; b; c \leftarrow, \leftarrow a, b \leftarrow c, c \leftarrow b\}$
  - stable model $\{b, c\}$

- $P_4 = \{a; b \leftarrow c, b \leftarrow \neg a, \neg c, a; c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$
More Examples

- $P_1 = \{a ; b ; c ←\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$

- $P_2 = \{a ; b ; c ← , ← a\}$
  - stable models $\{b\}$ and $\{c\}$

- $P_3 = \{a ; b ; c ← , ← a , b ← c , c ← b\}$
  - stable model $\{b, c\}$

- $P_4 = \{a ; b ← c , b ← \sim a, \sim c , a ; c ← \sim b\}$
  - stable models $\{a\}$ and $\{b\}$
More Examples

- $P_1 = \{ a ; b ; c \leftarrow \}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$

- $P_2 = \{ a ; b ; c \leftarrow , \leftarrow a \}$
  - stable models $\{b\}$ and $\{c\}$

- $P_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \}$
  - stable model $\{b, c\}$

- $P_4 = \{ a ; b \leftarrow c , b \leftarrow \neg a , \neg c , a ; c \leftarrow \neg b \}$
  - stable models $\{a\}$ and $\{b\}$
More Examples

- $P_1 = \{a ; b ; c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$

- $P_2 = \{a ; b ; c \leftarrow , \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$

- $P_3 = \{a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b\}$
  - stable model $\{b, c\}$

- $P_4 = \{a ; b \leftarrow c , b \leftarrow \neg a , \neg c , a ; c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$
More Examples

- \( P_1 = \{ a ; b ; c \leftarrow \} \)
  - stable models \( \{a\}, \{b\}, \) and \( \{c\} \)

- \( P_2 = \{ a ; b ; c \leftarrow , \leftarrow a \} \)
  - stable models \( \{b\} \) and \( \{c\} \)

- \( P_3 = \{ a ; b ; c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \} \)
  - stable model \( \{b, c\} \)

- \( P_4 = \{ a ; b \leftarrow c , b \leftarrow \sim a , \sim c , a ; c \leftarrow \sim b \} \)
  - stable models \( \{a\} \) and \( \{b\} \)
More Examples

- $P_1 = \{ a \leftarrow b \leftarrow c \}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$

- $P_2 = \{ a \leftarrow b \leftarrow c \leftarrow a \}$
  - stable models $\{b\}$ and $\{c\}$

- $P_3 = \{ a \leftarrow b \leftarrow c \leftarrow a \leftarrow c \leftarrow b \}$
  - stable model $\{b, c\}$

- $P_4 = \{ a \leftarrow b \leftarrow c \leftarrow b \leftarrow \neg a \leftarrow \neg c \leftarrow a \leftarrow c \leftarrow \neg b \}$
  - stable models $\{a\}$ and $\{b\}$
More Examples

- $P_1 = \{ a \mid b \mid c \leftarrow \}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{ a \mid b \mid c \leftarrow , \leftarrow a \}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{ a \mid b \mid c \leftarrow , \leftarrow a , b \leftarrow c , c \leftarrow b \}$
  - stable model $\{b, c\}$
- $P_4 = \{ a \mid b \leftarrow c , b \leftarrow \sim a , \sim c , a \mid c \leftarrow \sim b \}$
  - stable models $\{a\}$ and $\{b\}$
Some properties

- A disjunctive logic program may have zero, one, or multiple stable models.
- If $X$ is a stable model of a disjunctive logic program $P$, then $X$ is a model of $P$ (seen as a formula).
- If $X$ and $Y$ are stable models of a disjunctive logic program $P$, then $X \not\subset Y$.
- If $a \in X$ for some stable model $X$ of a disjunctive logic program $P$, then there is a rule $r \in P$ such that $B(r)^+ \subseteq X$, $B(r)^- \cap X = \emptyset$, and $H(r) \cap X = \{a\}$. 
Some properties

- A disjunctive logic program may have zero, one, or multiple stable models.
- If $X$ is a stable model of a disjunctive logic program $P$, then $X$ is a model of $P$ (seen as a formula).
- If $X$ and $Y$ are stable models of a disjunctive logic program $P$, then $X \not\subset Y$.
- If $a \in X$ for some stable model $X$ of a disjunctive logic program $P$, then there is a rule $r \in P$ such that $B(r)^+ \subseteq X$, $B(r)^- \cap X = \emptyset$, and $H(r) \cap X = \{a\}$. 
An example with variables

\[ P = \{ a(1, 2) \leftarrow b(X) ; c(Y) \leftarrow a(X, Y), \neg c(Y) \} \]

\[ \text{ground}(P) = \begin{cases} 
  a(1, 2) \leftarrow \\
  b(1) ; c(1) \leftarrow a(1, 1), \neg c(1) \\
  b(1) ; c(2) \leftarrow a(1, 2), \neg c(2) \\
  b(2) ; c(1) \leftarrow a(2, 1), \neg c(1) \\
  b(2) ; c(2) \leftarrow a(2, 2), \neg c(2) 
\end{cases} \]

For every stable model \( X \) of \( P \), we have

- \( a(1, 2) \in X \) and
- \( \{ a(1, 1), a(2, 1), a(2, 2) \} \cap X = \emptyset \)
Disjunctive logic programs

An example with variables

\[ P = \begin{cases} a(1, 2) \leftarrow \\ b(X); c(Y) \leftarrow a(X, Y), \sim c(Y) \end{cases} \]

\[ \text{ground}(P) = \begin{cases} a(1, 2) \leftarrow \\ b(1); c(1) \leftarrow a(1, 1), \sim c(1) \\ b(1); c(2) \leftarrow a(1, 2), \sim c(2) \\ b(2); c(1) \leftarrow a(2, 1), \sim c(1) \\ b(2); c(2) \leftarrow a(2, 2), \sim c(2) \end{cases} \]

For every stable model \( X \) of \( P \), we have

- \( a(1, 2) \in X \) and
- \( \{a(1, 1), a(2, 1), a(2, 2)\} \cap X = \emptyset \)
An example with variables

\[ P = \{ \begin{align*} & a(1,2) \leftarrow \\ & b(X) ; c(Y) \leftarrow a(X, Y), \neg c(Y) \end{align*} \} \]

\[ \text{ground}(P) = \{ \begin{align*} & a(1,2) \leftarrow \\ & b(1) ; c(1) \leftarrow a(1,1), \neg c(1) \\ & b(1) ; c(2) \leftarrow a(1,2), \neg c(2) \\ & b(2) ; c(1) \leftarrow a(2,1), \neg c(1) \\ & b(2) ; c(2) \leftarrow a(2,2), \neg c(2) \end{align*} \} \]

For every stable model \( X \) of \( P \), we have
- \( a(1,2) \in X \) and
- \( \{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset \)
An example with variables

Consider $X = \{a(1, 2), b(1)\}$

We get $\text{min}_\subseteq(\text{ground}(P)^X) = \{ \{a(1, 2), b(1)\}, \{a(1, 2), c(2)\} \}$

$X$ is a stable model of $P$ because $X \in \text{min}_\subseteq(\text{ground}(P)^X)$

$\text{ground}(P)^X = \begin{cases} 
    a(1, 2) \leftarrow \\
    b(1) \land c(1) \leftarrow a(1, 1), \sim c(1) \\
    b(1) \land c(2) \leftarrow a(1, 2), \sim c(2) \\
    b(2) \land c(1) \leftarrow a(2, 1), \sim c(1) \\
    b(2) \land c(2) \leftarrow a(2, 2), \sim c(2) 
\end{cases}$
An example with variables

\[
\text{ground}(P)^X = \begin{cases} 
    a(1, 2) & \leftarrow \\
    b(1) \land c(1) & \leftarrow a(1, 1), \sim c(1) \\
    b(1) \land c(2) & \leftarrow a(1, 2), \sim c(2) \\
    b(2) \land c(1) & \leftarrow a(2, 1), \sim c(1) \\
    b(2) \land c(2) & \leftarrow a(2, 2), \sim c(2) 
\end{cases}
\]

- Consider \( X = \{a(1, 2), b(1)\} \)
- We get \( \text{min}_{\subseteq}( \text{ground}(P)^X ) = \{ \{a(1, 2), b(1)\}, \{a(1, 2), c(2)\} \} \)
- \( X \) is a stable model of \( P \) because \( X \in \text{min}_{\subseteq}( \text{ground}(P)^X ) \)
An example with variables

Disjunctive logic programs

Consider $X = \{a(1, 2), b(1)\}$

We get $\min_{\subseteq}(ground(P)^X) = \{\{a(1, 2), b(1)\}, \{a(1, 2), c(2)\}\}$

$X$ is a stable model of $P$ because $X \in \min_{\subseteq}(ground(P)^X)$
An example with variables

\[ \text{ground}(P)^X = \{ \begin{array}{l}
  a(1, 2) \leftarrow \\
  b(1) ; c(1) \leftarrow a(1, 1), \neg c(1) \\
  b(1) ; c(2) \leftarrow a(1, 2), \neg c(2) \\
  b(2) ; c(1) \leftarrow a(2, 1), \neg c(1) \\
  b(2) ; c(2) \leftarrow a(2, 2), \neg c(2) 
\end{array} \} \]

- Consider \( X = \{ a(1, 2), b(1) \} \)
- We get \( \text{min}_{\subseteq}(\text{ground}(P)^X) = \{ \{ a(1, 2), b(1) \}, \{ a(1, 2), c(2) \} \} \)
- \( X \) is a stable model of \( P \) because \( X \in \text{min}_{\subseteq}(\text{ground}(P)^X) \)
An example with variables

\[ \text{Consider } X = \{ a(1, 2), b(1) \} \]

\[ \text{We get } \min_{\subseteq} (\text{ground}(P)^X) = \{ \{ a(1, 2), b(1) \}, \{ a(1, 2), c(2) \} \} \]

\[ \text{X is a stable model of P because } X \in \min_{\subseteq} (\text{ground}(P)^X) \]
Disjunctive logic programs

An example with variables

\[ \text{ground}(P)^X = \begin{cases} 
  a(1, 2) & \leftarrow \\
  b(1); c(1) & \leftarrow a(1, 1), \sim c(1) \\
  b(1); c(2) & \leftarrow a(1, 2), \sim c(2) \\
  b(2); c(1) & \leftarrow a(2, 1), \sim c(1) \\
  b(2); c(2) & \leftarrow a(2, 2), \sim c(2) 
\end{cases} \]

- Consider \( X = \{a(1, 2), c(2)\} \)
- We get \( \text{min}_{\subseteq}(\text{ground}(P)^X) = \{ \{a(1, 2)\} \} \)
- \( X \) is no stable model of \( P \) because \( X \not\in \text{min}_{\subseteq}(\text{ground}(P)^X) \)
An example with variables

Consider $X = \{a(1,2), c(2)\}$

- We get $\text{min}_\subseteq(\text{ground}(P)^X) = \{\{a(1,2)\}\}$
- $X$ is no stable model of $P$ because $X \notin \text{min}_\subseteq(\text{ground}(P)^X)$
An example with variables

Consider $X = \{a(1, 2), c(2)\}$

We get $\text{min}_{\subseteq}(\text{ground}(P)^X) = \{\{a(1, 2)\}\}$

$X$ is no stable model of $P$ because $X \not\in \text{min}_{\subseteq}(\text{ground}(P)^X)$
Disjunctive logic programs

An example with variables

\[ ground(P)^X = \begin{cases} 
    a(1, 2) & \leftarrow \\
    b(1) ; c(1) & \leftarrow a(1, 1), \sim c(1) \\
    b(1) ; c(2) & \leftarrow a(1, 2), \sim c(2) \\
    b(2) ; c(1) & \leftarrow a(2, 1), \sim c(1) \\
    b(2) ; c(2) & \leftarrow a(2, 2), \sim c(2) 
\end{cases} \]

- Consider \( X = \{a(1, 2), c(2)\} \)
- We get \( \text{min}_{\subseteq}(ground(P)^X) = \{ \{a(1, 2)\} \} \)
- \( X \) is no stable model of \( P \) because \( X \not\in \text{min}_{\subseteq}(ground(P)^X) \)
An example with variables

Consider $X = \{a(1, 2), c(2)\}$

We get $\text{min}_{\subseteq}(\text{ground}(P)^X) = \{\{a(1, 2)\}\}$

$X$ is no stable model of $P$ because $X \not\in \text{min}_{\subseteq}(\text{ground}(P)^X)$
Default negation in rule heads

Consider disjunctive rules of the form

\[ a_1 ; \ldots ; a_m ; \neg a_{m+1} ; \ldots ; \neg a_n \leftarrow a_{n+1}, \ldots, a_o, \neg a_{o+1}, \ldots, \neg a_p \]

where \(0 \leq m \leq n \leq o \leq p\) and each \(a_i\) is an atom for \(0 \leq i \leq p\)

Given a program \(P\) over \(A\), consider the program

\[
\widetilde{P} = \{H(r)^+ \leftarrow B(r) \cup \{\neg \widetilde{a} | a \in H(r)^-\} | r \in P\} \\
\cup \{\widetilde{a} \leftarrow \neg a | r \in P \text{ and } a \in H(r)^-\}
\]

A set \(X\) of atoms is a stable model of a disjunctive program \(P\) (with default negation in rule heads) over \(A\), if \(X = Y \cap A\) for some stable model \(Y\) of \(\widetilde{P}\) over \(A \cup \widetilde{A}\)
Default negation in rule heads

- Consider disjunctive rules of the form

\[ a_1 ; \ldots ; a_m ; \neg a_{m+1} ; \ldots ; \neg a_n \leftarrow a_{n+1}, \ldots, a_o, \neg a_{o+1}, \ldots, \neg a_p \]

where \( 0 \leq m \leq n \leq o \leq p \) and each \( a_i \) is an atom for \( 0 \leq i \leq p \)

- Given a program \( P \) over \( \mathcal{A} \), consider the program

\[ \tilde{P} = \{ H(r)^+ \leftarrow B(r) \cup \{ \neg \tilde{a} | a \in H(r)^- \} | r \in P \} \]
\[ \cup \{ \tilde{a} \leftarrow \neg a | r \in P \text{ and } a \in H(r)^- \} \]

- A set \( X \) of atoms is a stable model of a disjunctive program \( P \) (with default negation in rule heads) over \( \mathcal{A} \), if \( X = Y \cap \mathcal{A} \) for some stable model \( Y \) of \( \tilde{P} \) over \( \mathcal{A} \cup \tilde{\mathcal{A}} \)
Default negation in rule heads

Consider disjunctive rules of the form

$$a_1; \ldots; a_m; \neg a_{m+1}; \ldots; \neg a_n \leftarrow a_{n+1}, \ldots, a_o, \neg a_{o+1}, \ldots, \neg a_p$$

where $0 \leq m \leq n \leq o \leq p$ and each $a_i$ is an atom for $0 \leq i \leq p$

Given a program $P$ over $A$, consider the program

$$\tilde{P} = \{ H(r)^+ \leftarrow B(r) \cup \{ \neg \tilde{a} \mid a \in H(r)^- \} \mid r \in P \}$$
$$\cup \{ \tilde{a} \leftarrow \neg a \mid r \in P \text{ and } a \in H(r)^- \}$$

A set $X$ of atoms is a stable model of a disjunctive program $P$ (with default negation in rule heads) over $A$, if $X = Y \cap A$ for some stable model $Y$ of $\tilde{P}$ over $A \cup \tilde{A}$
An example

- The program

\[ P = \{a ; \neg a \leftarrow\} \]

yields

\[ \tilde{P} = \{a \leftarrow \neg \tilde{a}\} \cup \{\tilde{a} \leftarrow \neg a\} \]

- \( \tilde{P} \) has two stable models, \( \{a\} \) and \( \{\tilde{a}\} \)
- This induces the stable models \( \{a\} \) and \( \emptyset \) of \( P \)
An example

- The program

$$P = \{ a ; \sim a \leftarrow \}$$

yields

$$\tilde{P} = \{ a \leftarrow \sim \tilde{a} \} \cup \{ \tilde{a} \leftarrow \sim a \}$$

- $\tilde{P}$ has two stable models, $\{ a \}$ and $\{ \tilde{a} \}$
- This induces the stable models $\{ a \}$ and $\emptyset$ of $P$
An example

- The program
  \[ P = \{ a ; \neg a \leftarrow \} \]
  yields
  \[ \widetilde{P} = \{ a \leftarrow \neg \widetilde{a} \} \cup \{ \widetilde{a} \leftarrow \neg a \} \]
- \( \widetilde{P} \) has two stable models, \( \{ a \} \) and \( \{ \widetilde{a} \} \)
- This induces the stable models \( \{ a \} \) and \( \emptyset \) of \( P \)
An example

- The program

\[ P = \{ a ; \sim a \leftarrow \} \]

yields

\[ \tilde{P} = \{ a \leftarrow \sim \tilde{a} \} \cup \{ \tilde{a} \leftarrow \sim a \} \]

- \( \tilde{P} \) has two stable models, \( \{ a \} \) and \( \{ \tilde{a} \} \)
- This induces the stable models \( \{ a \} \) and \( \emptyset \) of \( P \)
Propositional theories

Outline

23 Two kinds of negation
24 Disjunctive logic programs
25 Propositional theories
26 Aggregates
27 Gringo language
Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$
  
  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)

- Notation
  
  $\top = (\bot \rightarrow \bot)$

  $\neg \phi = (\phi \rightarrow \bot)$

- A propositional theory is a finite set of formulas
Propositional theories

Formulas are formed from
- atoms in $\mathcal{A}$
- $\bot$

using
- conjunction ($\wedge$)
- disjunction ($\vee$)
- implication ($\rightarrow$)

Notation

\[ T = (\bot \rightarrow \bot) \]
\[ \sim \phi = (\phi \rightarrow \bot) \]

A propositional theory is a finite set of formulas
Propositional theories

Formulas are formed from
- atoms in $\mathcal{A}$
- $\bot$

using
- conjunction ($\land$)
- disjunction ($\lor$)
- implication ($\rightarrow$)

Notation

\[
\top = (\bot \rightarrow \bot) \\
\sim \phi = (\phi \rightarrow \bot)
\]

A propositional theory is a finite set of formulas
The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\land, \lor, \to\}$

If $\phi = \neg \psi = (\psi \to \bot)$, then $\phi^X = (\bot \to \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise.

The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$.
The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\land, \lor, \rightarrow\}$

If $\phi = \neg \psi = (\psi \rightarrow \bot)$,
then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise.

The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$.
Propositional theories

**Reduct**

- The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

- The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \phi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\land, \lor, \rightarrow\}$
  - If $\phi = \neg \psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

- The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$
The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
- If $\phi = \neg \psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$
Propositional theories

The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\land, \lor, \rightarrow\}$

If $\phi = \sim \psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise.

The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$.
The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\land, \lor, \rightarrow\}$

If $\phi = \sim \psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise.

The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$.
Propositional theories

The satisfaction relation $X \models \phi$ between a set $X$ of atoms and a (set of) formula(s) $\phi$ is defined as in propositional logic.

The reduct, $\phi^X$, of a formula $\phi$ relative to a set $X$ of atoms is defined recursively as follows:

- $\phi^X = \bot$ if $X \not\models \phi$
- $\phi^X = \phi$ if $\phi \in X$
- $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\land, \lor, \rightarrow\}$
- If $\phi = \neg \psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

The reduct, $\Phi^X$, of a propositional theory $\Phi$ relative to a set $X$ of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$.
Stable models

- A set $X$ of atoms satisfies a propositional theory $\Phi$, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$.

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\Phi$ is denoted by $\text{min}_{\subseteq}(\Phi)$.

- A set $X$ of atoms is a stable model of a propositional theory $\Phi$, if $X \in \text{min}_{\subseteq}(\Phi^X)$.

- If $X$ is a stable model of $\Phi$, then
  - $X \models \Phi$ and
  - $\text{min}_{\subseteq}(\Phi^X) = \{X\}$

- Note In general, this does not imply $X \in \text{min}_{\subseteq}(\Phi)$!
A set $X$ of atoms satisfies a propositional theory $\Phi$, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$.

The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\Phi$ is denoted by $\text{min}_{\subseteq}(\Phi)$.

A set $X$ of atoms is a stable model of a propositional theory $\Phi$, if $X \in \text{min}_{\subseteq}(\Phi^X)$.

If $X$ is a stable model of $\Phi$, then
- $X \models \Phi$ and
- $\text{min}_{\subseteq}(\Phi^X) = \{X\}$

Note: In general, this does not imply $X \in \text{min}_{\subseteq}(\Phi)$!
Propositional theories

Stable models

- A set $X$ of atoms satisfies a propositional theory $\Phi$, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$.

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\Phi$ is denoted by $\text{min}_{\subseteq}(\Phi)$.

- A set $X$ of atoms is a stable model of a propositional theory $\Phi$, if $X \in \text{min}_{\subseteq}(\Phi^X)$.

- If $X$ is a stable model of $\Phi$, then
  - $X \models \Phi$ and
  - $\text{min}_{\subseteq}(\Phi^X) = \{X\}$

- Note In general, this does not imply $X \in \text{min}_{\subseteq}(\Phi)$!
Stable models

- A set $X$ of atoms satisfies a propositional theory $\Phi$, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$.

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\Phi$ is denoted by $\text{min}_{\subseteq}(\Phi)$.

- A set $X$ of atoms is a stable model of a propositional theory $\Phi$, if $X \in \text{min}_{\subseteq}(\Phi^X)$.

- If $X$ is a stable model of $\Phi$, then
  - $X \models \Phi$ and
  - $\text{min}_{\subseteq}(\Phi^X) = \{X\}$

- Note: In general, this does not imply $X \in \text{min}_{\subseteq}(\Phi)$!
Stable models

- A set $X$ of atoms satisfies a propositional theory $\Phi$, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$.

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\Phi$ is denoted by $\text{min}_{\subseteq}(\Phi)$.

- A set $X$ of atoms is a stable model of a propositional theory $\Phi$, if $X \in \text{min}_{\subseteq}(\Phi^X)$.

- If $X$ is a stable model of $\Phi$, then
  - $X \models \Phi$ and
  - $\text{min}_{\subseteq}(\Phi^X) = \{X\}$

- Note In general, this does not imply $X \in \text{min}_{\subseteq}(\Phi)$!
Two examples

\( \Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \} \)

- For \( X = \{ p, q, r \} \), we get
  \[ \Phi_1^{\{ p, q, r \}} = \{ p \lor (p \rightarrow (q \land r)) \} \]
  and \( \min \subseteq (\Phi_1^{\{ p, q, r \}}) = \{ \emptyset \} \)

- For \( X = \emptyset \), we get
  \[ \Phi_1^{\emptyset} = \{ \bot \lor (\bot \rightarrow \bot) \} \]
  and \( \min \subseteq (\Phi_1^{\emptyset}) = \{ \emptyset \} \)

\( \Phi_2 = \{ p \lor (\sim p \rightarrow (q \land r)) \} \)

- For \( X = \emptyset \), we get
  \[ \Phi_2^{\emptyset} = \{ \bot \} \]
  and \( \min \subseteq (\Phi_2^{\emptyset}) = \emptyset \)

- For \( X = \{ p \} \), we get
  \[ \Phi_2^{\{ p \}} = \{ p \lor (\bot \rightarrow \bot) \} \]
  and \( \min \subseteq (\Phi_2^{\{ p \}}) = \{ \emptyset \} \)

- For \( X = \{ q, r \} \), we get
  \[ \Phi_2^{\{ q, r \}} = \{ \bot \lor (\top \rightarrow (q \land r)) \} \]
  and \( \min \subseteq (\Phi_2^{\{ q, r \}}) = \{ \{ q, r \} \} \)
Two examples

- \( \Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \} \)
  - For \( X = \{p, q, r\} \), we get
    \( \Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \} \) and \( \text{min}_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\} \times \)
  - For \( X = \emptyset \), we get
    \( \Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \} \) and \( \text{min}_{\subseteq}(\Phi_1^\emptyset) = \{\emptyset\} \)

- \( \Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \} \)
  - For \( X = \emptyset \), we get
    \( \Phi_2^\emptyset = \{ \bot \} \) and \( \text{min}_{\subseteq}(\Phi_2^\emptyset) = \emptyset \)
  - For \( X = \{p\} \), we get
    \( \Phi_2^{\{p\}} = \{ p \lor (\bot \rightarrow \bot) \} \) and \( \text{min}_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\} \)
  - For \( X = \{q, r\} \), we get
    \( \Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \} \) and \( \text{min}_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\} \)
Two examples

\( \Phi_1 = \{ p \lor (p \to (q \land r)) \} \)

\( \Phi_1^{\{p,q,r\}} = \{ p \lor (p \to (q \land r)) \} \) and \( \min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{ \emptyset \} \) \( \times \)

For \( X = \emptyset \), we get

\( \Phi_1^{\emptyset} = \{ \bot \lor (\bot \to \bot) \} \) and \( \min_{\subseteq}(\Phi_1^{\emptyset}) = \{ \emptyset \} \) \( \checkmark \)

\( \Phi_2 = \{ p \lor (\neg p \to (q \land r)) \} \)

\( \Phi_2^{\emptyset} = \{ \bot \} \) and \( \min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset \)

For \( X = \emptyset \), we get

\( \Phi_2^{\emptyset} = \{ p \lor (\bot \to \bot) \} \) and \( \min_{\subseteq}(\Phi_2^{\emptyset}) = \{ \emptyset \} \)

For \( X = \{ p \} \), we get

\( \Phi_2^{\{p\}} = \{ p \lor (\bot \to \bot) \} \) and \( \min_{\subseteq}(\Phi_2^{\{p\}}) = \{ \emptyset \} \)

For \( X = \{ q, r \} \), we get

\( \Phi_2^{\{q,r\}} = \{ \bot \lor (\top \to (q \land r)) \} \) and \( \min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{ \{ q, r \} \} \)
Two examples

- $\Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \}$
  - For $X = \{ p, q, r \}$, we get
    $\Phi_1^{p,q,r} = \{ p \lor (p \rightarrow (q \land r)) \}$ and $\text{min} \subseteq (\Phi_1^{p,q,r}) = \{ \emptyset \}$ ×
  - For $X = \emptyset$, we get
    $\Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \}$ and $\text{min} \subseteq (\Phi_1^\emptyset) = \{ \emptyset \}$ ✓

- $\Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \}$
  - For $X = \emptyset$, we get
    $\Phi_2^\emptyset = \{ \bot \}$ and $\text{min} \subseteq (\Phi_2^\emptyset) = \emptyset$
  - For $X = \{ p \}$, we get
    $\Phi_2^{\{ p \}} = \{ p \lor (\bot \rightarrow \bot) \}$ and $\text{min} \subseteq (\Phi_2^{\{ p \}}) = \{ \emptyset \}$
  - For $X = \{ q, r \}$, we get
    $\Phi_2^{\{ q, r \}} = \{ \bot \lor (\top \rightarrow (q \land r)) \}$ and $\text{min} \subseteq (\Phi_2^{\{ q, r \}}) = \{ \{ q, r \} \}$
Propositional theories

Two examples

1. \( \Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \} \)
   - For \( X = \{ p, q, r \} \), we get
     \[ \Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \} \] and \( \text{min}_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{ \emptyset \} \times \)
   - For \( X = \emptyset \), we get
     \[ \Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \} \] and \( \text{min}_{\subseteq}(\Phi_1^\emptyset) = \{ \emptyset \} \checkmark \)

2. \( \Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \} \)
   - For \( X = \emptyset \), we get
     \[ \Phi_2^\emptyset = \{ \bot \} \] and \( \text{min}_{\subseteq}(\Phi_2^\emptyset) = \emptyset \)
   - For \( X = \{ p \} \), we get
     \[ \Phi_2^\{p\} = \{ p \lor (\bot \rightarrow \bot) \} \] and \( \text{min}_{\subseteq}(\Phi_2^\{p\}) = \{ \emptyset \} \)
   - For \( X = \{ q, r \} \), we get
     \[ \Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \} \] and \( \text{min}_{\subseteq}(\Phi_2^{\{q,r\}}) = \{ \{ q, r \} \} \)
Two examples

- $\Phi_1 = \{p \lor (p \rightarrow (q \land r))\}$
  
  For $X = \{p, q, r\}$, we get
  $$\Phi_1^{\{p,q,r\}} = \{p \lor (p \rightarrow (q \land r))\}$$
  and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ (✗)

  For $X = \emptyset$, we get
  $$\Phi_1^\emptyset = \{\bot \lor (\bot \rightarrow \bot)\}$$
  and $\min_{\subseteq}(\Phi_1^\emptyset) = \{\emptyset\}$ (✓)

- $\Phi_2 = \{p \lor (\neg p \rightarrow (q \land r))\}$
  
  For $X = \emptyset$, we get
  $$\Phi_2^\emptyset = \{\bot\}$$
  and $\min_{\subseteq}(\Phi_2^\emptyset) = \emptyset$

  For $X = \{p\}$, we get
  $$\Phi_2^\{p\} = \{p \lor (\bot \rightarrow \bot)\}$$
  and $\min_{\subseteq}(\Phi_2^\{p\}) = \{\emptyset\}$

  For $X = \{q, r\}$, we get
  $$\Phi_2^{\{q,r\}} = \{\bot \lor (\top \rightarrow (q \land r))\}$$
  and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$
Two examples

- $\Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \}$
  - For $X = \{ p, q, r \}$, we get $\Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \}$ and $\text{min}_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{ \emptyset \}$ ×
  - For $X = \emptyset$, we get $\Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \}$ and $\text{min}_{\subseteq}(\Phi_1^\emptyset) = \{ \emptyset \}$ ✓

- $\Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \}$
  - For $X = \emptyset$, we get $\Phi_2^\emptyset = \{ \bot \}$ and $\text{min}_{\subseteq}(\Phi_2^\emptyset) = \emptyset$ ×
  - For $X = \{ p \}$, we get $\Phi_2^\{p\} = \{ p \lor (\bot \rightarrow \bot) \}$ and $\text{min}_{\subseteq}(\Phi_2^\{p\}) = \{ \emptyset \}$
  - For $X = \{ q, r \}$, we get $\Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \}$ and $\text{min}_{\subseteq}(\Phi_2^{\{q,r\}}) = \{ \{ q, r \} \}$
Two examples

$\Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \}$
- For $X = \{ p, q, r \}$, we get
  $\Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \}$ and $\text{min}\subseteq(\Phi_1^{\{p,q,r\}}) = \{ \emptyset \}$ \(\times\)
- For $X = \emptyset$, we get
  $\Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \}$ and $\text{min}\subseteq(\Phi_1^\emptyset) = \{ \emptyset \}$ \(\checkmark\)

$\Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \}$
- For $X = \emptyset$, we get
  $\Phi_2^\emptyset = \{ \bot \}$ and $\text{min}\subseteq(\Phi_2^\emptyset) = \emptyset \times$
- For $X = \{ p \}$, we get
  $\Phi_2^{\{p\}} = \{ p \lor (\bot \rightarrow \bot) \}$ and $\text{min}\subseteq(\Phi_2^{\{p\}}) = \{ \emptyset \}$ \(\checkmark\)
  For $X = \{ q, r \}$, we get
  $\Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \}$ and $\text{min}\subseteq(\Phi_2^{\{q,r\}}) = \{ \{ q, r \} \}$
Propositional theories

Two examples

\[ \Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \} \]
- For \( X = \{ p, q, r \} \), we get
  \[ \Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \} \] and \( \min \subseteq (\Phi_1^{\{p,q,r\}}) = \{\emptyset\} \times \)
- For \( X = \emptyset \), we get
  \[ \Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \} \] and \( \min \subseteq (\Phi_1^\emptyset) = \{\emptyset\} \checkmark \)

\[ \Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \} \]
- For \( X = \emptyset \), we get
  \[ \Phi_2^\emptyset = \{ \bot \} \] and \( \min \subseteq (\Phi_2^\emptyset) = \emptyset \times \)
- For \( X = \{ p \} \), we get
  \[ \Phi_2^\{p\} = \{ p \lor (\bot \rightarrow \bot) \} \] and \( \min \subseteq (\Phi_2^\{p\}) = \{\emptyset\} \times \)
- For \( X = \{ q, r \} \), we get
  \[ \Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \} \] and \( \min \subseteq (\Phi_2^{\{q,r\}}) = \{\{q, r\}\} \)
Two examples

- $\Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \}$ and $\min \subseteq (\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ \(\times\)
  - For $X = \emptyset$, we get
    $\Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \}$ and $\min \subseteq (\Phi_1^\emptyset) = \{\emptyset\}$ \(\checkmark\)

- $\Phi_2 = \{ p \lor (\neg p \rightarrow (q \land r)) \}$
  - For $X = \emptyset$, we get
    $\Phi_2^\emptyset = \{ \bot \}$ and $\min \subseteq (\Phi_2^\emptyset) = \emptyset$ \(\times\)
  - For $X = \{p\}$, we get
    $\Phi_2^\{p\} = \{ p \lor (\bot \rightarrow \bot) \}$ and $\min \subseteq (\Phi_2^\{p\}) = \{\emptyset\}$ \(\times\)
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \}$ and $\min \subseteq (\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$ \(\checkmark\)
Two examples

- $\Phi_1 = \{p \lor (p \rightarrow (q \land r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_{1}^{\{p,q,r\}} = \{p \lor (p \rightarrow (q \land r))\}$ and $\text{min} \subseteq (\Phi_{1}^{\{p,q,r\}}) = \{\emptyset\}$ $\times$
  - For $X = \emptyset$, we get
    $\Phi_{1}^{\emptyset} = \{\bot \lor (\bot \rightarrow \bot)\}$ and $\text{min} \subseteq (\Phi_{1}^{\emptyset}) = \{\emptyset\}$ $\checkmark$

- $\Phi_2 = \{p \lor (\neg p \rightarrow (q \land r))\}$
  - For $X = \emptyset$, we get
    $\Phi_{2}^{\emptyset} = \{\bot\}$ and $\text{min} \subseteq (\Phi_{2}^{\emptyset}) = \emptyset$ $\times$
  - For $X = \{p\}$, we get
    $\Phi_{2}^{\{p\}} = \{p \lor (\bot \rightarrow \bot)\}$ and $\text{min} \subseteq (\Phi_{2}^{\{p\}}) = \{\emptyset\}$ $\times$
  - For $X = \{q, r\}$, we get
    $\Phi_{2}^{\{q,r\}} = \{\bot \lor (\top \rightarrow (q \land r))\}$ and $\text{min} \subseteq (\Phi_{2}^{\{q,r\}}) = \{\{q, r\}\}$ $\checkmark$
Two examples

- \( \Phi_1 = \{ p \lor (p \rightarrow (q \land r)) \} \)
  - For \( X = \{ p, q, r \} \), we get
    \[ \Phi_1^{\{p,q,r\}} = \{ p \lor (p \rightarrow (q \land r)) \} \] and \( \min \subseteq (\Phi_1^{\{p,q,r\}}) = \{ \emptyset \} \times \)
  - For \( X = \emptyset \), we get
    \[ \Phi_1^\emptyset = \{ \bot \lor (\bot \rightarrow \bot) \} \] and \( \min \subseteq (\Phi_1^\emptyset) = \{ \emptyset \} \checkmark \)

- \( \Phi_2 = \{ p \lor (\sim p \rightarrow (q \land r)) \} \)
  - For \( X = \emptyset \), we get
    \[ \Phi_2^\emptyset = \{ \bot \} \] and \( \min \subseteq (\Phi_2^\emptyset) = \emptyset \times \)
  - For \( X = \{ p \} \), we get
    \[ \Phi_2^\{p\} = \{ p \lor (\bot \rightarrow \bot) \} \] and \( \min \subseteq (\Phi_2^\{p\}) = \{ \emptyset \} \times \)
  - For \( X = \{ q, r \} \), we get
    \[ \Phi_2^{\{q,r\}} = \{ \bot \lor (\top \rightarrow (q \land r)) \} \] and \( \min \subseteq (\Phi_2^{\{q,r\}}) = \{ \{ q, r \} \} \checkmark \)
Propositional theories

Relationship to logic programs

- The translation, $\tau[\phi \leftarrow \psi]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if $\phi$ is an atom
  - $\tau[\neg\phi] = \neg\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \land \tau[\psi])$
  - $\tau[\langle \phi; \psi \rangle] = (\tau[\phi] \lor \tau[\psi])$

- The translation of a logic program $P$ is $\tau[P] = \{\tau[r] \mid r \in P\}$

- Given a logic program $P$ and a set $X$ of atoms, $X$ is a stable model of $P$ iff $X$ is a stable model of $\tau[P]$. 
The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:

- $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
- $\tau[\bot] = \bot$
- $\tau[\top] = \top$
- $\tau[\phi] = \phi$ if $\phi$ is an atom
- $\tau[\neg \phi] = \neg \tau[\phi]$
- $\tau[(\phi, \psi)] = (\tau[\phi] \land \tau[\psi])$
- $\tau[(\phi; \psi)] = (\tau[\phi] \lor \tau[\psi])$

The translation of a logic program $P$ is $\tau[P] = \{\tau[r] \mid r \in P\}$

Given a logic program $P$ and a set $X$ of atoms, $X$ is a stable model of $P$ iff $X$ is a stable model of $\tau[P]$. 
The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:

- $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
- $\tau[\perp] = \perp$
- $\tau[\top] = \top$
- $\tau[\phi] = \phi$ if $\phi$ is an atom
- $\tau[\lnot \phi] = \lnot \tau[\phi]$
- $\tau[(\phi, \psi)] = (\tau[\phi] \land \tau[\psi])$
- $\tau[(\phi; \psi)] = (\tau[\phi] \lor \tau[\psi])$

The translation of a logic program $P$ is $\tau[P] = \{\tau[r] | r \in P\}$

Given a logic program $P$ and a set $X$ of atoms, $X$ is a stable model of $P$ iff $X$ is a stable model of $\tau[P]$.
The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:

- $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
- $\tau[\bot] = \bot$
- $\tau[\top] = \top$
- $\tau[\phi] = \phi$ if $\phi$ is an atom
- $\tau[\sim \phi] = \sim \tau[\phi]$
- $\tau[(\phi, \psi)] = (\tau[\phi] \land \tau[\psi])$
- $\tau[(\phi; \psi)] = (\tau[\phi] \lor \tau[\psi])$

The translation of a logic program $P$ is $\tau[P] = \{\tau[r] | r \in P\}$

Given a logic program $P$ and a set $X$ of atoms, $X$ is a stable model of $P$ iff $X$ is a stable model of $\tau[P]$. 
Propositional theories

Logic programs as propositional theories

- The normal logic program $P = \{ p \leftarrow \neg q, \; q \leftarrow \neg p \}$ corresponds to $\tau[P] = \{ \neg q \rightarrow p, \; \neg p \rightarrow q \}$
  - stable models: $\{p\}$ and $\{q\}$

- The disjunctive logic program $P = \{ p \; ; \; q \leftarrow \}$ corresponds to $\tau[P] = \{ \top \rightarrow p \lor q \}$
  - stable models: $\{p\}$ and $\{q\}$

- The nested logic program $P = \{ p \leftarrow \neg\neg p \}$ corresponds to $\tau[P] = \{ \neg\neg p \rightarrow p \}$
  - stable models: $\emptyset$ and $\{p\}$
Logic programs as propositional theories

- The normal logic program $P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \}$
  corresponds to $\tau[P] = \{ \neg q \rightarrow p, \ \neg p \rightarrow q \}$
  stable models: $\{p\}$ and $\{q\}$

- The disjunctive logic program $P = \{ p ; q \leftarrow \}$
  corresponds to $\tau[P] = \{ \top \rightarrow p \lor q \}$
  stable models: $\{p\}$ and $\{q\}$

- The nested logic program $P = \{ p \leftarrow \neg
  \neg p \}$
  corresponds to $\tau[P] = \{ \neg\neg p \rightarrow p \}$
  stable models: $\emptyset$ and $\{p\}$
Propositional theories

Logic programs as propositional theories

- The normal logic program $P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \}$ corresponds to $\tau[P] = \{ \neg q \rightarrow p, \ \neg p \rightarrow q \}$
  - stable models: $\{p\}$ and $\{q\}$

- The disjunctive logic program $P = \{ p ; q \leftarrow \}$ corresponds to $\tau[P] = \{ T \rightarrow p \lor q \}$
  - stable models: $\{p\}$ and $\{q\}$

- The nested logic program $P = \{ p \leftarrow \neg \neg p \}$ corresponds to $\tau[P] = \{ \neg \neg p \rightarrow p \}$
  - stable models: $\emptyset$ and $\{p\}$
Propositional theories

Logic programs as propositional theories

- The normal logic program $P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \}$ corresponds to $\tau[P] = \{ \neg q \rightarrow p, \ \neg p \rightarrow q \}$
  - stable models: $\{ p \}$ and $\{ q \}$

- The disjunctive logic program $P = \{ p ; q \leftarrow \}$ corresponds to $\tau[P] = \{ \top \rightarrow p \lor q \}$
  - stable models: $\{ p \}$ and $\{ q \}$

- The nested logic program $P = \{ p \leftarrow \neg \neg p \}$ corresponds to $\tau[P] = \{ \neg \neg p \rightarrow p \}$
  - stable models: $\emptyset$ and $\{ p \}$
Logic programs as propositional theories

- The normal logic program $P = \{ p \leftarrow \neg q, \ q \leftarrow \neg p \}$ corresponds to $\tau[P] = \{ \neg q \rightarrow p, \ \neg p \rightarrow q \}$
  - stable models: $\{p\}$ and $\{q\}$

- The disjunctive logic program $P = \{ p \ ; \ q \leftarrow \}$ corresponds to $\tau[P] = \{ \top \rightarrow p \lor q \}$
  - stable models: $\{p\}$ and $\{q\}$

- The nested logic program $P = \{ p \leftarrow \neg \neg p \}$ corresponds to $\tau[P] = \{ \neg \neg p \rightarrow p \}$
  - stable models: $\emptyset$ and $\{p\}$
Logic programs as propositional theories

- The normal logic program $P = \{ p ← \sim q, \ q ← \sim p \}$ corresponds to $\tau[P] = \{ \sim q → p, \ \sim p → q \}$
  - stable models: $\{ p \}$ and $\{ q \}$

- The disjunctive logic program $P = \{ p ; q ← \}$ corresponds to $\tau[P] = \{ \top → p \lor q \}$
  - stable models: $\{ p \}$ and $\{ q \}$

- The nested logic program $P = \{ p ← \sim \sim p \}$ corresponds to $\tau[P] = \{ \sim \sim p → p \}$
  - stable models: $\emptyset$ and $\{ p \}$
23 Two kinds of negation
24 Disjunctive logic programs
25 Propositional theories
26 Aggregates
27 Gringo language
Motivation

- Aggregates provide a general way to obtain a single value from a collection of input values
- Popular aggregate (functions)
  - average
  - count
  - maximum
  - minimum
  - sum
- Cardinality and weight constraints rely on count and sum aggregates
Aggregates

Syntax

- An aggregate has the form:

\[ \alpha \{ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \sim a_{m+1}, \ldots, w_n : \sim a_n \} \prec k \]

where for \( 1 \leq i \leq n \)

- \( \alpha \) stands for a function mapping multisets over \( \mathbb{Z} \) to \( \mathbb{Z} \cup \{ +\infty, -\infty \} \)
- \( \prec \) stands for a relation between \( \mathbb{Z} \cup \{ +\infty, -\infty \} \) and \( \mathbb{Z} \)
- \( k \in \mathbb{Z} \)
- \( a_i \) are atoms and
- \( w_i \) are integers

- Example: \textit{sum} \( \{ 30 : \text{hd}(a), \ldots, 50 : \text{hd}(m) \} \leq 300 \)
A (positive) aggregate $\alpha \{ w_1 : a_1, \ldots, w_n : a_n \} \prec k$ can be represented by the formula:

$$\bigwedge_{I \subseteq \{1, \ldots, n\}, \alpha \{ w_i \mid i \in I \} \not\prec k} \left( \bigwedge_{i \in I} a_i \rightarrow \bigvee_{i \in I^c} a_i \right)$$

where $I^c = \{1, \ldots, n\} \setminus I$ and $\not\prec$ is the complement of $\prec$.

Then, $\alpha \{ w_1 : a_1, \ldots, w_n : a_n \} \prec k$ is true in $X$ iff the above formula is true in $X$. 
Consider $\text{sum}\{1 : p, 1 : q\} \neq 1$

That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$

**Calculemus!**

<table>
<thead>
<tr>
<th>$l$</th>
<th>${w_i \mid i \in l}$</th>
<th>$\sum{w_i \mid i \in l}$</th>
<th>$\sum{w_i \mid i \in l} = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>${}$</td>
<td>$0$</td>
<td>$\text{false}$</td>
</tr>
<tr>
<td>${1}$</td>
<td>${1}$</td>
<td>$1$</td>
<td>$\text{true}$</td>
</tr>
<tr>
<td>${2}$</td>
<td>${1}$</td>
<td>$1$</td>
<td>$\text{true}$</td>
</tr>
<tr>
<td>${1, 2}$</td>
<td>${1, 1}$</td>
<td>$2$</td>
<td>$\text{false}$</td>
</tr>
</tbody>
</table>

- We get $(p \rightarrow q) \land (q \rightarrow p)$
- Analogously, we obtain $(p \lor q) \land \neg(p \land q)$ for $\text{sum}\{1 : p, 1 : q\} = 1$
Consider \( \text{sum}\{1 : p, 1 : q\} \neq 1 \)
That is, \( a_1 = p, a_2 = q \) and \( w_1 = 1, w_2 = 1 \)

**Calculemus!**

<table>
<thead>
<tr>
<th>( l )</th>
<th>( { w_i \mid i \in l } )</th>
<th>( \sum { w_i \mid i \in l } )</th>
<th>( \sum { w_i \mid i \in l } = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>( { } )</td>
<td>0</td>
<td>( false )</td>
</tr>
<tr>
<td>( {1} )</td>
<td>( {1} )</td>
<td>1</td>
<td>( true )</td>
</tr>
<tr>
<td>( {2} )</td>
<td>( {1} )</td>
<td>1</td>
<td>( true )</td>
</tr>
<tr>
<td>( {1, 2} )</td>
<td>( {1, 1} )</td>
<td>2</td>
<td>( false )</td>
</tr>
</tbody>
</table>

We get \((p \rightarrow q) \land (q \rightarrow p)\)

Analogously, we obtain \((p \lor q) \land \neg(p \land q)\) for \( \text{sum}\{1 : p, 1 : q\} = 1 \)
Consider $\text{sum}\{1 : p, 1 : q\} \neq 1$

That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$

**Calculemus!**

<table>
<thead>
<tr>
<th>$l$</th>
<th>${w_i \mid i \in l}$</th>
<th>$\sum{w_i \mid i \in l}$</th>
<th>$\sum{w_i \mid i \in l} = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>${}$</td>
<td>0</td>
<td>false</td>
</tr>
<tr>
<td>${1}$</td>
<td>${1}$</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>${2}$</td>
<td>${1}$</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>${1, 2}$</td>
<td>${1, 1}$</td>
<td>2</td>
<td>false</td>
</tr>
</tbody>
</table>

- We get $(p \rightarrow q) \land (q \rightarrow p)$
- Analogously, we obtain $(p \lor q) \land \neg(p \land q)$ for $\text{sum}\{1 : p, 1 : q\} = 1$
Monotonicity

- **Monotone aggregates**
  - For instance,
    - $B(r)^+$
    - $\text{sum}\{1 : p, 1 : q\} > 1$ amounts to $p \land q$
  - We get a simpler characterization: $\bigwedge_{I \subseteq \{1,\ldots,n\}, \alpha\{w_i | i \in I\} \not\prec k} \bigvee_{i \in I} a_i$

- **Anti-monotone aggregates**
  - For instance,
    - $B(r)^-$
    - $\text{sum}\{1 : p, 1 : q\} < 1$ amounts to $\neg p \land \neg q$
  - We get a simpler characterization: $\bigwedge_{I \subseteq \{1,\ldots,n\}, \alpha\{w_i | i \in I\} \not\prec k} \neg \bigwedge_{i \in I} a_i$

- **Non-monotone aggregates**
  - For instance, $\text{sum}\{1 : p, 1 : q\} \neq 1$ is non-monotone.
Monotonicity

■ Monotone aggregates
  ■ For instance,
    ■ \( B(r)^+ \)
    ■ \( sum\{1 : p, 1 : q\} > 1 \) amounts to \( p \land q \)
  ■ We get a simpler characterization: \( \land_{I \subseteq \{1, \ldots, n\}, \alpha\{w_i|i \in I\} \not\preceq k \lor_{i \in I} a_i \) 

■ Anti-monotone aggregates
  ■ For instance,
    ■ \( B(r)^- \)
    ■ \( sum\{1 : p, 1 : q\} < 1 \) amounts to \( \neg p \land \neg q \)
  ■ We get a simpler characterization: \( \land_{I \subseteq \{1, \ldots, n\}, \alpha\{w_i|i \in I\} \not\preceq k \land_{i \in I} a_i \) 

■ Non-monotone aggregates
  ■ For instance, \( sum\{1 : p, 1 : q\} \neq 1 \) is non-monotone.
Outline

23 Two kinds of negation
24 Disjunctive logic programs
25 Propositional theories
26 Aggregates
27 Gringo language
Gringo language

Gringo language

Problem

Solution

Modeling

Interpreting

Logic Program

Grounder

Solver

Stable Models

Modeling

Solving

Solving

- aspif format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard ASP-Core-2
- **aspi format** is a **machine-oriented** standard for ground programs
- **gringo format** is a **user-oriented** language for (non-ground) programs extending the ASP language standard **ASP-Core-2**
Gringo language

Problem

Logic Program

Solving

Grounder

Solver

Stable Models

Solution

Modeling

Modeling Interpreting

Gringo format

aspi format

- aspi format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard ASP-Core-2
Gringo language

Problem

Modeling

Logic Program

gringo format

Grounder

Solving

Solver

Stable Models

Interpreting

Solution

aspif format

- aspif format is a machine-oriented standard for ground programs
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard ASP-Core-2
Terms and literals

- Terms $t$
- Tuples $t$
- Atoms $a, \neg a$
- Symbolic literals $a, \sim a, \sim\sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots ; t_n : L_n \} \prec_2 s_2$
- Aggregate literals $a, \sim a, \sim\sim a$
- Literals
Terms and literals

- **Terms** $t$ are formed from:
  - constant symbols, e.g., $c, d, \ldots$
  - function symbols, e.g., $f, g, \ldots$
  - numeric symbols, e.g., $1, 2, \ldots$
  - variable symbols, e.g., $X, Y, \ldots$
  - parentheses (, )
  - tuple delimiters $⟨, ⟩$ (omitted whenever possible)

- **Tuples** $t$

- **Atoms** $a, \neg a$

- **Symbolic literals** $a, \neg a, \neg\neg a$

- **Arithmetic literals** $t_1 ≺ t_2$

- **Conditional literals** $l : L$

- **Aggregate atoms** $s_1 ≺_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} ≺_2 s_2$

- **Aggregate literals** $a, \neg a, \neg\neg a$

- **Literals**
Terms and literals

- Terms $t$ are formed from
  - constants, eg $c, d, \ldots$
  - functions, eg $f, g, \ldots$
  - numerics, eg $1, 2, \ldots$
  - variables, eg $x, y, \ldots$
  - parentheses (, )
  - tuple delimiters ⟨, ⟩

- Tuples $t$

- Atoms $a, \neg a$

- Symbolic literals $a, \neg a, \neg\neg a$

- Arithmetic literals $t_1 \prec t_2$

- Conditional literals $l : L$

- Aggregate atoms $s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots ; t_n : L_n \} \prec_2 s_2$

- Aggregate literals $a, \neg a, \neg\neg a$

- Literals
Terms and literals

- **Terms** \( t \) are formed from
  - constants, eg \( c, d, \ldots \)
  - functions, eg \( f, g, \ldots \)
  - numerics, eg \( 1, 2, \ldots \)
  - variables, eg \( X, Y, \ldots, _ \)
  - parentheses (, )
  - tuple delimiters \( ⟨, ⟩ \)
  
  eg \( f(3, c, Z), g(42, _, _), \) or \( f((3, c), X) \)

- **Tuples** \( t \)
- **Atoms** \( a, \neg a \)
- **Symbolic literals** \( a, \neg a, \neg\neg a \)
- **Arithmetic literals** \( t_1 \prec t_2 \)
- **Conditional literals** \( l : L \)
- **Aggregate atoms** \( s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2 \)
- **Aggregate literals** \( a, \neg a, \neg\neg a \)
- **Literals**
Terms and literals

- Terms $t$
- Tuples $t$ of terms
- Atoms $a, \neg a$
- Symbolic literals $a, \neg a, \neg \neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2$
- Aggregate literals $a, \neg a, \neg \neg a$
- Literals
Terms and literals

- Terms \( t \)
- Tuples \( t \)
- (Negated) Atoms \( a, \neg a \) are formed from
  - predicate symbols, eg \( p, q, \ldots \)
  - parentheses (, )
  - tuples of terms
- Symbolic literals \( a, \neg a, \neg \neg a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \)
- Aggregate atoms \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2 \)
- Aggregate literals \( a, \neg a, \neg \neg a \)
- Literals
Terms and literals

- Terms $t$
- Tuples $t$
- Atoms $a, \neg a$ are formed from
  - predicates, eg $p, q, \ldots$
  - parentheses $(, )$
  - tuples of terms
- Symbolic literals $a, \neg a, \neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2$
- Aggregate literals $a, \neg a, \neg\neg a$
- Literals
Terms and literals

- Terms  $t$
- Tuples  $t$
- Atoms  $a, \neg a$ are formed from
  - predicates, eg $p, q, \ldots$
  - parentheses $(, )$
  - tuples of terms

  eg  $-p(f(3,c,Z),g(42,_,_))$ or $q()$ written as $q$

- Symbolic literals  $a, \sim a, \sim \sim a$
- Arithmetic literals  $t_1 \prec t_2$
- Conditional literals  $l : L$
- Aggregate atoms  $s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2$
- Aggregate literals  $a, \sim a, \sim \sim a$
- Literals
Terms and literals

- Terms \( t \)
- Tuples \( t \)
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \sim a, \sim \sim a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \)
- Aggregate atoms \( s_1 \prec_1 \alpha\{t_1 : L_1; \ldots ; t_n : L_n\} \prec_2 s_2 \)
- Aggregate literals \( a, \sim a, \sim \sim a \)
- Literals
Terms and literals

- **Terms** \( t \)
- **Tuples** \( t \)
- **Atoms** \( a, \neg a, \bot, \top \)
  - viz \#false and \#true
- **Symbolic literals** \( a, \neg a, \neg\neg a \)
- **Arithmetic literals** \( t_1 \prec t_2 \)
- **Conditional literals** \( l : L \)
- **Aggregate atoms** \( s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2 \)
- **Aggregate literals** \( a, \neg a, \neg\neg a \)
- **Literals**
Terms and literals

- Terms $t$
- Tuples $t$
- Atoms $a$, $\neg a$, $\perp$, $\top$
- Symbolic literals $a$, $\sim a$, $\sim \sim a$
- Arithmetic literals $t_1 \preceq t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \preceq_1 \alpha \{ t_1 : L_1 ; \ldots ; t_n : L_n \} \preceq_2 s_2$
- Aggregate literals $a$, $\sim a$, $\sim \sim a$
- Literals
Terms and literals

- Terms $t$
- Tuples $t$
- Atoms $a, \neg a, \bot, \top$
- Symbolic literals $a, \neg a, \neg\neg a$
  - e.g. $p(a,X)$, ‘not $p(a,X)$’, ‘not not $p(a,X)$’
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2$
- Aggregate literals $a, \neg a, \neg\neg a$
- Literals
Terms and literals

- **Terms** \( t \)
- **Tuples** \( t \)
- **Atoms** \( a, \neg a, \bot, \top \)
- **Symbolic literals** \( a, \sim a, \sim \sim a \)
- **Arithmetic literals** \( t_1 \prec t_2 \) where
  - \( t_1 \) and \( t_2 \) are terms
  - \( \prec \) is a comparison symbol
- **Conditional literals** \( l : L \)
- **Aggregate atoms** \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2 \)
- **Aggregate literals** \( a, \sim a, \sim \sim a \)
- **Literals**
Terms and literals

- Terms \( t \)
- Tuples \( t \)
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \sim a, \sim \sim a \)
- Arithmetic literals \( t_1 \prec t_2 \) where
  - \( t_1 \) and \( t_2 \) are terms
  - \( \prec \) is a comparison symbol
  - eg \( 3 < 1 \) or \( f(42) = X \)
- Conditional literals \( l : L \)
- Aggregate atoms \( s_1 \prec_1 \alpha \{ t_1 : L_1, \ldots ; t_n : L_n \} \prec_2 s_2 \)
- Aggregate literals \( a, \sim a, \sim \sim a \)
- Literals
Terms and literals

- Terms \( t \)
- Tuples \( t, L \) of literals
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \sim a, \sim \sim a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \) where
  - \( l \) is a symbolic or arithmetic literal
  - \( L \) is a tuple of symbol or arithmetic literals
- Aggregate atoms \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2 \)
- Aggregate literals \( a, \sim a, \sim \sim a \)
- Literals
Terms and literals

- Terms \( t \)
- Tuples \( t, L \)
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \neg a, \neg\neg a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \) where
  - \( l \) is a symbolic or arithmetic literal
  - \( L \) is a tuple of symbol or arithmetic literals
  - \( l : L \) is written as \( l \) whenever \( L \) is empty
- Aggregate atoms \( s_1 \prec_1 a\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2 \)
- Aggregate literals \( a, \neg a, \neg\neg a \)
- Literals
Terms and literals

- Terms \( t \)
- Tuples \( t, L \)
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \sim a, \sim\sim a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \) where
  - \( l \) is a symbolic or arithmetic literal
  - \( L \) is a tuple of symbol or arithmetic literals

  eg ‘\( p(X,Y) : q(X), r(Y) \)’ or \( p(42) \) or ‘\#false : q’

- Aggregate atoms \( s_1 \prec_1 \alpha \{ t_1 : L_1, \ldots ; t_n : L_n \} \prec_2 s_2 \)
- Aggregate literals \( a, \sim a, \sim\sim a \)
- Literals
Gringo language

Terms and literals

- **Terms**  $t$
- **Tuples**  $t, L$
- **Atoms**  $a, \neg a, \bot, \top$
- **Symbolic literals**  $a, \sim a, \sim\sim a$
- **Arithmetic literals**  $t_1 \prec t_2$
- **Conditional literals**  $l : L$
- **Aggregate atoms**  $s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2$  where
  - $\alpha$ is an aggregate name
  - $t_1 : L_1, \ldots, t_n : L_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - $s_1$ and $s_2$ are terms
- **Aggregate literals**  $a, \sim a, \sim\sim a$
- **Literals**
Terms and literals

- Terms $t$
- Tuples $t, L$
- Atoms $a, \neg a, \bot, \top$
- Symbolic literals $a, \neg a, \neg \neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2$ where
  - $\alpha$ is an aggregate name
  - $t_1 : L_1, \ldots, t_n : L_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - $s_1$ and $s_2$ are terms
  - one (or both) of ‘$s_1 \prec_1’ and ‘$\prec_2 s_2’ can be omitted

- Aggregate literals $a, \neg a, \neg \neg a$
- Literals

\[ \text{Gringo language} \]

\[ \text{Terms and literals} \]

- Terms $t$
- Tuples $t, L$
- Atoms $a, \neg a, \bot, \top$
- Symbolic literals $a, \neg a, \neg \neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2$ where
  - $\alpha$ is an aggregate name
  - $t_1 : L_1, \ldots, t_n : L_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - $s_1$ and $s_2$ are terms
  - one (or both) of ‘$s_1 \prec_1’ and ‘$\prec_2 s_2’ can be omitted

- Aggregate literals $a, \neg a, \neg \neg a$
- Literals
Terms and literals

- **Terms**  \( t \)
- **Tuples**  \( t, L \)
- **Atoms**  \( a, \neg a, \bot, \top \)
- **Symbolic literals**  \( a, \sim a, \sim\sim a \)
- **Arithmetic literals**  \( t_1 \prec t_2 \)
- **Conditional literals**  \( l : L \)
- **Aggregate atoms**  \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2 \) where
  - \( \alpha \) is an aggregate name
  - \( t_1 : L_1, \ldots, t_n : L_n \) are conditional literals
  - \( \prec_1 \) and \( \prec_2 \) are comparison symbols
  - \( s_1 \) and \( s_2 \) are terms
  - omitting \( \prec_1 \) or \( \prec_2 \) defaults to \( \leq \)
- **Aggregate literals**  \( a, \sim a, \sim\sim a \)
- **Literals** are conditional or aggregate literals
Terms and literals

- Terms $t$
- Tuples $t, L$
- Atoms $a, \neg a, \bot, \top$
- Symbolic literals $a, \sim a, \sim \sim a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha \{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2$ where
  - $\alpha$ is an aggregate name
  - $t_1 : L_1, \ldots, t_n : L_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - $s_1$ and $s_2$ are terms

  eg $10 \leq \# \text{sum} \{6,C:\text{course}(C); 3,S:\text{seminar}(S)\} \leq 20$

- Aggregate literals $a, \sim a, \sim \sim a$
- Literals are conditional or aggregate literals

Potassco
Terms and literals

- Terms \( t \)
- Tuples \( t, L \)
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \sim a, \sim\sim a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \)
- Aggregate atoms \( s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2 \) where
  - \( \alpha \) is an aggregate name
  - \( t_1 : L_1, \ldots, t_n : L_n \) are conditional literals
  - \( \prec_1 \) and \( \prec_2 \) are comparison symbols
  - \( s_1 \) and \( s_2 \) are terms

  eg 10 \#sum \{6,C:course(C); 3,S:seminar(S)\} 20
- Aggregate literals \( a, \neg a, \sim\sim a \)
- Literals are conditional or aggregate literals
Terms and literals

- Terms  \( t \)
- Tuples  \( t, L \)
- Atoms  \( a, \neg a, \bot, \top \)
- Symbolic literals  \( a, \sim a, \sim \sim a \)
- Arithmetic literals  \( t_1 \prec t_2 \)
- Conditional literals  \( l : L \)
- Aggregate atoms  \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots ; t_n : L_n \} \prec_2 s_2 \)
- Aggregate literals  \( a, \sim a, \sim \sim a \) where
  - \( a \) is an aggregate atom
- Literals are conditional or aggregate literals
Terms and literals

- **Terms** \( t \)
- **Tuples** \( t, L \)
- **Atoms** \( a, \neg a, \bot, \top \)
- **Symbolic literals** \( a, \sim a, \sim \sim a \)
- **Arithmetic literals** \( t_1 \prec t_2 \)
- **Conditional literals** \( l : L \)
- **Aggregate atoms** \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2 \)
- **Aggregate literals** \( a, \sim a, \sim \sim a \) where
  - \( a \) is an aggregate atom
  - `eg` not 10 \( \# \text{sum} \{6, \text{course}(C); 3, \text{seminar}(S)\} \) 20
- **Literals** are conditional or aggregate literals
Terms and literals

- Terms \( t \)
- Tuples \( t, L \)
- Atoms \( a, \neg a, \bot, \top \)
- Symbolic literals \( a, \sim a, \sim \sim a \)
- Arithmetic literals \( t_1 \prec t_2 \)
- Conditional literals \( l : L \)
- Aggregate atoms \( s_1 \prec_1 \alpha \{ t_1 : L_1; \ldots; t_n : L_n \} \prec_2 s_2 \)
- Aggregate literals \( a, \sim a, \sim \sim a \)
- Literals are conditional or aggregate literals
Terms and literals

- Terms $t$
- Tuples $t, L$
- Atoms $a, \neg a, \bot, \top$
- Symbolic literals $a, \neg a, \neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : L$
- Aggregate atoms $s_1 \prec_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} \prec_2 s_2$
- Aggregate literals $a, \neg a, \neg\neg a$
- Literals are conditional or aggregate literals

For a detailed account please consult the user’s guide!
Rules are of the form

\[ l_1; \ldots ; l_m \leftarrow l_{m+1}, \ldots , l_n \]  

where

- \( l_i \) is a conditional literal for \( 1 \leq i \leq m \) and
- \( l_i \) is a literal for \( m + 1 \leq i \leq n \)

Note Semicolons ‘;’ must be used in (2) instead of commas ‘,’ whenever some \( l_i \) is a (genuine) conditional literal for \( 1 \leq i \leq n \)

Example  \( a(X) :- b(X) : c(X), d(X); e(x) \).
Rules are of the form

\[ l_1 ; \ldots ; l_m \leftarrow l_{m+1}, \ldots, l_n \]  

(2)

where

- \( l_i \) is a conditional literal for \( 1 \leq i \leq m \) and
- \( l_i \) is a literal for \( m + 1 \leq i \leq n \)

Note Semicolons ‘;’ must be used in (2) instead of commas ‘,’ whenever some \( l_i \) is a (genuine) conditional literal for \( 1 \leq i \leq n \)

Example \( a(X) : - b(X) : c(X), d(X); e(x) \).
Rules are of the form

\[ l_1; \ldots ; l_m \leftarrow l_{m+1}, \ldots , l_n \]  

where

- \( l_i \) is a conditional literal for \( 1 \leq i \leq m \) and
- \( l_i \) is a literal for \( m + 1 \leq i \leq n \)

Note Semicolons ‘;’ must be used in (2) instead of commas ‘,’ whenever some \( l_i \) is a (genuine) conditional literal for \( 1 \leq i \leq n \)

Example  \( a(X) : - b(X) : c(X), d(X); e(x) \).
Shortcuts

A rule of the form

\[ s_1 \prec_1 \alpha\{t_1 : l_1 : L_1; \ldots; t_k : l_k : L_k\} \prec_2 s_2 \leftarrow l_{m+1}, \ldots, l_n \]

where

- \( \alpha, \prec_i, s_i, t_j \) are as given above for \( i = 1, 2 \) and \( 1 \leq j \leq k \)
- \( l_j : L_j \) is a conditional literal for \( 1 \leq j \leq k \)
- \( l_i \) is a literal for \( m + 1 \leq i \leq n \) (as in (2))

is a shorthand for the following \( k + 1 \) rules

\[ \{l_j\} \leftarrow l_{m+1}, \ldots, l_n, L_j \quad \text{for } 1 \leq j \leq k \]
\[ \leftarrow l_{m+1}, \ldots, l_n, \sim s_1 \prec_1 \alpha\{t_1 : l_1, L_1; \ldots; t_k : l_k, L_k\} \prec_2 s_2 \]

Example: \( 10 < \#\text{sum} \{C, X, Y : \text{edge}(X, Y) : \text{cost}(X, Y, C)\} \)
A rule of the form

\[ s_1 \prec_1 \alpha \{ t_1 : l_1 : L_1; \ldots; t_k : l_k : L_k \} \prec_2 s_2 \leftarrow l_{m+1}, \ldots, l_n \]

where

- \( \alpha, \prec_i, s_i, t_j \) are as given above for \( i = 1, 2 \) and \( 1 \leq j \leq k \)
- \( l_j : L_j \) is a conditional literal for \( 1 \leq j \leq k \)
- \( l_i \) is a literal for \( m + 1 \leq i \leq n \) (as in (2))

is a shorthand for the following \( k + 1 \) rules

\[ \{ l_j \} \leftarrow l_{m+1}, \ldots, l_n, L_j \quad \text{for} \quad 1 \leq j \leq k \]
\[ \leftarrow l_{m+1}, \ldots, l_n, \sim s_1 \prec_1 \alpha \{ t_1 : l_1, L_1; \ldots; t_k : l_k, L_k \} \prec_2 s_2 \]

Example: \[ 10 < \# \text{sum} \{ C, X, Y : \text{edge}(X,Y) : \text{cost}(X,Y,C) \} \].
Shortcuts

- A rule of the form

\[ s_1 \prec_1 \alpha \{ t_1 : l_1 : L_1; \ldots; t_k : l_k : L_k \} \prec_2 s_2 \leftarrow l_{m+1}, \ldots, l_n \]

where

- \( \alpha, \prec_i, s_i, t_j \) are as given above for \( i = 1, 2 \) and \( 1 \leq j \leq k \)
- \( l_j : L_j \) is a conditional literal for \( 1 \leq j \leq k \)
- \( l_i \) is a literal for \( m + 1 \leq i \leq n \) (as in (2))

is a shorthand for the following \( k + 1 \) rules

\[ \{ l_j \} \leftarrow l_{m+1}, \ldots, l_n, L_j \quad \text{for } 1 \leq j \leq k \]

\[ \leftarrow l_{m+1}, \ldots, l_n, \sim s_1 \prec_1 \alpha \{ t_1 : l_1, L_1; \ldots; t_k : l_k, L_k \} \prec_2 s_2 \]

- Example 10 < \#\text{sum} \{ C,X,Y : \text{edge}(X,Y) : \text{cost}(X,Y,C) \}.
The expression

\[ s_1 \{ l_1 : L_1; \ldots; l_k : L_k \} \leq s_2 \]

is a shortcut for

- \[ s_1 \leq \text{count}\{ t_1 : l_1 : L_1; \ldots; t_k : l_k : L_k \} \leq s_2 \]
  if it appears in the head of a rule and
- \[ s_1 \leq \text{count}\{ t_1 : l_1, L_1; \ldots; t_k : l_k, L_k \} \leq s_2 \]
  if it appears in the body of a rule

where \( t_i \neq t_j \) whenever \( L_i \neq L_j \) for \( i \neq j \) and \( 1 \leq i, j \leq k \)

- Note one (or both) of \( s_1 \) and \( s_2 \) can be omitted
The expression
\[
\begin{align*}
  s_1 \{ l_1 : L_1 ; \ldots ; l_k : L_k \} \ s_2
\end{align*}
\]
is a shortcut for

- \( s_1 \leq \text{count}\{ t_1 : l_1 : L_1 ; \ldots ; t_k : l_k : L_k \} \leq s_2 \)
  if it appears in the head of a rule and
- \( s_1 \leq \text{count}\{ t_1 : l_1 , L_1 ; \ldots ; t_k : l_k , L_k \} \leq s_2 \)
  if it appears in the body of a rule

where \( t_i \neq t_j \) whenever \( L_i \neq L_j \) for \( i \neq j \) and \( 1 \leq i , j \leq k \)

- Note one (or both) of \( s_1 \) and \( s_2 \) can be omitted
Examples

\begin{itemize}
\item \{a; b\}
\end{itemize}

```
$ gringo --text <(echo "\{a;b\}.")
#count\{1,0,a:a;1,0,b:b\}.
```

gringo generates two distinct term tuples 1,0,a and 1,0,b

\begin{itemize}
\item 1 = \{ q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) \}
\end{itemize}
Examples

- `{a; b}`

```
$ gringo --text <(echo "\{a;b\}."))
#count\{1,0,a:a;1,0,b:b\}.

gringo generates two distinct term tuples 1,0,a and 1,0,b
```

```latex
1 = \{ q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) \}
```
Gringo language

Examples

- \{a; b\}

```bash
$ gringo --text <(echo "\{a;b\}.")
#count\{1,0,a:a;1,0,b:b\}.
```

*gringo* generates two distinct term tuples \(1,0,a\) and \(1,0,b\)

- \(1 = \{ q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) \} \)
Examples

- \{a; b\}

```
$ gringo --text <(echo "\{a;b\}.")
#count\{1,0,a:a;1,0,b:b\}.
```

*gringo* generates two distinct term tuples 1,0,a and 1,0,b

- \(1 = \{ q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) \}\)
Gringo language

Examples

- \{a; b\}

$ gringo --text <(echo "\{a;b\}.")
#count\{1,0,a:a;1,0,b:b\}.

*gringo* generates two distinct term tuples 1,0,a and 1,0,b

- 1 = \{ q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) \}
Weak constraints

Syntax A weak constraint is of the form

\[ \sim l_1, \ldots, l_n. [w@p, t_1, \ldots, t_m] \]

where

- \( l_1, \ldots, l_n \) are literals
- \( t_1, \ldots, t_m, w, \) and \( p \) are terms
- \( w \) and \( p \) stand for a weight and priority level (\( p = 0 \) if ‘@\( p \)’ is omitted)

Example The weak constraint

\[ \sim \text{hd}(I,P,C). [C@2,I] \]

amounts to the minimize statement

\[ \#\text{minimize}\{ C@2,I : \text{hd}(I,P,C) \} . \]
Weak constraints

**Syntax** A weak constraint is of the form

\[ \sim l_1, \ldots, l_n. [w@p, t_1, \ldots, t_m] \]

where

- \( l_1, \ldots, l_n \) are literals
- \( t_1, \ldots, t_m, w, \) and \( p \) are terms

- \( w \) and \( p \) stand for a weight and priority level (\( p = 0 \) if ‘@\( p \)’ is omitted)

**Example** The weak constraint

\[ \sim \text{hd}(I,P,C). [C@2,I] \]

amounts to the minimize statement

\[ \#\text{minimize}\{ C@2,I : \text{hd}(I,P,C) \}. \]
Weak constraints

Syntax A weak constraint is of the form

\[ \sim l_1, \ldots, l_n. [w@p, t_1, \ldots, t_m] \]

where
- \( l_1, \ldots, l_n \) are literals
- \( t_1, \ldots, t_m, w, \) and \( p \) are terms
- \( w \) and \( p \) stand for a weight and priority level (\( p = 0 \) if ‘@p’ is omitted)

Example The weak constraint

\[ \sim \text{hd}(I,P,C). [C@2,I] \]

amounts to the minimize statement

\[ \text{#minimize}\{ C@2,I : \text{hd}(I,P,C) \} \].

Weak constraints

- Syntax A **weak constraint** is of the form

  \[ \sim l_1, \ldots, l_n. [w@p, t_1, \ldots, t_m] \]

  where
  - \(l_1, \ldots, l_n\) are literals
  - \(t_1, \ldots, t_m, w, \) and \(p\) are terms
  - \(w\) and \(p\) stand for a weight and priority level (\(p = 0\) if ‘@p’ is omitted)

- Example The weak constraint

  \[ \sim \text{hd}(I,P,C). [C@2,I] \]

  amounts to the minimize statement

  \#minimize\{ C@2,I : \text{hd}(I,P,C) \}.
Some more directives

- **Output**

  \[ \#\text{show.} \quad \#\text{show } p/n. \quad \#\text{show } t : l_1, \ldots, l_n. \]

- **Projection**

  \[ \#\text{project } p/n. \quad \#\text{project } a : l_1, \ldots, l_n. \]

- **Heuristics**

  \[ \#\text{heuristic } a : l_1, \ldots, l_n. [k@p, m] \]

- **Acyclicity**

  \[ \#\text{edge } (u, v) : l_1, \ldots, l_n. \]
Some more directives

- **Output**
  
  ```
  #show.
  #show p/n.
  #show t : l₁,...,lₙ.
  ```

- **Projection**
  
  ```
  #project p/n.
  #project a : l₁,...,lₙ.
  ```

- **Heuristics**
  
  ```
  #heuristic a : l₁,...,lₙ. [k@p, m]
  ```

- **Acyclicity**
  
  ```
  #edge (u, v) : l₁,...,lₙ.
  ```
Some more directives

- **Output**
  
  ```
  #show.  #show p/n.  #show t : l₁,...,lₙ.
  ```

- **Projection**
  
  ```
  #project p/n.  #project a : l₁,...,lₙ.
  ```

- **Heuristics**
  
  ```
  #heuristic a : l₁,...,lₙ. [k@p, m]
  ```

- **Acyclicity**
  
  ```
  #edge (u, v) : l₁,...,lₙ.
  ```
Some more directives

- **Output**

  ```
  #show.
  #show p/n.
  #show t : l₁,...,lₙ.
  ```

- **Projection**

  ```
  #project p/n.
  #project a : l₁,...,lₙ.
  ```

- **Heuristics**

  ```
  #heuristic a : l₁,...,lₙ. [k@p, m]
  ```

- **Acyclicity**

  ```
  #edge (u, v) : l₁,...,lₙ.
  ```
**gringo 3 versus 4/5**

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lpars*e and *gringo* 3

**Example** The *gringo* 3 rule

\[
\text{r}(X) : \text{p}(X) : \text{not} \text{ q}(X) :- \text{r}(X) : \text{p}(X) : \text{not} \text{ q}(X),
\]

1 \{ \text{r}(X) : \text{p}(X) : \text{not} \text{ q}(X) \}.

can be written as follows in the language of *gringo* 4/5:

\[
\text{r}(X) : \text{p}(X), \text{not} \text{ q}(X) :- \text{r}(X) : \text{p}(X), \text{not} \text{ q}(X);
\]

**Note** Directives \#compute, \#domain, and \#hide are discontinued

**Attention**

- The languages of *gringo* 3 and 4/5 are not fully compatible
- Many example programs in the literature are written for *gringo* 3
The input language of *gringo* series 4/5 comprises

- ASP-Core-2
- concepts from *lpars* and *gringo* 3

**Example** The *gringo* 3 rule

```
r(X) : p(X) : not q(X) :- r(X) : p(X) : not q(X),
```

1 { r(X) : p(X) : not q(X) }.

can be written as follows in the language of *gringo* 4/5:

```
r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
```

**Note** Directives #compute, #domain, and #hide are discontinued

**Attention**

- The languages of *gringo* 3 and 4/5 are not fully compatible
- Many example programs in the literature are written for *gringo* 3
The input language of *gringo* series 4/5 comprises

- ASP-Core-2
- concepts from *lparse* and *gringo* 3

Example The *gringo* 3 rule

```
r(X) : p(X) : not q(X) :- r(X) : p(X) : not q(X),
1 { r(X) : p(X) : not q(X) }.
```

can be written as follows in the language of *gringo* 4/5:

```
r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
1 <= #count { 1,r(X) : r(X), p(X), not q(X) }.
```

Note Directives #compute, #domain, and #hide are discontinued

Attention

- The languages of *gringo* 3 and 4/5 are not fully compatible
- Many example programs in the literature are written for *gringo* 3
Gringo language

**gringo 3 versus 4/5**

- The input language of *gringo* series 4/5 comprises
  - ASP-Core-2
  - concepts from *lparse* and *gringo* 3
- Example The *gringo* 3 rule
  - \[ r(X) : p(X) : not q(X) :- r(X) : p(X) : not q(X), \\
  \qquad 1 \{ r(X) : p(X) : not q(X) \}. \]
  - can be written as follows in the language of *gringo* 4/5:
  - \[ r(X) : p(X), not q(X) :- r(X) : p(X), not q(X); \\
  \qquad 1 \{ r(X) : p(X), not q(X) \}. \]
- Note Directives #compute, #domain, and #hide are discontinued
- Attention
  - The languages of *gringo* 3 and 4/5 are not fully compatible
  - Many example programs in the literature are written for *gringo* 3
The input language of *gringo* series 4/5 comprises

- ASP-Core-2
- concepts from *lparse* and *gringo* 3

**Example** The *gringo* 3 rule

```
r(X) : p(X) : not q(X) :- r(X) : p(X) : not q(X),
1 { r(X) : p(X) : not q(X) }.
```

can be written as follows in the language of *gringo* 4/5:

```
r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
1 { r(X) : p(X), not q(X) }.
```

**Note** Directives `#compute`, `#domain`, and `#hide` are discontinued

**Attention**

- The languages of *gringo* 3 and 4/5 are not fully compatible
- Many example programs in the literature are written for *gringo* 3
The input language of *gringo* series 4/5 comprises
- ASP-Core-2
- concepts from *lparse* and *gringo* 3

Example The *gringo* 3 rule
- \( r(X) : p(X) : \text{not } q(X) :- r(X) : p(X) : \text{not } q(X) \),
- \( 1 \{ r(X) : p(X) : \text{not } q(X) \} \).

can be written as follows in the language of *gringo* 4/5:
- \( r(X) : p(X), \text{not } q(X) :- r(X) : p(X), \text{not } q(X) \);
- \( 1 \{ r(X) : p(X), \text{not } q(X) \} \).

Note Directives #compute, #domain, and #hide are discontinued

Attention
- The languages of *gringo* 3 and 4/5 are not fully compatible
- Many example programs in the literature are written for *gringo* 3
Computational Aspects: Overview

28 Consequence operator
29 Computation from first principles
30 Complexity
Outline

28 Consequence operator
29 Computation from first principles
30 Complexity
Let $P$ be a positive program and $X$ a set of atoms

- The consequence operator $T_P$ is defined as follows:

\[ T_PX = \{ h(r) \mid r \in P \text{ and } B(r) \subseteq X \} \]

- Iterated applications of $T_P$ are written as $T_P^j$ for $j \geq 0$, where
  - $T_P^0X = X$ and
  - $T_P^iX = T_P T_P^{i-1}X$ for $i \geq 1$

For any positive program $P$, we have

- $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
- $X \subseteq Y$ implies $T_PX \subseteq T_PY$
- $Cn(P)$ is the smallest fixpoint of $T_P$
Let $P$ be a positive program and $X$ a set of atoms

The consequence operator $T_P$ is defined as follows:

$$T_P X = \{h(r) \mid r \in P \text{ and } B(r) \subseteq X\}$$

Iterated applications of $T_P$ are written as $T_P^j$ for $j \geq 0$, where

- $T_P^0 X = X$ and
- $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

For any positive program $P$, we have

- $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
- $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
- $Cn(P)$ is the smallest fixpoint of $T_P$
Let $P$ be a positive program and $X$ a set of atoms

The consequence operator $T_P$ is defined as follows:

$$T_P X = \{ h(r) \mid r \in P \text{ and } B(r) \subseteq X \}$$

Iterated applications of $T_P$ are written as $T_P^j$ for $j \geq 0$, where

- $T_P^0 X = X$ and
- $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

For any positive program $P$, we have

- $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
- $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
- $Cn(P)$ is the smallest fixpoint of $T_P$
Consider the program

\[ P = \{ p \leftarrow, \ q \leftarrow, \ r \leftarrow p, \ s \leftarrow q, t, \ t \leftarrow r, \ u \leftarrow v \} \]

We get

\[
\begin{align*}
T^0_P \emptyset &= \emptyset \\
T^1_P \emptyset &= \{ p, q \} = T_P T^0_P \emptyset = T_P \emptyset \\
T^2_P \emptyset &= \{ p, q, r \} = T_P T^1_P \emptyset = T_P \{ p, q \} \\
T^3_P \emptyset &= \{ p, q, r, t \} = T_P T^2_P \emptyset = T_P \{ p, q, r \} \\
T^4_P \emptyset &= \{ p, q, r, t, s \} = T_P T^3_P \emptyset = T_P \{ p, q, r, t \} \\
T^5_P \emptyset &= \{ p, q, r, t, s \} = T_P T^4_P \emptyset = T_P \{ p, q, r, t, s \} \\
T^6_P \emptyset &= \{ p, q, r, t, s \} = T_P T^5_P \emptyset = T_P \{ p, q, r, t, s \}
\end{align*}
\]

\[ Cn(P) = \{ p, q, r, t, s \} \] is the smallest fixpoint of \( T_P \) because

- \( T_P \{ p, q, r, t, s \} = \{ p, q, r, t, s \} \) and
- \( T_P X \neq X \) for each \( X \subset \{ p, q, r, t, s \} \)
An example

Consider the program

\[ P = \{ p \leftarrow, \; q \leftarrow, \; r \leftarrow p, \; s \leftarrow q, t, \; t \leftarrow r, \; u \leftarrow v \} \]

We get

\[
\begin{align*}
T_P^0 \emptyset &= \emptyset \\
T_P^1 \emptyset &= \{ p, q \} = T_P T_P^0 \emptyset = T_P \emptyset \\
T_P^2 \emptyset &= \{ p, q, r \} = T_P T_P^1 \emptyset = T_P \{ p, q \} \\
T_P^3 \emptyset &= \{ p, q, r, t \} = T_P T_P^2 \emptyset = T_P \{ p, q, r \} \\
T_P^4 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^3 \emptyset = T_P \{ p, q, r, t \} \\
T_P^5 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^4 \emptyset = T_P \{ p, q, r, t, s \} \\
T_P^6 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^5 \emptyset = T_P \{ p, q, r, t, s \}
\end{align*}
\]

\[ Cn(P) = \{ p, q, r, t, s \} \] is the smallest fixpoint of \( T_P \) because

- \( T_P \{ p, q, r, t, s \} = \{ p, q, r, t, s \} \) and
- \( T_P X \neq X \) for each \( X \subseteq \{ p, q, r, t, s \} \)
Consider the program

\[ P = \{ p \leftarrow, q \leftarrow, r \leftarrow p, s \leftarrow q, t, t \leftarrow r, u \leftarrow v \} \]

We get

\[
\begin{align*}
T_P^0 \emptyset &= \emptyset \\
T_P^1 \emptyset &= \{ p, q \} = T_P T_P^0 \emptyset = T_P \emptyset \\
T_P^2 \emptyset &= \{ p, q, r \} = T_P T_P^1 \emptyset = T_P \{ p, q \} \\
T_P^3 \emptyset &= \{ p, q, r, t \} = T_P T_P^2 \emptyset = T_P \{ p, q, r \} \\
T_P^4 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^3 \emptyset = T_P \{ p, q, r, t \} \\
T_P^5 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^4 \emptyset = T_P \{ p, q, r, t, s \} \\
T_P^6 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^5 \emptyset = T_P \{ p, q, r, t, s \}
\end{align*}
\]

\[ Cn(P) = \{ p, q, r, t, s \} \] is the smallest fixpoint of \( T_P \) because

- \( T_P \{ p, q, r, t, s \} = \{ p, q, r, t, s \} \) and
- \( T_P X \neq X \) for each \( X \subset \{ p, q, r, t, s \} \)
Outline

28 Consequence operator

29 Computation from first principles

30 Complexity
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(A \setminus U)$ describe a three-valued model of the program

- **Observation**
  
  $X \subseteq Y$ implies $P^Y \subseteq P^X$ implies $\text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
  - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

First Idea

Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$

- $L$ and $U$ constitute lower and upper bounds on $X$
- $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } \text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$$

Properties

Let $X$ be a stable model of normal logic program $P$

- If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
- If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
- If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

First Idea  Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } \text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$$

Properties  Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
    - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
    - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(A \setminus U)$ describe a three-valued model of the program

- **Observation**
  \[ X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X) \]

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

First Idea  Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
- $L$ and $U$ constitute lower and upper bounds on $X$
- $L$ and $(A \setminus U)$ describe a three-valued model of the program

Observation

$X \subseteq Y$ implies $P^Y \subseteq P^X$ implies $Cn(P^Y) \subseteq Cn(P^X)$

Properties  Let $X$ be a stable model of normal logic program $P$
- If $L \subseteq X$, then $X \subseteq Cn(P^L)$
- If $X \subseteq U$, then $Cn(P^U) \subseteq X$
- If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**

  
  $$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } \text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$$

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
  - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

- First Idea
  Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- Observation

  $$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties
  Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**

\[
X \subseteq Y \text{ implies } P_Y \subseteq P_X \text{ implies } Cn(P_Y) \subseteq Cn(P_X)
\]

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq Cn(P_L)$
  - If $X \subseteq U$, then $Cn(P_U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P_U) \subseteq X \subseteq U \cap Cn(P_L)$
Approximating stable models

- First Idea: Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(A \setminus U)$ describe a three-valued model of the program

- Observation
  \[ X \subseteq Y \implies P^Y \subseteq P^X \implies \text{Cn}(P^Y) \subseteq \text{Cn}(P^X) \]

- Properties: Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
  - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

- Second Idea
  
  repeat
  
  replace $L$ by $L \cup Cn(P^U)$
  replace $U$ by $U \cap Cn(P^L)$
  until $L$ and $U$ do not change anymore

- Observations
  
  At each iteration step
  
  - $L$ becomes larger (or equal)
  - $U$ becomes smaller (or equal)
  
  $L \subseteq X \subseteq U$ is invariant for every stable model $X$ of $P$

  If $L \nsubseteq U$, then $P$ has no stable model
  If $L = U$, then $L$ is a stable model of $P$
Approximating stable models

- **Second Idea**

  ```
  repeat
  replace \( L \) by \( L \cup \text{Cn}(P^U) \)
  replace \( U \) by \( U \cap \text{Cn}(P^L) \)
  until \( L \) and \( U \) do not change anymore
  ```

- **Observations**

  - At each iteration step
    - \( L \) becomes larger (or equal)
    - \( U \) becomes smaller (or equal)
  
  - \( L \subseteq X \subseteq U \) is invariant for every stable model \( X \) of \( P \)
  
  - If \( L \nsubseteq U \), then \( P \) has no stable model
  - If \( L = U \), then \( L \) is a stable model of \( P \)
Approximating stable models

Second Idea

\[
\text{repeat} \quad \begin{align*}
\text{replace } L & \text{ by } L \cup Cn(P^U) \\
\text{replace } U & \text{ by } U \cap Cn(P^L)
\end{align*}
\text{until } L \text{ and } U \text{ do not change anymore}
\]

Observations

- At each iteration step
  - $L$ becomes larger (or equal)
  - $U$ becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every stable model $X$ of $P$
- If $L \not\subseteq U$, then $P$ has no stable model
- If $L = U$, then $L$ is a stable model of $P$
Second Idea

\[\text{repeat}\]
\[\text{replace } L \text{ by } L \cup Cn(P^U)\]
\[\text{replace } U \text{ by } U \cap Cn(P^L)\]
\[\text{until } L \text{ and } U \text{ do not change anymore}\]

Observations

- At each iteration step
  - \(L\) becomes larger (or equal)
  - \(U\) becomes smaller (or equal)
- \(L \subseteq X \subseteq U\) is invariant for every stable model \(X\) of \(P\)
- If \(L \nsubseteq U\), then \(P\) has no stable model
- If \(L = U\), then \(L\) is a stable model of \(P\)
The simplistic expand algorithm

\[
\text{expand}_P(L, U) \\
\text{repeat} \\
L' \leftarrow L \\
U' \leftarrow U \\
L \leftarrow L' \cup Cn(P^{U'}) \\
U \leftarrow U' \cap Cn(P^{L'}) \\
\text{if } L \not\subseteq U \text{ then return} \\
\text{until } L = L' \text{ and } U = U'
\]
An example

\[ P = \{ \begin{align*}
  a &\leftarrow \\
b &\leftarrow a, \neg c \\
d &\leftarrow b, \neg e \\
e &\leftarrow \neg d
\end{align*} \} \]

<table>
<thead>
<tr>
<th>(L')</th>
<th>(Cn(P^{U'}))</th>
<th>(L)</th>
<th>(U')</th>
<th>(Cn(P^{L'}))</th>
<th>(U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\emptyset)</td>
<td>({a})</td>
<td>({a})</td>
<td>({a, b, c, d, e})</td>
<td>({a, b, d, e})</td>
</tr>
<tr>
<td>2</td>
<td>({a})</td>
<td>({a, b})</td>
<td>({a, b})</td>
<td>({a, b, d, e})</td>
<td>({a, b, d, e})</td>
</tr>
<tr>
<td>3</td>
<td>({a, b})</td>
<td>({a, b})</td>
<td>({a, b})</td>
<td>({a, b, d, e})</td>
<td>({a, b, d, e})</td>
</tr>
</tbody>
</table>

Note: We have \(\{a, b\} \subseteq X\) and \((A \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset\) for every stable model \(X\) of \(P\).
Computation from first principles

An example

\[ P = \begin{cases} 
    a \leftarrow \\
    b \leftarrow a, \sim c \\
    d \leftarrow b, \sim e \\
    e \leftarrow \sim d 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>( L' )</th>
<th>( Cn(P^U') )</th>
<th>( L )</th>
<th>( U' )</th>
<th>( Cn(P^{L'}) )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>( {a} )</td>
<td>( {a} )</td>
<td>( {a, b, c, d, e} )</td>
<td>( {a, b, d, e} )</td>
<td>( {a, b, d, e} )</td>
</tr>
<tr>
<td>2</td>
<td>( {a} )</td>
<td>( {a, b} )</td>
<td>( {a, b} )</td>
<td>( {a, b, d, e} )</td>
<td>( {a, b, d, e} )</td>
<td>( {a, b, d, e} )</td>
</tr>
<tr>
<td>3</td>
<td>( {a, b} )</td>
<td>( {a, b} )</td>
<td>( {a, b} )</td>
<td>( {a, b, d, e} )</td>
<td>( {a, b, d, e} )</td>
<td>( {a, b, d, e} )</td>
</tr>
</tbody>
</table>

Note: We have \( \{a, b\} \subseteq X \) and \((A \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset\) for every stable model \( X \) of \( P \).
Computation from first principles

An example

\[ P = \begin{cases} 
    a \leftarrow \\
    b \leftarrow a, \sim c \\
    d \leftarrow b, \sim e \\
    e \leftarrow \sim d 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>( L' )</th>
<th>( Cn(P^{U'}) )</th>
<th>( L )</th>
<th>( U' )</th>
<th>( Cn(P^{L'}) )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>{a}</td>
<td>{a}</td>
<td>{a, b, c, d, e}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
<tr>
<td>2</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
<tr>
<td>3</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
</tbody>
</table>

Note: We have \( \{a, b\} \subseteq X \) and \((A \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset\) for every stable model \(X\) of \(P\).
The simplistic expand algorithm

- $\text{expand}_P$
  - tightens the approximation on stable models
  - is stable model preserving
Computation from first principles

Let’s expand with \( d \)!

\[
P = \left\{ \begin{array}{l}
a \leftarrow \\
b \leftarrow a, \sim c \\
d \leftarrow b, \sim e \\
e \leftarrow \sim d
\end{array} \right. \]

\[
\begin{array}{cccccccc}
& L' & Cn(P'^U) & L & U' & Cn(P'^L) & U \\
1 & \{d\} & \{a\} & \{a,d\} & \{a,b,c,d,e\} & \{a,b,d\} & \{a,b,d\} \\
2 & \{a,d\} & \{a,b,d\} & \{a,b,d\} & \{a,b,d\} & \{a,b,d\} & \{a,b,d\} \\
3 & \{a,b,d\} & \{a,b,d\} & \{a,b,d\} & \{a,b,d\} & \{a,b,d\} & \{a,b,d\}
\end{array}
\]

\[\text{Note} \ \{a,b,d\} \text{ is a stable model of } P\]
Let’s expand with $d$!

$$P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d
\end{cases}$$

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${d}$</td>
<td>${a}$</td>
<td>${a, d}$</td>
<td>${a, b, c, d, e}$</td>
<td>${a, b, d}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, d\}$ is a stable model of $P$
Let’s expand with \( d \)!

\[
P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d 
\end{cases}
\]

<table>
<thead>
<tr>
<th>( L' )</th>
<th>( \text{Cn}(P^{U'}) )</th>
<th>( L )</th>
<th>( U' )</th>
<th>( \text{Cn}(P^{L'}) )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {d}</td>
<td>{a}</td>
<td>{a, d}</td>
<td>{a, b, c, d, e}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
</tr>
<tr>
<td>2 {a, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
</tr>
<tr>
<td>3 {a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
</tr>
</tbody>
</table>

- Note \( \{a, b, d\} \) is a stable model of \( P \)
Let’s expand with $\sim d$!

