

Answer Set Programming

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

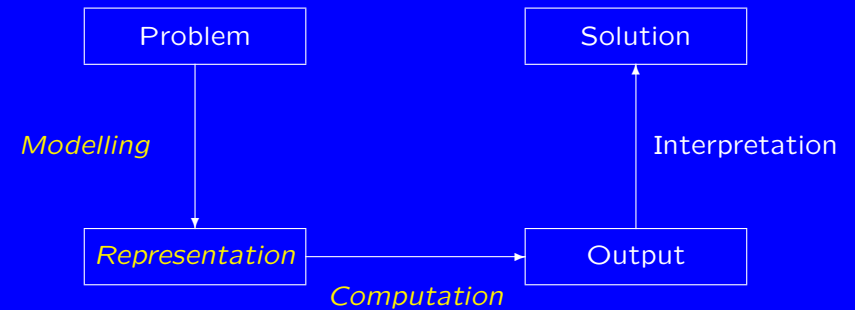
1

Goal: Declarative problem solving

“What is the problem?”

instead of

“How to solve the problem?”



5

Goal: Declarative problem solving (ctd)

Wanted An approach for modelling and solving “typical” problems in intelligent systems design

Desiderata

- Expressive power
- Ease of modelling
 - intuitive semantics
 - concise and transparent encodings
 - modular programming
- Performance

6

Proposal: Answer set programming (ASP)

- has its roots in
 - (logic-based) knowledge representation and reasoning
 - (deductive) databases
 - constraint solving (in particular, SAT solving)
 - logic programming (with negation)
- allows for solving all search problems within NP (and NP^{NP}, resp.) (over finite domains; otherwise Turing-complete)
- allows for using powerful off-the-shelf systems (nowadays capable of dealing with millions of variables)

$$\text{ASP} = \text{KR} + \text{DB} + \text{Search}$$

7

Root: Logic Programming with negation

- Algorithm = Logic + Control (Kowalski, 1979)
- Logic as a programming language
 - Prolog (Colmerauer, Kowalski)
- Features of Prolog
 - Declarative (relational) programming language
 - Based on SLD(NF) Resolution
 - Top-down query evaluation
 - Terms as data structures
 - Parameter passing by unification
 - Solutions are extracted from instantiations of variables occurring in the query

8

Prolog (ctd)

Prolog offers *negation as failure* via operator *not*.

For instance,

```
info(a).
ask(X) :- not info(X).
```

cannot be captured by

$$info(a) \wedge \forall x(\neg info(x) \rightarrow ask(x))$$

but by appeal to *Clark's completion* by

$$\begin{aligned} & \forall x(x = a \leftrightarrow info(x)) \wedge \forall x(\neg info(x) \leftrightarrow ask(x)) \\ \iff & info(a) \wedge \forall x(x \neq a \leftrightarrow ask(x)) \end{aligned}$$

10

Prolog: Programming in logic

Prolog is great, it's *almost* declarative!

To see this, consider

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

and compare it to

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

An interpretation in classical logic amounts to

$$\forall xy(on(x,y) \vee \exists z(on(x,z) \wedge above(z,y)) \rightarrow above(x,y))$$

9

Prolog (ctd)

Clark's completion is sometimes too syntactical.

Consider

- $p :- p.$
yielding $p \leftrightarrow p$ having models \emptyset and $\{p\}$
- $p :- q. \quad q :- p.$
yielding $p \leftrightarrow q$ having models \emptyset and $\{p, q\}$
- $p :- p. \quad q :- \text{not } p.$
yielding $(p \leftrightarrow p) \wedge (q \leftrightarrow \neg p)$ having models $\{q\}$ and $\{p\}$

Or even more complex yet analogous situations.

11

Model-based Problem Solving

Common approach (e.g. Prolog)

1. Provide a specification of the problem.
2. A solution is given by a *derivation* of an appropriate query.

Model-based approach

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

Automated planning, Kautz and Selman, mid 90s:

represent planning problems as propositional theories
so that models not proofs describe solutions (e.g. Satplan)

12

Model-based Problem Solving

Specification	Associated Structures
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models ✗
first-order theories	models
default theories	extensions
...	