Let $P$ be the following program:

$$P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d 
\end{cases}$$

Here is the case table:

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, e}$</td>
<td>${a, e}$</td>
<td>${a, b, c, e}$</td>
<td>${a, b, d, e}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, e\}$ is a stable model of $P$. 
Let’s expand with $\sim d$!

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \sim d \end{array} \right\}$$

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, e}$</td>
<td>${a, e}$</td>
<td>${a, b, c, e}$</td>
<td>${a, b, d, e}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, e\}$ is a stable model of $P$.
Let’s expand with $\sim d$!

$P = \{ a \leftarrow \  
  b \leftarrow a, \sim c  
  d \leftarrow b, \sim e  
  e \leftarrow \sim d \}$

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, e}$</td>
<td>${a, e}$</td>
<td>${a, b, c, e}$</td>
<td>${a, b, d, e}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, e\}$ is a stable model of $P$
A simplistic solving algorithm

\[ \text{solve}_P(L, U) \]

\[
(L, U) \leftarrow \text{expand}_P(L, U) \quad \text{// propagation}
\]

\[
\text{if } L \not\subseteq U \text{ then failure} \quad \text{// failure}
\]

\[
\text{if } L = U \text{ then output } L \quad \text{// success}
\]

\[
\text{else choose } a \in U \setminus L \quad \text{// choice}
\]

\[
\text{solve}_P(L \cup \{a\}, U)
\]

\[
\text{solve}_P(L, U \setminus \{a\})
\]
A simplistic solving algorithm

- Close to the approach taken by the ASP solver smodels, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (expand)
    - making one choice at a time by appeal to a heuristic (choose)
- Heuristic choices are made on atoms
A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (`expand`)
    - making one choice at a time by appeal to a heuristic (`choose`)
- Heuristic choices are made on atoms
A simplistic solving algorithm

- Close to the approach taken by the ASP solver smodels, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
- Heuristic choices are made on atoms
A simplistic solving algorithm

- Close to the approach taken by the ASP solver \texttt{smodels}, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (\texttt{expand})
    - making one choice at a time by appeal to a heuristic (\texttt{choose})
- Heuristic choices are made on atoms
Complexity

Outline

28 Consequence operator
29 Computation from first principles
30 Complexity
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^p_2$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^p_2$-complete
Complexity

Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta_2^P$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^P_2$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is \(\text{co-NP}\)-complete
  - Deciding whether $a$ is in a stable model of $P$ is \(\text{NP}^{\text{NP}}\)-complete

- For a disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is \(\text{co-NP}\)-complete
  - Deciding whether $a$ is in a stable model of $P$ is \(\text{NP}^{\text{NP}}\)-complete

- For a disjunctive logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is \(\text{co-NP}^{\text{NP}}\)-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is \(\Delta^P_3\)-complete

- For a propositional theory $\Phi$:
  - Deciding whether $X$ is a stable model of $\Phi$ is \(\text{co-NP}\)-complete
  - Deciding whether $a$ is in a stable model of $\Phi$ is \(\text{NP}^{\text{NP}}\)-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is co-$NP$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP^{NP}$-complete

- For a disjunctive logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is co-$NP$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP^{NP}$-complete

- For a disjunctive logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is co-$NP^{NP}$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^P_3$-complete

- For a propositional theory $\Phi$:
  - Deciding whether $X$ is a stable model of $\Phi$ is co-$NP$-complete
  - Deciding whether $a$ is in a stable model of $\Phi$ is $NP^{NP}$-complete
Axiomatic Characterization: Overview

31 Completion
32 Tightness
33 Loops and Loop Formulas
Completion

32 Tightness

33 Loops and Loop Formulas
Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

Observation Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom.

Idea The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart.
Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

Observation Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom.

Idea The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart.
Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

Observation Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom.

Idea The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart.
Let $P$ be a normal logic program

The completion $CF(P)$ of $P$ is defined as follows

$$CF(P) = \left\{ a \leftrightarrow \bigvee_{r \in P, h(r) = a} BF(B(r)) \mid a \in A(P) \right\}$$

where

$$BF(B(r)) = \bigwedge_{a \in B(r)}^+ a \land \bigwedge_{a \in B(r)}^- \neg a$$
An example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \\
b \leftarrow \sim a \\
c \leftarrow a, \sim d \\
d \leftarrow \sim c, \sim e \\
e \leftarrow b, \sim f \\
e \leftarrow e \\
\end{array} \right\} \]

\[ CF(P) = \left\{ \begin{array}{l}
a \leftrightarrow \top \\
b \leftrightarrow \neg a \\
c \leftrightarrow a \land \neg d \\
d \leftrightarrow \neg c \land \neg e \\
e \leftrightarrow (b \land \neg f) \lor e \\
f \leftrightarrow \bot \\
\end{array} \right\} \]
An example

\[ P = \{ a \leftarrow, \quad b \leftarrow \neg a, \quad c \leftarrow a, \neg d, \quad d \leftarrow \neg c, \neg e, \quad e \leftarrow b, \neg f, \quad e \leftarrow e \} \]

\[ CF(P) = \{ a \leftrightarrow \top, \quad b \leftrightarrow \neg a, \quad c \leftrightarrow a \land \neg d, \quad d \leftrightarrow \neg c \land \neg e, \quad e \leftrightarrow (b \land \neg f) \lor e, \quad f \leftrightarrow \bot \} \]
\( CF(P) \) is logically equivalent to \( \overleftarrow{CF}(P) \cup \overrightarrow{CF}(P) \), where

\[
\begin{align*}
\overleftarrow{CF}(P) & = \left\{ a \leftarrow \bigvee_{B \in B_P(a)} BF(B) \mid a \in A(P) \right\} \\
\overrightarrow{CF}(P) & = \left\{ a \rightarrow \bigvee_{B \in B_P(a)} BF(B) \mid a \in A(P) \right\}
\end{align*}
\]

\( B_P(a) = \left\{ B(r) \mid r \in P \text{ and } h(r) = a \right\} \)

\( \overleftarrow{CF}(P) \) characterizes the classical models of \( P \)

\( \overrightarrow{CF}(P) \) completes \( P \) by adding necessary conditions for all atoms
Completion

A closer look

- $\overleftarrow{CF}(P)$ is logically equivalent to $\overleftarrow{CF}(P) \cup \overrightarrow{CF}(P)$, where
  
  \[
  \overleftarrow{CF}(P) = \left\{ a \leftarrow \bigvee_{B \in B_p(a)} BF(B) \mid a \in A(P) \right\}
  \]
  \[
  \overrightarrow{CF}(P) = \left\{ a \rightarrow \bigvee_{B \in B_p(a)} BF(B) \mid a \in A(P) \right\}
  \]
  \[
  B_p(a) = \left\{ B(r) \mid r \in P \text{ and } h(r) = a \right\}
  \]

- $\overleftarrow{CF}(P)$ characterizes the classical models of $P$
- $\overrightarrow{CF}(P)$ completes $P$ by adding necessary conditions for all atoms
A closer look

\[ P = \{ \begin{align*}
  a & \leftarrow \\
  b & \leftarrow \sim a \\
  c & \leftarrow a, \sim d \\
  d & \leftarrow \sim c, \sim e \\
  e & \leftarrow b, \sim f \\
  e & \leftarrow e
\end{align*} \} \]
A closer look

\[ P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow \sim a \\
  c \leftarrow a, \sim d \\
  d \leftarrow \sim c, \sim e \\
  e \leftarrow b, \sim f \\
  e \leftarrow e 
\end{cases} \]

\[ \overleftarrow{CF}(P) = \begin{cases} 
  a \leftarrow \top \\
  b \leftarrow \neg a \\
  c \leftarrow a \land \neg d \\
  d \leftarrow \neg c \land \neg e \\
  e \leftarrow (b \land \neg f) \lor e \\
  f \leftarrow \bot 
\end{cases} \]
A closer look

\[
\overline{CF(P)} = \left\{ \begin{array}{l}
a & \leftarrow & \top \\
b & \leftarrow & \neg a \\
c & \leftarrow & a \land \neg d \\
d & \leftarrow & \neg c \land \neg e \\
e & \leftarrow & (b \land \neg f) \lor e \\
f & \leftarrow & \bot \end{array} \right\}
\]
A closer look

\[ \overleftarrow{CF}(P) = \begin{cases} 
    a & \leftarrow \top \\
    b & \leftarrow \neg a \\
    c & \leftarrow a \land \neg d \\
    d & \leftarrow \neg c \land \neg e \\
    e & \leftarrow (b \land \neg f) \lor e \\
    f & \leftarrow \bot 
\end{cases} \]

\[ \overrightarrow{CF}(P) = \begin{cases} 
    a & \rightarrow \top \\
    b & \rightarrow \neg a \\
    c & \rightarrow a \land \neg d \\
    d & \rightarrow \neg c \land \neg e \\
    e & \rightarrow (b \land \neg f) \lor e \\
    f & \rightarrow \bot 
\end{cases} \]

\[ \overrightarrow{CF}(P) = \overleftarrow{CF}(P) \]
A closer look

\[ \underset{CF}{\Rightarrow} (P) = \begin{cases} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \land \neg d \\ d \leftarrow \neg c \land \neg e \\ e \leftarrow (b \land \neg f) \lor e \\ f \leftarrow \bot \end{cases} \]

\[ \underset{\neg CF}{\Rightarrow} (P) = \begin{cases} a \rightarrow \top \\ b \rightarrow \neg a \\ c \rightarrow a \land \neg d \\ d \rightarrow \neg c \land \neg e \\ e \rightarrow (b \land \neg f) \lor e \\ f \rightarrow \bot \end{cases} \]

\[ CF (P) = \begin{cases} a \leftrightarrow \top \\ b \leftrightarrow \neg a \\ c \leftrightarrow a \land \neg d \\ d \leftrightarrow \neg c \land \neg e \\ e \leftrightarrow (b \land \neg f) \lor e \\ f \leftrightarrow \bot \end{cases} \]
A closer look

\[ \left\{ \begin{array}{l}
a \leftrightarrow \top \\
b \leftrightarrow \neg a \\
c \leftrightarrow a \land \neg d \\
d \leftrightarrow \neg c \land \neg e \\
e \leftrightarrow (b \land \neg f) \lor e \\
f \leftrightarrow \bot 
\end{array} \right. = \overline{\overline{CF(P)}} \]

\[ \left\{ \begin{array}{l}
a \rightarrow \top \\
b \rightarrow \neg a \\
c \rightarrow a \land \neg d \\
d \rightarrow \neg c \land \neg e \\
e \rightarrow (b \land \neg f) \lor e \\
f \rightarrow \bot 
\end{array} \right. \]

\[ \overline{CF(P)} = \overline{\overline{CF(P)}} \cup \overline{CF(P)} \]
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.
- In other words, every stable model of $P$ is a supported model of $P$.
- By definition, every supported model of $P$ is also a model of $P$. 
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.
- In other words, every stable model of $P$ is a supported model of $P$.
- By definition, every supported model of $P$ is also a model of $P$. 
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.

In other words, every stable model of $P$ is a supported model of $P$.

By definition, every supported model of $P$ is also a model of $P$. 

Completion
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.
- In other words, every stable model of $P$ is a supported model of $P$.
- By definition, every supported model of $P$ is also a model of $P$. 
An example

\[ P = \left\{ \begin{array}{ccc}
  a & \leftarrow & c \\
  b & \leftarrow & \sim a \\
  c & \leftarrow & a, \sim d \\
  d & \leftarrow & \sim c, \sim e \\
  e & \leftarrow & b, \sim f \\
  e & \leftarrow & e
\end{array} \right\} \]

- \( P \) has 21 models, including \( \{a, c\}, \{a, d\} \), but also \( \{a, b, c, d, e, f\} \)
- \( P \) has 3 supported models, namely \( \{a, c\}, \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has 2 stable models, namely \( \{a, c\} \) and \( \{a, d\} \)
An example

\[ P = \{ \begin{array}{ccc}
  a & \leftarrow & c \\
  b & \leftarrow & \sim a \\
  c & \leftarrow & a, \sim d \\
  d & \leftarrow & \sim c, \sim e \\
  e & \leftarrow & b, \sim f \\
  e & \leftarrow & e \\
\end{array} \} \]

- \( P \) has 21 models, including \( \{ a, c \} \), \( \{ a, d \} \), but also \( \{ a, b, c, d, e, f \} \)
- \( P \) has 3 supported models, namely \( \{ a, c \} \), \( \{ a, d \} \), and \( \{ a, c, e \} \)
- \( P \) has 2 stable models, namely \( \{ a, c \} \) and \( \{ a, d \} \)
An example

\[ P = \left\{ \begin{array}{ccc} a \leftarrow & c \leftarrow a, \neg d & e \leftarrow b, \neg f \\ b \leftarrow \neg a & d \leftarrow \neg c, \neg e & e \leftarrow e \end{array} \right\} \]

- \( P \) has 21 models, including \( \{a, c\}, \{a, d\} \), but also \( \{a, b, c, d, e, f\} \)
- \( P \) has 3 supported models, namely \( \{a, c\}, \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has 2 stable models, namely \( \{a, c\} \) and \( \{a, d\} \)
An example

\[ P = \left\{ \begin{array}{lll}
    a & \leftarrow & \quad c \leftarrow a, \sim d \\
    b & \leftarrow & \sim a \\
    d & \leftarrow & \sim c, \sim e \\
    e & \leftarrow & b, \sim f \\
    e & \leftarrow & e
\end{array} \right\} \]

- \( P \) has 21 models, including \{a, c\}, \{a, d\}, but also \{a, b, c, d, e, f\}
- \( P \) has 3 supported models, namely \{a, c\}, \{a, d\}, and \{a, c, e\}
- \( P \) has 2 stable models, namely \{a, c\} and \{a, d\}
Outline

31 Completion
32 Tightness
33 Loops and Loop Formulas
Question: What causes the mismatch between supported and stable models?

Hint: Consider the unstable yet supported model \( \{a, c, e\} \) of \( P \).

Answer: The mismatch between supported and stable models is caused by cyclic derivations.

Atoms in a stable model can be “derived” from a program in a finite number of steps.

Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps.

But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
The mismatch

□ Question What causes the mismatch between supported and stable models?

□ Hint Consider the unstable yet supported model \( \{a, c, e\} \) of \( P \)!

□ Answer The mismatch between supported and stable models is caused by cyclic derivations

- Atoms in a stable model can be "derived" from a program in a finite number of steps

- Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps

- But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model
The mismatch

Question What causes the mismatch between supported and stable models?

Hint Consider the unstable yet supported model \{a, c, e\} of $P$.

Answer The mismatch between supported and stable models is caused by cyclic derivations.

- Atoms in a stable model can be “derived” from a program in a finite number of steps.
- Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps.

Note But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
Question What causes the mismatch between supported and stable models?

Hint Consider the unstable yet supported model \( \{a, c, e\} \) of \( P \)!

Answer The mismatch between supported and stable models is caused by cyclic derivations.

- Atoms in a stable model can be “derived” from a program in a finite number of steps.
- Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps.

Note But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
The mismatch

**Question** What causes the mismatch between supported and stable models?

**Hint** Consider the unstable yet supported model \{a, c, e\} of P!

**Answer** The mismatch between supported and stable models is caused by cyclic derivations

- Atoms in a stable model can be "derived" from a program in a finite number of steps
- Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps

*Note* But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
The mismatch

- **Question** What causes the mismatch between supported and stable models?

- **Hint** Consider the unstable yet supported model \( \{a, c, e\} \) of \( P \! \) !

- **Answer** The mismatch between supported and stable models is caused by cyclic derivations
  - Atoms in a stable model can be “derived” from a program in a finite number of steps
  - Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps

  **Note** But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model
Non-cyclic derivations

Let $X$ be a stable model of normal logic program $P$

- For every atom $a \in X$, there is a finite sequence of positive rules

\[
\langle r_1, \ldots, r_n \rangle
\]

such that

1. $h(r_1) = a$
2. $B(r_i)^+ \subseteq \{ h(r_j) \mid i < j \leq n \}$ for $1 \leq i \leq n$
3. $r_i \in PX$ for $1 \leq i \leq n$

- That is, each atom of $X$ has a non-cyclic derivation from $PX$

- Example There is no finite sequence of rules providing a derivation for $e$ from $P\{a,c,e\}$
Non-cyclic derivations

Let $X$ be a stable model of normal logic program $P$

- For every atom $a \in X$, there is a finite sequence of positive rules

  $$\langle r_1, \ldots, r_n \rangle$$

  such that

  1. $h(r_1) = a$
  2. $B(r_i)^+ \subseteq \{ h(r_j) \mid i < j \leq n \}$ for $1 \leq i \leq n$
  3. $r_i \in P^X$

- That is, each atom of $X$ has a non-cyclic derivation from $P^X$

Example There is no finite sequence of rules providing a derivation for $e$ from $P\{a,c,e\}$
Non-cyclic derivations

Let $X$ be a stable model of normal logic program $P$

- For every atom $a \in X$, there is a finite sequence of positive rules $\langle r_1, \ldots, r_n \rangle$ such that
  
  1. $h(r_1) = a$
  2. $B(r_i)^+ \subseteq \{ h(r_j) \mid i < j \leq n \}$ for $1 \leq i \leq n$
  3. $r_i \in P^X$

- That is, each atom of $X$ has a non-cyclic derivation from $P^X$

- Example: There is no finite sequence of rules providing a derivation for $e$ from $P\{a, c, e\}$
Positive atom dependency graph

- The origin of (potential) circular derivations can be read off the positive atom dependency graph $G(P)$ of a logic program $P$ given by

  $$(A(P), \{(a, b) \mid r \in P, a \in B(r)^+, h(r) = b\})$$

- A logic program $P$ is called tight, if $G(P)$ is acyclic.
The origin of (potential) circular derivations can be read off the positive atom dependency graph $G(P)$ of a logic program $P$ given by

$$(A(P), \{(a, b) \mid r \in P, a \in B(r)^+, h(r) = b\})$$

A logic program $P$ is called tight, if $G(P)$ is acyclic.
Example

\[ P = \{ a \leftarrow, c \leftarrow a, \neg d \quad e \leftarrow b, \neg f \} \]

\[ G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\}) \]

- \( P \) has supported models: \( \{a, c\} \), \( \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
Example

\[ P = \{ \begin{array}{llllll}
  a & \leftarrow & c & \leftarrow & a, \neg d & e & \leftarrow & b, \neg f \\
  b & \leftarrow & \neg a & d & \leftarrow & \neg c, \neg e & e & \leftarrow & e
\end{array} \} \]

\[ G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\}) \]

- \( P \) has supported models: \( \{a, c\}, \{a, d\}, \) and \( \{a, c, e\} \)
- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
Example

\[ P = \begin{cases} 
    a & \leftarrow \ 
    c & \leftarrow a, \neg d \\
    b & \leftarrow \neg a \\
    d & \leftarrow \neg c, \neg e \\
    e & \leftarrow e \\
\end{cases} \]

\[ G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\}) \]

- \( P \) has supported models: \( \{a, c\} \), \( \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
Example

- \( P = \{ \begin{align*}
  a &\leftarrow \\
  c &\leftarrow a, \sim d \\
  e &\leftarrow b, \sim f \\
  b &\leftarrow \sim a \\
  d &\leftarrow \sim c, \sim e \\
  e &\leftarrow e
\end{align*} \} \)

- \( G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\}) \)

- \( P \) has supported models: \( \{a, c\}, \{a, d\}, \) and \( \{a, c, e\} \)

- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
Tight programs

- A logic program $P$ is called tight, if $G(P)$ is acyclic

- For tight programs, stable and supported models coincide:

Fages' Theorem

Let $P$ be a tight normal logic program and $X \subseteq A(P)$
Then, $X$ is a stable model of $P$ iff $X \models CF(P)$
Tight programs

- A logic program $P$ is called tight, if $G(P)$ is acyclic.
- For tight programs, stable and supported models coincide.

Fages' Theorem

Let $P$ be a tight normal logic program and $X \subseteq A(P)$.
Then, $X$ is a stable model of $P$ iff $X \models CF(P)$.
Tight programs

- A logic program $P$ is called tight, if $G(P)$ is acyclic.
- For tight programs, stable and supported models coincide:

Fages’ Theorem

Let $P$ be a tight normal logic program and $X \subseteq A(P)$.
Then, $X$ is a stable model of $P$ iff $X \models CF(P)$. 
Another example

\[ P = \{ \begin{align*}
a & \leftarrow \lnot b \\
c & \leftarrow a, b \\
d & \leftarrow a \\
e & \leftarrow \lnot a, \lnot b \\
b & \leftarrow \lnot a \\
c & \leftarrow d \\
d & \leftarrow b, c
\end{align*} \} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \( \{a, c, d\}, \{b\} \), and \( \{b, c, d\} \)
- \( P \) has stable models: \( \{a, c, d\} \) and \( \{b\} \)
Another example

\[ P = \{ \begin{align*}
& a \leftarrow \neg b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \neg a, \neg b \\
& b \leftarrow \neg a & c \leftarrow d & d \leftarrow b, c
\end{align*} \} \]

\[ G(P) = (\{ a, b, c, d, e \}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \( \{a, c, d\}, \{b\}, \) and \( \{b, c, d\} \)
- \( P \) has stable models: \( \{a, c, d\} \) and \( \{b\} \)
Another example

\[ P = \left\{ \begin{array}{l}
    a \leftarrow \sim b \\
    c \leftarrow a, b \\
    d \leftarrow a \\
    e \leftarrow \sim a, \sim b \\
    b \leftarrow \sim a \\
    c \leftarrow d \\
    d \leftarrow b, c
\end{array} \right\} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \{a, c, d\}, \{b\}, and \{b, c, d\}
- \( P \) has stable models: \{a, c, d\} and \{b\}
Another example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \sim b \\
e \leftarrow \sim a, \sim b \\
b \leftarrow \sim a \\
c \leftarrow d \\
d \leftarrow b, c
\end{array} \right. \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \( \{a, c, d\} \), \( \{b\} \), and \( \{b, c, d\} \)
- \( P \) has stable models: \( \{a, c, d\} \) and \( \{b\} \)
Outline

31 Completion
32 Tightness
33 Loops and Loop Formulas
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

- **Idea** Add formulas prohibiting circular support of sets of atoms.

- **Note** Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

- **Idea** Add formulas prohibiting circular support of sets of atoms.

- **Note** Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Motivation

Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

Observation Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

Idea Add formulas prohibiting circular support of sets of atoms.

Note Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

- **Idea** Add formulas prohibiting circular support of sets of atoms.

- **Note** Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Let $P$ be a normal logic program, and let $G(P) = (A(P), E)$ be the positive atom dependency graph of $P$.

- A set $\emptyset \subset L \subseteq A(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$.
  That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

- We denote the set of all loops of $P$ by $\text{loop}(P)$.

- Note: A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 
Loops

Let \( P \) be a normal logic program, and let \( G(P) = (A(P), E) \) be the positive atom dependency graph of \( P \).

- A set \( \emptyset \subset L \subseteq A(P) \) is a loop of \( P \) if it induces a non-trivial strongly connected subgraph of \( G(P) \).
  
  That is, each pair of atoms in \( L \) is connected by a path of non-zero length in \( (L, E \cap (L \times L)) \).

- We denote the set of all loops of \( P \) by \( \text{loop}(P) \).

- Note: A program \( P \) is tight iff \( \text{loop}(P) = \emptyset \).
Loops

Let $P$ be a normal logic program, and let $G(P) = (A(P), E)$ be the positive atom dependency graph of $P$. A set $\emptyset \subset L \subseteq A(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$. That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

- We denote the set of all loops of $P$ by $\text{loop}(P)$.
- Note A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 

Torsten Schaub (KRR@UP)
Loops

Let $P$ be a normal logic program, and let $G(P) = (A(P), E)$ be the positive atom dependency graph of $P$.

A set $\emptyset \subset L \subseteq A(P)$ is a loop of $P$, if it induces a non-trivial strongly connected subgraph of $G(P)$. That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

We denote the set of all loops of $P$ by $\text{loop}(P)$.

Note: A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 
Loops

Let $P$ be a normal logic program, and let $G(P) = (A(P), E)$ be the positive atom dependency graph of $P$

- A set $\emptyset \subset L \subseteq A(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$.
  That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$

- We denote the set of all loops of $P$ by $\text{loop}(P)$

- Note: A program $P$ is tight iff $\text{loop}(P) = \emptyset$
Example

Let $P = \begin{cases} a \leftarrow & c \leftarrow a, \neg d \\
 b \leftarrow \neg a & d \leftarrow \neg c, \neg e \\
 & e \leftarrow b, \neg f \\
 & e \leftarrow e \end{cases}$

Then $\text{loop}(P) = \{\{e\}\}$
Loops and Loop Formulas

Example

\[ P = \left\{ \begin{array}{ll}
  a & \leftarrow c \\
  b & \leftarrow \neg a \\
  c & \leftarrow a, \neg d \\
  d & \leftarrow \neg c, \neg e \\
  e & \leftarrow b, \neg f \\
  e & \leftarrow e
  \end{array} \right\} \]

\[ \text{loop}(P) = \{ \{ e \} \} \]
Another example

$P = \begin{cases} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c \end{cases}$

$\text{loop}(P) = \{\{c, d\}\}$
Loops and Loop Formulas

Another example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \neg b \\
  c \leftarrow a, b \\
  d \leftarrow a \\
  e \leftarrow \neg a, \neg b \\
  b \leftarrow \neg a \\
  c \leftarrow d \\
  d \leftarrow b, c
\end{array} \right\} \]

\[ \text{loop}(P) = \{\{c, d\}\} \]
Yet another example

\[ P = \{ a \leftarrow \sim b, \ c \leftarrow a, \ d \leftarrow b, c, \ e \leftarrow b, \sim a, \ b \leftarrow \sim a, \ c \leftarrow b, d, \ d \leftarrow e, \ e \leftarrow c, d \} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]
Yet another example

\[ P = \left\{ \begin{array}{l}
ap \leftarrow \neg b \\
c \leftarrow a \\
d \leftarrow b, c \\
e \leftarrow b, \neg a \\
b \leftarrow \neg a \\
c \leftarrow b, d \\
d \leftarrow e \\
e \leftarrow c, d \\
\end{array} \right\} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]
Yet another example

\[ P = \begin{cases} 
  a \leftarrow \lnot b & c \leftarrow a \\
  d \leftarrow b, c & e \leftarrow b, \lnot a \\
  b \leftarrow \lnot a & c \leftarrow b, d \\
  d \leftarrow e & e \leftarrow c, d 
\end{cases} \]

\[ \text{loop}(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\} \]
Loops and Loop Formulas

Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq A(P)$, define the external supports of $L$ for $P$ as
  
  $$ES_P(L) = \{ r \in P \mid h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = B(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is
  
  $$LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))$$
  
  $$\leftrightarrow (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)$$

- Note The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) \mid L \in loop(P) \}$
Loops and Loop Formulas

Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq A(P)$, define the external supports of $L$ for $P$ as

$$ES_P(L) = \{ r \in P | h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = B(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is

$$LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))$$

$$\leftrightarrow \left(\bigwedge_{B \in EB_P(L)} \neg BF(B) \right) \rightarrow \left(\bigwedge_{a \in L} \neg a \right)$$

- Note The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) | L \in \text{loop}(P) \}$
Loops and Loop Formulas

Loops and Loop Formulas

Let \( P \) be a normal logic program

- For \( L \subseteq A(P) \), define the external supports of \( L \) for \( P \) as
  \[
  ES_P(L) = \{ r \in P \mid h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \}
  \]

- Define the external bodies of \( L \) in \( P \) as \( EB_P(L) = B(ES_P(L)) \)

- The (disjunctive) loop formula of \( L \) for \( P \) is
  \[
  LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))
  \]
  \[
  \leftrightarrow (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)
  \]

- Note The loop formula of \( L \) enforces all atoms in \( L \) to be false whenever \( L \) is not externally supported

- Define \( LF(P) = \{ LF_P(L) \mid L \in \text{loop}(P) \} \)
Loops and Loop Formulas

Let $P$ be a normal logic program

- For $L \subseteq A(P)$, define the external supports of $L$ for $P$ as
  \[ ES_P(L) = \{ r \in P \mid h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \} \]

- Define the external bodies of $L$ in $P$ as $EB_P(L) = B(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is
  \[ LF_P(L) = \left( \bigvee_{a \in L} a \right) \rightarrow \left( \bigvee_{B \in EB_P(L)} BF(B) \right) \]
  \[ \leftrightarrow \left( \bigwedge_{B \in EB_P(L)} \neg BF(B) \right) \rightarrow \left( \bigwedge_{a \in L} \neg a \right) \]

- Note: The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) \mid L \in \text{loop}(P) \}$
Example

\[ P = \{ a \leftarrow \quad c \leftarrow a, \neg d \\
       b \leftarrow \neg a \\
       d \leftarrow \neg c, \neg e \\
       e \leftarrow e \} \]

\[ \text{loop}(P) = \{ \{e\} \} \]

\[ \text{LF}(P) = \{ e \rightarrow b \land \neg f \} \]
Example

\[ P = \{ \begin{align*}
  a & \leftarrow c \\
  c & \leftarrow a, \sim d \\
  b & \leftarrow \sim a \\
  d & \leftarrow \sim c, \sim e \\
  e & \leftarrow b, \sim f \\
  e & \leftarrow e
\end{align*} \} \]

\[ \text{loop}(P) = \{ \{ e \} \} \]
\[ \text{LF}(P) = \{ e \rightarrow b \land \neg f \} \]
Another example

\[ P = \{ \quad a \leftarrow \sim b \quad c \leftarrow a, b \quad d \leftarrow a \quad e \leftarrow \sim a, \sim b \quad b \leftarrow \sim a \quad c \leftarrow d \quad d \leftarrow b, c \quad \} \]

\[ \text{loop}(P) = \{ \{ c, d \} \} \]

\[ \text{LF}(P) = \{ c \lor d \rightarrow (a \land b) \lor a \} \]
Another example

\[ P = \{ a \leftarrow \neg b, c \leftarrow a, b, d \leftarrow a, e \leftarrow \neg a, \neg b, b \leftarrow \neg a, c \leftarrow d, d \leftarrow b, c \} \]

- \( \text{loop}(P) = \{\{c, d\}\} \)
- \( \text{LF}(P) = \{c \lor d \rightarrow (a \land b) \lor a\} \)
Yet another example

\[ P = \{ \begin{array}{l} a \leftarrow \sim b \quad c \leftarrow a \quad d \leftarrow b, c \quad e \leftarrow b, \sim a \\ b \leftarrow \sim a \quad c \leftarrow b, d \quad d \leftarrow e \quad e \leftarrow c, d \end{array} \} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\}\} \]

\[ \text{LF}(P) = \{ \begin{array}{l} c \lor d \rightarrow a \lor e \\ d \lor e \rightarrow (b \land c) \lor (b \land \sim a) \\ c \lor d \lor e \rightarrow a \lor (b \land \sim a) \end{array} \} \]
Yet another example

\[ P = \{ a \leftarrow \sim b \quad c \leftarrow a \quad d \leftarrow b, c \quad e \leftarrow b, \sim a \\
     b \leftarrow \sim a \quad c \leftarrow b, d \quad d \leftarrow e \quad e \leftarrow c, d \} \]

\[ \text{loop}(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\} \]

\[ \text{LF}(P) = \begin{cases} 
c \lor d \rightarrow a \lor e \\
d \lor e \rightarrow (b \land c) \lor (b \land \sim a) \\
c \lor d \lor e \rightarrow a \lor (b \land \sim a) \end{cases} \]
Yet another example

\[ P = \{ a \leftarrow \neg b, \; c \leftarrow a, \; d \leftarrow b, c, \; e \leftarrow b, \neg a, \; b \leftarrow \neg a, \; c \leftarrow b, d, \; d \leftarrow e, \; e \leftarrow c, d \} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]

\[ \text{LF}(P) = \begin{cases} 
    c \lor d \rightarrow a \lor e \\
    d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\
    c \lor d \lor e \rightarrow a \lor (b \land \neg a)
\end{cases} \]
Yet another example

\[ P = \{ \begin{array}{l}
  a \leftarrow \sim b \\
  c \leftarrow a \\
  d \leftarrow b, c \\
  e \leftarrow b, \sim a \\
  b \leftarrow \sim a \\
  c \leftarrow b, d \\
  d \leftarrow e \\
  e \leftarrow c, d \\
\end{array} \} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]

\[ \text{LF}(P) = \{ \begin{array}{l}
  c \lor d \rightarrow a \lor e \\
  d \lor e \rightarrow (b \land c) \lor (b \land \sim a) \\
  c \lor d \lor e \rightarrow a \lor (b \land \sim a) \\
\end{array} \} \]
Yet another example

\[ P = \{ \begin{array}{llllll}
  a & \leftarrow & \sim b & c & \leftarrow & a \\
  b & \leftarrow & \sim a & c & \leftarrow & b, d \\
  d & \leftarrow & b, c & e & \leftarrow & b, \sim a \\
  b & \leftarrow & \sim a & c & \leftarrow & b, d \\
  d & \leftarrow & e & e & \leftarrow & c, d \\
 \end{array} \} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]

\[ \text{LF}(P) = \{ \begin{array}{ll}
  c \lor d \rightarrow a \lor e \\
  d \lor e \rightarrow (b \land c) \lor (b \land \sim a) \\
  c \lor d \lor e \rightarrow a \lor (b \land \sim a) \\
\end{array} \} \]
Theorem

Let $P$ be a normal logic program and $X \subseteq A(P)$.

Then, $X$ is a stable model of $P$ iff $X \models CF(P) \cup LF(P)$. 

Lin-Zhao Theorem
Loops and loop formulas: Properties

Let $X$ be a supported model of normal logic program $P$.

- Then, $X$ is a stable model of $P$ iff
  - $X \models \{LF_P(U) \mid U \subseteq A(P)\}$
  - $X \models \{LF_P(U) \mid U \subseteq X\}$
  - $X \models \{LF_P(L) \mid L \in \text{loop}(P)\}$, that is, $X \models LF(P)$
  - $X \models \{LF_P(L) \mid L \in \text{loop}(P) \text{ and } L \subseteq X\}$

- Note: If $X$ is not a stable model of $P$, then there is a loop $L \subseteq X \setminus \text{Cn}(P^X)$ such that $X \not\models LF_P(L)$.
Loops and loop formulas: Properties

Let $X$ be a supported model of normal logic program $P$

- Then, $X$ is a stable model of $P$ iff
  - $X \models \{ \text{LF}_P(U) \mid U \subseteq A(P) \}$
  - $X \models \{ \text{LF}_P(U) \mid U \subseteq X \}$
  - $X \models \{ \text{LF}_P(L) \mid L \in \text{loop}(P) \}$, that is, $X \models \text{LF}(P)$
  - $X \models \{ \text{LF}_P(L) \mid L \in \text{loop}(P) \text{ and } L \subseteq X \}$

- Note: If $X$ is not a stable model of $P$, then there is a loop $L \subseteq X \setminus \text{Cn}(P^X)$ such that $X \not\models \text{LF}_P(L)$
Loops and loop formulas: Properties

Let $X$ be a supported model of normal logic program $P$

- Then, $X$ is a stable model of $P$ iff
  - $X \models \{LF_P(U) \mid U \subseteq A(P)\}$
  - $X \models \{LF_P(U) \mid U \subseteq X\}$
  - $X \models \{LF_P(L) \mid L \in loop(P)\}$, that is, $X \models LF(P)$
  - $X \models \{LF_P(L) \mid L \in loop(P) \text{ and } L \subseteq X\}$

- Note If $X$ is not a stable model of $P$, then there is a loop $L \subseteq X \setminus Cn(P^X)$ such that $X \not\models LF_P(L)$
Loops and loop formulas: Properties (ctd)

- **Result** If $\mathcal{P} \not\subseteq \mathcal{NC}^{1/poly}$, then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program $P$, both of the following conditions hold:
  1. The propositional variables in $\mathcal{T}[P]$ are a subset of $A(P)$
  2. The size of $\mathcal{T}[P]$ is polynomial in the size of $P$

- **Note** Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case)

- **Observations**
  - Translation $\mathcal{CF}(P) \cup \mathcal{LF}(P)$ preserves the vocabulary of $P$
  - The number of loops in $\text{loop}(P)$ may be exponential in $|A(P)|$

---

3 A conjecture from complexity theory that is believed to be true
Loops and loop formulas: Properties (ctd)

- **Result** If $\mathcal{P} \not\subset \mathcal{NC}^1/poly$,\(^3\) then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program $P$, both of the following conditions hold:
  1. The propositional variables in $\mathcal{T}[P]$ are a subset of $A(P)$
  2. The size of $\mathcal{T}[P]$ is polynomial in the size of $P$

- **Note** Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case)

- **Observations**
  - Translation $CF(P) \cup LF(P)$ preserves the vocabulary of $P$
  - The number of loops in $\text{loop}(P)$ may be exponential in $|A(P)|$

---

\(^3\)A conjecture from complexity theory that is believed to be true
Loops and loop formulas: Properties (ctd)

- **Result** If $\mathcal{P} \not\subseteq NC^1/poly$, then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program $P$, both of the following conditions hold:
  1. The propositional variables in $\mathcal{T}[P]$ are a subset of $A(P)$
  2. The size of $\mathcal{T}[P]$ is polynomial in the size of $P$

- **Note** Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case)

- **Observations**
  - Translation $CF(P) \cup LF(P)$ preserves the vocabulary of $P$
  - The number of loops in $\text{loop}(P)$ may be exponential in $|A(P)|$

---

3 A conjecture from complexity theory that is believed to be true
Operational Characterization: Overview

34 Partial Interpretations
35 Fitting Operator
36 Unfounded Sets
37 Well-Founded Operator
Outline

34 Partial Interpretations

35 Fitting Operator

36 Unfounded Sets

37 Well-Founded Operator
Partial Interpretations

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation maps atoms onto truth values true, false, and unknown

- Representation \( \langle T, F \rangle \), where
  - \( T \) is the set of all true atoms and
  - \( F \) is the set of all false atoms
  - Truth of atoms in \( A \setminus (T \cup F) \) is unknown

- Properties
  - \( \langle T, F \rangle \) is conflicting if \( T \cap F \neq \emptyset \)
  - \( \langle T, F \rangle \) is total if \( T \cup F = A \) and \( T \cap F = \emptyset \)

- Definition For \( \langle T_1, F_1 \rangle \) and \( \langle T_2, F_2 \rangle \), define
  - \( \langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle \) iff \( T_1 \subseteq T_2 \) and \( F_1 \subseteq F_2 \)
  - \( \langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle \)
Interlude: Partial interpretations
or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*

- **Representation** \( \langle T, F \rangle \), where
  - \( T \) is the set of all *true* atoms and
  - \( F \) is the set of all *false* atoms
  - Truth of atoms in \( A \setminus (T \cup F) \) is *unknown*

- **Properties**
  - \( \langle T, F \rangle \) is conflicting if \( T \cap F \neq \emptyset \)
  - \( \langle T, F \rangle \) is total if \( T \cup F = A \) and \( T \cap F = \emptyset \)

- **Definition** For \( \langle T_1, F_1 \rangle \) and \( \langle T_2, F_2 \rangle \), define
  - \( \langle T_1, F_1 \rangle \subseteq \langle T_2, F_2 \rangle \) iff \( T_1 \subseteq T_2 \) and \( F_1 \subseteq F_2 \)
  - \( \langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle \)
Interlude: Partial interpretations
or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*.

- **Representation** \( \langle T, F \rangle \), where
  - \( T \) is the set of all *true* atoms and
  - \( F \) is the set of all *false* atoms
  - Truth of atoms in \( A \setminus (T \cup F) \) is *unknown*

- **Properties**
  - \( \langle T, F \rangle \) is **conflicting** if \( T \cap F \neq \emptyset \)
  - \( \langle T, F \rangle \) is **total** if \( T \cup F = A \) and \( T \cap F = \emptyset \)

- **Definition** For \( \langle T_1, F_1 \rangle \) and \( \langle T_2, F_2 \rangle \), define
  - \( \langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle \) iff \( T_1 \subseteq T_2 \) and \( F_1 \subseteq F_2 \)
  - \( \langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle \)
Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*

- **Representation** \( \langle T, F \rangle \), where
  - \( T \) is the set of all *true* atoms and
  - \( F \) is the set of all *false* atoms
  - Truth of atoms in \( A \setminus (T \cup F) \) is *unknown*

- **Properties**
  - \( \langle T, F \rangle \) is **conflicting** if \( T \cap F \neq \emptyset \)
  - \( \langle T, F \rangle \) is **total** if \( T \cup F = A \) and \( T \cap F = \emptyset \)

- **Definition** For \( \langle T_1, F_1 \rangle \) and \( \langle T_2, F_2 \rangle \), define
  - \( \langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle \) iff \( T_1 \subseteq T_2 \) and \( F_1 \subseteq F_2 \)
  - \( \langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle \)
Outline

34 Partial Interpretations
35 Fitting Operator
36 Unfounded Sets
37 Well-Founded Operator
Basic idea

- **Idea**: Extend $T_P$ to normal logic programs

- **Logical background**: The idea is to turn a program’s completion into an operator such that
  - the head atom of a rule must be *true* if the rule’s body is *true*
  - an atom must be *false* if the body of each rule having it as head is *false*
Fitting Operator

Definition

Let $P$ be a normal logic program

Define

$$
\Phi_P \langle T, F \rangle = \langle T_P \langle T, F \rangle, F_P \langle T, F \rangle \rangle
$$

where

$$
T_P \langle T, F \rangle = \{ h(r) \mid r \in P, B(r)^+ \subseteq T, B(r)^- \subseteq F \}$$

$$
F_P \langle T, F \rangle = \{ a \in A(P) \mid B(r)^+ \cap F \neq \emptyset \text{ or } B(r)^- \cap T \neq \emptyset \text{ for each } r \in P \text{ such that } h(r) = a \}$$
Let $P$ be a normal logic program

Define

$$
\Phi_P(T, F) = \langle T_P(T, F), F_P(T, F) \rangle
$$

where

$$
T_P(T, F) = \{ h(r) \mid r \in P, B(r)^+ \subseteq T, B(r)^- \subseteq F \} 
$$

$$
F_P(T, F) = \{ a \in A(P) \mid 
B(r)^+ \cap F \neq \emptyset \text{ or } B(r)^- \cap T \neq \emptyset 
$$

for each $r \in P$ such that $h(r) = a \}$$
Fitting Operator

Example

Let's iterate $\Phi_P$ on $\langle \{a\}, \{d\} \rangle$:

- $\Phi_P(\langle \{a\}, \{d\} \rangle) = \langle \{a, c\}, \{b, f\} \rangle$
- $\Phi_P(\langle \{a, c\}, \{b, f\} \rangle) = \langle \{a\}, \{b, d, f\} \rangle$
- $\Phi_P(\langle \{a\}, \{b, d, f\} \rangle) = \langle \{a, c\}, \{b, f\} \rangle$
- ...
Example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \\
  b \leftarrow \sim a \\
  c \leftarrow a, \sim d \\
  d \leftarrow \sim c, \sim e \\
  e \leftarrow b, \sim f \\
  e \leftarrow e
\end{array} \right\} \]

Let's iterate \( \Phi_P \) on \( \langle \{a\}, \{d\} \rangle \):

\[
\begin{align*}
\Phi_P\langle \{a\}, \{d\} \rangle &= \langle \{a, c\}, \{b, f\} \rangle \\
\Phi_P\langle \{a, c\}, \{b, f\} \rangle &= \langle \{a\}, \{b, d, f\} \rangle \\
\Phi_P\langle \{a\}, \{b, d, f\} \rangle &= \langle \{a, c\}, \{b, f\} \rangle \\
\vdots
\end{align*}
\]
Define the iterative variant of $\Phi_P$ analogously to $T_P$:

$$
\Phi_P^0 \langle T, F \rangle = \langle T, F \rangle \\
\Phi_P^{i+1} \langle T, F \rangle = \Phi_P \Phi_P^i \langle T, F \rangle
$$

Define the Fitting semantics of a normal logic program $P$ as the partial interpretation:

$$
\bigcup_{i \geq 0} \Phi_P^i \langle \emptyset, \emptyset \rangle
$$
Define the iterative variant of $\Phi_P$ analogously to $T_P$:

$$\Phi_P^0 \langle T, F \rangle = \langle T, F \rangle \quad \Phi_P^{i+1} \langle T, F \rangle = \Phi_P \Phi_P^i \langle T, F \rangle$$

Define the Fitting semantics of a normal logic program $P$ as the partial interpretation:

$$\bigcup_{i \geq 0} \Phi_P^i \langle \emptyset, \emptyset \rangle$$
Fitting Operator

Example

\[ P = \left\{ \begin{array}{lll}
a & \leftarrow & c \\
       & \leftarrow & a,
       & \sim & d \\
b & \leftarrow & \sim a \\
d & \leftarrow & \sim c,
       & \sim & e \\
e & \leftarrow & b,
       & \sim & f \\
e & \leftarrow & e 
\end{array} \right\} \]

\[
\Phi^0 \langle \emptyset, \emptyset \rangle = \langle \emptyset, \emptyset \rangle
\]
\[
\Phi^1 \langle \emptyset, \emptyset \rangle = \Phi \langle \emptyset, \emptyset \rangle = \langle \{a\}, \{f\} \rangle
\]
\[
\Phi^2 \langle \emptyset, \emptyset \rangle = \Phi \langle \{a\}, \{f\} \rangle = \langle \{a\}, \{b, f\} \rangle
\]
\[
\Phi^3 \langle \emptyset, \emptyset \rangle = \Phi \langle \{a\}, \{b, f\} \rangle = \langle \{a\}, \{b, f\} \rangle
\]
\[
\bigcup_{i \geq 0} \Phi^i \langle \emptyset, \emptyset \rangle = \langle \{a\}, \{b, f\} \rangle
\]
Fitting Operator

Example

\[ P = \left\{ a \leftarrow c \leftarrow a, \sim d \quad e \leftarrow b, \sim f \right. \right\} \]

\[ \Phi^0 \langle \emptyset, \emptyset \rangle = \langle \emptyset, \emptyset \rangle \]
\[ \Phi^1 \langle \emptyset, \emptyset \rangle = \Phi \langle \emptyset, \emptyset \rangle = \langle \{a\}, \{f\} \rangle \]
\[ \Phi^2 \langle \emptyset, \emptyset \rangle = \Phi \langle \{a\}, \{f\} \rangle = \langle \{a\}, \{b, f\} \rangle \]
\[ \Phi^3 \langle \emptyset, \emptyset \rangle = \Phi \langle \{a\}, \{b, f\} \rangle = \langle \{a\}, \{b, f\} \rangle \]

\[ \bigsqcup_{i \geq 0} \Phi^i \langle \emptyset, \emptyset \rangle = \langle \{a\}, \{b, f\} \rangle \]
Let $P$ be a normal logic program

- $\Phi_P^{\langle \emptyset, \emptyset \rangle}$ is monotonic
  That is, $\Phi_P^i \langle \emptyset, \emptyset \rangle \subseteq \Phi_P^{i+1} \langle \emptyset, \emptyset \rangle$

- The Fitting semantics of $P$ is
  - not conflicting,
  - and generally not total
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Define $\langle T, F \rangle$ as a **Fitting fixpoint** of $P$ if $\Phi_P\langle T, F \rangle = \langle T, F \rangle$.

- The Fitting semantics is the $\sqsubseteq$-least Fitting fixpoint of $P$.
- Any other Fitting fixpoint extends the Fitting semantics.
- Total Fitting fixpoints correspond to supported models.
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Define $\langle T, F \rangle$ as a **Fitting fixpoint** of $P$ if $\Phi_P \langle T, F \rangle = \langle T, F \rangle$.

- The Fitting semantics is the $\sqsubseteq$-least Fitting fixpoint of $P$.
- Any other Fitting fixpoint extends the Fitting semantics.
- Total Fitting fixpoints correspond to supported models.
Fitting Operator

Example

\[
P = \left\{ \begin{array}{lll}
  a & \leftarrow & a, \sim d \\
  c & \leftarrow & a, \sim d \\
  b & \leftarrow & \sim a \\
  d & \leftarrow & \sim c, \sim e \\
  e & \leftarrow & b, \sim f \\
  e & \leftarrow & e
\end{array} \right\}
\]

- \( P \) has three total Fitting fixpoints:
  - \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  - \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)
  - \( \langle \{a, c, e\}, \{b, d, f\} \rangle \)

- \( P \) has three supported models, two of them are stable models
Example

\[ P = \begin{cases} 
  a \leftarrow & & c \leftarrow a, \neg d & & e \leftarrow b, \neg f \\
  b \leftarrow \neg a & & d \leftarrow \neg c, \neg e & & e \leftarrow e 
\end{cases} \]

- \( P \) has three total Fitting fixpoints:
  1. \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  2. \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)
  3. \( \langle \{a, c, e\}, \{b, d, f\} \rangle \)

- \( P \) has three supported models, two of them are stable models
Example

\[ P = \left\{ \begin{array}{lll}
a & \leftarrow & a, \lnot d \\
b & \leftarrow & \lnot a \\
c & \leftarrow & a, \lnot d \\
d & \leftarrow & \lnot c, \lnot e \\
e & \leftarrow & b, \lnot f \\
e & \leftarrow & e \\
\end{array} \right\} \]

- \( P \) has three total Fitting fixpoints:
  1. \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  2. \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)
  3. \( \langle \{a, c, e\}, \{b, d, f\} \rangle \)

- \( P \) has three supported models, two of them are stable models.
Fitting Operator

Example

\[
P = \begin{cases} 
  a \leftarrow \quad c \leftarrow a, \neg d \\
  b \leftarrow \neg a \\
  d \leftarrow \neg c, \neg e \\
  e \leftarrow b, \neg f \\
\end{cases}
\]

- \( P \) has three total Fitting fixpoints:
  1. \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  2. \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)
  3. \( \langle \{a, c, e\}, \{b, d, f\} \rangle \)

- \( P \) has three supported models, two of them are stable models
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\Phi_P \langle T, F \rangle = \langle T', F' \rangle$.
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$.

That is, $\Phi_P$ is stable model preserving.

Hence, $\Phi_P$ can be used for approximating stable models and so for propagation in ASP-solvers.

However, $\Phi_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models.

Note: The problem is the same as with program completion.

The missing piece is non-circularity of derivations!
Properties

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation

- Let $\Phi_P \langle T, F \rangle = \langle T', F' \rangle$

- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

That is, $\Phi_P$ is stable model preserving

Hence, $\Phi_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\Phi_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

Note The problem is the same as with program completion

The missing piece is non-circularity of derivations!
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\Phi_P \langle T, F \rangle = \langle T', F' \rangle$.
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$.

That is, $\Phi_P$ is stable model preserving.

Hence, $\Phi_P$ can be used for approximating stable models and so for propagation in ASP-solvers.

However, $\Phi_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models.

Note The problem is the same as with program completion.

The missing piece is non-circularity of derivations.
Properties

Let \( P \) be a normal logic program, and let \( \langle T, F \rangle \) be a partial interpretation.

- Let \( \Phi_P(T, F) = \langle T', F' \rangle \).
- If \( X \) is a stable model of \( P \) such that \( T \subseteq X \) and \( X \cap F = \emptyset \), then \( T' \subseteq X \) and \( X \cap F' = \emptyset \).

That is, \( \Phi_P \) is stable model preserving.

Hence, \( \Phi_P \) can be used for approximating stable models and so for propagation in ASP-solvers.

However, \( \Phi_P \) is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models.

Note: The problem is the same as with program completion.

The missing piece is non-circularity of derivations!
Fitting Operator

Properties

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\Phi_P(\langle T, F \rangle) = \langle T', F' \rangle$.
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$.

That is, $\Phi_P$ is stable model preserving.

Hence, $\Phi_P$ can be used for approximating stable models and so for propagation in ASP-solvers.

- However, $\Phi_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models.

Note: The problem is the same as with program completion.

The missing piece is non-circularity of derivations!
Fitting Operator

Properties

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation

- Let $\Phi_P \langle T, F \rangle = \langle T', F' \rangle$
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$
  That is, $\Phi_P$ is stable model preserving
  Hence, $\Phi_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\Phi_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models
  Note The problem is the same as with program completion
  The missing piece is non-circularity of derivations!
Example

\[ P = \left\{ \begin{array}{c} a \leftarrow b \\ b \leftarrow a \end{array} \right\} \]

\[ \Phi^0_P(\emptyset,\emptyset) = (\emptyset,\emptyset) \]
\[ \Phi^1_P(\emptyset,\emptyset) = (\emptyset,\emptyset) \]

That is, Fitting semantics cannot assign \textit{false} to \(a\) and \(b\), although they can never become \textit{true}!
\[ P = \{ \begin{array}{cc} a & \leftarrow b \\ b & \leftarrow a \end{array} \} \]

\[ \Phi^0_P(\emptyset, \emptyset) = (\emptyset, \emptyset) \]
\[ \Phi^1_P(\emptyset, \emptyset) = (\emptyset, \emptyset) \]

That is, Fitting semantics cannot assign \emph{false} to \emph{a} and \emph{b}, although they can never become \emph{true}!
Example

\[ P = \{ a \leftarrow b, b \leftarrow a \} \]

\[ \Phi^0_P\langle\emptyset, \emptyset\rangle = \langle\emptyset, \emptyset\rangle \]
\[ \Phi^1_P\langle\emptyset, \emptyset\rangle = \langle\emptyset, \emptyset\rangle \]

That is, Fitting semantics cannot assign \textit{false} to \(a\) and \(b\), although they can never become \textit{true}!
Outline

34 Partial Interpretations
35 Fitting Operator
36 Unfounded Sets
37 Well-Founded Operator
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

A set $U \subseteq A(P)$ is an unfounded set of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that:

\begin{align*}
B(r)^+ \cap F \neq \emptyset \text{ or } B(r)^- \cap T \neq \emptyset \text{ or } \\
B(r)^+ \cap U \neq \emptyset
\end{align*}

Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

Rules satisfying Condition 1 are not usable for further derivations.

Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require another atom in $U$ to be true.
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

A set $U \subseteq A(P)$ is an **unfounded set** of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that

\[
B(r)^+ \cap F \neq \emptyset \text{ or } B(r)^- \cap T \neq \emptyset \text{ or } B(r)^+ \cap U \neq \emptyset
\]

Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

Rules satisfying Condition 1 are not usable for further derivations.

Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an(other) atom in $U$ to be true.
Unfounded Sets

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

A set $U \subseteq A(P)$ is an **unfounded set of $P$ wrt $\langle T, F \rangle$**, if for each rule $r \in P$ such that $h(r) \in U$, we have that:

1. $B(r)^+ \cap F \neq \emptyset$ or $B(r)^- \cap T \neq \emptyset$ or $B(r)^+ \cap U \neq \emptyset$

Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

Rules satisfying Condition 1 are not usable for further derivations.

Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require another atom in $U$ to be true.
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- A set $U \subseteq A(P)$ is an **unfounded set** of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that
  
  $$B(r)^+ \cap F \neq \emptyset \text{ or } B(r)^- \cap T \neq \emptyset \text{ or } B(r)^+ \cap U \neq \emptyset$$

- Intuitively, $\langle T, F \rangle$ is what we already know about $P$
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an(other) atom in $U$ to be true
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

A set $U \subseteq A(P)$ is an unfounded set of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that:

1. $B(r)^+ \cap F \neq \emptyset$ or $B(r)^- \cap T \neq \emptyset$ or $B(r)^+ \cap U \neq \emptyset$

Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

Rules satisfying Condition 1 are not usable for further derivations.

Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an(other) atom in $U$ to be true.
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation. A set $U \subseteq A(P)$ is an unfounded set of $P$ with respect to $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that:

1. $B(r)^+ \cap F \neq \emptyset$ or $B(r)^- \cap T \neq \emptyset$ or
2. $B(r)^+ \cap U \neq \emptyset$

Intuitively, $\langle T, F \rangle$ is what we already know about $P$. Rules satisfying Condition 1 are not usable for further derivations. Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an (other) atom in $U$ to be true.
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- A set $U \subseteq A(P)$ is an *unfounded set* of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that
  
  $\begin{align*}
  1 & \quad B(r)^+ \cap F \neq \emptyset \text{ or } B(r)^- \cap T \neq \emptyset \text{ or } \\
  2 & \quad B(r)^+ \cap U \neq \emptyset
  \end{align*}$

- Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

- Rules satisfying Condition 1 are not usable for further derivations.

- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an(other) atom in $U$ to be true.
Unfounded Sets

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

A set $U \subseteq A(P)$ is an unfounded set of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that

1. $B(r)^+ \cap F \neq \emptyset$ or $B(r)^- \cap T \neq \emptyset$ or
2. $B(r)^+ \cap U \neq \emptyset$

Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

Rules satisfying Condition 1 are not usable for further derivations.

Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an(other) atom in $U$ to be true.
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

A set $U \subseteq A(P)$ is an **unfounded set** of $P$ wrt $\langle T, F \rangle$, if for each rule $r \in P$ such that $h(r) \in U$, we have that

1. $B(r)^+ \cap F \neq \emptyset$ or $B(r)^- \cap T \neq \emptyset$ or
2. $B(r)^+ \cap U \neq \emptyset$

Intuitively, $\langle T, F \rangle$ is what we already know about $P$.

Rules satisfying Condition 1 are not usable for further derivations.

Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in $U$ require an(other) atom in $U$ to be true.
Example

\[ P = \left\{ \begin{array}{c}
  a \leftarrow b \\
  b \leftarrow a
\end{array} \right\} \]

- \emptyset \text{ is an unfounded set (by definition)}
- \{a\} \text{ is not an unfounded set of } \mathcal{P} \text{ wrt } \langle \emptyset, \emptyset \rangle
- \{a\} \text{ is an unfounded set of } \mathcal{P} \text{ wrt } \langle \emptyset, \{b\} \rangle
- \{a\} \text{ is not an unfounded set of } \mathcal{P} \text{ wrt } \langle \{b\}, \emptyset \rangle
- \text{Analogously for } \{b\}
- \{a, b\} \text{ is an unfounded set of } \mathcal{P} \text{ wrt } \langle \emptyset, \emptyset \rangle
- \{a, b\} \text{ is an unfounded set of } \mathcal{P} \text{ wrt any partial interpretation}
Unfounded Sets

Example

\[ P = \{ a \leftarrow b, b \leftarrow a \} \]

- \( \emptyset \) is an unfounded set (by definition)
- \( \{a\} \) is not an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{a\} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \{b\} \rangle \)
- \( \{a\} \) is not an unfounded set of \( P \) wrt \( \langle \{b\}, \emptyset \rangle \)
- Analogously for \( \{b\} \)
- \( \{a, b\} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{a, b\} \) is an unfounded set of \( P \) wrt any partial interpretation
Example

\[ P = \left\{ \begin{array}{c} a \leftarrow b \\ b \leftarrow a \end{array} \right\} \]

- \emptyset is an unfounded set (by definition)
- \{a\} is not an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \{a\} is an unfounded set of \( P \) wrt \( \langle \emptyset, \{b\} \rangle \)
- \{a\} is not an unfounded set of \( P \) wrt \( \langle \{b\}, \emptyset \rangle \)
- Analogously for \{b\}
- \{a, b\} is an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \{a, b\} is an unfounded set of \( P \) wrt any partial interpretation
Example

\[ P = \{ a \leftarrow b, b \leftarrow a \} \]

- \( \emptyset \) is an unfounded set (by definition)
- \( \{ a \} \) is not an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{ a \} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \{ b \} \rangle \)
- \( \{ a \} \) is not an unfounded set of \( P \) wrt \( \langle \{ b \}, \emptyset \rangle \)
- Analogously for \( \{ b \} \)
- \( \{ a, b \} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{ a, b \} \) is an unfounded set of \( P \) wrt any partial interpretation
Example

\[ P = \left\{ \begin{array}{ll}
  a & \leftarrow b \\
  b & \leftarrow a
\end{array} \right\} \]

- \( \emptyset \) is an unfounded set (by definition)
- \( \{a\} \) is not an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{a\} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \{b\} \rangle \)
- \( \{a\} \) is not an unfounded set of \( P \) wrt \( \langle \{b\}, \emptyset \rangle \)

Analogously for \( \{b\} \)

- \( \{a, b\} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{a, b\} \) is an unfounded set of \( P \) wrt any partial interpretation
Unfounded Sets

Example

\[ P = \{ a \leftarrow b, b \leftarrow a \} \]

- \{a\} is not an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \{a\} is an unfounded set of \( P \) wrt \( \langle \emptyset, \{b\} \rangle \)
- \{a\} is not an unfounded set of \( P \) wrt \( \langle \{b\}, \emptyset \rangle \)
- Analogously for \{b\}
  - \{a, b\} is an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
  - \{a, b\} is an unfounded set of \( P \) wrt any partial interpretation
Unfounded Sets

Example

\[ P = \{ a \leftarrow b, \ b \leftarrow a \} \]

- \( \emptyset \) is an unfounded set (by definition)
- \( \{a\} \) is not an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{a\} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \{b\} \rangle \)
- \( \{a\} \) is not an unfounded set of \( P \) wrt \( \langle \{b\}, \emptyset \rangle \)
- Analogously for \( \{b\} \)
- \( \{a, b\} \) is an unfounded set of \( P \) wrt \( \langle \emptyset, \emptyset \rangle \)
- \( \{a, b\} \) is an unfounded set of \( P \) wrt any partial interpretation
Unfounded Sets

Example

$$P = \{ \begin{array}{c} a \leftarrow b \\ b \leftarrow a \end{array} \}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of $P$ wrt $\langle \emptyset, \emptyset \rangle$
- $\{a\}$ is an unfounded set of $P$ wrt $\langle \emptyset, \{b\} \rangle$
- $\{a\}$ is not an unfounded set of $P$ wrt $\langle \{b\}, \emptyset \rangle$

- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of $P$ wrt $\langle \emptyset, \emptyset \rangle$
- $\{a, b\}$ is an unfounded set of $P$ wrt any partial interpretation
Greatest unfounded sets

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- **Observation** The union of two unfounded sets is an unfounded set.
- The greatest unfounded set of $P$ wrt $\langle T, F \rangle$ is the union of all unfounded sets of $P$ wrt $\langle T, F \rangle$.
  
  It is denoted by $U_P(T, F)$.

- Alternatively, we may define

  $$U_P(T, F) = A(P) \setminus Cn(\{r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$$

- **Note** $Cn(\{r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$ contains all non-circularly derivable atoms from $P$ wrt $\langle T, F \rangle$. 


Greatest unfounded sets

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- **Observation** The union of two unfounded sets is an unfounded set.
- The greatest unfounded set of $P$ wrt $\langle T, F \rangle$ is the union of all unfounded sets of $P$ wrt $\langle T, F \rangle$.
  It is denoted by $U_P(\langle T, F \rangle)$.
- Alternatively, we may define
  \[
  U_P(\langle T, F \rangle) = A(P) \setminus Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)
  \]
- **Note** $Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$ contains all non-circularly derivable atoms from $P$ wrt $\langle T, F \rangle$. 
Greatest unfounded sets

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Observation: The union of two unfounded sets is an unfounded set.
- The greatest unfounded set of $P$ wrt $\langle T, F \rangle$ is the union of all unfounded sets of $P$ wrt $\langle T, F \rangle$.

It is denoted by $U_P \langle T, F \rangle$.

- Alternatively, we may define

$$U_P \langle T, F \rangle = A(P) \setminus Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$$

- Note: $Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$ contains all non-circularly derivable atoms from $P$ wrt $\langle T, F \rangle$. 
Greatest unfounded sets

Let \( P \) be a normal logic program, and let \( \langle T, F \rangle \) be a partial interpretation.

- **Observation** The union of two unfounded sets is an unfounded set.

- **The greatest unfounded set of** \( P \) **wrt** \( \langle T, F \rangle \) **is the union of all unfounded sets of** \( P \) **wrt** \( \langle T, F \rangle \).

  It is denoted by \( U_P\langle T, F \rangle \).

  Alternatively, we may define

  \[
  U_P\langle T, F \rangle = A(P) \setminus Cn\left( \{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T \right)
  \]

- **Note** \( Cn\left( \{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T \right) \) contains all non-circularly derivable atoms from \( P \) wrt \( \langle T, F \rangle \).
Greatest unfounded sets

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- **Observation** The union of two unfounded sets is an unfounded set.
- The greatest unfounded set of $P$ wrt $\langle T, F \rangle$ is the union of all unfounded sets of $P$ wrt $\langle T, F \rangle$.
  It is denoted by $U_P\langle T, F \rangle$.
- Alternatively, we may define
  
  $$U_P\langle T, F \rangle = A(P) \setminus Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$$

- **Note** $Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$ contains all non-circularly derivable atoms from $P$ wrt $\langle T, F \rangle$. 

Unfounded Sets
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- **Observation** The union of two unfounded sets is an unfounded set.
- The greatest unfounded set of $P$ wrt $\langle T, F \rangle$ is the union of all unfounded sets of $P$ wrt $\langle T, F \rangle$.
  
  It is denoted by $U_P \langle T, F \rangle$.

- Alternatively, we may define
  
  $$U_P \langle T, F \rangle = A(P) \setminus Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$$

- **Note** $Cn(\{ r \in P \mid B(r)^+ \cap F = \emptyset \}^T)$ contains all non-circularly derivable atoms from $P$ wrt $\langle T, F \rangle$. 
Well-founded operator

Let \( P \) be a normal logic program, and let \( \langle T, F \rangle \) be a partial interpretation.

- **Observation** Condition 1 (in the definition of an unfounded set) corresponds to \( F_P\langle T, F \rangle \) of Fitting’s \( \Phi_P\langle T, F \rangle \).

- **Idea** Extend (negative part of) Fitting’s operator \( \Phi_P \).
  That is,
  - keep definition of \( T_P\langle T, F \rangle \) from \( \Phi_P\langle T, F \rangle \) and
  - replace \( F_P\langle T, F \rangle \) from \( \Phi_P\langle T, F \rangle \) by \( U_P\langle T, F \rangle \).

- In words, an atom must be *false* if it belongs to the greatest unfounded set.

- **Definition** \( \Omega_P\langle T, F \rangle = \langle T_P\langle T, F \rangle, U_P\langle T, F \rangle \rangle \)
  \( \Phi_P\langle T, F \rangle \sqsubseteq \Omega_P\langle T, F \rangle \).
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $F_P\langle T, F \rangle$ of Fitting's $\Phi_P\langle T, F \rangle$.
- Idea Extend (negative part of) Fitting's operator $\Phi_P$.
  
  That is,
  - keep definition of $T_P\langle T, F \rangle$ from $\Phi_P\langle T, F \rangle$ and
  - replace $F_P\langle T, F \rangle$ from $\Phi_P\langle T, F \rangle$ by $U_P\langle T, F \rangle$.

- In words, an atom must be false if it belongs to the greatest unfounded set.

- Definition $\Omega_P\langle T, F \rangle = \langle T_P\langle T, F \rangle, U_P\langle T, F \rangle \rangle$.
- Property $\Phi_P\langle T, F \rangle \subseteq \Omega_P\langle T, F \rangle$. 
Well-founded operator

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- **Observation** Condition 1 (in the definition of an unfounded set) corresponds to $F_P\langle T, F \rangle$ of Fitting’s $\Phi_P\langle T, F \rangle$.
- **Idea** Extend (negative part of) Fitting’s operator $\Phi_P$.
  That is,
  - keep definition of $T_P\langle T, F \rangle$ from $\Phi_P\langle T, F \rangle$ and
  - replace $F_P\langle T, F \rangle$ from $\Phi_P\langle T, F \rangle$ by $U_P\langle T, F \rangle$.

- In words, an atom must be false if it belongs to the greatest unfounded set.

- **Definition** $\Omega_P\langle T, F \rangle = \langle T_P\langle T, F \rangle, U_P\langle T, F \rangle \rangle$.
- **Property** $\Phi_P\langle T, F \rangle \subseteq \Omega_P\langle T, F \rangle$. 

---

Well-Founded Operator
Well-founded operator

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- **Observation** Condition 1 (in the definition of an unfounded set) corresponds to $F_P \langle T, F \rangle$ of Fitting’s $\Phi_P \langle T, F \rangle$.
- **Idea** Extend (negative part of) Fitting’s operator $\Phi_P$.
  That is,
  - keep definition of $T_P \langle T, F \rangle$ from $\Phi_P \langle T, F \rangle$ and
  - replace $F_P \langle T, F \rangle$ from $\Phi_P \langle T, F \rangle$ by $U_P \langle T, F \rangle$.

- **In words**, an atom must be *false* if it belongs to the greatest unfounded set.

- **Definition** $\Omega_P \langle T, F \rangle = \langle T_P \langle T, F \rangle, U_P \langle T, F \rangle \rangle$.
- **Property** $\Phi_P \langle T, F \rangle \subseteq \Omega_P \langle T, F \rangle$. 

Torsten Schaub (KRR@UP)  
Answer Set Solving in Practice  
February 18, 2019  
272 / 653
Well-founded operator

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $F_P \langle T, F \rangle$ of Fitting's $\Phi_P \langle T, F \rangle$
- Idea Extend (negative part of) Fitting's operator $\Phi_P$
  That is,
  - keep definition of $T_P \langle T, F \rangle$ from $\Phi_P \langle T, F \rangle$ and
  - replace $F_P \langle T, F \rangle$ from $\Phi_P \langle T, F \rangle$ by $U_P \langle T, F \rangle$
- In words, an atom must be false if it belongs to the greatest unfounded set

- Definition $\Omega_P \langle T, F \rangle = \langle T_P \langle T, F \rangle, U_P \langle T, F \rangle \rangle$
- Property $\Phi_P \langle T, F \rangle \sqsubseteq \Omega_P \langle T, F \rangle$
Well-Founded Operator

Example

\[ P = \left\{ \begin{array}{llll}
    & a & \leftarrow & c \\
    b & \leftarrow & \sim a \\
    & d & \leftarrow & \sim c, \sim e \\
    & e & \leftarrow & b, \sim f \\
\end{array} \right\} \]

Let's iterate \( \Omega_{P_1} \) on \( \langle \{c\}, \emptyset \rangle \):

\[
\begin{align*}
\Omega_P\langle \{c\}, \emptyset \rangle &= \langle \{a\}, \{d, f\} \rangle \\
\Omega_P\langle \{a\}, \{d, f\} \rangle &= \langle \{a, c\}, \{b, e, f\} \rangle \\
\Omega_P\langle \{a, c\}, \{b, e, f\} \rangle &= \langle \{a\}, \{b, d, e, f\} \rangle \\
\Omega_P\langle \{a\}, \{b, d, e, f\} \rangle &= \langle \{a, c\}, \{b, e, f\} \rangle \\
\vdots
\end{align*}
\]
Example

\[ P = \begin{cases} 
  a \leftarrow c \\ 
  c \leftarrow a, \sim d \\ 
  e \leftarrow b, \sim f \\ 
  b \leftarrow \sim a \\ 
  d \leftarrow \sim c, \sim e \\ 
  e \leftarrow e 
\end{cases} \]

Let's iterate \( \Omega_{P_1} \) on \( \langle \{c\}, \emptyset \rangle \):

\[
\Omega_P \langle \{c\}, \emptyset \rangle = \langle \{a\}, \{d, f\} \rangle
\]

\[
\Omega_P \langle \{a\}, \{d, f\} \rangle = \langle \{a, c\}, \{b, e, f\} \rangle
\]

\[
\Omega_P \langle \{a, c\}, \{b, e, f\} \rangle = \langle \{a\}, \{b, d, e, f\} \rangle
\]

\[
\Omega_P \langle \{a\}, \{b, d, e, f\} \rangle = \langle \{a, c\}, \{b, e, f\} \rangle
\]

\[
\vdots
\]
Well-founded semantics

- Define the iterative variant of $\Omega_P$ analogously to $\Phi_P$:

  $\Omega_P^0 \langle T, F \rangle = \langle T, F \rangle$
  $\Omega_P^{i+1} \langle T, F \rangle = \Omega_P \Omega_P^i \langle T, F \rangle$

- Define the well-founded semantics of a normal logic program $P$ as the partial interpretation:

  $\bigsqcup_{i \geq 0} \Omega_P^i \langle \emptyset, \emptyset \rangle$
Well-founded semantics

- Define the iterative variant of $\Omega_P$ analogously to $\Phi_P$:

  $\Omega^0_P(T, F) = \langle T, F \rangle$

  $\Omega^{i+1}_P(T, F) = \Omega_P \Omega^i_P(T, F)$

- Define the well-founded semantics of a normal logic program $P$ as the partial interpretation:

  $\bigcup_{i \geq 0} \Omega^i_P(\emptyset, \emptyset)$
Well-Founded Operator

Example

\[ P = \left\{ \begin{array}{ccc}
    a & \leftarrow & c \\
    b & \leftarrow & \neg a \\
    c & \leftarrow & a, \neg d \\
    d & \leftarrow & \neg c, \neg e \\
    e & \leftarrow & b, \neg f \\
    e & \leftarrow & e
\end{array} \right\} \]

\[ \Omega^0(\emptyset, \emptyset) = \langle \emptyset, \emptyset \rangle \]
\[ \Omega^1(\emptyset, \emptyset) = \Omega(\emptyset, \emptyset) = \langle \{a\}, \{f\} \rangle \]
\[ \Omega^2(\emptyset, \emptyset) = \Omega(\{a\}, \{f\}) = \langle \{a\}, \{b, e, f\} \rangle \]
\[ \Omega^3(\emptyset, \emptyset) = \Omega(\{a\}, \{b, e, f\}) = \langle \{a\}, \{b, e, f\} \rangle \]

\[ \bigcup_{i \geq 0} \Omega^i(\emptyset, \emptyset) = \langle \{a\}, \{b, e, f\} \rangle \]
Example

\[ P = \left\{ a \leftarrow c \leftarrow a, \not\sim d \leftarrow \not\sim a, \not\sim d \leftarrow \not\sim c, \not\sim e \leftarrow e \right\} \]

\[
\begin{align*}
\Omega^0 \langle \emptyset, \emptyset \rangle &= \langle \emptyset, \emptyset \rangle \\
\Omega^1 \langle \emptyset, \emptyset \rangle &= \Omega \langle \emptyset, \emptyset \rangle &= \langle \{a\}, \{f\} \rangle \\
\Omega^2 \langle \emptyset, \emptyset \rangle &= \Omega \langle \{a\}, \{f\} \rangle &= \langle \{a\}, \{b, e, f\} \rangle \\
\Omega^3 \langle \emptyset, \emptyset \rangle &= \Omega \langle \{a\}, \{b, e, f\} \rangle &= \langle \{a\}, \{b, e, f\} \rangle \\
\bigcup_{i \geq 0} \Omega^i \langle \emptyset, \emptyset \rangle &= \langle \{a\}, \{b, e, f\} \rangle
\end{align*}
\]
Let $P$ be a normal logic program

- $\Omega_P(\emptyset, \emptyset)$ is monotonic
  That is, $\Omega_P^i(\emptyset, \emptyset) \subseteq \Omega_P^{i+1}(\emptyset, \emptyset)$

- The well-founded semantics of $P$ is
  - not conflicting,
  - and generally not total

- We have $\bigcup_{i \geq 0} \Phi_P^i(\emptyset, \emptyset) \subseteq \bigcup_{i \geq 0} \Omega_P^i(\emptyset, \emptyset)$
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Define $\langle T, F \rangle$ as a well-founded fixpoint of $P$ if $\Omega_P \langle T, F \rangle = \langle T, F \rangle$

- The well-founded semantics is the $\sqsubseteq$-least well-founded fixpoint of $P$
- Any other well-founded fixpoint extends the well-founded semantics
- Total well-founded fixpoints correspond to stable models
Well-founded fixpoints

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Define $\langle T, F \rangle$ as a well-founded fixpoint of $P$ if $\Omega_P \langle T, F \rangle = \langle T, F \rangle$.

- The well-founded semantics is the $\sqsubseteq$-least well-founded fixpoint of $P$.
- Any other well-founded fixpoint extends the well-founded semantics.
- Total well-founded fixpoints correspond to stable models.
Well-founded fixpoints: Example

\[ P = \left\{ \begin{array}{ccc}
a & \leftarrow & c \leftarrow a, \sim d \\
b & \leftarrow & \sim a \\
d & \leftarrow & \sim c, \sim e \\
e & \leftarrow & b, \sim f \\
e & \leftarrow & e 
\end{array} \right\} \]

- \( P \) has two total well-founded fixpoints:
  - \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  - \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)

- Both of them represent stable models
Well-founded fixpoints: Example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \ 
  c \leftarrow a, \neg d \\
  b \leftarrow \neg a \\
  d \leftarrow \neg c, \neg e \\
  e \leftarrow b, \neg f
\end{array} \right\} \]

- \( P \) has two total well-founded fixpoints:
  1. \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  2. \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)

- Both of them represent stable models
Well-founded fixpoints: Example

\[ P = \begin{cases} \text{a} & \leftarrow & \text{c} & \leftarrow & \text{a}, \sim \text{d} & \text{e} & \leftarrow & \text{b}, \sim \text{f} \\ \text{b} & \leftarrow & \sim \text{a} & \text{d} & \leftarrow & \sim \text{c}, \sim \text{e} & \text{e} & \leftarrow & \text{e} \end{cases} \]

- \( P \) has two total well-founded fixpoints:
  1. \( \langle \{ \text{a}, \text{c} \}, \{ \text{b}, \text{d}, \text{e}, \text{f} \} \rangle \)
  2. \( \langle \{ \text{a}, \text{d} \}, \{ \text{b}, \text{c}, \text{e}, \text{f} \} \rangle \)

- Both of them represent stable models
Well-founded fixpoints: Example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \ \\
b \leftarrow \ \sim a \ \\
c \leftarrow a, \sim d \ \\
d \leftarrow \sim c, \sim e \ \\
e \leftarrow b, \sim f \\
e \leftarrow e \end{array} \right. \]

- \( P \) has two total well-founded fixpoints:
  1. \( \langle \{a, c\}, \{b, d, e, f\} \rangle \)
  2. \( \langle \{a, d\}, \{b, c, e, f\} \rangle \)

- Both of them represent stable models
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\Omega_P \langle T, F \rangle = \langle T', F' \rangle$
  - If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$
  - That is, $\Omega_P$ is stable model preserving.
  - Hence, $\Omega_P$ can be used for approximating stable models and so for propagation in ASP-solvers.

- In contrast to $\Phi_P$, operator $\Omega_P$ is sufficient for propagation because total fixpoints correspond to stable models.

- Note In addition to $\Omega_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view).
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\Omega_P \langle T, F \rangle = \langle T', F' \rangle$
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$.

That is, $\Omega_P$ is stable model preserving.

Hence, $\Omega_P$ can be used for approximating stable models and so for propagation in ASP-solvers.

In contrast to $\Phi_P$, operator $\Omega_P$ is sufficient for propagation because total fixpoints correspond to stable models.

Note: In addition to $\Omega_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view).
Well-Founded Operator

Properties

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\Omega_P \langle T, F \rangle = \langle T', F' \rangle$
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

That is, $\Omega_P$ is stable model preserving.

Hence, $\Omega_P$ can be used for approximating stable models and so for propagation in ASP-solvers.

- In contrast to $\Phi_P$, operator $\Omega_P$ is sufficient for propagation because total fixpoints correspond to stable models.

- Note: In addition to $\Omega_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view).
Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation

- Let $Ω_P(\langle T, F \rangle) = \langle T', F' \rangle$
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$
  That is, $Ω_P$ is stable model preserving
  Hence, $Ω_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- In contrast to $Φ_P$, operator $Ω_P$ is sufficient for propagation because total fixpoints correspond to stable models

- Note In addition to $Ω_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)
Well-Founded Operator

Properties

Let $P$ be a normal logic program, and let $\langle T, F \rangle$ be a partial interpretation

- Let $\Omega_P \langle T, F \rangle = \langle T', F' \rangle$
- If $X$ is a stable model of $P$ such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$
  
  That is, $\Omega_P$ is stable model preserving
  
  Hence, $\Omega_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- In contrast to $\Phi_P$, operator $\Omega_P$ is sufficient for propagation because total fixpoints correspond to stable models

- Note In addition to $\Omega_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)
Proof-theoretic Characterization:
Overview

38 Tableau Calculi
39 Tableau Calculi for ASP
40 Tableau Calculi characterizing ASP solvers
41 Proof complexity
Tableau Calculi

Tableau Calculi for ASP
Tableau Calculi characterizing ASP solvers
Proof complexity
Motivation

- **Goal** Analyze computations in ASP solvers
- **Wanted** A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP solvers
- **Idea** View stable model computations as derivations in an inference system

Consider Tableau-based proof systems for analyzing ASP solving.
Motivation

- **Goal** Analyze computations in ASP solvers
- **Wanted** A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP solvers
- **Idea** View stable model computations as derivations in an inference system

Consider Tableau-based proof systems for analyzing ASP solving
Motivation

- **Goal** Analyze computations in ASP solvers
- **Wanted** A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP solvers
- **Idea** View stable model computations as derivations in an inference system
  
  Consider **Tableau-based proof systems** for analyzing ASP solving
Tableau calculi

- Traditionally, tableau calculi are used for
  - automated theorem proving and
  - proof theoretical analysis
  in classical as well as non-classical logics

- General idea Given an input, prove some property by decomposition
  Decomposition is done by applying deduction rules

- For details, see Handbook of Tableau Methods, Kluwer, 1999
A tableau is a (mostly binary) tree

A branch in a tableau is a path from the root to a leaf

A branch containing $\gamma_1, \ldots, \gamma_m$ can be extended by applying tableau rules of form

$$\frac{\gamma_1, \ldots, \gamma_m}{\alpha_1} \quad \frac{\gamma_1, \ldots, \gamma_m}{\beta_1 \mid \ldots \mid \beta_n}$$

Rules of the former format append entries $\alpha_1, \ldots, \alpha_n$ to the branch

Rules of the latter format create multiple sub-branches for $\beta_1, \ldots, \beta_n$
General definitions

- A tableau is a (mostly binary) tree
- A branch in a tableau is a path from the root to a leaf
- A branch containing $\gamma_1, \ldots, \gamma_m$ can be extended by applying tableau rules of form

$$\gamma_1, \ldots, \gamma_m$$

$$\alpha_1$$

$$\vdots$$

$$\alpha_n$$

$$\gamma_1, \ldots, \gamma_m$$

$$\beta_1 \mid \ldots \mid \beta_n$$

- Rules of the former format append entries $\alpha_1, \ldots, \alpha_n$ to the branch
- Rules of the latter format create multiple sub-branches for $\beta_1, \ldots, \beta_n$
General definitions

- A tableau is a (mostly binary) tree
- A branch in a tableau is a path from the root to a leaf
- A branch containing $\gamma_1, \ldots, \gamma_m$ can be extended by applying tableau rules of form

$$\frac{\gamma_1, \ldots, \gamma_m}{\alpha_1} \quad \frac{\gamma_1, \ldots, \gamma_m}{\beta_1 \mid \ldots \mid \beta_n}$$

- Rules of the former format append entries $\alpha_1, \ldots, \alpha_n$ to the branch
- Rules of the latter format create multiple sub-branches for $\beta_1, \ldots, \beta_n$
Example

A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from $\neg$, $\land$, and $\lor$, consists of rules

\[
\begin{align*}
\frac{\neg
\neg \alpha}{\alpha} & \quad \frac{\alpha_1 \land \alpha_2}{\alpha_1 \quad \alpha_2} & \quad \frac{\beta_1 \lor \beta_2}{\beta_1 \quad \beta_2}
\end{align*}
\]

All rules are semantically valid, when interpreting entries in a branch conjunctively and distinct (sub-)branches as connected disjunctively.