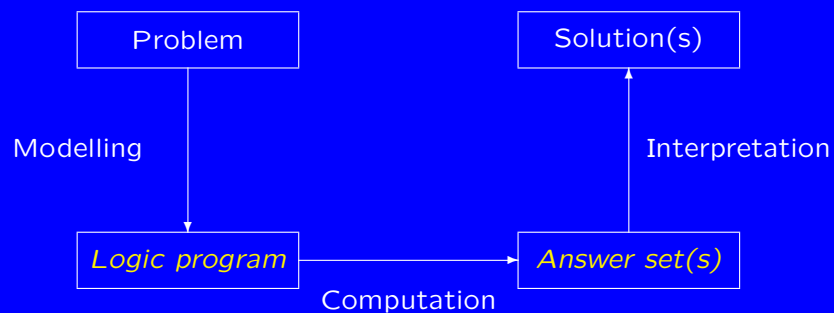
13

Problem solving in ASP I

High-level language

Basic Idea

- Encode problem (class+instance) as a set of rules
- Read off solutions from answer sets of the rules



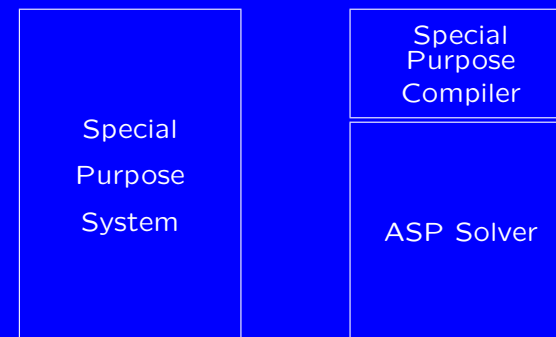
14

Problem solving in ASP II

Low-level language

Basic Idea

- Compile a problem automatically into a logic program
- Solve the original problem by solving its compilation



15

What is ASP good for?

Combinatorial search problems (some with substantial amount of data):

- auctions
- bio-informatics
- computer-aided verification
- configuration
- constraint satisfaction
- diagnosis
- ...
- information integration
- planning and scheduling
- security analysis
- semantic web
- wire-routing
- zoology and linguistics

My favorite: Using ASP as a basis for a decision support system for NASA's space shuttle (Nogueira et al., 01)

☞ See also <http://www.kr.tuwien.ac.at/research/WASP/report.html>

16

Action description languages

Low-level language: E.g. \mathcal{A} (Gelfond & Lifschitz, 1990)

```
move(b,l) causes on(b,l)
inertial on(b,l)
```

is translated into

```
% effect of moving a block
on(B,L,T+1) :- move(B,L,T),
               block(B), location(L), time(T), T<lasttime.

% inertia
on(B,L,T+1) :- on(B,L,T), not neg_on(B,L,T+1),
               location(L), block(B), time(T), T<lasttime.
```

E.g. (Grell, 2006) describes pathways in metabolic networks.

18

Configuration

High-level language: A very simple example

```
1 { disk(ide), disk(scsi), disk(fw) } 2 ← computer(X)
keyboard(us) ∨ keyboard(german) ← computer(X)
controller(scsi) ← disk(scsi)
monitor(15in) ← not other-monitor(15in)
other-monitor(Y) ← monitor(X), X ≠ Y
← monitor(21in) ∧ graphics(evil)

computer(desktop) ←
← controller(scsi)
```

E.g. (Syrjänen, 99) describes a Debian Linux configurator

See also <http://www.tcs.hut.fi/research/logic/WASP/configuration>

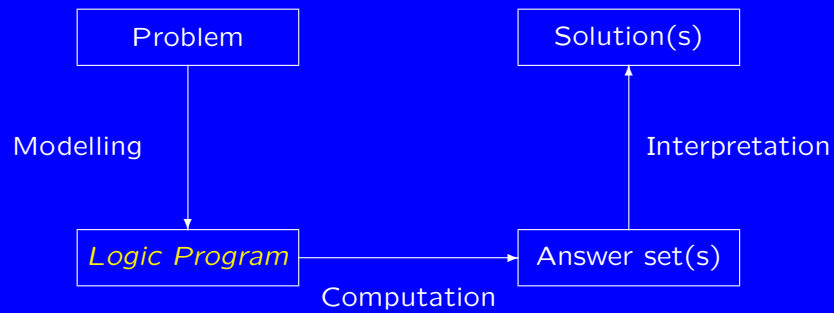
17

Answer Set Programming

- Syntax
- Semantics
- Examples
- Variables and grounding
- Integrity constraints
- Computation

19

Problem solving in ASP: Syntax



20

Normal logic programs

- A (normal) *rule*, r , is an ordered pair of the form

$$A_0 \leftarrow A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n,$$

where $n \geq m \geq 0$, and each A_i ($0 \leq i \leq n$) is an atom.