A propositional formula $\varphi$ is unsatisfiable iff there is a tableau with $\varphi$ as the root node such that

1. all other entries can be produced by tableau rules and
2. every branch contains some formulas $\alpha$ and $\neg \alpha$
A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from $\neg$, $\land$, and $\lor$, consists of rules:

- $\neg\neg\alpha \rightarrow \alpha$
- $\alpha_1 \land \alpha_2 \rightarrow \alpha_1$
- $\alpha_2 \rightarrow \beta_1 \lor \beta_2$

All rules are semantically valid, when interpreting entries in a branch conjunctively and distinct (sub-)branches as connected disjunctively.

A propositional formula $\varphi$ is unsatisfiable iff there is a tableau with $\varphi$ as the root node such that:

1. all other entries can be produced by tableau rules and
2. every branch contains some formulas $\alpha$ and $\neg\alpha$
A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from $\neg$, $\land$, and $\lor$, consists of rules:

- $\neg\neg \alpha \quad \alpha$
- $\alpha_1 \land \alpha_2 \quad \alpha_1$
- $\alpha_2$
- $\beta_1 \lor \beta_2 \quad \beta_1$
- $\beta_2$

All rules are semantically valid, when interpreting entries in a branch conjunctively and distinct (sub-)branches as connected disjunctively.

A propositional formula $\varphi$ is unsatisfiable iff there is a tableau with $\varphi$ as the root node such that:

1. all other entries can be produced by tableau rules and
2. every branch contains some formulas $\alpha$ and $\neg \alpha$
Example

\((1)\)  \(a \land ((\neg b \land (\neg a \lor b)) \lor \neg\neg\neg a)\)  
\(\varphi\)

\((2)\)  \(a\)  
\(\text{[1]}\)

\((3)\)  \((\neg b \land (\neg a \lor b)) \lor \neg\neg\neg a\)  
\(\text{[1]}\)

\((4)\)  \(\neg b \land (\neg a \lor b)\)  
\(\text{[3]}\)

\((5)\)  \(\neg b\)  
\(\text{[4]}\)

\((6)\)  \(\neg a \lor b\)  
\(\text{[4]}\)

\((7)\)  \(\neg a\)  
\(\text{[6]}\)

\((8)\)  \(b\)  
\(\text{[6]}\)

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)  
Hence,  \(a \land ((\neg b \land (\neg a \lor b)) \lor \neg\neg\neg a)\) is unsatisfiable
Tableau Calculi

Example

(1) \( a \land (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \) \[\varphi\]
(2) \( a \) \[1\]
(3) \( (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \) \[1\]

(4) \( \neg b \land (\neg a \lor b) \) \[3\]
(5) \( \neg b \) \[4\]
(6) \( \neg a \lor b \) \[4\]
(7) \( \neg a \) \[6\]
(8) \( b \) \[6\]

(9) \( \neg \neg \neg a \) \[3\]
(10) \( \neg a \) \[9\]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
Hence, \( a \land (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \) is unsatisfiable
Example

\( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) \[
\begin{align*}
(1) & \quad a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \\
(2) & \quad a \\
(3) & \quad (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \\
(4) & \quad \neg b \land (\neg a \lor b) \\
(5) & \quad \neg b \\
(6) & \quad \neg a \lor b \\
(7) & \quad \neg a \\
(8) & \quad b \\
(9) & \quad \neg \neg \neg a \\
(10) & \quad \neg a
\end{align*}
\]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
Hence, \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) is unsatisfiable
Example

(1) \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) [\( \phi \)]
(2) \( a \) [1]
(3) \( (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \) [1]

(4) \( \neg b \land (\neg a \lor b) \) [3]
(5) \( \neg b \) [4]
(6) \( \neg a \lor b \) [4]
(7) \( \neg a \) [6]
(8) \( b \) [6]

(9) \( \neg \neg \neg a \) [3]
(10) \( \neg a \) [9]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) is unsatisfiable
Example

(1) \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) \[ \phi \]

(2) \( \neg \neg \neg a \) \[ 1 \]

(3) \( \neg b \land (\neg a \lor b) \lor \neg \neg \neg a \) \[ 1 \]

(4) \( \neg b \land (\neg a \lor b) \) \[ 3 \]

(5) \( \neg b \) \[ 4 \]

(6) \( \neg a \lor b \) \[ 4 \]

(7) \( \neg a \) \[ 6 \]

(8) \( b \) \[ 6 \]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) is unsatisfiable
Example

(1) \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) [\( \varphi \)]
(2) \( a \) [1]
(3) \( (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \) [1]
(4) \( \neg b \land (\neg a \lor b) \) [3]
(5) \( \neg b \) [4]
(6) \( \neg a \lor b \) [4]
(7) \( \neg a \) [6]
(8) \( b \) [6]
(9) \( \neg \neg \neg a \) [3]
(10) \( \neg a \) [9]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
Hence, \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \) is unsatisfiable
Example

1. \( a \land \left( \left( \neg b \land \left( \neg a \lor b \right) \right) \lor \neg \neg \neg a \right) \) \[\varphi\]
2. \( a \) \[1\]
3. \( \left( \neg b \land \left( \neg a \lor b \right) \right) \lor \neg \neg \neg a \) \[1\]
4. \( \neg b \land \left( \neg a \lor b \right) \) \[3\]
5. \( \neg b \) \[4\]
6. \( \neg a \lor b \) \[4\]
7. \( \neg a \) \[6\]
8. \( b \) \[6\]
9. \( \neg \neg \neg a \) \[3\]
10. \( \neg a \) \[9\]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, \( a \land \left( \left( \neg b \land \left( \neg a \lor b \right) \right) \lor \neg \neg \neg a \right) \) is unsatisfiable
All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, $a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a)$ is unsatisfiable
Example

1. \( a \land (\neg b \land (\neg a \lor b)) \lor \neg \neg a \) \[\varphi\]
2. \( a \) \[1\]
3. \( (\neg b \land (\neg a \lor b)) \lor \neg \neg a \) \[1\]
4. \( \neg b \land (\neg a \lor b) \) \[3\]
5. \( \neg b \) \[4\]
6. \( \neg a \lor b \) \[4\]
7. \( \neg a \) \[6\]
8. \( b \) \[6\]
9. \( \neg \neg a \) \[3\]
10. \( \neg a \) \[9\]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, \( a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg a) \) is unsatisfiable
Example

(1) \[ a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \]  \[ \varphi \]

(2) \[ a \]  \[ 1 \]

(3) \[ (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \]  \[ 1 \]

(4) \[ \neg b \land (\neg a \lor b) \]  \[ 3 \]

(5) \[ \neg b \]  \[ 4 \]

(6) \[ \neg a \land b \]  \[ 4 \]

(7) \[ \neg a \]  \[ 6 \]

(8) \[ b \]  \[ 6 \]

(9) \[ \neg \neg a \]  \[ 3 \]

(10) \[ \neg a \]  \[ 9 \]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, \[ a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg a) \] is unsatisfiable
Example

\begin{align*}
(1) & \quad a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a) \quad [\varphi] \\
(2) & \quad a \quad [1] \\
(3) & \quad (\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a \quad [1] \\
(4) & \quad \neg b \land (\neg a \lor b) \quad [3] \\
(5) & \quad \neg b \quad [4] \\
(6) & \quad \neg a \lor b \quad [4] \\
(7) & \quad \neg a \quad [6] \\
(8) & \quad b \quad [6] \\
(9) & \quad \neg \neg \neg a \quad [3] \\
(10) & \quad \neg a \quad [9]
\end{align*}

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \land ((\neg b \land (\neg a \lor b)) \lor \neg \neg \neg a)$ is unsatisfiable
Tableau Calculi for ASP

Outline

38 Tableau Calculi

39 Tableau Calculi for ASP

40 Tableau Calculi characterizing ASP solvers

41 Proof complexity
Tableaux and ASP

- A tableau rule captures an elementary inference scheme in an ASP solver.
- A branch in a tableau corresponds to a successful or unsuccessful computation of a stable model.
- An entire tableau represents a traversal of the search space.
Tableaux and ASP

- A tableau rule captures an elementary inference scheme in an ASP solver.
- A branch in a tableau corresponds to a successful or unsuccessful computation of a stable model.
- An entire tableau represents a traversal of the search space.
Tableaux and ASP

- A tableau rule captures an elementary inference scheme in an ASP solver.
- A branch in a tableau corresponds to a successful or unsuccessful computation of a stable model.
- An entire tableau represents a traversal of the search space.
A (signed) **tableau** for a logic program $P$ is a binary tree such that
- the root node of the tree consists of the rules in $P$;
- the other nodes in the tree are entries of the form $Tv$ or $Fv$, called **signed literals**, where $v$ is a variable,
- generated by extending a tableau using deduction rules (given below)

An entry $Tv$ ($Fv$) reflects that variable $v$ is *true* (*false*) in a corresponding variable assignment

A set of signed literals constitutes a partial assignment

For a normal logic program $P$,
- atoms of $P$ in $A(P)$ and
- bodies of $P$ in $B(P)$

can occur as variables in signed literals
A (signed) tableau for a logic program $P$ is a binary tree such that:
- The root node of the tree consists of the rules in $P$;
- The other nodes in the tree are entries of the form $T_v$ or $F_v$, called signed literals, where $v$ is a variable,
- Generated by extending a tableau using deduction rules (given below).

An entry $T_v$ ($F_v$) reflects that variable $v$ is true (false) in a corresponding variable assignment.

A set of signed literals constitutes a partial assignment.

For a normal logic program $P$,
- Atoms of $P$ in $A(P)$ and
- Bodies of $P$ in $B(P)$
can occur as variables in signed literals.
ASP-specific definitions

A (signed) tableau for a logic program $P$ is a binary tree such that
- the root node of the tree consists of the rules in $P$;
- the other nodes in the tree are entries of the form $Tv$ or $Fv$, called signed literals, where $v$ is a variable,
- generated by extending a tableau using deduction rules (given below)

An entry $Tv$ ($Fv$) reflects that variable $v$ is true (false) in a corresponding variable assignment

A set of signed literals constitutes a partial assignment

For a normal logic program $P$,
- atoms of $P$ in $A(P)$ and
- bodies of $P$ in $B(P)$

can occur as variables in signed literals
Tableau rules for ASP at a glance

(FTB) \[ \frac{p \leftarrow l_1, \ldots, l_n}{t_l, \ldots, t_l} \]
\[ \frac{t_{l_1}, \ldots, t_{l_n}}{T \{ l_1, \ldots, l_n \}} \]

(BFB) \[ \frac{t_{l_1}, \ldots, t_{l_i-1}, t_{l_i+1}, \ldots, t_l}{f_{l_i}} \]
\[ \frac{p \leftarrow l_1, \ldots, l_n}{F \{ l_1, \ldots, l_n \}} \]

(FTA) \[ \frac{p \leftarrow l_1, \ldots, l_n}{T \{ l_1, \ldots, l_n \}} \]
\[ \frac{t_{l_1}, \ldots, t_{l_i}}{T_p} \]

(BFA) \[ \frac{p \leftarrow l_1, \ldots, l_n}{F_p} \]
\[ \frac{F \{ l_1, \ldots, l_n \}}{F \{ l_1, \ldots, l_i, \ldots, l_n \}} \]

(BTB) \[ \frac{T \{ l_1, \ldots, l_i, \ldots, l_n \}}{t_{l_i}} \]

(Cut[X]) \[ \frac{T_v \mid F_v}{\# [X]} \]
More concepts

- **A tableau calculus** is a set of tableau rules

- A branch in a tableau is conflicting, if it contains both $\mathbf{T}_v$ and $\mathbf{F}_v$ for some variable $v$

- A branch in a tableau is total for a program $P$, if it contains either $\mathbf{T}_v$ or $\mathbf{F}_v$ for each $v \in A(P) \cup B(P)$

- A branch in a tableau of some calculus $\mathcal{T}$ is closed, if no rule in $\mathcal{T}$ other than $\text{Cut}$ can produce any new entries

- A branch in a tableau is complete, if it is either conflicting or both total and closed

- A tableau is complete, if all its branches are complete

- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program $P$, if every branch in the tableau is conflicting
More concepts

- A **tableau calculus** is a set of tableau rules.
- A branch in a tableau is **conflicting**, if it contains both $Tv$ and $Fv$ for some variable $v$.
- A branch in a tableau is **total** for a program $P$, if it contains either $Tv$ or $Fv$ for each $v \in A(P) \cup B(P)$.
- A branch in a tableau of some calculus $\mathcal{T}$ is **closed**, if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries.
- A branch in a tableau is **complete**, if it is either conflicting or both total and closed.
- A tableau is complete, if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program $P$, if every branch in the tableau is conflicting.
More concepts

- **A tableau calculus** is a set of tableau rules.
- A branch in a tableau is **conflicting**, if it contains both $Tv$ and $Fv$ for some variable $v$.
- A branch in a tableau is **total** for a program $P$, if it contains either $Tv$ or $Fv$ for each $v \in A(P) \cup B(P)$.
- A branch in a tableau of some calculus $\mathcal{T}$ is **closed**, if no rule in $\mathcal{T}$ other than $Cut$ can produce any new entries.
- A branch in a tableau is **complete**, if it is either conflicting or both total and closed.
- A tableau is complete, if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program $P$, if every branch in the tableau is conflicting.
More concepts

- A **tableau calculus** is a set of tableau rules.

- A branch in a tableau is **conflicting**, if it contains both $T_v$ and $F_v$ for some variable $v$.

- A branch in a tableau is **total** for a program $P$, if it contains either $T_v$ or $F_v$ for each $v \in A(P) \cup B(P)$.

- A branch in a tableau of some calculus $\mathcal{T}$ is **closed**, if no rule in $\mathcal{T}$ other than $\text{Cut}$ can produce any new entries.

- A branch in a tableau is **complete**, if it is either conflicting or both total and closed.

- A tableau is complete, if all its branches are complete.

- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program $P$, if every branch in the tableau is conflicting.
More concepts

- A **tableau calculus** is a set of tableau rules.

- A branch in a tableau is **conflicting**, if it contains both $T_v$ and $F_v$ for some variable $v$.

- A branch in a tableau is **total** for a program $P$, if it contains either $T_v$ or $F_v$ for each $v \in A(P) \cup B(P)$.

- A branch in a tableau of some calculus $\mathcal{T}$ is **closed**, if no rule in $\mathcal{T}$ other than $Cut$ can produce any new entries.

- A branch in a tableau is **complete**, if it is either conflicting or both total and closed.

- A tableau is complete, if all its branches are complete.

- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program $P$, if every branch in the tableau is conflicting.
More concepts

- A **tableau calculus** is a set of tableau rules.
- A branch in a tableau is **conflicting**, if it contains both $T_v$ and $F_v$ for some variable $v$.
- A branch in a tableau is **total** for a program $P$, if it contains either $T_v$ or $F_v$ for each $v \in A(P) \cup B(P)$.
- A branch in a tableau of some calculus $\mathcal{T}$ is **closed**, if no rule in $\mathcal{T}$ other than $Cut$ can produce any new entries.
- A branch in a tableau is **complete**, if it is either conflicting or both total and closed.
- A tableau is **complete**, if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a **refutation** of $\mathcal{T}$ for a program $P$, if every branch in the tableau is conflicting.
More concepts

- A **tableau calculus** is a set of tableau rules.
- A branch in a tableau is **conflicting**, if it contains both $T \neg v$ and $F \neg v$ for some variable $\neg v$.
- A branch in a tableau is **total** for a program $P$, if it contains either $T \neg v$ or $F \neg v$ for each $\neg v \in A(P) \cup B(P)$.
- A branch in a tableau of some calculus $T$ is **closed**, if no rule in $T$ other than $Cut$ can produce any new entries.
- A branch in a tableau is **complete**, if it is either conflicting or both total and closed.
- A tableau is **complete**, if all its branches are complete.
- A tableau of some calculus $T$ is a **refutation** of $T$ for a program $P$, if every branch in the tableau is conflicting.
Consider the program

\[
P = \begin{cases}
  a & \leftarrow \\
  c & \leftarrow \neg b, \neg d \\
  d & \leftarrow a, \neg c
\end{cases}
\]

having stable models \(\{a, c\}\) and \(\{a, d\}\)
(Previewed) Example

\[ a \leftarrow \]
\[ c \leftarrow \lnot b, \lnot d \]
\[ d \leftarrow a, \lnot c \]

\[ \begin{align*}
T & \emptyset \\
Ta & \\
Fb & \\
Tc & T \{ \lnot b, \lnot d \}
\end{align*} \]

\[ \begin{align*}
BTA & \ \\
BTB & \\
FFB & \\
(BTA) & F \{ a, \lnot c \}
\end{align*} \]

\[ \begin{align*}
BFA & \ \\
BFB & \\
FTB & \\
(BFA) & Td \\
(FTB) & T \{ a, \lnot c \}
\end{align*} \]
(Previewed) Example

\[
\begin{align*}
    a & \leftarrow \\
    c & \leftarrow \neg b, \neg d \\
    d & \leftarrow a, \neg c
\end{align*}
\]

\[
\begin{array}{llll}
    \text{(FTB)} & T \emptyset & F c \\
    \text{(FTA)} & \textcolor{red}{T} a & \text{BFA} & \textcolor{blue}{F} \{\neg b, \neg d\} \\
    \text{(FFA)} & \textcolor{red}{F} b & \text{BFB} & \textcolor{blue}{T} d \\
    \text{(Cut}[A(P)]\text{)} & \textcolor{red}{T} c & \text{FTB} & \text{FTB} \\
    \text{(BTA)} & \text{FTB} & \text{FFB} & \text{FTB} \\
    \text{(BTB)} & \text{FFB} & \text{FTB} & \text{FFB} \\
    \text{(FFB)} & \text{FTB} & \text{FFB} & \text{FTB}
\end{array}
\]
(Previewed) Example

\[
a \leftarrow \\
c \leftarrow \neg b, \neg d \\
d \leftarrow a, \neg c
\]

\[
T \emptyset \\
T a \\
F b
\]

\[
(Tc) \\
T \{\neg b, \neg d\} \\
F d \\
F \{a, \neg c\}
\]

\[
(Fc) \\
F \{\neg b, \neg d\} \\
T d \\
T \{a, \neg c\}
\]
(Previewed) Example

\[ a \leftarrow \]
\[ c \leftarrow \neg b, \neg d \]
\[ d \leftarrow a, \neg c \]

\[ T \emptyset \]
\[ T a \]
\[ F b \]

\[ F c \]

\[ T \{ \neg b, \neg d \} \]
\[ F d \]
\[ F \{ a, \neg c \} \]

\[ (BTA) \]
\[ (BTA) \]
\[ (FFA) \]

\[ (FTB) \]
\[ (FTA) \]
\[ (FFA) \]

\[ (Cut[A(P)]) \]
Tableau Calculi for ASP

(Previewed) Example

\[\begin{align*}
a & \leftarrow \\
c & \leftarrow \sim b, \sim d \\
d & \leftarrow a, \sim c
\end{align*}\]

\(\begin{align*}
T & \emptyset \\
T & a \\
F & b
\end{align*}\)

\(\begin{align*}
T & \{\sim b, \sim d\} \\
F & \{a, \sim c\}
\end{align*}\)

\(\begin{align*}
F & \{\sim b, \sim d\} \\
T & d
\end{align*}\)

\(\begin{align*}
T & \{a, \sim c\}
\end{align*}\)
(Previewed) Example

\[
\begin{align*}
  a & \leftarrow \\
  c & \leftarrow \neg b, \neg d \\
  d & \leftarrow a, \neg c
\end{align*}
\]

\[
\begin{align*}
  T & \emptyset \\
  T & a \\
  F & b \\
  T & c \\
  F & \{\neg b, \neg d\} \\
  F & \{a, \neg c\} \\
  F & c \\
  F & \{\neg b, \neg d\} \\
  T & d \\
  T & \{a, \neg c\}
\end{align*}
\]
Auxiliary definitions

For a literal \( l \), define conjugation functions \( t \) and \( f \) as follows:

\[
\begin{align*}
t(l) &= \begin{cases} 
  T & \text{if } l \text{ is an atom} \\
  F_a & \text{if } l = \sim a \text{ for an atom } a
\end{cases} \\
f(l) &= \begin{cases} 
  F & \text{if } l \text{ is an atom} \\
  T_a & \text{if } l = \sim a \text{ for an atom } a
\end{cases}
\end{align*}
\]

Examples: \( t a = T a \), \( f a = F a \), \( t \sim a = F a \), and \( f \sim a = T a \)
Auxiliary definitions

- For a literal \( l \), define conjugation functions \( t \) and \( f \) as follows:

\[
\begin{align*}
    t/l &= \begin{cases} 
    T/l & \text{if } l \text{ is an atom} \\
    F/a & \text{if } l = \neg a \text{ for an atom } a
    \end{cases} \\
    f/l &= \begin{cases} 
    F/l & \text{if } l \text{ is an atom} \\
    T/a & \text{if } l = \neg a \text{ for an atom } a
    \end{cases}
\end{align*}
\]

- Examples \( ta = Ta, fa = Fa, t\neg a = Fa, \) and \( f\neg a = Ta \)
Auxiliary definitions

- Some tableau rules require conditions for their application
- Such conditions are specified as provisos

\[
\frac{\text{prerequisites}}{\text{consequence}} \quad \text{(proviso)}
\]

proviso: some condition(s)

- Note All tableau rules given in the sequel are stable model preserving
Forward true body (FTB)

- Prerequisites: All of a body's literals are \textit{true}
- Consequence: The body is \textit{true}
- Tableau Rule FTB

\[
p \leftarrow l_1,\ldots,l_n \\
\text{t}_{1},\ldots,\text{t}_{n} \\
\text{T}\{l_1,\ldots,l_n\}
\]

Example

\[
a \leftarrow b, \sim c \\
\text{T}b \\
\text{F}c \\
\text{T}\{b, \sim c\}
\]
Forward true body (FTB)

- **Prerequisites** All of a body’s literals are true
- **Consequence** The body is true
- **Tableau Rule FTB**

\[
p \leftarrow l_1, \ldots, l_n
\]
\[
t_1, \ldots, t_n
\]
\[
T\{l_1, \ldots, l_n\}
\]

- **Example**

\[
a \leftarrow b, \neg c
\]
\[
Tb
\]
\[
Fc
\]
\[
T\{b, \neg c\}
\]
Backward false body (BFB)

- **Prerequisites**: A body is *false*, and all its literals except for one are *true*
- **Consequence**: The residual body literal is *false*
- **Tableau Rule BFB**

\[
\begin{align*}
F\{l_1, \ldots, l_i, \ldots, l_n\} \\
\quad \quad \quad t_{l_1}, \ldots, t_{l_{i-1}}, t_{l_{i+1}}, \ldots, t_{l_n} \\
\quad \quad \quad f/t_i
\end{align*}
\]

- **Examples**

\[
\begin{align*}
F\{b, \sim c\} & \quad Tc \\
\quad \quad \quad Tb \\
\quad \quad \quad Fb & \quad \quad \quad Fc \\
\end{align*}
\]
Backward false body (BFB)

- **Prerequisites** A body is false, and all its literals except for one are true.
- **Consequence** The residual body literal is false.
- **Tableau Rule BFB**

\[
\frac{F\{l_1, \ldots, l_i, \ldots, l_n\}}{t/1, \ldots, t/i-1, t/i+1, \ldots, t/n \quad f/l_i}
\]

- **Examples**

\[
\begin{align*}
F\{b, \lnot c\} & \quad F\{b, \lnot c\} \\
Tb & \quad Fb \\
Tc & \quad Fc
\end{align*}
\]
Forward false body (FFB)

- **Prerequisites** Some literal of a body is *false*
- **Consequence** The body is *false*
- **Tableau Rule FFB**

\[
p \leftarrow l_1, \ldots, l_i, \ldots, l_n
\]
\[f / i\]
\[
F\{l_1, \ldots, l_i, \ldots, l_n\}
\]

- **Examples**

\[
a \leftarrow b, \lnot c
\]
\[
F b
\]
\[
F\{b, \lnot c\}
\]
\[
a \leftarrow b, \lnot c
\]
\[
T c
\]
\[
F\{b, \lnot c\}\]
Forward false body (FFB)

- **Prerequisites** Some literal of a body is *false*
- **Consequence** The body is *false*
- **Tableau Rule** FFB

\[ p \leftarrow l_1, \ldots, l_i, \ldots, l_n \]
\[ \text{ff } l_i \]
\[ \text{F}\{l_1, \ldots, l_i, \ldots, l_n\} \]

- **Examples**

\[ a \leftarrow b, \sim c \]
\[ \text{F} b \]
\[ \text{F}\{b, \sim c\} \]
\[ a \leftarrow b, \sim c \]
\[ \text{T} c \]
\[ \text{F}\{b, \sim c\} \]
Backward true body (BTB)

- **Prerequisites** A body is *true*
- **Consequence** The body’s literals are *true*
- **Tableau Rule BTB**

$$
\begin{array}{c}
\text{T}\{l_1, \ldots, l_i, \ldots, l_n\} \\
\text{t}/i \\
\end{array}
$$

- **Examples**

$$
\begin{array}{c}
\text{T}\{b, \sim c\} \\
\text{T}\{b, \sim c\} \\
\text{T}b \\
\text{F}c \\
\end{array}
$$
Backward true body (BTB)

- Prerequisites *A body is true*
- Consequence *The body's literals are true*
- Tableau Rule BTB

\[ T\{l_1, \ldots, l_i, \ldots, l_n\} \]
\[ t/l_i \]

- Examples

\[ T\{b, \sim c\} \]
\[ Tb \]

\[ T\{b, \sim c\} \]
\[ Fc \]
Consider rule body $B = \{ l_1, \ldots, l_n \}$

- Rules FTB and BFB amount to implication

  $$ l_1 \land \cdots \land l_n \rightarrow B $$

- Rules FFB and BTB amount to implication

  $$ B \rightarrow l_1 \land \cdots \land l_n $$

- Together they yield

  $$ B \leftrightarrow l_1 \land \cdots \land l_n $$
Tableau rules for bodies

Consider rule body $B = \{l_1, \ldots, l_n\}$

- Rules FTB and BFB amount to implication
  \[ l_1 \land \cdots \land l_n \rightarrow B \]

- Rules FFB and BTB amount to implication
  \[ B \rightarrow l_1 \land \cdots \land l_n \]

- Together they yield
  \[ B \leftrightarrow l_1 \land \cdots \land l_n \]
Tableau rules for bodies

Consider rule body $B = \{l_1, \ldots, l_n\}$

- Rules FTB and BFB amount to implication
  \[ l_1 \land \cdots \land l_n \rightarrow B \]

- Rules FFB and BTB amount to implication
  \[ B \rightarrow l_1 \land \cdots \land l_n \]

- Together they yield
  \[ B \leftrightarrow l_1 \land \cdots \land l_n \]
Forward true atom (FTA)

- **Prerequisites** Some of an atom’s bodies is *true*
- **Consequence** The atom is *true*
- **Tableau Rule FTA**

\[
p \leftarrow l_1, \ldots, l_n \\
T \{l_1, \ldots, l_n\} \\
\hline
T \ p
\]

- **Examples**

\[
a \leftarrow b, \sim c \\
T \{b, \sim c\} \\
\hline
T \ a
\]

\[
a \leftarrow d, \sim e \\
T \{d, \sim e\} \\
\hline
T \ a
\]
Forward true atom (FTA)

- Prerequisites **Some of an atom’s bodies is true**
- Consequence **The atom is true**
- Tableau Rule FTA

\[ p \leftarrow l_1, \ldots, l_n \]
\[ T\{l_1, \ldots, l_n\} \]
\[ Tp \]

- Examples

\[ a \leftarrow b, \sim c \]
\[ T\{b, \sim c\} \]
\[ Ta \]

\[ a \leftarrow d, \sim e \]
\[ T\{d, \sim e\} \]
\[ Ta \]
Backward false atom (BFA)

- **Prerequisites**  
  An atom is *false*

- **Consequence**  
  The bodies of all rules with the atom as head are *false*

- **Tableau Rule BFA**

\[
p \leftarrow l_1, \ldots, l_n
\]
\[
F_p
\]
\[
F\{l_1, \ldots, l_n\}
\]

- **Examples**

\[
a \leftarrow b, \neg c
\]
\[
F_a
\]
\[
F\{b, \neg c\}
\]

\[
a \leftarrow d, \neg e
\]
\[
F_a
\]
\[
F\{d, \neg e\}
\]
Backward false atom (BFA)

- **Prerequisites** An atom is *false*
- **Consequence** The bodies of all rules with the atom as head are *false*
- **Tableau Rule BFA**

\[
p \leftarrow l_1, \ldots, l_n \\
\underline{F_p} \\
\underline{F\{l_1, \ldots, l_n\}}
\]

- **Examples**

\[
a \leftarrow b, \sim c \\
\underline{F_a} \\
\underline{F\{b, \sim c\}}
\]

\[
a \leftarrow d, \sim e \\
\underline{F_a} \\
\underline{F\{d, \sim e\}}
\]
Forward false atom (FFA)

- **Prerequisites** For some atom, the bodies of all rules with the atom as head are *false*
- **Consequence** The atom is *false*
- **Tableau Rule FFA**

\[
\frac{FB_1, \ldots, FB_m}{Fp} (B_P(p) = \{B_1, \ldots, B_m\})
\]

- **Example**

\[
\frac{F\{b, \sim c\} \quad F\{d, \sim e\}}{Fa} (B_P(a) = \{\{b, \sim c\}, \{d, \sim e\}\})
\]
Forward false atom (FFA)

- **Prerequisites** For some atom, the bodies of all rules with the atom as head are *false*
- **Consequence** The atom is *false*
- **Tableau Rule FFA**

\[
\frac{\text{FB}_1, \ldots, \text{FB}_m}{\text{FP}} \quad (\text{BP}(p) = \{B_1, \ldots, B_m\})
\]

- **Example**

\[
\frac{\text{F}\{b, \sim c\} \quad \text{F}\{d, \sim e\}}{\text{Fa}} \quad (\text{BP}(a) = \{\{b, \sim c\}, \{d, \sim e\}\})
\]
Backward true atom (BTA)

- **Prerequisites** An atom is *true*, and the bodies of all rules with the atom as head except for one are *false*
- **Consequence** The residual body is *true*
- **Tableau Rule BTA**

\[
\begin{align*}
&T_B_i \\
&\text{T}_p \\
&F_{B_1}, \ldots, F_{B_i-1}, F_{B_{i+1}}, \ldots, F_{B_m} \\
\hline
&\text{T}_B_i \\
&(B_P(p) = \{B_1, \ldots, B_m\})
\end{align*}
\]

- **Examples**

\[
\begin{align*}
&T_a \\
&F\{b, \sim c\} \\
&\text{T}\{d, \sim e\} \\
\hline
&(\ast)
\end{align*}
\]

\[
\begin{align*}
&T_a \\
&F\{d, \sim e\} \\
&\text{T}\{b, \sim c\} \\
\hline
&(\ast)
\end{align*}
\]

\[(\ast) \quad B_P(a) = \{\{b, \sim c\}, \{d, \sim e\}\}\]
Backward true atom (BTA)

- **Prerequisites** An atom is *true*, and the bodies of all rules with the atom as head except for one are *false*
- **Consequence** The residual body is *true*
- **Tableau Rule BTA**

\[
\frac{T_p}{F B_1, \ldots, F B_{i-1}, F B_{i+1}, \ldots, F B_m} \quad (B_P(p) = \{B_1, \ldots, B_m\})
\]

- **Examples**

\[
\frac{T_a}{F\{b, \sim c\}} \quad (\ast) \quad \frac{T_a}{F\{d, \sim e\}} \quad (\ast)
\]

\[
\frac{T}{\{d, \sim e\}} \quad (\ast) \quad B_P(a) = \{\{b, \sim c\}, \{d, \sim e\}\}
\]
Tableau Calculi for ASP

Tableau rules for atoms

Consider an atom $p$ such that $B_P(p) = \{B_1, \ldots, B_m\}$

- Rules FTA and BFA amount to implication

  \[
  B_1 \lor \cdots \lor B_m \rightarrow p
  \]

- Rules FFA and BTA amount to implication

  \[
  p \rightarrow B_1 \lor \cdots \lor B_m
  \]

- Together they yield

  \[
  p \iff B_1 \lor \cdots \lor B_m
  \]
Tableau rules for atoms

Consider an atom $p$ such that $B_P(p) = \{B_1, \ldots, B_m\}$

- Rules FTA and BFA amount to implication
  
  \[ B_1 \lor \cdots \lor B_m \rightarrow p \]

- Rules FFA and BTA amount to implication
  
  \[ p \rightarrow B_1 \lor \cdots \lor B_m \]

- Together they yield
  
  \[ p \iff B_1 \lor \cdots \lor B_m \]
Tableau Calculi for ASP

Consider an atom $p$ such that $B_P(p) = \{B_1, \ldots, B_m\}$

- Rules FTA and BFA amount to implication
  \[ B_1 \lor \cdots \lor B_m \rightarrow p \]

- Rules FFA and BTA amount to implication
  \[ p \rightarrow B_1 \lor \cdots \lor B_m \]

- Together they yield
  \[ p \leftrightarrow B_1 \lor \cdots \lor B_m \]
Let $P$ be a normal logic program

- The eight tableau rules introduced so far essentially provide (straightforward) inferences from $CF(P)$
Preliminaries for unfounded sets

Let $P$ be a normal logic program

- For $P' \subseteq P$, define the greatest unfounded set of $P$ wrt $P'$ as

  $U_P(P') = A(P) \setminus Cn((P')^\emptyset)$

- For a loop $L \in loop(P)$, define the external bodies of $L$ as

  $EB_P(L) = \{B(r) \mid r \in P, h(r) \in L, B(r)^+ \cap L = \emptyset\}$
Preliminaries for unfounded sets

Let $P$ be a normal logic program

- For $P' \subseteq P$, define the greatest unfounded set of $P$ wrt $P'$ as

$$U_P(P') = A(P) \setminus Cn((P')^\emptyset)$$

- For a loop $L \in \text{loop}(P)$, define the external bodies of $L$ as

$$EB_P(L) = \{ B(r) \mid r \in P, h(r) \in L, B(r)^+ \cap L = \emptyset \}$$
Well-founded negation (WFN)

- **Prerequisites** An atom is in the greatest unfounded set wrt rules whose bodies are *false*.
- **Consequence** The atom is *false*.
- **Tableau Rule WFN**

\[
\begin{align*}
\frac{F B_1, \ldots, F B_m}{F p} & \quad (p \in U_P(\{r \in P \mid B(r) \notin \{B_1, \ldots, B_m\}\}))
\end{align*}
\]

- **Examples**

\[
\begin{align*}
a & \leftarrow \sim b \\
F \{\sim b\} & \quad (\ast) \\
F a & \quad (\ast)
\end{align*}
\]

\[
\begin{align*}
a & \leftarrow a \\
F \{\sim b\} & \quad (\ast) \\
F a & \quad (\ast)
\end{align*}
\]

\[
\begin{align*}
(\ast) \quad a & \in U_P(P \setminus \{a \leftarrow \sim b\})
\end{align*}
\]
Well-founded negation (WFN)

- **Prerequisites**: An atom is in the greatest unfounded set wrt rules whose bodies are *false*

- **Consequence**: The atom is *false*

- **Tableau Rule WFN**

\[
\frac{FB_1, \ldots, FB_m}{Fp} \quad (p \in UP(\{r \in P \mid B(r) \not\in \{B_1, \ldots, B_m\}\}))
\]

- **Examples**

\[
\begin{align*}
a & \leftarrow \lnot b \\
F\{\lnot b\} & \quad (\ast) \\
F a & \quad (\ast)
\end{align*}
\]

\[
\begin{align*}
a & \leftarrow a \\
F\{\lnot b\} & \quad (\ast)
\end{align*}
\]

\[
\begin{align*}
a \in UP(P \setminus \{a \leftarrow \lnot b\})
\end{align*}
\]
Well-founded justification (WFJ)

- **Prerequisites** *A true* atom is in the greatest unfounded set wrt rules whose bodies are *false*, if a particular body is made *false*

- **Consequence** The respective body is *true*

- **Tableau Rule WFJ**

\[
\begin{array}{c}
T_p \\
FB_1, \ldots, FB_{i-1}, FB_{i+1}, \ldots, FB_m \\
TB_i
\end{array} \quad (p \in UP(\{r \in P \mid B(r) \notin \{B_1, \ldots, B_m\}\}))
\]

- **Examples**

\[
\begin{align*}
t & \leftarrow \sim b \\
T_a \quad & (\ast) \\
T_{\sim b} & \\
(t \{\sim b\}) & \\
\end{align*}
\]

\[
\begin{align*}
a & \leftarrow a \\
T_a \quad & (\ast) \\
T_{\sim b} & \\
(t \{\sim b\}) & \\
\end{align*}
\]

\[
\begin{align*}
(\ast) \quad a & \in UP(P \setminus \{a \leftarrow \sim b\})
\end{align*}
\]
Well-founded justification (WFJ)

- **Prerequisites** A *true* atom is in the greatest unfounded set wrt rules whose bodies are *false*, if a particular body is made *false*
- **Consequence** The respective body is *true*
- **Tableau Rule WFJ**

\[
\frac{T_p \quad F_{B_1}, \ldots, F_{B_{i-1}}, F_{B_{i+1}}, \ldots, F_{B_m}}{T_{B_i} \quad (p \in U_P(\{r \in P \mid B(r) \notin \{B_1, \ldots, B_m\}))}
\]

- **Examples**

\[
\begin{align*}
& a \leftarrow \neg b \\
&T_a \\
&T\{\neg b\} \quad (\ast)
\end{align*}
\]

\[
\begin{align*}
& a \leftarrow a \\
&T_a \\
&T\{\neg b\} \quad (\ast)
\end{align*}
\]

\[
(\ast) \quad a \in U_P(P \setminus \{a \leftarrow \neg b\})
\]
Well-founded tableau rules

- Tableau rules WFN and WFJ ensure non-circular support for *true* atoms

Note

1. WFN subsumes falsifying atoms via FFA,
2. WFJ can be viewed as “backward propagation” for unfounded sets,
3. WFJ subsumes backward propagation of *true* atoms via BTA
Well-founded tableau rules

- Tableau rules WFN and WFJ ensure non-circular support for *true* atoms

**Note**

1. WFN subsumes falsifying atoms via FFA,
2. WFJ can be viewed as “backward propagation” for unfounded sets,
3. WFJ subsumes backward propagation of *true* atoms via BTA
Relationship with well-founded operator

Let $P$ be a normal logic program, $\langle T, F \rangle$ a partial interpretation, and $P' = \{ r \in P \mid B(r)^+ \cap F = \emptyset \text{ and } B(r)^- \cap T = \emptyset \}$. The following conditions are equivalent:

1. $p \in U_P(\langle T, F \rangle)$
2. $p \in U_P(P')$

Hence, the well-founded operator $\Omega$ and WFN coincide.

Note: In contrast to $\Omega$, WFN does not necessarily require a rule body to contain a false literal for the rule being inapplicable.
Relationship with well-founded operator

Let $P$ be a normal logic program, $\langle T, F \rangle$ a partial interpretation, and $P' = \{ r \in P \mid B(r)^+ \cap F = \emptyset \text{ and } B(r)^- \cap T = \emptyset \}$.

The following conditions are equivalent

1. $p \in \bigcup_P \langle T, F \rangle$
2. $p \in \bigcup_P (P')$

Hence, the well-founded operator $\Omega$ and WFN coincide.

Note In contrast to $\Omega$, WFN does not necessarily require a rule body to contain a false literal for the rule being inapplicable.
Relationship with well-founded operator

Let \( P \) be a normal logic program, \( \langle T, F \rangle \) a partial interpretation, and
\[
P' = \{ r \in P \mid B(r)^+ \cap F = \emptyset \text{ and } B(r)^- \cap T = \emptyset \}.
\]

The following conditions are equivalent

1. \( p \in U_P \langle T, F \rangle \)
2. \( p \in U_P (P') \)

Hence, the well-founded operator \( \Omega \) and WFN coincide

Note In contrast to \( \Omega \), WFN does not necessarily require a rule body to contain a false literal for the rule being inapplicable
Relationship with well-founded operator

Let $P$ be a normal logic program, $\langle T, F \rangle$ a partial interpretation, and $P' = \{ r \in P \mid B(r)^+ \cap F = \emptyset \text{ and } B(r)^- \cap T = \emptyset \}$.

- The following conditions are equivalent
  1. $p \in U_P(\langle T, F \rangle)$
  2. $p \in U_P(P')$

- Hence, the well-founded operator $\Omega$ and WFN coincide

- Note In contrast to $\Omega$, WFN does not necessarily require a rule body to contain a false literal for the rule being inapplicable
Forward loop (FL)

- **Prerequisites** The external bodies of a loop are false
- **Consequence** The atoms in the loop are false
- **Tableau Rule FL**

\[
\frac{FB_1, \ldots, FB_m}{Fp} \quad (p \in L, L \in \text{loop}(P), EB_P(L) = \{B_1, \ldots, B_m\})
\]

- **Example**

\[
\begin{align*}
\text{a} & \leftarrow \text{a} \\
\text{a} & \leftarrow \neg \text{b} \\
\text{F} \{\neg \text{b}\} & \leftarrow \text{a} \\
& \quad (EB_P(\{\text{a}\}) = \{\{\neg \text{b}\}\})
\end{align*}
\]
Forward loop (FL)

- Prerequisites: The external bodies of a loop are \textit{false}.
- Consequence: The atoms in the loop are \textit{false}.
- Tableau Rule FL:

\[
\begin{align*}
\frac{F B_1, \ldots, F B_m}{F p} & \quad (p \in L, L \in \text{loop}(P), EB_P(L) = \{B_1, \ldots, B_m\})
\end{align*}
\]

- Example:

\[
\begin{align*}
a & \leftarrow a \\
a & \leftarrow \sim b \\
F \{\sim b\} & \quad (EB_P(\{a\}) = \{\{\sim b\}\})
\end{align*}
\]
Backward loop (BL)

- **Prerequisites** An atom of a loop is *true*, and all external bodies except for one are *false*
- **Consequence** The residual external body is *true*
- **Tableau Rule BL**

\[
\begin{align*}
T_p & \\
\text{FP} & = \{F B_1, \ldots, F B_{i-1}, F B_{i+1}, \ldots, F B_m\} \\
T B_i & \quad (p \in L, L \in \text{loop}(P), EB_P(L) = \{B_1, \ldots, B_m\})
\end{align*}
\]

- **Example**

\[
\begin{align*}
a & \leftarrow a \\
\text{a} & \leftarrow \neg b \\
Ta & \\
\text{T}\{\neg b\} & \quad (EB_P(\{a\}) = \{\{\neg b\}\})
\end{align*}
\]
Backward loop (BL)

- **Prerequisites** An atom of a loop is *true*, and all external bodies except for one are *false*.
- **Consequence** The residual external body is *true*.
- **Tableau Rule BL**

\[
\begin{align*}
T_p & \quad F B_{1}, \ldots, F B_{i-1}, F B_{i+1}, \ldots, F B_{m} \\
 & \quad T B_i \\
\end{align*}
\]

\( (p \in L, L \in \text{loop}(P), EB_{P}(L) = \{B_1, \ldots, B_m\}) \)

- **Example**

\[
\begin{align*}
a & \leftarrow a \\
a & \leftarrow \neg b \\
T_a & \quad (EB_P(\{a\}) = \{\{\neg b\}\})
\end{align*}
\]
Tableau rules for loops

- Tableau rules FL and BL ensure non-circular support for true atoms
  - For a loop \( L \) such that \( EB_P(L) = \{B_1, \ldots, B_m\} \), they amount to implications of form
    \[
    \bigvee_{p \in L} p \rightarrow B_1 \lor \cdots \lor B_m
    \]
  - Comparison to well-founded tableau rules yields
    - FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
    - BL cannot simulate inferences via WFJ
Tableau rules for loops

- Tableau rules FL and BL ensure non-circular support for *true* atoms.
- For a loop $L$ such that $EB_P(L) = \{B_1, \ldots, B_m\}$, they amount to implications of form

$$\bigvee_{p \in L} p \rightarrow B_1 \lor \cdots \lor B_m$$

- Comparison to well-founded tableau rules yields
  - FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
  - BL cannot simulate inferences via WFJ.
Tableau rules for loops

- Tableau rules FL and BL ensure non-circular support for *true* atoms
- For a loop $L$ such that $EB_P(L) = \{B_1, \ldots, B_m\}$, they amount to implications of form

  $$\bigvee_{p \in L} p \rightarrow B_1 \lor \cdots \lor B_m$$

- Comparison to well-founded tableau rules yields
  - FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
  - BL cannot simulate inferences via WFJ
Tableau Calculi for ASP

Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas

- In practice, ASP solvers such as *smodels* and *clasp*
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
Tableau Calculi for ASP

Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas
- In practice, ASP solvers such as smodels and clasp
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas

- In practice, ASP solvers such as smodels and clasp
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas

- In practice, ASP solvers such as *smodels* and *clasp*
  - Exploit strongly connected components of positive atom dependency graphs
    - Can be viewed as an interpolation of FL
  - Do not directly implement BL (and neither WFJ)
    - Probably difficult to do efficiently
  - Could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
Case analysis by *Cut*

- Up to now, all tableau rules are deterministic. That is, rules extend a single branch but cannot create sub-branches.
- In general, closing a branch leads to a partial assignment.
- Case analysis is done by $\text{Cut}[C]$ where $C \subseteq A(P) \cup B(P)$.

Case analysis by *Cut*

- Up to now, all tableau rules are deterministic
  That is, rules extend a single branch but cannot create sub-branches
- In general, closing a branch leads to a partial assignment
- Case analysis is done by $\text{Cut}[C]$ where $C \subseteq A(P) \cup B(P)$
Case analysis by \textit{Cut}

- Up to now, all tableau rules are deterministic. That is, rules extend a single branch but cannot create sub-branches.
- In general, closing a branch leads to a partial assignment.
- Case analysis is done by \textit{Cut}[C] where $C \subseteq A(P) \cup B(P)$.
Case analysis by *Cut*

- Prerequisites **None**
- Consequence **Two alternative (complementary) entries**
- Tableau Rule **Cut[C]**

\[
\begin{align*}
T_v & \mid F_v \quad (v \in C)
\end{align*}
\]

- Examples

\[
\begin{align*}
a & \leftarrow \sim b \\
b & \leftarrow \sim a \\
\hline
T_a & \mid F_a \\
(C = A(P))
\end{align*}
\]

\[
\begin{align*}
a & \leftarrow \sim b \\
b & \leftarrow \sim a \\
\hline
T\{\sim b\} & \mid F\{\sim b\} \\
(C = B(P))
\end{align*}
\]
Case analysis by *Cut*

- **Prerequisites** None
- **Consequence** Two alternative (complementary) entries
- **Tableau Rule** *Cut*[\(C\)]

\[
\begin{array}{c|c}
T_v & F_v \\
\end{array} \quad (v \in C)
\]

- **Examples**

\[
\begin{align*}
a & \leftarrow \neg b \\
b & \leftarrow \neg a \\
\hline
T_a & F_a \\
\end{align*}
\]

\((C = A(P))\)

\[
\begin{align*}
a & \leftarrow \neg b \\
b & \leftarrow \neg a \\
\hline
T\{\neg b\} & F\{\neg b\} \\
\end{align*}
\]

\((C = B(P))\)
Well-known tableau calculi

- Fitting’s operator $\Phi$ applies forward propagation without sophisticated unfounded set checks

$$T_\Phi = \{FTB, FTA, FFB, FFA\}$$

- Well-founded operator $\Omega$ replaces negation of single atoms with negation of unfounded sets

$$T_\Omega = \{FTB, FTA, FFB, WFN\}$$

- “Local” propagation via a program’s completion can be determined by elementary inferences on atoms and rule bodies

$$T_{\text{completion}} = \{FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA\}$$
Well-known tableau calculi

- Fitting’s operator $\Phi$ applies forward propagation without sophisticated unfounded set checks

\[ \mathcal{T}_\Phi = \{ FTB, FTA, FFB, FFA \} \]

- Well-founded operator $\Omega$ replaces negation of single atoms with negation of unfounded sets

\[ \mathcal{T}_\Omega = \{ FTB, FTA, FFB, WFN \} \]

- “Local” propagation via a program’s completion can be determined by elementary inferences on atoms and rule bodies

\[ \mathcal{T}_{\text{completion}} = \{ FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA \} \]
Well-known tableau calculi

- Fitting’s operator $\Phi$ applies forward propagation without sophisticated unfounded set checks
  \[
  \mathcal{T}_\Phi = \{FTB, FTA, FFB, FFA\}
  \]

- Well-founded operator $\Omega$ replaces negation of single atoms with negation of unfounded sets
  \[
  \mathcal{T}_\Omega = \{FTB, FTA, FFB, WFN\}
  \]

- “Local” propagation via a program’s completion can be determined by elementary inferences on atoms and rule bodies
  \[
  \mathcal{T}_{\text{completion}} = \{FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA\}
  \]
Tableau Calculi characterizing ASP solvers

Outline

38 Tableau Calculi
39 Tableau Calculi for ASP
40 Tableau Calculi characterizing ASP solvers
41 Proof complexity
Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing

\[
\begin{align*}
T_{cmodels-1} &= T_{completion} \cup \{\text{Cut}[A(P) \cup B(P)]\} \\
T_{assat} &= T_{completion} \cup \{\text{FL}\} \cup \{\text{Cut}[A(P) \cup B(P)]\} \\
T_{smodels} &= T_{completion} \cup \{\text{WFN}\} \cup \{\text{Cut}[A(P)]\} \\
T_{noMoRe} &= T_{completion} \cup \{\text{WFN}\} \cup \{\text{Cut}[B(P)]\} \\
T_{nomore++} &= T_{completion} \cup \{\text{WFN}\} \cup \{\text{Cut}[A(P) \cup B(P)]\}
\end{align*}
\]

- SAT-based ASP solvers, assat and cmodels, incrementally add loop formulas to a program’s completion
- Native ASP solvers, smodels, dlv, noMoRe, and nomore++, essentially differ only in their Cut rules
Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing

\[ T_{\text{cmodels-1}} = T_{\text{completion}} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \]
\[ T_{\text{assat}} = T_{\text{completion}} \cup \{ \text{FL} \} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \]
\[ T_{\text{smodels}} = T_{\text{completion}} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[A(P)] \} \]
\[ T_{\text{noMoRe}} = T_{\text{completion}} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[B(P)] \} \]
\[ T_{\text{nomore}++} = T_{\text{completion}} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \]

- SAT-based ASP solvers, \textit{assat} and \textit{cmodels}, incrementally add loop formulas to a program’s completion
- Native ASP solvers, \textit{smodels}, \textit{dlv}, \textit{noMoRe}, and \textit{nomore}++, essentially differ only in their \textit{Cut} rules
ASP solvers combine propagation with case analysis

We obtain the following tableau calculi characterizing

\[ T_{cmodels-1} = T_{completion} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \]
\[ T_{assat} = T_{completion} \cup \{ \text{FL} \} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \]
\[ T_{smodels} = T_{completion} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[A(P)] \} \]
\[ T_{noMoRe} = T_{completion} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[B(P)] \} \]
\[ T_{nomore++} = T_{completion} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \]

SAT-based ASP solvers, assat and cmodels, incrementally add loop formulas to a program’s completion

Native ASP solvers, smodels, dlv, noMoRe, and nomore++, essentially differ only in their Cut rules
Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing
  \[
  \begin{align*}
  \mathcal{T}_{\text{cmodels-1}} &= \mathcal{T}_{\text{completion}} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \\
  \mathcal{T}_{\text{assat}} &= \mathcal{T}_{\text{completion}} \cup \{ \text{FL} \} \cup \{ \text{Cut}[A(P) \cup B(P)] \} \\
  \mathcal{T}_{\text{smodels}} &= \mathcal{T}_{\text{completion}} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[A(P)] \} \\
  \mathcal{T}_{\text{noMoRe}} &= \mathcal{T}_{\text{completion}} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[B(P)] \} \\
  \mathcal{T}_{\text{nomore}++} &= \mathcal{T}_{\text{completion}} \cup \{ \text{WFN} \} \cup \{ \text{Cut}[A(P) \cup B(P)] \}
  \end{align*}
\]
- SAT-based ASP solvers, assat and cmodels, incrementally add loop formulas to a program’s completion
- Native ASP solvers, smodels, dlv, noMoRe, and nomore++, essentially differ only in their Cut rules
Outline

38 Tableau Calculi
39 Tableau Calculi for ASP
40 Tableau Calculi characterizing ASP solvers
41 Proof complexity
Proof complexity

- **Proof complexity** is used for describing the relative efficiency of different proof systems
  - It compares proof systems based on minimal refutations
  - It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$
  - Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$
- The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient!
Proof complexity

- **Proof complexity** is used for describing the relative efficiency of different proof systems.
  - It compares proof systems based on **minimal refutations**.
  - It is independent of heuristics.

- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$.
  Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$.

- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$.

- The size of tableaux is simply the number of their entries.

- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient!
Proof complexity

- **Proof complexity** is used for describing the relative efficiency of different proof systems
  - It compares proof systems based on minimal refutations
  - It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$
  - Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$
- The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient!
Proof complexity

Proof complexity is used for describing the relative efficiency of different proof systems. It compares proof systems based on minimal refutations. It is independent of heuristics.

A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$. Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$.

For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$.

The size of tableaux is simply the number of their entries.

We do not need to know the precise number of entries: Counting required Cut applications is sufficient!
Proof complexity

- **Proof complexity** is used for describing the relative efficiency of different proof systems.
  - It compares proof systems based on minimal refutations.
  - It is independent of heuristics.

- A proof system $\mathcal{T}$ **polynomially simulates** a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$.
  - Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$.

- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite **witnessing family** of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$.
  - The size of tableaux is simply the number of their entries.

- We do not need to know the precise number of entries: Counting required **Cut** applications is sufficient!
**Proof complexity**

### $T_{smodels}$ versus $T_{noMoRe}$

- $T_{smodels}$ restricts $Cut$ to $A(P)$ and $T_{noMoRe}$ to $B(P)$

  *Are both approaches similar or is one of them superior to the other?*

- Let $\{P^n_a\}$, $\{P^n_b\}$, and $\{P^n_c\}$ be infinite families of programs where

  \[
  P^n_a = \begin{cases}
  x \leftarrow \sim x \\
  x \leftarrow a_1, b_1 \\
  \vdots \\
  x \leftarrow a_n, b_n
  \end{cases}
  \]

  \[
  P^n_b = \begin{cases}
  x \leftarrow c_1, \ldots, c_n, \sim x \\
  c_1 \leftarrow a_1 \\
  c_1 \leftarrow b_1 \\
  \vdots \\
  c_n \leftarrow a_n, c_n \leftarrow b_n
  \end{cases}
  \]

  \[
  P^n_c = \begin{cases}
  a_1 \leftarrow \sim b_1 \\
  b_1 \leftarrow \sim a_1 \\
  \vdots \\
  a_n \leftarrow \sim b_n \\
  b_n \leftarrow \sim a_n
  \end{cases}
  \]

- In minimal refutations for $P^n_a \cup P^n_c$, the number of applications of $Cut[A(P^n_a \cup P^n_c)]$ with $T_{noMoRe}$ is linear in $n$, whereas $T_{smodels}$ requires exponentially many applications of $Cut[A(P^n_a \cup P^n_c)]$

- Vice versa, minimal refutations for $P^n_b \cup P^n_c$ require linearly many applications of $Cut[A(P^n_b \cup P^n_c)]$ with $T_{smodels}$ and exponentially many applications of $Cut[B(P^n_b \cup P^n_c)]$ with $T_{noMoRe}$
Proof complexity

\[ \mathcal{T}_{\text{smodels}} \text{ versus } \mathcal{T}_{\text{noMoRe}} \]

- \( \mathcal{T}_{\text{smodels}} \) restricts \( \text{Cut} \) to \( A(P) \) and \( \mathcal{T}_{\text{noMoRe}} \) to \( B(P) \).
- Are both approaches similar or is one of them superior to the other?

- Let \( \{P_a^n\}, \{P_b^n\}, \) and \( \{P_c^n\} \) be infinite families of programs where

\[
\begin{align*}
P_a^n &= \left\{ \begin{array}{l} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \\
P_b^n &= \left\{ \begin{array}{l} x \leftarrow c_1, \ldots, c_n, \sim x \\ c_1 \leftarrow a_1, c_1 \leftarrow b_1 \\ \vdots \\ c_n \leftarrow a_n, c_n \leftarrow b_n \end{array} \right\} \\
P_c^n &= \left\{ \begin{array}{l} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}
\end{align*}
\]

- In minimal refutations for \( P_a^n \cup P_c^n \), the number of applications of \( \text{Cut}[B(P_a^n \cup P_c^n)] \) with \( \mathcal{T}_{\text{noMoRe}} \) is linear in \( n \), whereas \( \mathcal{T}_{\text{smodels}} \) requires exponentially many applications of \( \text{Cut}[A(P_a^n \cup P_c^n)] \).
- Vice versa, minimal refutations for \( P_b^n \cup P_c^n \) require linearly many applications of \( \text{Cut}[A(P_b^n \cup P_c^n)] \) with \( \mathcal{T}_{\text{smodels}} \) and exponentially many applications of \( \text{Cut}[B(P_b^n \cup P_c^n)] \) with \( \mathcal{T}_{\text{noMoRe}} \).
Proof complexity

\( T_{\text{smodels}} \) versus \( T_{\text{noMoRe}} \)

- \( T_{\text{smodels}} \) restricts \( \text{Cut} \) to \( A(P) \) and \( T_{\text{noMoRe}} \) to \( B(P) \)
  
  Are both approaches similar or is one of them superior to the other?

- Let \( \{P^n_a\}, \{P^n_b\}, \text{ and } \{P^n_c\} \) be infinite families of programs where

  \[
  P^n_a = \begin{cases} 
  x \leftarrow \sim x \\
  x \leftarrow a_1, b_1 \\
  \vdots \\
  x \leftarrow a_n, b_n 
  \end{cases}
  \]

  \[
  P^n_b = \begin{cases} 
  x \leftarrow c_1, \ldots, c_n, \sim x \\
  c_1 \leftarrow a_1 \\
  c_1 \leftarrow b_1 \\
  \vdots \\
  c_n \leftarrow a_n \\
  c_n \leftarrow b_n 
  \end{cases}
  \]

  \[
  P^n_c = \begin{cases} 
  a_1 \leftarrow \sim b_1 \\
  b_1 \leftarrow \sim a_1 \\
  \vdots \\
  a_n \leftarrow \sim b_n \\
  b_n \leftarrow \sim a_n 
  \end{cases}
  \]

- In minimal refutations for \( P^n_a \cup P^n_c \), the number of applications of \( \text{Cut}[B(P^n_a \cup P^n_c)] \) with \( T_{\text{noMoRe}} \) is linear in \( n \), whereas \( T_{\text{smodels}} \) requires exponentially many applications of \( \text{Cut}[A(P^n_a \cup P^n_c)] \)

- Vice versa, minimal refutations for \( P^n_b \cup P^n_c \) require linearly many applications of \( \text{Cut}[A(P^n_b \cup P^n_c)] \) with \( T_{\text{smodels}} \) and exponentially many applications of \( \text{Cut}[B(P^n_b \cup P^n_c)] \) with \( T_{\text{noMoRe}} \)
Proof complexity

\( T_{\text{smodels}} \) versus \( T_{\text{noMoRe}} \)

- \( T_{\text{smodels}} \) restricts \( \text{Cut} \) to \( A(P) \) and \( T_{\text{noMoRe}} \) to \( B(P) \)
  - Are both approaches similar or is one of them superior to the other?

- Let \( \{P^n_a\}, \{P^n_b\}, \text{ and } \{P^n_c\} \) be infinite families of programs where

  \[
  P^n_a = \{ \begin{array}{l}
  x \gets \neg x \\
  x \gets a_1, b_1 \\
  \vdots \\
  x \gets a_n, b_n
  \end{array} \}
  \]

  \[
  P^n_b = \{ \begin{array}{l}
  x \gets c_1, \ldots, c_n, \neg x \\
  c_1 \gets a_1 \\
  c_1 \gets b_1 \\
  \vdots \\
  c_n \gets a_n \\
  c_n \gets b_n
  \end{array} \}
  \]

  \[
  P^n_c = \{ \begin{array}{l}
  a_1 \gets \neg b_1 \\
  b_1 \gets \neg a_1 \\
  \vdots \\
  a_n \gets \neg b_n \\
  b_n \gets \neg a_n
  \end{array} \}
  \]

- In minimal refutations for \( P^n_a \cup P^n_c \), the number of applications of \( \text{Cut}[B(P^n_a \cup P^n_c)] \) with \( T_{\text{noMoRe}} \) is linear in \( n \), whereas \( T_{\text{smodels}} \) requires exponentially many applications of \( \text{Cut}[A(P^n_a \cup P^n_c)] \)

- Vice versa, minimal refutations for \( P^n_b \cup P^n_c \) require linearly many applications of \( \text{Cut}[A(P^n_b \cup P^n_c)] \) with \( T_{\text{smodels}} \) and exponentially many applications of \( \text{Cut}[B(P^n_b \cup P^n_c)] \) with \( T_{\text{noMoRe}} \).
\[ \mathcal{T}_{\text{smmodels}} \text{ versus } \mathcal{T}_{\text{noMoRe}} \]

- \( \mathcal{T}_{\text{smmodels}} \) restricts Cut to \( A(P) \) and \( \mathcal{T}_{\text{noMoRe}} \) to \( B(P) \).
  - Are both approaches similar or is one of them superior to the other?

- Let \( \{P^n_a\}, \{P^n_b\}, \) and \( \{P^n_c\} \) be infinite families of programs where
  
  \[
  P^n_a = \begin{cases} 
  x \leftarrow \sim x \\
  x \leftarrow a_1, b_1 \\
  \vdots \\
  x \leftarrow a_n, b_n 
  \end{cases}
  \]

  \[
  P^n_b = \begin{cases} 
  x \leftarrow c_1, \ldots, c_n, \sim x \\
  c_1 \leftarrow a_1 \\
  c_1 \leftarrow b_1 \\
  \vdots \\
  c_n \leftarrow a_n \\
  c_n \leftarrow b_n 
  \end{cases}
  \]

  \[
  P^n_c = \begin{cases} 
  a_1 \leftarrow \sim b_1 \\
  b_1 \leftarrow \sim a_1 \\
  \vdots \\
  a_n \leftarrow \sim b_n \\
  b_n \leftarrow \sim a_n 
  \end{cases}
  \]

- In minimal refutations for \( P^n_a \cup P^n_c \), the number of applications of Cut\( [B(P^n_a \cup P^n_c)] \) with \( \mathcal{T}_{\text{noMoRe}} \) is linear in \( n \), whereas \( \mathcal{T}_{\text{smmodels}} \) requires exponentially many applications of Cut\( [A(P^n_a \cup P^n_c)] \).

- Vice versa, minimal refutations for \( P^n_b \cup P^n_c \) require linearly many applications of Cut\( [A(P^n_b \cup P^n_c)] \) with \( \mathcal{T}_{\text{smmodels}} \) and exponentially many applications of Cut\( [B(P^n_b \cup P^n_c)] \) with \( \mathcal{T}_{\text{noMoRe}} \).
Proof complexity

\textbf{\(\mathcal{T}_{smodels} \text{ versus } \mathcal{T}_{noMoRe}\)}

- \(\mathcal{T}_{smodels}\) restricts \textit{Cut} to \(A(P)\) and \(\mathcal{T}_{noMoRe}\) to \(B(P)\)
  
  Are both approaches similar or is one of them superior to the other?

- Let \(\{P^n_a\}, \{P^n_b\},\) and \(\{P^n_c\}\) be infinite families of programs where

\[
\begin{align*}
  P^n_a &= \left\{ \begin{array}{l}
  x \leftarrow \sim x \\
  x \leftarrow a_1, b_1 \\
  \vdots \\
  x \leftarrow a_n, b_n 
  \end{array} \right\} \\
  P^n_b &= \left\{ \begin{array}{l}
  x \leftarrow c_1, \ldots, c_n, \sim x \\
  c_1 \leftarrow a_1 \\
  c_1 \leftarrow b_1 \\
  \vdots \\
  c_n \leftarrow a_n \\
  c_n \leftarrow b_n 
  \end{array} \right\} \\
  P^n_c &= \left\{ \begin{array}{l}
  a_1 \leftarrow \sim b_1 \\
  b_1 \leftarrow \sim a_1 \\
  \vdots \\
  a_n \leftarrow \sim b_n \\
  b_n \leftarrow \sim a_n 
  \end{array} \right\}
\end{align*}
\]

- In minimal refutations for \(P^n_a \cup P^n_c\), the number of applications of \(\text{Cut}[B(P^n_a \cup P^n_c)]\) with \(\mathcal{T}_{noMoRe}\) is linear in \(n\), whereas \(\mathcal{T}_{smodels}\) requires exponentially many applications of \(\text{Cut}[A(P^n_a \cup P^n_c)]\)

- Vice versa, minimal refutations for \(P^n_b \cup P^n_c\) require linearly many applications of \(\text{Cut}[A(P^n_b \cup P^n_c)]\) with \(\mathcal{T}_{smodels}\) and exponentially many applications of \(\text{Cut}[B(P^n_b \cup P^n_c)]\) with \(\mathcal{T}_{noMoRe}\).
\( \mathcal{T}_{\text{smodels}} \) versus \( \mathcal{T}_{\text{noMoRe}} \)

- \( \mathcal{T}_{\text{smodels}} \) restricts \( \text{Cut} \) to \( A(P) \) and \( \mathcal{T}_{\text{noMoRe}} \) to \( B(P) \)
  Are both approaches similar or is one of them superior to the other?

- Let \( \{ P^n_a \} \), \( \{ P^n_b \} \), and \( \{ P^n_c \} \) be infinite families of programs where

  \[
  P^n_a = \begin{cases} 
  x \leftarrow x \\
  x \leftarrow a_1, b_1 \\
  \vdots \\
  x \leftarrow a_n, b_n 
  \end{cases}
  \]

  \[
  P^n_b = \begin{cases} 
  x \leftarrow c_1, \ldots, c_n, \sim x \\
  c_1 \leftarrow a_1 \\
  c_1 \leftarrow b_1 \\
  \vdots \\
  c_n \leftarrow a_n \\
  c_n \leftarrow b_n 
  \end{cases}
  \]

  \[
  P^n_c = \begin{cases} 
  a_1 \leftarrow \sim b_1 \\
  b_1 \leftarrow \sim a_1 \\
  \vdots \\
  a_n \leftarrow \sim b_n \\
  b_n \leftarrow \sim a_n 
  \end{cases}
  \]

- In minimal refutations for \( P^n_a \cup P^n_c \), the number of applications of \( \text{Cut}[B(P^n_a \cup P^n_c)] \) with \( \mathcal{T}_{\text{noMoRe}} \) is linear in \( n \), whereas \( \mathcal{T}_{\text{smodels}} \) requires exponentially many applications of \( \text{Cut}[A(P^n_a \cup P^n_c)] \)

- Vice versa, minimal refutations for \( P^n_b \cup P^n_c \) require linearly many applications of \( \text{Cut}[A(P^n_b \cup P^n_c)] \) with \( \mathcal{T}_{\text{smodels}} \) and exponentially many applications of \( \text{Cut}[B(P^n_b \cup P^n_c)] \) with \( \mathcal{T}_{\text{noMoRe}} \)
\( \mathcal{T}_{\text{smodels}} \) versus \( \mathcal{T}_{\text{noMoRe}} \)

- \( \mathcal{T}_{\text{smodels}} \) restricts \textit{Cut} to \( A(P) \) and \( \mathcal{T}_{\text{noMoRe}} \) to \( B(P) \)
  - Are both approaches similar or is one of them superior to the other?

- Let \( \{P^n_a\}, \{P^n_b\}, \) and \( \{P^n_c\} \) be infinite families of programs where

\[
P^n_a = \begin{cases} 
  x \leftarrow \sim x \\
  x \leftarrow a_1, b_1 \\
  \vdots \\
  x \leftarrow a_n, b_n
\end{cases}
\]

\[
P^n_b = \begin{cases} 
  x \leftarrow c_1, \ldots, c_n, \sim x \\
  c_1 \leftarrow a_1 \\
  c_1 \leftarrow b_1 \\
  \vdots \\
  c_n \leftarrow a_n \\
  c_n \leftarrow b_n
\end{cases}
\]

\[
P^n_c = \begin{cases} 
  a_1 \leftarrow \sim b_1 \\
  b_1 \leftarrow \sim a_1 \\
  \vdots \\
  a_n \leftarrow \sim b_n \\
  b_n \leftarrow \sim a_n
\end{cases}
\]

- In minimal refutations for \( P^n_a \cup P^n_c \), the number of applications of \textit{Cut}[B(P^n_a \cup P^n_c)] with \( \mathcal{T}_{\text{noMoRe}} \) is linear in \( n \), whereas \( \mathcal{T}_{\text{smodels}} \) requires exponentially many applications of \textit{Cut}[A(P^n_a \cup P^n_c)]

- Vice versa, minimal refutations for \( P^n_b \cup P^n_c \) require \textit{linearly} many applications of \textit{Cut}[A(P^n_b \cup P^n_c)] with \( \mathcal{T}_{\text{smodels}} \) and exponentially many applications of \textit{Cut}[B(P^n_b \cup P^n_c)] with \( \mathcal{T}_{\text{noMoRe}} \)
Relative efficiency

- As witnessed by \( \{ P^n_a \cup P^n_c \} \) and \( \{ P^n_b \cup P^n_c \} \), respectively, \( T_{\text{smodels}} \) and \( T_{\text{noMoRe}} \) do not polynomially simulate one another.

- Any refutation of \( T_{\text{smodels}} \) or \( T_{\text{noMoRe}} \) is a refutation of \( T_{\text{nomore}++} \) (but not vice versa).

- Hence:
  - both \( T_{\text{smodels}} \) and \( T_{\text{noMoRe}} \) are polynomially simulated by \( T_{\text{nomore}++} \) and
  - \( T_{\text{nomore}++} \) is polynomially simulated by neither \( T_{\text{smodels}} \) nor \( T_{\text{noMoRe}} \).

- More generally, the proof system obtained with \( \text{Cut}[A(P) \cup B(P)] \) is exponentially stronger than the ones with either \( \text{Cut}[A(P)] \) or \( \text{Cut}[B(P)] \).

- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers.
Relative efficiency

As witnessed by \(\{P_a^n \cup P_c^n\}\) and \(\{P_b^n \cup P_c^n\}\), respectively, \(T_{smodels}\) and \(T_{noMoRe}\) do not polynomially simulate one another.

Any refutation of \(T_{smodels}\) or \(T_{noMoRe}\) is a refutation of \(T_{nomore}^{++}\) (but not vice versa).

Hence

- both \(T_{smodels}\) and \(T_{noMoRe}\) are polynomially simulated by \(T_{nomore}^{++}\) and
- \(T_{nomore}^{++}\) is polynomially simulated by neither \(T_{smodels}\) nor \(T_{noMoRe}\)

More generally, the proof system obtained with \(Cut[A(P) \cup B(P)]\) is exponentially stronger than the ones with either \(Cut[A(P)]\) or \(Cut[B(P)]\).

Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers.
Proof complexity

Relative efficiency

- As witnessed by \( \{P^a_n \cup P^b_n\} \) and \( \{P^c_n \cup P^c_n\} \), respectively, \( T_{\text{smodels}} \) and \( T_{\text{noMoRe}} \) do not polynomially simulate one another.

- Any refutation of \( T_{\text{smodels}} \) or \( T_{\text{noMoRe}} \) is a refutation of \( T_{\text{nomore}^{++}} \) (but not vice versa).

- Hence
  - both \( T_{\text{smodels}} \) and \( T_{\text{noMoRe}} \) are polynomially simulated by \( T_{\text{nomore}^{++}} \) and
  - \( T_{\text{nomore}^{++}} \) is polynomially simulated by neither \( T_{\text{smodels}} \) nor \( T_{\text{noMoRe}} \).

- More generally, the proof system obtained with \( \text{Cut}[A(P) \cup B(P)] \) is exponentially stronger than the ones with either \( \text{Cut}[A(P)] \) or \( \text{Cut}[B(P)] \).

- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers.
Relative efficiency

- As witnessed by \( \{P^n_a \cup P^n_c\} \) and \( \{P^n_b \cup P^n_c\} \), respectively, \( \mathcal{T}_{\text{smo}dels} \) and \( \mathcal{T}_{\text{noMo}Re} \) do not polynomially simulate one another.
- Any refutation of \( \mathcal{T}_{\text{smo}dels} \) or \( \mathcal{T}_{\text{noMo}Re} \) is a refutation of \( \mathcal{T}_{\text{nomo}re^{++}} \) (but not vice versa).
- Hence:
  - both \( \mathcal{T}_{\text{smo}dels} \) and \( \mathcal{T}_{\text{noMo}Re} \) are polynomially simulated by \( \mathcal{T}_{\text{nomo}re^{++}} \) and
  - \( \mathcal{T}_{\text{nomo}re^{++}} \) is polynomially simulated by neither \( \mathcal{T}_{\text{smo}dels} \) nor \( \mathcal{T}_{\text{noMo}Re} \)

- More generally, the proof system obtained with \( \text{Cut}[A(P) \cup B(P)] \) is exponentially stronger than the ones with either \( \text{Cut}[A(P)] \) or \( \text{Cut}[B(P)] \).
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers.
Relative efficiency

- As witnessed by \( \{P^n_a \cup P^n_c\} \) and \( \{P^n_b \cup P^n_c\} \), respectively, \( T_{smodels} \) and \( T_{noMoRe} \) do not polynomially simulate one another.
- Any refutation of \( T_{smodels} \) or \( T_{noMoRe} \) is a refutation of \( T_{nomore}^{++} \) (but not vice versa).
- Hence
  - both \( T_{smodels} \) and \( T_{noMoRe} \) are polynomially simulated by \( T_{nomore}^{++} \)
  - \( T_{nomore}^{++} \) is polynomially simulated by neither \( T_{smodels} \) nor \( T_{noMoRe} \)

- More generally, the proof system obtained with \( Cut[A(P) \cup B(P)] \) is exponentially stronger than the ones with either \( Cut[A(P)] \) or \( Cut[B(P)] \).
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers.
Proper complexity

\( T_{\text{smodels}}: \) Example tableau

\[(r_1) \quad a \leftarrow \neg b \quad (r_2) \quad b \leftarrow d, \neg a \quad (r_3) \quad c \leftarrow b, d \]

\[(r_4) \quad c \leftarrow g \quad (r_5) \quad d \leftarrow c \quad (r_6) \quad d \leftarrow g \]

\[(r_7) \quad e \leftarrow f, \neg c \quad (r_8) \quad f \leftarrow \neg g \quad (r_9) \quad g \leftarrow \neg a, \neg f \]

\begin{align*}
(1) & \quad T_a & \text{[Cut]} \\
(2) & \quad T\{\neg b\} & \text{[BTA: } r_1, 1]\text{]}
\quad F_b & \text{[BTB: 2]} \\
(4) & \quad \text{F}\{d, \neg a\} & \text{[BFA: } r_2, 3]\text{]}
\quad \text{F}\{\neg a, \neg f\} & \text{[FFB: } r_9, 1]\text{]}
\quad \text{F}_g & \text{[FFA: } r_9, 5]\text{]}
\quad \text{T}\{\neg g\} & \text{[FTB: } r_8, 6]\text{]}
\quad \text{T}_f & \text{[FTA: } r_3, 7]\text{]}
\quad \text{F}\{b, d\} & \text{[FFB: } r_3, 3]\text{]}
\quad \text{F}\{g\} & \text{[FFB: } r_4, r_6, 6]\text{]}
\quad \text{F}_c & \text{[FFA: } r_3, r_4, 9, 10]\text{]}
\quad \text{F}\{c\} & \text{[FFB: } r_5, 11]\text{]}
\quad \text{F}_d & \text{[FFA: } r_5, r_6, 10, 12]\text{]}
\quad \text{T}\{f, \neg c\} & \text{[FTB: } r_7, 8, 11]\text{]}
\quad \text{T}_e & \text{[FTA: } r_7, 14]\text{]}
\end{align*}

\begin{align*}
(16) & \quad \text{F}_a & \text{[Cut]} \\
(17) & \quad \text{F}\{\neg b\} & \text{[BFA: } r_1, 16]\text{]}
\quad \text{T}_b & \text{[BFB: 17]} \\
(18) & \quad \text{T}\{d, \neg a\} & \text{[BTA: } r_2, 18]\text{]}
\quad \text{T}_d & \text{[BTB: 19]} \\
(19) & \quad \text{T}\{b, d\} & \text{[FBF: } r_3, 18, 20]\text{]}
\quad \text{T}_c & \text{[FTA: } r_3, 21]\text{]}
\quad \text{F}\{f, \neg c\} & \text{[FFA: } r_7, 22]\text{]}
\quad \text{F}_e & \text{[FFA: } r_7, 23]\text{]}
\quad \text{T}\{c\} & \text{[FTB: } r_5, 22]\text{]}
\end{align*}

\begin{align*}
(26) & \quad \text{T}_f & \text{[Cut]} \\
(27) & \quad \text{F}\{\neg a, \neg f\} & \text{[FFB: } r_9, 26]\text{]}
\quad \text{F}_c & \text{[WFN: 27]} \\
(28) & \quad \text{F}_f & \text{[Cut]} \\
\end{align*}

\begin{align*}
(29) & \quad \text{T}\{\neg a, \neg f\} & \text{[Cut]} \\
(30) & \quad \text{T}\{\neg a, \neg f\} & \text{[Cut]} \\
\end{align*}
\( T_{\text{noMoRe}}: \) Example tableau

\[(r_1) \quad a \leftarrow \neg b\]
\[(r_4) \quad c \leftarrow g\]
\[(r_7) \quad e \leftarrow f, \neg c\]
\[(r_2) \quad b \leftarrow d, \neg a\]
\[(r_5) \quad d \leftarrow c\]
\[(r_8) \quad f \leftarrow \neg g\]
\[(r_3) \quad c \leftarrow b, d\]
\[(r_6) \quad d \leftarrow g\]
\[(r_9) \quad g \leftarrow \neg a, \neg f\]

\( T_{\text{noMoRe}}: \) Example tableau

\[(1) \quad T\{\neg b\} \quad [\text{Cut}]\]
\[(2) \quad T a \quad [\text{FTA: } r_1, 1]\]
\[(3) \quad F b \quad [\text{BTB: } 1]\]
\[(4) \quad F\{d, \neg a\} \quad [\text{BFA: } r_2, 3]\]
\[(5) \quad F\{\neg a, \neg f\} \quad [\text{FFB: } r_9, 2]\]
\[(6) \quad F g \quad [\text{FFA: } r_9, 5]\]
\[(7) \quad T\{\neg g\} \quad [\text{FTB: } r_8, 6]\]
\[(8) \quad T f \quad [\text{FTA: } r_3, 7]\]
\[(9) \quad F\{b, d\} \quad [\text{FFB: } r_4, r_6, 6]\]
\[(10) \quad F\{g\} \quad [\text{FFA: } r_3, r_4, 9, 10]\]
\[(11) \quad F c \quad [\text{FFA: } r_5, 11]\]
\[(12) \quad F\{c\} \quad [\text{FFB: } r_5, r_6, 10, 12]\]
\[(13) \quad F d \quad [\text{FFA: } r_5, r_6, 10, 12]\]
\[(14) \quad T\{f, \neg c\} \quad [\text{FTB: } r_7, 8, 11]\]
\[(15) \quad T e \quad [\text{FTA: } r_7, 14]\]
\[(16) \quad F\{\neg b\} \quad [\text{Cut}]\]
\[(17) \quad F a \quad [\text{FFA: } r_1, 16]\]
\[(18) \quad T b \quad [\text{BBF: } 16]\]
\[(19) \quad T\{d, \neg a\} \quad [\text{BTA: } r_2, 18]\]
\[(20) \quad T d \quad [\text{BTB: } 19]\]
\[(21) \quad T\{b, d\} \quad [\text{FTB: } r_3, 18, 20]\]
\[(22) \quad T c \quad [\text{FTA: } r_3, 21]\]
\[(23) \quad F\{f, \neg c\} \quad [\text{FFA: } r_7, 22]\]
\[(24) \quad F e \quad [\text{FFA: } r_7, 23]\]
\[(25) \quad T\{c\} \quad [\text{FTB: } r_5, 22]\]
\[(26) \quad T\{\neg g\} \quad [\text{Cut}]\]
\[(27) \quad F g \quad [\text{BTB: } 26]\]
\[(28) \quad F\{g\} \quad [\text{FFB: } r_4, r_6, 27]\]
\[(29) \quad F c \quad [\text{WFN: } 28]\]
\[(30) \quad F\{\neg g\} \quad [\text{Cut}]\]
\[(31) \quad T g \quad [\text{BBF: } 30]\]
\[(32) \quad T\{g\} \quad [\text{FTB: } r_4, r_6, 31]\]
\[(33) \quad F f \quad [\text{FFA: } r_8, 30]\]
\[(34) \quad T\{\neg a, \neg f\} \quad [\text{FTB: } r_9, 17, 33]\]
$\mathcal{T}_{nomore^{++}}$: Example tableau

\[
\begin{align*}
(r_1) & \quad a \leftarrow \neg b \\
(r_4) & \quad c \leftarrow g \\
(r_7) & \quad e \leftarrow f, \neg c \\
(r_2) & \quad b \leftarrow d, \neg a \\
(r_5) & \quad d \leftarrow c \\
(r_8) & \quad f \leftarrow \neg g \\
(r_3) & \quad c \leftarrow b, d \\
(r_6) & \quad d \leftarrow g \\
(r_9) & \quad g \leftarrow \neg a, \neg f
\end{align*}
\]
Conflict-driven ASP Solving: Overview

42 Motivation
43 Boolean constraints
44 Nogoods from logic programs
45 Conflict-driven nogood learning
Motivation

Outline

42 Motivation
43 Boolean constraints
44 Nogoods from logic programs
45 Conflict-driven nogood learning
Goal: Approach to computing stable models of logic programs, based on concepts from
- Constraint Processing (CP) and
- Satisfiability Testing (SAT)

Idea: View inferences in ASP as unit propagation on nogoods

Benefits:
- A uniform constraint-based framework for different kinds of inferences in ASP
- Advanced techniques from the areas of CP and SAT
- Highly competitive implementation
42 Motivation

43 Boolean constraints

44 Nogoods from logic programs

45 Conflict-driven nogood learning
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence $(\sigma_1, \ldots, \sigma_n)$ of signed literals $\sigma_i$ of form $T v$ or $F v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$.
- $T v$ expresses that $v$ is true and $F v$ that it is false.
- The complement, $\overline{\sigma}$, of a literal $\sigma$ is defined as $T v = F \overline{\sigma}$ and $F v = T \overline{\sigma}$.
- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals.
- Given this, we access true and false propositions in $A$ via

$$A^T = \{ v \in \text{dom}(A) \mid T v \in A \} \text{ and } A^F = \{ v \in \text{dom}(A) \mid F v \in A \}$$
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence
  
  \[(\sigma_1, \ldots, \sigma_n)\]
  
  of signed literals $\sigma_i$ of form $T_v$ or $F_v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$

- $T_v$ expresses that $v$ is true and $F_v$ that it is false

- The complement, $\overline{\sigma}$, of a literal $\sigma$ is defined as $\overline{T_v} = F_v$ and $\overline{F_v} = T_v$

- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$

- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$

- We sometimes identify an assignment with the set of its literals

- Given this, we access true and false propositions in $A$ via

  \[A^T = \{ v \in \text{dom}(A) \mid T_v \in A \} \text{ and } A^F = \{ v \in \text{dom}(A) \mid F_v \in A \}\]
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence
  $$(\sigma_1, \ldots, \sigma_n)$$
  of signed literals $\sigma_i$ of form $T_v$ or $F_v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$
- $T_v$ expresses that $v$ is true and $F_v$ that it is false
- The complement, $\overline{\sigma}$, of a literal $\sigma$ is defined as $\overline{T_v} = F_v$ and $\overline{F_v} = T_v$
- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$
- We sometimes identify an assignment with the set of its literals
- Given this, we access true and false propositions in $A$ via
  $$A^T = \{ v \in \text{dom}(A) \mid T_v \in A \} \text{ and } A^F = \{ v \in \text{dom}(A) \mid F_v \in A \}$$
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence $(\sigma_1, \ldots, \sigma_n)$ of signed literals $\sigma_i$ of form $T v$ or $F v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$.
- $T v$ expresses that $v$ is true and $F v$ that it is false.
- The complement, $\overline{\sigma}$, of a literal $\sigma$ is defined as $\overline{T v} = F v$ and $\overline{F v} = T v$.
- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals.
- Given this, we access true and false propositions in $A$ via $A^T = \{ v \in \text{dom}(A) \mid T v \in A \}$ and $A^F = \{ v \in \text{dom}(A) \mid F v \in A \}$.
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence $(\sigma_1, \ldots, \sigma_n)$ of signed literals $\sigma_i$ of form $T_v$ or $F_v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$.

- $T_v$ expresses that $v$ is true and $F_v$ that it is false.

- The complement, $\overline{\sigma}$, of a literal $\sigma$ is defined as $\overline{T_v} = F_v$ and $\overline{F_v} = T_v$.

- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$.

- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.

- We sometimes identify an assignment with the set of its literals.

- Given this, we access true and false propositions in $A$ via

$$A^T = \{v \in \text{dom}(A) \mid T_v \in A\} \text{ and } A^F = \{v \in \text{dom}(A) \mid F_v \in A\}$$
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence $(\sigma_1, \ldots, \sigma_n)$

  of signed literals $\sigma_i$ of form $T_v$ or $F_v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$

- $T_v$ expresses that $v$ is true and $F_v$ that it is false

- The complement, $\bar{\sigma}$, of a literal $\sigma$ is defined as $\bar{T_v} = F_v$ and $\bar{F_v} = T_v$

- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$

- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$

- We sometimes identify an assignment with the set of its literals

- Given this, we access true and false propositions in $A$ via

$$A^T = \{v \in \text{dom}(A) \mid T_v \in A\} \text{ and } A^F = \{v \in \text{dom}(A) \mid F_v \in A\}$$
Assignments

- An assignment $A$ over $\text{dom}(A) = A(P) \cup B(P)$ is a sequence $(\sigma_1, \ldots, \sigma_n)$ of signed literals $\sigma_i$ of form $T_v$ or $F_v$ for $v \in \text{dom}(A)$ and $1 \leq i \leq n$.
- $T_v$ expresses that $v$ is true and $F_v$ that it is false.
- The complement, $\overline{\sigma}$, of a literal $\sigma$ is defined as $\overline{T_v} = F_v$ and $\overline{F_v} = T_v$.
- $A \circ \sigma$ stands for the result of appending $\sigma$ to $A$.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals.
- Given this, we access true and false propositions in $A$ via

$$A^T = \{ v \in \text{dom}(A) \mid T_v \in A \} \quad \text{and} \quad A^F = \{ v \in \text{dom}(A) \mid F_v \in A \}$$
Nogoods, solutions, and unit propagation

- A nogood is a set \(\{\sigma_1, \ldots, \sigma_n\}\) of signed literals, expressing a constraint violated by any assignment containing \(\sigma_1, \ldots, \sigma_n\).