- A (normal) *logic program* is a finite *set* of rules.
- Notation

$$\text{head}(r) = A_0$$

$$\text{body}(r) = \{A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n\}$$

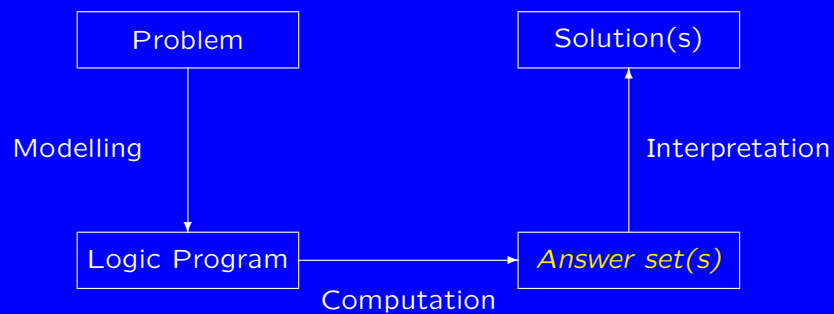
$$\text{body}^+(r) = \{A_1, \dots, A_m\}$$

$$\text{body}^-(r) = \{A_{m+1}, \dots, A_n\}$$

- A program is called *positive* if $\text{body}^-(r) = \emptyset$ for all its rules.

21

Problem solving in ASP: Semantics



22

Answer set: Formal Definition

Positive programs

- A set of atoms X is *closed under* a positive program Π iff for any $r \in \Pi$, $\text{head}(r) \in X$ whenever $\text{body}^+(r) \subseteq X$.
- The *smallest* set of atoms which is closed under a positive program Π is denoted by $C_n(\Pi)$.
- The set $C_n(\Pi)$ of atoms is the *answer set* of a *positive* program Π .

23

Some “logical” remarks

- Positive rules are also referred to as *definite clauses*.
- Definite clauses are disjunctions with exactly one positive atom.
 - ➔ $A_0 \vee \neg A_1 \vee \dots \vee \neg A_m$
- 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.
- A set of Horn clauses has a smallest model or none.
- This smallest model is the intended semantics of a set of Horn clauses.
 - ☞ Given a positive program Π , $C_n(\Pi)$ corresponds to the smallest model of the set of definite clauses corresponding to Π .

24

Answer set: Formal Definition

Normal programs

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

$$\Pi^X = \{ \text{head}(r) \leftarrow \text{body}^+(r) \mid r \in \Pi \text{ and } \text{body}^-(r) \cap X = \emptyset \}.$$

- A set X of atoms is an *answer set* of a program Π if $C_n(\Pi^X) = X$.
Recall: $C_n(\Pi^X)$ is the \subseteq -smallest (classical) model of Π^X .

Intuition: X is *stable* under “applying rules from Π ”

Note: Every atom in X is justified by an “applying rule from Π ”

26

Answer set: Basic idea

- Consider the logical formula Φ and its three (classical) models: $\{a, b\}$, $\{b, c\}$, and $\{a, b, c\}$.
 - ☞ $\Phi \quad b \wedge (b \wedge \neg c \rightarrow a)$
- This formula has one answer set: $\{a, b\}$.
To see why, consider the logic program Π
 - ☞ $\Pi \quad \begin{array}{l} b \leftarrow \\ a \leftarrow b, \text{ not } c \end{array}$
- Informally, a set of atoms X is an *answer set* of a logic program Π
 - if X is a (classical) model of Π and
 - if all atoms in X are *justified* by some rule in Π .
- ☞ logically rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932)

25

A closer look at Π^X

In other words, given a set of atoms X from Π ,

Π^X is obtained from Π by deleting

1. each rule having a *not* A in its body with $A \in X$ and then
2. all negative atoms of the form *not* A in the bodies of the remaining rules.