- An assignment \(A\) such that \(A^T \cup A^F = \text{dom}(A)\) and \(A^T \cap A^F = \emptyset\) is a solution for a set \(\Delta\) of nogoods, if \(\delta \not\subseteq A\) for all \(\delta \in \Delta\).

- For a nogood \(\delta\), a literal \(\sigma \in \delta\), and an assignment \(A\), we say that \(\overline{\sigma}\) is unit-resulting for \(\delta\) wrt \(A\), if
  1. \(\delta \setminus A = \{\sigma\}\) and
  2. \(\overline{\sigma} \not\in A\)

- For a set \(\Delta\) of nogoods and an assignment \(A\), unit propagation is the iterated process of extending \(A\) with unit-resulting literals until no further literal is unit-resulting for any nogood in \(\Delta\).
Nogoods, solutions, and unit propagation

- A nogood is a set $\{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_n$.

- An assignment $A$ such that $A^T \cup A^F = \text{dom}(A)$ and $A^T \cap A^F = \emptyset$ is a solution for a set $\Delta$ of nogoods, if $\delta \nsubseteq A$ for all $\delta \in \Delta$.

- For a nogood $\delta$, a literal $\sigma \in \delta$, and an assignment $A$, we say that $\sigma$ is unit-resulting for $\delta$ wrt $A$, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$.

- For a set $\Delta$ of nogoods and an assignment $A$, unit propagation is the iterated process of extending $A$ with unit-resulting literals until no further literal is unit-resulting for any nogood in $\Delta$. 
A nogood is a set \( \{ \sigma_1, \ldots, \sigma_n \} \) of signed literals, expressing a constraint violated by any assignment containing \( \sigma_1, \ldots, \sigma_n \).

An assignment \( A \) such that \( A^T \cup A^F = dom(A) \) and \( A^T \cap A^F = \emptyset \) is a solution for a set \( \Delta \) of nogoods, if \( \delta \not\subseteq A \) for all \( \delta \in \Delta \).

For a nogood \( \delta \), a literal \( \sigma \in \delta \), and an assignment \( A \), we say that \( \overline{\sigma} \) is unit-resulting for \( \delta \) wrt \( A \), if

1. \( \delta \setminus A = \{ \sigma \} \) and
2. \( \overline{\sigma} \not\in A \)

For a set \( \Delta \) of nogoods and an assignment \( A \), unit propagation is the iterated process of extending \( A \) with unit-resulting literals until no further literal is unit-resulting for any nogood in \( \Delta \).
Nogoods, solutions, and unit propagation

- A nogood is a set \( \{\sigma_1, \ldots, \sigma_n\} \) of signed literals, expressing a constraint violated by any assignment containing \( \sigma_1, \ldots, \sigma_n \).

- An assignment \( A \) such that \( A^T \cup A^F = \text{dom}(A) \) and \( A^T \cap A^F = \emptyset \) is a solution for a set \( \Delta \) of nogoods, if \( \delta \not\subseteq A \) for all \( \delta \in \Delta \).

- For a nogood \( \delta \), a literal \( \sigma \in \delta \), and an assignment \( A \), we say that \( \overline{\sigma} \) is unit-resulting for \( \delta \) wrt \( A \), if
  1. \( \delta \setminus A = \{\sigma\} \) and
  2. \( \overline{\sigma} \not\in A \)

- For a set \( \Delta \) of nogoods and an assignment \( A \), unit propagation is the iterated process of extending \( A \) with unit-resulting literals until no further literal is unit-resulting for any nogood in \( \Delta \).
Outline

42 Motivation
43 Boolean constraints
44 Nogoods from logic programs
45 Conflict-driven nogood learning
Outline

42 Motivation

43 Boolean constraints

44 Nogoods from logic programs
   ▪ Nogoods from program completion
   ▪ Nogoods from loop formulas

45 Conflict-driven nogood learning
   ▪ CDNL-ASP Algorithm
   ▪ Nogood Propagation
   ▪ Conflict Analysis
The completion of a logic program $P$ can be defined as follows:

$$\{ v_B \leftrightarrow a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \mid B \in B(P) \text{ and } B = \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}\}$$

$$\cup \{ a \leftrightarrow v_{B_1} \lor \cdots \lor v_{B_k} \mid a \in A(P) \text{ and } B_P(a) = \{B_1, \ldots, B_k\}\} ,$$

where $B_P(a) = \{B(r) \mid r \in P \text{ and } h(r) = a\}$.
The (body-oriented) equivalence

\[ \nu_B \leftrightarrow a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \]

can be decomposed into two implications:
The (body-oriented) equivalence

\[ \nu_B \iff a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \]

can be decomposed into two implications:

1. \[ \nu_B \implies a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \]

is equivalent to the conjunction of

\[ \neg \nu_B \lor a_1, \ldots, \neg \nu_B \lor a_m, \neg \nu_B \lor \neg a_{m+1}, \ldots, \neg \nu_B \lor \neg a_n \]

and induces the set of nogoods

\[ \Delta(B) = \{ \{ T_B, F_{a_1} \}, \ldots, \{ T_B, F_{a_m} \}, \{ T_B, T_{a_{m+1}} \}, \ldots, \{ T_B, T_{a_n} \} \} \]
The (body-oriented) equivalence

\[ \nu_B \leftrightarrow a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \]

can be decomposed into two implications:

\[ 2 \quad a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \rightarrow \nu_B \]

gives rise to the nogood

\[ \delta(B) = \{ F_B, T_a_1, \ldots, T_a_m, F_{a_{m+1}}, \ldots, F_{a_n} \} \]
Nogoods from logic programs
via program completion

- Analogously, the (atom-oriented) equivalence

\[ a \leftrightarrow v_{B_1} \lor \cdots \lor v_{B_k} \]

yields the nogoods

1. \( \Delta(a) = \{ \{ Fa, TB_1 \}, \ldots, \{ Fa, TB_k \} \} \) and

2. \( \delta(a) = \{ Ta, FB_1, \ldots, FB_k \} \)
For an atom $a$ where $B_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{T_a, FB_1, \ldots, FB_k\} \quad \text{and} \quad \{\{Fa, TB_1\}, \ldots, \{Fa, TB_k\}\}$$

**Example** Given Atom $x$ with $B(x) = \{\{y\}, \{\neg z\}\}$, we obtain

$$\begin{align*}
x & \leftarrow y \\
\text{ } & \text{ } \\
x & \leftarrow \neg z
\end{align*}$$

$$\begin{align*}
\{Tx, F\{y\}, F\{\neg z\}\} \\
\{ \{Fx, T\{y\}\}, \{Fx, T\{\neg z\}\}\}
\end{align*}$$

For nogood $\{Tx, F\{y\}, F\{\neg z\}\}$, the signed literal $Fx$ is unit-resulting wrt assignment $(F\{y\}, F\{\neg z\})$ and $T\{\neg z\}$ is unit-resulting wrt assignment $(Tx, F\{y\})$.
Nogoods from logic programs

atom-oriented nogoods

For an atom $a$ where $B_P(a) = \{B_1, \ldots, B_k\}$, we get

\[
\{T_a, F_{B_1}, \ldots, F_{B_k}\} \quad \text{and} \quad \{\{F_a, T_{B_1}\}, \ldots, \{F_a, T_{B_k}\}\}
\]

Example Given Atom $x$ with $B(x) = \{\{y\}, \{\sim z\}\}$, we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \sim z
\end{align*}
\]

\[
\{T_x, F\{y\}, F\{\sim z\}\}
\]

\[
\{\{F_x, T\{y\}\}, \{F_x, T\{\sim z\}\}\}
\]

For nogood $\{T_x, F\{y\}, F\{\sim z\}\}$, the signed literal $F_x$ is unit-resulting wrt assignment $(F\{y\}, F\{\sim z\})$ and $T\{\sim z\}$ is unit-resulting wrt assignment $(T_x, F\{y\})$.
Nogoods from logic programs
atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{B_1, \ldots, B_k\} \), we get

\[
\{T_a, FB_1, \ldots, FB_k\} \quad \text{and} \quad \{\{Fa, TB_1\}, \ldots, \{Fa, TB_k\}\}
\]

Example Given Atom \( x \) with \( B(x) = \{\{y\}, \{\neg z\}\} \), we obtain

\[
x \leftarrow y \quad \{T_x, F\{y\}, F\{\neg z\}\}
\]
\[
x \leftarrow \neg z \quad \{\{F_x, T\{y\}\}, \{F_x, T\{\neg z\}\}\}
\]

For nogood \( \{T_x, F\{y\}, F\{\neg z\}\} \), the signed literal

- \( Fx \) is unit-resulting wrt assignment \( (F\{y\}, F\{\neg z\}) \) and
- \( T\{\neg z\} \) is unit-resulting wrt assignment \( (T_x, F\{y\}) \)
Nogoods from logic programs

atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{ B_1, \ldots, B_k \} \), we get

\[
\{ T_a, F B_1, \ldots, F B_k \} \quad \text{and} \quad \{ \{ F a, T B_1 \}, \ldots, \{ F a, T B_k \} \}
\]

Example: Given Atom \( x \) with \( B(x) = \{ \{ y \}, \{ \neg z \} \} \), we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \neg z
\end{align*}
\]

\[
\{ T_x, F \{ y \}, F \{ \neg z \} \} \\
\{ \{ F x, T \{ y \} \}, \{ F x, T \{ \neg z \} \} \}
\]

For nogood \( \{ T_x, F \{ y \}, F \{ \neg z \} \} \), the signed literal

- \( F x \) is unit-resulting wrt assignment \( (F \{ y \}, F \{ \neg z \}) \) and
- \( T \{ \neg z \} \) is unit-resulting wrt assignment \( (T x, F \{ y \}) \)
Nogoods from logic programs
atom-oriented nogoods

For an atom $a$ where $BP(a) = \{B_1, \ldots, B_k\}$, we get

$$\{Ta, FB_1, \ldots, FB_k\}$$ and $$\{Fa, TB_1, \ldots, Fa, TB_k\}$$

Example Given Atom $x$ with $B(x) = \{\{y\}, \{\neg z\}\}$, we obtain

$$x \leftarrow y \quad \{Tx, F\{y\}, F\{\neg z\}\}$$
$$x \leftarrow \neg z \quad \{\{Fx, T\{y\}\}, \{Fx, T\{\neg z\}\}\}$$

For nogood $\{Tx, F\{y\}, F\{\neg z\}\}$, the signed literal

- $Fx$ is unit-resulting wrt assignment $(F\{y\}, F\{\neg z\})$ and
- $T\{\neg z\}$ is unit-resulting wrt assignment $(Tx, F\{y\})$
Nogoods from logic programs
atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{ B_1, \ldots, B_k \} \), we get

\[
\{ T a, F B_1, \ldots, F B_k \} \quad \text{and} \quad \{ \{ F a, T B_1 \}, \ldots, \{ F a, T B_k \} \}
\]

Example: Given Atom \( x \) with \( B(x) = \{ \{ y \}, \{ \sim z \} \} \), we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \sim z
\end{align*}
\]

\[
\{ T x, F \{ y \}, F \{ \sim z \} \} \quad \{ \{ F x, T \{ y \} \}, \{ F x, T \{ \sim z \} \} \}
\]

For nogood \( \{ T x, F \{ y \}, F \{ \sim z \} \} \), the signed literal

\[ F x \]

is unit-resulting wrt assignment \( (F \{ y \}, F \{ \sim z \}) \) and

\[ T \{ \sim z \} \]

is unit-resulting wrt assignment \( (T x, F \{ y \}) \)
Nogoods from logic programs
atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{B_1, \ldots, B_k\} \), we get

\[
\{T a, F B_1, \ldots, F B_k\} \quad \text{and} \quad \{\{F a, T B_1\}, \ldots, \{F a, T B_k\}\}
\]

Example Given Atom \( x \) with \( B(x) = \{\{y\}, \{\sim z\}\} \), we obtain

\[
x \leftarrow y \quad \{T x, F\{y\}, F\{\sim z\}\}
\]

\[
x \leftarrow \sim z \quad \{\{F x, T\{y\}\}, \{F x, T\{\sim z\}\}\}
\]

For nogood \( \{T x, F\{y\}, F\{\sim z\}\} \), the signed literal

- \( F x \) is unit-resulting wrt assignment \( (F\{y\}, F\{\sim z\}) \) and
- \( T\{\sim z\} \) is unit-resulting wrt assignment \( (T x, F\{y\}) \)
Nogoods from logic programs
atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{ B_1, \ldots, B_k \} \), we get

\[
\{ T_a, F B_1, \ldots, F B_k \} \quad \text{and} \quad \{ \{ F a, T B_1 \}, \ldots, \{ F a, T B_k \} \}
\]

Example Given Atom \( x \) with \( B(x) = \{ \{ y \}, \{ \neg z \} \} \), we obtain

\[
\begin{align*}
&x \leftarrow y \\
&x \leftarrow \neg z \\
&\{ T x, F \{ y \}, F \{ \neg z \} \} \\
&\{ \{ F x, T \{ y \} \}, \{ F x, T \{ \neg z \} \} \}
\end{align*}
\]

For nogood \( \{ T x, F \{ y \}, F \{ \neg z \} \} \), the signed literal

\[
\begin{align*}
&\text{\( F x \) is unit-resulting wrt assignment \( (F \{ y \}, F \{ \neg z \}) \) and} \\
&\text{\( T \{ \neg z \} \) is unit-resulting wrt assignment \( (T x, F \{ y \}) \)}
\end{align*}
\]
Nogoods from logic programs
atom-oriented nogoods

- For an atom $a$ where $B_P(a) = \{B_1, \ldots, B_k\}$, we get

\[
\{T a, FB_1, \ldots, FB_k\} \quad \text{and} \quad \{\{Fa, TB_1\}, \ldots, \{Fa, TB_k\}\}
\]

- Example Given Atom $x$ with $B(x) = \{\{y\}, \{\sim z\}\}$, we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \sim z
\end{align*}
\]

\[
\{T x, F\{y\}, F\{\sim z\}\} \\
\{\{Fx, T\{y\}\}, \{Fx, T\{\sim z\}\}\}
\]

For nogood $\{T x, F\{y\}, F\{\sim z\}\}$, the signed literal

- $Fx$ is unit-resulting wrt assignment $(F\{y\}, F\{\sim z\})$ and
- $T\{\sim z\}$ is unit-resulting wrt assignment $(T x, F\{y\})$
Nogoods from logic programs
atom-oriented nogoods

For an atom \(a\) where \(BP(a) = \{B_1, \ldots, B_k\}\), we get

\[
\{Ta, FB_1, \ldots, FB_k\} \quad \text{and} \quad \{\{Fa, TB_1\}, \ldots, \{Fa, TB_k\}\}
\]

Example Given Atom \(x\) with \(B(x) = \{\{y\}, \{\neg z\}\}\), we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \neg z
\end{align*}
\]

\[
\{Tx, F\{y\}, F\{\neg z\}\} \quad \{\{Fx, T\{y\}\}, \{Fx, T\{\neg z\}\}\}
\]

For nogood \(\{Tx, F\{y\}, F\{\neg z\}\}\), the signed literal

\[
\begin{align*}
F_x & \text{ is unit-resulting wrt assignment } (F\{y\}, F\{\neg z\}) \text{ and} \\
T\{\neg z\} & \text{ is unit-resulting wrt assignment } (Tx, F\{y\})
\end{align*}
\]
Nogoods from logic programs

atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{ B_1, \ldots, B_k \} \), we get

\[
\{ T_a, F B_1, \ldots, F B_k \} \quad \text{and} \quad \{ \{ F_a, T B_1 \}, \ldots, \{ F_a, T B_k \} \}
\]

Example

Given Atom \( x \) with \( B(x) = \{ \{ y \}, \{ \lnot z \} \} \), we obtain

\[
\begin{align*}
& x \leftarrow y \\
& x \leftarrow \lnot z \\
& \{ T_x, F\{ y \}, F\{ \lnot z \} \} \\
& \{ \{ F_x, T\{ y \} \}, \{ F_x, T\{ \lnot z \} \} \}
\end{align*}
\]

For nogood \( \{ T_x, F\{ y \}, F\{ \lnot z \} \} \), the signed literal

- \( F_x \) is unit-resulting wrt assignment \((F\{ y \}, F\{ \lnot z \})\) and
- \( T\{ \lnot z \} \) is unit-resulting wrt assignment \((T_x, F\{ y \})\)
Nogoods from logic programs
atom-oriented nogoods

For an atom $a$ where $B_P(a) = \{B_1, \ldots, B_k\}$, we get

$$\{T_a, FB_1, \ldots, FB_k\} \text{ and } \{\{F_a, TB_1\}, \ldots, \{F_a, TB_k\}\}$$

Example Given Atom $x$ with $B(x) = \{\{y\}, \{\sim z\}\}$, we obtain

$$x \leftarrow y$$

$$x \leftarrow \sim z$$

$$\{T_x, F\{y\}, F\{\sim z\}\}$$

$$\{\{F_x, T\{y\}\}, \{F_x, T\{\sim z\}\}\}$$

For nogood $\{T_x, F\{y\}, F\{\sim z\}\}$, the signed literal

- $F_x$ is unit-resulting wrt assignment $(F\{y\}, F\{\sim z\})$ and
- $T\{\sim z\}$ is unit-resulting wrt assignment $(T_x, F\{y\})$
For an atom \( a \) where \( B_P(a) = \{B_1, \ldots, B_k\} \), we get

\[
\{T a, F B_1, \ldots, F B_k\} \quad \text{and} \quad \{\{F a, T B_1\}, \ldots, \{F a, T B_k\}\}
\]

Example: Given Atom \( x \) with \( B(x) = \{\{y\}, \{\sim z\}\} \), we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \sim z
\end{align*}
\]

\[
\{T x, F\{y\}, F\{\sim z\}\} \quad \{\{F x, T\{y\}\}, \{F x, T\{\sim z\}\}\}
\]

For nogood \( \{T x, F\{y\}, F\{\sim z\}\} \), the signed literal

- \( F x \) is unit-resulting wrt assignment \((F\{y\}, F\{\sim z\})\) and
- \( T\{\sim z\} \) is unit-resulting wrt assignment \((T x, F\{y\})\)
Nogoods from logic programs
atom-oriented nogoods

For an atom \( a \) where \( B_P(a) = \{ B_1, \ldots, B_k \} \), we get

\[
\{ T_a, \overline{F} B_1, \ldots, \overline{F} B_k \} \quad \text{and} \quad \{ \{ \overline{F} a, T B_1 \}, \ldots, \{ \overline{F} a, T B_k \} \}
\]

**Example** Given Atom \( x \) with \( B(x) = \{ \{ y \}, \{ \overline{z} \} \} \), we obtain

\[
\begin{align*}
x & \leftarrow y \\
x & \leftarrow \overline{z}
\end{align*}
\]

\[
\{ T x, \overline{F} \{ y \}, \overline{F} \{ \overline{z} \} \}
\]

\[
\{ \{ \overline{F} x, T \{ y \} \}, \{ \overline{F} x, T \{ \overline{z} \} \}\}
\]

For nogood \( \{ T x, \overline{F} \{ y \}, \overline{F} \{ \overline{z} \} \} \), the signed literal

- \( \overline{F} x \) is unit-resulting wrt assignment \( (\overline{F} \{ y \}, \overline{F} \{ \overline{z} \}) \) and

- \( T \{ \overline{z} \} \) is unit-resulting wrt assignment \( (T x, \overline{F} \{ y \}) \)
For a body $B = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\}$, we get
\[
\{FB, Ta_1, \ldots, Ta_m, Fa_{m+1}, \ldots, Fa_n\}\\
\{\{TB, Fa_1\}, \ldots, \{TB, Fa_m\}, \{TB, Ta_{m+1}\}, \ldots, \{TB, Ta_n\}\}
\]

Example Given Body $\{x, \neg y\}$, we obtain

\[
\begin{align*}
\ldots & \leftarrow x, \neg y \\
\vdots & \\
\ldots & \leftarrow x, \neg y
\end{align*}
\]
\[
\{F\{x, \neg y\}, Tx, Fy\}\\
\{\{T\{x, \neg y\}, Fx\}, \{T\{x, \neg y\}, Ty\}\}
\]

For nogood $\delta(\{x, \neg y\}) = \{F\{x, \neg y\}, Tx, Fy\}$, the signed literal
- $T\{x, \neg y\}$ is unit-resulting wrt assignment $(Tx, Fy)$ and
- $Ty$ is unit-resulting wrt assignment $(F\{x, \neg y\}, Tx)$
Nogoods from logic programs

body-oriented nogoods

For a body $B = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\}$, we get

$$\{FB, Ta_1, \ldots, Ta_m, Fa_{m+1}, \ldots, Fa_n\}$$

$$\{\{TB, Fa_1\}, \ldots, \{TB, Fa_m\}, \{TB, Ta_{m+1}\}, \ldots, \{TB, Ta_n\}\}$$

Example Given Body $\{x, \neg y\}$, we obtain

$$\ldots \leftarrow x, \neg y$$

$$\ldots \leftarrow x, \neg y$$

$$\{F\{x, \neg y\}, Tx, Fy\}$$

$$\{\{T\{x, \neg y\}, Fx\}, \{T\{x, \neg y\}, Ty\}\}$$

For nogood $\delta(\{x, \neg y\}) = \{F\{x, \neg y\}, Tx, Fy\}$, the signed literal

- $Tx$ is unit-resulting wrt assignment $(Tx, Fy)$ and
- $Ty$ is unit-resulting wrt assignment $(F\{x, \neg y\}, Tx)$
For a body $B = \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\}$, we get

$$\{FB, Ta_1, \ldots, Ta_m, Fa_{m+1}, \ldots, Fa_n\}$$

$$\{\{TB, Fa_1\}, \ldots, \{TB, Fa_m\}, \{TB, Ta_{m+1}\}, \ldots, \{TB, Ta_n\}\}$$

Example Given Body $\{x, \neg y\}$, we obtain

$$\ldots \leftarrow x, \neg y$$

$$\ldots \leftarrow x, \neg y$$

$$\{F\{x, \neg y\}, Tx, Fy\}$$

$$\{\{T\{x, \neg y\}, Fx\}, \{T\{x, \neg y\}, Ty\}\}$$

For nogood $\delta(\{x, \neg y\}) = \{F\{x, \neg y\}, Tx, Fy\}$, the signed literal

- $T\{x, \neg y\}$ is unit-resulting wrt assignment $(Tx, Fy)$ and
- $Ty$ is unit-resulting wrt assignment $(F\{x, \neg y\}, Tx)$
Characterization of stable models 
for tight logic programs

Let $P$ be a logic program and

$$\Delta_P = \{\delta(a) \mid a \in A(P)\} \cup \{\delta \in \Delta(a) \mid a \in A(P)\}$$

$$\cup \{\delta(B) \mid B \in B(P)\} \cup \{\delta \in \Delta(B) \mid B \in B(P)\}$$

**Theorem**

Let $P$ be a tight logic program. Then,

$X \subseteq A(P)$ is a stable model of $P$ iff

$X = A^T \cap A(P)$ for a (unique) solution $A$ for $\Delta_P$
Let $P$ be a logic program and

$$\Delta_P = \{\delta(a) \mid a \in A(P)\} \cup \{\delta \in \Delta(a) \mid a \in A(P)\}$$
$$\cup \{\delta(B) \mid B \in B(P)\} \cup \{\delta \in \Delta(B) \mid B \in B(P)\}$$

Theorem

Let $P$ be a tight logic program. Then, $X \subseteq A(P)$ is a stable model of $P$ iff $X = A^T \cap A(P)$ for a (unique) solution $A$ for $\Delta_P$.
Characterization of stable models

for tight logic programs, i.e. free of positive recursion

Let $P$ be a logic program and

$$
\Delta_P = \{\delta(a) \mid a \in A(P)\} \cup \{\delta \in \Delta(a) \mid a \in A(P)\} \\
\cup \{\delta(B) \mid B \in B(P)\} \cup \{\delta \in \Delta(B) \mid B \in B(P)\}
$$

**Theorem**

Let $P$ be a tight logic program. Then,

$X \subseteq A(P)$ is a stable model of $P$ iff

$X = A^T \cap A(P)$ for a (unique) solution $A$ for $\Delta_P$
Outline

42 Motivation

43 Boolean constraints

44 Nogoods from logic programs
   - Nogoods from program completion
   - Nogoods from loop formulas

45 Conflict-driven nogood learning
   - CDNL-ASP Algorithm
   - Nogood Propagation
   - Conflict Analysis
Let $P$ be a normal logic program and recall that:

- For $L \subseteq A(P)$, the external supports of $L$ for $P$ are
  \[ ES_P(L) = \{ r \in P \mid h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \} \]

- The (disjunctive) loop formula of $L$ for $P$ is
  \[ LF_P(L) = (\bigvee_{A \in L} A) \rightarrow (\bigvee_{r \in ES_P(L)} B(r)) \]
  \[ \leftrightarrow (\bigwedge_{r \in ES_P(L)} \neg B(r)) \rightarrow (\bigwedge_{A \in L} \neg A) \]

  Note: The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported.

- The external bodies of $L$ for $P$ are
  \[ EB_P(L) = \{ B(r) \mid r \in ES_P(L) \} \]
Let $P$ be a normal logic program and recall that:

- For $L \subseteq A(P)$, the external supports of $L$ for $P$ are
  \[ ES_P(L) = \{ r \in P \mid h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \} \]

- The (disjunctive) loop formula of $L$ for $P$ is
  \[ LF_P(L) = (\bigvee_{A \in L} A) \rightarrow (\bigvee_{r \in ES_P(L)} B(r)) \]
  \[ \iff (\bigwedge_{r \in ES_P(L)} \neg B(r)) \rightarrow (\bigwedge_{A \in L} \neg A) \]

  Note: The loop formula of $L$ enforces all atoms in $L$ to be \textit{false} whenever $L$ is not externally supported

- The external bodies of $L$ for $P$ are
  \[ EB_P(L) = \{ B(r) \mid r \in ES_P(L) \} \]
Let $P$ be a normal logic program and recall that:

- For $L \subseteq A(P)$, the external supports of $L$ for $P$ are 
  $$ES_P(L) = \{ r \in P \mid h(r) \in L \text{ and } B(r)^+ \cap L = \emptyset \}$$

- The (disjunctive) loop formula of $L$ for $P$ is
  $$LF_P(L) = (\bigvee_{A \in L} A) \rightarrow (\bigvee_{r \in ES_P(L)} B(r))$$
  $$\iff (\bigwedge_{r \in ES_P(L)} \neg B(r)) \rightarrow (\bigwedge_{A \in L} \neg A)$$

- **Note** The loop formula of $L$ enforces all atoms in $L$ to be *false* whenever $L$ is not externally supported

- The external bodies of $L$ for $P$ are 
  $$EB_P(L) = \{ B(r) \mid r \in ES_P(L) \}$$
For a logic program \( P \) and some \( \emptyset \subset U \subseteq A(P) \), define the loop nogood of an atom \( a \in U \) as
\[
\lambda(a, U) = \{ T_a, F B_1, \ldots, F B_k \}
\]
where \( E B_P(U) = \{ B_1, \ldots, B_k \} \).

We get the following set of loop nogoods for \( P \):
\[
\Lambda_P = \bigcup_{\emptyset \subset U \subseteq A(P)} \{ \lambda(a, U) \mid a \in U \}
\]

The set \( \Lambda_P \) of loop nogoods denies cyclic support among *true* atoms.
For a logic program $P$ and some $\emptyset \subset U \subseteq A(P)$, define the loop nogood of an atom $a \in U$ as

$$\lambda(a, U) = \{Ta, FB_1, \ldots, FB_k\}$$

where $EB_P(U) = \{B_1, \ldots, B_k\}$.

We get the following set of loop nogoods for $P$:

$$\Lambda_P = \bigcup_{\emptyset \subset U \subseteq A(P)} \{\lambda(a, U) \mid a \in U\}$$

The set $\Lambda_P$ of loop nogoods denies cyclic support among true atoms.
Nogoods from logic programs

For a logic program $P$ and some $\emptyset \subset U \subseteq A(P)$, define the loop nogood of an atom $a \in U$ as

$$\lambda(a, U) = \{Ta, FB_1, \ldots, FB_k\}$$

where $EB_P(U) = \{B_1, \ldots, B_k\}$

We get the following set of loop nogoods for $P$:

$$\Lambda_P = \bigcup_{\emptyset \subset U \subseteq A(P)} \{\lambda(a, U) \mid a \in U\}$$

The set $\Lambda_P$ of loop nogoods denies cyclic support among true atoms.
Example

Consider the program

\[
\begin{align*}
    x & \leftarrow \sim y \\
    y & \leftarrow \sim x \\
    u & \leftarrow x \\
    u & \leftarrow v \\
    v & \leftarrow u, y
\end{align*}
\]

For \( u \) in the set \( \{ u, v \} \), we obtain the loop nogood:

\[
\lambda(u, \{ u, v \}) = \{ T_u, F\{x\} \}
\]

Similarly for \( v \) in \( \{ u, v \} \), we get:

\[
\lambda(v, \{ u, v \}) = \{ T_v, F\{x\} \}
\]
Example

- Consider the program

\[
\begin{align*}
&x \leftarrow \neg y \\
&u \leftarrow x \\
&y \leftarrow \neg x \\
&u \leftarrow v \\
&v \leftarrow u, y
\end{align*}
\]

- For \( u \) in the set \( \{u, v\} \), we obtain the loop nogood:

\[
\lambda(u, \{u, v\}) = \{T u, F \{x\}\}
\]

Similarly for \( v \) in \( \{u, v\} \), we get:

\[
\lambda(v, \{u, v\}) = \{T v, F \{x\}\}
\]
Example

Consider the program

\[
\begin{align*}
  x & \leftarrow \neg y \\
  y & \leftarrow \neg x \\
  u & \leftarrow x \\
  u & \leftarrow v \\
  v & \leftarrow u, y
\end{align*}
\]

For \( u \) in the set \( \{u, v\} \), we obtain the loop nogood:

\[
\lambda(u, \{u, v\}) = \{\text{T} u, \text{F}\{x\}\}
\]

Similarly for \( v \) in \( \{u, v\} \), we get:

\[
\lambda(v, \{u, v\}) = \{\text{T} v, \text{F}\{x\}\}
\]
Characterization of stable models

**Theorem**

*Let $P$ be a logic program. Then,*

\[ X \subseteq A(P) \text{ is a stable model of } P \iff X = A^T \cap A(P) \text{ for a (unique) solution } A \text{ for } \Delta_P \cup \Lambda_P \]

**Some remarks**

- Nogoods in $\Lambda_P$ augment $\Delta_P$ with conditions checking for unfounded sets, in particular, those being loops.
- While $|\Delta_P|$ is linear in the size of $P$, $\Lambda_P$ may contain exponentially many (non-redundant) loop nogoods.
Characterization of stable models

Theorem

Let \( P \) be a logic program. Then,

\[ X \subseteq A(P) \text{ is a stable model of } P \text{ iff } X = A^T \cap A(P) \text{ for a (unique) solution } A \text{ for } \Delta_P \cup \Lambda_P \]

Some remarks

- Nogoods in \( \Lambda_P \) augment \( \Delta_P \) with conditions checking for unfounded sets, in particular, those being loops.
- While \(|\Delta_P|\) is linear in the size of \( P \), \( \Lambda_P \) may contain exponentially many (non-redundant) loop nogoods.
Conflict-driven nogood learning

Outline

42 Motivation
43 Boolean constraints
44 Nogoods from logic programs
45 Conflict-driven nogood learning
Towards conflict-driven search

Boolean constraint solving algorithms pioneered for SAT led to:

- **Traditional DPLL-style approach**  
  (DPLL stands for ‘Davis-Putnam-Logemann-Loveland’)
  - (Unit) propagation
  - (Chronological) backtracking

- **Modern CDCL-style approach**  
  (CDCL stands for ‘Conflict-Driven Constraint Learning’)
  - (Unit) propagation
  - Conflict analysis (via resolution)
  - Learning + Backjumping + Assertion

in ASP, eg *smodels*  
in ASP, eg *clasp*
Conflict-driven nogood learning

DPLL-style solving

loop

propagate  // deterministically assign literals

if no conflict then

if all variables assigned then return solution
else decide  // non-deterministically assign some literal

else

if top-level conflict then return unsatisfiable
else

backtrack  // unassign literals propagated after last decision
flip      // assign complement of last decision literal
Conflict-driven nogood learning

CDCL-style solving

**loop**

**propagate**  // deterministically assign literals

if no conflict then

if all variables assigned then return solution
else decide  // non-deterministically assign some literal

else

if top-level conflict then return unsatisfiable
else

analyze  // analyze conflict and add conflict constraint
backjump  // unassign literals until conflict constraint is unit

Torsten Schaub (KRR@UP)  Answer Set Solving in Practice  February 18, 2019 350 / 653
Outline

42 Motivation

43 Boolean constraints

44 Nogoods from logic programs
   - Nogoods from program completion
   - Nogoods from loop formulas

45 Conflict-driven nogood learning
   - CDNL-ASP Algorithm
   - Nogood Propagation
   - Conflict Analysis
Outline of CDNL-ASP algorithm

- Keep track of deterministic consequences by unit propagation on:
  - Program completion $[\Delta_P]$
  - Loop nogoods, determined and recorded on demand $[\Lambda_P]$
  - Dynamic nogoods, derived from conflicts and unfounded sets $[\nabla]$

- When a nogood in $\Delta_P \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution (until reaching a Unique Implication Point, short: UIP)
  - Learn the derived conflict nogood $\delta$
  - Backjump to the earliest (heuristic) choice such that the complement of the UIP is unit-resulting for $\delta$
  - Assert the complement of the UIP and proceed (by unit propagation)

- Terminate when either:
  - Finding a stable model (a solution for $\Delta_P \cup \Lambda_P$)
  - Deriving a conflict independently of (heuristic) choices
Outline of CDNL-ASP algorithm

- Keep track of deterministic consequences by unit propagation on:
  - Program completion $[\Delta_P]$
  - Loop nogoods, determined and recorded on demand $[\Lambda_P]$
  - Dynamic nogoods, derived from conflicts and unfounded sets $[\nabla]$

- When a nogood in $\Delta_P \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution (until reaching a Unique Implication Point, short: UIP)
  - Learn the derived conflict nogood $\delta$
  - Backjump to the earliest (heuristic) choice such that the complement of the UIP is unit-resulting for $\delta$
  - Assert the complement of the UIP and proceed (by unit propagation)

- Terminate when either:
  - Finding a stable model (a solution for $\Delta_P \cup \Lambda_P$)
  - Deriving a conflict independently of (heuristic) choices
Outline of CDNL-ASP algorithm

- Keep track of deterministic consequences by unit propagation on:
  - Program completion \([\Delta_P]\)
  - Loop nogoods, determined and recorded on demand \([\Lambda_P]\)
  - Dynamic nogoods, derived from conflicts and unfounded sets \([\nabla]\)

- When a nogood in \(\Delta_P \cup \nabla\) becomes violated:
  - Analyze the conflict by resolution (until reaching a Unique Implication Point, short: UIP)
  - Learn the derived conflict nogood \(\delta\)
  - Backjump to the earliest (heuristic) choice such that the complement of the UIP is unit-resulting for \(\delta\)
  - Assert the complement of the UIP and proceed (by unit propagation)

- Terminate when either:
  - Finding a stable model (a solution for \(\Delta_P \cup \Lambda_P\))
  - Deriving a conflict independently of (heuristic) choices
Algorithm 1: CDNL-ASP

Input : A normal program $P$
Output : A stable model of $P$ or “no stable model”

$A := \emptyset$  // assignment over $A(P) \cup B(P)$
$\n := \emptyset$  // set of recorded nogoods
$dl := 0$  // decision level

loop
  $(A, \n) := \text{NogoodPropagation}(P, \n, A)$
  
  if $\varepsilon \subseteq A$ for some $\varepsilon \in \Delta_P \cup \n$ then  // conflict
    if $\max(\{d\text{level}(\sigma) \mid \sigma \in \varepsilon\} \cup \{0\}) = 0$ then return no stable model
  
    $(\delta, dl) := \text{ConflictAnalysis}(\varepsilon, P, \n, A)$
    $\n := \n \cup \{\delta\}$  // (temporarily) record conflict nogood
    $A := A \setminus \{\sigma \in A \mid dl < d\text{level}(\sigma)\}$  // backjumping
  
  else if $A^T \cup A^F = A(P) \cup B(P)$ then  // stable model
    return $A^T \cap A(P)$
  
  else
    $\sigma_d := \text{Select}(P, \n, A)$  // decision
    $dl := dl + 1$
    $d\text{level}(\sigma_d) := dl$
    $A := A \circ \sigma_d$

// end of loop
Observations

- Decision level $dl$, initially set to 0, is used to count the number of heuristically chosen literals in assignment $A$
- For a heuristically chosen literal $\sigma_d = Ta$ or $\sigma_d = Fa$, respectively, we require $\sigma \in (A(P) \cup B(P)) \setminus (AT \cup AF)$
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of $\sigma$, viz. the value $dl$ had when $\sigma$ was assigned
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_P \cup \nabla$
- A conflict at decision level 0 (where $A$ contains no heuristically chosen literals) indicates non-existence of stable models
- A nogood $\delta$ derived by conflict analysis is asserting, that is, some literal is unit-resulting for $\delta$ at a decision level $k < dl$
  - After learning $\delta$ and backjumping to decision level $k$, at least one literal is newly derivable by unit propagation
  - No explicit flipping of heuristically chosen literals!
Observations

- Decision level $dl$, initially set to 0, is used to count the number of heuristically chosen literals in assignment $A$.
- For a heuristically chosen literal $\sigma_d = Ta$ or $\sigma_d = Fa$, respectively, we require $a \in (A(P) \cup B(P)) \setminus (AT \cup AF)$.
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of $\sigma$, viz. the value $dl$ had when $\sigma$ was assigned.
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_P \cup \nabla$.
- A conflict at decision level 0 (where $A$ contains no heuristically chosen literals) indicates non-existence of stable models.
- A nogood $\delta$ derived by conflict analysis is asserting, that is, some literal is unit-resulting for $\delta$ at a decision level $k < dl$.
  - After learning $\delta$ and backjumping to decision level $k$, at least one literal is newly derivable by unit propagation.
  - No explicit flipping of heuristically chosen literals!
Observations

- Decision level $dl$, initially set to 0, is used to count the number of heuristically chosen literals in assignment $A$.

- For a heuristically chosen literal $\sigma_d = Ta$ or $\sigma_d = Fa$, respectively, we require $a \in (A(P) \cup B(P)) \setminus (A^T \cup A^F)$.

- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of $\sigma$, viz. the value $dl$ had when $\sigma$ was assigned.

- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_P \cup \nabla$.

- A conflict at decision level 0 (where $A$ contains no heuristically chosen literals) indicates non-existence of stable models.

- A nogood $\delta$ derived by conflict analysis is asserting, that is, some literal is unit-resulting for $\delta$ at a decision level $k < dl$.
  - After learning $\delta$ and backjumping to decision level $k$, at least one literal is newly derivable by unit propagation.
  - No explicit flipping of heuristically chosen literals!
Consider

\[
P = \begin{cases}
    x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\
y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y
\end{cases}
\]

<table>
<thead>
<tr>
<th>(dl)</th>
<th>(\sigma_d)</th>
<th>(\overline{\sigma})</th>
<th>(\delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(T u)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(F{\sim x, \sim y})</td>
<td>(F_w)</td>
<td>({T_w, F{\sim x, \sim y}} = \delta(w))</td>
</tr>
<tr>
<td>3</td>
<td>(F{\sim y})</td>
<td>(F_x)</td>
<td>({T_x, F{\sim y}} = \delta(x))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F{x})</td>
<td>({T{x}, F_x} \in \Delta({x}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F{x, y})</td>
<td>({T{x, y}, F_x} \in \Delta({x, y}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>({T u, F{x}, F{x, y}} = \lambda(u, {u, v}))</td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[ P = \left\{ \begin{array}{l}
x \leftarrow \neg y \\
u \leftarrow x, y \\
y \leftarrow \neg x \\
u \leftarrow v \\
v \leftarrow u, y \\
\end{array} \right\} \]

<table>
<thead>
<tr>
<th>( dl )</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\neg x, \neg y} )</td>
<td>( F_w )</td>
<td>( {T_w, F{\neg x, \neg y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\neg y} )</td>
<td>( F_x )</td>
<td>( {T_x, F{\neg y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td>( F{x} )</td>
<td>( T{x}, F_x \in \Delta({x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( F{x, y} )</td>
<td>( T{x, y}, F_x \in \Delta({x, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( {Tu, F{x}, F{x, y}} = \lambda(u, {u, y}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[ P = \{ x \leftarrow \neg y \quad u \leftarrow x, y \quad v \leftarrow x \quad w \leftarrow \neg x, \neg y \quad y \leftarrow \neg x \quad u \leftarrow v \quad v \leftarrow u, y \} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( T_u )</td>
<td>( w )</td>
<td>( { T_w, F{ \neg x, \neg y } } = \delta(w) )</td>
</tr>
<tr>
<td>2</td>
<td>( F{ \neg x, \neg y } )</td>
<td>( F_w )</td>
<td>( { T_x, F{ \neg y } } = \delta(x) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{ \neg y } )</td>
<td>( F_x )</td>
<td>( { T{ x }, F_x } \in \Delta({ x }) )</td>
</tr>
<tr>
<td></td>
<td>( F{ x } )</td>
<td>( F{ x, y } )</td>
<td>( { T{ x, y }, F_x } \in \Delta({ x, y }) )</td>
</tr>
<tr>
<td></td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td></td>
<td>( { T_u, F{ x }, F{ x, y } } = \lambda(u, { u, y }) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[ P = \{ x \leftarrow \neg y, \quad u \leftarrow x, y, \quad v \leftarrow x, \quad w \leftarrow \neg x, \neg y, \quad y \leftarrow \neg x, \quad u \leftarrow v, \quad v \leftarrow u, y \} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\neg x, \neg y} )</td>
<td>( Fw )</td>
<td>( { Tw, F{\neg x, \neg y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\neg y} )</td>
<td>( Fx )</td>
<td>( { Tx, F{\neg y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( F{x} )</td>
<td>( { T{x}, Fx} \in \Delta({x}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( F{x, y} )</td>
<td>( { T{x, y}, Fx} \in \Delta({x, y}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td></td>
<td>( { Tu, F{x}, F{x, y}} = \lambda(u, {u, y}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[
P = \begin{cases} 
  x \leftarrow \lnot y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \lnot x, \lnot y \\
  y \leftarrow \lnot x & u \leftarrow v & v \leftarrow u, y 
\end{cases}
\]

<table>
<thead>
<tr>
<th>dl</th>
<th>(\sigma_d)</th>
<th>(\overline{\sigma})</th>
<th>(\delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(T_u)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(F{\lnot x, \lnot y})</td>
<td>(F_w)</td>
<td>({T_w, F{\lnot x, \lnot y}} = \delta(w))</td>
</tr>
<tr>
<td>3</td>
<td>(F{\lnot y})</td>
<td>(F_x)</td>
<td>({T_x, F{\lnot y}} = \delta(x))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F{x})</td>
<td>({T{x}, F_x} \in \Delta({x}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F{x, y})</td>
<td>({T{x, y}, F_x} \in \Delta({x, y}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\vdots)</td>
<td>(\vdots)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>({T_u, F{x}, F{x, y}} = \lambda(u, {u, y}))</td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[ P = \{ x \leftarrow \sim y, u \leftarrow x, y, v \leftarrow x, w \leftarrow \sim x, \sim y, y \leftarrow \sim x, u \leftarrow v, v \leftarrow u, y \} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( T u )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\sim x, \sim y} )</td>
<td>( Fw )</td>
<td>( { Tw, F{\sim x, \sim y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\sim y} )</td>
<td>( Fx )</td>
<td>( { Tx, F{\sim y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( F{x} )</td>
<td>( { T{x}, Fx} \in \Delta({x}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( F{x, y} )</td>
<td>( { T{x, y}, Fx} \in \Delta({x, y}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( { Tu, F{x}, F{x, y}} = \lambda(u, {u, v}) )</td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[
P = \left\{ \begin{array}{l}
x \leftarrow \neg y \\
u \leftarrow x, y \\
v \leftarrow x \\
w \leftarrow \neg x, \neg y \\
y \leftarrow \neg x \\
u \leftarrow v \\
v \leftarrow u, y \\
\end{array} \right. \\right\}
\]

<table>
<thead>
<tr>
<th>(dl)</th>
<th>(\sigma_d)</th>
<th>(\bar{\sigma})</th>
<th>(\delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(Tu)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(F{\neg x, \neg y})</td>
<td>(Fw)</td>
<td>({Tw, F{\neg x, \neg y}} = \delta(w))</td>
</tr>
<tr>
<td>3</td>
<td>(F{\neg y})</td>
<td>(Fx) (F{x}) (F{x, y})</td>
<td>({Tx, F{\neg y}} = \delta(x)) ({T{x}, Fx} \in \Delta({x})) ({T{x, y}, Fx} \in \Delta({x, y})) (...)</td>
</tr>
</tbody>
</table>
Consider

\[
P = \left\{ \begin{array}{c}
x \leftarrow \neg y \\
y \leftarrow \neg x \\
 u \leftarrow x, y \\
v \leftarrow x \\
w \leftarrow \neg x, \neg y \\
u \leftarrow v \\
v \leftarrow u, y \\
w \leftarrow \neg x, \neg y \\
\end{array} \right\}
\]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( T_u )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
& T_x & \{T_u, F_x\} \in \nabla \\
& \vdots & \vdots \\
& T_v & \{F_v, T\{x\}\} \in \Delta(v) \\
& F_y & \{T_y, F\{\neg x\}\} = \delta(y) \\
& F_w & \{T_w, F\{\neg x, \neg y\}\} = \delta(w)
\end{align*}
\]
Example: CDNL-ASP

Consider

\[ P = \left\{ x \leftarrow \sim y, \ u \leftarrow x, y, \ v \leftarrow x, \ w \leftarrow \sim x, \sim y, \ y \leftarrow \sim x, \ u \leftarrow v, \ v \leftarrow u, y \right\} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>\sigma_d</th>
<th>\overline{\sigma}</th>
<th>\delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\textbf{T}u</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textbf{T}x</td>
<td>{\textbf{T}u, \textbf{F}x} \in \nabla</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\vdots</td>
<td>\vdots</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textbf{T}v</td>
<td>{\textbf{F}v, \textbf{T}{x}} \in \Delta(v)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textbf{F}y</td>
<td>{\textbf{T}y, \textbf{F}{\sim x}} = \delta(y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\textbf{F}w</td>
<td>{\textbf{T}w, \textbf{F}{\sim x, \sim y}} = \delta(w)</td>
<td></td>
</tr>
</tbody>
</table>
Example: CDNL-ASP

Consider

\[ P = \left\{ x \leftarrow \neg y \quad u \leftarrow x, y \quad v \leftarrow x \quad w \leftarrow \neg x, \neg y \right\} \]

\[
\begin{array}{|c|c|c|c|}
\hline
dl & \sigma_d & \bar{\sigma} & \delta \\
\hline
1 & Tu & & \\
\hline
\end{array}
\]

\[
\begin{align*}
Tx & \{Tu, Fx\} \in \nabla \\
\vdots & \vdots \\
Tv & \{Fv, T\{x\}\} \in \Delta(v) \\
Fy & \{Ty, F\{\neg x\}\} = \delta(y) \\
Fw & \{Tw, F\{\neg x, \neg y\}\} = \delta(w)
\end{align*}
\]
Example: CDNL-ASP

Consider

\[ P = \left\{ \begin{array}{l} x \leftarrow \neg y \quad u \leftarrow x, y \\ y \leftarrow \neg x \quad u \leftarrow v \\ v \leftarrow u, y \end{array} \right\} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \overline{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Tx )</td>
<td>( {Tu, Fx} \in \nabla )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Tv )</td>
<td>( {Fv, T{x}} \in \Delta(v) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Fy )</td>
<td>( {Ty, F{\neg x}} = \delta(y) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Fw )</td>
<td>( {Tw, F{\neg x, \neg y}} = \delta(w) )</td>
</tr>
</tbody>
</table>
Outline

42 Motivation

43 Boolean constraints

44 Nogoods from logic programs
   - Nogoods from program completion
   - Nogoods from loop formulas

45 Conflict-driven nogood learning
   - CDNL-ASP Algorithm
   - Nogood Propagation
   - Conflict Analysis
Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on $\Delta_P$ and $\nabla$;
  - Unfounded sets $U \subseteq A(P)$
- Note that $U$ is unfounded if $EB_P(U) \subseteq A^F$
  - Note For any $a \in U$, we have $(\lambda(a, U) \setminus \{Ta\}) \subseteq A$
- An "interesting" unfounded set $U$ satisfies:
  $$\emptyset \subset U \subseteq (A(P) \setminus A^F)$$
- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of $P$
  - Note Tight programs do not yield "interesting" unfounded sets!
- Given an unfounded set $U$ and some $a \in U$, adding $\lambda(a, U)$ to $\nabla$ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$
Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on $\Delta_P$ and $\nabla$;
  - Unfounded sets $U \subseteq A(P)$
- Note that $U$ is unfounded if $EB_P(U) \subseteq A^F$
  - Note For any $a \in U$, we have $(\lambda(a, U) \setminus \{Ta\}) \subseteq A$
- An “interesting” unfounded set $U$ satisfies:

  $\emptyset \subset U \subseteq (A(P) \setminus A^F)$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of $P$
  - Note Tight programs do not yield “interesting” unfounded sets!
- Given an unfounded set $U$ and some $a \in U$, adding $\lambda(a, U)$ to $\nabla$ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$
Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on $\Delta_P$ and $\nabla$;
  - Unfounded sets $U \subseteq A(P)$

- Note that $U$ is unfounded if $EB_P(U) \subseteq A^F$
  - Note For any $a \in U$, we have $(\lambda(a, U) \setminus \{T a\}) \subseteq A$

- An “interesting” unfounded set $U$ satisfies:
  \[ \emptyset \subset U \subseteq (A(P) \setminus A^F) \]

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of $P$
  - Note Tight programs do not yield “interesting” unfounded sets!

- Given an unfounded set $U$ and some $a \in U$, adding $\lambda(a, U)$ to $\nabla$ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$
Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on $\Delta_P$ and $\nabla$;
  - Unfounded sets $U \subseteq A(P)$
- Note that $U$ is unfounded if $EB_P(U) \subseteq A^F$
  - Note For any $a \in U$, we have $(\lambda(a, U) \setminus \{Ta\}) \subseteq A$
- An “interesting” unfounded set $U$ satisfies:
  \[
  \emptyset \subset U \subseteq (A(P) \setminus A^F)
  \]
- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of $P$
  - Note Tight programs do not yield “interesting” unfounded sets!
- Given an unfounded set $U$ and some $a \in U$, adding $\lambda(a, U)$ to $\nabla$ triggers a conflict or further derivations by unit propagation
  - Note Add loop nogoods atom by atom to eventually falsify all $a \in U$
Algorithm 2: **NogoodPropagation**

<table>
<thead>
<tr>
<th>Input</th>
<th>A normal program $P$, a set $\nabla$ of nogoods, and an assignment $A$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>An extended assignment and set of nogoods.</td>
</tr>
</tbody>
</table>

$U := \emptyset$  

// unfounded set

loop

repeat

if $\delta \subseteq A$ for some $\delta \in \Delta_P \cup \nabla$ then return $(A, \nabla)$  
  // conflict

$\Sigma := \{ \delta \in \Delta_P \cup \nabla \mid \delta \setminus A = \{ \sigma \}, \sigma \notin A \}$

if $\Sigma \neq \emptyset$ then let $\sigma \in \delta \setminus A$ for some $\delta \in \Sigma$ in

  $dlevel(\sigma) := \max\{dlevel(\rho) \mid \rho \in \delta \setminus \{ \sigma \}\} \cup \{0\}$

  $A := A \diamond \sigma$

until $\Sigma = \emptyset$

if $\text{loop}(P) = \emptyset$ then return $(A, \nabla)$

$U := U \setminus A^F$

if $U = \emptyset$ then $U := \text{UnfoundedSet}(P, A)$

if $U = \emptyset$ then return $(A, \nabla)$  
  // no unfounded set $\emptyset \subset U \subseteq A(P) \setminus A^F$

let $a \in U$ in

$\nabla := \nabla \cup \{ \{ Ta \} \cup \{ FB \mid B \in EB_P(U) \} \}$

// record loop nogood

// unit-resulting nogoods

// record loop nogood

// no unfounded set $\emptyset \subset U \subseteq A(P) \setminus A^F$
Requirements for \texttt{UNFOUNDEDSet}

- Implementations of \texttt{UNFOUNDEDSet} must guarantee the following for a result $U$
  \begin{enumerate}
  \item $U \subseteq (A(P) \setminus A^F)$
  \item $EB_P(U) \subseteq A^F$
  \item $U = \emptyset$ iff there is no nonempty unfounded subset of $(A(P) \setminus A^F)$
  \end{enumerate}

- Beyond that, there are various alternatives, such as:
  \begin{itemize}
  \item Calculating the greatest unfounded set
  \item Calculating unfounded sets within strongly connected components of the positive atom dependency graph of $P$
  \item Usually, the latter option is implemented in ASP solvers
  \end{itemize}
Requirements for **UNFOUNDEDSET**

- Implementations of **UNFOUNDEDSET** must guarantee the following for a result $U$
  1. $U \subseteq (A(P) \setminus A^F)$
  2. $EB_P(U) \subseteq A^F$
  3. $U = \emptyset$ iff there is no nonempty unfounded subset of $(A(P) \setminus A^F)$

- Beyond that, there are various alternatives, such as:
  - Calculating the greatest unfounded set
  - Calculating unfounded sets within strongly connected components of the positive atom dependency graph of $P$
  - Usually, the latter option is implemented in ASP solvers
Example: NogoodPropagation

Consider

\[ P = \begin{cases} x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\ y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y \end{cases} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \overline{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\sim x, \sim y} )</td>
<td>( Fw )</td>
<td>( {Tw, F{\sim x, \sim y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\sim y} )</td>
<td>(Fx)</td>
<td>( {Tx, F{\sim y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td>( F{x} )</td>
<td>( {T{x}, Fx} \in \Delta({x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( F{x, y} )</td>
<td>( {T{x, y}, Fx} \in \Delta({x, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{\sim x} )</td>
<td>( {F{\sim x}, Fx} = \delta({\sim x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Ty )</td>
<td>( {F{\sim y}, Fy} = \delta({\sim y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{v} )</td>
<td>( {Tu, F{x, y}, F{v}} = \delta(u) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{u, y} )</td>
<td>( {F{u, y}, Tu, Ty} = \delta({u, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Tv )</td>
<td>( {Fv, T{u, y}} \in \Delta(v) )</td>
<td></td>
</tr>
</tbody>
</table>

\[ \{Tu, F\{x\}, F\{x, y\}\} = \lambda(u, \{u, v\}) \]

\[ x \]
Conflict-driven nogood learning

Outline

42 Motivation

43 Boolean constraints

44 Nogoods from logic programs
   - Nogoods from program completion
   - Nogoods from loop formulas

45 Conflict-driven nogood learning
   - CDNL-ASP Algorithm
   - Nogood Propagation
   - Conflict Analysis
Outline of Conflict Analysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_P \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$
  - Note that all but the first literal assigned at $dl$ have been unit-resulting for nogoods $\varepsilon \in \Delta_P \cup \nabla$
  - If $\sigma \in \delta$ has been unit-resulting for $\varepsilon$, we obtain a new violated nogood by resolving $\delta$ and $\varepsilon$ as follows:
    $$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $$(\delta \setminus A[\sigma]) = \{\sigma\}$$
  - Iterated resolution progresses in inverse order of assignment
  - Iterated resolution stops as soon as it generates a nogood $\delta$ containing exactly one literal $\sigma$ assigned at decision level $dl$
    - This literal $\sigma$ is called First Unique Implication Point (First-UIP)
    - All literals in $$(\delta \setminus \{\sigma\})$$ are assigned at decision levels smaller than $dl$
Conflict analysis is triggered whenever some nogood $\delta \in \Delta_P \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$

- Note that all but the first literal assigned at $dl$ have been unit-resulting for nogoods $\varepsilon \in \Delta_P \cup \nabla$
- If $\sigma \in \delta$ has been unit-resulting for $\varepsilon$, we obtain a new violated nogood by resolving $\delta$ and $\varepsilon$ as follows:

$$((\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\bar{\sigma}\}))$$

Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$

- Iterated resolution progresses in inverse order of assignment
- Iterated resolution stops as soon as it generates a nogood $\delta$ containing exactly one literal $\sigma$ assigned at decision level $dl$
- This literal $\sigma$ is called First Unique Implication Point (First-UIP)
- All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than $dl$
Outline of Conflict Analysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_P \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$
  - Note that all but the first literal assigned at $dl$ have been unit-resulting for nogoods $\varepsilon \in \Delta_P \cup \nabla$
  - If $\sigma \in \delta$ has been unit-resulting for $\varepsilon$, we obtain a new violated nogood by resolving $\delta$ and $\varepsilon$ as follows:
    $$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$
  - Iterated resolution progresses in inverse order of assignment

- Iterated resolution stops as soon as it generates a nogood $\delta$ containing exactly one literal $\sigma$ assigned at decision level $dl$
  - This literal $\sigma$ is called First Unique Implication Point (First-UIP)
  - All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than $dl$
Algorithm 3: ConflictAnalysis

Input : A non-empty violated nogood $\delta$, a normal program $P$, a set $\nabla$ of nogoods, and an assignment $A$.

Output : A derived nogood and a decision level.

loop
  let $\sigma \in \delta$ such that $\delta \setminus A[\sigma] = \{\sigma\}$ in
  $k := \max(\{dlevel(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\})$
  if $k = dlevel(\sigma)$ then
    let $\varepsilon \in \Delta_P \cup \nabla$ such that $\varepsilon \setminus A[\sigma] = \{\overline{\sigma}\}$ in
    $\delta := (\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$ // resolution
  else return $(\delta, k)$
Example: ConflictAnalysis

Consider

\[
P = \begin{cases} 
  x & \leftarrow \sim y \\
  u & \leftarrow x, y \\
  v & \leftarrow x \\
  w & \leftarrow \sim x, \sim y \\
  y & \leftarrow \sim x \\
  u & \leftarrow v \\
  v & \leftarrow u, y 
\end{cases}
\]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\sim x, \sim y} )</td>
<td>( Fw )</td>
<td>( { Tw, F{\sim x, \sim y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\sim y} )</td>
<td>( Fx )</td>
<td>( { Tx, F{\sim y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td>( F{x} )</td>
<td>( {Tx, Fx} \in \Delta({x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( F{x, y} )</td>
<td>( {Tx, y, Fx} \in \Delta({x, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{\sim x} )</td>
<td>( {F{\sim x}, Fx} = \delta({\sim x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Ty )</td>
<td>( {F{\sim y}, Fy} = \delta({\sim y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{v} )</td>
<td>( {Tu, F{x, y}, F{v}} = \delta(u) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{u, y} )</td>
<td>( {F{u, y}, Tu, Ty} = \delta({u, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Tv )</td>
<td>( {Fv, Tu, u, y} \in \Delta(v) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( {Tu, F{x}, F{x, y}} = \lambda(u, {u, v}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \{Tu, Fx\} \)  
\( \{Tu, Fx, F\{x\}\} \)
Example: ConflictAnalysis

Consider

\[ P = \{ x \leftarrow \neg y, u \leftarrow x, y, v \leftarrow x, w \leftarrow \neg x, \neg y, \\
                y \leftarrow \neg x, u \leftarrow v, v \leftarrow u, y \} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\neg x, \neg y} )</td>
<td>( Fw )</td>
<td>( {Tw, F{\neg x, \neg y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\neg y} )</td>
<td>(Fx)</td>
<td>( {Tx, F{\neg y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td>( F{x} )</td>
<td>( {T{x}, Fx} \in \Delta({x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( F{x, y} )</td>
<td>( {T{x, y}, Fx} \in \Delta({x, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{\neg x} )</td>
<td>( F{\neg x}, Fx } = \delta({\neg x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Ty )</td>
<td>( F{\neg y}, Fy } = \delta({\neg y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{v} )</td>
<td>( {Tu, F{x, y}, F{v}} = \delta(u) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{u, y} )</td>
<td>( F{u, y}, Tu, Ty } = \delta({u, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Tv )</td>
<td>( {Fv, T{u, y}} \in \Delta(v) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \{Tu, Fx\} \)
\( \{Tu, Fx, F\{x\}\} \)
\( \{Tu, F\{x\}, F\{x, y\}\} = \lambda(u, \{u, v\}) \)
Consider

\[ P = \begin{cases} 
  x \leftarrow \sim y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \sim x, \sim y \\
  y \leftarrow \sim x & u \leftarrow v & v \leftarrow u, y 
\end{cases} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \overline{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F{\sim x, \sim y}</td>
<td>Fw</td>
<td>{Tw, F{\sim x, \sim y}} = \delta(w)</td>
</tr>
<tr>
<td>3</td>
<td>F{\sim y}</td>
<td>Fx</td>
<td>{Tx, F{\sim y}} = \delta(x)</td>
</tr>
<tr>
<td></td>
<td>F{x}</td>
<td>{Tx, F{x}} = \delta({x})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F{x, y}</td>
<td>{Tx, y}, Fx \in \Delta({x, y})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T{\sim x}</td>
<td>{F{\sim x}, Fx} = \delta({\sim x})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ty</td>
<td>{F{\sim y}, Fy} = \delta({\sim y})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T{v}</td>
<td>{Tu, F{x, y}, F{v}} = \delta(u)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T{u, y}</td>
<td>{F{u, y}, Tu, Ty} = \delta({u, y})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tv</td>
<td>{Fv, T{u, y}} \in \Delta(v)</td>
<td></td>
</tr>
</tbody>
</table>

\( \{Tu, Fx\} \)
\( \{Tu, Fx, F\{x\}\} \)

\( \{Tu, F\{x\}, F\{x, y\}\} = \lambda(u, \{u, v\}) \)
Consider

\[ P = \left\{ \begin{array}{llll}
\text{x} & \leftarrow & \sim \text{y} & \\
\text{u} & \leftarrow & \text{x}, \text{y} & \\
\text{v} & \leftarrow & \text{x} & \\
\text{w} & \leftarrow & \sim \text{x}, \sim \text{y} & \\
\text{y} & \leftarrow & \sim \text{x} & \\
\text{u} & \leftarrow & \text{v} & \\
\text{v} & \leftarrow & \text{u}, \text{y} & \\
\end{array} \right\} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \overline{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{T} u )</td>
<td>( \text{T} u )</td>
<td>( \text{T} w, \text{F}{\sim \text{x}, \sim \text{y}} = \delta(\text{w}) )</td>
</tr>
<tr>
<td>2</td>
<td>( \text{F}{\sim \text{x}, \sim \text{y}} )</td>
<td>( \text{F}{\sim \text{x}, \sim \text{y}} )</td>
<td>( \text{F}{\sim \text{x}, \sim \text{y}} )</td>
</tr>
<tr>
<td>3</td>
<td>( \text{F}{\sim \text{y}} )</td>
<td>( \text{F}{\sim \text{y}} )</td>
<td>( \text{F}{\sim \text{y}} )</td>
</tr>
<tr>
<td>4</td>
<td>( \text{F}{\sim \text{y}} )</td>
<td>( \text{F}{\sim \text{y}} )</td>
<td>( \text{F}{\sim \text{y}} )</td>
</tr>
<tr>
<td>5</td>
<td>( \text{T}{\sim \text{x}} )</td>
<td>( \text{T}{\sim \text{x}} )</td>
<td>( \text{T}{\sim \text{x}} )</td>
</tr>
<tr>
<td>6</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
</tr>
<tr>
<td>7</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
</tr>
<tr>
<td>8</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
</tr>
<tr>
<td>9</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
<td>( \text{T}{\sim \text{y}} )</td>
</tr>
</tbody>
</table>

\[ \{\text{T} u, \text{F}\{\sim \text{y}\}\} \]
Consider

\[ P = \left\{ \begin{array}{ll}
  x & \leftarrow \neg y \\
  u & \leftarrow x, y \\
  v & \leftarrow x \\
  w & \leftarrow \neg x, \neg y \\
  y & \leftarrow \neg x \\
  u & \leftarrow v \\
  v & \leftarrow u, y \\
\end{array} \right\} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( \mathbf{F}{\neg x, \neg y})</td>
<td>( \mathbf{T}w, \mathbf{F}{\neg x, \neg y}) = ( \delta(w) )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( \mathbf{F}{\neg y})</td>
<td>( \mathbf{T}x, \mathbf{F}{\neg y}) = ( \delta(x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{F}{x})</td>
<td>( \mathbf{T}{x}, \mathbf{F}{x}) ( \in \Delta({x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{F}{x, y})</td>
<td>( \mathbf{T}{x, y}, \mathbf{F}{x}) ( \in \Delta({x, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{T}{\neg x})</td>
<td>( \mathbf{F}{\neg x}, \mathbf{F}{x}) = ( \delta({\neg x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{T}y )</td>
<td>( \mathbf{F}{\neg y}, \mathbf{F}{y}) = ( \delta({\neg y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{T}{v})</td>
<td>( \mathbf{T}u, \mathbf{F}{x, y}, \mathbf{F}{v}) = ( \delta(u) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{T}{u, y})</td>
<td>( \mathbf{F}{u, y}, \mathbf{T}u, \mathbf{T}y) = ( \delta({u, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{T}v )</td>
<td>( \mathbf{F}v, \mathbf{T}{u, y}) ( \in \Delta(v) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathbf{T}u, \mathbf{F}{x}, \mathbf{F}{x, y}) = ( \lambda(u, {u, v}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \mathbf{F}\{x\} \)

\( \mathbf{F}\{y\} \)

\( \mathbf{T}u \)

\( \mathbf{T}v \)
Example: ConflictAnalysis

Consider

\[ P = \left\{ \begin{array}{l}
x \leftarrow \neg y \\
u \leftarrow x, y \\
v \leftarrow x \\
w \leftarrow \neg x, \neg y \\
y \leftarrow \neg x \\
u \leftarrow v \\
v \leftarrow u, y \\
\end{array} \right\} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \overline{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( Tu )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\neg x, \neg y} )</td>
<td>( Fw )</td>
<td>( {Tw, F{\neg x, \neg y}} = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\neg y} )</td>
<td>( Fx )</td>
<td>( {Tx, F{\neg y}} = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( F{x} )</td>
<td>( {T{x}, Fx} \in \Delta({x}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( F{x, y} )</td>
<td>( {T{x, y}, Fx} \in \Delta({x, y}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T{\neg x} )</td>
<td>( {F{\neg x}, Fx} = \delta({\neg x}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Ty )</td>
<td>( {F{\neg y}, Fy} = \delta({\neg y}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Tu )</td>
<td>( {Tu, F{x, y}, F{v}} = \delta(u) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T{u, y} )</td>
<td>( {F{u, y}, Tu, Ty} = \delta({u, y}) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T{v} )</td>
<td>( {Fv, Tu, T{u, y}} \in \Delta(v) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( TV )</td>
<td>( {Tu, F{x}, F{x, y}} = \lambda(u, {u, v}) )</td>
</tr>
</tbody>
</table>

\[ \{Tu, Fx\} \]
\[ \{Tu, Fx, F\{x\}\} \]
Example: ConflictAnalysis

Consider

\[ P = \left\{ \begin{array}{ll}
  x \leftarrow \neg y & u \leftarrow x, y \\
  y \leftarrow \neg x & v \leftarrow x \\
  u \leftarrow \neg x, \neg y & w \leftarrow \neg x, \neg y \\
\end{array} \right\} \]

<table>
<thead>
<tr>
<th>dl</th>
<th>( \sigma_d )</th>
<th>( \bar{\sigma} )</th>
<th>( \delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( T_u )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( F{\neg x, \neg y} )</td>
<td>( F_w )</td>
<td>{( T_w, F{\neg x, \neg y} } = \delta(w) )</td>
</tr>
<tr>
<td>3</td>
<td>( F{\neg y} )</td>
<td>( F_x )</td>
<td>{( T_x, F{\neg y} } = \delta(x) )</td>
</tr>
<tr>
<td></td>
<td>( F{x} )</td>
<td>{( T{x}, F_x } \in \Delta({x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( F{x, y} )</td>
<td>{( T{x, y}, F_x } \in \Delta({x, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{\neg x} )</td>
<td>{( F{\neg x}, F_x } = \delta({\neg x}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T_y )</td>
<td>{( F{\neg y}, F_y } = \delta({\neg y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{v} )</td>
<td>{( T_u, F{x, y}, F{v} } = \delta(u) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{u, y} )</td>
<td>{( F{u, y}, T_u, T_y } = \delta({u, y}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T{v} )</td>
<td>{( Fv, T{u, y} } \in \Delta({v}) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( T_u )</td>
<td>{( T_u, F{x}, F{x, y} } = \lambda(u, {u, v}) )</td>
<td></td>
</tr>
</tbody>
</table>

\[ \{T_u, F\} \]
\[ \{T_u, F\{x\}, F\{x, y\}\} \]
**Example: ConflictAnalysis**

Consider

\[
P = \begin{cases}
  x \leftarrow \sim y & u \leftarrow x, y \\
  u \leftarrow \sim y & v \leftarrow x \\
  y \leftarrow \sim x & u \leftarrow v \\
  u \leftarrow v & v \leftarrow u, y
\end{cases}
\]

<table>
<thead>
<tr>
<th>dℓ</th>
<th>(σ_d)</th>
<th>(\overline{σ})</th>
<th>(δ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(F{\sim x, \sim y})</td>
<td>(T_w, F{\sim x, \sim y}) = (δ(w))</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(F{\sim y})</td>
<td>(T_x, F{\sim y}) = (δ(x))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(F{x})</td>
<td>(T{x}, Fx) (\in) (Δ({x}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(F{x, y})</td>
<td>(T{x, y}, Fx) (\in) (Δ({x, y}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(T{\sim x})</td>
<td>(F{\sim x}, Fx) = (δ({\sim x}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Ty)</td>
<td>(F{\sim y}, Fy) = (δ({\sim y}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Tu)</td>
<td>(T_u, F{x, y}, F{v}) = (δ(u))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Tu)</td>
<td>(F{u, y}, Tu, Ty) = (δ({u, y}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Tv)</td>
<td>(Fv, Tu, T{u, y}) (\in) (Δ(v))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Tu, F{x})</td>
<td>(Tu, F{x, y}) = (λ(u, {u, v}))</td>
<td></td>
</tr>
</tbody>
</table>

\[\{Tu, Fx\}\]
\[\{Tu, Fx, F\{x\}\}\]
Example: ConflictAnalysis

Consider

\[
P = \left\{ \begin{array}{l}
  x \leftarrow \neg y \\
  u \leftarrow x, y \\
  v \leftarrow x \\
  w \leftarrow \neg x, \neg y \\
  y \leftarrow \neg x \\
  u \leftarrow v \\
  v \leftarrow u, y \\
\end{array} \right. \]

<table>
<thead>
<tr>
<th>dl</th>
<th>(\sigma_d)</th>
<th>(\bar{\sigma})</th>
<th>(\delta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(Tu)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(F{\neg x, \neg y})</td>
<td>(Fw)</td>
<td>({Tw, F{\neg x, \neg y}} = \delta(w))</td>
</tr>
<tr>
<td>3</td>
<td>(F{\neg y})</td>
<td>(Fx)</td>
<td>({Tx, F{\neg y}} = \delta(x))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F{x})</td>
<td>({T{x}, Fx} \in \Delta({x}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F{x, y})</td>
<td>({T{x, y}, Fx} \in \Delta({x, y}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(T{\neg x})</td>
<td>({F{\neg x}, Fx} = \delta({\neg x}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Ty)</td>
<td>({F{\neg y}, Fy} = \delta({\neg y}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(T{v})</td>
<td>({Tu, F{x, y}, F{v}} = \delta(u))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(T{u, y})</td>
<td>({F{u, y}, Tu, Ty} = \delta({u, y}))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Tv)</td>
<td>({Fv, T{u, y}} \in \Delta(v))</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>({Tu, F{x}, F{x, y}} = \lambda(u, {u, v}))</td>
</tr>
</tbody>
</table>

\(\times\)
Consider:

$$P = \left\{ \begin{array}{cccc}
x \leftarrow \neg y & u \leftarrow x, y & v \leftarrow x & w \leftarrow \neg x, \neg y \\
y \leftarrow \neg x & u \leftarrow v & v \leftarrow u, y \\
\end{array} \right\}$$

<table>
<thead>
<tr>
<th>dl \sigma_d</th>
<th>\bar{\sigma}</th>
<th>\delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 T_u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 F{\neg x, \neg y}</td>
<td>F_w</td>
<td>{T_w, F{\neg x, \neg y}} = \delta(w)</td>
</tr>
<tr>
<td>3 F{\neg y}</td>
<td>F_x {T_x, F{\neg y}} = \delta(x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F{x} {T{x}, F_x} \in \Delta({x})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F{x, y} {T{x, y}, F_x} \in \Delta({x, y})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T{\neg x} {F{\neg x}, F_x} = \delta({\neg x})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_y {F{\neg y}, F_y} = \delta({\neg y})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T{v} {T_u, F{x, y}, F{v}} = \delta(u)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T{u, y} {F{u, y}, T_u, T_y} = \delta({u, y})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_v {F_v, T{u, y}} \in \Delta(v)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{T_u, F{x}, F{x, y}} = \lambda(u, {u, v})</td>
<td></td>
</tr>
</tbody>
</table>

Example: ConflictAnalysis
Remarks

- There always is a First-UIP at which conflict analysis terminates
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level $dl$
- The nogood $\delta$ containing First-UIP $\sigma$ is violated by $A$, viz. $\delta \subseteq A$
- We have $k = \max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$
  - After recording $\delta$ in $\nabla$ and backjumping to decision level $k$,
    $\bar{\sigma}$ is unit-resulting for $\delta$!
  - Such a nogood $\delta$ is called asserting
- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal!
Remarks

- There always is a First-UIP at which conflict analysis terminates.
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level $dl$.
- The nogood $\delta$ containing First-UIP $\sigma$ is violated by $A$, viz. $\delta \subseteq A$.
- We have $k = \max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\} \cup \{0\}\} < dl$.
  - After recording $\delta$ in $\nabla$ and backjumping to decision level $k$, $\overline{\sigma}$ is unit-resulting for $\delta$!
  - Such a nogood $\delta$ is called asserting.
  
- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal!
Remarks

- There always is a First-UIP at which conflict analysis terminates
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level $dl$
- The nogood $\delta$ containing First-UIP $\sigma$ is violated by $A$, viz. $\delta \subseteq A$
- We have $k = \max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\} \cup \{0\}\}) < dl$
  - After recording $\delta$ in $\nabla$ and backjumping to decision level $k$, $\overline{\sigma}$ is unit-resulting for $\delta$!
  - Such a nogood $\delta$ is called asserting

- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal!
Remarks

- There always is a First-UIP at which conflict analysis terminates.
  - In the worst, resolution stops at the heuristically chosen literal assigned at decision level $dl$.

- The nogood $\delta$ containing First-UIP $\sigma$ is violated by $A$, viz. $\delta \subseteq A$.

- We have $k = \max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\} \cup \{0\}) < dl$.
  - After recording $\delta$ in $\nabla$ and backjumping to decision level $k$, $\overline{\sigma}$ is unit-resulting for $\delta$.
  - Such a nogood $\delta$ is called asserting.

- Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal!
Multi-shot ASP Solving: Overview

Motivation

#program and #external declaration

Module composition

States and operations

Incremental reasoning

Boardgaming
Motivation

#program and #external declaration

Module composition

States and operations

Incremental reasoning

Boardgaming
Claim ASP is an under-the-hood technology

That is, in practice, it mainly serves as a solving engine within an encompassing software environment

Single-shot solving: \( \text{ground} \mid \text{solve} \)

Multi-shot solving: \( \text{ground} \mid \text{solve} \)

\( \Rightarrow \) continuously changing logic programs

Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology

That is, in practice, it mainly serves as a solving engine within an encompassing software environment

Single-shot solving: \(\text{ground} \mid \text{solve}\)

Multi-shot solving: \(\text{ground} \mid \text{solve}\)

\[\Rightarrow \text{continuously changing logic programs}\]

Application areas

Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology
That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: $ground \mid solve$
- Multi-shot solving: $ground \mid solve$
  - continuously changing logic programs

Application areas
Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology

That is, in practice, it mainly serves as a solving engine within an encompassing software environment.

- **Single-shot solving**: \(\text{ground} \mid \text{solve}\)
- **Multi-shot solving**: \(\text{ground} \mid \text{solve}\)

continuously changing logic programs

- Application areas
  - Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Motivation

- Claim **ASP is an under-the-hood technology**
  
  That is, in practice, it mainly serves as a solving engine within an encompassing software environment.

- **Single-shot solving**: `ground | solve`

- **Multi-shot solving**: `ground* | solve*`

  \[ \Rightarrow \text{continuously changing logic programs} \]

- Application areas
  
  Agents, Assisted Living, Robotics, Planning, Query-answering, etc.
Claim ASP is an under-the-hood technology

That is, in practice, it mainly serves as a solving engine within an encompassing software environment.

Single-shot solving: $ground \mid solve$

Multi-shot solving: $(ground^* \mid solve^*)^*$

continuously changing logic programs

Application areas

Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology

That is, in practice, it mainly serves as a solving engine within an encompassing software environment.

Single-shot solving: \( \text{ground} \mid \text{solve} \)

Multi-shot solving: \( (\text{input} \mid \text{ground}\ast \mid \text{solve}\ast)^\ast \)

\( \Rightarrow \) continuously changing logic programs

Application areas

Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology
That is, in practice, it mainly serves as a solving engine within an encompassing software environment

Single-shot solving: \( ground \mid solve \)

Multi-shot solving: \( (input \mid ground^* \mid solve^* \mid theory)^* \)
\[ \Rightarrow \text{continuously changing logic programs} \]

Application areas
Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology
That is, in practice, it mainly serves as a solving engine within an encompassing software environment

Single-shot solving: $ground \mid solve$

Multi-shot solving: $(input \mid ground^* \mid solve^* \mid theory \mid \ldots)^*$

- continuously changing logic programs

Application areas

Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Claim ASP is an under-the-hood technology
That is, in practice, it mainly serves as a solving engine
within an encompassing software environment

- Single-shot solving: $ground \mid solve$
- Multi-shot solving: $(input \mid ground^* \mid solve^* \mid theory \mid \ldots)^*$
  - continuously changing logic programs

Application areas
Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Motivation

- **Claim** ASP is an under-the-hood technology
  
  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- **Single-shot solving:** \( \text{ground} \mid \text{solve} \)

- **Multi-shot solving:** \( (\text{input} \mid \text{ground}^* \mid \text{solve}^* \mid \text{theory} \mid \ldots)^* \)
  
  \(\Rightarrow\) continuously changing logic programs

- Application areas
  
  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
Clingo = ASP + Control

- **ASP**
  - `#program <name> [ (<parameters>) ]`
    - `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - `#external mark(X,Y,P,t) : field(X,Y), player(P).`

- **Control**
  - **Python** ([www.python.org](http://www.python.org))
    - `prg.solve(), prg.ground(parts), ...`
  - C, Lua, and Prolog embeddings are available too

- **Integration**
  - in ASP: embedded scripting language (`#script`)
  - in Python: library import (import clingo)
Clingo = ASP + Control

- **ASP**
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`

- **Control**
  - Python ([www.python.org](http://www.python.org))
    - `prg.solve(), prg.ground(parts), ...`
  - C, Lua, and Prolog embeddings are available too

- **Integration**
  - in ASP: embedded scripting language (`#script`)
  - in Python: library import (import clingo)
Clingo = ASP + Control

- **ASP**
  - `#program <name> [ (<parameters>) ]`
  - Example: `#program play(t).`
  - `#external <atom> [ : <body> ]`
  - Example: `#external mark(X,Y,P,t) : field(X,Y), player(P).`

- **Control**
  - Python ([www.python.org](http://www.python.org))
    - `prg.solve(), prg.ground(parts), ...`
  - C, Lua, and Prolog embeddings are available too

- **Integration**
  - in ASP: embedded scripting language (`#script`)
  - in Python: library import (import clingo)
Clingo = ASP + Control

- **ASP**
  - #program <name> [ (<parameters>) ]
  - Example #program play(t).
  - #external <atom> [ : <body> ]
  - Example #external mark(X,Y,P,t) : field(X,Y), player(P).

- **Control**
  - Python ([www.python.org](http://www.python.org))
    - Example prg.solve(), prg.ground(parts), ...
  - C, Lua, and Prolog embeddings are available too

- **Integration**
  - in ASP: embedded scripting language (#script)
  - in Python: library import (import clingo)
Clingo = ASP + Control

ASPs

- #program <name> [ (<parameters>) ]
  - Example #program play(t).
- #external <atom> [ : <body> ]
  - Example #external mark(X,Y,P,t) : field(X,Y), player(P).

Control

- Python (www.python.org)
  - Example prg.solve(), prg.ground(parts), ...
- C, Lua, and Prolog embeddings are available too

Integration

- in ASP: embedded scripting language (#script)
- in Python: library import (import clingo)
Motivation

Clingo = ASP + Control

- **ASP**
  - #program <name> [ (<parameters>) ]
  - Example #program play(t).
  - #external <atom> [ : <body> ]
  - Example #external mark(X,Y,P,t) : field(X,Y), player(P).

- **Control**
  - Python ([www.python.org](http://www.python.org))
    - Example prg.solve(), prg.ground(parts), ...
  - C, Lua, and Prolog embeddings are available too

- **Integration**
  - in ASP: embedded scripting language (#script)
  - in Python: library import (import clingo)
Clingo = ASP + Control

- **ASP**
  - #program <name> [ (<parameters>) ]
    - Example #program play(t).
  - #external <atom> [ : <body> ]
    - Example #external mark(X,Y,P,t) : field(X,Y), player(P).

- **Control**
  - Python (www.python.org)
    - Example prg.solve(), prg.ground(parts), ...
  - C, Lua, and Prolog embeddings are available too

- **Integration**
  - in ASP: embedded scripting language (#script)
  - in Python: library import (import clingo)
Motivation

Vanilla clingo

#script (python)
def main(prg):
    parts = []
    parts.append(('base', []))
    prg.ground(parts)
    prg.solve()

#end.

Torsten Schaub (KRR@UP)
Vanilla clingo

#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
Vanilla `clingo`

```python
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```
Motivation

Hello world!

#script (python)
def main(prg):
    print("Hello world!")
#end.

$ clingo hello.lp
clingot version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models : 0+
Calls  : 1
Time   : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Hello world!

```python
#script (python)
def main(prg):
    print("Hello world!")
#end.
```

```
$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models : 0+
Calls  : 1
Time   : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
```

Torsten Schaub (KRR@UP)
#script (python)
def main(prg):
    print("Hello world!")
#end.

$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models : 0+
Calls  : 1
Time   : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Hello world!

#script (python)
def main(prg):
    print("Hello world!")
#end.

$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models : 0+
Calls  : 1
Time   : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Preview on incremental solving

#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
Preview on incremental solving

#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
Preview on incremental solving

#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
Preview on incremental solving

#program base.

p(0).

#program step (t).

p(t) :- p(t-1).

#program check (t).
#external plug(t).

:- not p(42), plug(t).
A program declaration is of form

\[
\text{#program } n(p_1, \ldots, p_k)
\]

where \( n, p_1, \ldots, p_k \) are non-integer constants

We call \( n \) the name of the declaration and \( p_1, \ldots, p_k \) its parameters

Convention Different occurrences of program declarations with the same name share the same parameters

Example

\[
\text{#program acid(k).}
\text{b(k).}
\text{c(X,k) :- a(X).}
\text{#program base.}
\text{a(2).}
\]
A program declaration is of form

\[
\text{#program } n(p_1, \ldots, p_k)
\]

where \( n, p_1, \ldots, p_k \) are non-integer constants

We call \( n \) the name of the declaration and \( p_1, \ldots, p_k \) its parameters

Convention Different occurrences of program declarations with the same name share the same parameters

Example

\[
\text{#program acid(k).}
\]
\[
b(k).
\]
\[
c(X,k) :- a(X).
\]
\[
\text{#program base.}
\]
\[
a(2).
\]
A program declaration is of form

\[ \text{#program } n(p_1, \ldots, p_k) \]

where \( n, p_1, \ldots, p_k \) are non-integer constants

- We call \( n \) the name of the declaration and \( p_1, \ldots, p_k \) its parameters

- Convention Different occurrences of program declarations with the same name share the same parameters

Example

\[ \text{#program acid(k).} \]
\[ b(k). \]
\[ c(X,k) :- a(X). \]
\[ \text{#program base.} \]
\[ a(2). \]
A program declaration is of form

```
#program n(p₁,...,pₖ)
```

where \( n, p₁, ..., pₖ \) are non-integer constants

- We call \( n \) the name of the declaration and \( p₁, ..., pₖ \) its parameters

- Convention Different occurrences of program declarations with the same name share the same parameters

Example

```
#program acid(k).
b(k).
c(X,k) :- a(X).
#program base.
a(2).
```
A program declaration is of form

```
#program n(p_1, ..., p_k)
```

where \( n, p_1, ..., p_k \) are non-integer constants

- We call \( n \) the name of the declaration and \( p_1, ..., p_k \) its parameters
- Convention Different occurrences of program declarations with the same name share the same parameters

Example

```
#program acid(k).
b(k).
c(X,k) :- a(X).
#program base.
a(2).
```
Scope of #program declarations

- The scope of an occurrence of a program declaration in a list of rules and declarations consists of the set of all rules and non-program declarations appearing between the occurrence and the next occurrence of a program declaration or the end of the list.

- Rules and non-program declarations outside the scope of any program declaration are implicitly preceded by a base program declaration.

Example

```
a(1).
#program acid(k).
b(k).
c(X,k) :- a(X).
#program base.
a(2).
```
The scope of an occurrence of a program declaration in a list of rules and declarations consists of the set of all rules and non-program declarations appearing between the occurrence and the next occurrence of a program declaration or the end of the list.

Rules and non-program declarations outside the scope of any program declaration are implicitly preceded by a base program declaration.

Example

```
a(1).
#program acid(k).
b(k).
c(X,k) :- a(X).
#program base.
a(2).
```
Scope of \texttt{#program} declarations

- The \texttt{scope} of an occurrence of a \texttt{program} declaration in a list of rules and declarations consists of the set of all rules and non-program declarations appearing between the occurrence and the next occurrence of a \texttt{program} declaration or the end of the list.

- Rules and non-program declarations outside the scope of any \texttt{program} declaration are implicitly preceded by a \texttt{base} program declaration.

\textbf{Example}

\begin{verbatim}
  a(1).
  \texttt{#program acid(k).}
  b(k).
  c(X,k) :- a(X).
  \texttt{#program base.}
  a(2).
\end{verbatim}
Scope of \texttt{#program} declarations

- The scope of an occurrence of a program declaration in a list of rules and declarations consists of the set of all rules and non-program declarations appearing between the occurrence and the next occurrence of a program declaration or the end of the list.

- Rules and non-program declarations outside the scope of any program declaration are implicitly preceded by a \texttt{base} program declaration.

Example

\begin{verbatim}
#program acid(k).
b(k).
c(X,k) :- a(X).
#program base.
a(2).
\end{verbatim}
Scope of \texttt{#program} declarations

- The \textbf{scope} of an occurrence of a program declaration in a list of rules and declarations consists of the set of all rules and non-program declarations appearing between the occurrence and the next occurrence of a program declaration or the end of the list.

- Rules and non-program declarations outside the scope of any program declaration are implicitly preceded by a \texttt{base} program declaration.

\textbf{Example}

\begin{verbatim}
a(1).
#program acid(k).
b(k).
c(X,k) :- a(X).
#program base.
a(2).
\end{verbatim}
Scope of #program declarations

- Given a list $R$ of (non-ground) rules and declarations and a name $n$, we define $R(n)$ as the set of all rules and non-program declarations in the scope of all occurrences of program declarations with name $n$.
- We often refer to $R(n)$ as a subprogram of $R$.

Example

- $R(\text{base}) = \{a(1), a(2)\}$
- $R(\text{acid}) = \{b(k), c(X, k) \leftarrow a(X)\}$

Given a name $n$ with associated parameters $(p_1, \ldots, p_k)$, the instantiation of $R(n)$ with a term tuple $(t_1, \ldots, t_k)$ results in the set

$$R(n)[p_1/t_1, \ldots, p_k/t_k]$$

obtained by replacing in $R(n)$ each occurrence of $p_i$ by $t_i$. 
Scope of \#program declarations

- Given a list \( R \) of (non-ground) rules and declarations and a name \( n \), we define \( R(n) \) as the set of all rules and non-program declarations in the scope of all occurrences of program declarations with name \( n \).
- We often refer to \( R(n) \) as a subprogram of \( R \).

Example:
- \( R(\text{base}) = \{a(1), a(2)\} \)
- \( R(\text{acid}) = \{b(k), c(X, k) \leftarrow a(X)\} \)

Given a name \( n \) with associated parameters \( (p_1, \ldots, p_k) \), the instantiation of \( R(n) \) with a term tuple \( (t_1, \ldots, t_k) \) results in the set

\[
R(n)[p_1/t_1, \ldots, p_k/t_k]
\]

obtained by replacing in \( R(n) \) each occurrence of \( p_i \) by \( t_i \).
Scope of #program declarations

- Given a list $R$ of (non-ground) rules and declarations and a name $n$, we define $R(n)$ as the set of all rules and non-program declarations in the scope of all occurrences of program declarations with name $n$.
- We often refer to $R(n)$ as a subprogram of $R$.

Example

- $R(\text{base}) = \{a(1), a(2)\}$
- $R(\text{acid}) = \{b(k), c(X, k) \leftarrow a(X)\}$

- Given a name $n$ with associated parameters $(p_1, \ldots, p_k)$, the instantiation of $R(n)$ with a term tuple $(t_1, \ldots, t_k)$ results in the set

$$R(n)[p_1/t_1, \ldots, p_k/t_k]$$

obtained by replacing in $R(n)$ each occurrence of $p_i$ by $t_i$. 
Scope of #program declarations

- Given a list $R$ of (non-ground) rules and declarations and a name $n$, we define $R(n)$ as the set of all rules and non-program declarations in the scope of all occurrences of program declarations with name $n$.
- We often refer to $R(n)$ as a subprogram of $R$.
- Example:
  - $R(\text{base}) = \{a(1), a(2)\}$
  - $R(\text{acid}) = \{b(k), c(X, k) ← a(X)\}$
- Given a name $n$ with associated parameters $(p_1, \ldots, p_k)$, the instantiation of $R(n)$ with a term tuple $(t_1, \ldots, t_k)$ results in the set 
  $$R(n)[p_1/t_1, \ldots, p_k/t_k]$$
  obtained by replacing in $R(n)$ each occurrence of $p_i$ by $t_i$. 
Scope of "#program" declarations

- Given a list $R$ of (non-ground) rules and declarations and a name $n$, we define $R(n)$ as the set of all rules and non-program declarations in the scope of all occurrences of program declarations with name $n$.
- We often refer to $R(n)$ as a subprogram of $R$.

Example

- $R(\text{base}) = \{a(1), a(2)\}$
- $R(\text{acid})[k/42] = \{b(k), c(X, k) \leftarrow a(X)\}[k/42]$

- Given a name $n$ with associated parameters $(p_1, \ldots, p_k)$, the instantiation of $R(n)$ with a term tuple $(t_1, \ldots, t_k)$ results in the set

$$R(n)[p_1/t_1, \ldots, p_k/t_k]$$

obtained by replacing in $R(n)$ each occurrence of $p_i$ by $t_i$. 
Scope of #program declarations

- Given a list $R$ of (non-ground) rules and declarations and a name $n$, we define $R(n)$ as the set of all rules and non-program declarations in the scope of all occurrences of program declarations with name $n$.
- We often refer to $R(n)$ as a subprogram of $R$.

Example

- $R(\text{base}) = \{a(1), a(2)\}$
- $R(\text{acid})[k/42] = \{b(42), c(X, 42) \leftarrow a(X)\}$

Given a name $n$ with associated parameters $(p_1, \ldots, p_k)$, the instantiation of $R(n)$ with a term tuple $(t_1, \ldots, t_k)$ results in the set

$$R(n)[p_1/t_1, \ldots, p_k/t_k]$$

obtained by replacing in $R(n)$ each occurrence of $p_i$ by $t_i$. 
Contextual grounding

- Rules are grounded relative to a set of atoms, called atom base.
- Given a set $R$ of (non-ground) rules and two sets $C, D$ of ground atoms, we define an instantiation of $R$ relative to $C$ as a ground program $\text{ground}_C(R)$ over $D$ subject to the following conditions:

$$C \subseteq D \subseteq C \cup h(\text{ground}_C(R))$$

$$\text{ground}_C(R) \subseteq \{ h(r) \leftarrow B(r)^+ \cup \{ \neg a \mid a \in B(r)^- \cap D \} \mid r \in \text{ground}(R), B(r)^+ \subseteq D \}$$

- Example: Given $R = \{ a(X) \leftarrow f(X), e(X); b(X) \leftarrow f(X), \neg e(X) \}$ and $C = \{ f(1), f(2), e(1) \}$, we obtain

$$\text{ground}_C(R) = \{ a(1) \leftarrow f(1), e(1); b(1) \leftarrow f(1), \neg e(1); b(2) \leftarrow f(2) \}$$
Contextual grounding

- Rules are grounded relative to a set of atoms, called atom base.
- Given a set $R$ of (non-ground) rules and two sets $C, D$ of ground atoms, we define an instantiation of $R$ relative to $C$ as a ground program $\text{ground}_C(R)$ over $D$ subject to the following conditions:

$$C \subseteq D \subseteq C \cup h(\text{ground}_C(R))$$

$$\text{ground}_C(R) \subseteq \{ h(r) \leftarrow B(r)^+ \cup \{ \neg a \mid a \in B(r)^- \cap D \} \mid r \in \text{ground}(R), B(r)^+ \subseteq D \}$$

- Example: Given $R = \{ a(X) \leftarrow f(X), e(X); b(X) \leftarrow f(X), \neg e(X) \}$ and $C = \{ f(1), f(2), e(1) \}$, we obtain

$$\text{ground}_C(R) = \{ a(1) \leftarrow f(1), e(1); b(1) \leftarrow f(1), \neg e(1); b(2) \leftarrow f(2) \}$$
Contextual grounding

- Rules are grounded relative to a set of atoms, called atom base.
- Given a set $R$ of (non-ground) rules and two sets $C, D$ of ground atoms, we define an instantiation of $R$ relative to $C$ as a ground program $\text{ground}_C(R)$ over $D$ subject to the following conditions:

$$C \subseteq D \subseteq C \cup h(\text{ground}_C(R))$$

$$\text{ground}_C(R) \subseteq \{ h(r) \leftarrow B(r)^+ \cup \{ \neg a \mid a \in B(r)^- \cap D \} \mid r \in \text{ground}(R), B(r)^+ \subseteq D \}$$

- Example: Given $R = \{ a(X) \leftarrow f(X), e(X); b(X) \leftarrow f(X), \neg e(X) \}$ and $C = \{ f(1), f(2), e(1) \}$, we obtain

$$\text{ground}_C(R) = \{ a(1) \leftarrow f(1), e(1); \ b(1) \leftarrow f(1), \neg e(1) \ b(2) \leftarrow f(2) \}$$
An external declaration is of form

```
#external a : B
```

where $a$ is an atom and $B$ a rule body

A logic program with external declarations is said to be extensible

Example

```
#external e(X) : f(X), X < 2.
f(1..2).
a(X) :- f(X), e(X).
b(X) :- f(X), not e(X).
```
An external declaration is of form

$$\texttt{#external } a : B$$

where $a$ is an atom and $B$ a rule body

A logic program with external declarations is said to be extensible

Example

```prolog
#external e(X) : f(X), X < 2.
f(1..2).
a(X) :- f(X), e(X).
b(X) :- f(X), not e(X).
```

Torsten Schaub (KRR@UP)
An external declaration is of form

\[
\text{#external } a : B
\]

where \( a \) is an atom and \( B \) a rule body.

A logic program with external declarations is said to be extensible.

Example

\[
\text{#external } e(X) : f(X), X < 2.
f(1..2).
a(X) :- f(X), e(X).
b(X) :- f(X), \neg e(X).
\]
Grounding extensible logic programs

Given an extensible program $R$, we define

$$Q = \{ a \leftarrow B, \varepsilon \mid \text{(#external } a : B) \in R \}$$

$$R' = \{ a \leftarrow B \in R \}$$

Note An external declaration is treated as a rule $a \leftarrow B, \varepsilon$ where $\varepsilon$ is a ground marking atom

Given an atom base $C$, the ground instantiation of an extensible logic program $R$ is defined as a (ground) logic program $P$ with externals $E$ where

$$P = \{ r \in \text{ground}_{C \cup \{ \varepsilon \}}(R' \cup Q) \mid \varepsilon \notin B(r) \}$$

$$E = \{ h(r) \mid r \in \text{ground}_{C \cup \{ \varepsilon \}}(R' \cup Q), \varepsilon \in B(r) \}$$

Note The marking atom $\varepsilon$ appears neither in $P$ nor $E$, respectively, and $P$ is a logic program over $C \cup E \cup h(P)$
Grounding extensible logic programs

- Given an extensible program $R$, we define
  
  $Q = \{ a \leftarrow B, \varepsilon \mid (#\text{external } a : B) \in R \}$
  
  $R' = \{ a \leftarrow B \in R \}$

- **Note** An external declaration is treated as a rule $a \leftarrow B, \varepsilon$ where $\varepsilon$ is a ground marking atom

- Given an atom base $C$, the ground instantiation of an extensible logic program $R$ is defined as a (ground) logic program $P$ with externals $E$

  
  $P = \{ r \in \text{ground}_{C \cup \{\varepsilon\}}(R' \cup Q) \mid \varepsilon \notin B(r) \}$

  $E = \{ h(r) \mid r \in \text{ground}_{C \cup \{\varepsilon\}}(R' \cup Q), \varepsilon \in B(r) \}$

- **Note** The marking atom $\varepsilon$ appears neither in $P$ nor $E$, respectively, and $P$ is a logic program over $C \cup E \cup h(P)$
Grounding extensible logic programs

- Given an extensible program $R$, we define
  
  \[
  Q = \{ a \leftarrow B, \varepsilon \mid (\#\text{external } a : B) \in R \} \\
  R' = \{ a \leftarrow B \in R \}
  \]

- Note An external declaration is treated as a rule $a \leftarrow B, \varepsilon$ where $\varepsilon$ is a ground marking atom.

- Given an atom base $C$, the ground instantiation of an extensible logic program $R$ is defined as a (ground) logic program $P$ with externals $E$ where
  
  \[
  P = \{ r \in \text{ground}_{C \cup \{ \varepsilon \}}(R' \cup Q) \mid \varepsilon \notin B(r) \} \\
  E = \{ h(r) \mid r \in \text{ground}_{C \cup \{ \varepsilon \}}(R' \cup Q), \varepsilon \in B(r) \}
  \]

- Note The marking atom $\varepsilon$ appears neither in $P$ nor $E$, respectively, and $P$ is a logic program over $C \cup E \cup h(P)$. 

Torsten Schaub (KRR@UP)
Grounding extensible logic programs

- Given an extensible program $R$, we define
  
  \[
  Q = \{ a \leftarrow B, \varepsilon \mid \text{(#external } a : B) \in R \} \\
  R' = \{ a \leftarrow B \in R \}
  \]

- Note An external declaration is treated as a rule $a \leftarrow B, \varepsilon$ where $\varepsilon$ is a ground marking atom.

- Given an atom base $C$, the ground instantiation of an extensible logic program $R$ is defined as a (ground) logic program $P$ with externals $E$ where

  \[
  P = \{ r \in \text{ground}_{C \cup \{ \varepsilon \}} (R' \cup Q) \mid \varepsilon \notin B(r) \} \\
  E = \{ h(r) \mid r \in \text{ground}_{C \cup \{ \varepsilon \}} (R' \cup Q), \varepsilon \in B(r) \}
  \]

- Note The marking atom $\varepsilon$ appears neither in $P$ nor $E$, respectively, and $P$ is a logic program over $C \cup E \cup h(P)$.
Example

- Extensible program

  ```prolog
  #external e(X) : f(X), g(X).
  f(1). f(2).
  a(X) :- f(X), e(X).
  b(X) :- f(X), not e(X).
  
  Atom base \{g(1)\} \cup \{\varepsilon\}
  ```

- Ground program

  ```prolog
  f(1). f(2).
  a(1) :- f(1), e(1).
  b(1) :- f(1), not e(1).
  b(2) :- f(2).
  
  with externals \{e(1)\}
  ```
Example

- **Extensible program**

  \[
  e(X) :- f(X), g(X), \varepsilon.
  f(1). f(2).
  a(X) :- f(X), e(X).
  b(X) :- f(X), \text{not } e(X).
  \]

  Atom base \(\{g(1)\} \cup \{\varepsilon\}\)

- **Ground program**

  \[
  f(1). f(2).
  a(1) :- f(1), e(1).
  b(1) :- f(1), \text{not } e(1). \quad b(2) :- f(2).
  \]

  with externals \(\{e(1)\}\)
Example

Extensible program

\[
\begin{align*}
  e(1) & : - f(1), g(1), \varepsilon. \\
  e(2) & : - f(2), g(2), \varepsilon. \\
  f(1). & \\
  f(2). & \\
  a(X) & : - f(X), e(X). \\
  b(X) & : - f(X), \text{not } e(X).
\end{align*}
\]

Atom base \(\{g(1)\} \cup \{\varepsilon\}\)

Ground program

\[
\begin{align*}
  f(1). & \\
  f(2). & \\
  a(1) & : - f(1), e(1). \\
  b(1) & : - f(1), \text{not } e(1). \\
  b(2) & : - f(2).
\end{align*}
\]

with externals \(\{e(1)\}\)
Example

- Extensible program

\[
\begin{align*}
e(1) & : - f(1), g(1), \varepsilon. & e(2) & : - f(2), g(2), \varepsilon. \\
f(1). & f(2). \\
a(1) & : - f(1), e(1). & a(2) & : - f(2), e(2). \\
b(1) & : - f(1), \text{not } e(1). & b(2) & : - f(2), \text{not } e(2). \\
\end{align*}
\]

Atom base \( \{g(1)\} \cup \{\varepsilon\} \)

- Ground program

\[
\begin{align*}
f(1). & f(2). \\
a(1) & : - f(1), e(1). \\
b(1) & : - f(1), \text{not } e(1). & b(2) & : - f(2). \\
\end{align*}
\]

with externals \( \{e(1)\} \)
Example

- Extensible program

\[
\begin{align*}
e(1) & : - f(1), g(1), \varepsilon. \\
e(2) & : - f(2), g(2), \varepsilon. \\
f(1). & f(2). \\
a(1) & : - f(1), e(1). \\
b(1) & : - f(1), \text{not } e(1). \quad b(2) : - f(2), \text{not } e(2). \\
\end{align*}
\]

Atom base \( \{g(1)\} \cup \{\varepsilon\} \)

- Ground program

\[
\begin{align*}
f(1). & f(2). \\
a(1) & : - f(1), e(1). \\
b(1) & : - f(1), \text{not } e(1). \quad b(2) : - f(2). \\
\end{align*}
\]

with externals \( \{e(1)\} \)
Example

- Extensible program

\[
e(1) :- f(1), g(1), \varepsilon. \quad e(2) :- f(2), g(2), \varepsilon.
\]

\[
f(1). \quad f(2).
\]

\[
a(1) :- f(1), e(1). \quad a(2) :- f(2), e(2).
\]

\[
b(1) :- f(1), \text{not } e(1). \quad b(2) :- f(2), \text{not } e(2).
\]

Atom base \( \{g(1)\} \cup \{\varepsilon\} \)

- Ground program

\[
f(1). \quad f(2).
\]

\[
a(1) :- f(1), e(1).
\]

\[
b(1) :- f(1), \text{not } e(1). \quad b(2) :- f(2).
\]

with externals \( \{e(1)\} \)
Example

- Extensible program

\[
\begin{align*}
\text{e}(1) & : - f(1), \text{g}(1), \varepsilon. \\
\text{e}(2) & : - f(2), \text{g}(2), \varepsilon.
\end{align*}
\]

\[
\begin{align*}
f(1). \\
f(2).
\end{align*}
\]

\[
\begin{align*}
\text{a}(1) & : - f(1), \text{e}(1). \\
\text{a}(2) & : - f(2), \text{e}(2).
\end{align*}
\]

\[
\begin{align*}
\text{b}(1) & : - f(1), \text{not e}(1). \\
\text{b}(2) & : - f(2), \text{not e}(2).
\end{align*}
\]

Atom base \( \{\text{g}(1)\} \cup \{\varepsilon\} \)

- Ground program

\[
\begin{align*}
f(1). \\
f(2).
\end{align*}
\]

\[
\begin{align*}
a(1) & : - f(1), \text{e}(1). \\
b(1) & : - f(1), \text{not e}(1). \\
b(2) & : - f(2).
\end{align*}
\]

with externals \( \{\text{e}(1)\} \)
Example

- Extensible program
  
  \[
  \begin{align*}
  e(1) & :- f(1), g(1), \varepsilon. \\
  f(1). & f(2). \\
  a(1) & :- f(1), e(1). \\
  b(1) & :- f(1), \text{not } e(1). \quad b(2) :- f(2).
  \end{align*}
  \]

  Atom base \( \{g(1)\} \cup \{\varepsilon\} \)

- Ground program
  
  \[
  \begin{align*}
  f(1). & f(2). \\
  a(1) & :- f(1), e(1). \\
  b(1) & :- f(1), \text{not } e(1). \quad b(2) :- f(2).
  \end{align*}
  \]

  with externals \( \{e(1)\} \)
Example

- Extensible program

  \[\begin{align*}
  &e(1) :- f(1), g(1), \varepsilon. \\
  &f(1). f(2). \\
  &a(1) :- f(1), e(1). \\
  &b(1) :- f(1), \text{not } e(1). \\
  &b(2) :- f(2).
  \end{align*}\]

  Atom base \(\{g(1)\} \cup \{\varepsilon\}\)

- Ground program

  \[\begin{align*}
  &f(1). f(2). \\
  &a(1) :- f(1), e(1). \\
  &b(1) :- f(1), \text{not } e(1). \\
  &b(2) :- f(2).
  \end{align*}\]

  with externals \(\{e(1)\}\)
Example

- **Extensible program**
  
  \[
  \begin{align*}
    e(1) & : - f(1), g(1), \varepsilon. \\
    f(1). & f(2). \\
    a(1) & : - f(1), e(1). \\
    b(1) & : - f(1), \text{not } e(1). \quad b(2) : - f(2).
  \end{align*}
  \]

  Atom base \( \{ g(1) \} \cup \{ \varepsilon \} \)

- **Ground program**
  
  \[
  \begin{align*}
    f(1). & f(2). \\
    a(1) & : - e(1). \\
    b(1) & : - \text{not } e(1). \quad b(2).
  \end{align*}
  \]

  with externals \( \{ \text{e(1)} \} \)
Outline

46 Motivation
47 #program and #external declaration
48 Module composition
49 States and operations
50 Incremental reasoning
51 Boardgaming
The assembly of subprograms can be characterized by means of modules:

A module $\mathcal{P}$ is a triple $(P, I, O)$ consisting of
- a (ground) program $P$ over $\text{ground}(\mathcal{A})$ and
- sets $I, O \subseteq \text{ground}(\mathcal{A})$ such that
  - $I \cap O = \emptyset$,
  - $A(P) \subseteq I \cup O$, and
  - $h(P) \subseteq O$

The elements of $I$ and $O$ are called input and output atoms denoted by $I(\mathcal{P})$ and $O(\mathcal{P})$

Similarly, we refer to (ground) program $P$ by $P(\mathcal{P})$
The assembly of subprograms can be characterized by means of modules:

A module $\mathcal{P}$ is a triple $(P, I, O)$ consisting of

- a (ground) program $P$ over $\text{ground}(\mathcal{A})$ and
- sets $I, O \subseteq \text{ground}(\mathcal{A})$ such that
  - $I \cap O = \emptyset$,
  - $A(P) \subseteq I \cup O$, and
  - $h(P) \subseteq O$

The elements of $I$ and $O$ are called input and output atoms denoted by $I(\mathcal{P})$ and $O(\mathcal{P})$

Similarly, we refer to (ground) program $P$ by $P(\mathcal{P})$
The assembly of subprograms can be characterized by means of modules:

A module $\mathcal{P}$ is a triple $(P, I, O)$ consisting of
- a (ground) program $P$ over $\text{ground}(A)$ and
- sets $I, O \subseteq \text{ground}(A)$ such that
  - $I \cap O = \emptyset$,
  - $A(P) \subseteq I \cup O$, and
  - $h(P) \subseteq O$

The elements of $I$ and $O$ are called input and output atoms
- denoted by $I(\mathcal{P})$ and $O(\mathcal{P})$
- Similarly, we refer to (ground) program $P$ by $P(\mathcal{P})$
The assembly of subprograms can be characterized by means of modules:

- A module $\mathbb{P}$ is a triple $(P, I, O)$ consisting of
  - a (ground) program $P$ over $\text{ground}(A)$ and
  - sets $I, O \subseteq \text{ground}(A)$ such that
    - $I \cap O = \emptyset$, 
    - $A(P) \subseteq I \cup O$, and
    - $h(P) \subseteq O$

- The elements of $I$ and $O$ are called input and output atoms
  - denoted by $I(\mathbb{P})$ and $O(\mathbb{P})$

- Similarly, we refer to (ground) program $P$ by $P(\mathbb{P})$
The assembly of subprograms can be characterized by means of modules:

A module $\mathcal{P}$ is a triple $(P, I, O)$ consisting of

- a (ground) program $P$ over $\text{ground}(\mathcal{A})$ and
- sets $I, O \subseteq \text{ground}(\mathcal{A})$ such that
  - $I \cap O = \emptyset$,
  - $A(P) \subseteq I \cup O$, and
  - $h(P) \subseteq O$

The elements of $I$ and $O$ are called input and output atoms
  - denoted by $I(\mathcal{P})$ and $O(\mathcal{P})$

Similarly, we refer to (ground) program $P$ by $P(\mathcal{P})$
Two modules $\mathbb{P}$ and $\mathbb{Q}$ are compositional, if

$$O(\mathbb{P}) \cap O(\mathbb{Q}) = \emptyset \text{ and }$$

$$O(\mathbb{P}) \cap S = \emptyset \text{ or } O(\mathbb{Q}) \cap S = \emptyset$$

for every strongly connected component $S$ of $\mathbb{P} \cup \mathbb{Q}$

Recursion between two modules to be joined is disallowed
Recursion within each module is allowed

The join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ is defined as the module

$$\left( \mathbb{P}(\mathbb{P}) \cup \mathbb{P}(\mathbb{Q}), \ (I(\mathbb{P}) \setminus O(\mathbb{Q})) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})), \ O(\mathbb{P}) \cup O(\mathbb{Q}) \right)$$

provided that $\mathbb{P}$ and $\mathbb{Q}$ are compositional
Two modules $\mathbb{P}$ and $\mathbb{Q}$ are compositional, if

\[
O(\mathbb{P}) \cap O(\mathbb{Q}) = \emptyset \quad \text{and} \quad O(\mathbb{P}) \cap S = \emptyset \quad \text{or} \quad O(\mathbb{Q}) \cap S = \emptyset
\]

for every strongly connected component $S$ of $\mathbb{P} \cup \mathbb{Q}$.

Recursion between two modules to be joined is disallowed.
Recursion within each module is allowed.

The join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ is defined as the module

\[
( \mathbb{P} \sqcup \mathbb{Q}, \ (I(\mathbb{P}) \setminus O(\mathbb{Q})) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})), \ O(\mathbb{P}) \cup O(\mathbb{Q}) )
\]

provided that $\mathbb{P}$ and $\mathbb{Q}$ are compositional.
Composing modules

- Two modules $P$ and $Q$ are compositional, if
  - $O(P) \cap O(Q) = \emptyset$ and
    - $O(P) \cap S = \emptyset$ or $O(Q) \cap S = \emptyset$
    for every strongly connected component $S$ of $P(P) \cup P(Q)$

- Recursion between two modules to be joined is disallowed
- Recursion within each module is allowed

The join, $P \sqcup Q$, of two modules $P$ and $Q$ is defined as the module

$$( P(P) \cup P(Q), (I(P) \setminus O(Q)) \cup (I(Q) \setminus O(P)), O(P) \cup O(Q) )$$

provided that $P$ and $Q$ are compositional
Composing modules

- Two modules $P$ and $Q$ are compositional, if
  - $O(P) \cap O(Q) = \emptyset$ and
  - $O(P) \cap S = \emptyset$ or $O(Q) \cap S = \emptyset$ for every strongly connected component $S$ of $P(P) \cup P(Q)$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion within each module is allowed

- The join, $P \sqcup Q$, of two modules $P$ and $Q$ is defined as the module
  \[
  ( P(P) \cup P(Q), (I(P) \setminus O(Q)) \cup (I(Q) \setminus O(P)), O(P) \cup O(Q) )
  \]
  provided that $P$ and $Q$ are compositional
Two modules \( P \) and \( Q \) are compositional, if

- \( O(P) \cap O(Q) = \emptyset \) and
- \( O(P) \cap S = \emptyset \) or \( O(Q) \cap S = \emptyset \)

for every strongly connected component \( S \) of \( P(P) \cup P(Q) \)

Note

- Recursion between two modules to be joined is disallowed
- Recursion within each module is allowed

The join, \( P \sqcup Q \), of two modules \( P \) and \( Q \) is defined as the module

\[
(P(P) \cup P(Q), (I(P) \setminus O(Q)) \cup (I(Q) \setminus O(P)), O(P) \cup O(Q))
\]

provided that \( P \) and \( Q \) are compositional
Composing modules

- Two modules $P$ and $Q$ are compositional, if
  - $O(P) \cap O(Q) = \emptyset$ and
  - $O(P) \cap S = \emptyset$ or $O(Q) \cap S = \emptyset$
    for every strongly connected component $S$ of $P(P) \cup P(Q)$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion within each module is allowed

- The join, $P \sqcup Q$, of two modules $P$ and $Q$ is defined as the module
  \[
  \left( P(P) \cup P(Q), (I(P) \setminus O(Q)) \cup (I(Q) \setminus O(P)), O(P) \cup O(Q) \right)
  \]
  provided that $P$ and $Q$ are compositional
Composing modules

- Two modules $P$ and $Q$ are compositional, if
  - $O(P) \cap O(Q) = \emptyset$ and
  - $O(P) \cap S = \emptyset$ or $O(Q) \cap S = \emptyset$
    for every strongly connected component $S$ of $P(P) \cup P(Q)$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion within each module is allowed

- The join, $P \sqcup Q$, of two modules $P$ and $Q$ is defined as the module

$$\left( P(P) \cup P(Q), \ (I(P) \setminus O(Q)) \cup (I(Q) \setminus O(P)), \ O(P) \cup O(Q) \right)$$

provided that $P$ and $Q$ are compositional
Composing logic programs with externals

- **Idea** Each ground instruction induces a module to be joined with the module representing the current program state.

- Given an atom base $C$, a (non-ground) extensible program $R$ induces the module

$$\mathcal{R}(C) = (P, (C \cup E) \setminus h(P), h(P))$$

via the ground program $P$ with externals $E$ obtained from $R$ and $C$.

- **Note** $E \setminus h(P)$ consists of atoms stemming from non-overwritten external declarations.
Composing logic programs with externals

- Idea Each ground instruction induces a module to be joined with the module representing the current program state.

- Given an atom base $C$, a (non-ground) extensible program $R$ induces the module

$$R(C) = (P, (C \cup E) \setminus h(P), h(P))$$

via the ground program $P$ with externals $E$ obtained from $R$ and $C$.

- Note $E \setminus h(P)$ consists of atoms stemming from non-overwritten external declarations.
Composing logic programs with externals

- **Idea** Each ground instruction induces a module to be joined with the module representing the current program state.

- **Given** an atom base $C$, a (non-ground) extensible program $R$ induces the module

$$R(C) = (P, (C \cup E) \setminus h(P), h(P))$$

via the ground program $P$ with externals $E$ obtained from $R$ and $C$.

- **Note** $E \setminus h(P)$ consists of atoms stemming from non-overwritten external declarations.
Example

- Atom base $C = \{g(1)\}$
- Extensible program $R$
  
  ```
  #external e(X) : f(X), g(X)
  f(1). f(2).
  a(X) :- f(X), e(X).
  b(X) :- f(X), not e(X).
  ```

- Module $R(C) = (P, (C \cup E) \setminus h(P), h(P))$

\[
\begin{pmatrix}
  f(1), f(2), \\
  a(1) \leftarrow f(1), e(1), \\
  b(1) \leftarrow f(1), \neg e(1), \\
  b(2) \leftarrow f(2)
\end{pmatrix}, \begin{pmatrix}
  g(1), \\
  e(1)
\end{pmatrix}, \begin{pmatrix}
  f(1), f(2), \\
  a(1), b(1), b(2)
\end{pmatrix}
\]
Example

- Atom base $C = \{g(1)\}$
- Ground program $P$
  
  $f(1). f(2).$
  
  $a(1) :- f(1), e(1).$
  
  $b(1) :- f(1), \neg e(1). b(2) :- f(2).$

  with externals $E = \{e(1)\}$
- Module $\mathbb{R}(C) = (P, (C \cup E) \setminus h(P), h(P))$

$$=
\begin{pmatrix}
  f(1), f(2), \\
  a(1) \leftarrow f(1), e(1), \\
  b(1) \leftarrow f(1), \neg e(1), \\
  b(2) \leftarrow f(2)
\end{pmatrix}
\cup
\begin{pmatrix}
  g(1), \\
  e(1)
\end{pmatrix}
\cup
\begin{pmatrix}
  f(1), f(2), \\
  a(1), \\
  b(1), b(2)
\end{pmatrix}$$
Example

- Atom base $C = \{g(1)\}$
- Ground program $P$
  
  \[
  f(1) \cdot f(2).
  a(1) \leftarrow f(1), e(1).
  b(1) \leftarrow f(1), \neg e(1).
  b(2) \leftarrow f(2).
  \]

  with externals $E = \{e(1)\}$
- Module $\mathbb{R}(C) = (P, (C \cup E) \setminus h(P), h(P))$

\[
\begin{pmatrix}
  f(1), f(2),
  a(1) \leftarrow f(1), e(1),
  b(1) \leftarrow f(1), \neg e(1),
  b(2) \leftarrow f(2)
\end{pmatrix},
\begin{pmatrix}
  g(1),
  e(1)
\end{pmatrix},
\begin{pmatrix}
  f(1), f(2),
  a(1),
  b(1), b(2)
\end{pmatrix}
\]
Example

- **Atom base** \( C = \{g(1)\} \)
- **Extensible program** \( R \)
  
  ```prolog
  #external e(X) : f(X), g(X)
  f(1). f(2).
  a(X) :- f(X), e(X).
  b(X) :- f(X), not e(X).
  ```

- **Module** \( \mathbb{R}(C) = (P, (C \cup E) \setminus h(P), h(P)) \)

\[
\begin{pmatrix}
  f(1), f(2),
  a(1) \leftarrow f(1), e(1),
  b(1) \leftarrow f(1), \neg e(1),
  b(2) \leftarrow f(2)
\end{pmatrix}
\begin{pmatrix}
  g(1),
  e(1)
\end{pmatrix}
\begin{pmatrix}
  f(1), f(2),
  a(1),
  b(1), b(2)
\end{pmatrix}
\]
Capturing program states by modules

- Each program state is captured by a module
  - The input and output atoms of each module provide the atom base

- The initial program state is given by the empty module
  \[ P_0 = (\emptyset, \emptyset, \emptyset) \]

- The program state succeeding \( P_i \) is captured by the module
  \[ P_{i+1} = P_i \cup R_{i+1}(I(P_i) \cup O(P_i)) \]

where \( R_{i+1}(I(P_i) \cup O(P_i)) \) captures the result of grounding an extensible program \( R \) relative to atom base \( I(P_i) \cup O(P_i) \)

- Note: The join leading to \( P_{i+1} \) can be undefined in case the constituent modules are non-compositional
Capturing program states by modules

- Each program state is captured by a module
  - The input and output atoms of each module provide the atom base

- The initial program state is given by the empty module
  \[ P_0 = (\emptyset, \emptyset, \emptyset) \]

- The program state succeeding \( P_i \) is captured by the module
  \[ P_{i+1} = P_i \cup R_{i+1}(I(P_i) \cup O(P_i)) \]
  where \( R_{i+1}(I(P_i) \cup O(P_i)) \) captures the result of grounding an extensible program \( R \) relative to atom base \( I(P_i) \cup O(P_i) \)

- Note: The join leading to \( P_{i+1} \) can be undefined in case the constituent modules are non-compositional
Capturing program states by modules

- Each program state is captured by a module
  - The input and output atoms of each module provide the atom base

- The initial program state is given by the empty module
  \[ \mathbb{P}_0 = (\emptyset, \emptyset, \emptyset) \]

- The program state succeeding \( \mathbb{P}_i \) is captured by the module
  \[ \mathbb{P}_{i+1} = \mathbb{P}_i \sqcup \mathbb{R}_{i+1}(I(\mathbb{P}_i) \cup O(\mathbb{P}_i)) \]

  where \( \mathbb{R}_{i+1}(I(\mathbb{P}_i) \cup O(\mathbb{P}_i)) \) captures the result of grounding an extensible program \( R \) relative to atom base \( I(\mathbb{P}_i) \cup O(\mathbb{P}_i) \)

- Note: The join leading to \( \mathbb{P}_{i+1} \) can be undefined in case the constituent modules are non-compositional
Capturing program states by modules

- Each program state is captured by a module
  - The input and output atoms of each module provide the atom base

- The initial program state is given by the empty module
  \[ P_0 = (\emptyset, \emptyset, \emptyset) \]

- The program state succeeding \( P_i \) is captured by the module
  \[ P_{i+1} = P_i \sqcup R_{i+1}(I(P_i) \cup O(P_i)) \]

  where \( R_{i+1}(I(P_i) \cup O(P_i)) \) captures the result of grounding an extensible program \( R \) relative to atom base \( I(P_i) \cup O(P_i) \)

- Note: The join leading to \( P_{i+1} \) can be undefined in case the constituent modules are non-compositional
Capturing program states by modules

- Each program state is captured by a module
  - The input and output atoms of each module provide the atom base

- The initial program state is given by the empty module

\[ P_0 = (\emptyset, \emptyset, \emptyset) \]

- The program state succeeding \( P_i \) is captured by the module

\[ P_{i+1} = P_i \cup R_{i+1}(I(P_i) \cup O(P_i)) \]

where \( R_{i+1}(I(P_i) \cup O(P_i)) \) captures the result of grounding an extensible program \( R \) relative to atom base \( I(P_i) \cup O(P_i) \)

- Note The join leading to \( P_{i+1} \) can be undefined in case the constituent modules are non-compositional
Capturing program states by modules

Let \((R_i)_{i>0}\) be a sequence of (non-ground) extensible programs, and let \(P_{i+1}\) be the ground program with externals \(E_{i+1}\) obtained from \(R_{i+1}\) and \(I(P_i) \cup O(P_i)\).

If \(\bigcup_{i \geq 0} P_i\) is compositional, then

1. \(P(\bigcup_{i \geq 0} P_i) = \bigcup_{i > 0} P_i\)
2. \(I(\bigcup_{i \geq 0} P_i) = \bigcup_{i > 0} E_i \setminus \bigcup_{i > 0} h(P_i)\)
3. \(O(\bigcup_{i \geq 0} P_i) = \bigcup_{i > 0} h(P_i)\)
Capturing program states by modules

Let \((R_i)_{i>0}\) be a sequence of (non-ground) extensible programs, and let \(P_{i+1}\) be the ground program with externals \(E_{i+1}\) obtained from \(R_{i+1}\) and \(I(P_i) \cup O(P_i)\)

If \(\bigsqcup_{i \geq 0} P_i\) is compositional, then

1. \(P(\bigsqcup_{i \geq 0} P_i) = \bigsqcup_{i>0} P_i\)
2. \(I(\bigsqcup_{i \geq 0} P_i) = \bigsqcup_{i>0} E_i \setminus \bigsqcup_{i>0} h(P_i)\)
3. \(O(\bigsqcup_{i \geq 0} P_i) = \bigsqcup_{i>0} h(P_i)\)
Outline

46 Motivation

47 #program and #external declaration

48 Module composition

49 States and operations

50 Incremental reasoning

51 Boardgaming
A clingo state is a triple 

$$(R, \mathcal{P}, V)$$

where

- $R$ is a collection of extensible (non-ground) logic programs
- $\mathcal{P}$ is a module
- $V$ is a three-valued assignment over $I(\mathcal{P})$
A **clingo state** is a triple

\[(R, P, V)\]

where

- \(R = (R_c)_{c \in C}\) is a collection of extensible (non-ground) logic programs where \(C\) is the set of all non-integer constants
- \(P\) is a module
- \(V\) is a three-valued assignment over \(I(P)\)
A *clingo* state is a triple

$$(R, P, V)$$

where

- $R = (R_c)_{c \in C}$ is a collection of extensible (non-ground) logic programs where $C$ is the set of all non-integer constants
- $P$ is a module
- $V = (V^t, V^u)$ is a three-valued assignment over $I(P)$
  where $V^f = I(P) \setminus (V^t \cup V^u)$
A clingo state is a triple

\((R, P, V)\)

where

- \(R = (R_c)_{c \in C}\) is a collection of extensible (non-ground) logic programs where \(C\) is the set of all non-integer constants
- \(P\) is a module
- \(V = (V^t, V^u)\) is a three-valued assignment over \(I(P)\)
  where \(V^f = I(P) \setminus (V^t \cup V^u)\)

Note: Input atoms in \(I(P)\) are taken to be false by default.
create

- \textit{create}(R) : \mapsto (R, \mathbb{P}, \mathbb{V})

for a list \( R \) of (non-ground) rules and declarations where

- \( R = (R(c))_{c \in \mathcal{C}} \)
- \( \mathbb{P} = (\emptyset, \emptyset, \emptyset) \)
- \( \mathbb{V} = (\emptyset, \emptyset) \)
create

- **create**(*R*): \( \mapsto (\mathcal{R}, \mathcal{P}, \mathcal{V}) \)

for a list *R* of (non-ground) rules and declarations where

- \( \mathcal{R} = (\mathcal{R}(c))_{c \in \mathcal{C}} \)
- \( \mathcal{P} = (\emptyset, \emptyset, \emptyset) \)
- \( \mathcal{V} = (\emptyset, \emptyset) \)
add

- **add** \((R) : (R_1, \mathbb{P}, V) \mapsto (R_2, \mathbb{P}, V)\)

for a list \(R\) of (non-ground) rules and declarations where

- \(R_1 = (R_c)_{c \in C}\) and \(R_2 = (R_c \cup R(c))_{c \in C}\)
add \((R)\) : \((R_1, P, V) \mapsto (R_2, P, V)\)

for a list \(R\) of (non-ground) rules and declarations where

- \(R_1 = (R_c)_{c \in C}\) and \(R_2 = (R_c \cup R(c))_{c \in C}\)
**ground**

- \( \text{ground}((n, p_n)_{n \in \mathbb{N}}) : (R, P_1, V_1) \mapsto (R, P_2, V_2) \)

for a collection \((n, p_n)_{n \in \mathbb{N}}\) such that \(N \subseteq C\) and \(p_n \in \mathcal{T}^k\) for some \(k\)

where

- \(P_2 = P_1 \cup R(I(P_1) \cup O(P_1))\)
- \(R(I(P_1) \cup O(P_1))\) is the module obtained from
  - extensible program \(\bigcup_{n \in \mathbb{N}} R_n[p/p_n]\) and
  - atom base \(I(P_1) \cup O(P_1)\)

for \((R_c)_{c \in C} = R\)

- \(V_2^t = \{ a \in I(P_2) \mid V_1(a) = t \}\)
- \(V_2^u = \{ a \in I(P_2) \mid V_1(a) = u \}\)
ground

\[ \text{ground}((n, p_n)_{n \in \mathbb{N}}) : (R, \mathbb{P}_1, V_1) \mapsto (R, \mathbb{P}_2, V_2) \]

for a collection \((n, p_n)_{n \in \mathbb{N}}\) such that \(N \subseteq C\) and \(p_n \in \mathcal{T}^k\) for some \(k\) where

- \(\mathbb{P}_2 = \mathbb{P}_1 \cup \mathbb{R}(I(\mathbb{P}_1) \cup O(\mathbb{P}_1))\)
- and \(\mathbb{R}(I(\mathbb{P}_1) \cup O(\mathbb{P}_1))\) is the module obtained from
  - extensible program \(\bigcup_{n \in \mathbb{N}} R_n[p/p_n]\) and
  - atom base \(I(\mathbb{P}_1) \cup O(\mathbb{P}_1)\)
- for \((R_c)_{c \in C} = R\)

\[ V_2^t = \{ a \in I(\mathbb{P}_2) \mid V_1(a) = t \} \]
\[ V_2^u = \{ a \in I(\mathbb{P}_2) \mid V_1(a) = u \} \]
The external status of an atom is eliminated once it becomes defined by a rule in some added program. This is accomplished by module composition, namely, the elimination of output atoms from input atoms. Jointly grounded subprograms are treated as a single subprogram. Ground\((((n,p),(n,p))(s)) = \text{ground}((n,p))(s)\) while \(\text{ground}((n,p))(\text{ground}((n,p))(s))\) leads to two non-compositional modules whenever \(h(R_n) \neq \emptyset\). Inputs stemming from added external declarations are set to false.
Notes

- The external status of an atom is eliminated once it becomes defined by a rule in some added program. This is accomplished by module composition, namely, the elimination of output atoms from input atoms.
- Jointly grounded subprograms are treated as a single subprogram.
- \( \text{ground}((n, p), (n, p))(s) = \text{ground}((n, p))(s) \) while \( \text{ground}((n, p))(\text{ground}((n, p))(s)) \) leads to two non-compositional modules whenever \( h(R_n) \neq \emptyset \).
- Inputs stemming from added external declarations are set to false.
Notes

- The external status of an atom is eliminated once it becomes defined by a rule in some added program. This is accomplished by module composition, namely, the elimination of output atoms from input atoms.

- Jointly grounded subprograms are treated as a single subprogram.

- $\text{ground}((n, p), (n, p))(s) = \text{ground}((n, p))(s)$ while $\text{ground}((n, p))(\text{ground}((n, p))(s))$ leads to two non-compositional modules whenever $h(R_n) \neq \emptyset$.

- Inputs stemming from added external declarations are set to false.
The external status of an atom is eliminated once it becomes defined by a rule in some added program. This is accomplished by module composition, namely, the elimination of output atoms from input atoms.

Jointly grounded subprograms are treated as a single subprogram.

\[
ground((n, p), (n, p))(s) = ground((n, p))(s)\]

while

\[
ground((n, p))(\ground((n, p))(s))\]

leads to two non-compositional modules whenever \( h(R_n) \neq \emptyset \).

Inputs stemming from added external declarations are set to false.
assignExternal

- $\text{assignExternal}(a, v) : (R, P, V_1) \mapsto (R, P, V_2)$

for a ground atom $a$ and $v \in \{t, u, f\}$ where

- if $v = t$
  - $V_t^2 = V_t^1 \cup \{a\}$ if $a \in \mathcal{I}(P)$, and $V_t^2 = V_t^1$ otherwise
  - $V_u^2 = V_u^1 \setminus \{a\}$

- if $v = u$
  - $V_t^2 = V_t^1 \setminus \{a\}$
  - $V_u^2 = V_u^1 \cup \{a\}$ if $a \in \mathcal{I}(P)$, and $V_u^2 = V_u^1$ otherwise

- if $v = f$
  - $V_t^2 = V_t^1 \setminus \{a\}$
  - $V_u^2 = V_u^1 \setminus \{a\}$

- Note: Only input atoms, that is, non-overwritten externals are affected
assignExternal

- \( \text{assignExternal}(a, v) : (R, P, V_1) \mapsto (R, P, V_2) \)

for a ground atom \( a \) and \( v \in \{ t, u, f \} \) where

- if \( v = t \)
  - \( V_2^t = V_1^t \cup \{a\} \) if \( a \in I(P) \), and \( V_2^t = V_1^t \) otherwise
  - \( V_2^u = V_1^u \setminus \{a\} \)

- if \( v = u \)
  - \( V_2^t = V_1^t \setminus \{a\} \)
  - \( V_2^u = V_1^u \cup \{a\} \) if \( a \in I(P) \), and \( V_2^u = V_1^u \) otherwise

- if \( v = f \)
  - \( V_2^t = V_1^t \setminus \{a\} \)
  - \( V_2^u = V_1^u \setminus \{a\} \)

- Note Only input atoms, that is, non-overwritten externals are affected
assignExternal

- \( assignExternal(a, \nu) : (R, P, V_1) \mapsto (R, P, V_2) \)

for a ground atom \( a \) and \( \nu \in \{t, u, f\} \) where

- if \( \nu = t \)
  - \( V^t_2 = V^t_1 \cup \{a\} \) if \( a \in I(P) \), and \( V^t_2 = V^t_1 \) otherwise
  - \( V^u_2 = V^u_1 \setminus \{a\} \)

- if \( \nu = u \)
  - \( V^t_2 = V^t_1 \setminus \{a\} \)
  - \( V^u_2 = V^u_1 \cup \{a\} \) if \( a \in I(P) \), and \( V^u_2 = V^u_1 \) otherwise
  - \( V^f_2 = V^f_1 \setminus \{a\} \)

- if \( \nu = f \)
  - \( V^t_2 = V^t_1 \setminus \{a\} \)
  - \( V^u_2 = V^u_1 \setminus \{a\} \)

Note Only input atoms, that is, non-overwritten externals are affected
releaseExternal

- \( \text{releaseExternal}(a) : (R, P_1, V_1) \mapsto (R, P_2, V_2) \)

for a ground atom \( a \) where

- \( P_2 = (P(P_1), I(P_1) \setminus \{a\}, O(P_1) \cup \{a\}) \) if \( a \in I(P_1) \), and
  \( P_2 = P_1 \) otherwise
- \( V^t_2 = V^t_1 \setminus \{a\} \)
- \( V^u_2 = V^u_1 \setminus \{a\} \)

**Notes**

releaseExternal only affects input atoms; defined atoms remain unaffected.
A released atom can never be re-defined, neither by a rule nor an external declaration.
A released (input) atom is made permanently false, since it is neither defined by any rule nor part of the input atoms.
releaseExternal

- \text{releaseExternal}(a) : (R, P_1, V_1) \mapsto (R, P_2, V_2)

for a ground atom \(a\) where

- \(P_2 = (P(P_1), I(P_1) \setminus \{a\}, O(P_1) \cup \{a\})\) if \(a \in I(P_1)\), and \(P_2 = P_1\) otherwise
- \(V^t_2 = V^t_1 \setminus \{a\}\)
- \(V^u_2 = V^u_1 \setminus \{a\}\)

Notes

- releaseExternal only affects input atoms; defined atoms remain unaffected
- A released atom can never be re-defined, neither by a rule nor an external declaration
- A released (input) atom is made permanently false, since it is neither defined by any rule nor part of the input atoms
releaseExternal

- \( \text{releaseExternal}(a) : (R, P_1, V_1) \mapsto (R, P_2, V_2) \)

for a ground atom \( a \) where

- \( P_2 = (P(P_1), I(P_1) \setminus \{a\}, O(P_1) \cup \{a\}) \) if \( a \in I(P_1) \), and
- \( P_2 = P_1 \) otherwise

- \( V^t_2 = V^t_1 \setminus \{a\} \)
- \( V^u_2 = V^u_1 \setminus \{a\} \)

Notes

- \( \text{releaseExternal} \) only affects input atoms; defined atoms remain unaffected
- A released atom can never be re-defined, neither by a rule nor an external declaration
- A released (input) atom is made permanently false, since it is neither defined by any rule nor part of the input atoms
releaseExternal

- \( \text{releaseExternal}(a) : (R, P_1, V_1) \mapsto (R, P_2, V_2) \)

for a ground atom \( a \) where

- \( P_2 = (P(P_1), I(P_1) \setminus \{a\}, O(P_1) \cup \{a\}) \) if \( a \in I(P_1) \), and \( P_2 = P_1 \) otherwise
- \( V^t_2 = V^t_1 \setminus \{a\} \)
- \( V^u_2 = V^u_1 \setminus \{a\} \)

Notes

- \( \text{releaseExternal} \) only affects input atoms; defined atoms remain unaffected
- A released atom can never be re-defined, neither by a rule nor an external declaration
- A released (input) atom is made permanently false, since it is neither defined by any rule nor part of the input atoms
releaseExternal

- \( \text{releaseExternal}(a) : (R, P_1, V_1) \mapsto (R, P_2, V_2) \)

for a ground atom \( a \) where

- \( P_2 = (P(P_1), I(P_1) \setminus \{a\}, O(P_1) \cup \{a\}) \) if \( a \in I(P_1) \), and
- \( P_2 = P_1 \) otherwise

- \( V^t_2 = V^t_1 \setminus \{a\} \)
- \( V^u_2 = V^u_1 \setminus \{a\} \)

Notes

- \( \text{releaseExternal} \) only affects input atoms; defined atoms remain unaffected
- A released atom can never be re-defined, neither by a rule nor an external declaration
- A released (input) atom is made permanently false, since it is neither defined by any rule nor part of the input atoms
solve

- \textbf{solve}((A^t, A^f)) : (R, P, V) \mapsto (R, P, V) \) prints the set

\[ \{ X \mid X \text{ is a stable model of } P \text{ wrt } V \text{ st } A^t \subseteq X \text{ and } A^f \cap X = \emptyset \} \]

where the stable models of a module \( P \) wrt an assignment \( V \) are given by the stable models of the program

\[ P(P) \cup \{ a \leftarrow \mid a \in V^t \} \cup \{ \{ a \} \leftarrow \mid a \in V^u \} \]
states and operations

- $solve((A^t, A^f)) : (R, P, V) \mapsto (R, P, V)$ prints the set

$$\{ X \mid X \text{ is a stable model of } P \text{ wrt } V \text{ st } A^t \subseteq X \text{ and } A^f \cap X = \emptyset $$

where the stable models of a module $P$ wrt an assignment $V$ are given by the stable models of the program

$$P(P) \cup \{ a \leftarrow \mid a \in V^t \} \cup \{ \{ a \} \leftarrow \mid a \in V^u \}$$
A script declaration is of form

```
#script(python) P #end
```

where \( P \) is a Python program

- Analogously for Lua
- main routine exercises control (from within clingo, not from Python)

```
#script(python)
def main(prg):
    prg.ground(["base",[]])
    prg.solve()
#end.
```

```
#script(python)
def main(prg):
    prg.ground(["acid",[42]])
    prg.solve()
#end.
```
A script declaration is of form

\[
\texttt{#script(python) } P \texttt{#end}
\]

where \( P \) is a Python program

Analogously for Lua

main routine exercises control (from within clingo, not from Python)

```
#script(python)
def main(prg):
    prg.ground([("base",[])])
    prg.solve()
#end.
```

```
#script(python)
def main(prg):
    prg.ground(["acid",[42]])
    prg.solve()
#end.
```
A script declaration is of form

```
#script(python) P #end
```

where $P$ is a Python program

Analogously for Lua

main routine exercises control (from within clingo, not from Python)

```
#script(python)
def main(prg):
    prg.ground(["base",[]])
    prg.solve()
#end.
```

```
#script(python)
def main(prg):
    prg.ground(["acid",[42]])
    prg.solve()
#end.
```
A script declaration is of form

```
#script(python)  P  #end
```

where $P$ is a Python program

Analogously for Lua

**main** routine exercises control (from within *clingo*, not from Python)

```
#script(python)
def main(prg):
    prg.ground(["base",[]])
    prg.solve()
#end.
```

```
#script(python)
def main(prg):
    prg.ground(["acid",[42]])
    prg.solve()
#end.
```
A script declaration is of form

```
#script(python)  P  #end
```

where $P$ is a Python program

Analogously for Lua

main routine exercises control (from within clingo, not from Python)

Example:

```
#script(python)
def main(prg):
    prg.ground(["base", []])
    prg.solve()
#end.
```

```
#script(python)
def main(prg):
    prg.ground(["acid", [42]])
    prg.solve()
#end.
```
A script declaration is of form

```
#script(python) P #end
```

where $P$ is a Python program

Analogously for Lua

**main** routine exercises control (from within *clingo*, not from Python)

**Example**

```python
#script(python)
def main(prg):
    prg.ground([("base",[])])
    prg.solve()
#end.
```

```python
#script(python)
def main(prg):
    prg.ground([("acid",[42]])
    prg.solve()
#end.
```
States and operations

A script declaration is of form

```
#script(python) P #end
```

where $P$ is a Python program

Analogously for Lua

- **main** routine exercises control (from within *clingo*, not from Python)

Examples

```
#script(python)
def main(prg):
    prg.ground([("base",[])])
    prg.solve()
#end.
```

```
#script(python)
def main(prg):
    prg.ground([("acid",[42])])
    prg.solve()
#end.
```
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]],["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
Example

```
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground([("succ", [1]),("succ", [2])])
    prg.solve()
    prg.ground([("succ", [3])])
    prg.solve()
#end.
```
Example

#internal p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#internal p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground([("succ", [1]),("succ", [2])])
    prg.solve()
    prg.ground([("succ", [3])])
    prg.solve()
#end.
States and operations

Extensible programs

- Initial clingo state

\[(R_0, \mathcal{P}_0, V_0) = ((R(\text{base}), R(\text{succ})), (\emptyset, \emptyset, \emptyset), (\emptyset, \emptyset))\]

where

\[
R(\text{base}) = \left\{ \begin{array}{l}
#\text{external } p(1) \quad p(0) \leftarrow p(3) \\
#\text{external } p(2) \quad p(0) \leftarrow \neg p(0) \\
#\text{external } p(3)
\end{array} \right. 
\]

\[
R(\text{succ}) = \left\{ \begin{array}{l}
#\text{external } p(n + 3) \\
p(n) \leftarrow p(n + 3) \\
p(n) \leftarrow \neg p(n + 1), \neg p(n + 2)
\end{array} \right. 
\]

- Initial atom base \( I(\mathcal{P}_0) \cup O(\mathcal{P}_0) = \emptyset \)
Extensible programs

- **Initial *clingo* state**

  \[(R_0, P_0, V_0) = ((R(base), R(succ)), (\emptyset, \emptyset, \emptyset), (\emptyset, \emptyset))\]

  where

  \[
  R(base) = \begin{cases} 
  \#external \ p(1) & p(0) \leftarrow p(3) \\
  \#external \ p(2) & p(0) \leftarrow \neg p(0) \\
  \#external \ p(3) & 
  \end{cases}
  \]

  \[
  R(succ) = \begin{cases} 
  \#external \ p(n+3) & \\
  p(n) \leftarrow p(n+3) \\
  p(n) \leftarrow \neg p(n+1), \neg p(n+2) & 
  \end{cases}
  \]

- **Initial atom base** \[I(P_0) \cup O(P_0) = \emptyset\]
Extensible programs

- Initial `clingo` state, or more precisely, state of `clingo` object ‘prg’

\[
(R_0, P_0, V_0) = ((R(\text{base}), R(\text{succ})), (\emptyset, \emptyset, \emptyset), (\emptyset, \emptyset))
\]

where

\[
R(\text{base}) = \begin{cases}
\#\text{external } p(1) & p(0) \leftarrow p(3) \\
\#\text{external } p(2) & p(0) \leftarrow \sim p(0) \\
\#\text{external } p(3)
\end{cases}
\]

\[
R(\text{succ}) = \begin{cases}
\#\text{external } p(n + 3) \\
p(n) \leftarrow p(n + 3) \\
p(n) \leftarrow \sim p(n + 1), \sim p(n + 2)
\end{cases}
\]

- Initial atom base \( I(P_0) \cup O(P_0) = \emptyset \)
Extensible programs

- Initial *clingo* state, or more precisely, state of *clingo* object ‘prg’

\[
create(R) = ((R(base), R(succ)), (\emptyset, \emptyset, \emptyset), (\emptyset, \emptyset))
\]

where \( R \) is the list of rules and declarations in Line 1-8 and

\[
R(base) = \left\{ \begin{array}{l}
\text{#external } p(1) \quad p(0) \leftarrow p(3) \\
\text{#external } p(2) \quad p(0) \leftarrow \neg p(0) \\
\text{#external } p(3)
\end{array} \right.
\]

\[
R(succ) = \left\{ \begin{array}{l}
\text{#external } p(n + 3) \\
p(n) \leftarrow p(n + 3) \\
p(n) \leftarrow \neg p(n + 1), \neg p(n + 2)
\end{array} \right.
\]

- Initial atom base \( I(P_0) \cup O(P_0) = \emptyset \)
Extensible programs

- Initial *clingo* state, or more precisely, state of *clingo* object ‘prg’

\[
create(R) = \left((R(\text{base}), R(\text{succ})), (\emptyset, \emptyset, \emptyset), (\emptyset, \emptyset)\right)
\]

where \( R \) is the list of rules and declarations in Line 1-8 and

\[
R(\text{base}) = \left\{ \begin{array}{l}
\text{#external } p(1) \quad p(0) \leftarrow p(3) \\
\text{#external } p(2) \quad p(0) \leftarrow \neg p(0) \\
\text{#external } p(3) 
\end{array} \right\}
\]

\[
R(\text{succ}) = \left\{ \begin{array}{l}
\text{#external } p(n + 3) \\
p(n) \leftarrow p(n + 3) \\
p(n) \leftarrow \neg p(n + 1), \neg p(n + 2)
\end{array} \right\}
\]

- Initial atom base \( I(P_0) \cup O(P_0) = \emptyset \)

- Note *create*(\( R \)) is invoked implicitly to create *clingo* object ‘prg’
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], "succ", [2])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun

def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()

#end.
prg.ground([("base", [])])

- Global *clingo* state \((R_0, P_0, V_0)\), including atom base \(\emptyset\)
- Input Extensible program \(R(\text{base})\)
- Output Module

\[
R_1(\emptyset) = (P_1, E_1, \{p(0)\}) \quad \text{where}
P_1 = \{p(0) \leftarrow p(3); \ p(0) \leftarrow \lnot p(0)\}
E_1 = \{p(1), p(2), p(3)\}
\]

- Result *clingo* state

\[
(R_1, P_1, V_1) = (R_0, P_0 \sqcup R_1(\emptyset), V_0)
\]

where

\[
P_1 = P_0 \sqcup R_1(\emptyset) = (\emptyset, \emptyset, \emptyset) \sqcup (P_1, E_1, \{p(0)\})
= (\{p(0) \leftarrow p(3); \ p(0) \leftarrow \lnot p(0)\}, \{p(1), p(2), p(3)\}, \{p(0)\})
\]
prg.ground(["base", []])

- Global \textit{clingo} state \((R_0, \mathcal{P}_0, V_0)\), including atom base \(\emptyset\)
- Input Extensible program \(R(\text{base})\)
- Output Module

\[
R_1(\emptyset) = (P_1, E_1, \{p(0)\}) \quad \text{where}
\]
\[
P_1 = \{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}
\]
\[
E_1 = \{p(1), p(2), p(3)\}
\]

- Result \textit{clingo} state

\[
(R_1, \mathcal{P}_1, V_1) = (R_0, \mathcal{P}_0 \uplus R_1(\emptyset), V_0)
\]

where

\[
\mathcal{P}_1 = \mathcal{P}_0 \uplus R_1(\emptyset) = (\emptyset, \emptyset, \emptyset) \uplus (P_1, E_1, \{p(0)\})
\]
\[
= (\{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}, \{p(1), p(2), p(3)\}, \{p(0)\})
\]
prg.ground([("base", [])])

- Global \textit{clingo} state $(R_0, P_0, V_0)$, including atom base $\emptyset$
- Input Extensible program $R(\text{base})$
- Output Module

$$R_1(\emptyset) = (P_1, E_1, \{p(0)\}) \quad \text{where}$$

$$P_1 = \{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}$$

$$E_1 = \{p(1), p(2), p(3)\}$$

- Result \textit{clingo} state

$$(R_1, P_1, V_1) = (R_0, P_0 \sqcup R_1(\emptyset), V_0)$$

where

$$P_1 = P_0 \sqcup R_1(\emptyset) = (\emptyset, \emptyset, \emptyset) \sqcup (P_1, E_1, \{p(0)\})$$

$$= (\{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}, \{p(1), p(2), p(3)\}, \{p(0)\})$$
States and operations

\[ \text{prg.ground}([\{"base\}, []]) \]

- Global *clingo* state \((R_0, P_0, V_0)\), including atom base \(\emptyset\)
- Input Extensible program \(R(\text{base})\)
- Output Module
  \[
  R_1(\emptyset) = (P_1, E_1, \{p(0)\}) \quad \text{where}
  \]
  \[
  P_1 = \{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}
  \]
  \[
  E_1 = \{p(1), p(2), p(3)\}
  \]
- Result *clingo* state
  \[
  (R_1, P_1, V_1) = (R_0, P_0 \sqcup R_1(\emptyset), V_0)
  \]
  where
  \[
  P_1 = P_0 \sqcup R_1(\emptyset) = (\emptyset, \emptyset, \emptyset) \sqcup (P_1, E_1, \{p(0)\})
  \]
  \[
  = (\{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}, \{p(1), p(2), p(3)\}, \{p(0)\})
  \]
prg.ground(["base", []])

- Global *clingo* state \((R_0, P_0, V_0)\), including atom base \(\emptyset\)
- Input Extensible program \(R(\text{base})\)
- Output Module

\[
R_1(\emptyset) = (P_1, E_1, \{p(0)\}) \quad \text{where} \\
P_1 = \{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\} \\
E_1 = \{p(1), p(2), p(3)\}
\]

- Result *clingo* state

\[
(R_1, P_1, V_1) = (R_0, P_0 \sqcup R_1(\emptyset), V_0)
\]

where

\[
P_1 = P_0 \sqcup R_1(\emptyset) = (\emptyset, \emptyset, \emptyset) \sqcup (P_1, E_1, \{p(0)\}) \\
= (\{p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0)\}, \{p(1), p(2), p(3)\}, \{p(0)\})
\]
Example

```prolog
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]],["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
```
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun

def main(prg):
    prg.ground(["base", []])
>> prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
prg.assign_external(Fun("p", [3]), True)

- **Global clingo state** $(R_1, P_1, V_1)$
- **Input assignment** $p(3) \mapsto t$
- **Result clingo state**

$$ (R_2, P_2, V_2) = (R_0, P_1, \{p(3)\}, \emptyset) $$
prg.assign_external(Fun("p", [3]), True)

- Global \textit{clingo} state \((R_1, P_1, V_1)\)
- Input assignment \(p(3) \leftrightarrow t\)
- Result \textit{clingo} state

\[(R_2, P_2, V_2) = (R_0, P_1, (\{p(3)\}, \emptyset))\]
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun

def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()

#end.
prg.solve()

- Global *clingo* state \( (R_2, P_2, V_2) \)
- Input empty assignment

- Result *clingo* state

\[
(R_2, P_2, V_2) = (R_0, P_1, (\{p(3)\}, \emptyset))
\]

- Print stable model \( \{p(0), p(3)\} \) of \( P_2 \) wrt \( V_2 \)
prg.solve()

- Global `clingo` state \( (R_2, P_2, V_2) \)
- Input empty assignment
- Result `clingo` state

\[
(R_2, P_2, V_2) = (R_0, P_1, (\{p(3)\}, \emptyset))
\]

- Print stable model \( \{p(0), p(3)\} \) of \( P_2 \) wrt \( V_2 \)
prg.solve()

- Global \textit{clingo} state \((R_2, P_2, V_2)\)
- Input empty assignment
- Result \textit{clingo} state

\[(R_2, P_2, V_2) = (R_0, P_1, \{p(3)\}, \emptyset)\]

- Print stable model \(\{p(0), p(3)\}\) of \(P_2\) wrt \(V_2\)
prg.solve()

- Global *cling*o state \((R_2, P_2, V_2)\)
- Input empty assignment
- Result *cling*o state
  \[(R_2, P_2, V_2) = (R_0, P_1, \{p(3)\}, \emptyset)\]
- Print stable model \(\{p(0), p(3)\}\) of \(P_2\) wrt \(V_2\)
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]],["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
>> prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]],"succ", [2])]
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
prg.assign_external(Fun("p",[3]),False)

- Global **clingo** state \((R_2, P_2, V_2)\)
- Input assignment \(p(3) \mapsto f\)
- Result **clingo** state

\[(R_3, P_3, V_3) = (R_0, P_1, (\emptyset, \emptyset))\]
prg.assign_external(Fun("p",[3]),False)

- Global `clingo` state \((R_2, P_2, V_2)\)
- Input assignment \(p(3) \mapsto f\)
- Result `clingo` state

\[(R_3, P_3, V_3) = (R_0, P_1, (\emptyset, \emptyset))\]
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground([("succ", [1]),("succ", [2])])
    prg.solve()
    prg.ground([("succ", [3])])
    prg.solve()
#end.
prg.solve()

- Global *clingo* state \((R_3, \mathbb{P}_3, V_3)\)
- Input empty assignment
- Result *clingo* state

\[(R_3, \mathbb{P}_3, V_3) = (R_0, \mathbb{P}_1, (\emptyset, \emptyset))\]
- Print no stable model of \(\mathbb{P}_3\) wrt \(V_3\)
prg.solve()

- Global *clingo* state \((R_3, P_3, V_3)\)
- Input empty assignment
- Result *clingo* state

\[(R_3, P_3, V_3) = (R_0, P_1, (\emptyset, \emptyset))\]

- Print no stable model of \(P_3\) wrt \(V_3\)
prg.solve()

- Global \textit{clingo} state \((R_3, P_3, V_3)\)
- Input empty assignment
- Result \textit{clingo} state

\[(R_3, P_3, V_3) = (R_0, P_1, (\emptyset, \emptyset))\]

- Print no stable model of \(P_3\) wrt \(V_3\)
Example

\#external \texttt{p(1;2;3)}.
\texttt{p(0) :- p(3).}
\texttt{p(0) :- not p(0).}

\#program \texttt{succ(n)}.
\#external \texttt{p(n+3)}.
\texttt{p(n) :- p(n+3).}
\texttt{p(n) :- not p(n+1), not p(n+2).}

\#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground([("succ", [1]),("succ", [2])])
    prg.solve()
    prg.ground([("succ", [3])])
    prg.solve()
#end.
States and operations

Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
prg.ground([("succ", [1]), ("succ", [2])])

- Global clingo state \((R_3, P_3, V_3)\), including atom base 
  \(I(P_3) \cup O(P_3) = \{p(0), p(1), p(2), p(3)\}\)
- Input Extensible program \(R(succ)[n/1] \cup R(succ)[n/2]\)
- Output Module

\[ R_4(I(P_3) \cup O(P_3)) = (P_4, \{p(0), p(4), \}, \{p(1), p(2)\}) \]

\[ P_4 = \{p(1) \leftarrow p(4); p(1) \leftarrow \neg p(2), \neg p(3);\}
  \{p(2) \leftarrow p(5); p(2) \leftarrow \neg p(3), \neg p(4)\} \]

\[ E_4 = \{p(4), p(5)\} \]

- Result clingo state

\[ (R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3) \]
prg.ground(["succ", [1]], ["succ", [2]])

- Global `clingo` state \((R_3, P_3, V_3)\), including atom base
  \(I(P_3) \cup O(P_3) = \{p(0), p(1), p(2), p(3)\}\)
- Input Extensible program \(R(\text{succ})[n/1] \cup R(\text{succ})[n/2]\)
- Output Module

\[
R_4(I(P_3) \cup O(P_3)) = \left( P_4, \begin{cases}
  \{p(0), p(4), p(3), p(5)\}, \\
  \{p(1), p(2)\}
\end{cases}, \begin{cases}
  p(1) \leftarrow p(4); \\
  p(1) \leftarrow \neg p(2), \neg p(3); \\
  p(2) \leftarrow p(5); \\
  p(2) \leftarrow \neg p(3), \neg p(4)
\end{cases}\right)
\]

where

\[P_4 = \begin{cases}
  p(1) \leftarrow p(4); \\
  p(1) \leftarrow \neg p(2), \neg p(3); \\
  p(2) \leftarrow p(5); \\
  p(2) \leftarrow \neg p(3), \neg p(4)
\end{cases}\]

\[E_4 = \{p(4), p(5)\}\]

- Result `clingo` state

\[(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)\]
\[
\text{prg.ground}\left([\left("\text{succ}\", [1]\right),\left("\text{succ}\", [2]\right)]\right)
\]

Result \textit{clingo} state

\[
(R_4, P_4, V_4) = (R_0, P_3 \sqcup R_4(I(P_3) \cup O(P_3)), V_3)
\]

where

\[
P_4 = \left( \begin{array}{l}
\{ p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3); \}
\{ p(0) \leftarrow \neg p(0); \ p(2) \leftarrow p(5); \ p(2) \leftarrow \neg p(3), \neg p(4) \}
\end{array} \right)
\]

\[
P_3 = \left( \begin{array}{l}
\{ p(0) \leftarrow p(3) \}
\{ p(0) \leftarrow \neg p(0) \}
\end{array} \right)
\]

\[
R_4(I(P_3) \cup O(P_3)) = \left( \begin{array}{l}
\{ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3); \}
\{ p(2) \leftarrow p(5); \ p(2) \leftarrow \neg p(3), \neg p(4) \}
\end{array} \right)
\]
prg.ground([("succ", [1]), ("succ", [2])])

- Result *clingo* state

\[
(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)
\]

where

\[
P_4 = \left( \{ p(0) \leftarrow p(3); p(1) \leftarrow p(4); p(1) \leftarrow \neg p(2), \neg p(3) \} \right) \{ p(0), p(1) \} \{ p(4) \} \{ p(0), p(1), p(4) \} \{ p(0) \} \{ p(1), p(2), p(3) \} \{ p(4) \}
\]

\[
P_3 = \left( \{ p(0) \leftarrow p(3) \} \{ p(0) \leftarrow \neg p(0) \} \{ p(1), p(2), p(3) \} \{ p(0) \} \right)
\]

\[
R_4(I(P_3) \cup O(P_3)) = \left( \{ p(1) \leftarrow p(4); p(1) \leftarrow \neg p(2), \neg p(3) \} \{ p(0), p(4) \} \{ p(1) \} \{ p(1), p(4) \} \{ p(0) \} \{ p(1), p(4) \} \{ p(0) \} \{ p(1), p(4) \} \{ p(0) \} \{ p(1), p(4) \} \{ p(0) \}
\]
prg.ground([("succ", [1]), ("succ", [2])])

Result clingo state

\[(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)\]

where

\[P_4 = \left( \begin{array}{l}
\{p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \sim p(2), \sim p(3)\}, \ \{p(4)\}, \ \{p(0), p(1)\} \\
\{p(0) \leftarrow \sim p(0); \ p(2) \leftarrow p(5); \ p(2) \leftarrow \sim p(3), \sim p(4)\} \end{array} \right)\]

\[P_3 = \left( \begin{array}{l}
\{p(0) \leftarrow p(3)\}, \ \{p(1), p(2), p(3)\}, \ \{p(0)\} \\
\{p(0) \leftarrow \sim p(0)\} \end{array} \right)\]

\[R_4(I(P_3) \cup O(P_3)) = \left( \begin{array}{l}
\{p(1) \leftarrow p(4); \ p(1) \leftarrow \sim p(2), \sim p(3)\}, \ \{p(0), p(4)\}, \ \{p(1)\} \\
\{p(2) \leftarrow p(5); \ p(2) \leftarrow \sim p(3), \sim p(4)\} \end{array} \right)\]
prg.ground([("succ", [1]), ("succ", [2])])

Result `clingo` state

\[(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)\]

where

\[P_4 = \left(\begin{array}{l}
\{p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3)\}
\cup \{p(4), p(0, p(1))\}
\end{array}\right)\]

\[P_3 = \left(\begin{array}{l}
\{p(0) \leftarrow p(3)\}, \{p(1), p(2), p(3)\}, \{p(0)\}\right)\]

\[R_4(I(P_3) \cup O(P_3)) = \left(\begin{array}{l}
\{p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3)\}
\cup \{p(0, p(4)), p(1)\}
\end{array}\right)\]
prg.ground([("succ", [1]), ("succ", [2])])

Result *clingo* state

\[(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)\]

where

\[P_4 = \left( \left\{ p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \lnot p(2), \lnot p(3) \right\}, \left\{ p(4) \right\}, \left\{ p(0), p(1) \right\} \right) \]

\[P_3 = \left( \left\{ p(0) \leftarrow p(3); \ p(0) \leftarrow \lnot p(0) \right\}, \left\{ p(1), p(2), p(3) \right\}, \left\{ p(0) \right\} \right) \]

\[R_4(I(P_3) \cup O(P_3)) = \left( \left\{ p(1) \leftarrow p(4); \ p(1) \leftarrow \lnot p(2), \lnot p(3) \right\}, \left\{ p(0), p(4) \right\}, \left\{ p(1) \right\} \right) \]

\[R_4(I(P_3) \cup O(P_3)) = \left( \left\{ p(1) \leftarrow p(4); \ p(1) \leftarrow \lnot p(2), \lnot p(3) \right\}, \left\{ p(0), p(4) \right\}, \left\{ p(1) \right\} \right) \]
prg.ground([("succ", [1]), ("succ", [2])])

Result *clingo* state

\[(R_4, P_4, V_4) = (R_0, P_3 \sqcup R_4(I(P_3) \cup O(P_3)), V_3)\]

where

\[
P_4 = \left(\begin{array}{l}
\{ p(0) \leftarrow p(3); p(1) \leftarrow p(4); p(1) \leftarrow \neg p(2), \neg p(3); \} \\
p(0) \leftarrow \neg p(0); p(2) \leftarrow p(5); p(2) \leftarrow \neg p(3), \neg p(4) \}
\end{array}\right) \left(\begin{array}{l}
\{ p(3), p(5) \} \\
\{ p(0), p(1), p(2) \}
\end{array}\right)
\]

\[
P_3 = \left(\begin{array}{l}
\{ p(0) \leftarrow p(3); p(0) \leftarrow \neg p(0) \}
\end{array}\right) \left(\begin{array}{l}
\{ p(1), p(2), p(3) \}, \{ p(0) \}
\end{array}\right)
\]

\[
R_4(I(P_3) \cup O(P_3)) = \left(\begin{array}{l}
\{ p(1) \leftarrow p(4); p(1) \leftarrow \neg p(2), \neg p(3); \} \\
p(2) \leftarrow p(5); p(2) \leftarrow \neg p(3), \neg p(4) \}
\end{array}\right) \left(\begin{array}{l}
\{ p(0), p(4), p(1) \}, \{ p(3), p(5) \}, \{ p(2) \}
\end{array}\right)
\]
\[\text{prg.ground}(\text{[["succ", [1]], ["succ", [2]]]])\]

- Result \textit{clingo} state

\[(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)\]

where

\[P_4 = \left(\{p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3)\}\ \cup \ \{p(4)\}, \ \{p(0), p(1)\}\right)\]

\[P_3 = \left(\{p(0) \leftarrow p(3)\}, \ \{p(1), p(2), p(3)\}, \ \{p(0)\}\right)\]

\[R_4(I(P_3) \cup O(P_3)) = \left(\{p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3)\}, \ \{p(0), p(4)\}, \ \{p(1)\}\right)\]
prg.ground([["succ", [1]], ["succ", [2]]])

■ Result *clingo* state

\[(R_4, P_4, V_4) = (R_0, P_3 \sqcup R_4(I(P_3) \cup O(P_3)), V_3)\]

where

\[P_4 = \left( \left\{ p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3); \right\}, \left\{ p(4), \right\}, \left\{ p(0), p(1), \right\} \right) \]

\[P_3 = \left( \left\{ p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0) \right\}, \left\{ p(1), p(2), p(3) \right\}, \left\{ p(0) \right\} \right) \]

\[R_4(I(P_3) \cup O(P_3)) = \left( \left\{ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3); \right\}, \left\{ p(0), p(4), \right\}, \left\{ p(1), \right\} \right) \]
prg.ground([["succ", [1]], ["succ", [2]]])

Result *clingo* state

$$(R_4, P_4, V_4) = (R_0, P_3 \cup R_4(I(P_3) \cup O(P_3)), V_3)$$

where

$$P_4 = \left( \left\{ \begin{array}{l} p(0) \leftarrow p(3); \ p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3); \ p(0) \leftarrow \neg p(0); \ p(2) \leftarrow p(5); \ p(2) \leftarrow \neg p(3), \neg p(4) \end{array} \right\}, \left\{ \begin{array}{l} p(4), \ \{ p(0), p(1), \} \end{array} \right\}, \left\{ \begin{array}{l} p(3), p(5) \end{array} \right\}, \left\{ \begin{array}{l} p(2) \end{array} \right\} \right)$$

$$P_3 = \left( \left\{ \begin{array}{l} p(0) \leftarrow p(3); \ p(0) \leftarrow \neg p(0) \end{array} \right\}, \left\{ p(1), p(2), p(3) \right\}, \left\{ p(0) \right\} \right)$$

$$R_4(I(P_3) \cup O(P_3)) = \left( \left\{ \begin{array}{l} p(1) \leftarrow p(4); \ p(1) \leftarrow \neg p(2), \neg p(3); \ p(2) \leftarrow p(5); \ p(2) \leftarrow \neg p(3), \neg p(4) \end{array} \right\}, \left\{ \begin{array}{l} p(0), p(4), \ \{ p(0), p(1), \} \end{array} \right\}, \left\{ \begin{array}{l} p(3), p(5) \end{array} \right\}, \left\{ \begin{array}{l} p(2) \end{array} \right\} \right)$$
States and operations

Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
prg.solve()

- Global *clingo* state \((R_4, P_4, V_4)\)
- Input empty assignment

- Result *clingo* state

  \[(R_4, P_4, V_4) = (R_0, P_4, V_3)\]

- Print no stable model of \(P_4\) wrt \(V_4\)
prg.solve()

- Global *clingo* state \((R_4, P_4, V_4)\)
- Input empty assignment
- Result *clingo* state

\[(R_4, P_4, V_4) = (R_0, P_4, V_3)\]

- Print no stable model of \(P_4\) wrt \(V_4\)
prg.solve()

- Global *clingo* state \((R_4, P_4, V_4)\)
- Input empty assignment
- Result *clingo* state

\[(R_4, P_4, V_4) = (R_0, P_4, V_3)\]

- Print no stable model of \(P_4\) wrt \(V_4\)
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun

def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]],["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()

#end.
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
prg.ground([("succ", [3])])

- **Global** clingo state \((R_4, P_4, V_4)\), including atom base
  \(I(P_4) \cup O(P_4) = \{p(0), p(1), p(2), p(3), p(4), p(5)\}\)

- **Input** Extensible program \(R(\text{succ})[n/3]\)

- **Output** Module

\[ R_5(I(P_4) \cup O(P_4)) = \left( P_5, \left\{ p(0), p(1), p(2) \right\}, \left\{ p(3) \right\} \right) \]

where \(P_5 = \{p(3) \leftarrow p(6); \ p(3) \leftarrow \sim p(4), \sim p(5)\}\)

\(E_5 = \{p(6)\}\)

- **Result** clingo state

\((R_5, P_5, V_5) = (R_0, P_4 \sqcup R_5(I(P_4) \cup O(P_4)), V_3)\)
prg.ground([("succ", [3])])

- Global `clingo` state \((R_4, P_4, V_4)\), including atom base
  \(I(P_4) \cup O(P_4) = \{p(0), p(1), p(2), p(3), p(4), p(5)\}\)
- Input Extensible program \(R(\text{succ})[n/3]\)
- Output Module

\[
R_5(I(P_4) \cup O(P_4)) = \left( P_5, \left\{ p(0), p(1), p(2), p(4), p(5), p(6) \right\}, \{p(3)\} \right)
\]

where \(P_5 = \{p(3) \leftarrow p(6); \ p(3) \leftarrow \sim p(4), \sim p(5)\}\)
\(E_5 = \{p(6)\}\)

- Result `clingo` state

\((R_5, P_5, V_5) = (R_0, P_4 \sqcup R_5(I(P_4) \cup O(P_4)), V_3)\)
prg.ground([("succ", [3])])

- Global `clingo` state \((R_4, P_4, V_4)\), including atom base
  \(I(P_4) \cup O(P_4) = \{p(0), p(1), p(2), p(3), p(4), p(5)\}\)
- Input Extensible program \(R(\text{succ})[n/3]\)
- Output Module

\[
\mathbb{R}_5(I(P_4) \cup O(P_4)) = \left( P_5, \left\{ p(0), p(1), p(2), \right\}, \{p(3)\} \right)
\]
where \(P_5 = \{p(3) \leftarrow p(6); \ p(3) \leftarrow \neg p(4), \neg p(5)\}\)
\(E_5 = \{p(6)\}\)

- Result `clingo` state

\[
(R_5, P_5, V_5) = (R_0, P_4 \sqcup \mathbb{R}_5(I(P_4) \cup O(P_4)), V_3)
\]
prg.ground([("succ", [3])])

Result clingo state

\((R_5, P_5, V_5) = (R_0, P_4 \sqcup R_5(I(P_4) \cup O(P_4)), V_3)\)

where

\(R_5 = (R(\text{base}), R(\text{succ}))\)

\[P(P_5) = \begin{cases} p(0) \leftarrow p(3); & p(1) \leftarrow p(4); & p(1) \leftarrow \neg p(2), \neg p(3); \\ p(0) \leftarrow \neg p(0); & p(2) \leftarrow p(5); & p(2) \leftarrow \neg p(3), \neg p(4); \\ p(3) \leftarrow p(6); & p(3) \leftarrow \neg p(4), \neg p(5) \end{cases} \]

\(I(P_5) = \{p(4), p(5), p(6)\}\)

\(O(P_5) = \{p(0), p(1), p(2), p(3)\}\)

\(V_5 = (\emptyset, \emptyset)\)
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground([("succ", [1]),("succ", [2])])
    prg.solve()
    prg.ground([("succ", [3])])
    prg.solve()
#end.
Example

#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground([("base", [])])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground([("succ", [1]),("succ", [2])])
    prg.solve()
    prg.ground([("succ", [3])])
    prg.solve()
#end.
prg.solve()

- Global `clingo` state \((R_5, P_5, V_5)\)
- Input empty assignment
- Result `clingo` state
  \[(R_5, P_5, V_5) = (R_0, P_5, V_3)\]
- Print stable model \(\{p(0), p(3)\}\) of \(P_5\) wrt \(V_5\)
prg.solve()

- Global *clingo* state \((R_5, P_5, V_5)\)
- Input empty assignment
- Result *clingo* state

\[(R_5, P_5, V_5) = (R_0, P_5, V_3)\]

- Print stable model \(\{p(0), p(3)\}\) of \(P_5\) wrt \(V_5\)
prg.solve()

- Global `clingo` state \((R_5, P_5, V_5)\)
- Input empty assignment
- Result `clingo` state

\[(R_5, P_5, V_5) = (R_0, P_5, V_3)\]

- Print stable model \(\{p(0), p(3)\}\) of \(P_5\) wrt \(V_5\)
#external p(1;2;3).
p(0) :- p(3).
p(0) :- not p(0).

#program succ(n).
#external p(n+3).
p(n) :- p(n+3).
p(n) :- not p(n+1), not p(n+2).

#script(python)
from clingo import Fun
def main(prg):
    prg.ground(["base", []])
    prg.assign_external(Fun("p", [3]), True)
    prg.solve()
    prg.assign_external(Fun("p", [3]), False)
    prg.solve()
    prg.ground(["succ", [1]], ["succ", [2]])
    prg.solve()
    prg.ground(["succ", [3]])
    prg.solve()
#end.
Clingo on the run

$ clingo simple.lp
clingo version 4.5.0
Reading from simple.lp
Solving...
Answer: 1
p(3) p(0)
Solving...
Solving...
Solving...
Solving...
Answer: 1
p(3) p(0)
SATISFIABLE

Models : 2+
Calls  : 4
Time   : 0.019s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.010s
$ clingo simple.lp
clingo version 4.5.0
Reading from simple.lp
Solving...
Answer: 1
p(3) p(0)
Solving...
Solving...
Answer: 1
p(3) p(0)
SATISFIABLE

Models : 2+
Calls : 4
Time : 0.019s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.010s
Outline

46 Motivation

47 #program and #external declaration

48 Module composition

49 States and operations

50 Incremental reasoning

51 Boardgaming
Incremental reasoning

Towers of Hanoi Instance

peg(a;b;c). disk(1..7).

init_on(1,a). init_on((2;7),b). init_on((3;4;5;6),c).
goal_on((3;4),a). goal_on((1;2;5;6;7),c).

Torsten Schaub (KRR@UP)
Answer Set Solving in Practice
February 18, 2019 424 / 653
Incremental reasoning

Towers of Hanoi Instance

peg(a; b; c). disk(1..7).

init_on(1, a). init_on((2; 7), b). init_on((3; 4; 5; 6), c).
goal_on((3; 4), a). goal_on((1; 2; 5; 6; 7), c).


#program base.

on(D,P,0) :- init_on(D,P).
Incremental reasoning

Towers of Hanoi Encoding

#program step(t).

1 { move(D,P,t) : disk(D), peg(P) } 1.

moved(D,t) :- move(D,_,t).
blocked(D,P,t) :- on(D+1,P,t-1), disk(D+1).
blocked(D,P,t) :- blocked(D+1,P,t), disk(D+1).
    :- move(D,P,t), blocked(D-1,P,t).
    :- moved(D,t), on(D,P,t-1), blocked(D,P,t).

on(D,P,t) :- on(D,P,t-1), not moved(D,t).
on(D,P,t) :- move(D,P,t).
    :- not 1 { on(D,P,t) : peg(P) } 1, disk(D).
Towers of Hanoi Encoding

#program  check(t).
#external  query(t).

:- goal_on(D,P), not on(D,P,t), query(t).
Incremental Solving (ASP)

#script (python)

```python
from clingo import SolveResult, Fun

def main(prg):
    ret, parts, step = SolveResult.UNSAT, [], 1
    parts.append(('base', []))
    while ret == SolveResult.UNSAT:
        parts.append(('step', [step]))
        parts.append(('check', [step]))
        prg.ground(parts)
        prg.release_external(Fun('query', [step-1]))
        prg.assign_external(Fun('query', [step]), True)
        ret, parts, step = prg.solve(), [], step+1

#end.
```
#script (python)

```python
from clingo import SolveResult, Fun

def main(prg):
    ret, parts, step = SolveResult.UNSAT, [], 1
    parts.append(('base', []))
    while ret == SolveResult.UNSAT:
        parts.append(('step', [step]))
        parts.append(('check', [step]))
        prg.ground(parts)
        prg.release_external(Fun('query', [step-1]))
        prg.assign_external(Fun('query', [step]), True)
    ret, parts, step = prg.solve(), [], step+1

#end.
```
Incremental Solving

$ clingo toh.lp tohCtrl.lp
clingo version 4.5.0
Reading from toh.lp ...
Solving...
Solving...
...
Solving...
Answer: 1
move(7,a,1) move(6,b,2) move(7,b,3) move(5,a,4) move(7,c,5) move(6,a,6) /
move(7,a,7) move(4,b,8) move(7,b,9) move(6,c,10) move(7,c,11) move(5,b,12) /
move(1,c,13) move(7,a,14) move(6,b,15) move(7,b,16) move(3,a,17) move(7,c,18) /
move(6,a,19) move(7,a,20) move(5,c,21) move(7,b,22) move(6,c,23) move(7,c,24) /
move(4,a,25) move(7,a,26) move(6,b,27) move(7,b,28) move(5,a,29) move(7,c,30) /
move(6,a,31) move(7,a,32) move(2,c,33) move(7,c,34) move(6,b,35) move(7,b,36) /
move(5,c,37) move(7,a,38) move(6,c,39) move(7,c,40) SATISFIABLE

Models : 1+
Calls : 40
Time : 0.312s (Solving: 0.22s 1st Model: 0.01s Unsat: 0.21s)
CPU Time : 0.300s
Incremental Solving

$ clingo toh.lp tohCtrl.lp
clingo version 4.5.0
Reading from toh.lp ...
Solving...
Solving...
[...]
Solving...
Answer: 1
move(7,a,1) move(6,b,2) move(7,b,3) move(5,a,4) move(7,c,5) move(6,a,6) \ 
move(7,a,7) move(4,b,8) move(7,b,9) move(6,c,10) move(7,c,11) move(5,b,12) \ 
move(1,c,13) move(7,a,14) move(6,b,15) move(7,b,16) move(3,a,17) move(7,c,18) \ 
move(6,a,19) move(7,a,20) move(5,c,21) move(7,b,22) move(6,c,23) move(7,c,24) \ 
move(4,a,25) move(7,a,26) move(6,b,27) move(7,b,28) move(5,a,29) move(7,c,30) \ 
move(6,a,31) move(7,a,32) move(2,c,33) move(7,c,34) move(6,b,35) move(7,b,36) \ 
move(5,c,37) move(7,a,38) move(6,c,39) move(7,c,40)
SATISFIABLE

Models : 1+
Calls : 40
Time : 0.312s (Solving: 0.22s 1st Model: 0.01s Unsat: 0.21s)
CPU Time : 0.300s
from sys import stdout
from clingo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(('base', []))

while ret == SolveResult.UNSAT:
    parts.append(('step', [step]))
    parts.append(('check', [step]))
    prg.ground(parts)
    prg.release_external(Fun('query', [step-1]))
    prg.assign_external(Fun('query', [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```python
from sys import stdout
from clingo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(('base', []))

while ret == SolveResult.UNSAT:
    parts.append(('step', [step]))
    parts.append(('check', [step]))
    prg.ground(parts)
    prg.release_external(Fun('query', [step-1]))
    prg.assign_external(Fun('query', [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```
Incremental Solving (tohCtrl.py)