27

A first example

$$\Pi = \{p \leftarrow p, q \leftarrow \text{not } p\}$$

X	Π	Π^X	$Cn(\Pi^X)$
\emptyset	$p \leftarrow p$ $q \leftarrow \text{not } p$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$ ✗
$\{p\}$	$p \leftarrow p$ $q \leftarrow \text{not } p$	$p \leftarrow p$ $q \leftarrow$	\emptyset ✗
$\{q\}$	$p \leftarrow p$ $q \leftarrow \text{not } p$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$ ✓
$\{p, q\}$	$p \leftarrow p$ $q \leftarrow \text{not } p$	$p \leftarrow p$ $q \leftarrow$	\emptyset ✗

28

A second example

$$\Pi = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

X	Π	Π^X	$Cn(\Pi^X)$
\emptyset	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$ ✗
$\{p\}$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow$ $q \leftarrow$	$\{p\}$ ✓
$\{q\}$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow$ $q \leftarrow$	$\{q\}$ ✓
$\{p, q\}$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow$ $q \leftarrow$	\emptyset ✗

29

A third example

$$\Pi = \{p \leftarrow \text{not } p\}$$

X	Π	Π^X	$Cn(\Pi^X)$
\emptyset	$p \leftarrow \text{not } p$	$p \leftarrow$	$\{p\}$ ✗
$\{p\}$	$p \leftarrow \text{not } p$	$p \leftarrow$	\emptyset ✗

⚠ A program may have zero, one, or multiple answer sets!

30

Answer set: Alternative Definition

Let Π be a normal program and X a set of atoms.

- The set of *generating rules* of X relative to Π is defined by

$$\Pi_X = \{r \in \Pi \mid \text{body}^+(r) \subseteq X \text{ and } \text{body}^-(r) \cap X = \emptyset\}.$$

- X is an answer set of Π iff X is a \subseteq -minimal model of Π_X .
- Or, X is an answer set of Π iff $X \in \min_{\subseteq}(\Pi_X)$, where $\min_{\subseteq}(\Pi)$ is the set of \subseteq -minimal models of a program Π (cf. Footnote ?? on Slide ??).

31

The second example revisited

$$\Pi = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$$

X	Π	Π_X	"logically"	$\min_{\subseteq}(\Pi_X)$
\emptyset	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \vee q$	$\{p\}, \{q\}$ ✗
$\{p\}$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \vee q$	$\{p\}, \{q\}$ ✓
$\{q\}$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$	$p \vee q$	$\{p\}, \{q\}$ ✓
$\{p, q\}$	$p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$			\emptyset ✗

32

A closer look at Cn

Inductive characterisation

Let Π be a positive program and X a set of atoms.

- The *immediate consequence operator* T_{Π} is defined as follows:

$$T_{\Pi}X = \{\text{head}(r) \mid r \in \Pi \text{ and } \text{body}(r) \subseteq X\}$$

- Iterated applications of T_{Π} are written as T_{Π}^j for $j \geq 0$, where $T_{\Pi}^0X = X$ and $T_{\Pi}^iX = T_{\Pi}T_{\Pi}^{i-1}X$ for $i \geq 1$.

Theorem $Cn(\Pi) = \bigcup_{i \geq 0} T_{\Pi}^i \emptyset$ for any positive program Π .

Proposition $X \subseteq Y$ implies $T_{\Pi}X \subseteq T_{\Pi}Y$ for any positive program Π .

➔ In other words, T_{Π} is monotone.

➔ By Tarski's Theorem, $Cn(\Pi)$ is the smallest fixpoint of T_{Π} .

33

Let's iterate T_{Π}

$$\Pi = \{p \leftarrow, q \leftarrow, r \leftarrow p, s \leftarrow q, t, t \leftarrow r, u \leftarrow v\}$$

$$\begin{aligned} T_{\Pi}^0 \emptyset &= \emptyset \\ T_{\Pi}^1 \emptyset &= \{p, q\} = T_{\Pi} T_{\Pi}^0 \emptyset = T_{\Pi}(\emptyset) \\ T_{\Pi}^2 \emptyset &= \{p, q, r\} = T_{\Pi} T_{\Pi}^1 \emptyset = T_{\Pi}(\{p, q\}) \\ T_{\Pi}^3 \emptyset &= \{p, q, r, t\} = T_{\Pi} T_{\Pi}^2 \emptyset = T_{\Pi}(\{p, q, r\}) \\ T_{\Pi}^4 \emptyset &= \{p, q, r, t, s\} = T_{\Pi} T_{\Pi}^3 \emptyset = T_{\Pi}(\{p, q, r, t\}) \\ T_{\Pi}^5 \emptyset &= \{p, q, r, t, s\} = T_{\Pi} T_{\Pi}^4 \emptyset = T_{\Pi}(\{p, q, r, t, s\}) \\ T_{\Pi}^6 \emptyset &= \{p, q, r, t, s\} = T_{\Pi} T_{\Pi}^5 \emptyset = T_{\Pi}(\{p, q, r, t, s\}) \end{aligned}$$

To see that $Cn(\Pi) = \{p, q, r, t, s\}$ is the smallest fixpoint of T_{Π} , note that $T_{\Pi}\{p, q, r, t, s\} = \{p, q, r, t, s\}$ and $T_{\Pi}X \neq X$ for every $X \subseteq \{p, q, r, t, s\}$.

34

Programs with Variables

Let Π be a logic program.

Herbrand universe U^{Π} : Set of constants in Π

Herbrand base B^{Π} : Set of (variable-free) atoms constructible from U^{Π}
 We usually denote this as \mathcal{A} .

Ground instances of $r \in \Pi$: Set of variable-free rules obtained by replacing all variables in r by elements from U^{Π} :

$$\text{ground}(r) = \{r\theta \mid \theta : \text{var}(r) \rightarrow U^{\Pi}\}$$

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

Ground instantiation of Π

$$\text{ground}(\Pi) = \{\text{ground}(r) \mid r \in \Pi\}$$

35

An example

$$\Pi = \{ r(a,b) \leftarrow, r(b,c) \leftarrow, t(X,Y) \leftarrow r(X,Y) \}$$

$$U^\Pi = \{a, b, c\}$$

$$B^\Pi = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$\text{ground}(\Pi) = \left\{ \begin{array}{l} r(a,b) \leftarrow, \\ r(b,c) \leftarrow, \\ t(a,a) \leftarrow r(a,a), \quad t(b,a) \leftarrow r(b,a), \quad t(c,a) \leftarrow r(c,a), \\ t(a,b) \leftarrow r(a,b), \quad t(b,b) \leftarrow r(b,b), \quad t(c,b) \leftarrow r(c,b), \\ t(a,c) \leftarrow r(a,c), \quad t(b,c) \leftarrow r(b,c), \quad t(c,c) \leftarrow r(c,c) \end{array} \right\}$$

36

Answer sets of programs with Variables

Let Π be a normal logic program with variables.

We define a set X of (*ground*) atoms as an *answer set* of Π if $Cn(\text{ground}(\Pi)^X) = X$.

37

Programs with Integrity Constraints

Purpose Integrity constraints eliminate unwanted candidate solutions

Syntax An integrity constraints is of the form

$$\leftarrow A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n,$$

where $n \geq m \geq 1$, and each A_i ($1 \leq i \leq n$) is a atom.

Example $\leftarrow \text{monitor}(21\text{in}), \text{graphics}(\text{evil})$

Implementation For a new symbol x , map

$$\begin{aligned} &\leftarrow A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n \\ \mapsto &x \leftarrow A_1, \dots, A_m, \text{not } A_{m+1}, \dots, \text{not } A_n, \text{not } x \end{aligned}$$

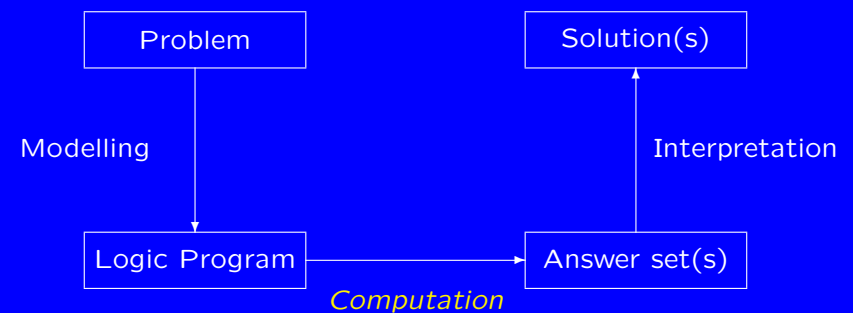
Another example $\Pi = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p\}$

versus $\Pi' = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p, \leftarrow p\}$

versus $\Pi'' = \{p \leftarrow \text{not } q, q \leftarrow \text{not } p, \leftarrow \text{not } p\}$

38

Problem solving in ASP: Computation



39

Standard Computation Scheme

Global parameters: Logic program Π and its set of atoms \mathcal{A} .