```python
from sys import stdout
from clingo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(("base", []))
while ret == SolveResult.UNSAT:
    parts.append(("step", [step]))
    parts.append(("check", [step]))
    prg.ground(parts)
    prg.release_external(Fun("query", [step-1]))
    prg.assign_external(Fun("query", [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```
Incremental Solving (tohCtrl.py)

```python
from sys import stdout
from clingo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(('base', []))

while ret == SolveResult.UNSAT:
    parts.append(('step', [step]))
    parts.append(('check', [step]))
    prg.ground(parts)
    prg.release_external(Fun('query', [step-1]))
    prg.assign_external(Fun('query', [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```
Incremental Solving (tohCtrl.py)

```python
from sys import stdout
from clingo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(("base", []))

while ret == SolveResult.UNSAT:
    parts.append(("step", [step]))
    parts.append(("check", [step]))
    prg.ground(parts)
    prg.release_external(Fun("query", [step-1]))
    prg.assign_external(Fun("query", [step]), True)

f = lambda m: stdout.write(str(m))
ret, parts, step = prg.solve(on_model=f), [], step+1
```

Torsten Schaub (KRR@UP)
Incremental Solving (Python)

$ python tohCtrl.py

move(7,c,40) move(7,a,20) move(7,c,18) move(6,a,31) move(6,b,15) move(7,b,36) \
move(7,c,24) move(7,c,11) move(3,a,17) move(6,a,19) move(7,b,3) move(7,c,5) \
move(7,a,1) move(6,b,35) move(6,c,10) move(6,a,6) move(6,b,2) move(7,b,9) \
move(7,a,7) move(4,b,8) move(7,a,38) move(7,b,16) move(5,a,29) move(7,b,22) \
move(6,c,39) move(6,c,23) move(5,b,12) move(4,a,25) move(1,c,13) move(5,a,4) \
move(7,a,14) move(7,a,26) move(6,b,27) move(7,a,32) move(7,b,28) move(7,c,30) \
move(2,c,33) move(5,c,21) move(7,c,34) move(5,c,37)
Incremental Solving (Python)

```plaintext
$ python tohCtrl.py
move(7,c,40) move(7,a,20) move(7,c,18) move(6,a,31) move(6,b,15) move(7,b,36) \
move(7,c,24) move(7,c,11) move(3,a,17) move(6,a,19) move(7,b,3) move(7,c,5) \
move(7,a,1) move(6,b,35) move(6,c,10) move(6,a,6) move(6,b,2) move(7,b,9) \
move(7,a,7) move(4,b,8) move(7,a,38) move(7,b,16) move(5,a,29) move(7,b,22) \
move(6,c,39) move(6,c,23) move(5,b,12) move(4,a,25) move(1,c,13) move(5,a,4) \
move(7,a,14) move(7,a,26) move(6,b,27) move(7,a,32) move(7,b,28) move(7,c,30) \
move(2,c,33) move(5,c,21) move(7,c,34) move(5,c,37)
```
Outline

46 Motivation
47 #program and #external declaration
48 Module composition
49 States and operations
50 Incremental reasoning
51 Boardgaming
Alex Rudolph’s Ricochet Robots
Solving goal(13) from cornered robots

- Four robots roaming
  - horizontally
  - vertically
up to blocking objects, ricocheting (optionally)

- Goal Robot on target (sharing same color)
Alex Rudolph’s Ricochet Robots
Solving goal(13) from cornered robots

- Four robots roaming
  - horizontally
  - vertically
  up to blocking objects, ricocheting (optionally)

- Goal Robot on target (sharing same color)
Alex Rudolph’s Ricochet Robots
Solving goal(13) from cornered robots

- Four robots roaming
  - horizontally
  - vertically
  up to blocking objects, ricocheting (optionally)

- Goal Robot on target (sharing same color)
Alex Rudolph’s Ricochet Robots
Solving goal(13) from cornered robots

- Four robots roaming
  - horizontally
  - vertically
  up to blocking objects, ricocheting (optionally)

- Goal Robot on target (sharing same color)
Solving $\text{goal}(13)$ from cornered robots (ctd)
Solving \texttt{goal(13)} from cornered robots (ctd)
Solving \texttt{goal(13)} from cornered robots (ctd)
Solving $\text{goal}(13)$ from cornered robots (ctd)
Solving $\text{goal}(13)$ from cornered robots (ctd)
Solving \texttt{goal(13)} from cornered robots (ctd)
Solving $\text{goal}(13)$ from cornered robots (ctd)
Solving $\text{goal}(13)$ from cornered robots (ctd)
```
dim(1..16).

barrier( 2, 1, 1, 0). barrier(13,11, 1, 0). barrier( 9, 7, 0, 1).
barrier(10, 1, 1, 0). barrier(11,12, 1, 0). barrier(11, 7, 0, 1).
barrier( 4, 2, 1, 0). barrier(14,13, 1, 0). barrier(14, 7, 0, 1).
barrier(14, 2, 1, 0). barrier( 6,14, 1, 0). barrier(16, 9, 0, 1).
barrier( 2, 3, 1, 0). barrier( 3,15, 1, 0). barrier( 2,10, 0, 1).
barrier(11, 3, 1, 0). barrier(10,15, 1, 0). barrier( 5,10, 0, 1).
barrier( 7, 4, 1, 0). barrier( 4,16, 1, 0). barrier( 8,10, 0,-1).
barrier( 3, 7, 1, 0). barrier(12,16, 1, 0). barrier( 9,10, 0,-1).
barrier(14, 7, 1, 0). barrier( 5, 1, 0, 1). barrier( 9,10, 0, 1).
barrier( 7, 8, 1, 0). barrier(15, 1, 0, 1). barrier(14,10, 0, 1).
barrier(10, 8,-1, 0). barrier( 2, 2, 0, 1). barrier( 1,12, 0, 1).
barrier(11, 8, 1, 0). barrier(12, 3, 0, 1). barrier(11,12, 0, 1).
barrier( 7, 9, 1, 0). barrier( 7, 4, 0, 1). barrier( 7,13, 0, 1).
barrier(10, 9,-1, 0). barrier(16, 4, 0, 1). barrier(15,13, 0, 1).
barrier( 4,10, 1, 0). barrier( 1, 6, 0, 1). barrier(10,14, 0, 1).
barrier( 2,11, 1, 0). barrier( 4, 7, 0, 1). barrier( 3,15, 0, 1).
barrier( 8,11, 1, 0). barrier( 8, 7, 0, 1).
```
targets.lp

#external goal(1..16).

target(red, 5, 2) :- goal(1).
target(red, 15, 2) :- goal(2).
target(green, 2, 3) :- goal(3).
target(blue, 12, 3) :- goal(4).
target(yellow, 7, 4) :- goal(5).
target(blue, 4, 7) :- goal(6).
target(green, 14, 7) :- goal(7).
target(yellow, 11, 8) :- goal(8).
target(yellow, 5, 10) :- goal(9).
target(green, 2, 11) :- goal(10).
target(red, 14, 11) :- goal(11).
target(green, 11, 12) :- goal(12).
target(yellow, 15, 13) :- goal(13).
target(blue, 7, 14) :- goal(14).
target(red, 3, 15) :- goal(15).
target(blue, 10, 15) :- goal(16).

robot(red;green;blue;yellow).

#external pos((red;green;blue;yellow),1..16,1..16).
Boardgaming

ricochet.lp

time(1..horizon).
dir(-1,0;1,0;0,-1;0,1).

stop( DX, DY, X, Y ) :- barrier(X,Y,DX,DY).
stop(-DX,-DY,X+DX,Y+DY) :- stop(DX,DY,X,Y).
pos(R,X,Y,0) :- pos(R,X,Y).

1 { move(R,DX,DY,T) : robot(R), dir(DX,DY) } 1 :- time(T).
move(R,T) :- move(R,_,_,T).
halt(DX,DY,X-DX,Y-DY,T) :- pos(_,X,Y,T), dir(DX,DY), dim(X-DX), dim(Y-DY),
not stop(-DX,-DY,X,Y), T < horizon.
goto(R,DX,DY,X,Y,T) :- pos(R,X,Y,T), dir(DX,DY), T < horizon.
goto(R,DX,DY,X+DX,Y+DY,T) :- goto(R,DX,DY,X,Y,T), dim(X+DX), dim(Y+DY),
not stop(DX,DY,X,Y), not halt(DX,DY,X,Y,T).
pos(R,X,Y,T) :- move(R,DX,DY,T), goto(R,DX,DY,X,Y,T-1),
not goto(R,DX,DY,X+DX,Y+DY,T-1).
pos(R,X,Y,T) :- pos(R,X,Y,T-1), time(T), not move(R,T).

:- target(R,X,Y), not pos(R,X,Y,horizon).

#show move/4.
Solving \texttt{goal(13)} from cornered robots

\begin{verbatim}
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \ 
   <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). \ goal(13).")

   clingo version 4.5.0
   Reading from board.lp ...
   Solving...
   Answer: 1
   move(red,0,1,1) move(red,1,0,2) move(red,0,1,3) move(red,-1,0,4) move(red,0,1,5) \ 
   move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
   SATISFIABLE

   Models : 1+
   Calls   : 1
   Time    : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
   CPU Time: 1.880s

$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \ 
   <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). \ goal(13).")

   clingo version 4.5.0
   Reading from board.lp ...
   Solving...
   UNSATISFIABLE

   Models : 0
   Calls   : 1
   Time    : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
   CPU Time: 2.800s
\end{verbatim}
Solving \texttt{goal(13)} from cornered robots

\begin{verbatim}
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \\
    <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
Answer: 1
move(red,0,1,1) move(red,1,0,2) move(red,0,1,3) move(red,-1,0,4) move(red,0,1,5) \\
move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
SATISFIABLE

Models : 1+
Calls : 1
Time : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
CPU Time : 1.880s
\end{verbatim}

\begin{verbatim}
$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \\
    <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
UNSATISFIABLE

Models : 0
Calls : 1
Time : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
CPU Time : 2.800s
\end{verbatim}
Solving \texttt{goal(13)} from cornered robots

$$\texttt{cling} \texttt{o \ b} \texttt{oard.lp \ \texttt{t}a\texttt{r}g\texttt{e}t\texttt{s}\.lp \ \texttt{r}i\texttt{c}o\texttt{h}e\texttt{t}.lp \ -c \ \texttt{h}o\texttt{r}i\texttt{z}o\texttt{n}=9 \ \backslash}$$
$$\texttt{ \langle \texttt{e}cho \ "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). \ \texttt{g}o\texttt{a}l(13).\" \rangle}$$

\texttt{cling}o \ \texttt{v}ersion \ 4.5.0
\texttt{R}ead\texttt{ing} \ \texttt{f}rom \ \texttt{bo}ard\.\texttt{l}p \ \ldots
\texttt{S}olving\ldots
\texttt{A}n\texttt{sw}er: \ 1
\texttt{move(red,0,1,1) \ move(red,1,0,2) \ move(red,0,1,3) \ move(red,-1,0,4) \ move(red,0,1,5) \ move(yellow,0,-1,6) \ move(red,1,0,7) \ move(yellow,0,1,8) \ move(yellow,-1,0,9) \ SATISFIABLE}

\texttt{Models} \ : \ 1+
\texttt{Calls} \ : \ 1
\texttt{Time} \ : \ 1.895s \ (\texttt{Solving}: \ 1.45s \ \texttt{1st \ Model}: \ 1.45s \ \texttt{Unsat}: \ 0.00s)
\texttt{CPU \ Time} \ : \ 1.880s

$$\texttt{cling} \texttt{o \ b} \texttt{oard.lp \ \texttt{t}a\texttt{r}g\texttt{e}t\texttt{s}\.lp \ \texttt{r}i\texttt{c}o\texttt{h}e\texttt{t}.lp \ -c \ \texttt{h}o\texttt{r}i\texttt{z}o\texttt{n}=8 \ \backslash}$$
$$\texttt{ \langle \texttt{e}cho \ "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). \ \texttt{g}o\texttt{a}l(13).\" \rangle}$$

\texttt{cling}o \ \texttt{v}ersion \ 4.5.0
\texttt{R}ead\texttt{ing} \ \texttt{f}rom \ \texttt{bo}ard\.\texttt{l}p \ \ldots
\texttt{S}olving\ldots
\texttt{UNSATISFIABLE}

\texttt{Models} \ : \ 0
\texttt{Calls} \ : \ 1
\texttt{Time} \ : \ 2.817s \ (\texttt{Solving}: \ 2.41s \ \texttt{1st \ Model}: \ 0.00s \ \texttt{Unsat}: \ 2.41s)
\texttt{CPU \ Time} \ : \ 2.800s
Solving \texttt{goal(13)} from cornered robots

\begin{verbatim}
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \ 
  <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). goal(13).")

clingo version 4.5.0
Reading from board.lp ... Solving...
Answer: 1
move(red,0,1,1) move(red,1,0,2) move(red,0,1,3) move(red,-1,0,4) move(red,0,1,5) \ 
move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
SATISFIABLE

Models : 1+
Calls : 1
Time : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
CPU Time : 1.880s
\end{verbatim}

\begin{verbatim}
$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \ 
  <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). goal(13).")

clingo version 4.5.0
Reading from board.lp ... Solving...
UNSATISFIABLE

Models : 0
Calls : 1
Time : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
CPU Time : 2.800s
\end{verbatim}
goon(T) :- target(R,X,Y), T = 0..horizon, not pos(R,X,Y,T).

:- move(R,DX,DY,T-1), time(T), not goon(T-1), not move(R,DX,DY,T).

#minimize{ 1,T : goon(T) }. 
Solving \textit{goal(13)} from cornered robots

\begin{verbatim}
$ clingo board.lp targets.lp ricochet.lp optimization.lp -c horizon=20 --quiet=1,0 \\
  <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
Optimization: 20
Optimization: 19
Optimization: 18
Optimization: 17
Optimization: 16
Optimization: 15
Optimization: 14
Optimization: 13
Optimization: 12
Optimization: 11
Optimization: 10
Optimization: 9
Answer: 12
move(blue,0,-1,1) move(blue,1,0,2) move(yellow,0,-1,3) move(blue,0,1,4) move(yellow,-1,0,5) \\
move(blue,1,0,6) move(blue,0,-1,7) move(yellow,1,0,8) move(yellow,0,1,9) move(yellow,0,1,10) \\
move(yellow,0,1,11) move(yellow,0,1,12) move(yellow,0,1,13) move(yellow,0,1,14) move(yellow,0,1,15) \\
move(yellow,0,1,16) move(yellow,0,1,17) move(yellow,0,1,18) move(yellow,0,1,19) move(yellow,0,1,20)
OPTIMUM FOUND

Models : 12
Optimum : yes
Optimization : 9
Calls : 1
Time : 16.145s (Solving: 15.01s 1st Model: 3.35s Unsat: 2.02s)
CPU Time : 16.080s
\end{verbatim}
Solving \texttt{goal(13)} from cornered robots

```
$ clingo board.lp targets.lp ricochet.lp optimization.lp -c horizon=20 --quiet=1,0 \
   <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16). goal(13).")
```

clingo version 4.5.0

Reading from board.lp ...

Solving...

Optimization: 20
Optimization: 19
Optimization: 18
Optimization: 17
Optimization: 16
Optimization: 15
Optimization: 14
Optimization: 13
Optimization: 12
Optimization: 11
Optimization: 10
Optimization: 9
Answer: 12

```
move(blue,0,-1,1) move(blue,1,0,2) move(yellow,0,-1,3) move(blue,0,1,4) move(yellow,-1,0,5) \
move(blue,1,0,6) move(blue,0,-1,7) move(yellow,1,0,8) move(yellow,0,1,9) move(yellow,0,1,10) \
move(yellow,0,1,11) move(yellow,0,1,12) move(yellow,0,1,13) move(yellow,0,1,14) move(yellow,0,1,15) \
move(yellow,0,1,16) move(yellow,0,1,17) move(yellow,0,1,18) move(yellow,0,1,19) move(yellow,0,1,20)
```

OPTIMUM FOUND

Models : 12
Optimum : yes
Optimization : 9
Calls : 1
Time : 16.145s (Solving: 15.01s 1st Model: 3.35s Unsat: 2.02s)
CPU Time : 16.080s
Boardgaming

Playing in rounds

Round 1: goal(13)

Round 2: goal(4)
Control loop

1. Create an operational *clingo* object

2. Load and ground the logic programs encoding Ricochet Robot (relative to some fixed horizon) within the control object

3. While there is a goal, do the following
   1. Enforce the initial robot positions
   2. Enforce the current goal
   3. Solve the logic program contained in the control object
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground(["base", []])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
        self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]), Fun("pos", [Fun("blue"), 1, 16]),
            Fun("pos", [Fun("green"), 16, 1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
Variables of interest

- `last_positions` holds the starting positions of the robots for each turn.
- `last_solution` holds the last solution of a search call.
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step.
- `horizon` holds the maximum number of moves to find a solution.
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving.
Variables of interest

- \texttt{last\_positions} holds the starting positions of the robots for each turn

- \texttt{last\_solution} holds the last solution of a search call
  (Note that callbacks cannot return values directly)

- \texttt{undo\_external} holds a list containing the current goal and starting positions to be cleared upon the next step

- \texttt{horizon} holds the maximum number of moves to find a solution

- \texttt{ctl} holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving
Variables of interest

- `last_positions` holds the starting positions of the robots for each turn

- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)

- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step

- `horizon` holds the maximum number of moves to find a solution

- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving
**Variables of interest**

- `last_positions` holds the starting positions of the robots for each turn.
- `last_solution` holds the last solution of a search call. (Note that callbacks cannot return values directly.)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step.
- `horizon` holds the maximum number of moves to find a solution.
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving.
Variables of interest

- `last_positions` holds the starting positions of the robots for each turn.
- `last_solution` holds the last solution of a search call. (Note that callbacks cannot return values directly.)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step.
- `horizon` holds the maximum number of moves to find a solution.
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving.
Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
            self.ctl.ground(["base", []])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]), Fun("pos", [Fun("blue"), 1, 16]),
            Fun("pos", [Fun("green"), 16, 1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
Setup and control loop

```
horizon    = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]),
             Fun("pos", [Fun("blue"), 1, 16]),
             Fun("pos", [Fun("green"), 16, 1]),
             Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]),
             Fun("goal", [4]),
             Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
  print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a clingo object)
3. Playing in rounds
Boardgaming

Setup and control loop

```python
>> horizon = 15
>> encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
>> positions = [Fun("pos", [Fun("red"), 1, 1]),
               Fun("pos", [Fun("blue"), 1, 16]),
               Fun("pos", [Fun("green"), 16, 1]),
               Fun("pos", [Fun("yellow"), 16, 16])]
>> sequence = [Fun("goal", [13]),
              Fun("goal", [4]),
              Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a `clingo` object)
3. Playing in rounds
Setup and control loop

```python
horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]),
            Fun("pos", [Fun("blue"), 1, 16]),
            Fun("pos", [Fun("green"), 16, 1]),
            Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]),
            Fun("goal", [4]),
            Fun("goal", [7])]

>> player = Player(horizon, positions, encodings)
   for goal in sequence:
       print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a clingo object)
3. Playing in rounds
Setup and control loop

```python
horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]),
            Fun("pos", [Fun("blue"), 1, 16]),
            Fun("pos", [Fun("green"), 16, 1]),
            Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]),
            Fun("goal", [4]),
            Fun("goal", [7])]

player = Player(horizon, positions, encodings)
>> for goal in sequence:
    print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a clingo object)
3. Playing in rounds
Setup and control loop

```
horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]),
             Fun("pos", [Fun("blue"), 1, 16]),
             Fun("pos", [Fun("green"), 16, 1]),
             Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]),
            Fun("goal", [4]),
            Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a clingo object)
3. Playing in rounds
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
        self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]), Fun("pos", [Fun("blue"), 1, 16]),
            Fun("pos", [Fun("green"), 16, 1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]"
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(
        ['-c', 'horizon={0}'.format(self.horizon)])
    for x in files:
        self.ctl.load(x)
    self.ctl.ground([('base', [])])

1. Initializing variables
2. Creating clingo object
3. Loading encoding and instance
4. Grounding encoding and instance
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
    for x in files:
        self.ctl.load(x)
    self.ctl.ground(["base", []])

1. Initializing variables
2. Creating clingo object
3. Loading encoding and instance
4. Grounding encoding and instance
```python
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(["-c", 'horizon={0}'.format(self.horizon)])

    for x in files:
        self.ctl.load(x)
        self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating `clingo` object
3. Loading encoding and instance
4. Grounding encoding and instance
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])

    for x in files:
        self.ctl.load(x)
        self.ctl.ground([("base", [])])
```python
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
    for x in files:
        self.ctl.load(x)
    self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating `clingo` object
3. Loading encoding and instance
4. Grounding encoding and instance
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
    for x in files:
        self.ctl.load(x)
    self.ctl.ground([("base", [])])

1. Initializing variables
2. Creating \textit{clingo} object
3. Loading encoding and instance
4. Grounding encoding and instance
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground(["base", []])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]), Fun("pos", [Fun("blue"), 1, 16]),
             Fun("pos", [Fun("green"), 16, 1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model by passing user-defined on_model method
def solve(self, goal):
    >> for x in self.undo_external:
    >>     self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution

1. Unsetting previous external atoms  (viz. previous goal and positions)
2. Setting next external atoms        (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined on_model method
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
    self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model by passing user-defined on_model method
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
>> self.last_solution = None
>> self.ctl.solve(on_model=self.on_model)
>> return self.last_solution

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined on_model method
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model by passing user-defined `on_model` method
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model by passing user-defined on_model method
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground(["base", []])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
        self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]), Fun("pos", [Fun("blue"), 1, 16]), Fun("pos", [Fun("green"), 16, 1]), Fun("pos", [Fun("yellow"), 16, 16])]  
sequence = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]  
player = Player(horizon, positions, encodings)

for goal in sequence:
    print player.solve(goal)
def on_model(self, model):
    self.last_solution = model.atoms()
    self.last_positions = []
    for atom in model.atoms(Model.ATOMS):
        if (atom.name() == "pos" and
            len(atom.args()) == 4 and
            atom.args()[3] == self.horizon):
            self.last_positions.append(Fun("pos", atom.args()[:-1]))

1 Storing stable model
2 Extracting atoms (viz. last robot positions) by adding \texttt{pos(R,X,Y)} for each \texttt{pos(R,X,Y,horizon)}
def on_model(self, model):
    >>> self.last_solution = model.atoms()
    self.last_positions = []
    for atom in model.atoms(Model.ATOMS):
        if (atom.name() == "pos" and
            len(atom.args()) == 4 and
            atom.args()[3] == self.horizon):
            self.last_positions.append(Fun("pos", atom.args()[:-1]))

1. Storing stable model
2. Extracting atoms (viz. last robot positions) by adding \texttt{pos(R,X,Y)} for each \texttt{pos(R,X,Y,\text{horizon})}
```python
def on_model(self, model):
    self.last_solution = model.atoms()
    self.last_positions = []
    for atom in model.atoms(Model.ATOMS):
        if (atom.name() == "pos" and
            len(atom.args()) == 4 and
            atom.args()[3] == self.horizon):
            self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions) by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`
```python
def on_model(self, model):
    self.last_solution = model.atoms()
    self.last_positions = []
    for atom in model.atoms(Model.ATOMS):
        if (atom.name() == "pos" and
            len(atom.args()) == 4 and
            atom.args()[3] == self.horizon):
            self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. **Storing stable model**
2. **Extracting atoms** (viz. last robot positions) by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`
def on_model(self, model):
    self.last_solution = model.atoms()
    self.last_positions = []
    for atom in model.atoms(Model.ATOMS):
        if (atom.name() == "pos" and
            len(atom.args()) == 4 and
            atom.args()[3] == self.horizon):
            self.last_positions.append(Fun("pos", atom.args()[:-1]))

1. Storing stable model
2. Extracting atoms (viz. last robot positions) by adding $pos(R,X,Y)$ for each $pos(R,X,Y,\text{horizon})$
def on_model(self, model):
    self.last_solution = model.atoms()
    self.last_positions = []
    for atom in model.atoms(Model.ATOMS):
        if (atom.name() == "pos" and
            len(atom.args()) == 4 and
            atom.args()[3] == self.horizon):
            self.last_positions.append(Fun("pos", atom.args()[::-1]))

1. Storing stable model
2. Extracting atoms (viz. last robot positions) by adding pos(R,X,Y) for each pos(R,X,Y,horizon)
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
            self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"), 1, 1]), Fun("pos", [Fun("blue"), 1, 16]),
             Fun("pos", [Fun("green"), 16, 1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
Let’s play!

$ python ricochet.py

[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]

[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]

[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ python robotviz
Let’s play!

$ python ricochet.py
[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
 move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
 move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
 move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]
[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
 move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
 move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
 move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]
[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
 move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
 move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
 move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ python robotviz
Let’s play!

$ python ricochet.py

[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
 move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
 move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
 move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]

[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
 move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
 move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
 move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]

[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
 move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
 move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
 move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ python robotviz
ASP modulo theories: Overview

- Theory language
- Low-level semantics
- Intermediate Format
- Theory propagation
- Experiments
- Acyclicity checking
- Constraint Answer Set Programming
Motivation

- **Input** \( \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{SAT} \)
- **Output** \( \text{ASPmT} = \text{DB} + \text{KRR} + \text{LP} + \text{S} \)

- **ASP solving** ground | solve
  - logic programs with elusive theory atoms

- **Application areas**
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- **Input**  \( \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{SAT} \)
- **Output** \( \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{S} \)
- **ASP solving**  \( \text{ground} \ | \ \text{solve} \)
  - logic programs with elusive theory atoms
- **Application areas**
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- Input: $\text{ASP} = \text{DB+KRR+LP+SAT}$
- Output: $\text{ASPM}T = \text{DB+KRR+LP+SMT}$

- ASP solving $\text{ground} \mid \text{solve}$
  - logic programs with elusive theory atoms

- Application areas
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- **Input**: \( \text{ASP} = \text{DB+KRR+LP+SAT} \)
- **Output**: \( \text{ASPmT} = \text{DB+KRR+LP+SMT} \) — **NO!**

- **ASP solving** *ground* | *solve*
  - ➡️ *logic programs with elusive theory atoms*

- **Application areas**
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- Input  \( \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{SAT} \)
- Output \( \text{ASP}_{mT} = (\text{DB} + \text{KRR} + \text{LP} + \text{SAT})_{mT} \)

- ASP solving \( \text{ground} \mid \text{solve} \)
  - logic programs with elusive theory atoms

- Application areas
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- **Input**  \( \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{SAT} \)
- **Output**  \( \text{ASP}_{mT} = (\text{DB} + \text{KRR} + \text{LP} + \text{SAT})_{mT} \)
- **ASP solving**  \( ground \mid solve \)
  - logic programs with elusive theory atoms

- **Application areas**
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- **Input** \( \text{ASP} = \text{DB+KRR+LP+SAT} \)
- **Output** \( \text{ASP}^mT = (\text{DB+KRR+LP+SAT})^mT \)

- **ASP solving modulo theories** *ground \% theories | solve \% theories*
  - logic programs with elusive theory atoms

- **Application areas**
  - Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc
Motivation

- **Input** \( \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{SAT} \)
- **Output** \( \text{ASP}_{mT} = (\text{DB} + \text{KRR} + \text{LP} + \text{SAT})_{mT} \)

- ASP solving modulo theories *ground % theories | solve % theories*
  - logic programs with elusive theory atoms

- Application areas
  - Agents, Assisted Living, Robotics, Planning, Scheduling,
  - Bio- and Cheminformatics, etc
Motivation

Input: \[ \text{ASP} = \text{DB} + \text{KRR} + \text{LP} + \text{SAT} \]

Output: \[ \text{ASP}_{mT} = (\text{DB} + \text{KRR} + \text{LP} + \text{SAT})_{mT} \]

ASP solving modulo theories: \( \text{ground} \ % \ theories \ | \ \text{solve} \ % \ theories \)

⇒ logic programs with elusive theory atoms

Application areas:
- Agents, Assisted Living, Robotics, Planning, Scheduling,
- Bio- and Cheminformatics, etc
ASP solving process

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models
- Solution
- Interpreting

Modeling → Solving
ASP solving process *modulo theories*

- Problem
- Logic Program
- Grounding
- Solver
- Stable Models
- Solution
- Interpreting

Modeling → Logic Program → Grounder → Solver → Stable Models → Interpreting

Solving
ASP solving process modulo theories

Problem

Modeling

Logic Program

Grounder

Solver

Solving

Stable Models

Solution

Interpreting

Torsten Schaub (KRR@UP)

Answer Set Solving in Practice

February 18, 2019
clingo’s approach

Theory T
Grammar

T-ASP Program

gringo
T

clasp
T

T-ASP Solution
Outline

52 Theory language
53 Low-level semantics
54 Intermediate Format
55 Theory propagation
56 Experiments
57 Acyclicity checking
58 Constraint Answer Set Programming
ASP solving process *modulo theories*

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models
- Solution
- Interpreting

Modeling

Solving
ASP solving process \textit{modulo theories}

- Problem
  - Modeling
  - Logic Program
  - Grounding
  - Solver
  - Stable Models
- Solution
  - Interpreting

Theory language
#theory csp {
  linear_term {
    + : 5, unary;
    - : 5, unary;
    * : 4, binary, left;
    + : 3, binary, left;
    - : 3, binary, left
  }
  show_term {
    / : 1, binary, left
  }
  minimize_term {
    + : 5, unary;
    - : 5, unary;
  }
  dom_term {
    * : 4, binary, left;
    + : 3, binary, left;
    - : 3, binary, left;
    . : 1, binary, left
  }
  @ : 0, binary, left

  &dom/0 : dom_term, {=}, linear_term, any;
  &sum/0 : linear_term, {<=,=,>=,<,>,!=}, linear_term, any;
  &show/0 : show_term, directive;
  &distinct/0 : linear_term, any;
  &minimize/0 : minimize_term, directive
}. 
The example has exactly one solution

\{ s \mapsto 9, e \mapsto 5, n \mapsto 6, d \mapsto 7, m \mapsto 1, o \mapsto 0, r \mapsto 8, y \mapsto 2 \}
send + more = money

Each letter corresponds exactly to one digit and all variables have to be pairwisely distinct.

The example has exactly one solution:

\{ s \mapsto 9, \, e \mapsto 5, \, n \mapsto 6, \, d \mapsto 7, \, m \mapsto 1, \, o \mapsto 0, \, r \mapsto 8, \, y \mapsto 2 \}
Theory language

send + more = money

#include "csp.lp".

digit(1,3,s). digit(2,3,m). digit(sum,4,m).
digit(1,2,e). digit(2,2,o). digit(sum,3,o).
digit(1,1,n). digit(2,1,r). digit(sum,2,n).
digit(1,0,d). digit(2,0,e). digit(sum,1,e).


digit(sum,0,y).

base(10).
exp(E) :- digit(_,E,_,_).

power(1,0).
power(B*P,E) :- base(B), power(P,E-1), exp(E), E>0.

number(N) :- digit(N,_,_,_), N!= sum.
high(D) :- digit(N,E,D), not digit(N,E+1,_,_).

&dom {0..9} = X :- digit(_,_,X).

&sum { M*D : digit(N,E,D), power(M,E), number(N);
    ~M*D : digit(sum,E,D), power(M,E) } = 0.

&sum { D } > 0 :- high(D).

&distinct { D : digit(_,_,D) }.

&show { D : digit(_,_,D) }.
Theory language

```
#include "csp.lp".

digit(1,3,s). digit(2,3,m). digit(sum,4,m).
digit(1,2,e). digit(2,2,o). digit(sum,3,o).
digit(1,1,n). digit(2,1,r). digit(sum,2,n).
digit(1,0,d). digit(2,0,e). digit(sum,1,e).
    digit(sum,0,y).

base(10).
exp(E) :- digit(_,E,_).

power(1,0).
power(B*P,E) :- base(B), power(P,E-1), exp(E), E>0.

number(N) :- digit(N,_,_), N!= sum.
high(D) :- digit(N,E,D), not digit(N,E+1,_).

&dom {0..9} = X :- digit(_,_,X).

&sum { M*D : digit(N,E,D), power(M,E), number(N);
    ~M*D : digit(sum,E,D), power(M,E)         } = 0.

&sum { D } > 0 :- high(D).

&distinct { D : digit(_,_,D) }.

&show { D : digit(_,_,D) }.
```
The Theory language for the `send+more=money` puzzle is as follows:

```prolog
#include "csp.lp".

digit(1,3,s). digit(2,3,m). digit(sum,4,m).
digit(1,2,e). digit(2,2,o). digit(sum,3,o).
digit(1,1,n). digit(2,1,r). digit(sum,2,n).
digit(1,0,d). digit(2,0,e). digit(sum,1,e).
    digit(sum,0,y).

base(10).
exp(E) :- digit(_,E,_).

power(1,0).
power(B*P,E) :- base(B), power(P,E-1), exp(E), E>0.

number(N) :- digit(N,_,_), N!= sum.
high(D) :- digit(N,E,D), not digit(N,E+1,_).

&dom {0..9} = X :- digit(_,_,X).

&sum { M*D : digit(N,E,D), power(M,E), number(N);
       -M*D : digit(sum,E,D), power(M,E) } = 0.

&sum { D } > 0 :- high(D).
&distinct { D : digit(_,_,D) }.
&show { D : digit(_,_,D) }.
```
Theory language

send+more=money

digit(1,3,s). digit(2,3,m). digit(sum,4,m).
digit(1,2,e). digit(2,2,o). digit(sum,3,o).
digit(1,1,n). digit(2,1,r). digit(sum,2,n).
digit(1,0,d). digit(2,0,e). digit(sum,1,e).
    digit(sum,0,y).
base(10).
exp(0). exp(1). exp(2). exp(3). exp(4).

power(1,0).
power(10,1). power(100,2). power(1000,3). power(10000,4).

number(1). number(2).
high(s). high(m).

&dom{0..9}=s. &dom{0..9}=m. &dom{0..9}=e. &dom{0..9}=o. &dom{0..9}=n. &dom{0..9}=r. &dom{0..9}=d. &dom{0..9}=y.

&sum{ 1000*s; 100*e; 10*n; 1*d;
     1000*m; 100*o; 10*r; 1*e;
     -10000*m; -10000*o; -100*n; -10*e; -1*y } = 0.

&sum{s} > 0. &sum{m} > 0.

&distinct{s; m; e; o; n; r; d; y}.

&show{s; m; e; o; n; r; d; y}.
Outline

52 Theory language
53 Low-level semantics
54 Intermediate Format
55 Theory propagation
56 Experiments
57 Acyclicity checking
58 Constraint Answer Set Programming
ASP solving process **modulo theories**

- Problem
- Modeling
- Logic Program
- Grounding
- Solver
- Stable Models
- Interpreting
- Solution

**Solving**
ASP solving process \textit{modulo} theories

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models

Modeling

Solving

Solution

Interpreting

Low-level semantics
Low-level semantics

ASP modulo theories

- We distinguish theory atoms depending upon whether they are
  - defined via rules in the logic program, or
  - external otherwise, or
  - strict being equivalent to the associated constraint, or
  - non-strict only implying the associated constraint.

- Informally, a set $X \subseteq A \cup \mathcal{T}$ of atoms is a $\mathcal{T}$-stable model of a program $P$ if there is some $\mathcal{T}$-solution $S$ such that $X$ is a (regular) stable model of the program

$$P \cup \{a \leftarrow | a \in (\mathcal{T}_e \setminus h(P)) \cap S\}$$
$$\cup \{\leftarrow \neg a | a \in (\mathcal{T}_e \cap h(P)) \cap S\}$$
$$\cup \{\{a\} \leftarrow | a \in (\mathcal{T}_i \setminus h(P)) \cap S\}$$
$$\cup \{\leftarrow a | a \in (\mathcal{T} \cap h(P)) \setminus S\}$$
We distinguish theory atoms depending upon whether they are

- defined via rules in the logic program, or
- external otherwise, or
- strict being equivalent to the associated constraint, or
- non-strict only implying the associated constraint.

Informally, a set $X \subseteq A \cup \mathcal{T}$ of atoms is a $\mathcal{T}$-stable model of a program $P$ if there is some $\mathcal{T}$-solution $S$ such that $X$ is a (regular) stable model of the program

\[
P \cup \{a \leftarrow | a \in (\mathcal{T}_e \setminus h(P)) \cap S\} \\
\cup \{\leftarrow \neg a | a \in (\mathcal{T}_e \cap h(P)) \cap S\} \\
\cup \{\{a\} \leftarrow | a \in (\mathcal{T}_i \setminus h(P)) \cap S\} \\
\cup \{\leftarrow a | a \in (\mathcal{T} \cap h(P)) \setminus S\}
\]
We distinguish theory atoms depending upon whether they are
- defined via rules in the logic program, or
- external otherwise, or
- strict being equivalent to the associated constraint, or
- non-strict only implying the associated constraint.

Informally, a set $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\mathcal{T}$-stable model of a program $P$ if there is some $\mathcal{T}$-solution $S$ such that $X$ is a (regular) stable model of the program

$$P \cup \{a \leftarrow | a \in (\mathcal{T}_e \setminus h(P)) \cap S\}$$

$$\cup \{\leftarrow \sim a | a \in (\mathcal{T}_e \cap h(P)) \cap S\}$$

$$\cup \{\{a\} \leftarrow | a \in (\mathcal{T}_i \setminus h(P)) \cap S\}$$

$$\cup \{\leftarrow a | a \in (\mathcal{T} \cap h(P)) \setminus S\}$$
Low-level semantics

ASP modulo theories

- We distinguish theory atoms depending upon whether they are
  - defined via rules in the logic program, or
  - external otherwise, or
  - strict being equivalent to the associated constraint, $T_e$, or
  - non-strict only implying the associated constraint, $T_i$.

- Informally, a set $X \subseteq A \cup \mathcal{T}$ of atoms is a $\mathcal{T}$-stable model of a program $P$ if there is some $\mathcal{T}$-solution $S$ such that $X$ is a (regular) stable model of the program

\[
P \cup \{a \leftarrow \mid a \in (T_e \setminus h(P)) \cap S\} \\
\cup \{\leftarrow \neg a \mid a \in (T_e \cap h(P)) \cap S\} \\
\cup \{\{a\} \leftarrow \mid a \in (T_i \setminus h(P)) \cap S\} \\
\cup \{\leftarrow a \mid a \in (T \cap h(P)) \setminus S\}
\]
Outline

52 Theory language
53 Low-level semantics
54 Intermediate Format
55 Theory propagation
56 Experiments
57 Acyclicity checking
58 Constraint Answer Set Programming
ASP solving process \textit{modulo} theories

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models
- Solution
- Interpreting

Modeling

Solving
ASP solving process *modulo theories*

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models
- Solution

Modeling Interpreting

Solving
Intermediate Format

\begin{verbatim}
{a}.
b  :-  a.
c  :-  not  a.
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
\end{verbatim}

\textit{aspif} example
Intermediate Format

\textit{aspif} example

\{a\}.
b :- a.
c :- not a.

\begin{verbatim}
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
\end{verbatim}
aspif overview

- Rule statements
- Minimize statements
- Projection statements
- Output statements
- External statements
- Assumption statements
- Heuristic statements
- Edge statements
- Theory terms and atoms
- Comments
Intermediate Format

**aspif theory example**

task(1).
task(2).

duration(1, 200).
duration(2, 400).

&dom {1..1000} = beg(1).
&dom {1..1000} = end(1).
&dom {1..1000} = beg(2).
&dom {1..1000} = end(2).

&diff{end(1)-beg(1)}\leq200.
&diff{end(2)-beg(2)}\leq400.

&show{ beg/1; end/1 }.
aspif theory example

task(1).
task(2).
duration(1,200).
duration(2,400).

&dom {1..1000} = beg(1).
&dom {1..1000} = end(1).
&dom {1..1000} = beg(2).
&dom {1..1000} = end(2).

&diff{end(1)-beg(1)}<=200.
&diff{end(2)-beg(2)}<=400.

&show{ beg/1; end/1 }.
**aspif** theory example

```
task(1).
task(2).
duration(1,200).
duration(2,400).
&dom {1..1000} = beg(1).
&dom {1..1000} = end(1).
&dom {1..1000} = beg(2).
&dom {1..1000} = end(2).
&diff{end(1)-beg(1)}<=200.
&diff{end(2)-beg(2)}<=400.
&show{ beg/1; end/1 }.
```

**Only 6 (theory) atoms!**
Outline

52 Theory language
53 Low-level semantics
54 Intermediate Format
55 Theory propagation
56 Experiments
57 Acyclicity checking
58 Constraint Answer Set Programming
ASP solving process modulo theories

Problem → Modeling → Logic Program → Grounder → Solver → Stable Models → Solution

Solving

Interpreting
ASP solving process modulo theories

Problem

Logic Program

Grounder

Solver

Stable Models

Solution

Modeling

Solving

Interpreting

Torsten Schaub (KRR@UP)
ASP solving process \textbf{modulo theories}

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models
- Solution
- Interpreting

Theory propagation
**Theory propagation**

**Architecture of clasp**

- **Preprocessing**
  - Preprocessor
  - Program Builder

- **Logic Program**

- **Solver 1...n**
  - Decision Heuristic
  - Assignment Atoms/Bodies

- **Conflict Resolution**

- **Recording Nogoods**
  - Propagation
    - Unit Propagation
    - Post Propagation

- **Parallel Context**
  - Threads: $S_1, S_2, \ldots, S_n$
  - Counter: $T, W, \ldots, S$
  - Queue: $P_1, P_2, \ldots, P_n$

- **Shared Nogoods**

- **Shared Context**
  - Propositional Variables
  - Atoms
  - Bodies
  - Static Nogoods
  - Short Nogoods

- **Enumerator**

- **Nogood Distributor**

- **Preprocessing Program Builder**
  - Preprocessor
  - Program Builder

- **Preprocessing**
  - Preprocessor
  - Program Builder
Theory propagation

Conflict-driven constraint learning modulo theories

(1) initialize // register theory propagators and initialize watches

loop

propagate completion, loop, and recorded nogoods // deterministically assign literals

if no conflict then

if all variables assigned then

(C) if some $\delta \in \Delta_T$ is violated for $T \in T$ then record $\delta$ // theory propagator's check
else return variable assignment // $T$-stable model found

else

(P) propagate theories $T \in T$ // theory propagators may record theory nogoods

if no nogood recorded then decide // non-deterministically assign some literal
else

if top-level conflict then return unsatisfiable
else

analyze // resolve conflict and record a conflict constraint

(U) backjump // undo assignments until conflict constraint is unit
Propagator interface

**clingo**

- **SymbolicAtom**
  - `+ symbol`
  - `+ literal`

- **PropagateInit**
  - `+ num_threads`
  - `+ symbolic_atoms`
  - `+ theory_atoms`
  - `+ add_watch(lit)`
  - `+ solver_literal(lit)`

- **TheoryAtom**
  - `+ name`
  - `+ elements`
  - `+ guard`
  - `+ literal`

- **Assignment**
  - `+ decision_level`
  - `+ has_conflict`
  - `+ value(lit)`
  - `+ level(lit)`
  - `+ ...`

- **PropagateControl**
  - `+ thread_id`
  - `+ assignment`
  - `+ add_nogood(nogood, tag, lock)`
  - `+ propagate()`

- **interface**

**Propagator**

- `+ init(init)`
- `+ propagate(control, changes)`
- `+ undo(thread_id, assignment, changes)`
- `+ check(control)`
Theory propagation

The *dot* propagator

```python
#script (python)

import sys
import time

class Propagator:
    def __init__(self, init):
        self.sleep = .1
        for atom in init.symbolic_atoms:
            init.add_watch(init.solver_literal(atom.literal))

    def propagate(self, ctl, changes):
        for l in changes:
            sys.stdout.write("."
            sys.stdout.flush()
            time.sleep(self.sleep)
        return True

    def undo(self, solver_id, assign, undo):
        for l in undo:
            sys.stdout.write("\b \b")
            sys.stdout.flush()
            time.sleep(self.sleep)

def main(prg):
    prg.register_propagator(Propagator())
    prg.ground(["base", []])
    prg.solve()
    sys.stdout.write("\n")

#end.
```

Torsten Schaub (KRR@UP)
Outline

52 Theory language
53 Low-level semantics
54 Intermediate Format
55 Theory propagation
56 Experiments
57 Acyclicity checking
58 Constraint Answer Set Programming
## Difference logic propagation

<table>
<thead>
<tr>
<th>Problem</th>
<th>#</th>
<th>ASP T</th>
<th>ASP T</th>
<th>ASP modulo defined T</th>
<th>ASP modulo defined T</th>
<th>DL (stateless) external T</th>
<th>DL (stateless) external T</th>
<th>ASP modulo defined TO</th>
<th>ASP modulo defined TO</th>
<th>DL (stateless) external TO</th>
<th>DL (stateless) external TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow shop</td>
<td>120</td>
<td>569</td>
<td>110</td>
<td>283</td>
<td>40</td>
<td>382</td>
<td>70</td>
<td>177</td>
<td>30</td>
<td>281</td>
<td>50</td>
</tr>
<tr>
<td>Job shop</td>
<td>80</td>
<td>600</td>
<td>80</td>
<td>600</td>
<td>80</td>
<td>600</td>
<td>80</td>
<td>37</td>
<td>0</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>Open shop</td>
<td>60</td>
<td>405</td>
<td>40</td>
<td>214</td>
<td>20</td>
<td>213</td>
<td>20</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>260</td>
<td>525</td>
<td>230</td>
<td>366</td>
<td>140</td>
<td>398</td>
<td>170</td>
<td>72</td>
<td>30</td>
<td>109</td>
<td>50</td>
</tr>
</tbody>
</table>

- only non-strict interpretation of theory atoms
- defined versus external amounts to the difference between
  - &diff { end(T) - beg(T) } <= D :- duration(T,D).
  - :- duration(T,D), not &diff { end(T) - beg(T) } <= D.

- propagation
  - stateless Bellman-Ford algorithm
  - stateful Cotton-Maler algorithm
Experiments

Difference logic propagation

<table>
<thead>
<tr>
<th>Problem</th>
<th>#</th>
<th>ASP</th>
<th>ASP modulo</th>
<th>DL (stateless)</th>
<th>ASP modulo</th>
<th>DL (stateful)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>defined</td>
<td>external</td>
<td>T</td>
<td>external</td>
</tr>
<tr>
<td>Flow shop</td>
<td>120</td>
<td>569</td>
<td>382</td>
<td>70</td>
<td>177</td>
<td>30</td>
</tr>
<tr>
<td>Job shop</td>
<td>80</td>
<td>600</td>
<td>600</td>
<td>80</td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>Open shop</td>
<td>60</td>
<td>405</td>
<td>214</td>
<td>20</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>260</td>
<td>525</td>
<td>366</td>
<td>140</td>
<td>72</td>
<td>30</td>
</tr>
</tbody>
</table>

- **only non-strict interpretation of theory atoms**
- defined versus external amounts to the difference between
  - \&diff \{ end(T)-beg(T) \} <= D :- duration(T,D).
  - :- duration(T,D), not \&diff \{ end(T)-beg(T) \} <= D.

- propagation
  - stateless Bellman-Ford algorithm
  - stateful Cotton-Maler algorithm
Difference logic propagation

<table>
<thead>
<tr>
<th>Problem</th>
<th>#</th>
<th>ASP defined</th>
<th>ASP modulo DL (stateless) defined</th>
<th>ASP modulo DL (stateful) defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow shop</td>
<td>120</td>
<td>569</td>
<td>283</td>
<td>177</td>
</tr>
<tr>
<td>Job shop</td>
<td>80</td>
<td>600</td>
<td>600</td>
<td>37</td>
</tr>
<tr>
<td>Open shop</td>
<td>60</td>
<td>405</td>
<td>214</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>260</td>
<td>525</td>
<td>366</td>
<td>72</td>
</tr>
</tbody>
</table>

- only non-strict interpretation of theory atoms
- defined versus external amounts to the difference between
  - &diff { end(T)-beg(T) } <= D :- duration(T,D).
  - :- duration(T,D), not &diff { end(T)-beg(T) } <= D.

- propagation
  - stateless Bellman-Ford algorithm
  - stateful Cotton-Maler algorithm
Difference logic propagation

<table>
<thead>
<tr>
<th>Problem</th>
<th>#</th>
<th>ASP T</th>
<th>TO</th>
<th>ASP modulo defined T</th>
<th>TO</th>
<th>DL (stateless) external T</th>
<th>TO</th>
<th>ASP modulo defined T</th>
<th>TO</th>
<th>DL (stateful) external T</th>
<th>TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow shop</td>
<td>120</td>
<td>569</td>
<td>110</td>
<td>283</td>
<td>40</td>
<td>382</td>
<td>70</td>
<td>177</td>
<td>30</td>
<td>281</td>
<td>50</td>
</tr>
<tr>
<td>Job shop</td>
<td>80</td>
<td>600</td>
<td>80</td>
<td>600</td>
<td>80</td>
<td>600</td>
<td>80</td>
<td>37</td>
<td>0</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>Open shop</td>
<td>60</td>
<td>405</td>
<td>40</td>
<td>214</td>
<td>20</td>
<td>213</td>
<td>20</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>260</td>
<td>525</td>
<td>230</td>
<td>366</td>
<td>140</td>
<td>398</td>
<td>170</td>
<td>72</td>
<td>30</td>
<td>109</td>
<td>50</td>
</tr>
</tbody>
</table>

- only non-strict interpretation of theory atoms
- defined versus external amounts to the difference between
  - $\text{\&diff } \{ \text{end}(T)-\text{beg}(T) \} \leq D$ :- duration(T,D).
  - :- duration(T,D), not $\text{\&diff } \{ \text{end}(T)-\text{beg}(T) \} \leq D$.
- propagation
  - stateless Bellman-Ford algorithm
  - stateful Cotton-Maler algorithm
Difference logic propagation

<table>
<thead>
<tr>
<th>Problem</th>
<th>#</th>
<th>ASP</th>
<th>ASP modulo</th>
<th>DL (stateless)</th>
<th>ASP modulo</th>
<th>DL (stateful)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>defined</td>
<td>external</td>
<td>defined</td>
<td>external</td>
</tr>
<tr>
<td>Flow shop</td>
<td>120</td>
<td>569</td>
<td>283</td>
<td>382</td>
<td>177</td>
<td>281</td>
</tr>
<tr>
<td>Job shop</td>
<td>80</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>37</td>
<td>43</td>
</tr>
<tr>
<td>Open shop</td>
<td>60</td>
<td>405</td>
<td>214</td>
<td>213</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>260</td>
<td>525</td>
<td>366</td>
<td>398</td>
<td>72</td>
<td>109</td>
</tr>
</tbody>
</table>

- only non-strict interpretation of theory atoms
- defined versus external amounts to the difference between
  - &diff \{ end(T) - beg(T) \} \leq D : - duration(T,D).
  - duration(T,D), not &diff \{ end(T) - beg(T) \} \leq D.

- propagation
  - stateless Bellman-Ford algorithm
  - stateful Cotton-Maler algorithm
Outline

52  Theory language
53  Low-level semantics
54  Intermediate Format
55  Theory propagation
56  Experiments
57  Acyclicity checking
58  Constraint Answer Set Programming
Builtin acyclicity checking

- Edge statement

\[ \#\text{edge} \ (u, v) : \ l_1, \ldots, l_n. \] \hspace{1cm} (3)

- A set \( X \) of atoms is an acyclic stable of a logic program \( P \), if
  1. \( X \) is a stable model of \( P \) and
  2. the graph

\[
\left( \{ u, v \mid X \models l_1, \ldots, l_n, (3) \in P \}, \{(u, v) \mid X \models l_1, \ldots, l_n, (3) \in P \} \right)
\]

is acyclic
Builtin acyclicity checking

- Edge statement
  \[
  \#\text{edge} \ (u, v) : \ l_1, \ldots, l_n.
  \]  \hspace{1cm} (3)

- A set \( X \) of atoms is an acyclic stable of a logic program \( P \), if
  1. \( X \) is a stable model of \( P \) and
  2. the graph

  \[
  (\{u, v \mid X \models l_1, \ldots, l_n, (3) \in P\}, \{(u, v) \mid X \models l_1, \ldots, l_n, (3) \in P\})
  \]

  is acyclic
Outline

52 Theory language
53 Low-level semantics
54 Intermediate Format
55 Theory propagation
56 Experiments
57 Acyclicicity checking
58 Constraint Answer Set Programming
A constraint satisfaction problem (CSP) consists of
- a set $V$ of variables,
- a set $D$ of domains, and
- a set $C$ of constraints

such that
- each variable $v \in V$ has an associated domain $\text{dom}(v) \in D$;
- a constraint $c$ is a pair $(S, R)$ consisting of a $k$-ary relation $R$ on a
  vector $S \subseteq V^k$ of variables, called the scope of $R$

Note: For $S = (v_1, \ldots, v_k)$, we have $R \subseteq \text{dom}(v_1) \times \cdots \times \text{dom}(v_k)$
A constraint satisfaction problem (CSP) consists of
- a set \( V \) of variables,
- a set \( D \) of domains, and
- a set \( C \) of constraints

such that
- each variable \( v \in V \) has an associated domain \( \text{dom}(v) \in D \);
- a constraint \( c \) is a pair \((S, R)\) consisting of a \( k \)-ary relation \( R \) on a vector \( S \subseteq V^k \) of variables, called the scope of \( R \)

Note For \( S = (v_1, \ldots, v_k) \), we have \( R \subseteq \text{dom}(v_1) \times \cdots \times \text{dom}(v_k) \)
A constraint satisfaction problem (CSP) consists of

- a set $V$ of variables,
- a set $D$ of domains, and
- a set $C$ of constraints

such that

- each variable $v \in V$ has an associated domain $\text{dom}(v) \in D$;
- a constraint $c$ is a pair $(S, R)$ consisting of a $k$-ary relation $R$ on a vector $S \subseteq V^k$ of variables, called the scope of $R$

Note: For $S = (v_1, \ldots, v_k)$, we have $R \subseteq \text{dom}(v_1) \times \cdots \times \text{dom}(v_k)$
Example

Each letter corresponds exactly to one digit and all variables have to be pairwisely distinct

\[ \begin{align*}
  s & \rightarrow & 1 & \text{s} \\
  e & \rightarrow & 2 & \text{e} \\
  n & \rightarrow & 3 & \text{n} \\
  d & \rightarrow & 4 & \text{d} \\
  m & \rightarrow & 5 & \text{m} \\
  o & \rightarrow & 6 & \text{o} \\
  r & \rightarrow & 7 & \text{r} \\
  y & \rightarrow & 8 & \text{y} \\
\end{align*} \]

\[ \begin{align*}
  V &= \{s,e,n,d,m,o,r,y\} \\
  D &= \{\text{dom}(v) = \{0, \ldots, 9\} \mid v \in V\} \\
  C &= \{ (\vec{V}, \text{allDistinct}(V)), \\
        (\vec{V}, s \times 1000 + e \times 100 + n \times 10 + d + \\
         m \times 1000 + o \times 100 + r \times 10 + e == \\
         m \times 10000 + o \times 1000 + n \times 100 + e \times 10 + y), \\
        ((m), m == 1) \} \\
\end{align*} \]
Example

Each letter corresponds exactly to one digit and all variables have to be pairwise distinct.

\[
\begin{align*}
\text{send} & \quad + \quad \text{more} \\
\hline
\text{money} & \\
\end{align*}
\]

\[
V = \{s, e, n, d, m, o, r, y\}
\]

\[
D = \{\text{dom}(v) = \{0, \ldots, 9\} \mid v \in V\}
\]

\[
C = \{ (\vec{V}, \text{allDistinct}(V)), \\
(\vec{V}, s \times 1000 + e \times 100 + n \times 10 + d + m \times 1000 + o \times 100 + r \times 10 + e == m \times 10000 + o \times 1000 + n \times 100 + e \times 10 + y), \\
((m), m == 1)\}
\]
Example

Each letter corresponds exactly to one digit and all variables have to be pairwisely distinct.

The example has exactly one solution.

\{ s \mapsto 9, e \mapsto 5, n \mapsto 6, d \mapsto 7, m \mapsto 1, o \mapsto 0, r \mapsto 8, y \mapsto 2 \}
Constraint satisfaction problem

- **Notation** We use $S(c) = S$ and $R(c) = R$ to access the scope and the relation of a constraint $c = (S, R)$.

- For an assignment $A : V \rightarrow \bigcup_{v \in V} dom(v)$ and a constraint $(S, R)$ with scope $S = (v_1, \ldots, v_k)$, define

  $$sat_C(A) = \{ c \in C \mid A(S(c)) \in R(c) \}$$

  where $A(S) = (A(v_1), \ldots, A(v_k))$.
Constraint satisfaction problem

- **Notation** We use $S(c) = S$ and $R(c) = R$ to access the scope and the relation of a constraint $c = (S, R)$.

- For an assignment $A : V \to \bigcup_{v \in V} \text{dom}(v)$ and a constraint $(S, R)$ with scope $S = (v_1, \ldots, v_k)$, define

$$\text{sat}_C(A) = \{ c \in C \mid A(S(c)) \in R(c) \}$$

where $A(S) = (A(v_1), \ldots, A(v_k))$. 
A constraint logic program $P$ is a logic program over an extended alphabet $\mathcal{A} \cup \mathcal{C}$ where
- $\mathcal{A}$ is a set of regular atoms and
- $\mathcal{C}$ is a set of constraint atoms,
such that $h(r) \in \mathcal{A}$ for each $r \in P$

Given a set of literals $B$ and some set $\mathcal{B}$ of atoms, we define
$$B|_\mathcal{B} = (B^+ \cap \mathcal{B}) \cup \{\sim a \mid a \in B^- \cap \mathcal{B}\}$$
A constraint logic program $P$ is a logic program over an extended alphabet $\mathcal{A} \cup \mathcal{C}$ where

- $\mathcal{A}$ is a set of regular atoms and
- $\mathcal{C}$ is a set of constraint atoms,

such that $h(r) \in \mathcal{A}$ for each $r \in P$.

Given a set of literals $B$ and some set $\mathcal{B}$ of atoms, we define

$$B|_{\mathcal{B}} = (B^+ \cap \mathcal{B}) \cup \{\neg a \mid a \in B^- \cap \mathcal{B}\}$$
Constraint Answer Set Programming

- We identify constraint atoms with constraints via a function

\[ \gamma : \mathcal{C} \rightarrow \mathcal{C} \]

- Furthermore, \( \gamma(Y) = \{ \gamma(c) \mid c \in Y \} \) for any \( Y \subseteq \mathcal{C} \)

- Note Unlike regular atoms \( \mathcal{A} \), constraint atoms \( \mathcal{C} \) are not subject to the unique names assumption, e.g.

\[ \gamma(x < y) = \gamma(((y - 1) \leq -(x + 1)) \land (x \neq y)) \]

- A constraint logic program \( P \) is associated with a CSP as follows
  - \( C[P] = \gamma(\mathcal{A}(P) \cap \mathcal{C}) \),
  - \( V[P] \) is obtained from the constraint scopes in \( C[P] \),
  - \( D[P] \) is provided by a declaration
We identify constraint atoms with constraints via a function
\[ \gamma : \mathcal{C} \rightarrow \mathcal{C} \]

Furthermore, \( \gamma(Y) = \{ \gamma(c) \mid c \in Y \} \) for any \( Y \subseteq \mathcal{C} \)

Note Unlike regular atoms \( \mathcal{A} \), constraint atoms \( \mathcal{C} \) are not subject to the unique names assumption, eg.
\[ \gamma(x < y) = \gamma(((y - 1) \leq -(x + 1)) \land (x \neq y)) \]

A constraint logic program \( P \) is associated with a CSP as follows
- \( C[P] = \gamma(A(P) \cap \mathcal{C}) \)
- \( V[P] \) is obtained from the constraint scopes in \( C[P] \)
- \( D[P] \) is provided by a declaration
We identify constraint atoms with constraints via a function

$$\gamma : \mathcal{C} \rightarrow \mathcal{C}$$

Furthermore, $$\gamma(Y) = \{\gamma(c) \mid c \in Y\}$$ for any $$Y \subseteq \mathcal{C}$$

Note Unlike regular atoms $$\mathcal{A}$$, constraint atoms $$\mathcal{C}$$ are not subject to the unique names assumption, eg.

$$\gamma(x < y) = \gamma(((y - 1) \leq -(x + 1)) \land (x \neq y))$$

A constraint logic program $$P$$ is associated with a CSP as follows

- $$C[P] = \gamma(A(P) \cap \mathcal{C})$$,
- $$V[P]$$ is obtained from the constraint scopes in $$C[P]$$,
- $$D[P]$$ is provided by a declaration
Constraint Answer Set Programming

Let $P$ be a constraint logic program over $\mathcal{A} \cup \mathcal{C}$ and let $A : V[P] \rightarrow D[P]$ be an assignment, define the constraint reduct of as $P$ wrt $A$ as follows

$$P^A = \{ h(r) \leftarrow \text{body}(r)|_A \mid r \in P, \gamma(\text{body}(r)|_{\mathcal{C}^+}) \subseteq \text{sat}_{\mathcal{C}[P]}(A), \gamma(\text{body}(r)|_{\mathcal{C}^-}) \cap \text{sat}_{\mathcal{C}[P]}(A) = \emptyset \}$$

A set $X \subseteq \mathcal{A}$ of (regular) atoms is a constraint answer set of $P$ wrt $A$, if $X$ is an stable model of $P^A$.

Note That is, if $X$ is the $\subseteq$-smallest model of $(P^A)^X$
Let $P$ be a constraint logic program over $A \cup C$ and let $A : V[P] \rightarrow D[P]$ be an assignment,

define the constraint reduct of $P$ wrt $A$ as follows

$$P^A = \{ h(r) \leftarrow \text{body}(r)|_A \mid r \in P, \gamma(\text{body}(r)|_{C^+}) \subseteq \text{sat}_{C[P]}(A), \gamma(\text{body}(r)|_{C^-}) \cap \text{sat}_{C[P]}(A) = \emptyset \}$$

A set $X \subseteq A$ of (regular) atoms is a constraint answer set of $P$ wrt $A$, if $X$ is an stable model of $P^A$.

Note That is, if $X$ is the $\subseteq$-smallest model of $(P^A)^X$
Let $P$ be a constraint logic program over $A \cup C$ and let $A : V[P] \rightarrow D[P]$ be an assignment, define the constraint reduct of $P$ wrt $A$ as follows

$$P^A = \{ h(r) \leftarrow \text{body}(r)|_A \mid r \in P, \gamma(\text{body}(r)|_{C^+}) \subseteq \text{sat}_{C[P]}(A), \gamma(\text{body}(r)|_{C^-}) \cap \text{sat}_{C[P]}(A) = \emptyset \}$$

A set $X \subseteq A$ of (regular) atoms is a constraint answer set of $P$ wrt $A$, if $X$ is an stable model of $P^A$.

Note That is, if $X$ is the $\subseteq$-smallest model of $(P^A)^X$
Some Constraint Answer Set Programming (CASP) systems

- **adsolver**
  - extension of ASP solver *smodels*

- **clingcon**
  - extension of ASP system *clingo* (viz. *gringo* and *clasp*)
  - lazy approach

- **aspartame**
  - translational approach (independent of ASP system)
  - eager approach

- **aspmt, dlvhex, ezcsp, gasp, inca, ...**
Some Constraint Answer Set Programming (CASP) systems

- **adsolver**
  - extension of ASP solver *smodels*

- **clingcon**
  - extension of ASP system *clingo* (viz. *gringo* and *clasp*)
  - lazy approach

- **aspartame**
  - translational approach (independent of ASP system)
  - eager approach

- **aspmpt, dlvhex, ezcsp, gasp, inca, ...**
Some Constraint Answer Set Programming (CASP) systems

- **adsolver**
  - extension of ASP solver *smodels*

- **clingcon**
  - extension of ASP system *clingo* (viz. *gringo* and *clasp*)
  - lazy approach

- **aspartame**
  - translational approach (independent of ASP system)
  - eager approach

- *aspm*-*, dlv*-*, *ezcsp*, *gasp*, *inca*, ...
Some Constraint Answer Set Programming (CASP) systems

- **adsolver**
  - extension of ASP solver *smodels*

- **clingcon**
  - extension of ASP system *clingo* (viz. *gringo* and *clasp*)
  - lazy approach

- **aspartame**
  - translational approach (independent of ASP system)
  - eager approach

- **aspmpt, dlvhex, ezcsp, gasp, inca, ...**
aspartame’s eager approach

* based on order-encoding for CSPs
aspartame’s eager approach

- ASP Facts
- ASP Encoding

CASP Program

gringo

clasp

CASP Solution

* based on order-encoding for CSPs
aspartame’s eager approach

* based on order-encoding for CSPs
clingcon’s lazy approach

- **clingcon 1**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation*

[CASP Program] -> [gringo] -> [clasp] -> [CASP Solution]

- CSP
- CSP
- CSP
clingcon’s lazy approach

- **clingcon 1**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation*
**clingcon’s lazy approach**

![Diagram](image)

- **clingcon 1**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation*
**clingcon’s lazy approach**

- **clingcon 1**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation

---

Constraint Answer Set Programming
**clingcon**'s lazy approach

- **CASP Program**
  - **gringo**
    - CSP
  - **clasp**
    - CSP
  - **CASP Solution**

- **clingcon 1+2**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation

[Diagram showing the process from CASP Program through **gringo**, CSP, **clasp**, CSP, to CASP Solution. Diagram labeled with clingcon's lazy approach and details of the process.]
clingcon’s lazy approach

- **clingcon 1+2**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation*
clingcon’s lazy approach

- **clingcon 1+2**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation

Diagram:

1. CSP Grammar
2. CASP Program
3. gringo
4. clasp
5. CASP Solution

- CSP
- language extension
- propagation via *gecode*
- conflict minimization
- language specification
- lazy propagation
clingcon’s lazy approach

- **clingcon 1+2**
  - language extension
  - propagation via *gecode*
  - conflict minimization

- **clingcon 3**
  - language specification
  - lazy propagation*
clingcon’s approach

CSP Grammar

CASP Program

gringo

CSP

clasp

CSP

CASP Solution
clingcon instantiates clingo
Heuristic programming: Overview

59 Motivation
60 Heuristically modified ASP
61 Experimental results
59 Motivation

60 Heuristically modified ASP

61 Experimental results
Motivation

Observation  Sometimes it is advantageous to take a more application-oriented approach by including domain-specific information

- domain-specific knowledge can be added for improving propagation
- domain-specific heuristics can be used for making better choices

Idea  Incorporation of domain-specific heuristics by extending

- input language and/or solver options for expressing domain-specific heuristics
- solving capacities for integrating domain-specific heuristics
Observation Sometimes it is advantageous to take a more application-oriented approach by including domain-specific information.

- **domain-specific knowledge** can be added for improving propagation.
- **Domain-specific heuristics** can be used for making better choices.

Idea Incorporation of domain-specific heuristics by extending input language and/or solver options for expressing domain-specific heuristics and solving capacities for integrating domain-specific heuristics.
Observation: Sometimes it is advantageous to take a more application-oriented approach by including domain-specific information.

- Domain-specific knowledge can be added for improving propagation.
- Domain-specific heuristics can be used for making better choices.

Idea: Incorporation of domain-specific heuristics by extending input language and/or solver options for expressing domain-specific heuristics and solving capacities for integrating domain-specific heuristics.
Basic CDCL decision algorithm

**loop**

*propagate* // compute deterministic consequences

**if** no conflict **then**

*if* all variables assigned **then** return variable assignment

*else* decide // non-deterministically assign some literal

**else**

*if* top-level conflict **then** return unsatisfiable

*else*

*analyze* // analyze conflict and add a conflict constraint

*backjump* // undo assignments until conflict constraint is unit
Basic CDCL decision algorithm

**loop**

*propagate*  // compute deterministic consequences

*if* no conflict *then*

*if* all variables assigned *then return* variable assignment

*else decide*  // non-deterministically assign some literal

*else*

*if* top-level conflict *then return* unsatisfiable

*else*

*analyze*  // analyze conflict and add a conflict constraint

*backjump*  // undo assignments until conflict constraint is unit
Inside \textit{decide}

- Basic concepts
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \rightarrow \{\text{T}, \text{F}\}$
    \[ A^T = \{ a \in \mathcal{A} | T a \in \mathcal{A} \} \quad \text{and} \quad A^F = \{ a \in \mathcal{A} | F a \in \mathcal{A} \} \]

- Heuristic functions
  \[ h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{\text{T}, \text{F}\} \]

- Algorithmic scheme
  \begin{enumerate}
  \item $h(a) := \alpha \times h(a) + \beta(a)$
  \item $U := \mathcal{A} \setminus (A^T \cup A^F)$
  \item $C := \text{argmax}_{a \in U} h(a)$
  \item $a := \tau(C)$
  \item $\mathcal{A} := \mathcal{A} \cup \{ a \mapsto s(a) \}$
  \end{enumerate}

for each $a \in \mathcal{A}$
Motivation

Inside *decide*

- **Basic concepts**
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \rightarrow \{T, F\}$
    \[ A^T = \{ a \in \mathcal{A} \mid Ta \in A \} \quad \text{and} \quad A^F = \{ a \in \mathcal{A} \mid Fa \in A \} \]

- **Heuristic functions**
  \[ h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{T, F\} \]

- **Algorithmic scheme**
  1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
  2. $U := \mathcal{A} \setminus (A^T \cup A^F)$
  3. $C := \arg\max_{a \in U} h(a)$
  4. $a := \tau(C)$
  5. $A := A \cup \{ a \mapsto s(a) \}$
Basic concepts

- Atoms, \( \mathcal{A} \)
- Assignments, \( A : \mathcal{A} \to \{ \text{T}, \text{F} \} \)

\[ A^T = \{ a \in \mathcal{A} \mid \text{T}a \in \mathcal{A} \} \quad \text{and} \quad A^F = \{ a \in \mathcal{A} \mid \text{F}a \in \mathcal{A} \} \]

Heuristic functions

\( h : \mathcal{A} \to [0, +\infty) \) and \( s : \mathcal{A} \to \{ \text{T}, \text{F} \} \)

Algorithmic scheme

1. \( h(a) := \alpha \times h(a) + \beta(a) \) for each \( a \in \mathcal{A} \)
2. \( U := \mathcal{A} \setminus (A^T \cup A^F) \)
3. \( C := \arg\max_{a \in U} h(a) \)
4. \( a := \tau(C) \)
5. \( A := A \cup \{ a \mapsto s(a) \} \)
Motivation

Inside decide

- Basic concepts
  - Atoms, $\mathcal{A}$
  - Assignments, $A : \mathcal{A} \rightarrow \{T, F\}$
    
    $A^T = \{a \in \mathcal{A} | T_a \in A\}$ and $A^F = \{a \in \mathcal{A} | F_a \in A\}$

- Heuristic functions

  $h : \mathcal{A} \rightarrow [0, +\infty)$ and $s : \mathcal{A} \rightarrow \{T, F\}$

- Algorithmic scheme

  1. $h(a) := \alpha \times h(a) + \beta(a)$
     for each $a \in \mathcal{A}$
  2. $U := \mathcal{A} \setminus (A^T \cup A^F)$
  3. $C := \text{argmax}_{a \in U} h(a)$
  4. $a := \tau(C)$
  5. $A := \mathcal{A} \cup \{a \mapsto s(a)\}$
Outline

- Motivation
- Heuristically modified ASP
- Experimental results
Heuristic language

- **Heuristic directive**

  \[
  \text{#heuristic } a : l_1, \ldots, l_n. [k@p, m]
  \]

  where

  - \(a\) is an atom, and \(l_1, \ldots, l_n\) are literals
  - \(k\) and \(p\) are integers
  - \(m\) is a heuristic modifier

- **Heuristic modifiers**

  - `init` for initializing the heuristic value of \(a\) with \(k\)
  - `factor` for amplifying the heuristic value of \(a\) by factor \(k\)
  - `level` for ranking all atoms; the rank of \(a\) is \(k\)
  - `sign` for attributing the sign of \(k\) as truth value to \(a\)

- **Example**

  \[
  \text{#heuristic occurs}(A, T) : \text{action}(A), \text{time}(T). [T, \text{factor}]
  \]
Heuristic language

- **Heuristic directive**

  \[
  \#\text{heuristic } a : l_1, \ldots, l_n. [k@p, m]
  \]

  where

  - \( a \) is an atom, and \( l_1, \ldots, l_n \) are literals
  - \( k \) and \( p \) are integers
  - \( m \) is a heuristic modifier

- **Heuristic modifiers**

  - `init` for initializing the heuristic value of \( a \) with \( k \)
  - `factor` for amplifying the heuristic value of \( a \) by factor \( k \)
  - `level` for ranking all atoms; the rank of \( a \) is \( k \)
  - `sign` for attributing the sign of \( k \) as truth value to \( a \)

- **Example**

  \[
  \#\text{heuristic occurs}(A,T) : \text{action}(A), \text{time}(T). [T, \text{factor}]
  \]
Heuristic language

- **Heuristic directive**

  \[
  \#\text{heuristic } a : l_1, \ldots, l_n. [k@p, m]
  \]

  where

  - \(a\) is an atom, and \(l_1, \ldots, l_n\) are literals
  - \(k\) and \(p\) are integers
  - \(m\) is a heuristic modifier

- **Heuristic modifiers**

  - `init` for initializing the heuristic value of \(a\) with \(k\)
  - `factor` for amplifying the heuristic value of \(a\) by factor \(k\)
  - `level` for ranking all atoms; the rank of \(a\) is \(k\)
  - `sign` for attributing the sign of \(k\) as truth value to \(a\)

- **Example**

  \[
  \#\text{heuristic occurs}(A,T) : \text{action}(A), \text{time}(T). [T, \text{factor}]
  \]
Heuristic language

- Heuristic directive
  \[
  \text{#heuristic } a : l_1, \ldots, l_n. [k@p, m]
  \]
  where
  - $a$ is an atom, and $l_1, \ldots, l_n$ are literals
  - $k$ and $p$ are integers
  - $m$ is a heuristic modifier

- Heuristic modifiers
  - init for initializing the heuristic value of $a$ with $k$
  - factor for amplifying the heuristic value of $a$ by factor $k$
  - level for ranking all atoms; the rank of $a$ is $k$
  - sign for attributing the sign of $k$ as truth value to $a$

- Example
  \[
  \text{#heuristic occurs}(A,T) : \text{action}(A), \text{time}(T). [T, \text{factor}]
  \]
Heuristic language

- **Heuristic directive**
  
  \[ \texttt{#heuristic } a : l_1, \ldots, l_n. [k@p, m] \]

  where

  - \( a \) is an atom, and \( l_1, \ldots, l_n \) are literals
  - \( k \) and \( p \) are integers
  - \( m \) is a heuristic modifier

- **Heuristic modifiers**
  
  - \texttt{init} for initializing the heuristic value of \( a \) with \( k \)
  - \texttt{factor} for amplifying the heuristic value of \( a \) by factor \( k \)
  - \texttt{level} for ranking all atoms; the rank of \( a \) is \( k \)
  - \texttt{sign} for attributing the sign of \( k \) as truth value to \( a \)

- **Example**
  
  \[ \texttt{#heuristic occurs(mv,5) : action(mv), time(5). [5, factor]} \]
Heuristically modified ASP

Simple STRIPS planning

time(1..k).

holds(P,0) :- init(P).

\{ occ(A,T) : action(A) \} = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
Heuristically modified ASP

Simple STRIPS planning

time(1..k).

holds(P,0) :- init(P).

\{ \text{occ}(A,T) : \text{action}(A) \} = 1 :- \text{time}(T).
:- \text{occ}(A,T), \text{pre}(A,F), \text{not} \text{holds}(F,T-1).

holds(F,T) :- \text{occ}(A,T), \text{add}(A,F).
holds(F,T) :- \text{holds}(F,T-1), \text{time}(T), \text{not} \text{occ}(A,T) : \text{del}(A,F).

:- query(F), \text{not} \text{holds}(F,k).

#heuristic occurs(A,T) : \text{action}(A), \text{time}(T). [2, \text{factor}]
Simple STRIPS planning

time(1..k).

holds(P,0) :- init(P).

\{ occ(A,T) : action(A) \} = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic occurs(A,T) : action(A), time(T). [1, level]
Heuristically modified ASP

Simple STRIPS planning

time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic occurs(A,T) : action(A), time(T). [T, factor]
Heuristically modified ASP

Simple STRIPS planning

time(1..k).

holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).

#heuristic holds(F,T-1) : holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T) [t-T+1, false]
    fluent(F), time(T).
Heuristic options

Alternative for specifying structure-oriented heuristics in clasp

--dom-mod=<arg> : Default modification for domain heuristic
<arg>: <mod>[,<pick>]
<mod> : Modifier
{1=level|2=pos|3=true|4=neg|
  5=false|6=init|7=factor}
<pick> : Apply <mod> to
{0=all|1=scc|2=hcc|4=disj|
  8=min|16=show} atoms

Engage heuristic modifications (in both settings!)
--heuristic=Domain
Heuristic options

Alternative for specifying structure-oriented heuristics in \textit{clasp}

\texttt{--dom-mod=<arg>} : Default modification for domain heuristic
\texttt{<arg>} : \texttt{<mod>[,<pick>]}  
\texttt{<mod>} : Modifier  
\{1=level|2=pos|3=true|4=neg|  
\quad 5=false|6=init|7=factor\}  
\texttt{<pick>} : Apply \texttt{<mod>} to  
\{0=all|1=scc|2=hcc|4=disj|  
\quad 8=min|16=show\} atoms

Engage heuristic modifications (in both settings!)
\texttt{--heuristic=Domain}
Heuristic options

Alternative for specifying structure-oriented heuristics in *clasp*

`--dom-mod=<arg>` : Default modification for domain heuristic

<arg> : <mod>[,<pick>]

<mod> : Modifier

{1=level|2=pos|3=true|4=neg|5=false|6=init|7=factor}

<pick> : Apply <mod> to

{0=all|1=scc|2=hcc|4=disj|8=min|16=show} atoms

Engage heuristic modifications (in both settings!)

`--heuristic=Domain`
Inclusion-minimal stable models

- Consider a logic program containing a minimize statement of form:
  - \#minimize\{a_1, \ldots, a_n\}

- Computing one inclusion-minimal stable model can be done either via:
  - \#heuristic \( a_i \ [1, false] \) for \( i = 1, \ldots, n \), or
  - \(--\text{dom-mod}=5,16\)

- Computing all inclusion-minimal stable model can be done by adding \(--\text{enum-mod}=\text{domRec}\) to the two options.
Inclusion-minimal stable models

- Consider a logic program containing a minimize statement of form
  \[
  \text{#minimize}\{a_1,\ldots,a_n\}
  \]

- Computing one inclusion-minimal stable model can be done either via
  \[
  \text{#heuristic } a_i \ [1,\text{false}]. \quad \text{for } i = 1,\ldots,n, \text{ or}
  \]
  \[
  \text{--dom-mod}=5,16
  \]

- Computing all inclusion-minimal stable model can be done
  by adding \text{--enum-mod}=\text{domRec} to the two options
Inclusion-minimal stable models

- Consider a logic program containing a minimize statement of form
  #minimize\{a_1,\ldots,a_n\}

- Computing one inclusion-minimal stable model can be done either via
  #heuristic a_i [1,false]. for i = 1,\ldots, n, or
  --dom-mod=5,16

- Computing all inclusion-minimal stable model can be done
  by adding --enum-mod=domRec to the two options
Heuristic modifications to functions $h$ and $s$

- $\nu_{a,m}(A)$ — "value for modifier $m$ on atom $a$ wrt assignment $A$"

- $\text{init}$ and $\text{factor}$

  \[
  d_0(a) = \nu_{a,\text{init}}(A_0) + h_0(a) \\
  d_i(a) = \begin{cases} 
    \nu_{a,\text{factor}}(A_i) \times h_i(a) & \text{if } V_{a,\text{factor}}(A_i) \neq \emptyset \\
    h_i(a) & \text{otherwise}
  \end{cases}
  \]

- $\text{sign}$

  \[
  t_i(a) = \begin{cases} 
    T & \text{if } \nu_{a,\text{sign}}(A_i) > 0 \\
    F & \text{if } \nu_{a,\text{sign}}(A_i) < 0 \\
    s_i(a) & \text{otherwise}
  \end{cases}
  \]

- $\text{level}$

  \[
  \ell_{A_i}(A') = \arg\max_{a \in A'} \nu_{a,\text{level}}(A_i) \quad A' \subseteq A
  \]
Heuristically modified ASP

Heuristic modifications to functions $h$ and $s$

- $\nu_{a,m}(A)$ — "value for modifier $m$ on atom $a$ wrt assignment $A$"

- $d_0(a) = \nu_{a,\text{init}}(A_0) + h_0(a)$

- $d_i(a) = \begin{cases} 
\nu_{a,\text{factor}}(A_i) \times h_i(a) & \text{if } V_{a,\text{factor}}(A_i) \neq \emptyset \\

h_i(a) & \text{otherwise}
\end{cases}$

- $t_i(a) = \begin{cases} 
T & \text{if } \nu_{a,\text{sign}}(A_i) > 0 \\
F & \text{if } \nu_{a,\text{sign}}(A_i) < 0 \\

s_i(a) & \text{otherwise}
\end{cases}$

- $\ell_{A_i}(A') = \argmax_{a \in A'} \nu_{a,\text{level}}(A_i)$ with $A' \subseteq A$
Heuristics for $\nu$ and $h$

- $\nu_{a,m}(A)$ — "value for modifier $m$ on atom $a$ with respect to assignment $A$"

- $d_0(a) = \nu_{a,\text{init}}(A_0) + h_0(a)$

- $d_i(a) = \begin{cases} 
\nu_{a,\text{factor}}(A_i) \times h_i(a) & \text{if } V_{a,\text{factor}}(A_i) \neq \emptyset \\
\nu_{a,\text{init}} & \text{otherwise}
\end{cases}$

- $t_i(a) = \begin{cases} 
\text{T} & \text{if } \nu_{a,\text{sign}}(A_i) > 0 \\
\text{F} & \text{if } \nu_{a,\text{sign}}(A_i) < 0 \\
\text{F} & \text{otherwise}
\end{cases}$

- $s_i(a)$

- Level $\ell_{A_i}(A') = \arg\max_{a \in A'} \nu_{a,\text{level}}(A_i)$

- $A' \subseteq A$
Inside \textit{decide}, heuristically modified

\begin{enumerate}
\item $h(a) := d(a)$ for each $a \in A$
\item $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in A$
\item $U := \ell_A(A \setminus (A^T \cup A^F))$
\item $C := \arg\max_{a \in U} d(a)$
\item $a := \tau(C)$
\item $A := A \cup \{a \mapsto t(a)\}$
\end{enumerate}
Inside *decide*, heuristically modified

0. $h(a) := d(a)$ for each $a \in A$

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in A$

2. $U := \ell_A(\mathcal{A} \setminus (A^T \cup A^F))$

3. $C := \arg\max_{a \in U} d(a)$

4. $a := \tau(C)$

5. $A := A \cup \{a \mapsto t(a)\}$
Inside *decide*, heuristically modified

\begin{align*}
0 & \quad h(a) := d(a) \\
1 & \quad h(a) := \alpha \times h(a) + \beta(a) \\
2 & \quad U := \ell_A(A \setminus (A^T \cup A^F)) \\
3 & \quad C := \arg\max_{a \in U} d(a) \\
4 & \quad a := \tau(C) \\
5 & \quad A := A \cup \{a \mapsto t(a)\}
\end{align*}

for each $a \in A$

for each $a \in A$
Outline

- Motivation
- Heuristically modified ASP
- Experimental results
Abductive problems with optimization

<table>
<thead>
<tr>
<th>Setting</th>
<th>Diagnosis</th>
<th>Expansion</th>
<th>Repair (H)</th>
<th>Repair (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>base configuration</td>
<td>111.1s (115)</td>
<td>161.5s (100)</td>
<td>101.3s (113)</td>
<td>33.3s (27)</td>
</tr>
<tr>
<td>sign,-1</td>
<td>324.5s (407)</td>
<td>7.6s (3)</td>
<td>8.4s (5)</td>
<td>3.1s (0)</td>
</tr>
<tr>
<td>sign,-1 factor,2</td>
<td>310.1s (387)</td>
<td>7.4s (2)</td>
<td>3.5s (0)</td>
<td>3.2s (1)</td>
</tr>
<tr>
<td>sign,-1 factor,8</td>
<td>305.9s (376)</td>
<td>7.7s (2)</td>
<td>3.1s (0)</td>
<td>2.9s (0)</td>
</tr>
<tr>
<td>sign,-1 level,1</td>
<td>76.1s (83)</td>
<td>6.6s (2)</td>
<td>0.8s (0)</td>
<td>2.2s (1)</td>
</tr>
<tr>
<td>level,1</td>
<td>77.3s (86)</td>
<td>12.9s (5)</td>
<td>3.4s (0)</td>
<td>2.1s (0)</td>
</tr>
</tbody>
</table>

(abducibles subject to optimization via \#minimize statements)
# Abductive problems with optimization

<table>
<thead>
<tr>
<th>Setting</th>
<th>Diagnosis</th>
<th>Expansion</th>
<th>Repair (H)</th>
<th>Repair (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>base configuration</td>
<td>111.1s (115)</td>
<td>161.5s (100)</td>
<td>101.3s (113)</td>
<td>33.3s (27)</td>
</tr>
<tr>
<td>sign,-1</td>
<td>324.5s (407)</td>
<td>7.6s (3)</td>
<td>8.4s (5)</td>
<td>3.1s (0)</td>
</tr>
<tr>
<td>sign,-1 factor,2</td>
<td>310.1s (387)</td>
<td>7.4s (2)</td>
<td>3.5s (0)</td>
<td>3.2s (1)</td>
</tr>
<tr>
<td>sign,-1 factor,8</td>
<td>305.9s (376)</td>
<td>7.7s (2)</td>
<td>3.1s (0)</td>
<td>2.9s (0)</td>
</tr>
<tr>
<td>sign,-1 level,1</td>
<td>76.1s (83)</td>
<td>6.6s (2)</td>
<td>0.8s (0)</td>
<td>2.2s (1)</td>
</tr>
<tr>
<td>level,1</td>
<td>77.3s (86)</td>
<td>12.9s (5)</td>
<td>3.4s (0)</td>
<td>2.1s (0)</td>
</tr>
</tbody>
</table>

(abducibles subject to optimization via \#minimize statements)
Planning benchmarks

#heuristic holds(F,T-1) : holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T), fluent(F),time(T). [t-T+1, false]

<table>
<thead>
<tr>
<th>Problem</th>
<th>base configuration</th>
<th>#heuristic</th>
<th>base config. (SAT)</th>
<th>#heu. (SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks'00</td>
<td>134.4s (180/61)</td>
<td>9.2s (239/3)</td>
<td>163.2s (59)</td>
<td>2.6s (0)</td>
</tr>
<tr>
<td>Elevator'00</td>
<td>3.1s (279/0)</td>
<td>0.0s (279/0)</td>
<td>3.4s (0)</td>
<td>0.0s (0)</td>
</tr>
<tr>
<td>Freecell'00</td>
<td>288.7s (147/115)</td>
<td>184.2s (194/74)</td>
<td>226.4s (47)</td>
<td>52.0s (0)</td>
</tr>
<tr>
<td>Logistics'00</td>
<td>145.8s (148/61)</td>
<td>115.3s (168/52)</td>
<td>113.9s (23)</td>
<td>15.5s (3)</td>
</tr>
<tr>
<td>Depots'02</td>
<td>400.3s (51/184)</td>
<td>297.4s (115/135)</td>
<td>389.0s (64)</td>
<td>61.6s (0)</td>
</tr>
<tr>
<td>Driverlog'02</td>
<td>308.3s (108/143)</td>
<td>189.6s (169/92)</td>
<td>245.8s (61)</td>
<td>6.1s (0)</td>
</tr>
<tr>
<td>Rovers'02</td>
<td>245.8s (138/112)</td>
<td>165.7s (179/79)</td>
<td>162.9s (41)</td>
<td>5.7s (0)</td>
</tr>
<tr>
<td>Satellite'02</td>
<td>398.4s (73/186)</td>
<td>229.9s (155/106)</td>
<td>364.6s (82)</td>
<td>30.8s (0)</td>
</tr>
<tr>
<td>Zenotravel'02</td>
<td>350.7s (101/169)</td>
<td>239.0s (154/116)</td>
<td>224.5s (53)</td>
<td>6.3s (0)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>252.8s (1225/1031)</td>
<td>158.9s (1652/657)</td>
<td>187.2s (430)</td>
<td>17.1s (3)</td>
</tr>
</tbody>
</table>
#heuristic holds(F,T-1) : holds(F,T). [t-T+1, true]
#heuristic holds(F,T-1) : not holds(F,T), fluent(F),time(T). [t-T+1, false]

<table>
<thead>
<tr>
<th>Problem</th>
<th>base configuration</th>
<th>#heuristic</th>
<th>base config. (SAT)</th>
<th>#heu. (SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blocks’00</strong></td>
<td>134.4s (180/61)</td>
<td>9.2s (239/3)</td>
<td>163.2s (59)</td>
<td>2.6s (0)</td>
</tr>
<tr>
<td><strong>Elevator’00</strong></td>
<td>3.1s (279/0)</td>
<td>0.0s (279/0)</td>
<td>3.4s (0)</td>
<td>0.0s (0)</td>
</tr>
<tr>
<td><strong>Freecell’00</strong></td>
<td>288.7s (147/115)</td>
<td>184.2s (194/74)</td>
<td>226.4s (47)</td>
<td>52.0s (0)</td>
</tr>
<tr>
<td><strong>Logistics’00</strong></td>
<td>145.8s (148/61)</td>
<td>115.3s (168/52)</td>
<td>113.9s (23)</td>
<td>15.5s (3)</td>
</tr>
<tr>
<td><strong>Depots’02</strong></td>
<td>400.3s (51/184)</td>
<td>297.4s (115/135)</td>
<td>389.0s (64)</td>
<td>61.6s (0)</td>
</tr>
<tr>
<td><strong>Driverlog’02</strong></td>
<td>308.3s (108/143)</td>
<td>189.6s (169/92)</td>
<td>245.8s (61)</td>
<td>6.1s (0)</td>
</tr>
<tr>
<td><strong>Rovers’02</strong></td>
<td>245.8s (138/112)</td>
<td>165.7s (179/79)</td>
<td>162.9s (41)</td>
<td>5.7s (0)</td>
</tr>
<tr>
<td><strong>Satellite’02</strong></td>
<td>398.4s (73/186)</td>
<td>229.9s (155/106)</td>
<td>364.6s (82)</td>
<td>30.8s (0)</td>
</tr>
<tr>
<td><strong>Zenotravel’02</strong></td>
<td>350.7s (101/169)</td>
<td>239.0s (154/116)</td>
<td>224.5s (53)</td>
<td>6.3s (0)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>252.8s (1225/1031)</td>
<td>158.9s (1652/657)</td>
<td>187.2s (430)</td>
<td>17.1s (3)</td>
</tr>
</tbody>
</table>
### Experimental results

#### Planning benchmarks

---

**#heuristic holds(F,T-1):** holds(F,T).  \( [t-T+1, \text{true}] \)

**#heuristic holds(F,T-1): not holds(F,T), fluent(F), time(T). \( [t-T+1, \text{false}] \)**

<table>
<thead>
<tr>
<th>Problem</th>
<th>base configuration</th>
<th>#heuristic</th>
<th>base config. (SAT)</th>
<th>#heu. (SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blocks’00</strong></td>
<td>134.4s (180/61)</td>
<td>9.2s (239/3)</td>
<td>163.2s (59)</td>
<td>2.6s (0)</td>
</tr>
<tr>
<td><strong>Elevator’00</strong></td>
<td>3.1s (279/0)</td>
<td>0.0s (279/0)</td>
<td>3.4s (0)</td>
<td>0.0s (0)</td>
</tr>
<tr>
<td><strong>Freecell’00</strong></td>
<td>288.7s (147/115)</td>
<td>184.2s (194/74)</td>
<td>226.4s (47)</td>
<td>52.0s (0)</td>
</tr>
<tr>
<td><strong>Logistics’00</strong></td>
<td>145.8s (148/61)</td>
<td>115.3s (168/52)</td>
<td>113.9s (23)</td>
<td>15.5s (3)</td>
</tr>
<tr>
<td><strong>Depots’02</strong></td>
<td>400.3s (51/184)</td>
<td>297.4s (115/135)</td>
<td>389.0s (64)</td>
<td>61.6s (0)</td>
</tr>
<tr>
<td><strong>Driverlog’02</strong></td>
<td>308.3s (108/143)</td>
<td>189.6s (169/92)</td>
<td>245.8s (61)</td>
<td>6.1s (0)</td>
</tr>
<tr>
<td><strong>Rovers’02</strong></td>
<td>245.8s (138/112)</td>
<td>165.7s (179/79)</td>
<td>162.9s (41)</td>
<td>5.7s (0)</td>
</tr>
<tr>
<td><strong>Satellite’02</strong></td>
<td>398.4s (73/186)</td>
<td>229.9s (155/106)</td>
<td>364.6s (82)</td>
<td>30.8s (0)</td>
</tr>
<tr>
<td><strong>Zenotravel’02</strong></td>
<td>350.7s (101/169)</td>
<td>239.0s (154/116)</td>
<td>224.5s (53)</td>
<td>6.3s (0)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>252.8s (1225/1031)</td>
<td>158.9s (1652/657)</td>
<td>187.2s (430)</td>
<td>17.1s (3)</td>
</tr>
</tbody>
</table>

---

---

Torsten Schaub (KRR@UP)  
Answer Set Solving in Practice  
February 18, 2019  
508 / 653
Systems: Overview

Potassco
gringo
clap
clingo
clingcon
claspfolio
clavis
Potassco, the Potsdam Answer Set Solving Collection, bundles tools for ASP developed at the University of Potsdam, for instance:

- Grounder: gringo, lingo,
- Solver: clasp, claspfolio, claspar, aspeed
- Grounder+Solver: Clingo, Clingcon, ROSoClingo
- Further Tools: aspartame, aspcud, aspic, asprin, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- Benchmark repository: asparagus.cs.uni-potsdam.de
- Teaching material: potassco.org/teaching
Potassco, the Potsdam Answer Set Solving Collection, bundles tools for ASP developed at the University of Potsdam, for instance:

- **Grounder** gringo, lingo,
- **Solver** clasp, claspfolio, claspar, aspeed
- **Grounder+Solver** Clingo, Clingcon, ROSoClingo
- **Further Tools** aspartame, aspcud, aspic, asprin, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- Benchmark repository asparagus.cs.uni-potsdam.de
- Teaching material potassco.org/teaching
Potassco, the Potsdam Answer Set Solving Collection, bundles tools for ASP developed at the University of Potsdam, for instance:

- **Grounder** gringo, lingo,
- **Solver** clasp, claspfolio, claspar, aspeed
- **Grounder+Solver** Clingo, Clingcon, ROSoClingo
- **Further Tools** aspartame, aspcud, aspic, asprin, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- **Benchmark repository** asparagus.cs.uni-potsdam.de
- **Teaching material** potassco.org/teaching
Outline

62 Potassco

63 gringo

64 clasp

65 clingo

66 clingcon

67 claspfolio

68 clavis
- Accepts safe programs with aggregates
- Tolerates unrestricted use of function symbols (as long as it yields a finite ground instantiation :)
- Expressive power of a Turing machine

Basic architecture of *gringo*:

```
Parser -> Preprocessor -> Grounder

--ground

Output
--lparse
--text
--reify
```
Selected directives

- **Output**

  
  
  \[
  \#\text{show.} \quad \#\text{show } p/n. \quad \#\text{show } t : l_1, \ldots, l_n. 
  \]

- **Projection**

  
  
  \[
  \#\text{project } p/n. \quad \#\text{project } a : l_1, \ldots, l_n. 
  \]

- **Heuristics**

  
  
  \[
  \#\text{heuristic } a : l_1, \ldots, l_n. [k@p, m] 
  \]

- **Acyclicity**

  
  
  \[
  \#\text{edge } (u, v) : l_1, \ldots, l_n. 
  \]
Selected directives

- Output

  \#show.  \#show p/n.  \#show t : l_1, \ldots, l_n.

- Projection

  \#project p/n.  \#project a : l_1, \ldots, l_n.

- Heuristics

  \#heuristic a : l_1, \ldots, l_n. [k@p, m]

- Acyclicity

  \#edge (u, v) : l_1, \ldots, l_n.
Selected directives

- **Output**
  
  ```
  #show.    #show p/n.    #show t : l₁,...,lₙ.
  ```

- **Projection**
  
  ```
  #project p/n.    #project a : l₁,...,lₙ.
  ```

- **Heuristics**
  
  ```
  #heuristic a : l₁,...,lₙ. [k@p, m]
  ```

- **Acyclicity**
  
  ```
  #edge (u, v) : l₁,...,lₙ.
  ```
Selected directives

- **Output**
  
  \#show.  \#show \(p/n\).  \#show \(t : l_1, \ldots, l_n\).

- **Projection**
  
  \#project \(p/n\).  \#project \(a : l_1, \ldots, l_n\).

- **Heuristics**
  
  \#heuristic \(a : l_1, \ldots, l_n. [k@p, m]\)

- **Acyclicity**
  
  \#edge \((u, v) : l_1, \ldots, l_n\).
Potassco
gringo
clasp
clingo
clingcon
claspfolio
clavis
Outline

62 Potassco
63 gringo
64 clasp
   - Features
     - Parallel solving
     - Configuration
     - Disjunctive solving
65 clingo
66 clingcon
67 claspfolio
68 clavis
clasp is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:

- advanced preprocessing, including equivalence reasoning
- lookback-based decision heuristics
- restart policies
- nogood deletion
- progress saving
- dedicated data structures for binary and ternary nogoods
- lazy data structures (watched literals) for long nogoods
- dedicated data structures for cardinality and weight constraints
- lazy unfounded set checking based on “source pointers”
- tight integration of unit propagation and unfounded set checking
- various reasoning modes
- parallel search
- ...
clasp

- **clasp** is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on “source pointers”
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Torsten Schaub (KRR@UP)  Answer Set Solving in Practice  February 18, 2019
clasp

- **clasp** is a native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - advanced preprocessing, including equivalence reasoning
  - lookback-based decision heuristics
  - restart policies
  - nogood deletion
  - progress saving
  - dedicated data structures for binary and ternary nogoods
  - lazy data structures (watched literals) for long nogoods
  - dedicated data structures for cardinality and weight constraints
  - lazy unfounded set checking based on "source pointers"
  - tight integration of unit propagation and unfounded set checking
  - various reasoning modes
  - parallel search
  - ...

Torsten Schaub (KRR@UP) Answer Set Solving in Practice February 18, 2019 517 / 653
Reasoning modes of \textit{clasp}

- Beyond deciding (stable) model existence, \textit{clasp} allows for:
  - Optimization
  - Enumeration
  - Projective enumeration
  - Intersection and Union
  - and combinations thereof

- \textit{clasp} allows for:
  - ASP solving (\textit{smodels} format)
  - MaxSAT and SAT solving (extended \textit{dimacs} format)
  - PB solving (\textit{opb} and \textit{wbo} format)
Reasoning modes of clasp

- Beyond deciding (stable) model existence, clasp allows for:
  - Optimization
  - Enumeration (without solution recording)
  - Projective enumeration (without solution recording)
  - Intersection and Union (linear solution computation)
  - and combinations thereof

- clasp allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)
Outline

62 Potassco

63 gringo

64 clasp
  ■ Features
  ■ Parallel solving
  ■ Configuration
  ■ Disjunctive solving

65 clingo

66 clingcon

67 claspfolio

68 clavis
Parallel search in \textit{clasp}

- \textit{clasp}
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies
Parallel search in *clasp*

- *clasp*
  - pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
    - up to 64 configurable (non-hierarchic) threads
  - allows for parallel solving via search space splitting and/or competing strategies
    - both supported by solver portfolios
  - features different nogood exchange policies
Parallel search in \textit{clasp}

\begin{itemize}
  \item \textit{clasp}
    \begin{itemize}
      \item pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
        \begin{itemize}
          \item up to 64 configurable (non-hierarchic) threads
        \end{itemize}
      \item allows for parallel solving via search space splitting and/or competing strategies
        \begin{itemize}
          \item both supported by solver portfolios
        \end{itemize}
    \end{itemize}
  \item features different nogood exchange policies
\end{itemize}
Parallel search in \textit{clasp}

- \textit{clasp} pursues a coarse-grained, task-parallel approach to parallel search via shared memory multi-threading
  - up to 64 configurable (non-hierarchic) threads
- allows for parallel solving via search space splitting and/or competing strategies
  - both supported by solver portfolios
- features different nogood exchange policies
Sequential CDCL-style solving

**loop**

- **propagate**  // deterministically assign literals
- **if** no conflict **then**
  - **if** all variables assigned **then return** solution
  - **else decide**  // non-deterministically assign some literal
- **else**
  - **if** top-level conflict **then return** unsatisfiable
  - **else**
    - **analyze**  // analyze conflict and add conflict constraint
    - **backjump**  // unassign literals until conflict constraint is unit
Parallel CDCL-style solving in *clasp*

\[ \text{while} \ \text{work available} \]
\[ \ \text{while} \ \text{no (result) message to send} \]
\[ \text{communicate} \quad // \text{exchange information with other solver} \]
\[ \text{propagate} \quad // \text{deterministically assign literals} \]
\[ \text{if} \ \text{no conflict} \ \text{then} \]
\[ \quad \text{if} \ \text{all variables assigned} \ \text{then send} \ \text{solution} \]
\[ \quad \text{else} \ \text{decide} \quad // \text{non-deterministically assign some literal} \]
\[ \text{else} \]
\[ \quad \text{if} \ \text{root-level conflict} \ \text{then send} \ \text{unsatisfiable} \]
\[ \quad \text{else if} \ \text{external conflict} \ \text{then send} \ \text{unsatisfiable} \]
\[ \quad \text{else} \]
\[ \quad \text{analyze} \quad // \text{analyze conflict and add conflict constraint} \]
\[ \quad \text{backjump} \quad // \text{unassign literals until conflict constraint is unit} \]
\[ \text{communicate} \quad // \text{exchange results (and receive work)} \]
Parallel CDCL-style solving in \textit{clasp}

\begin{verbatim}
while work available
  while no (result) message to send
    communicate // exchange information with other solver
    propagate // deterministically assign literals
  if no conflict then
    if all variables assigned \textbf{then send} solution
    else \textbf{decide} // non-deterministically assign some literal
  else
    if root-level conflict \textbf{then send} unsatisfiable
    else if external conflict \textbf{then send} unsatisfiable
    else
      analyze // analyze conflict and add conflict constraint
      backjump // unassign literals until conflict constraint is unit
    communicate // exchange results (and receive work)
\end{verbatim}
Parallel CDCL-style solving in \textit{clasp}

\begin{verbatim}
while work available
  while no (result) message to send
    communicate  // exchange information with other solver
    propagate  // deterministically assign literals
  if no conflict then
    if all variables assigned then send solution
    else decide  // non-deterministically assign some literal
  else
    if root-level conflict then send unsatisfiable
    else if external conflict then send unsatisfiable
    else
      analyze  // analyze conflict and add conflict constraint
      backjump  // unassign literals until conflict constraint is unit
    communicate  // exchange results (and receive work)
\end{verbatim}
Parallel CDCL-style solving in *clasp*

```plaintext
while work available
  while no (result) message to send
    communicate  // exchange information with other solver
    propagate    // deterministically assign literals
  if no conflict then
    if all variables assigned then send solution
    else decide    // non-deterministically assign some literal
  else
    if root-level conflict then send unsatisfiable
    else if external conflict then send unsatisfiable
    else
      analyze      // analyze conflict and add conflict constraint
      backjump     // unassign literals until conflict constraint is unit
    communicate  // exchange results (and receive work)
```
Parallel CDCL-style solving in \textit{clasp}

\begin{verbatim}
while work available
  while no (result) message to send
    communicate  // exchange information with other solver
    propagate    // deterministically assign literals
  if no conflict then
    if all variables assigned then send solution
    else decide  // non-deterministically assign some literal
  else
    if root-level conflict then send unsatisfiable
    else if external conflict then send unsatisfiable
    else
      analyze     // analyze conflict and add conflict constraint
      backjump    // unassign literals until conflict constraint is unit
    communicate  // exchange results (and receive work)
\end{verbatim}
Multi-threaded architecture of \textit{clasp}
Multi-threaded architecture of \textit{clasp}

- **Preprocessing**
  - Preprocessor
  - Program Builder

- **Solver**\(1 \ldots n\)
  - Decision Heuristic
  - Assignment Atoms/Bodies
  - Conflict Resolution

- **Coordination**
  - SharedContext
    - Propositional Variables
    - Atoms
    - Bodies
    - Static Nogoods
    - Short Nogoods
  - Nogood Distributor

- **Parallel Context**
  - Threads\(S_1 S_2 \ldots S_n\)
  - Counter\(T W \ldots S\)
  - Queue\(P_1 P_2 \ldots P_n\)

- **Enumerator**

- **Recorded Nogoods**

- **Propagations**
  - Unit Propagation
  - Post Propagation

- **Logic Program**
  - Preprocessing
  - Program Builder

- **Logic**

- **Torsten Schaub (KRR@UP)**
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
  - Program Builder
- **Logic Program**
- **Solver 1...n**
- **Coordination**
  - Shared Context
    - Propositional Variables
    - Atoms
    - Bodies
    - Static Nogoods
    - Short Nogoods
  - Decision Heuristic
  - Assignment Atoms/Bodies
  - Conflict Resolution
  - Recorded Nogoods
  - Unit Propagation
  - Post Propagation
  - Propagation
- **Enumerator**
  - Threads: $S_1, S_2, \ldots, S_n$
  - Counter: $T, W, \ldots, S$
  - Queue: $P_1, P_2, \ldots, P_n$
- **Parallel Context**
  - Threads: $S_1, S_2, \ldots, S_n$
  - Counter: $T, W, \ldots, S$
  - Queue: $P_1, P_2, \ldots, P_n$
  - Shared Nogoods
- **Nogood Distributor**
- **Clasp**

Torsten Schaub (KRR@UP)  
Answer Set Solving in Practice  
February 18, 2019
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
  - Program Builder
  - Logic Program

- **Coordination**
  - SharedContext
    - Propositional Variables
    - Atoms
    - Bodies
    - Static Nogoods
    - Short Nogoods
  - Decision Heuristic
  - Assignment
    - Atoms/Bodies
  - Conflict Resolution
  - Recorded Nogoods

- **Parallel-solving**
  - Enumerator
    - Threads: $S_1, S_2, \ldots, S_n$
    - Counter: $T, W, \ldots, S$
    - Queue: $P_1, P_2, \ldots, P_n$
  - Nogood Distributor
  - Shared Nogoods
  - Preprocessing
    - Program Builder
  - Logic Program

- **Solver**
  - Decision Heuristic
  - Assignment
  - Recorded Nogoods
  - Propagation
    - Unit Propagation
    - Post Propagation
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Preprocessor
  - Program Builder
- **Logic Program**
- **Solver 1...n**
  - Conflict Resolution
  - Recorded Nogoods
  - Propagation
    - Unit Propagation
    - Post Propagation
  - Assignment Atoms/Bodies
- **Decision Heuristic**
- **Coordination**
  - SharedContext
    - Propositional Variables
    - Atoms
    - Bodies
    - Static Nogoods
    - Short Nogoods
  - Nogood Distributor
- **ParallelContext**
  - Threads: $S_1, S_2, ..., S_n$
  - Counter: $T, W, ..., S$
  - Queue: $P_1, P_2, ..., P_n$
  - Shared Nogoods
- **Logic Program Builder**
  - Preprocessor
clasp in context

- Compare clasp (2.0.5) to the multi-threaded SAT solvers
  - cryptominisat (2.9.2)
  - manySat (1.1)
  - miraxt (2009)
  - plingeling (587f)

  all run with four and eight threads in their default settings

- 160/300 benchmarks from crafted category at SAT’11
  - all solvable by ppfolio in 1000 seconds
  - crafted SAT benchmarks are closest to ASP benchmarks
**clasp in context**

![Graph comparing solving times of different tools](image)

- clasp-t1
- cryptominisat-2.9.2-t4
- miraxt-2009-t4
- plingeling-587-t4
- manysat-1.1-t4

**Solved instances** vs **Time in seconds** for various tools.
Outline

62 Potassco
63 gringo
64 clasp
   - Features
   - Parallel solving
   - Configuration
   - Disjunctive solving
65 clingo
66 clingcon
67 claspfolio
68 clavis
Using `clasp`

--help[=<n>],-h : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
  <arg>: <n {1..64}>[,<mode {compete|split}>]
  <n> : Number of threads to use in search
  <mode>: Run competition or splitting based search [compete]

--configuration=<arg> : Configure default configuration [frumpy]
  <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
    frumpy: Use conservative defaults
    jumpy : Use aggressive defaults
    handy : Use defaults geared towards large problems
    crafty: Use defaults geared towards crafted problems
    trendy: Use defaults geared towards industrial problems

"-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g : Print default portfolio and exit
Using clasp

--help[=<n>], -h : Print {1=basic|2=more|3=full} help and exit

--parallel-mode, -t <arg>: Run parallel search with given number of threads
  <arg>: <n {1..64}>[,<mode {compete|split}>]
  <n> : Number of threads to use in search
  <mode>: Run competition or splitting based search [compete]

--configuration=<arg> : Configure default configuration [frumpy]
  <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
    frumpy: Use conservative defaults
    jumpy : Use aggressive defaults
    handy : Use defaults geared towards large problems
    crafty: Use defaults geared towards crafted problems
    trendy: Use defaults geared towards industrial problems

  "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio, -g : Print default portfolio and exit
Using `clasp`

```
--help[=<n>],-h : Print {1=basic|2=more|3=full} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
  <arg>: <n {1..64}>,<mode {compete|split}>
  <n> : Number of threads to use in search
  <mode>: Run competition or splitting based search [compete]

--configuration=<arg> : Configure default configuration [frumpy]
  <arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}
    frumpy: Use conservative defaults
    jumpy : Use aggressive defaults
    handy : Use defaults geared towards large problems
    crafty: Use defaults geared towards crafted problems
    trendy: Use defaults geared towards industrial problems

  "-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g : Print default portfolio and exit
```
Using *clasp*

---

**--help[=<n>],--help[=<n>],-h** : Print {1=basic|2=more|3=full} help and exit

**--parallel-mode,-t <arg>: Run parallel search with given number of threads**

<arg>: <n {1..64}>[,<mode {compete|split}>]

<n> : Number of threads to use in search

<mode>: Run competition or splitting based search [compete]

**--configuration=<arg>** : Configure default configuration [frumpy]

<arg>: {frumpy|jumpy|handy|crafty|trendy|chatty}

frumpy: Use conservative defaults

jumpy : Use aggressive defaults

handy : Use defaults geared towards large problems

crafty: Use defaults geared towards crafted problems

trendy: Use defaults geared towards industrial problems

"-t 4": Use 4 competing threads initialized via the default portfolio

**--print-portfolio,-g** : Print default portfolio and exit
Using `clasp`

--help[=<n>],-h : Print \{1=basic\|2=more\|3=full\} help and exit

--parallel-mode,-t <arg>: Run parallel search with given number of threads
   <arg>: <n \{1..64\}>[,<mode \{compete\|split\}>]
   <n> : Number of threads to use in search
   <mode>: Run competition or splitting based search [compete]

--configuration=<arg> : Configure default configuration [frumpy]
   <arg>: \{frumpy\|jumpy\|handy\|crafty\|trendy\|chatty\}
   frumpy: Use conservative defaults
   jumpy : Use aggressive defaults
   handy : Use defaults geared towards large problems
   crafty: Use defaults geared towards crafted problems
   trendy: Use defaults geared towards industrial problems

"-t 4": Use 4 competing threads initialized via the default portfolio

--print-portfolio,-g : Print default portfolio and exit
Comparing configurations on queensA.lp

<table>
<thead>
<tr>
<th>n</th>
<th>frumpy</th>
<th>jumpy</th>
<th>handy</th>
<th>crafty</th>
<th>trendy</th>
<th>-t 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.063</td>
<td>0.023</td>
<td>3.416</td>
<td>0.030</td>
<td>1.805</td>
<td>0.061</td>
</tr>
<tr>
<td>100</td>
<td>20.364</td>
<td>0.099</td>
<td>7.891</td>
<td>0.136</td>
<td>7.321</td>
<td>0.121</td>
</tr>
<tr>
<td>150</td>
<td>60.000</td>
<td>0.212</td>
<td>14.522</td>
<td>0.271</td>
<td>19.883</td>
<td>0.347</td>
</tr>
<tr>
<td>200</td>
<td>60.000</td>
<td>0.415</td>
<td>15.026</td>
<td>0.667</td>
<td>32.476</td>
<td>0.753</td>
</tr>
<tr>
<td>500</td>
<td>60.000</td>
<td>3.199</td>
<td>60.000</td>
<td>7.471</td>
<td>60.000</td>
<td>6.104</td>
</tr>
</tbody>
</table>

(times in seconds, cut-off at 60 seconds)
### Comparing configurations on queensA.lp

<table>
<thead>
<tr>
<th>n</th>
<th>frumpy</th>
<th>jumpy</th>
<th>handy</th>
<th>crafty</th>
<th>trendy</th>
<th>-t 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.063</td>
<td>0.023</td>
<td>3.416</td>
<td>0.030</td>
<td>1.805</td>
<td>0.061</td>
</tr>
<tr>
<td>100</td>
<td>20.364</td>
<td>0.099</td>
<td>7.891</td>
<td>0.136</td>
<td>7.321</td>
<td>0.121</td>
</tr>
<tr>
<td>150</td>
<td>60.000</td>
<td>0.212</td>
<td>14.522</td>
<td>0.271</td>
<td>19.883</td>
<td>0.347</td>
</tr>
<tr>
<td>200</td>
<td>60.000</td>
<td>0.415</td>
<td>15.026</td>
<td>0.667</td>
<td>32.476</td>
<td>0.753</td>
</tr>
<tr>
<td>500</td>
<td>60.000</td>
<td>3.199</td>
<td>60.000</td>
<td>7.471</td>
<td>60.000</td>
<td>6.104</td>
</tr>
</tbody>
</table>

(times in seconds, cut-off at 60 seconds)
Comparing configurations on queensA.lp

<table>
<thead>
<tr>
<th>n</th>
<th>frumpy</th>
<th>jumpy</th>
<th>handy</th>
<th>crafty</th>
<th>trendy</th>
<th>-t 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.063</td>
<td>0.023</td>
<td>3.416</td>
<td>0.030</td>
<td>1.805</td>
<td>0.061</td>
</tr>
<tr>
<td>100</td>
<td>20.364</td>
<td>0.099</td>
<td>7.891</td>
<td>0.136</td>
<td>7.321</td>
<td>0.121</td>
</tr>
<tr>
<td>150</td>
<td>60.000</td>
<td>0.212</td>
<td>14.522</td>
<td>0.271</td>
<td>19.883</td>
<td>0.347</td>
</tr>
<tr>
<td>200</td>
<td>60.000</td>
<td>0.415</td>
<td>15.026</td>
<td>0.667</td>
<td>32.476</td>
<td>0.753</td>
</tr>
<tr>
<td>500</td>
<td>60.000</td>
<td>3.199</td>
<td>60.000</td>
<td>7.471</td>
<td>60.000</td>
<td>6.104</td>
</tr>
</tbody>
</table>

(times in seconds, cut-off at 60 seconds)
Comparing configurations on queensA.lp

<table>
<thead>
<tr>
<th>n</th>
<th>frumpy</th>
<th>jumpy</th>
<th>handy</th>
<th>crafty</th>
<th>trendy</th>
<th>-t 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.063</td>
<td>0.023</td>
<td>3.416</td>
<td>0.030</td>
<td>1.805</td>
<td>0.061</td>
</tr>
<tr>
<td>100</td>
<td>20.364</td>
<td>0.099</td>
<td>7.891</td>
<td>0.136</td>
<td>7.321</td>
<td>0.121</td>
</tr>
<tr>
<td>150</td>
<td>60.000</td>
<td>0.212</td>
<td>14.522</td>
<td>0.271</td>
<td>19.883</td>
<td>0.347</td>
</tr>
<tr>
<td>200</td>
<td>60.000</td>
<td>0.415</td>
<td>15.026</td>
<td>0.667</td>
<td>32.476</td>
<td>0.753</td>
</tr>
<tr>
<td>500</td>
<td>60.000</td>
<td>3.199</td>
<td>60.000</td>
<td>7.471</td>
<td>60.000</td>
<td>6.104</td>
</tr>
</tbody>
</table>

(times in seconds, cut-off at 60 seconds)
Comparing configurations
on queensA.lp

<table>
<thead>
<tr>
<th>n</th>
<th>frumpy</th>
<th>jumpy</th>
<th>handy</th>
<th>crafty</th>
<th>trendy</th>
<th>-t 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.063</td>
<td>0.023</td>
<td>3.416</td>
<td>0.030</td>
<td>1.805</td>
<td>0.061</td>
</tr>
<tr>
<td>100</td>
<td>20.364</td>
<td>0.099</td>
<td>7.891</td>
<td>0.136</td>
<td>7.321</td>
<td>0.121</td>
</tr>
<tr>
<td>150</td>
<td>60.000</td>
<td>0.212</td>
<td>14.522</td>
<td>0.271</td>
<td>19.883</td>
<td>0.347</td>
</tr>
<tr>
<td>200</td>
<td>60.000</td>
<td>0.415</td>
<td>15.026</td>
<td>0.667</td>
<td>32.476</td>
<td>0.753</td>
</tr>
<tr>
<td>500</td>
<td>60.000</td>
<td>3.199</td>
<td>60.000</td>
<td>7.471</td>
<td>60.000</td>
<td>6.104</td>
</tr>
</tbody>
</table>

(times in seconds, cut-off at 60 seconds)
clasp’s default portfolio for parallel solving via clasp --print-portfolio

[solver.0]: --heuristic=Vsids,92 --restarts=L,60 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del-glue=2 --update-lbd=1 --strengthen=recursive --otfs=2 --counter-restarts=3 --contraction=250

[solver.1]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=basic,50,0 --del-init=3.0,500,19500 --del-grow=1.1,20.0

[solver.2]: --heuristic=Berkmin --restarts=x,100,1.5 --deletion=basic,75,0 --del-init=10.0,1000,9000 --del-grow=1.1,20.0

[solver.3]: --restarts=x,128,1.5 --deletion=basic,75,0 --del-init=3.0,1000,9000 --del-grow=1.1,20.0

[solver.4]: --heuristic=Vsids --restarts=L,100 --deletion=basic,75,2 --del-max=3.0,100,20000 --del-grow=1.1,20.0

[solver.5]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=sort,50,2 --del-max=2000000 --del-init=20.0,1000,1

[solver.6]: --heuristic=Berkmin,512 --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-grow=1.1,20.0

[solver.7]: --heuristic=Vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250

[solver.8]: --heuristic=Vsids --restarts=L,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2

[solver.9]: --heuristic=Berkmin,512 --restarts=F,16000 --lookahead=atom,50 --conv=auto --opt-heu=3

[solver.10]: --heuristic=Vmtf --strengthen=none --conv=auto --opt-heu=3

[solver.11]: --heuristic=Vsids --strengthen=recursive --restarts=x,100,1.3 --del-init=3.0,800,9200

[solver.12]: --heuristic=Vsids --restarts=L,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3

[solver.13]: --heuristic=Berkmin,512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0

[solver.14]: --no-lookback --heuristic=Unit --lookahead=atom --deletion=no --restarts=no

- **clasp**'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `-t 4` uses four threads with crafty, trendy, frumpy, and jumpy
**clasp's default portfolio for parallel solving**

via `clasp --print-portfolio`

```
[solver.0]: --heuristic=Vsids,92 --restarts=L,60 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del-glue=2 --update-lbd=3 --strengthen=recursive --otfs=2 --reverse-arcs=1 --local-restarts --save-progress=0 --contraction=0 --lookahead=atom --deletion=no --restarts=no
[solver.1]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=basic,50,0 --del-init=3.0,500,19500 --del-grow=1.1,20.0 --del-cfl=4 --counter-restarts=7 --counter-bump=1023 --reverse-arcs=1 --loops=common --opt-heu=2
[solver.2]: --heuristic=Berkmin --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,1000,20000 --del-grow=1.1,20000 --del-cfl=4 --strengthen=recursive --init-w=2 --lookahead=atom --deletion=no --restarts=no
[solver.3]: --restarts=x,128,1.5 --deletion=basic,75,0 --del-init=10.0,1000,9000 --del-grow=1.1,20.0 --del-cfl=4 --counter-restarts=7 --counter-bump=1023 --reverse-arcs=2 --contraction=250 --loops=common --opt-heu=2
[solver.4]: --heuristic=Vsids --restarts=L,100 --deletion=basic,75,2 --del-init=3.0,1000,20000 --del-grow=1.1,20000 --del-cfl=4 --counter-restarts=7 --counter-bump=1023 --reverse-arcs=2 --loops=common --opt-heu=3 --opt-strat=bb,1
[solver.5]: --heuristic=Vsids --restarts=D,100,0.7 --deletion=sort,50,2 --del-init=3.0,100,20000 --del-grow=1.1,20.0 --del-cfl=4 --counter-restarts=7 --counter-bump=1023 --reverse-arcs=2 --loops=common --opt-heu=3 --opt-strat=bb,1
[solver.6]: --heuristic=Berkmin,512 --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-grow=1.1,20000 --del-cfl=4 --counter-restarts=7 --counter-bump=1023 --reverse-arcs=2 --loops=common --opt-heu=3 --opt-strat=bb,2
```

- **clasp's portfolio is fully customizable**
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `-t 4` uses four threads with *crafty*, *trendy*, *frumpy*, and *jumpy*
### clasp's Default Portfolio for Parallel Solving

**clasp**'s default portfolio for parallel solving via `clasp --print-portfolio`

<table>
<thead>
<tr>
<th>Solver</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>solver.0</td>
<td>--heuristic=Vsids,92 --restarts=L,60 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del-glue=2 --update-lbd=3 --strengthen=recursive --otfs=2 --deletion=ipSort,75,2 --del-init=20.0,1000,19000</td>
</tr>
<tr>
<td>solver.1</td>
<td>--heuristic=Vsids --restarts=D,100,0.7 --deletion=basic,50,0 --del-init=3.0,500,19500 --del-grow=1.1</td>
</tr>
<tr>
<td>solver.2</td>
<td>--heuristic=Berkmin --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-max=400000 --contraction=250 --loops=common --save-p=180 --del-grow=1.1 --strengthen=local --sign-def=4 --restart-on-model --opt-heu=2</td>
</tr>
<tr>
<td>solver.3</td>
<td>--restarts=x,128,1.5 --deletion=basic,75,0 --del-init=10.0,1000,9000 --del-grow=1.1,20.0 --del-cfl=2 --counter-restarts=3 --counter-bump=1023 --reverse-arcs=2 --contraction=250 --heuristic=domain --dom-mod=4,8 --opt-strat=bb,1</td>
</tr>
<tr>
<td>solver.4</td>
<td>--heuristic=Vsids --restarts=L,100 --deletion=basic,50,0 --del-max=2000000 --del-estimate=1 --del-glue=2 --update-lbd=3 --strengthen=recursive --otfs=2 --deletion=ipSort,75,2 --del-init=20.0,1000,19000</td>
</tr>
<tr>
<td>solver.5</td>
<td>--heuristic=Berkmin,512 --restarts=x,100,1.5 --deletion=basic,75 --del-init=3.0,200,40000 --del-grow=1.1,20.0 --del-cfl=2 --counter-restarts=3 --counter-bump=1023 --reverse-arcs=2 --contraction=250 --heuristic=domain --dom-mod=4,8 --opt-strat=bb,1</td>
</tr>
<tr>
<td>solver.6</td>
<td>--heuristic=Berkmin,512 --restarts=F,16000 --lookahead=atom,50</td>
</tr>
<tr>
<td>solver.7</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.3 --del-init=3.0,800,9200</td>
</tr>
<tr>
<td>solver.8</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.5,15 --contraction=0 --contr=0</td>
</tr>
<tr>
<td>solver.9</td>
<td>--heuristic=Vsids --strength=recursive --restarts=F,16000 --lookahead=atom,50</td>
</tr>
<tr>
<td>solver.10</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.3 --del-init=3.0,800,9200</td>
</tr>
<tr>
<td>solver.11</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.5,15 --contraction=0 --contr=0</td>
</tr>
<tr>
<td>solver.12</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.5,15 --contraction=0 --contr=0</td>
</tr>
<tr>
<td>solver.13</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.5,6 --local-restarts --init-w=2 --opt-heu=3</td>
</tr>
<tr>
<td>solver.14</td>
<td>--heuristic=Vsids --strength=recursive --restarts=x,100,1.5,6 --local-restarts --init-w=2 --opt-heu=3</td>
</tr>
</tbody>
</table>

- **clasp**'s portfolio is fully customizable
- Configurations are assigned in a round-robin fashion to threads during parallel solving
- `-t 4` uses four threads with crafty, trendy, frumpy, and jumpy
Outline

62 Potassco

63 gringo

64 clasp
  - Features
  - Parallel solving
  - Configuration
  - Disjunctive solving

65 clingo

66 clingcon

67 claspfolio

68 clavis
clasp

- *clasp* is a multi-threaded solver for disjunctive logic programs
- aiming at an equitable interplay between “generating” and “testing” solver units
- allowing for a bidirectional dynamic information exchange between solver units for orthogonal tasks
clasp is a multi-threaded solver for disjunctive logic programs

- aiming at an equitable interplay between “generating” and “testing” solver units
- allowing for a bidirectional dynamic information exchange between solver units for orthogonal tasks
Clingo species

Before

- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After

- Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

Clingo series 4 = ASP + Control

Multi-shot ASP solving deals with continuously changing programs

See Multi-shot ASP Solving for details
Clingo species

Before

- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After

- Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

- Clingo series 4 = ASP + Control
- Multi-shot ASP solving deals with continuously changing programs
- See Multi-shot ASP Solving for details
Clingo species

Before

- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After

- Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

Clingo series 4 = ASP + Control

Multi-shot ASP solving deals with continuously changing programs

See Multi-shot ASP Solving for details
Clingo species

Before
- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After
- Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

Clingo series 4 = ASP + Control
- Multi-shot ASP solving deals with continuously changing programs
- See Multi-shot ASP Solving for details
Clingo species

**Before**
- **Clingo** — easy solving
- **iClingo** — incremental solving
- **oClingo** — reactive solving

**After**
- **Clingo** — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

**Clingo series 4 = ASP + Control**

*Multi-shot ASP solving deals with continuously changing programs*

*See *Multi-shot ASP Solving* for details*
Clingo species

**Before**
- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

**After**
- Clingo — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

Clingo series 4 = ASP + Control

Multi-shot ASP solving deals with continuously changing programs

See Multi-shot ASP Solving for details
Clingo species

Before
- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After
- Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

Clingo series 4 = ASP + Control

Multi-shot ASP solving deals with continuously changing programs

See Multi-shot ASP Solving for details
Clingo species

**Before**

- *Clingo* — easy solving
- *iClingo* — incremental solving
- *oClingo* — reactive solving

**After**

- *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

*Clingo* series 4 = ASP + Control

Multi-shot ASP solving deals with continuously changing programs

See *Multi-shot ASP Solving* for details
Before

- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After

- Clingo — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

Clingo species

\[ \text{Clingo} = \text{gringo} \mid \text{clasp} \]

\[ \text{Clingo} = (\text{gringo}^* \mid \text{clasp}^*)^* \]

Clingo series 4 = ASP + Control

Multi-shot ASP solving deals with continuously changing programs

See Multi-shot ASP Solving for details
Clingo species

Before
- Clingo — easy solving
- iClingo — incremental solving
- oClingo — reactive solving

After
- Clingo — easy solving
  + incremental solving
  + reactive solving
  + complex solving

Clingo series 4 = ASP + Control
- Multi-shot ASP solving deals with continuously changing programs
- See Multi-shot ASP Solving for details
Clingo species

**Before**

- *Clingo* — easy solving
- *iClingo* — incremental solving
- *oClingo* — reactive solving

**After**

- *Clingo* — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

*Clingo series 4 = ASP + Control*

Multi-shot ASP solving deals with continuously changing programs

See *Multi-shot ASP Solving* for details
Clingo species

**Before**
- \textit{Clingo} — easy solving
- \textit{iClingo} — incremental solving
- \textit{oClingo} — reactive solving

**After**
- \textit{Clingo} — easy solving
  - + incremental solving
  - + reactive solving
  - + complex solving

\textit{Clingo series 4} = ASP + Control

- Multi-shot ASP solving deals with continuously changing programs
- See \textit{Multi-shot ASP Solving} for details
Hybrid grounding and solving
Solving in hybrid domains, like Bio-Informatics

Basic architecture of **clingcon**:
Pouring Water into Buckets on a Scale

time(0..t).
$domain(0..500).
bucket(a).
volume(a,0) $== 0.
bucket(b).
volume(b,0) $== 100.

1 $\{$ pour(B,T) : bucket(B) $\}$ 1 :- time(T), T < t.

1 $\leq$ amount(B,T) :- pour(B,T), T < t.
amount(B,T) $\leq$ 30 :- pour(B,T), T < t.
amount(B,T) $==$ 0 :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $==$ volume(B,T) $+$ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $<$ volume(B,T), bucket(B;C), time(T).
up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t). \quad domain(0..500).
bucket(a). \quad volume(a,0) \equiv 0.
bucket(b). \quad volume(b,0) \equiv 100.

1 \{ pour(B,T) : bucket(B) \} 1 :- time(T), T < t.

1 \leq amount(B,T) :- pour(B,T), T < t.

amount(B,T) \leq 30 :- pour(B,T), T < t.

amount(B,T) \equiv 0 :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) \equiv volume(B,T) + amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) <$\leq$ volume(B,T), bucket(B;C), time(T).

up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t).
$domain(0..500).
bucket(a).
volume(a,0) $== 0.
bucket(b).
volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

1 $\leq$ amount(B,T) :- pour(B,T), T < t.
amount(B,T) $\leq$ 30 :- pour(B,T), T < t.
amount(B,T) $== 0$ :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) \star amount(B,T)$ :- bucket(B), time(T), T < t.
down(B,T) :- volume(C,T) $<$ volume(B,T), bucket(B;C), time(T).
    up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t).
\$domain(0..500).
bucket(a).
\$volume(a,0) == 0.
bucket(b).
\$volume(b,0) == 100.

1 \{ pour(B,T) : bucket(B) \} 1 :- time(T), T < t.

1 \$<= amount(B,T) :- pour(B,T), T < t.
amount(B,T) \$<= 30 :- pour(B,T), T < t.
amount(B,T) \$== 0 :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) \$== volume(B,T) \$+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) \$< volume(B,T), bucket(B;C), time(T).
up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t). $domain(0..500).
bucket(a). volume(a,0) $== 0.
bucket(b). volume(b,0) $== 100.

1 \{ pour(B,T) : bucket(B) \} 1 :- time(T), T < t.

:- pour(B,T), T < t, not (1 $<= amount(B,T)).

amount(B,T) $<= 30 :- pour(B,T), T < t.
amount(B,T) $== 0 :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t).
$\text{domain}(0..500)$.
bucket(a).
volume(a,0) $==$ 0.
bucket(b).
volume(b,0) $==$ 100.

$1 \{ \text{pour}(B,T) : \text{bucket}(B) \} 1 :- \text{time}(T), T < t.$

$:- \text{pour}(B,T), T < t, 1 > \text{amount}(B,T).$

\text{amount}(B,T) $\leq 30$  $:- \text{pour}(B,T), T < t.$

\text{amount}(B,T) $== 0$  $:- \text{not pour}(B,T), \text{bucket}(B), \text{time}(T), T < t.$

\text{volume}(B,T+1) $== \text{volume}(B,T) + \text{amount}(B,T)$  $:- \text{bucket}(B), \text{time}(T), T < t.$

down(B,T) :- \text{volume}(C,T) $<$ \text{volume}(B,T), \text{bucket}(B;C), \text{time}(T).

up(B,T) :- \text{not down}(B,T), \text{bucket}(B), \text{time}(T).

$:- \text{up}(a,t).$
Pouring Water into Buckets on a Scale

time(0..t).
$domain(0..500).
bucket(a).
volume(a,0) $== 0.
bucket(b).
volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

:- pour(B,T), T < t, 1 $> amount(B,T).
:- pour(B,T), T < t, amount(B,T) $> 30.
amount(B,T) $== 0 :- not pour(B,T), bucket(B), time(T), T < t.

volume(B,T+1) $== volume(B,T) $+ amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t).
$domain(0..500).
bucket(a).
volume(a,0) == 0.
bucket(b).
volume(b,0) == 100.

\[ 1 \{ \text{pour}(B,T) : \text{bucket}(B) \} 1 : - \text{time}(T), T < t. \]

\[ : - \text{pour}(B,T), T < t, 1 > \text{amount}(B,T). \]
\[ : - \text{pour}(B,T), T < t, \text{amount}(B,T) > 30. \]
\[ : - \text{not pour}(B,T), \text{bucket}(B), \text{time}(T), T < t, \text{amount}(B,T) != 0. \]

volume(B,T+1) == volume(B,T) + amount(B,T) :- bucket(B), time(T), T < t.

down(B,T) :- volume(C,T) < volume(B,T), bucket(B;C), time(T).
up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

time(0..t). $\textit{domain}(0..500).
bucket(a). volume(a,0) $== 0.
bucket(b). volume(b,0) $== 100.

1 { pour(B,T) : bucket(B) } 1 :- time(T), T < t.

:- pour(B,T), T < t, 1 $> amount(B,T).
:- pour(B,T), T < t, amount(B,T) $> 30.
:- not pour(B,T), bucket(B), time(T), T < t, amount(B,T) $!= 0.

:- bucket(B), time(T), T < t, volume(B,T+1) $!= volume(B,T)+amount(B,T).

down(B,T) :- volume(C,T) $< volume(B,T), bucket(B;C), time(T).
up(B,T) :- not down(B,T), bucket(B), time(T).

:- up(a,t).
Pouring Water into Buckets on a Scale

$\text{clingcon} --\text{const t=4 balance.lp} --\text{text}$

time(0). ... time(4).
bucket(a).
bucket(b).

1 \{ pour(b,0), pour(a,0) \} 1.

:- pour(a,0), 1 $> amount(a,0).
:- pour(b,0), 1 $> amount(b,0).

:- pour(a,0), amount(a,0) $> 30.
:- pour(b,0), amount(b,0) $> 30.

:- not pour(a,0), amount(a,0) $!= 0.
:- not pour(b,0), amount(b,0) $!= 0.

:- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
:- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

:= up(a,4).
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --text

time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1. 1 { pour(b,3), pour(a,3) } 1.

:- pour(a,0), 1 > amount(a,0). :- pour(a,3), 1 > amount(a,3).
:- pour(b,0), 1 > amount(b,0). :- pour(b,3), 1 > amount(b,3).

:- not pour(a,0), amount(a,0) != 0. :- not pour(a,3), amount(a,3) != 0.
:- not pour(b,0), amount(b,0) != 0. :- not pour(b,3), amount(b,3) != 0.

:- volume(a,1) != (volume(a,0) + amount(a,0)). :- volume(a,4) != (volume(a,3) + amount(a,3)).
:- volume(b,1) != (volume(b,0) + amount(b,0)). :- volume(b,4) != (volume(b,3) + amount(b,3)).

down(a,0) :- volume(a,0) < volume(a,0). down(a,4) :- volume(a,4) < volume(a,4).
down(a,0) :- volume(b,0) < volume(a,0). down(a,4) :- volume(b,4) < volume(a,4).
down(b,0) :- volume(a,0) < volume(b,0). down(b,4) :- volume(a,4) < volume(b,4).
down(b,0) :- volume(b,0) < volume(b,0). down(b,4) :- volume(b,4) < volume(b,4).

up(a,0) :- not down(a,0). up(a,4) :- not down(a,4).
up(b,0) :- not down(b,0). up(b,4) :- not down(b,4).

:- up(a,4).
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --text

time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1.

:- pour(a,0), 1 $> amount(a,0).
:- pour(b,0), 1 $> amount(b,0).

:- pour(a,0), amount(a,0) $> 30.
:- pour(b,0), amount(b,0) $> 30.

:- not pour(a,0), amount(a,0) $!= 0.
:- not pour(b,0), amount(b,0) $!= 0.

:- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
:- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

:- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
:- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

:- up(a,4).
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --text

time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1. ... 1 { pour(b,3), pour(a,3) } 1.

:- pour(a,0), 1 $> amount(a,0).
:- pour(b,0), 1 $> amount(b,0).

:- pour(a,0), amount(a,0) $> 30.
:- pour(b,0), amount(b,0) $> 30.

:- not pour(a,0), amount(a,0) $!= 0.
:- not pour(b,0), amount(b,0) $!= 0.

:- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
:- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

:- up(a,4).
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --text

time(0). ... time(4).
bucket(a).
bucket(b).

1 { pour(b,0), pour(a,0) } 1. ... 1 { pour(b,3), pour(a,3) } 1.

:- pour(a,0), 1 $> amount(a,0).
:- pour(b,0), 1 $> amount(b,0).

:- pour(a,0), amount(a,0) $> 30.
:- pour(b,0), amount(b,0) $> 30.

:- not pour(a,0), amount(a,0) $!= 0.
:- not pour(b,0), amount(b,0) $!= 0.

:- volume(a,1) $!= (volume(a,0) $+ amount(a,0)).
:- volume(b,1) $!= (volume(b,0) $+ amount(b,0)).

:- volume(a,4) $!= (volume(a,3) $+ amount(a,3)).
:- volume(b,4) $!= (volume(b,3) $+ amount(b,3)).

down(a,0) :- volume(a,0) $< volume(a,0).
down(a,0) :- volume(b,0) $< volume(a,0).
down(b,0) :- volume(a,0) $< volume(b,0).
down(b,0) :- volume(b,0) $< volume(b,0).

up(a,0) :- not down(a,0).
up(b,0) :- not down(b,0).

:- up(a,4).
Pouring Water into Buckets on a Scale

$ \text{clingcon} --\text{const} \ t=4 \ \text{balance.lp} \ 0$

Answer: 1
pour(a,0) pour(a,1) pour(a,2) pour(a,3)

amount(a,0)=[11..30] amount(b,0)=0 1 $> \ amount(b,0) \ amount(a,0) \ != 0$
amount(a,1)=[11..30] amount(b,1)=0 1 $> \ amount(b,1) \ amount(a,1) \ != 0$
amount(a,2)=[11..30] amount(b,2)=0 1 $> \ amount(b,2) \ amount(a,2) \ != 0$
amount(a,3)=[11..30] amount(b,3)=0 1 $> \ amount(b,3) \ amount(a,3) \ != 0$

volume(a,0)=0 volume(b,0)=100 volume(a,0) $< \ volume(b,0)$
volume(a,1)=[11..30] volume(b,1)=100 volume(a,1) $< \ volume(b,1)$
volume(a,2)=[41..60] volume(b,2)=100 volume(a,2) $< \ volume(b,2)$
volume(a,3)=[71..90] volume(b,3)=100 volume(a,3) $< \ volume(b,3)$
volume(a,4)=[101..120] volume(b,4)=100 volume(b,4) $< \ volume(a,4)$

SATISFIABLE

Models : 1
Time : 0.000
$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)  pour(a,1)  pour(a,2)  pour(a,3)

amount(a,0)=[11..30]  amount(b,0)=0  1 $> amount(b,0)  amount(a,0) $!= 0
amount(a,1)=[11..30]  amount(b,1)=0  1 $> amount(b,1)  amount(a,1) $!= 0
amount(a,2)=[11..30]  amount(b,2)=0  1 $> amount(b,2)  amount(a,2) $!= 0
amount(a,3)=[11..30]  amount(b,3)=0  1 $> amount(b,3)  amount(a,3) $!= 0

volume(a,0)=0  volume(b,0)=100  volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]  volume(b,1)=100  volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]  volume(b,2)=100  volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]  volume(b,3)=100  volume(a,3) $< volume(b,3)
volume(a,4)=[101..120]  volume(b,4)=100  volume(b,4) $< volume(a,4)

SATISFIABLE

Models : 1
Time : 0.000
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp 0$

Answer: 1
pour(a,0)  pour(a,1)  pour(a,2)  pour(a,3)

amount(a,0)=[11..30]  amount(b,0)=0
amount(a,1)=[11..30]  amount(b,1)=0
amount(a,2)=[11..30]  amount(b,2)=0
amount(a,3)=[11..30]  amount(b,3)=0

1 $>\ amount(b,0)  amount(a,0) \neq 0$
1 $>\ amount(b,1)  amount(a,1) \neq 0$
1 $>\ amount(b,2)  amount(a,2) \neq 0$
1 $>\ amount(b,3)  amount(a,3) \neq 0$

volume(a,0)=0  volume(b,0)=100
volume(a,1)=11..30  volume(b,1)=100
volume(a,2)=41..60  volume(b,2)=100
volume(a,3)=71..90  volume(b,3)=100
volume(a,4)=101..120  volume(b,4)=100

volume(a,0) $< volume(b,0)$
volume(a,1) $< volume(b,1)$
volume(a,2) $< volume(b,2)$
volume(a,3) $< volume(b,3)$
volume(b,4) $< volume(a,4)$

SATISFIABLE

Models : 1
Time : 0.000
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp 0$

Answer: 1
pour(a,0) pour(a,1) pour(a,2) pour(a,3)

amount(a,0)=[11..30] amount(b,0)=0
amount(a,1)=[11..30] amount(b,1)=0
amount(a,2)=[11..30] amount(b,2)=0
amount(a,3)=[11..30] amount(b,3)=0

1 $> amount(b,0) amount(a,0) $!= 0
1 $> amount(b,1) amount(a,1) $!= 0
1 $> amount(b,2) amount(a,2) $!= 0
1 $> amount(b,3) amount(a,3) $!= 0

volume(a,0)=0 volume(b,0)=100
volume(a,1)=11..30 volume(b,1)=100
volume(a,2)=41..60 volume(b,2)=100
volume(a,3)=71..90 volume(b,3)=100
volume(a,4)=101..120 volume(b,4)=100

volume(a,0) $< volume(b,0)$
volume(a,1) $< volume(b,1)$
volume(a,2) $< volume(b,2)$
volume(a,3) $< volume(b,3)$
volume(b,4) $< volume(a,4)$

SATISFIABLE

Models : 1
Time : 0.000
Pouring Water into Buckets on a Scale

$\text{clingcon} --\text{const} \ t=4 \ \text{balance.lp} \ 0$

Answer: 1

pour(a,0) pour(a,1) pour(a,2) pour(a,3)

amount(a,0)=\[11..30\] amount(b,0)=0 1 $>$ amount(b,0) amount(a,0) $!=$ 0
amount(a,1)=\[11..30\] amount(b,1)=0 1 $>$ amount(b,1) amount(a,1) $!=$ 0
amount(a,2)=\[11..30\] amount(b,2)=0 1 $>$ amount(b,2) amount(a,2) $!=$ 0
amount(a,3)=\[11..30\] amount(b,3)=0 1 $>$ amount(b,3) amount(a,3) $!=$ 0

volume(a,0)=0 volume(b,0)=100 volume(a,0) $<$ volume(b,0)
volume(a,1)=\[11..30\] volume(b,1)=100 volume(a,1) $<$ volume(b,1)
volume(a,2)=\[41..60\] volume(b,2)=100 volume(a,2) $<$ volume(b,2)
volume(a,3)=\[71..90\] volume(b,3)=100 volume(a,3) $<$ volume(b,3)
volume(a,4)=\[101..120\] volume(b,4)=100 volume(b,4) $<$ volume(a,4)

SATISFIABLE

Models : 1
Time : 0.000
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp 0

Answer: 1
pour(a,0)  pour(a,1)  pour(a,2)  pour(a,3)

amount(a,0)=[11..30]  amount(b,0)=0  1 $> amount(b,0)  amount(a,0) $!= 0
amount(a,1)=[11..30]  amount(b,1)=0  1 $> amount(b,1)  amount(a,1) $!= 0
amount(a,2)=[11..30]  amount(b,2)=0  1 $> amount(b,2)  amount(a,2) $!= 0
amount(a,3)=[11..30]  amount(b,3)=0  1 $> amount(b,3)  amount(a,3) $!= 0

volume(a,0)=0  volume(b,0)=100  volume(a,0) $< volume(b,0)
volume(a,1)=[11..30]  volume(b,1)=100  volume(a,1) $< volume(b,1)
volume(a,2)=[41..60]  volume(b,2)=100  volume(a,2) $< volume(b,2)
volume(a,3)=[71..90]  volume(b,3)=100  volume(a,3) $< volume(b,3)
volume(a,4)=[101..120]  volume(b,4)=100  volume(b,4) $< volume(a,4)

SATISFIABLE

Models : 1
Time      : 0.000
Pouring Water into Buckets on a Scale

$ \text{clingcon} \ --\text{const t=4 balance.lp} \ --\text{csp-num-as=1}$

Answer: 1
pour(a,0)  pour(a,1)  pour(a,2)  pour(a,3)

<table>
<thead>
<tr>
<th>amount(a,0)</th>
<th>amount(b,0)</th>
<th>1 $\geq$ amount(b,0)</th>
<th>amount(a,0) $\neq$ 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>volume(a,0)</th>
<th>volume(b,0)</th>
<th>volume(a,0) $\leq$ volume(b,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

SATISFIABLE

Models : 1+
Time : 0.000
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0) pour(a,1) pour(a,2) pour(a,3)

amount(a,0)=11 amount(b,0)=0 1 $> amount(b,0) amount(a,0) $!= 0
amount(a,1)=30 amount(b,1)=0 1 $> amount(b,1) amount(a,1) $!= 0
amount(a,2)=30 amount(b,2)=0 1 $> amount(b,2) amount(a,2) $!= 0
amount(a,3)=30 amount(b,3)=0 1 $> amount(b,3) amount(a,3) $!= 0

volume(a,0)=0 volume(b,0)=100 volume(a,0) $< volume(b,0)
volume(a,1)=11 volume(b,1)=100 volume(a,1) $< volume(b,1)
volume(a,2)=41 volume(b,2)=100 volume(a,2) $< volume(b,2)
volume(a,3)=71 volume(b,3)=100 volume(a,3) $< volume(b,3)
volume(a,4)=101 volume(b,4)=100 volume(b,4) $< volume(a,4)

SATISFIABLE

Models : 1+
Time : 0.000
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0)  pour(a,1)  pour(a,2)  pour(a,3)

amount(a,0)=11  amount(b,0)=0  1 $> amount(b,0)  amount(a,0) $!= 0
amount(a,1)=30  amount(b,1)=0  1 $> amount(b,1)  amount(a,1) $!= 0
amount(a,2)=30  amount(b,2)=0  1 $> amount(b,2)  amount(a,2) $!= 0
amount(a,3)=30  amount(b,3)=0  1 $> amount(b,3)  amount(a,3) $!= 0

volume(a,0)=0  volume(b,0)=100  volume(a,0) $< volume(b,0)
volume(a,1)=11  volume(b,1)=100  volume(a,1) $< volume(b,1)
volume(a,2)=41  volume(b,2)=100  volume(a,2) $< volume(b,2)
volume(a,3)=71  volume(b,3)=100  volume(a,3) $< volume(b,3)
volume(a,4)=101  volume(b,4)=100  volume(b,4) $< volume(a,4)

SATISFIABLE

Models : 1+
Time : 0.000
Pouring Water into Buckets on a Scale

$ clingcon --const t=4 balance.lp --csp-num-as=1

Answer: 1
pour(a,0) pour(a,1) pour(a,2) pour(a,3)

amount(a,0)=11 amount(b,0)=0 1 $> amount(b,0) amount(a,0) $!= 0
amount(a,1)=30 amount(b,1)=0 1 $> amount(b,1) amount(a,1) $!= 0
amount(a,2)=30 amount(b,2)=0 1 $> amount(b,2) amount(a,2) $!= 0
amount(a,3)=30 amount(b,3)=0 1 $> amount(b,3) amount(a,3) $!= 0

volume(a,0)=0 volume(b,0)=100 volume(a,0) $< volume(b,0)
volume(a,1)=11 volume(b,1)=100 volume(a,1) $< volume(b,1)
volume(a,2)=41 volume(b,2)=100 volume(a,2) $< volume(b,2)
volume(a,3)=71 volume(b,3)=100 volume(a,3) $< volume(b,3)
volume(a,4)=101 volume(b,4)=100 volume(b,4) $< volume(a,4)

SATISFIABLE

Models : 1+
Time : 0.000
Outline

62 Potassco
63 gringo
64 clasp
65 clingo
66 clingcon
67 claspfolio
68 clavis
Automatic selection of some *clasp* configuration among several predefined ones via (learned) classifiers

Basic architecture of *claspfolio*: 

```
gringo → clasp → Prediction → clasp

Models
```
Instance Feature Clusters (after PCA)
Solving with \textit{clasp} (as usual)

\$ clasp queens500 --quiet \\
clasp version 2.0.2 \\
Reading from queens500 \\
Solving... \\
SATISFIABLE \\
Models : 1+ \\
Time : 11.445s (Solving: 10.58s 1st Model: 10.55s Unsat: 0.00s) \\
CPU Time : 11.410s
Solving with *clasp* (as usual)

$ clasp queens500 --quiet

clap version 2.0.2
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time : 11.445s (Solving: 10.58s 1st Model: 10.55s Unsat: 0.00s)
CPU Time : 11.410s
$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time : 4.780s
Solving with claspfolio

$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time : 4.780s
Solving with `claspfolio`

```bash
$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time    : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time: 4.780s
```
Solving with `claspfolio`

$ claspfolio queens500 --quiet

PRESOLVING
Reading from queens500
Solving...
claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time : 4.785s (Solving: 3.96s 1st Model: 3.92s Unsat: 0.00s)
CPU Time : 4.780s
Feature-extraction with *claspfolio*

```bash
$ claspfolio --features queens500

PRESOLVING
Reading from queens500
Solving...
UNKNOWN
Features : 84998,3994,0,250000,1.020,62.594,63.844,21.281,84998, \ 
3994,100,250000,1.020,62.594,63.844,21.281,84998,3994,250,250000, \ 
1.020,62.594,63.844,21.281,84998,3994,475,250000,1.020,62.594, \ 
63.844,21.281,757989,757989,0,510983,506992,3990,1,0,127.066,9983, \ 
1023958,502993,1994,518971,1,0,0,254994,0,3990,0.100,0.000,99.900, \ 
0,270303,812,4,0,812,2223,2223,262,262,2.738,2.738,0.000,812,812, \ 
2270.982,0,0.000

$ claspfolio --list-features

dmaxLearnt,Constraints,LearntConstraints,FreeVars,Vars/FreeVars, ...
```
Feature-extraction with claspfolio

$ claspfolio --features queens500

PRESOLVING
Reading from queens500
Solving...
UNKNOWN
Features : 84998,3994,0,250000,1.020,62.594,63.844,21.281,84998, \ 
3994,100,250000,1.020,62.594,63.844,21.281,84998,3994,250,250000, \ 
1.020,62.594,63.844,21.281,84998,3994,475,250000,1.020,62.594, \ 
63.844,21.281,757989,757989,0,510983,506992,3990,1,0,127.066,9983, \ 
1023958,502993,1994,518971,1,0,0,254994,0,3990,0.100,0.000,99.900, \ 
0,270303,812,4,0,812,2223,2223,262,262,2.738,2.738,0.000,812,812, \ 
2270.982,0,0.000

$ claspfolio --list-features

maxLearnt,Constraints,LearntConstraints,FreeVars,Vars/FreeVars, ...
Feature-extraction with `claspfolio`

$ claspfolio --features queens500

PRESOLVING
Reading from queens500
Solving...
UNKNOWN
Features : 84998,3994,0,250000,1.020,62.594,63.844,21.281,84998, \\ 3994,100,250000,1.020,62.594,63.844,21.281,84998,3994,250,250000, \\ 1.020,62.594,63.844,21.281,84998,3994,475,250000,1.020,62.594, \\ 63.844,21.281,757989,757989,0,510983,506992,3990,1,0,127.066,9983, \\ 1023958,502993,1994,518971,1,0,0,254994,0,3990,0.100,0.000,99.900, \\ 0,270303,812,4,0,812,2223,2223,262,262,2.738,2.738,0.000,812,812, \\ 2270.982,0,0.000

$ claspfolio --list-features

maxLearnt,Constraints,LearntConstraints,FreeVars,Vars/FreeVars, ...
Prediction with *claspfolio*

```bash
$ claspfolio queens500 --decisionvalues

PRESOLVING
Reading from queens500
Solving...

Portfolio Decision Values:
```

UNKNOWN

---

*Torsten Schaub (KRR@UP)*

*Answer Set Solving in Practice*

*February 18, 2019*
Prediction with *claspfolio*

$ claspfolio queens500 --decisionvalues

PRESOLVING
Reading from queens500
Solving...

Portfolio Decision Values:

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th></th>
<th>Value</th>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.437538</td>
<td>10</td>
<td>3.639444</td>
<td>19</td>
<td>3.726391</td>
</tr>
<tr>
<td>2</td>
<td>3.501728</td>
<td>11</td>
<td>3.483334</td>
<td>20</td>
<td>3.020325</td>
</tr>
<tr>
<td>3</td>
<td>3.784733</td>
<td>12</td>
<td>3.271890</td>
<td>21</td>
<td>3.220219</td>
</tr>
<tr>
<td>4</td>
<td>3.672955</td>
<td>13</td>
<td>3.344085</td>
<td>22</td>
<td>3.998709</td>
</tr>
<tr>
<td>5</td>
<td>3.557408</td>
<td>14</td>
<td>3.315235</td>
<td>23</td>
<td>3.961214</td>
</tr>
<tr>
<td>6</td>
<td>3.942037</td>
<td>15</td>
<td>3.620479</td>
<td>24</td>
<td>3.512924</td>
</tr>
<tr>
<td>7</td>
<td>3.335304</td>
<td>16</td>
<td>3.396838</td>
<td>25</td>
<td>3.078143</td>
</tr>
<tr>
<td>8</td>
<td>3.375315</td>
<td>17</td>
<td>3.238764</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3.432931</td>
<td>18</td>
<td>3.403484</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UNKNOWN
Prediction with claspfolio

$ claspfolio queens500 --decisionvalues

PRESOLVING
Reading from queens500
Solving...

Portfolio Decision Values:

UNKNOWN
Solving with *claspfolio* (slightly verbosely)

```bash
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt
    --modelpath=./models/
queens500 --quiet --autoverbose=1
    --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time    : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time: 4.760s
```
Solving with claspfolio (slightly verbosely)

$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt \       
  --modelpath=./models/ \                     
  queens500 --quiet --autoverbose=1 \         
  --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time    : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time: 4.760s
Solving with `claspfolio` (slightly verbosely)

```bash
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt \
  --modelpath=./models/ \
queens500 --quiet --autoverbose=1 \
  --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time : 4.760s
```
Solving with claspfolio (slightly verbosely)

$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt \  
  --modelpath=./models/  \  
  queens500 --quiet --autoverbose=1  \  
  --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time : 4.760s
Solving with *claspfolio* (slightly verbosely)

```
$ claspfolio queens500 --quiet --autoverbose=1

PRESOLVING
Reading from queens500
Solving...

Chosen configuration: [20]
clasp --configurations=./models/portfolio.txt \ 
   --modelpath=./models/ \ 
   queens500 --quiet --autoverbose=1 \ 
   --heu=VSIDS --sat-pre=20,25,120 --trans-ext=integ

claspfolio version 1.0.1 (based on clasp version 2.0.2)
Reading from queens500
Solving...
SATISFIABLE

Models : 1+
Time     : 4.783s (Solving: 3.96s 1st Model: 3.93s Unsat: 0.00s)
CPU Time : 4.760s
```
Analysis and visualization toolchain for clasp

- clavis
  - Event logger integrated in clasp
  - Records CDCL events like propagation, conflicts, restarts, ...
  - Generated logfiles readable with different backends
  - Easily configurable
  - Applicable to clasp variants like hclasp

- insight
  - Visualization backend for clavis
  - Combines information about problem structure and solving process
  - Networks for structural and aggregated information
  - Plots for temporal information and navigation
clavis

- Analysis and visualization toolchain for clasp
  - clavis
    - Event logger integrated in clasp
    - Records CDCL events like propagation, conflicts, restarts, ...
    - Generated logfiles readable with different backends
    - Easily configurable
    - Applicable to clasp variants like hclasp
  - insight
    - Visualization backend for clavis
    - Combines information about problem structure and solving process
    - Networks for structural and aggregated information
    - Plots for temporal information and navigation
- **clavis**
  - Analysis and visualization toolchain for clasp
  - Event logger integrated in clasp
  - Records CDCL events like propagation, conflicts, restarts, ... 
  - Generated logfiles readable with different backends
  - Easily configurable
  - Applicable to clasp variants like hclasp

- **insight**
  - Visualization backend for clavis
  - Combines information about problem structure and solving process
  - Networks for structural and aggregated information
  - Plots for temporal information and navigation
Visualization Examples

8-Queens: program interaction graph
Visualization Examples

Towers of Hanoi: program interaction graph
Colors showing flipped assignments
Visualization Examples

Towers of Hanoi: flipped assignments between decisions
Visualization Examples

Towers of Hanoi: flipped assignments between decisions (zoomed in)
Visualization Examples

Towers of Hanoi: learned nogoods during zoomed in segment projected onto program interaction graph layout
Visualization Examples

Towers of Hanoi: learned nogoods during zoomed in segment compared to program interaction graph
- Symbol table shows additional information about variables
- Search bar and symbol table allow for dynamic change of the view
- Symbol table shows additional information about variables
- Search bar and symbol table allow for dynamic change of the view
Symbol table shows additional information about variables

Search bar and symbol table allow for dynamic change of the view
Advanced Modeling: Overview

69 Tweaking N-Queens
70 Do’s and Don’t’s
71 Hints
Anything left to worry about?

- **ASP offers**
  - rich yet easy modeling languages
  - efficient instantiation procedures
  - powerful search engines

- **BUT** The problem encoding (still) matters!

- **Example** Sort a list with 8 elements
  - divide-and-conquer \( \sim 8(\log_2 8) = 16 \) “operations”
  - permutation guessing \( \sim 8!/2 = 20160 \) “operations”
Anything left to worry about?

- **ASP offers**
  - rich yet easy modeling languages
  - efficient instantiation procedures
  - powerful search engines

- **BUT** The problem encoding (still) matters!

- **Example** Sort a list with 8 elements
  - divide-and-conquer $\sim 8(\log_2 8) = 16$ “operations”
  - permutation guessing $\sim 8!/2 = 20160$ “operations”
Anything left to worry about?

- **ASP offers**
  - rich yet easy modeling languages
  - efficient instantiation procedures
  - powerful search engines

- **BUT** The problem encoding (still) matters!

- **Example** Sort a list with 8 elements
  - divide-and-conquer \( \sim 8(\log_2 8) = 16 \) “operations”
  - permutation guessing \( \sim \frac{8!}{2} = 20160 \) “operations”
Anything left to worry about?

- ASP offers
  - rich yet easy modeling languages
  - efficient instantiation procedures
  - powerful search engines

- **BUT** The problem encoding (still) matters!

- **Example** Sort a list with 8 elements
  - divide-and-conquer \( \sim 8(\log_2 8) = 16 \) “operations”
  - permutation guessing \( \sim \frac{8!}{2} = 20160 \) “operations”
Tweaking $N$-Queens

Do’s and Don’t’s

Hints
Problem Specification

Given an $N \times N$ chessboard, place $N$ queens such that they do not attack each other (neither horizontally, vertically, nor diagonally)

$N = 4$

Chessboard

Placement
Problem Specification

Given an $N \times N$ chessboard, place $N$ queens such that they do not attack each other (neither horizontally, vertically, nor diagonally)

$N = 4$

Chessboard

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Placement

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Q</td>
</tr>
</tbody>
</table>
Tweaking \(N\)-Queens

A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of \texttt{queen} in a stable model

```
queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- queen(X,Y), queen(X,Y'), Y < Y'.

% DISPLAY
#show queen/2.
```

Torsten Schaub (KRR@UP) Answer Set Solving in Practice February 18, 2019 561 / 653
A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of queen in a stable model

```lp
queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- queen(X,Y), queen(X,Y'), Y < Y'.

% DISPLAY
#show queen/2.
```
A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of queen in a stable model

```lp
queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- queen(X,Y), queen(X',Y), X < X'.

% DISPLAY
#show queen/2.
```

Anything missing?
Torsten Schaub (KRR@UP)
Answer Set Solving in Practice
February 18, 2019 561 / 653
A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of queen in a stable model

queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- queen(X,Y), queen(X’,Y’), X < X’, X’-X = |Y’-Y|.

% DISPLAY
#show queen/2.
A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of `queen` in a stable model

```lp
% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
[...]

% DISPLAY
#show queen/2.
```
A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of queen in a stable model

```prolog
queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
[...]

% DISPLAY
#show queen/2.
```

Anything missing?

Torsten Schaub (KRR@UP)
A First Encoding

1. Each square may host a queen
2. No row, column, or diagonal hosts two queens
3. A placement is given by instances of queen in a stable model
4. We have to place (at least) $N$ queens

queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
[...]
:- not n { queen(X,Y) }.

% DISPLAY
#show queen/2.
Tweaking $N$-Queens

A First Encoding
Let’s Place 8 Queens!

```bash
gringo -c n=8 queens_0.lp | clasp --stats
```

Answer: 1
queen(1,6) queen(2,3) queen(3,1) queen(4,7)
queen(5,5) queen(6,8) queen(7,2) queen(8,4)
SATISFIABLE

Models : 1+
Time   : 0.006s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Choices : 18
Conflicts : 13
Restarts : 0

Variables : 793
Constraints : 729
A First Encoding
Let's Place 8 Queens!

gringo -c n=8 queens_0.lp | clasp --stats

Answer: 1
queen(1,6) queen(2,3) queen(3,1) queen(4,7)
queen(5,5) queen(6,8) queen(7,2) queen(8,4)
SATISFIABLE

Models : 1+
Time : 0.006s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Choices : 18
Conflicts : 13
Restarts : 0

Variables : 793
Constraints : 729
A First Encoding

Let’s Place 8 Queens!

```
gringo -c n=8 queens_0.lp | clasp --stats
```

Answer: 1

queen(1,6) queen(2,3) queen(3,1) queen(4,7)
queen(5,5) queen(6,8) queen(7,2) queen(8,4)

SATISFIABLE

Models : 1+
Time : 0.006s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Choices : 18
Conflicts : 13
Restarts : 0

Variables : 793
Constraints : 729
A First Encoding
Let’s Place 8 Queens!

```
gringo -c n=8 queens_0.lp | clasp --stats
```

Answer: 1

```
queen(1,6) queen(2,3) queen(3,1) queen(4,7)
queen(5,5) queen(6,8) queen(7,2) queen(8,4)
```

SATISFIABLE

Models : 1+
Time : 0.006s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
Choices : 18
Conflicts : 13
Restarts : 0

Variables : 793
Constraints : 729
A First Encoding
Let’s Place 22 Queens!

gringo -c n=22 queens_0.lp | clasp --stats

Answer: 1
queen(1,10) queen(2,6) queen(3,16) queen(4,14) queen(5,8) ...
SATISFIABLE

Models : 1+
Time : 150.531s (Solving: 150.37s 1st Model: 150.34s Unsat: 0.00s)
CPU Time : 147.480s
Choices : 594960
Conflicts : 574565
Restarts : 19

Variables : 17271
Constraints : 16787
A First Encoding
Let’s Place 22 Queens!

gringo -c n=22 queens_0.lp | clasp --stats

Answer: 1
queen(1,10) queen(2,6) queen(3,16) queen(4,14) queen(5,8) ...
SATISFIABLE

Models: 1+
Time: 150.531s (Solving: 150.37s 1st Model: 150.34s Unsat: 0.00s)
CPU Time: 147.480s
Choices: 594960
Conflicts: 574565
Restarts: 19

Variables: 17271
Constraints: 16787
Tweaking $N$-Queens

A First Refinement

At least $N$ queens?

Exactly one queen per row and column!

queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- queen(X,Y), queen(X,Y'), Y < Y'.
:- queen(X,Y), queen(X',Y), X < X'.
:- queen(X,Y), queen(X',Y'), X < X', X'-X = |Y'-Y|.
:- not n { queen(X,Y) }.

% DISPLAY
#show queen/2.
At least \( N \) queens?

Exactly one queen per row and column!

```lp
% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- queen(X,Y), queen(X',Y), X < X'.
:- queen(X,Y), queen(X',Y'), X < X', X'-X = |Y'-Y|.
:- not n { queen(X,Y) }.

% DISPLAY
#show queen/2.
```
A First Refinement

At least $N$ queens?

Exactly one queen per row and column!

```
queens_0.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- queen(X,Y), queen(X',Y'), X < X', X'-X = |Y'-Y|.
:- not n { queen(X,Y) }.

% DISPLAY
#show queen/2.
```
At least $N$ queens?

Exactly one queen per row and column!

```latex
\texttt{queens\_1.lp}

\% DOMAIN
\#const n=4. square(1..n,1..n).

\% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

\% TEST
:- X = 1..n, not 1 \{ queen(X,Y) \} 1.
:- Y = 1..n, not 1 \{ queen(X,Y) \} 1.
:- queen(X,Y), queen(X’,Y’), X < X’, X’-X = |Y’-Y|.

\% DISPLAY
#show queen/2.
```
A First Refinement
Let’s Place 22 Queens!

```
gringo -c n=22 queens_1.lp | clasp --stats
```

Answer: 1
queen(1,18) queen(2,10) queen(3,21) queen(4,3) queen(5,5) ...
SATISFIABLE

Models : 1+
Time : 0.113s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.020s
Choices : 132
Conflicts : 105
Restarts : 1

Variables : 7238
Constraints : 6710
A First Refinement
Let’s Place 22 Queens!

```
gringo -c n=22 queens_1.lp | clasp --stats
```

Answer: 1
```
queen(1,18) queen(2,10) queen(3,21) queen(4,3) queen(5,5) ...
```

SATISFIABLE

Models : 1+
Time : 0.113s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.020s
Choices : 132
Conflicts : 105
Restarts : 1

Variables : 7238
Constraints : 6710
A First Refinement
Let’s Place 122 Queens!

```
gringo -c n=122 queens_1.lp | clasp --stats
Answer: 1
queen(1,24) queen(2,52) queen(3,37) queen(4,60) queen(5,76) ...
SATISFIABLE
```

Models : 1+
Time : 79.475s (Solving: 1.06s 1st Model: 1.06s Unsat: 0.00s)
CPU Time : 6.930s
Choices : 1373
Conflicts : 845
Restarts : 4

Variables : 1211338
Constraints : 1196210
A First Refinement
Let’s Place 122 Queens!

gringo -c n=122 queens_1.lp | clasp --stats

Answer: 1
queen(1,24) queen(2,52) queen(3,37) queen(4,60) queen(5,76) ...
SATISFIABLE

Models : 1+
Time : 79.475s (Solving: 1.06s 1st Model: 1.06s Unsat: 0.00s)
CPU Time : 6.930s
Choices : 1373
Conflicts : 845
Restarts : 4

Variables : 1211338
Constraints : 1196210
A First Refinement
Let's Place 122 Queens!

gringo -c n=122 queens_1.lp | clasp --stats

Answer: 1
queen(1,24) queen(2,52) queen(3,37) queen(4,60) queen(5,76) ...
SATISFIABLE

Models : 1+
Time : 79.475s (Solving: 1.06s 1st Model: 1.06s Unsat: 0.00s)
CPU Time : 6.930s
Choices : 1373
Conflicts : 845
Restarts : 4

Variables : 1211338
Constraints : 1196210
Tweaking $N$-Queens

A First Refinement
Where Time Has Gone

```
time gringo -c n=122 queens_1.lp | clasp --stats
1241358 7402724 24950848
real 1m15.468s
user 1m15.980s
sys 0m0.090s
```
A First Refinement
Where Time Has Gone

time(gringo -c n=122 queens_1.lp | wc)

1241358 7402724 24950848

real 1m15.468s
user 1m15.980s
sys 0m0.090s
Tweaking $N$-Queens

A First Refinement
Where Time Has Gone

```
time(gringo -c n=122 queens_1.lp | wc)
```

```
1241358 7402724 24950848
real 1m15.468s
user 1m15.980s
sys  0m0.090s
```
A First Refinement
Where Time Has Gone

time(gringo -c n=122 queens_1.lp | wc)

1241358 7402724 24950848

real 1m15.468s
user 1m15.980s
sys 0m0.090s
A First Refinement

Grounding Time \sim Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
Tweaking \( N \)-Queens

A First Refinement

Grounding Time \( \sim \) Space

queens_1.lp

\% DOMAIN
#const n=4. square(1..n,1..n).

\% GENERATE
\{ queen(X,Y) \} :- square(X,Y).

\% TEST
:- X := 1..n, not 1 #count\{ queen(X,Y) \} 1.
:- Y := 1..n, not 1 #count\{ queen(X,Y) \} 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

\% DISPLAY
#show queen/2.
Tweaking N-Queens

A First Refinement

Grounding Time $\sim$ Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
Tweaking $N$-Queens

A First Refinement
Grounding Time $\sim$ Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
A First Refinement

Grounding Time \sim \text{Space}

queens_1.lp

\% DOMAIN
#const n=4. square(1..n,1..n).
\text{O}(n \times n)

\% GENERATE
\{ queen(X,Y) \} :- square(X,Y).
\text{O}(n \times n)

\% TEST
:- X := 1..n, not 1 \#count\{ queen(X,Y) \} 1.
\text{O}(n \times n)

:- Y := 1..n, not 1 \#count\{ queen(X,Y) \} 1.
\text{O}(n \times n)

:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.
\text{O}(n^2 \times n^2)

\% DISPLAY
#show queen/2.
Tweaking $N$-Queens

A First Refinement
Grounding Time $\sim$ Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
Tweaking N-Queens

A First Refinement

Grounding Time ~ Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
A First Refinement

Grounding Time $\sim$ Space

```
queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
```
Tweaking N-Queens

A First Refinement
Grounding Time $\sim$ Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).
O($n \times n$)

% GENERATE
{ queen(X,Y) } :- square(X,Y).
O($n \times n$)

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
O($n \times n$)
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
O($n \times n$)
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.
O($n^2 \times n^2$)

% DISPLAY
#show queen/2.
queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.
A First Refinement

Grounding Time \sim Space

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
{ queen(X,Y) } :- square(X,Y).

% TEST
:- X := 1..n, not 1 #count{ queen(X,Y) } 1.
:- Y := 1..n, not 1 #count{ queen(X,Y) } 1.
:- queen(X1,Y1), queen(X2,Y2), X1 < X2, X2-X1 == |Y2-Y1|.

% DISPLAY
#show queen/2.

Diagonals make trouble!
Enumerating Diagonals

\( N = 4 \)

\[
\begin{array}{cccc}
4 & 3 & 2 & 1 \\
3 & 2 & 1 & 4 \\
2 & 1 & 4 & 3 \\
1 & 4 & 3 & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
4 & 3 & 2 & 1 \\
3 & 2 & 1 & 4 \\
2 & 1 & 4 & 3 \\
1 & 4 & 3 & 2 \\
\end{array}
\]

\( \#\text{diagonal}_1 = (#\text{row} + #\text{column}) - 1 \)  \( \#\text{diagonal}_2 = (#\text{row} - #\text{column}) + N \)

Note For each \( N \), indexes 1, \ldots, (2*N)−1 refer to squares on \#diagonal_{1/2}
Enumerating Diagonals

\( N = 4 \)

\[
\begin{array}{cccc}
4 & 5 & 6 & 7 \\
3 & 4 & 5 & 6 \\
2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
4 & 7 & 6 & 5 \\
3 & 6 & 5 & 4 \\
2 & 5 & 4 & 3 \\
1 & 4 & 3 & 2 \\
\end{array}
\]

\[#\text{diagonal}_1 = (\#\text{row} + \#\text{column}) - 1\]

\[#\text{diagonal}_2 = (\#\text{row} - \#\text{column}) + N\]

- Note: For each \( N \), indexes 1, \ldots, (2*\(N\)) – 1 refer to squares on \#diagonal_{1/2}
Tweaking $N$-Queens

Enumerating Diagonals

$N = 4$

$\#\text{diagonal}_1 = (\#\text{row} + \#\text{column}) - 1$

$\#\text{diagonal}_2 = (\#\text{row} - \#\text{column}) + N$

Note: For each $N$, indexes $1, \ldots, (2N) - 1$ refer to squares on $\#\text{diagonal}_{1/2}$.
Tweaking $N$-Queens

Enumerating Diagonals

$N = 4$

\[
\begin{array}{cccc}
4 & 5 & 6 & 7 \\
3 & 4 & 5 & 6 \\
2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
4 & 7 & 6 & 5 & 4 \\
3 & 6 & 5 & 4 & 3 \\
2 & 5 & 4 & 3 & 2 \\
1 & 4 & 3 & 2 & 1 \\
\end{array}
\]

$\#_{\text{diagonal}_1} = (\#_{\text{row}} + \#_{\text{column}}) - 1$

$\#_{\text{diagonal}_2} = (\#_{\text{row}} - \#_{\text{column}}) + N$

- Note For each $N$, indexes $1, \ldots, (2*N)-1$ refer to squares on $\#_{\text{diagonal}_1/2}$
A Second Refinement

Let’s go for Diagonals!

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- queen(X,Y), queen(X’,Y’), X < X’, X’-X = |Y’-Y|.

% DISPLAY
#show queen/2.
Tweaking $N$-Queens

A Second Refinement
Let’s go for Diagonals!