$answerset_{\Pi}(X, Y)$

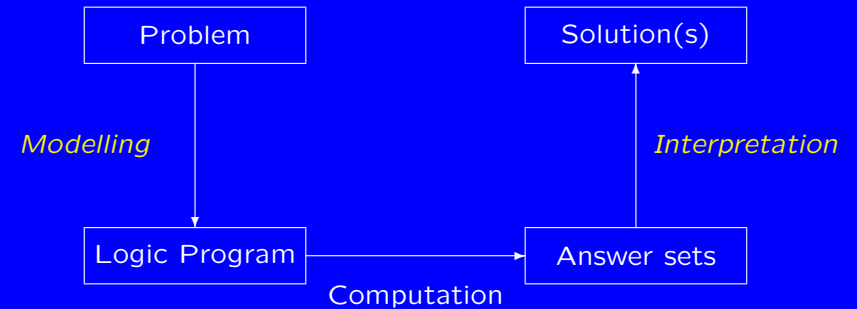
1. $(X, Y) \leftarrow propagation_{\Pi}(X, Y)$
2. **if** $(X \cap Y) \neq \emptyset$ **then fail**
3. **if** $(X \cup Y) = \mathcal{A}$ **then return**(X)
4. **select** $A \in \mathcal{A} \setminus (X \cup Y)$
5. $answerset_{\Pi}(X \cup \{A\}, Y)$
6. $answerset_{\Pi}(X, Y \cup \{A\})$

Comments:

- (X, Y) is supposed to be a 3-valued model such that $X \subseteq Z$ and $Y \cap Z = \emptyset$ for any answer set Z of Π .
- Key operations: $propagation_{\Pi}(X, Y)$ and '**select** $A \in \mathcal{A} \setminus (X \cup Y)$ '
- Worst case complexity: $\mathcal{O}(2^{|\mathcal{A}|})$

40

Modelling and Interpreting



41

Problem \mapsto Logic Program

For solving a problem class P for a problem instance I ,
encode

1. the problem instance I as a set of facts $C(I)$ and
2. the problem class P as a set of rules $C(P)$,

such that the solutions to P for I can be (polynomially) extracted
from the answer sets of $C(P) \cup C(I)$.

42

n -colorability of graphs

Problem instance A graph (V, E) .

Problem class Assign each vertex in V one of n colours such that no
two vertexes in V connected by an edge in E have the same color.

43

3-colorability of graphs

<i>C(I)</i>	vertex(1) ← edge(1,2) ← vertex(2) ← edge(2,3) ← vertex(3) ← edge(3,1) ←
<i>C(P)</i>	colored(V,r) ← not colored(V,b), not colored(V,g), vertex(V) colored(V,b) ← not colored(V,r), not colored(V,g), vertex(V) colored(V,g) ← not colored(V,r), not colored(V,b), vertex(V) ← edge(V,U), colored(V,C), colored(U,C), color(C)
Answer set	{ colored(1,r), colored(2,b), colored(3,g), ... }

44

n-colorability of graphs

with $n = 3$

<i>C(I)</i>	vertex(1) ← edge(1,2) ← vertex(2) ← edge(2,3) ← vertex(3) ← edge(3,1) ←
<i>C(P)</i>	color(r) ← color(b) ← color(g) ← colored(V,C) ← not othercolor(V,C), vertex(V), color(C) othercolor(V,C) ← colored(V,C'), $C \neq C'$, vertex(V), color(C), color(C') ← edge(V,U), colored(V,C), colored(U,C), color(C)
Answer set	{ colored(1,r), colored(2,b), colored(3,g), ... }

45

n-colorability of graphs

with $n = 3$

C(I) vertex(1). vertex(2). vertex(3).
edge(1,2). edge(2,3). edge(3,1).