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X+Y)-1 }. % Diagonal 1

% DISPLAY
#show queen/2.
A Second Refinement
Let’s go for Diagonals!

queens_1.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X+Y)-1 }. % Diagonal 1
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X-Y)+n }. % Diagonal 2

% DISPLAY
#show queen/2.
A Second Refinement
Let’s go for Diagonals!

queens_2.lp

% DOMAIN
#const n=4. square(1..n,1..n).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X+Y)-1 }. % Diagonal 1
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X-Y)+n }. % Diagonal 2

% DISPLAY
#show queen/2.
A Second Refinement
Let’s Place 122 Queens!

gringo -c n=122 queens_2.lp | clasp --stats

Answer: 1
queen(1,98) queen(2,54) queen(3,89) queen(4,83) queen(5,59) ...
SATISFIABLE

Models : 1+
Time : 2.211s (Solving: 0.13s 1st Model: 0.13s Unsat: 0.00s)
CPU Time : 0.210s
Choices : 11036
Conflicts : 499
Restarts : 3

Variables : 16098
Constraints : 970
Tweaking $N$-Queens

A Second Refinement
Let’s Place 122 Queens!

```
gringo -c n=122 queens_2.lp | clasp --stats
```

Answer: 1
queen(1,98) queen(2,54) queen(3,89) queen(4,83) queen(5,59) ...
SATISFIABLE

Models : 1+
Time : 2.211s (Solving: 0.13s 1st Model: 0.13s Unsat: 0.00s)
CPU Time : 0.210s
Choices : 11036
Conflicts : 499
Restarts : 3

Variables : 16098
Constraints : 970
A Second Refinement
Let’s Place 122 Queens!

```
gringo -c n=122 queens_2.lp | clasp --stats

Answer: 1
queen(1,98) queen(2,54) queen(3,89) queen(4,83) queen(5,59) ...
SATISFIABLE

Models : 1+
Time : 2.211s (Solving: 0.13s 1st Model: 0.13s Unsat: 0.00s)
CPU Time : 0.210s
Choices : 11036
Conflicts : 499
Restarts : 3

Variables : 16098
Constraints : 970
```
A Second Refinement
Let’s Place 300 Queens!

```
gringo -c n=300 queens_2.lp | clasp --stats
```

Answer: 1
queen(1,62) queen(2,232) queen(3,176) queen(4,241) queen(5,207) ...
SATISFIABLE

Models : 1+
Time : 35.450s (Solving: 6.69s 1st Model: 6.68s Unsat: 0.00s)
CPU Time : 7.250s
Choices : 141445
Conflicts : 7488
Restarts : 9

Variables : 92994
Constraints : 2394
A Second Refinement
Let’s Place 300 Queens!

gringo -c n=300 queens_2.lp | clasp --stats

Answer: 1
queen(1,62) queen(2,232) queen(3,176) queen(4,241) queen(5,207) ...
SATISFIABLE

Models : 1+
Time : 35.450s (Solving: 6.69s 1st Model: 6.68s Unsat: 0.00s)
CPU Time : 7.250s
Choices : 141445
Conflicts : 7488
Restarts : 9

Variables : 92994
Constraints : 2394
A Second Refinement
Let’s Place 300 Queens!

gringo -c n=300 queens_2.lp | clasp --stats

Answer: 1
queen(1,62) queen(2,232) queen(3,176) queen(4,241) queen(5,207) ...
SATISFIABLE

Models : 1+
Time : 35.450s (Solving: 6.69s 1st Model: 6.68s Unsat: 0.00s)
CPU Time : 7.250s
Choices : 141445
Conflicts : 7488
Restarts : 9

Variables : 92994
Constraints : 2394
Tweaking $N$-Queens

A Third Refinement
Let’s Precalculate Indexes!

queens_2.lp

% DOMAIN
#const n=4. square(1..n,1..n).
diag1(X,Y,(X+Y)-1) :- square(X,Y). diag2(X,Y,(X-Y)+n) :- square(X,Y).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X+Y)-1 }. % Diagonal 1
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X-Y)+n }. % Diagonal 2

% DISPLAY
#show queen/2.
A Third Refinement
Let’s Precalculate Indexes!

queens_2.lp

% DOMAIN
#const n=4. square(1..n,1..n).
diag1(X,Y,(X+Y)-1) :- square(X,Y). diag2(X,Y,(X-Y)+n) :- square(X,Y).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X+Y)-1 }. % Diagonal 1
:- D = 1..2*n-1, 2 { queen(X,Y) : D = (X-Y)+n }. % Diagonal 2

% DISPLAY
#show queen/2.
Tweaking $N$-Queens

A Third Refinement
Let's Precalculate Indexes!

queens_2.lp

% DOMAIN
#const n=4. square(1..n,1..n).
diag1(X,Y,(X+Y)-1) :- square(X,Y). diag2(X,Y,(X-Y)+n) :- square(X,Y).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : diag1(X,Y,D) }. % Diagonal 1
:- D = 1..2*n-1, 2 { queen(X,Y) : diag2(X,Y,D) }. % Diagonal 2

% DISPLAY
#show queen/2.
A Third Refinement
Let’s Precalculate Indexes!

queens_3.lp

% DOMAIN
#const n=4. square(1..n,1..n).
diag1(X,Y,(X+Y)-1) :- square(X,Y). diag2(X,Y,(X-Y)+n) :- square(X,Y).

% GENERATE
0 { queen(X,Y) } 1 :- square(X,Y).

% TEST
:- X = 1..n, not 1 { queen(X,Y) } 1.
:- Y = 1..n, not 1 { queen(X,Y) } 1.
:- D = 1..2*n-1, 2 { queen(X,Y) : diag1(X,Y,D) }. % Diagonal 1
:- D = 1..2*n-1, 2 { queen(X,Y) : diag2(X,Y,D) }. % Diagonal 2

% DISPLAY
#show queen/2.
Tweaking $N$-Queens

A Third Refinement

Let’s Place 300 Queens!

```
gringo -c n=300 queens_3.lp | clasp --stats
```

Answer: 1

```
queen(1,62) queen(2,232) queen(3,176) queen(4,241) queen(5,207) ...
```

SATISFIABLE

Models : 1+

```
Time : 8.889s (Solving: 6.61s 1st Model: 6.60s Unsat: 0.00s)
```

CPU Time : 7.320s

Choices : 141445

Conflicts : 7488

Restarts : 9

Variables : 92994

Constraints : 2394
A Third Refinement
Let’s Place 300 Queens!

```bash
gringo -c n=300 queens_3.lp | clasp --stats
```

Answer: 1
queen(1,62) queen(2,232) queen(3,176) queen(4,241) queen(5,207) ...
SATISFIABLE

Models : 1+
Time : 8.889s (Solving: 6.61s 1st Model: 6.60s Unsat: 0.00s)
CPU Time : 7.320s
Choices : 141445
Conflicts : 7488
Restarts : 9

Variables : 92994
Constraints : 2394
A Third Refinement
Let’s Place 300 Queens!

gringo -c n=300 queens_3.lp | clasp --stats

Answer: 1
queen(1,62) queen(2,232) queen(3,176) queen(4,241) queen(5,207) ...
SATISFIABLE

Models : 1+
Time : 8.889s (Solving: 6.61s 1st Model: 6.60s Unsat: 0.00s)
CPU Time : 7.320s
Choices : 141445
Conflicts : 7488
Restarts : 9

Variables : 92994
Constraints : 2394
A Third Refinement
Let’s Place 600 Queens!

```
gringo -c n=600 queens_3.lp | clasp --stats
```

Answer: 1
queen(1,477) queen(2,365) queen(3,455) queen(4,470) queen(5,237) ...
SATISFIABLE

Models : 1+
Time : 76.798s (Solving: 65.81s 1st Model: 65.75s Unsat: 0.00s)
CPU Time : 68.620s
Choices : 869379
Conflicts : 25746
Restarts : 12

Variables : 365994
Constraints : 4794
A Third Refinement
Let’s Place 600 Queens!

```
gringo -c n=600 queens_3.lp | clasp --stats

Answer: 1
queen(1,477) queen(2,365) queen(3,455) queen(4,470) queen(5,237) ...
SATISFIABLE

Models : 1+
Time   : 76.798s (Solving: 65.81s 1st Model: 65.75s Unsat: 0.00s)
CPU Time : 68.620s
Choices : 869379
Conflicts : 25746
Restarts : 12

Variables : 365994
Constraints : 4794
```
A Case for Oracles
Let’s Place 600 Queens!

```
gringo -c n=600 queens_3.lp | clasp --stats
--heuristic=vsids --trans-ext=dynamic
```

Answer: 1
queen(1,477) queen(2,365) queen(3,455) queen(4,470) queen(5,237) ...
SATISFIABLE

Models : 1+
Time : 76.798s (Solving: 65.81s 1st Model: 65.75s Unsat: 0.00s)
CPU Time : 68.620s
Choices : 869379
Conflicts : 25746
Restarts : 12

Variables : 365994
Constraints : 4794
Tweaking $N$-Queens

A Case for Oracles

Let’s Place 600 Queens!

gringo -c n=600 queens_3.lp | clasp --stats --heuristic=vsids --trans-ext=dynamic

Answer: 1
queen(1,477) queen(2,365) queen(3,455) queen(4,470) queen(5,237) ...
SATISFIABLE

Models : 1+
Time : 76.798s (Solving: 65.81s 1st Model: 65.75s Unsat: 0.00s)
CPU Time : 68.620s
Choices : 869379
Conflicts : 25746
Restarts : 12

Variables : 365994
Constraints : 4794
gringo \(-c \text{ n=600 queens}_3\).lp \mid \text{ clasp --stats}
\(--\text{heuristic=vsids} \ --\text{trans-ext=dynamic}\)

Answer: 1
queen(1,422) queen(2,458) queen(3,224) queen(4,408) queen(5,405) ...  
SATISFIABLE

Models : 1+
Time : 37.454s (Solving: 26.38s 1st Model: 26.26s Unsat: 0.00s)
CPU Time : 29.580s
Choices : 961315
Conflicts : 3222
Restarts : 7

Variables : 365994
Constraints : 4794
A Case for Oracles
Let’s Place 600 Queens!

gringo -c n=600 queens_3.lp | clasp --stats
--heuristic=vsids --trans-ext=dynamic

Answer: 1
queen(1,422) queen(2,458) queen(3,224) queen(4,408) queen(5,405) ...
SATISFIABLE

Models     : 1+
Time       : 37.454s (Solving: 26.38s 1st Model: 26.26s Unsat: 0.00s)
CPU Time   : 29.580s
Choices    : 961315
Conflicts  : 3222
Restarts   : 7

Variables  : 365994
Constraints: 4794
A Case for Oracles
Let’s Place 600 Queens!

gringo -c n=600 queens_3.lp | clasp --stats --heuristic=vsids --trans-ext=dynamic

Answer: 1
queen(1,90) queen(2,452) queen(3,494) queen(4,145) queen(5,84) ...
SATISFIABLE

Models : 1+
Time : 22.654s (Solving: 10.53s 1st Model: 10.47s Unsat: 0.00s)
CPU Time : 15.750s
Choices : 1058729
Conflicts : 2128
Restarts : 6

Variables : 403123
Constraints : 49636
Do's and Don't's

Outline

Tweaking $N$-Queens

Do's and Don't's

Hints
Do’s and Don’t’s

Implementing Universal Quantification

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change
2. use variable-sized conjunction (via ‘:’) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

**Example:** vegetables to buy

veg(asparagus). veg(cucumber).
pro(asparagus, cheap). pro(cucumber, cheap).
pro(asparagus, fresh). pro(cucumber, fresh).
pro(asparagus, tasty). pro(cucumber, tasty).
Implementing Universal Quantification

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change ✗
2. use variable-sized conjunction (via ‘:’) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

**Example:** vegetables to buy

veg(asparagus).  veg(cucumber).
pro(asparagus,cheap).  pro(cucumber,cheap).
pro(asparagus,fresh).  pro(cucumber,fresh).
pro(asparagus,tasty).  pro(cucumber,tasty).

buy(X) :- veg(X), pro(X,cheap), pro(X,fresh), pro(X,tasty).
Implementing Universal Quantification

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change
2. use variable-sized conjunction (via ‘;’) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

**Example:** vegetables to buy

```
veg(asparagus). veg(cucumber).
pro(asparagus, cheap). pro(cucumber, cheap).
pro(asparagus, fresh). pro(cucumber, fresh).
pro(asparagus, tasty). pro(cucumber, tasty).
pro(asparagus, clean).

buy(X) :- veg(X), pro(X, cheap), pro(X, fresh), pro(X, tasty), pro(X, clean).
```
Implementing Universal Quantification

**Goal:** identify objects such that **ALL** properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change
2. use variable-sized conjunction (via ‘:’) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

**Example:** vegetables to buy

veg(asparagus). veg(cucumber).
pro(asparagus,cheap). pro(cucumber,cheap).
pro(asparagus,fresh). pro(cucumber,fresh).
pro(asparagus,tasty). pro(cucumber,tasty).
pro(asparagus,clean).
Implementing Universal Quantification

**Goal:** identify objects such that **ALL** properties from a “list” hold

1. **check all properties explicitly** ... obsolete if properties change
2. **use variable-sized conjunction (via ‘::’)** ... adapts to changing facts
3. **use negation of complement** ... adapts to changing facts

**Example:** vegetables to buy

```
veg(asparagus). veg(cucumber).
pro(asparagus, cheap). pro(cucumber, cheap). pre(cheap).
pro(asparagus, fresh). pro(cucumber, fresh). pre(fresh).
pro(asparagus, tasty). pro(cucumber, tasty). pre(tasty).
pro(asparagus, clean).

buy(X) :- veg(X), pro(X, P) : pre(P).
```
Do's and Don't's

Implementing Universal Quantification

Goal: identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change
2. use variable-sized conjunction (via ‘:’) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

Example: vegetables to buy

veg(asparagus). veg(cucumber).
pro(asparagus,cheap). pro(cucumber,cheap). pre(cheap).
pro(asparagus,fresh). pro(cucumber,fresh). pre(fresh).
pro(asparagus,tasty). pro(cucumber,tasty). pre(tasty).
pro(asparagus,clean). pre(clean).

buy(X) :- veg(X), pro(X,P) : pre(P).
Do’s and Don’t’s

Implementing Universal Quantification

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change ✗
2. use variable-sized conjunction (via ‘:’) ... adapts to changing facts ✓
3. use negation of complement ... adapts to changing facts ✓

**Example:** vegetables to buy

veg(asparagus). veg(cucumber).
pro(asparagus,cheap). pro(cucumber,cheap). pre(cheap).
pro(asparagus,fresh). pro(cucumber,fresh). pre(fresh).
pro(asparagus,tasty). pro(cucumber,tasty). pre(tasty).
pro(asparagus,clean).

Torsten Schaub (KRR@UP)
**Do’s and Don’t’s**

**Implementing Universal Quantification**

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change
2. use variable-sized conjunction (via ‘;’) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

**Example:** vegetables to buy

```
veg(asparagus). veg(cucumber).
pro(asparagus,cheap). pro(cucumber,cheap). pre(cheap).
pro(asparagus,fresh). pro(cucumber,fresh). pre(fresh).
pro(asparagus,tasty). pro(cucumber,tasty). pre(tasty).
pro(asparagus,clean).

buy(X) :- veg(X), not bye(X). bye(X) :- veg(X), pre(P), not pro(X,P).
```
Implementing Universal Quantification

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change
2. use variable-sized conjunction (via ‘:) ... adapts to changing facts
3. use negation of complement ... adapts to changing facts

**Example:** vegetables to buy

```prolog
veg(asparagus). veg(cucumber).
pro(asparagus,cheap). pro(cucumber,cheap). pre(cheap).
pro(asparagus,fresh). pro(cucumber,fresh). pre(fresh).
pro(asparagus,tasty). pro(cucumber,tasty). pre(tasty).
pro(asparagus,clean). pre(clean).

buy(X) :- veg(X), not bye(X).
bye(X) :- veg(X), pre(P), not pro(X,P).
```
Do's and Don'ts

Implementing Universal Quantification

**Goal:** identify objects such that ALL properties from a “list” hold

1. check all properties explicitly ... obsolete if properties change ✗
2. use variable-sized conjunction (via ‘:’) ... adapts to changing facts ✓
3. use negation of complement ... adapts to changing facts ✓

**Example:** vegetables to buy

```
veg(asparagus). veg(cucumber).
pro(asparagus,cheap). pro(cucumber,cheap). pre(cheap).
pro(asparagus,fresh). pro(cucumber,fresh). pre(fresh).
pro(asparagus,tasty). pro(cucumber,tasty). pre(tasty).
pro(asparagus,clean). pre(clean).
```

```
buy(X) :- veg(X), not bye(X). bye(X) :- veg(X), pre(P), not pro(X,P).
```
Do's and Don'ts

Running Example: Latin Square

Given: an $N \times N$ board

\[
\begin{array}{cccccc}
1 & & & & & \\
2 & & & & & \\
3 & & & & & \\
4 & & & & & \\
5 & & & & & \\
6 & & & & & \\
\end{array}
\]

represented by facts:

\[
square(1,1). \ldots \ square(1,6). \  \\
square(2,1). \ldots \ square(2,6). \  \\
square(3,1). \ldots \ square(3,6). \  \\
square(4,1). \ldots \ square(4,6). \  \\
square(5,1). \ldots \ square(5,6). \  \\
square(6,1). \ldots \ square(6,6). \  \\
\]

Wanted: assignment of $1, \ldots, N$

\[
\begin{array}{cccccc}
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 3 & 4 & 5 & 6 & 1 \\
3 & 3 & 4 & 5 & 6 & 1 & 2 \\
4 & 4 & 5 & 6 & 1 & 2 & 3 \\
5 & 5 & 6 & 1 & 2 & 3 & 4 \\
6 & 6 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

represented by atoms:

\[
\text{num}(1,1,1) \ \text{num}(1,2,2) \ldots \ \text{num}(1,6,6) \\
\text{num}(2,1,2) \ \text{num}(2,2,3) \ldots \ \text{num}(2,6,1) \\
\text{num}(3,1,3) \ \text{num}(3,2,4) \ldots \ \text{num}(3,6,2) \\
\text{num}(4,1,4) \ \text{num}(4,2,5) \ldots \ \text{num}(4,6,3) \\
\text{num}(5,1,5) \ \text{num}(5,2,6) \ldots \ \text{num}(5,6,4) \\
\text{num}(6,1,6) \ \text{num}(6,2,1) \ldots \ \text{num}(6,6.5) \\
\]
Running Example: Latin Square

**Given:** an $N \times N$ board

1
2
3
4
5
6

1 2 3 4 5 6

**Wanted:** assignment of 1, \ldots, $N$

1 2 3 4 5 6
2 3 4 5 6 1
3 4 5 6 1 2
4 5 6 1 2 3
5 6 1 2 3 4
6 1 2 3 4 5

represented by facts:

\begin{align*}
square(1,1). & \quad \ldots \quad square(1,6).
square(2,1). & \quad \ldots \quad square(2,6).
square(3,1). & \quad \ldots \quad square(3,6).
square(4,1). & \quad \ldots \quad square(4,6).
square(5,1). & \quad \ldots \quad square(5,6).
square(6,1). & \quad \ldots \quad square(6,6).
\end{align*}

represented by atoms:

\begin{align*}
num(1,1,1) & \quad num(1,2,2) \quad \ldots \quad num(1,6,6)
num(2,1,2) & \quad num(2,2,3) \quad \ldots \quad num(2,6,1)
num(3,1,3) & \quad num(3,2,4) \quad \ldots \quad num(3,6,2)
num(4,1,4) & \quad num(4,2,5) \quad \ldots \quad num(4,6,3)
num(5,1,5) & \quad num(5,2,6) \quad \ldots \quad num(5,6,4)
num(6,1,6) & \quad num(6,2,1) \quad \ldots \quad num(6,6,5)
\end{align*}
Projecting Irrelevant Details Out

A Latin square encoding

\[
\text{\% DOMAIN} \\
\text{\#const n=32. square(1..n,1..n).}
\]

\[
\text{\% GENERATE} \\
1 \{ \text{num}(X,Y,N) : N = 1..n \} 1 :- \text{square}(X,Y).
\]

\[
\text{\% TEST} \\
:- \text{square}(X,Y), N = 1..n, \text{not num}(X,Y',N) : \text{square}(X,Y'). \\
:- \text{square}(X,Y), N = 1..n, \text{not num}(X',Y,N) : \text{square}(X',Y).
\]

Note unreused "singleton variables"

griego latin_0.lp | wc
105480 2558984 14005258

griego latin_1.lp | wc
42056 273672 1690522
Do's and Don't's

Projecting Irrelevant Details Out

A Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- square(X,Y), N = 1..n, not num(X,Y',N) : square(X,Y').
:- square(X,Y), N = 1..n, not num(X',Y,N) : square(X',Y).

Note unreused "singleton variables"
Projecting Irrelevant Details Out

A Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- square(X,Y), N = 1..n, not num(X,Y’,N) : square(X,Y’).
:- square(X,Y), N = 1..n, not num(X’,Y,N) : square(X’,Y).

Note unreused “singleton variables”

gringo latin_0.lp | wc
105480 2558984 14005258

gringo latin_1.lp | wc
42056 273672 1690522
Projecting Irrelevant Details Out

A Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
squareX(X) :- square(X,Y). squareY(Y) :- square(X,Y).

% GENERATE
1 \{ num(X,Y,N) : N = 1..n \} 1 :- square(X,Y).

% TEST
:- squareX(X), N = 1..n, not num(X,Y',N) : square(X,Y').
:- squareY(Y), N = 1..n, not num(X',Y,N) : square(X',Y).

Note unreused “singleton variables”

gringo latin_0.lp | wc
105480 2558984 14005258

gringo latin_1.lp | wc
42056 273672 1690522
Do's and Don'ts

Projecting Irrelevant Details Out

A Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
squareX(X) :- square(X,Y). squareY(Y) :- square(X,Y).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- squareX(X), N = 1..n, not num(X,Y’,N) : square(X,Y’).
:- squareY(Y), N = 1..n, not num(X’,Y,N) : square(X’,Y).

Note unreused “singleton variables”

gringo latin_0.lp | wc
105480 2558984 14005258

gringo latin_1.lp | wc
42056 273672 1690522
Unraveling Symmetric Inequalities

Another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- num(X,Y,N), num(X,Y',N), Y != Y'.
:- num(X,Y,N), num(X',Y,N), X != X'.

Note duplicate ground rules
(swapping Y/Y' and X/X' gives the “same”)

griego latin_2.lp  2071560 12389384 40906946
gringo latin_3.lp  1055752 6294536 21099558
Unraveling Symmetric Inequalities

Another Latin square encoding

\%
\% DOMAIN
#const n=32. square(1..n,1..n).

\%
\% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

\%
\% TEST
:- num(X,Y,N), num(X,Y',N), Y != Y'.
:- num(X,Y,N), num(X',Y,N), X != X'.

Note duplicate ground rules
(swapping Y/Y' and X/X' gives the "same")
Unraveling Symmetric Inequalities

Another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- num(X,Y,N), num(X,Y’,N), Y != Y’.
:- num(X,Y,N), num(X’,Y,N), X != X’.

Note duplicate ground rules
(switching $Y/Y’$ and $X/X’$ gives the “same”)

griingo latin_2.lp | wc
2071560 12389384 40906946
griingo latin_3.lp | wc
1055752 6294536 21099558
Unraveling Symmetric Inequalities

Another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- num(X,Y,N), num(X,Y',N), Y < Y'.
:- num(X,Y,N), num(X',Y,N), X < X'.

Note duplicate ground rules
(swapping Y/Y' and X/X' gives the “same”)

```
gringo latin_2.lp | wc
2071560 12389384 40906946
```
```
gringo latin_3.lp | wc
1055752 6294536 21099558
```
Unraveling Symmetric Inequalities

Another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- num(X,Y,N), num(X,Y',N), Y < Y'.
:- num(X,Y,N), num(X',Y,N), X < X'.

Note duplicate ground rules
(swapping Y/Y' and X/X' gives the “same”)

gringo latin_2.lp | wc
2071560 12389384 40906946

gringo latin_3.lp | wc
1055752 6294536 21099558
Linearizing Existence Tests

Do's and Don't's

Still another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- num(X,Y,N), num(X,Y',N), Y < Y'.
:- num(X,Y,N), num(X',Y,N), X < X'.

Note uniqueness of N in a row/column checked by enumerating pairs!
Do's and Dont's

Linearizing Existence Tests

Still another Latin square encoding

```prolog
% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- num(X,Y,N), num(X,Y',N), Y < Y'.
:- num(X,Y,N), num(X',Y,N), X < X'.
```

Note uniqueness of \( N \) in a row/column checked by enumerating pairs!
Do's and Don't's

Linearizing Existence Tests

Still another Latin square encoding

%%% DOMAIN
#const n=32. square(1..n,1..n).

%%% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

%%% TEST
:- num(X,Y,N), num(X,Y',N), Y < Y'.
:- num(X,Y,N), num(X',Y,N), X < X'.

• Note uniqueness of N in a row/column checked by enumerating pairs!

gringo latin_3.lp | wc
1055752 6294536 21099558

gringo latin_4.lp | wc
228360 1205256 4780744
# Linearizing Existence Tests

## Still another Latin square encoding

```prolog
% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
gtX(X-1,Y,N) :- num(X,Y,N), 1 < X.
gtY(X,Y-1,N) :- num(X,Y,N), 1 < Y.
gtX(X-1,Y,N) :- gtX(X,Y,N), 1 < X.
gtY(X,Y-1,N) :- gtY(X,Y,N), 1 < Y.
  :- num(X,Y,N), gtX(X,Y,N).
  :- num(X,Y,N), gtY(X,Y,N).
```

- **Note** uniqueness of $N$ in a row/column checked by enumerating pairs!

```
gringo latin_3.lp | wc
1055752 6294536 21099558
```

```
gringo latin_4.lp | wc
228360 1205256 4780744
```

---

Do's and Don't's
Linearizing Existence Tests

Still another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
gtX(X-1,Y,N) :- num(X,Y,N), 1 < X.  
gtY(X,Y-1,N) :- num(X,Y,N), 1 < Y.  
gtX(X-1,Y,N) :- gtX(X,Y,N), 1 < X.  
:- num(X,Y,N), gtX(X,Y,N).

Note uniqueness of N in a row/column checked by enumerating pairs!

gringo latin_3.lp | wc
1055752 6294536 21099558

ggringo latin_4.lp | wc
228360 1205256 4780744
Do's and Don't's

Linearizing Existence Tests

Still another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
gtX(X-1,Y,N) :- num(X,Y,N), 1 < X.  
gtY(X,Y-1,N) :- num(X,Y,N), 1 < Y.  
gtX(X-1,Y,N) :- gtX(X,Y,N), 1 < X.  
     :- num(X,Y,N), gtX(X,Y,N).  
gtY(X,Y-1,N) :- gtY(X,Y,N), 1 < Y.  
     :- num(X,Y,N), gtY(X,Y,N).

Note uniqueness of N in a row/column checked by enumerating pairs!

gringo latin_3.lp | wc
1055752 6294536 21099558

gringo latin_4.lp | wc
228360 1205256 4780744
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
sigma(S) :- S = #sum { X:square(X,n) }.

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = { num(X,Y,N) }.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = { num(X,Y,N) }.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3. #show sigma/1.
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
sigma(S) :- S = #sum { X:square(X,n) }.

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = { num(X,Y,N) }.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = { num(X,Y,N) }.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3. #show sigma/1.
Do's and Dont's

Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
sigma(S) :- S = #sum { X:square(X,n) }.

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = { num(X,Y,N) }.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = { num(X,Y,N) }.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3. #show sigma/1.
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
sigma(S) :- S = \#sum \{ X:square(X,n) \}.

% GENERATE
1 \{ num(X,Y,N) : N = 1..n \} 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = \{ num(X,Y,N) \}.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = \{ num(X,Y,N) \}.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3. #show sigma/1.
Do’s and Don’ts

Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
sigma(S) :- S = #sum { X:square(X,n) }.

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C { num(X,Y,N) } C, C = 0..n.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C { num(X,Y,N) } C, C = 0..n.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3. #show sigma/1.

- Note internal transformation by gringo
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).
sigma(S) :- S = #sum { X:square(X,n) }.

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = { num(X,Y,N) }.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = { num(X,Y,N) }.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3. #show sigma/1.

gringo latin_5.lp | wc
304136 5778440 30252505

gringo latin_6.lp | wc
48136 373768 2185042
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = { num(X,Y,N) }.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = { num(X,Y,N) }.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3.

gringo latin_5.lp | wc
gringo latin_6.lp | wc
48136 373768 2185042
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% DEFINE + TEST
occX(X,N,C) :- X = 1..n, N = 1..n, C = { num(X,Y,N) }.
occY(Y,N,C) :- Y = 1..n, N = 1..n, C = { num(X,Y,N) }.
:- occX(X,N,C), C != 1. :- occY(Y,N,C), C != 1.

% DISPLAY
#show num/3.

gringo latin_5.lp | wc
304136 5778440 30252505
gingo latin_6.lp | wc
48136 373768 2185042
Assigning Aggregate Values

Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.

gingo latin_5.lp \ wc
gringo latin_6.lp \ wc

304136 5778440 30252505
Yet another Latin square encoding

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.

gringo latin_5.lp | wc
304136 5778440 30252505

gringo latin_6.lp | wc
48136 373768 2185042
The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.
Do's and Don't's

Breaking Symmetries

The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.

Note **many symmetric solutions**
(mirroring, rotation, value permutation)
Breaking Symmetries

The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.

Note easy and safe to fix a full row/column!
Do's and Don'ts

Breaking Symmetries

The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- square(1,Y), not num(1,Y,Y). % Symmetry Breaking

% DISPLAY
#show num/3.

- Note easy and safe to fix a full row/column!
The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- square(1,Y), not num(1,Y,Y). % Symmetry Breaking

% DISPLAY
#show num/3.

Note Let's compare enumeration speed!
The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.

gringo -c n=5 latin_6.lp | clasp -q 0
The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.

% DISPLAY
#show num/3.

gringo -c n=5 latin_6.lp | clasp -q 0

Models : 161280    Time : 2.078s
The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- square(1,Y), not num(1,Y,Y). % Symmetry Breaking

% DISPLAY
#show num/3.

gringo -c n=5 latin_7.lp | clasp -q 0

Models : 161280  Time : 2.078s
The ultimate Latin square encoding?

% DOMAIN
#const n=32. square(1..n,1..n).

% GENERATE
1 { num(X,Y,N) : N = 1..n } 1 :- square(X,Y).

% TEST
:- X = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- Y = 1..n, N = 1..n, not 1 { num(X,Y,N) } 1.
:- square(1,Y), not num(1,Y,Y). % Symmetry Breaking

% DISPLAY
#show num/3.

gringo -c n=5 latin_7.lp | clasp -q 0

Models : 1344      Time : 0.024s
Hints

Encode With Care!

1. **Create a working encoding**
   - Q1: Do you need to modify the encoding if the facts change?
   - Q2: Are all variables significant (or statically functionally dependent)?
   - Q3: Can there be (many) identical ground rules?
   - Q4: Do you enumerate pairs of values (to test uniqueness)?
   - Q5: Do you assign dynamic aggregate values (to check a fixed bound)?
   - Q6: Do you admit (obvious) symmetric solutions?
   - Q7: Do you have additional domain knowledge simplifying the problem?
   - Q8: Are you aware of anything else that, if encoded, would reduce grounding and/or solving efforts?

2. **Revise until no “Yes” answer!**
   - Note: If the format of facts makes encoding painful (for instance, abusing grounding for “scientific calculations”), revise the fact format as well.
Hints

Encode With Care!

1. Create a working encoding
   Q1: Do you need to modify the encoding if the facts change?
   Q2: Are all variables significant (or statically functionally dependent)?
   Q3: Can there be (many) identical ground rules?
   Q4: Do you enumerate pairs of values (to test uniqueness)?
   Q5: Do you assign dynamic aggregate values (to check a fixed bound)?
   Q6: Do you admit (obvious) symmetric solutions?
   Q7: Do you have additional domain knowledge simplifying the problem?
   Q8: Are you aware of anything else that, if encoded, would reduce grounding and/or solving efforts?

2. Revise until no “Yes” answer!
   - Note: If the format of facts makes encoding painful (for instance, abusing grounding for “scientific calculations”), revise the fact format as well.
Hints

Encode With Care!

1. **Create a working encoding**
   - Q1: Do you need to modify the encoding if the facts change?
   - Q2: Are all variables significant (or statically functionally dependent)?
   - Q3: Can there be (many) identic ground rules?
   - Q4: Do you enumerate pairs of values (to test uniqueness)?
   - Q5: Do you assign dynamic aggregate values (to check a fixed bound)?
   - Q6: Do you admit (obvious) symmetric solutions?
   - Q7: Do you have additional domain knowledge simplifying the problem?
   - Q8: Are you aware of anything else that, if encoded, would reduce grounding and/or solving efforts?

2. **Revise until no “Yes” answer!**

   - Note If the format of facts makes encoding painful (for instance, abusing grounding for “scientific calculations”), revise the fact format as well.
Hints

Encode With Care!

1. Create a working encoding
   - Q1: Do you need to modify the encoding if the facts change?
   - Q2: Are all variables significant (or statically functionally dependent)?
   - Q3: Can there be (many) identical ground rules?
   - Q4: Do you enumerate pairs of values (to test uniqueness)?
   - Q5: Do you assign dynamic aggregate values (to check a fixed bound)?
   - Q6: Do you admit (obvious) symmetric solutions?
   - Q7: Do you have additional domain knowledge simplifying the problem?
   - Q8: Are you aware of anything else that, if encoded, would reduce grounding and/or solving efforts?

2. Revise until no “Yes” answer!

   Note: If the format of facts makes encoding painful (for instance, abusing grounding for “scientific calculations”), revise the fact format as well.

Torsten Schaub (KRR@UP)

Answer Set Solving in Practice

February 18, 2019

586 / 653
Encode With Care!

1. Create a working encoding
   Q1: Do you need to modify the encoding if the facts change?
   Q2: Are all variables significant (or statically functionally dependent)?
   Q3: Can there be (many) identical ground rules?
   Q4: Do you enumerate pairs of values (to test uniqueness)?
   Q5: Do you assign dynamic aggregate values (to check a fixed bound)?
   Q6: Do you admit (obvious) symmetric solutions?
   Q7: Do you have additional domain knowledge simplifying the problem?
   Q8: Are you aware of anything else that, if encoded, would reduce grounding and/or solving efforts?

2. Revise until no “Yes” answer!

   Note If the format of facts makes encoding painful (for instance, abusing grounding for “scientific calculations”), revise the fact format as well.
Hints

Encode With Care!

1. Create a working encoding
   - Q1: Do you need to modify the encoding if the facts change?
   - Q2: Are all variables significant (or statically functionally dependent)?
   - Q3: Can there be (many) identical ground rules?
   - Q4: Do you enumerate pairs of values (to test uniqueness)?
   - Q5: Do you assign dynamic aggregate values (to check a fixed bound)?
   - Q6: Do you admit (obvious) symmetric solutions?
   - Q7: Do you have additional domain knowledge simplifying the problem?
   - Q8: Are you aware of anything else that, if encoded, would reduce grounding and/or solving efforts?

2. Revise until no “Yes” answer!
   - Note If the format of facts makes encoding painful (for instance, abusing grounding for “scientific calculations”), revise the fact format as well.
Some Hints on (Preventing) Debugging

Kinds of errors

- syntactic ... follow error messages by the grounder
- semantic ... (most likely) encoding/intention mismatch

Ways to identify semantic errors (early)

develop and test incrementally

prepare toy instances with "interesting features"
build the encoding bottom-up and verify additions (e.g., new predicates)

compare the encoded to the intended meaning

check whether the grounding fits (use gringo --text)
if stable models are unintended, investigate conditions that fail to hold
if stable models are missing, examine integrity constraints (add heads)

ask tools (e.g., http://www.kr.tuwien.ac.at/research/projects/mmdasp/)
Some Hints on (Preventing) Debugging

Kinds of errors

- **syntactic** ... follow error messages by the grounder
- **semantic** ... (most likely) encoding/intention mismatch

Ways to identify semantic errors (early)

1. develop and test incrementally
   - prepare toy instances with “interesting features”
   - build the encoding bottom-up and verify additions (e.g., new predicates)
2. compare the encoded to the intended meaning
   - check whether the grounding fits (use gringo --text)
   - if stable models are unintended, investigate conditions that fail to hold
   - if stable models are missing, examine integrity constraints (add heads)
3. ask tools (e.g., http://www.kr.tuwien.ac.at/research/projects/mmdasp/)
Some Hints on (Preventing) Debugging

Kinds of errors

- syntactic ... follow error messages by the grounder
- semantic ... (most likely) encoding/intention mismatch

Ways to identify semantic errors (early)

1. develop and test incrementally
   - prepare toy instances with “interesting features”
   - build the encoding bottom-up and verify additions (e.g., new predicates)
2. compare the encoded to the intended meaning
   - check whether the grounding fits (use gringo --text)
   - if stable models are unintended, investigate conditions that fail to hold
   - if stable models are missing, examine integrity constraints (add heads)
3. ask tools (e.g., http://www.kr.tuwien.ac.at/research/projects/mmdasp/)
Some Hints on (Preventing) Debugging

Kinds of errors

- **syntactic** ... follow error messages by the grounder
- **semantic** ... (most likely) encoding/intention mismatch

Ways to identify semantic errors (early)

1. **develop and test incrementally**
   - prepare toy instances with "interesting features"
   - build the encoding bottom-up and verify additions (eg. new predicates)

2. **compare the encoded to the intended meaning**
   - check whether the grounding fits (use gringo --text)
   - if stable models are unintended, investigate conditions that fail to hold
   - if stable models are missing, examine integrity constraints (add heads)

Some Hints on (Preventing) Debugging

Kinds of errors

- syntactic ... follow error messages by the grounder
- semantic ... (most likely) encoding/intention mismatch

Ways to identify semantic errors (early)

1. develop and test incrementally
   - prepare toy instances with “interesting features”
   - build the encoding bottom-up and verify additions (eg. new predicates)
2. compare the encoded to the intended meaning
   - check whether the grounding fits (use gringo --text)
   - if stable models are unintended, investigate conditions that fail to hold
   - if stable models are missing, examine integrity constraints (add heads)
3. ask tools (eg. http://www.kr.tuwien.ac.at/research/projects/mmdasp/)
Some Hints on (Preventing) Debugging

Kinds of errors

- syntactic ... follow error messages by the grounder
- semantic ... (most likely) encoding/intention mismatch

Ways to identify semantic errors (early)

1. develop and test incrementally
   - prepare toy instances with “interesting features”
   - build the encoding bottom-up and verify additions (e.g. new predicates)

2. compare the encoded to the intended meaning
   - check whether the grounding fits (use gringo --text)
   - if stable models are unintended, investigate conditions that fail to hold
   - if stable models are missing, examine integrity constraints (add heads)

3. ask tools (e.g. http://www.kr.tuwien.ac.at/research/projects/mmdasp/)
Overcoming Performance Bottlenecks

Grounding

- monitor time spent by and output size of gringo
  1. system tools (eg. `time(gringo [...] | wc)`)  
  2. grounding info (eg. `gringo --text`)
- Note once identified, reformulate “critical” logic program parts

Solving

- check solving statistics (use `clasp --stats`)
  - if great search efforts (Conflicts/Choices/Restarts), then
  - try prefabricated settings (using clasp option ‘--configuration’)
  - try auto-configuration (offered by claspfolio or accclingo)
  - try manual fine-tuning (requires expert knowledge!)
  - if possible, reformulate the problem or add domain knowledge ("redundant" constraints) to help the solver
Overcoming Performance Bottlenecks

Grounding

- **monitor** time spent by and output size of gringo
  1. system tools (e.g. `time(gringo [...] | wc)`)
  2. grounding info (e.g. `gringo --text`)
- Note once identified, **reformulate** “critical” logic program parts

Solving

- check solving statistics (use `clasp --stats`)
- Note if great search efforts (Conflicts/Choices/Restarts), then try prefabricated settings (using clasp option ‘--configuration’)
  try auto-configuration (offered by claspfolio or accclingo)
  try manual fine-tuning (requires expert knowledge!)
  if possible, reformulate the problem or add domain knowledge ("redundant" constraints) to help the solver
Overcoming Performance Bottlenecks

Grounding

- monitor time spent by and output size of gringo
  - system tools (eg. time(gringo [...] | wc))
  - grounding info (eg. gringo --text)
- Note once identified, reformulate "critical" logic program parts

Solving

- check solving statistics (use clasp --stats)
- Note if great search efforts (Conflicts/Choices/Restarts), then
  - try prefabricated settings (using clasp option ‘--configuration’)
  - try auto-configuration (offered by claspfolio or accclingo)
  - try manual fine-tuning (requires expert knowledge!)
  - if possible, reformulate the problem or add domain knowledge ("redundant" constraints) to help the solver
Overcoming Performance Bottlenecks

Grounding

- monitor time spent by and output size of gringo
  1. system tools (eg. `time(gringo [...] | wc)`)  
  2. grounding info (eg. `gringo --text`)
- Note once identified, reformulate “critical” logic program parts

Solving

- check solving statistics (use `clasp --stats`)
- Note if great search efforts (Conflicts/Choices/Restarts), then
  1. try prefabricated settings (using clasp option ‘--configuration’)  
  2. try auto-configuration (offered by claspfolio or accclingo)  
  3. try manual fine-tuning (requires expert knowledge!)  
  4. if possible, reformulate the problem or add domain knowledge (∼ redundant” constraints) to help the solver
Hints

Overcoming Performance Bottlenecks

Grounding

- monitor **time** spent by and output **size** of gringo
  1. system tools (eg. `time(gringo [...] | wc)`)  
  2. grounding info (eg. `gringo --text`)
- Note once identified, **reformulate** “critical” logic program parts

Solving

- check solving statistics (use `clasp --stats`)
- Note if great search efforts (**Conflicts**/**Choices**/**Restarts**), then
  1. try prefabricated settings (using `clasp` option ‘--configuration’)
  2. try auto-configuration (offered by claspfolio or accclingo)
  3. try manual fine-tuning (requires expert knowledge!)
  4. if possible, reformulate the problem or add domain knowledge ("redundant" constraints) to help the solver
Overcoming Performance Bottlenecks

**Grounding**

- monitor **time** spent by and output **size** of **gringo**
  1. system tools (eg. `time(gringo [...] | wc)`)  
  2. grounding info (eg. `gringo --text`)
- **Note** once identified, **reformulate** “critical” logic program parts

**Solving**

- check solving statistics (use `clasp --stats`)
- **Note** if great search efforts (**Conflicts**/*Choices*/**Restarts**), then
  1. try prefabricated settings (using clasp option ‘--configuration’)  
  2. try auto-configuration (offered by claspfolio or accclingo)  
  3. try manual fine-tuning (requires expert knowledge!)  
  4. if possible, reformulate the problem or add domain knowledge (**"redundant"** constraints) to help the solver
Preferences and optimization: Overview

Motivation
The asprin framework
Preliminaries
Language
Implementation
Summary
Motivation

The asprin framework

Preliminaries

Language

Implementation

Summary
Preferences are pervasive

The identification of preferred, or optimal, solutions is often indispensable in real-world applications.

In many cases, this also involves the combination of various qualitative and quantitative preferences.

Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems.

Example: \#minimize\{40 : sauna, 70 : dive\}
Preferences are pervasive

The identification of preferred, or optimal, solutions is often indispensable in real-world applications.

In many cases, this also involves the combination of various qualitative and quantitative preferences.

Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems.

Example: \#minimize\{40 : sauna, 70 : dive\}
Preferences are pervasive

The identification of preferred, or optimal, solutions is often indispensable in real-world applications. In many cases, this also involves the combination of various qualitative and quantitative preferences.

Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems.

Example

\[
\#minimize\{40 : \text{sauna}, 70 : \text{dive}\}
\]
Preferences are pervasive

The identification of preferred, or optimal, solutions is often indispensable in real-world applications.

In many cases, this also involves the combination of various qualitative and quantitative preferences.

Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems.

Example: \#minimize\{40 : sauna, 70 : dive\}
The asprin framework

Outline

72 Motivation

73 The asprin framework

74 Preliminaries

75 Language

76 Implementation

77 Summary
asprin is a framework for handling preferences among the stable models of logic programs

- general because it captures numerous existing approaches to preference from the literature
- flexible because it allows for an easy implementation of new or extended existing approaches

asprin builds upon advanced control capacities for incremental and meta solving, allowing for

- search for specific preferred solutions without any modifications to the ASP solver
- significantly reducing redundancies
- via an implementation through ordinary ASP encodings
The asprin framework

Approach

- **asprin** is a framework for handling preferences among the stable models of logic programs
  - **general** because it captures numerous existing approaches to preference from the literature
  - **flexible** because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - high customizability via an implementation through ordinary ASP encodings
asprin is a framework for handling preferences among the stable models of logic programs

- general because it captures numerous existing approaches to preference from the literature
- flexible because it allows for an easy implementation of new or extended existing approaches

asprin builds upon advanced control capacities for incremental and meta solving, allowing for

- search for specific preferred solutions without any modifications to the ASP solver
- continuous integrated solving process significantly reducing redundancies
- high customizability via an implementation through ordinary ASP encodings
asprin is a framework for handling preferences among the stable models of logic programs

- general because it captures numerous existing approaches to preference from the literature
- flexible because it allows for an easy implementation of new or extended existing approaches

asprin builds upon advanced control capacities for incremental and meta solving, allowing for

- search for specific preferred solutions without any modifications to the ASP solver
- continuous integrated solving process significantly reducing redundancies
- high customizability via an implementation through ordinary ASP encodings
Example

```
#preference(costs, less(weight)) {40 : sauna, 70 : dive}
#preference(fun, superset) {sauna, dive, hike, ∼bunji}
#preference(temps, aso) {dive > sauna || hot, sauna > dive || ∼hot}
#preference(all, pareto) {name(costs), name(fun), name(temps)}

#optimize(all)
```
Outline

- Motivation
- The asprin framework
- Preliminaries
- Language
- Implementation
- Summary
A strict partial order $\succ$ on the stable models of a logic program

That is, $X \succ Y$ means that $X$ is preferred to $Y$

A stable model $X$ is $\succ$-preferred, if there is no other stable model $Y$ such that $Y \succ X$

A preference type is a (parametric) class of preference relations
A strict partial order $\succeq$ on the stable models of a logic program. That is, $X \succeq Y$ means that $X$ is preferred to $Y$.

A stable model $X$ is $\succeq$-preferred, if there is no other stable model $Y$ such that $Y \succeq X$.

A preference type is a (parametric) class of preference relations.
A strict partial order $\succ$ on the stable models of a logic program. That is, $X \succ Y$ means that $X$ is preferred to $Y$.

A stable model $X$ is $\succ$-preferred, if there is no other stable model $Y$ such that $Y \succ X$.

A preference type is a (parametric) class of preference relations.
A strict partial order $\succ$ on the stable models of a logic program. That is, $X \succ Y$ means that $X$ is preferred to $Y$.

A stable model $X$ is $\succ$-preferred, if there is no other stable model $Y$ such that $Y \succ X$.

A preference type is a (parametric) class of preference relations.
Outline

- Motivation
- The asprin framework
- Preliminaries
- Language
- Implementation
- Summary
Language

- weighted formula \( w_1, \ldots, w_l : \phi \)
  where each \( w_i \) is a term and \( \phi \) is a Boolean formula

- naming atom \( \text{name}(s) \)
  where \( s \) is the name of a preference

- preference element \( \Phi_1 \succ \cdots \succ \Phi_m \| \Phi \)
  where each \( \Phi_r \) is a set of weighted formulas and \( \Phi \) is a non-weighted formula

- preference statement \( \# \text{preference}(s, t)\{e_1, \ldots, e_n\} \)
  where \( s \) and \( t \) represent the preference statement and its type
  and each \( e_j \) is a preference element

- optimization directive \( \# \text{optimize}(s) \)
  where \( s \) is the name of a preference

- preference specification is a set \( S \) of preference statements and a directive
  \( \# \text{optimize}(s) \) such that \( S \) is an acyclic, closed, and \( s \in S \)
- weighted formula \( w_1, \ldots, w_l : \phi \) where each \( w_i \) is a term and \( \phi \) is a Boolean formula
- naming atom \( \text{name}(s) \) where \( s \) is the name of a preference
- preference element \( \Phi_1 > \cdots > \Phi_m \parallel \Phi \) where each \( \Phi_r \) is a set of weighted formulas and \( \Phi \) is a non-weighted formula
- preference statement \( \#\text{preference}(s, t)\{e_1, \ldots, e_n\} \) where \( s \) and \( t \) represent the preference statement and its type and each \( e_j \) is a preference element
- optimization directive \( \#\text{optimize}(s) \) where \( s \) is the name of a preference
- preference specification is a set \( S \) of preference statements and a directive \( \#\text{optimize}(s) \) such that \( S \) is an acyclic, closed, and \( s \in S \)
Language

- weighted formula $w_1, \ldots, w_l : \phi$
  where each $w_i$ is a term and $\phi$ is a Boolean formula

- naming atom $name(s)$
  where $s$ is the name of a preference

- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each $\Phi_r$ is a set of weighted formulas and $\Phi$ is a non-weighted formula

- preference statement $\#preference(s, t)\{e_1, \ldots, e_n\}$
  where $s$ and $t$ represent the preference statement and its type
  and each $e_j$ is a preference element

- optimization directive $\#optimize(s)$
  where $s$ is the name of a preference

- preference specification is a set $S$ of preference statements and a directive
  $\#optimize(s)$ such that $S$ is an acyclic, closed, and $s \in S$
A preference type $t$ is a function mapping a set of preference elements, $E$, to a (strict) preference relation, $t(E)$, on sets of atoms.

The domain of $t$, $\text{dom}(t)$, fixes its admissible preference elements.

Example $\text{less}(\text{cardinality})$

$$(X, Y) \in \text{less}(\text{cardinality})(E)$$

if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$

$\text{dom}(\text{less}(\text{cardinality})) = \mathcal{P}(\{a, \neg a \mid a \in A\})$

(where $\mathcal{P}(X)$ denotes the power set of $X$)
A preference type $t$ is a function mapping a set of preference elements, $E$, to a (strict) preference relation, $t(E)$, on sets of atoms.

The domain of $t$, $\text{dom}(t)$, fixes its admissible preference elements.

Example $\text{less}(\text{cardinality})$

$(X, Y) \in \text{less}(\text{cardinality})(E)$

if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$

$\text{dom(less(cardinality))} = \mathcal{P}(\{a, \neg a \mid a \in A\})$

(where $\mathcal{P}(X)$ denotes the power set of $X$)
A preference type $t$ is a function mapping a set of preference elements, $E$, to a (strict) preference relation, $t(E)$, on sets of atoms. The domain of $t$, $\text{dom}(t)$, fixes its admissible preference elements.

Example $\text{less}(\text{cardinality})$

- $(X, Y) \in \text{less}(\text{cardinality})(E)$ if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$
- $\text{dom}(\text{less}(\text{cardinality})) = \mathcal{P}(\{a, \neg a \mid a \in A\})$
  (where $\mathcal{P}(X)$ denotes the power set of $X$)
A preference type $t$ is a function mapping a set of preference elements, $E$, to a (strict) preference relation, $t(E)$, on sets of atoms.

- The domain of $t$, $\text{dom}(t)$, fixes its admissible preference elements.

- Example $\text{less}(\text{cardinality})$

  - $(X, Y) \in \text{less}(\text{cardinality})(E)$ if $|\{ l \in E \mid X \models l \}| < |\{ l \in E \mid Y \models l \}|$

  - $\text{dom(less(cardinality))} = \mathcal{P}(\{a, \neg a \mid a \in A\})$
    (where $\mathcal{P}(X)$ denotes the power set of $X$)
Preference type

- A preference type $t$ is a function mapping a set of preference elements, $E$, to a (strict) preference relation, $t(E)$, on sets of atoms.
- The domain of $t$, $\text{dom}(t)$, fixes its admissible preference elements.
- Example: $\text{less}(\text{cardinality})$
  - $(X, Y) \in \text{less}(\text{cardinality})(E)$ if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$
  - $\text{dom}(\text{less}(\text{cardinality})) = \mathcal{P}(\{a, \neg a \mid a \in A\})$
    (where $\mathcal{P}(X)$ denotes the power set of $X$)
More examples

- **more(weight)** is defined as
  - \((X, Y) \in more(weight)(E)\) if \(\sum_{(w:l)\in E, X \models l} w > \sum_{(w:l)\in E, Y \models l} w\)
  - \(dom(more(weight)) = \mathcal{P}(\{w : a, w : \neg a \mid w \in \mathbb{Z}, a \in \mathcal{A}\})\); and

- **subset** is defined as
  - \((X, Y) \in subset(E)\) if \(\{l \in E \mid X \models l\} \subset \{l \in E \mid Y \models l\}\)
  - \(dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})\).

- **pareto** is defined as
  - \((X, Y) \in pareto(E)\) if \(\bigwedge_{name(s)\in E} (X \succeq_{s} Y) \land \bigvee_{name(s)\in E} (X \succeq_{s} Y)\)
  - \(dom(pareto) = \mathcal{P}(\{n \mid n \in \mathbb{N}\})\);

- **lexico** is defined as
  - \((X, Y) \in lexico(E)\) if \(\bigvee_{w:name(s)\in E} ((X \succeq_{s} Y) \land \bigwedge_{v:name(s')\in E, v < w} (X =_{s'} Y))\)
  - \(dom(lexico) = \mathcal{P}(\{w : n \mid w \in \mathbb{Z}, n \in \mathbb{N}\})\).
A preference relation is obtained by applying a preference type to an admissible set of preference elements.

\[ \text{#preference}(s, t) E \text{ declares preference relation } t(E) \text{ denoted by } \succ_s \]

Example: \[ \text{#preference}(1, \text{less(cardinality)})\{a, \neg b, c\} \]
declares \[ X \succ_1 Y \text{ as } |\{l \in \{a, \neg b, c\} \mid X \models l\}| < |\{l \in \{a, \neg b, c\} \mid Y \models l\}| \]

where \( \succ_1 \) stands for \( \text{less(cardinality)}(\{a, \neg b, c\}) \)
A preference relation is obtained by applying a preference type to an admissible set of preference elements.

\[ \#\text{preference}(s, t) \ E \] declares preference relation \( t(E) \) denoted by \( \succ_s \).

Example \( \#\text{preference}(1, \text{less}(\text{cardinality})) \{a, \neg b, c\} \) declares

\[ X \succ_1 Y \text{ as } |\{l \in \{a, \neg b, c\} \mid X \models l\}| < |\{l \in \{a, \neg b, c\} \mid Y \models l\}| \]

where \( \succ_1 \) stands for \( \text{less}(\text{cardinality})(\{a, \neg b, c\}) \).
A preference relation is obtained by applying a preference type to an admissible set of preference elements.

\( \#preference(s, t) E \) declares preference relation \( t(E) \) denoted by \( \succ_s \).

Example \( \#preference(1, \text{less}(\text{cardinality}))(\{a, \neg b, c\}) \) declares \( X \succ_1 Y \) as

\[
\left| \{l \in \{a, \neg b, c\} \mid X \models l \} \right| < \left| \{l \in \{a, \neg b, c\} \mid Y \models l \} \right|
\]

where \( \succ_1 \) stands for \( \text{less}(\text{cardinality})(\{a, \neg b, c\}) \).
Outline

- Motivation
- The asprin framework
- Preliminaries
- Language
- Implementation
- Summary
Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$

- Preference program Let $s$ be a preference statement declaring $\succsim_s$ and let $P_s$ be a logic program.

We define $P_s$ as a preference program for $s$, if for all sets $X, Y \subseteq A$, we have

$$X \succsim_s Y \text{ iff } P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note $P_s$ usually consists of an encoding $E_{ts}$ of $t_s$, facts $F_s$ representing the preference statement, and auxiliary rules $A$

- Note Dynamic versions of $H_X$ and $H_Y$ must be used for optimization
Preference program

- **Reification** $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$

- **Preference program** Let $s$ be a preference statement declaring $\succ_s$ and let $P_s$ be a logic program.

We define $P_s$ as a preference program for $s$, if for all sets $X, Y \subseteq A$, we have

$$X \succ_s Y \text{ iff } P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- **Note** $P_s$ usually consists of an encoding $E_{t_s}$ of $t_s$, facts $F_s$ representing the preference statement, and auxiliary rules $A$.

- **Note** Dynamic versions of $H_X$ and $H_Y$ must be used for optimization.
Preference program

- Reification: $H_X = \{ \text{holds}(a) \mid a \in X \}$ and $H'_X = \{ \text{holds}'(a) \mid a \in X \}$

- Preference program: Let $s$ be a preference statement declaring $\succ_S$ and let $P_s$ be a logic program. We define $P_s$ as a preference program for $s$, if for all sets $X, Y \subseteq A$, we have

  \[ X \succ_S Y \text{ iff } P_s \cup H_X \cup H'_Y \text{ is satisfiable} \]

- Note: $P_s$ usually consists of an encoding $E_{t_s}$ of $t_s$, facts $F_s$ representing the preference statement, and auxiliary rules $A$.

- Note: Dynamic versions of $H_X$ and $H_Y$ must be used for optimization.
Preference program

- Reification: $H_X = \{ \text{holds}(a) \mid a \in X \}$ and $H'_X = \{ \text{holds}'(a) \mid a \in X \}$

- Preference program: Let $s$ be a preference statement declaring $\succ_s$ and let $P_s$ be a logic program.

  We define $P_s$ as a preference program for $s$, if for all sets $X, Y \subseteq A$, we have

  $$X \succ_s Y \iff P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note: $P_s$ usually consists of an encoding $E_{t_s}$ of $t_s$, facts $F_s$ representing the preference statement, and auxiliary rules $A$.

- Note: Dynamic versions of $H_X$ and $H_Y$ must be used for optimization.
$\#preference(3, \text{subset})\{a, \neg b, c\}$

\[
E_{\text{subset}} = \begin{cases} 
\text{better(P)} : - \text{preference(P,subset)}, \\
\text{holds}'(X) : \text{preference}(P,_,_,\text{for}(X),_), \text{holds}(X);
1 \ #\sum \{1,X : \text{not holds}(X), \text{holds}'(X), \\
\text{preference}(P,_,_,\text{for}(X),_)}.
\end{cases}
\]

\[
F_3 = \begin{cases} 
\text{preference}(3,\text{subset}). \ \text{preference}(3,1,1,\text{for}(a),()). \\
\text{preference}(3,2,1,\text{for}(\neg b),()). \\
\text{preference}(3,3,1,\text{for}(c),()).
\end{cases}
\]

\[
A = \begin{cases} 
\text{holds}(\neg A) : - \text{not holds}(A), \text{preference}(_,_,_,\text{for}(\neg A),_). \\
\text{holds}'(\neg A) : - \text{not holds}'(A),\text{preference}(_,_,_,\text{for}(\neg A),_).
\end{cases}
\]

\[
H_{\{a,b\}} = \begin{cases} 
\text{holds}(a). \ \text{holds}(b).
\end{cases}
\]

\[
H'_{\{a\}} = \begin{cases} 
\text{holds}'(a).
\end{cases}
\]

We get a stable model containing better(3) indicating that \(\{a, b\} \succ_3 \{a\}\), or \(\{a\} \subset \{a, \neg b\}\).
We get a stable model containing `better(3)` indicating that
\( \{a, b\} \succ_3 \{a\} \), or \( \{a\} \subset \{a, \neg b\} \)
Basic algorithm \(solveOpt(P, s)\)

\[
\begin{align*}
\textbf{Input} & : \text{A program } P \text{ over } \mathcal{A} \text{ and preference statement } s \\
\textbf{Output} & : \text{A } \succsim_s\text{-preferred stable model of } P, \text{ if } P \text{ is satisfiable, and } \bot \text{ otherwise}
\end{align*}
\]

\[
Y \leftarrow solve(P)
\]

\[
\text{if } Y = \bot \text{ then return } \bot
\]

\[
\text{repeat}
\]

\[
X \leftarrow Y
\]

\[
Y \leftarrow solve(P \cup E_t \cup F_s \cup R_A \cup H'_X) \cap \mathcal{A}
\]

\[
\text{until } Y = \bot
\]

\[
\text{return } X
\]

where \(R_X = \{ \text{holds}(a) \leftarrow a \mid a \in X \}\)
Sketched Python Implementation

```python
#script (python)
from gringo import *
holds = []

def getHolds():
    global holds
    return holds

def onModel(model):
    global holds
    holds = []
    for a in model.atoms():
        if (a.name() == "_holds"): holds.append(a.args()[0])

def main(prg):
    step = 1
    prg.ground([("base", [])])
    while True:
        if step > 1:
            prg.ground([("doholds", [step-1]), ("preference", [0, step-1])])
        ret = prg.solve(on_model=onModel)
        if ret == SolveResult.UNSAT: break
        step = step+1

#program base.  #program doholds(m).
#show _holds(X,0) : _holds(X,0).  _holds(X,m) :- X = @getHolds().

#end.
#script (python)

def onModel(model):
    global holds
    holds = []
    for a in model.atoms():
        if (a.name() == "_holds"): holds.append(a.args()[0])

def main(prg):
    step = 1
    prg.ground([("base", [])])
    while True:
        if step > 1: prg.ground([("doholds", [step-1]), ("preference", [0, step-1])])
        ret = prg.solve(on_model=onModel)
        if ret == SolveResult.UNSAT: break
        step = step+1

#end.

#program base.

#program doholds(m).

#show _holds(X,0) : _holds(X,0).  _holds(X,m) :- X = @getHolds().

#end.
**Vanilla minimize statements**

- **Emulating the minimize statement**

  
  ```
  #minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
  ```

  *in asprin amounts to*

  ```
  #preference(myminimize,less(weight))
  { C,(X,Y) :: cycle(X,Y) : cost(X,Y,C) }
  #optimize(myminimize).
  ```

- **Note** *asprin* separates the declaration of preferences from the actual optimization directive
Vanilla minimize statements

- Emulating the minimize statement

  \[
  \#\text{minimize} \{ \text{C,X,Y} : \text{cycle}(X,Y), \text{cost}(X,Y,C) \}.
  \]

  in asprin amounts to

  \[
  \#\text{preference}(\text{myminimize}, \text{less}(\text{weight}))
  \{ \text{C,(X,Y)} :: \text{cycle}(X,Y) : \text{cost}(X,Y,C) \}.
  \]

  \#\text{optimize}(\text{myminimize}).

- Note asprin separates the declaration of preferences from the actual optimization directive
Example in *asprin*'s input language

```prolog
#preference(costs,less(weight)){
    C :: sauna : cost(sauna,C);
    C :: dive : cost(dive,C)
}.
#preference(fun,superset){ sauna; dive; hike; not bunji }.
#preference(temps,aso){
    dive > sauna || hot;
    sauna > dive || not hot
}.
#preference(all,pareto){name(costs); name(fun); name(temps)}.

#optimize(all).
```
asprin’s library

- **Basic preference types**
  - subset and superset
  - less(cardinality) and more(cardinality)
  - less(weight) and more(weight)
  - aso (Answer Set Optimization)
  - poset (Qualitative Preferences)

- **Composite preference types**
  - neg
  - and
  - pareto
  - lexico

- See *Potassco Guide* on how to define further types
asprin’s library

- Basic preference types
  - subset and superset
  - less(cardinality) and more(cardinality)
  - less(weight) and more(weight)
  - aso (Answer Set Optimization)
  - poset (Qualitative Preferences)

- Composite preference types
  - neg
  - and
  - pareto
  - lexico

See Potassco Guide on how to define further types
asprin’s library

- Basic preference types
  - subset and superset
  - less(cardinality) and more(cardinality)
  - less(weight) and more(weight)
  - aso (Answer Set Optimization)
  - poset (Qualitative Preferences)

- Composite preference types
  - neg
  - and
  - pareto
  - lexico

- See Potassco Guide on how to define further types
Outline

72 Motivation
73 The asprin framework
74 Preliminaries
75 Language
76 Implementation
77 Summary
asprin stands for “ASP for Preference handling”
asprin is a general, flexible, and extendable framework for preference handling in ASP
asprin caters to
- off-the-shelf users using the preference relations in asprin’s library
- preference engineers customizing their own preference relations
asprin stands for “ASP for Preference handling”

asprin is a general, flexible, and extendable framework for preference handling in ASP

asprin caters to
  - off-the-shelf users using the preference relations in asprin’s library
  - preference engineers customizing their own preference relations
asprin stands for “ASP for Preference handling”

asprin is a general, flexible, and extendable framework for preference handling in ASP

asprin caters to
- off-the-shelf users using the preference relations in asprin’s library
- preference engineers customizing their own preference relations
Grounding: Overview

78 Background
79 Bottom Up Grounding
80 Semi-naive Evaluation Based Grounding
81 On-the-fly Simplifications
82 Rule Instantiation
Outline

- Background
- Bottom Up Grounding
- Semi-naive Evaluation Based Grounding
- On-the-fly Simplifications
- Rule Instantiation
- some grounders (in chronological order)
  - lparse (grounding using domain predicates)
  - dlv (semi-naive evaluation based grounding)
  - gringo (semi-naive evaluation based since version 3)
Background

Hamiltonian Cycle Instance

% vertices
node(a). node(b).
node(c). node(d).

% edges
edge(a,b). edge(a,c).
edge(b,c). edge(b,d).
edge(c,a). edge(c,d).
edge(d,a).

% starting point (for presentation purposes)
start(a).
Hamiltonian Cycle Encoding

% generate path
path(X, Y) :- not omit(X, Y), edge(X, Y).
omit(X, Y) :- not path(X, Y), edge(X, Y).

% at most one incoming/outgoing edge
:- path(X, Y), path(X', Y), X < X'.
:- path(X, Y), path(X, Y'), Y < Y'.

% at least one incoming/outgoing edge
on_path(Y) :- path(X, Y), path(Y, Z).
:- node(X), not on_path(X).

% connectedness
reach(X) :- start(X).
reach(Y) :- reach(X), path(X, Y).
:- node(X), not reach(X).
Grounding

- **Safety**
  - each variable has to occur in a positive body element
  - consider: \( p(X) :- \text{not } q(X) \).

- **Herbrand universe**
  - all constants in program and
  - all functions over function symbols in program

- **Herbrand base**
  - all atoms over predicates in program
  - with terms from Herbrand universe

- **Instance of a rule**
  - all variables replaced with elements from Herbrand universe

- **Grounding of a program**
  - \( ground(P) \) is the union of all instances of rules in \( P \)
Example: Size of Grounding

% Herbrand Universe: \{a,b,c,d\}
12 facts from instance
% path(X,Y) :- not omit(X,Y), edge(X,Y).
% omit(X,Y) :- not path(X,Y), edge(X,Y).
% reach(Y) :- reach(X), path(X,Y).
16 rules + 16 rules + 16 rules
% on_path(Y) :- path(X,Y), path(Y,Z).
% :- path(X,Y), path(X',Y), X < X'.
% :- path(X,Y), path(X,Y'), Y < Y'.
64 rules + 64 rules + 64 rules
% reach(X) :- start(X).
% :- node(X), not on_path(X).
% :- node(X), not reach(X).
4 rules + 4 rules + 4 rules
Example: Unnecessary Rules I

\[
\begin{align*}
\text{path}(X,Y) & : - \text{not} \text{ omit}(X,Y), \text{edge}(X,Y). \\
\text{path}(a,a) & : - \text{not} \text{ omit}(a,a), \text{edge}(a,a). \\
\text{path}(a,b) & : - \text{not} \text{ omit}(a,b), \text{edge}(a,b). \\
\text{path}(a,c) & : - \text{not} \text{ omit}(a,c), \text{edge}(a,c). \\
\text{path}(a,d) & : - \text{not} \text{ omit}(a,d), \text{edge}(a,d). \\
\vdots \\
\text{path}(d,a) & : - \text{not} \text{ omit}(d,a), \text{edge}(d,a). \\
\text{path}(d,b) & : - \text{not} \text{ omit}(d,b), \text{edge}(d,b). \\
\text{path}(d,c) & : - \text{not} \text{ omit}(d,c), \text{edge}(d,d). \\
\text{path}(d,d) & : - \text{not} \text{ omit}(d,d), \text{edge}(d,d). 
\end{align*}
\]
Example: Unnecessary Rules II

\% :- path(X,Y), path(X',Y), X < X'.
:- path(a,a), path(a,a), a < a.
:- path(a,b), path(a,b), a < a.
:- path(a,c), path(a,c), a < a.
:- path(a,d), path(a,d), a < a.

:- path(a,a), path(b,a), a < b.
:- path(a,b), path(b,b), a < b.
:- path(a,c), path(b,c), a < b.
:- path(a,d), path(b,d), a < b.

:- path(d,d), path(d,d), d < d.
Bottom Up Grounding

- ground relevant rules by incrementally extending the Herbrand base
- \( \text{ground}_D(P) = \{ r \in \text{ground}(P) \mid B(r)^+ \subseteq D, \text{all comparison literals in } \text{body}(r) \text{ are satisfied} \} \)

```python
function GROUND_BOTTOM_UP(P, D)
    G ← ground_D(P)
    if head(G) ⊈ D then
        return GROUND_BOTTOM_UP(P, D ∪ head(G))
    return G
```

- given safe program \( P \) and set of ground facts \( I \) (typically corresponds to encoding and instance), \( P \cup I \) is equivalent to \( \text{GROUND_BOTTOM_UP}(P, \text{head}(I)) \cup I \)
Example: Bottom Up Grounding Step 1

% Step 1
path(a,b) :- not omit(a,b), edge(a,b).

% 7 rules total
path(d,a) :- not omit(d,a), edge(d,a).

omit(a,b) :- not path(a,b), edge(a,b).

% 7 rules total
omit(d,a) :- not path(d,a), edge(d,a).

:- node(a), not on_path(a). :- node(b), not on_path(b).
:- node(c), not on_path(c). :- node(d), not on_path(d).

:- node(a), not reach(a). :- node(b), not reach(b).
:- node(c), not reach(c). :- node(d), not reach(d).

reach(a) :- start(a).
Example: Bottom Up Grounding Step 2

% Step 2 and rules of Step 1
:- path(a,c), path(b,c), a < b.
:- path(b,d), path(c,d), b < c.
:- path(c,a), path(d,a), c < d.

:- path(a,b), path(a,c), b < c.
:- path(c,a), path(c,d), a < d.
:- path(b,c), path(b,d), c < d.

on_path(a) :- path(a,b), path(c,a).

on_path(d) :- path(d,a), path(c,d).

reach(b) :- reach(a), path(a,b).
reach(c) :- reach(a), path(a,c).
Example: Bottom Up Grounding Step 3 and 4

% Step 3 and rules of Step 2
reach(c) :- reach(b), path(b,c).
reach(d) :- reach(b), path(b,d).
reach(a) :- reach(c), path(c,a).
reach(d) :- reach(c), path(c,d).

% Step 4 and rules of Step 3
reach(a) :- reach(d), path(d,a).
Properties of Bottom Up Grounding

- grounds only relevant rules
  - each positive body literal has a non-cyclic derivation (ignoring negative literals)
- regrounds rules from previous steps

```plaintext
function GROUND_BOTTOM_UP(P, D)
    G ← ground_D(P)
    if head(G) ⊈ D then
        return GROUND_BOTTOM_UP(P, D ∪ head(G))
    return G
```

- does not perform simplifications
Improving Bottom Up Grounding

- use dependencies to **focus** grounding
  - begin with partial Herbrand base given by facts
  - use rule dependency graph of program to obtain **components** that can be grounded successively
- adapt semi-naive evaluation put forward in the database field
  - avoids redundancies when grounding
- perform **simplifications** during grounding
  - remove literals from rule bodies if possible
  - omit rules if body cannot be satisfied
Program Dependencies

- dependency graph of program $P$
  - rule $r_2$ depends on rule $r_1$
    - if $b \in B(r_2)^+ \cup B(r_2)^-$ unifies with $h \in head(r_1)$
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$

- positive dependency graph of program $P$
  - rule $r_2$ positively depends on rule $r_1$
    - if $b \in B(r_2)^+$ unifies with $h \in head(r_1)$
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$

- let $L_P = (C_{1,1}, \ldots, C_{1,m_1}, \ldots, C_{n,1}, \ldots, C_{n,m_n})$ where
  - $(C_1, \ldots, C_n)$ is a topological ordering of $G_P$
  - $(C_{i,1}, \ldots, C_{i,m_i})$ is a topological ordering of each $G_{C_i}^+$
Example: Dependencies

Component\_1,\_1: \texttt{omit(X,Y) :- not path(X,Y), edge(X,Y)}.  
Component\_1,\_2: \texttt{path(X,Y) :- not omit(X,Y), edge(X,Y)}.  
Component\_2,\_1: \texttt{:- path(X,Y), path(X',Y), X < X'}.  
Component\_3,\_1: \texttt{:- path(X,Y), path(X,Y'), Y < Y'}.  
Component\_4,\_1: \texttt{on\_path(Y) :- path(X,Y), path(Y,Z)}.  
Component\_5,\_1: \texttt{:- node(X), not on\_path(X)}.  
Component\_6,\_1: \texttt{reach(X) :- start(X)}.  
Component\_7,\_1: \texttt{reach(Y) :- reach(X), path(X,Y)}.  
Component\_8,\_1: \texttt{:- node(X), not reach(X)}. 
function GROUND_WITH_DEPENDENCIES(P, D)
  G ← ∅
  foreach C in L_P do
    G′ ← GROUND_BOTTOM_UP(C, D)
    (G, D) ← (G ∪ G′, D ∪ head(G′))
  return G

given safe program P and set of facts I, P ∪ I is equivalent to
GROUND_WITH_DEPENDENCIES(P, head(I)) ∪ I
Example: Grounding with Dependencies

% Component\textsubscript{1,1}
\begin{align*}
\text{omit}(a,b) & : \neg \text{path}(a,b), \text{edge}(a,b). \\
\end{align*}
\begin{itemize}
\item \% 7 rules total
\end{itemize}
\begin{align*}
\text{omit}(d,a) & : \neg \text{path}(d,a), \text{edge}(d,a). \\
\end{align*}

% Component\textsubscript{1,2}
\begin{align*}
\text{path}(a,b) & : \neg \text{omit}(a,b), \text{edge}(a,b). \\
\end{align*}
\begin{itemize}
\item \% 7 rules total
\end{itemize}
\begin{align*}
\text{path}(d,a) & : \neg \text{omit}(d,a), \text{edge}(d,a). \\
\end{align*}

...  

- no regrounding if there is no positive recursion in a component
Example: Grounding Component

% Step 1
reach(b) :- reach(a), path(a,b).
reach(c) :- reach(a), path(a,c).

% Step 2 and rules of Step 1
reach(c) :- reach(b), path(b,c).
reach(d) :- reach(b), path(b,d).
reach(a) :- reach(c), path(c,a).
reach(d) :- reach(c), path(c,d).

% Step 3 and rules of Step 2
reach(a) :- reach(d), path(d,a).

% less regrounding but still...
Outline

78 Background
79 Bottom Up Grounding
80 Semi-naive Evaluation Based Grounding
81 On-the-fly Simplifications
82 Rule Instantiation
Recursive Atoms

given $L_P = (C_1, \ldots, C_n)$, an atom $a_1$ is recursive in component $C_i$ if $a_1$ unifies $a_2$ such that
- $r_1 \in C_i$ and $r_2 \in C_j$ with $i \leq j$,
- $a_1 \in B(r_1)^+ \cup B(r_1)^-$, and
- $a_2 \in head(r_2)$
Example: Recursive Atoms

Component 1,1:
\[ \text{omit}(X, Y) \leftarrow \neg \text{path}(X, Y), \text{edge}(X, Y). \]

Component 1,2:
\[ \text{path}(X, Y) \leftarrow \neg \text{omit}(X, Y), \text{edge}(X, Y). \]

Component 2,1:
\[ \text{path}(X, Y), \text{path}(X', Y), X < X'. \]

Component 3,1:
\[ \text{path}(X, Y), \text{path}(X, Y'), Y < Y'. \]

Component 4,1:
\[ \text{on_path}(Y) \leftarrow \text{path}(X, Y), \text{path}(Y, Z). \]

Component 5,1:
\[ \neg \text{node}(X), \neg \text{on_path}(X). \]

Component 6,1:
\[ \text{reach}(X) \leftarrow \text{start}(X). \]

Component 7,1:
\[ \text{reach}(Y) \leftarrow \text{reach}(X), \text{path}(X, Y). \]

Component 8,1:
\[ \neg \text{node}(X), \neg \text{reach}(X). \]
Preparing Components

- the set of prepared rules for \( r \in C \) is

\[
\begin{align*}
&h : - n(b_1), a(b_2), a(b_3), \ldots, a(b_{i-2}), a(b_{i-1}), a(b_i), B \\
&h : - o(b_1), n(b_2), a(b_3), \ldots, a(b_{i-2}), a(b_{i-1}), a(b_i), B \\
&\vdots \quad \vdots \\
&h : - o(b_1), o(b_2), o(b_3), \ldots, o(b_{i-2}), n(b_{i-1}), a(b_i), B \\
&h : - o(b_1), o(b_2), o(b_3), \ldots, o(b_{i-2}), o(b_{i-1}), n(b_i), B \\
\text{or } \{h : - n(b_{i+1}), \ldots, n(b_j), b_{j+1}, \ldots, b_n\} \text{ if } i = 0
\end{align*}
\]

where \( \text{body}(r) = \{b_1, \ldots, b_i, b_{i+1}, \ldots, b_j, b_{j+1}, \ldots, b_n\} \),

- \( b_k \in B(r)^+ \) for \( 1 \leq k \leq i \) is recursive,
- \( b_k \in B(r)^+ \) for \( i < k \leq j \) is not recursive, and
- \( B = a(b_{i+1}), \ldots, a(b_j), b_{j+1}, \ldots, b_n \)

- a prepared component is the union of all its prepared rules
Example: Preparing Components

% prepared Component 1,1
omit(X,Y) :- n(edge(X,Y)), not path(X,Y).
% prepared Component 1,2
path(X,Y) :- n(edge(X,Y)), not omit(X,Y).
% prepared Component 2,1
    :- n(path(X,Y)), n(path(X',Y)), X < X'.
...
% prepared Component 7,1
reach(Y) :- n(reach(X)), a(path(X,Y)).
...
function GROUND_SEMI_NAIVE(P, A)
    G ← ∅
    foreach C in L_P do
        (O, N) ← (∅, A)
        repeat
            let \(D_p = \{p(a) \mid a \in D\}\) for set \(D\) of atoms
            \(G' \leftarrow \text{ground}_{O_o \cup N_n \cup A_a}(\text{prepared } C)\)
            \(N \leftarrow \text{head}(G') \setminus A\)
            \((G, O, A) \leftarrow (G \cup G', A, N \cup A)\)
        until \(N = ∅\)
    return \(G\) with \(o/1, n/1, a/1\) stripped from positive bodies

given safe program \(P\) and set of facts \(I\), \(P \cup I\) is equivalent to
\(\text{GROUND\_SEMI\_NAIVE}(P, \text{head}(I)) \cup I\)
Example: Grounding Component

% grounding of
% reach(Y) :- n(reach(X)), a(path(X,Y)).

% Step 1 with N = A from previous step (reach(a) ∈ A)
reach(b) :- n(reach(a)), a(path(a,b)).
reach(c) :- n(reach(a)), a(path(a,c)).

% Step 2 with N = { reach(b), reach(c) }
reach(c) :- n(reach(b)), a(path(b,c)).
reach(d) :- n(reach(b)), a(path(b,d)).
reach(a) :- n(reach(c)), a(path(c,a)).
reach(d) :- n(reach(c)), a(path(c,d)).

% Step 3 with N = { reach(d) }
reach(a) :- n(reach(d)), a(path(d,a)).
Example: Grounding Component

% grounding of
% reach(Y) :- n(reach(X)), a(path(X,Y)).

% Step 1 with N = A from previous step (reach(a) ∈ A)
reach(b) :- reach(a), path(a,b).
reach(c) :- reach(a), path(a,c).

% Step 2 with N = { reach(b), reach(c) }
reach(c) :- reach(b), path(b,c).
reach(d) :- reach(b), path(b,d).
reach(a) :- reach(c), path(c,a).
reach(d) :- reach(c), path(c,d).

% Step 3 with N = { reach(d) }
reach(a) :- reach(d), path(d,a).

% without n/1 and a/1 of course
Example: Nonlinear Programs

trans(U,V) :- edge(U,V).
trans(U,W) :- trans(U,V), trans(V,W).

% prepared Component 1:
trans(U,V) :- n(edge(U,V)).

% prepared Component 2:
trans(U,W) :- n(trans(U,V)), a(trans(V,W)).
trans(U,W) :- o(trans(U,V)), n(trans(V,W)).
Example: Nonlinear Programs

\[
\text{trans}(U,V) :- \text{edge}(U,V).
\]
\%
\[
\text{trans}(U,W) :- \text{trans}(U,V), \text{trans}(V,W).
\]
\%
better written as:
\[
\text{trans}(U,W) :- \text{trans}(U,V), \text{edge}(V,W).
\]

\%
prepared Component 1:
\[
\text{trans}(U,V) :- \text{n(edge}(U,V)).
\]

\%
prepared Component 2:
\[
\text{trans}(U,W) :- \text{n(trans}(U,V)), \text{a(edge}(V,W)).
\]
Outline

78 Background
79 Bottom Up Grounding
80 Semi-naive Evaluation Based Grounding
81 On-the-fly Simplifications
82 Rule Instantiation
On-the-fly Simplifications

Propagation of Facts

- simplifications are performed on-the-fly (rules are printed immediately but not stored in gringo)
- maintain a set of fact atoms
- remove facts from positive body
- discard rules with negative literals over a fact
- discard rules whenever the head is a fact
- gather new facts whenever a rule body is empty
Example: Propagation of Facts

\[\text{path}(a,b) :- \text{not omi}(a,b), \text{edge}(a,b).\]
\[\text{reach}(a) :- \text{start}(a).\]
Example: Propagation of Facts

... path(a,b) :- not omit(a,b).
...
reach(a). % reach(a) is added as fact
Example: Propagation of Facts

... path(a,b) :- not omit(a,b).
...
reach(a). % reach(a) is added as fact
...

:- node(a), not reach(a).
...
Example: Propagation of Facts

... path(a,b) :- not omit(a,b).
...
reach(a). % reach(a) is added as fact
...
:- node(a), not reach(a). % rule is discarded
...
Propagating Negative Literals

- non-recursive negative literals not in the current base $A$ can be removed from rule bodies
- stratified logic programs are completely evaluated during grounding
- consider the instance where node $d$ is not reachable
Example: Propagation of Negative Literals

\[
\begin{align*}
\text{path}(a,b) & : \neg \text{omit}(a,b). \\
\text{path}(a,c) & : \neg \text{omit}(a,c). \\
\text{path}(b,c) & : \neg \text{omit}(b,c). \\
\text{path}(c,a) & : \neg \text{omit}(c,a). \\
\text{path}(d,a) & : \neg \text{omit}(d,a). \\
\vdots \\
\text{reach}(a). \\
\text{reach}(b) & : \text{path}(a,b). \\
\text{reach}(c) & : \text{path}(a,c). \\
\text{reach}(d) & : \text{path}(b,c), \text{reach}(b). \\
\vdots \\
% \text{reach}(X) \text{ is not recursive and reach}(d) & \notin A \\
& : \neg \text{reach}(b). \\
& : \neg \text{reach}(c). \\
& : \neg \text{reach}(d). \ % \text{remove } \neg \text{reach}(d) \text{ from body}
\end{align*}
\]
Example: Propagation of Negative Literals

path(a,b) :- not omit(a,b).
path(a,c) :- not omit(a,c).
path(b,c) :- not omit(b,c).
path(c,a) :- not omit(c,a).
path(d,a) :- not omit(d,a).
...
reach(a).
reach(b) :- path(a,b).
reach(c) :- path(a,c).
reach(c) :- path(b,c), reach(b).
...
% reach(X) is not recursive and reach(d) \notin A
:- not reach(b).
:- not reach(c).
:- . % inconsistency detected during grounding
On-the-fly Simplifications

Conclusion/Summary

- grounding algorithms for normal logic programs (with integrity constraints)
- language features not covered here
  - (recursive) aggregates
  - conditional literals
  - optimization statements
  - disjunctions
  - arithmetic functions
  - syntactic sugar to write more compact encodings
  - safety of = relation (for aggregates and terms)
  - python/lua integration
    - external functions
    - control over grounding and solving
Outline

78 Background
79 Bottom Up Grounding
80 Semi-naive Evaluation Based Grounding
81 On-the-fly Simplifications
82 Rule Instantiation
the following slides show how to ground individual rules

I am probably not going to show them
given safe rule \( r \), the tuple \((b_1, \ldots, b_n)\) is a safe body order if

- \( \{b_1, \ldots, b_n\} = \text{body}(r) \)
- the body \( \{b_1, \ldots, b_i\} \) is safe for each \( i \)

for example given rule \(:= \text{node}(X), \text{not reach}(X)\).

- \( (\text{node}(X), \text{not reach}(X)) \) is a safe body order
- \( (\text{not reach}(X), \text{node}(X)) \) is not a safe body order
given safe rule \( r \), the tuple \( (b_1, \ldots, b_n) \) is a safe body order if
- \( \{b_1, \ldots, b_n\} = \text{body}(r) \)
- the body \( \{b_1, \ldots, b_i\} \) is safe for each \( i \)

for example given rule \( :- \text{node}(X), \neg \text{reach}(X) \).
- \( (\text{node}(X), \neg \text{reach}(X)) \) is a safe body order
- \( (\neg \text{reach}(X), \text{node}(X)) \) is not a safe body order
Rule Instantiation

Matching Body Literals

- $\text{match}_{F,D}(\sigma, b)$ is the set of all matches for literal $b$
  - $\sigma$ is a substitution
  - $F$ are facts (set of ground atoms)
  - $D$ is the domain (set of ground atoms)
  - $\sigma' \in \text{match}_{F,D}(\sigma, b)$ if
    - $\sigma \subseteq \sigma'$ and $\text{vars}(b) \subseteq \text{vars}(\sigma') \subseteq \text{vars}(b) \cup \text{vars}(\sigma)$,
    - $b\sigma'$ holds if $b$ is a comparison literal,
    - $b\sigma' \in D$ if $b$ is an atom, and
    - $a\sigma' \notin F$ if $b$ is a symbolic literal of form $\text{not } a$

For example given body: $p(X), q(X,Y), \text{not } r(Y)$

- $F = \{r(3)\}$ and $D = \{p(1), q(1,2), q(1,3), r(3)\}$
- $\text{match}_{F,D}(\emptyset, p(X)) = \{\{X \mapsto 1\}\}$
- $\text{match}_{F,D}(\{X \mapsto 1\}, q(X,Y)) = \{\{X \mapsto 1, Y \mapsto 2\}, \{X \mapsto 1, Y \mapsto 3\}\}$
- $\text{match}_{F,D}(\{X \mapsto 1, Y \mapsto 2\}, \text{not } r(Y)) = \{\{X \mapsto 1, Y \mapsto 2\}\}$
- $\text{match}_{F,D}(\{X \mapsto 1, Y \mapsto 3\}, \text{not } r(Y)) = \emptyset$
Matching Body Literals

- \( \text{match}_{F,D}(\sigma, b) \) is the set of all matches for literal \( b \)
  - \( \sigma \) is a substitution
  - \( F \) are facts (set of ground atoms)
  - \( D \) is the domain (set of ground atoms)
  - \( \sigma' \in \text{match}_{F,D}(\sigma, b) \) if
    - \( \sigma \subseteq \sigma' \) and \( \text{vars}(b) \subseteq \text{vars}(\sigma') \subseteq \text{vars}(b) \cup \text{vars}(\sigma) \),
    - \( b\sigma' \) holds if \( b \) is a comparison literal,
    - \( b\sigma' \in D \) if \( b \) is an atom, and
    - \( a\sigma' \notin F \) if \( b \) is a symbolic literal of form not \( a \)

- For example given body: \( p(X), q(X,Y), \text{not } r(Y) \)
  - \( F = \{ r(3) \} \) and \( D = \{ p(1), q(1,2), q(1,3), r(3) \} \)
  - \( \text{match}_{F,D}(\emptyset, p(X)) = \{ \{ X \mapsto 1 \} \} \)
  - \( \text{match}_{F,D}(\{ X \mapsto 1 \}, q(X,Y)) = \{ \{ X \mapsto 1, Y \mapsto 2 \}, \{ X \mapsto 1, Y \mapsto 3 \} \} \)
  - \( \text{match}_{F,D}(\{ X \mapsto 1, Y \mapsto 2 \}, \text{not } r(Y)) = \{ \{ X \mapsto 1, Y \mapsto 2 \} \} \)
  - \( \text{match}_{F,D}(\{ X \mapsto 1, Y \mapsto 3 \}, \text{not } r(Y)) = \emptyset \)
Rule Grounding by Backtracking

function GROUND_BACKTRACK\(_r,R,D(\sigma, F, (b_1, \ldots, b_n))\)
    if \(n = 0\) then
        let \(H = head(r\sigma)\)
        \(B = B(r\sigma)^+ \setminus F \cup \{\neg a\sigma \mid a \in B(r)^- \setminus R, a \in D\} \cup \{\neg a\sigma \mid a \in B(r)^- \cap R\}\)
        if \(B = \emptyset\) then \(F \leftarrow F \cup H\)
        return (\(\{H :- B \mid B^- \cap F = \emptyset, H \cap F = \emptyset\}, F\)\)
    else
        \(G \leftarrow \emptyset\)
        foreach \(\sigma' \in match_F,D(\sigma, b_1)\) do
            \((G, F) \leftarrow (G, F) \sqcup GROUND_BACKTRACK\(_r,R,D(\sigma', F, (b_2, \ldots, b_n))\)\)
        return (\(G, F\)\)
ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - Rapid application development tool
- ASP has a growing range of applications
ASP is a viable tool for Knowledge Representation and Reasoning.
ASP offers efficient and versatile off-the-shelf solving technology.
ASP offers an expanding functionality and ease of use:
- Rapid application development tool
ASP has a growing range of applications.

**ASP = DB + LP + KR + SAT**
ASP is a viable tool for Knowledge Representation and Reasoning

- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - Rapid application development tool
- ASP has a growing range of applications

\[
\text{ASP} = \text{DB} + \text{LP} + \text{KR} + \text{SMT}
\]
ASP is a viable tool for Knowledge Representation and Reasoning

- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - Rapid application development tool
- ASP has a growing range of applications

http://potassco.org