C(P) color(r). color(b). color(g).

```
colored(V,C) :- not othercolor(V,C),
               vertex(V),color(C).
othercolor(V,C) :- colored(V,C1), C != C1,
                  vertex(V),color(C),color(C1).
                :- edge(V,U),color(C),
                  colored(V,C),colored(U,C).
```

46

Let it run!

```
torsten@belle-ile 507 > lparse 3color.lp | smodels 0
```

```
smodels version 2.25. Reading...done
```

```
Answer: 1
```

```
Stable Model: colored(3,g) othercolor(2,g) othercolor(1,g)
othercolor(3,b) colored(2,b) othercolor(1,b) othercolor(3,r)
othercolor(2,r) colored(1,r) color(g) color(b) color(r)
edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2) vertex(1)
```

47

Here's the rest!

```
Answer: 2
Stable Model: colored(3,g) othercolor(2,g) othercolor(1,g) othercolor(3,b)
othercolor(2,b) colored(1,b) othercolor(3,r) colored(2,r) othercolor(1,r)
color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2)
vertex(1)
Answer: 3
Stable Model: othercolor(3,g) colored(2,g) othercolor(1,g) colored(3,b)
othercolor(2,b) othercolor(1,b) othercolor(3,r) othercolor(2,r) colored(1,r)
color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2)
vertex(1)
Answer: 4
Stable Model: othercolor(3,g) othercolor(2,g) colored(1,g) colored(3,b)
othercolor(2,b) othercolor(1,b) othercolor(3,r) colored(2,r) othercolor(1,r)
color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2)
vertex(1)
Answer: 5
Stable Model: othercolor(3,g) colored(2,g) othercolor(1,g) othercolor(3,b)
othercolor(2,b) colored(1,b) colored(3,r) othercolor(2,r) othercolor(1,r)
color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2)
vertex(1)
Answer: 6
Stable Model: othercolor(3,g) othercolor(2,g) colored(1,g) othercolor(3,b)
colored(2,b) othercolor(1,b) colored(3,r) othercolor(2,r) othercolor(1,r)
color(g) color(b) color(r) edge(3,1) edge(2,3) edge(1,2) vertex(3) vertex(2)
vertex(1)
False
```

48

Basic Methodology

Generate and Test (or: Guess and Check) approach

Generator Generate potential candidates answer sets
(typically through non-deterministic constructs)

Tester Eliminate non-valid Candidates
(typically through integrity constraints)

50

And finally some statistics!

```
Duration: 0.010
Number of choice points: 5
Number of wrong choices: 5
Number of atoms: 28
Number of rules: 45
Number of picked atoms: 42
Number of forced atoms: 0
Number of truth assignments: 347
Size of searchspace (removed): 9 (0)
```

49

Satisfiability

Problem instance A propositional formula ϕ .

Problem class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true.

51

Satisfiability

Consider formula $(a \vee \neg b) \wedge (\neg a \vee b)$.

Generator

$a \leftarrow \text{not } \hat{a}$
 $\hat{a} \leftarrow \text{not } a$
 $b \leftarrow \text{not } \hat{b}$
 $\hat{b} \leftarrow \text{not } b$

Tester

$\leftarrow \text{not } a, b$
 $\leftarrow a, \text{not } b$

Answer set

$A_1 = \{a, b\}$
 $A_2 = \{\hat{a}, \hat{b}\}$

52

n -Queens Problem

A solution to $n = 4$:

	Q		
			Q
Q			
		Q	

53

n -Queens in answer set programming

$q(X, Y)$ gives the legal positions of the queens^a

$q(X, Y) \leftarrow \text{not } \neg q(X, Y)$
 $\neg q(X, Y) \leftarrow \text{not } q(X, Y)$
 $\leftarrow q(X, Y), q(X', Y), X \neq X'$
 $\leftarrow q(X, Y), q(X, Y'), Y \neq Y'$
 $\leftarrow q(X, Y), q(X', Y'), |X - X'| = |Y - Y'|, X \neq X', Y \neq Y'$
 $\leftarrow \text{not } \text{hasq}(X)$
 $\text{hasq}(X) \leftarrow q(X, Y)$

^aregard $\neg q(X, Y)$ as an independent auxiliary atom

54

n -Queens

(in the `smodels` language)

```
q(X,Y) :- d(X), d(Y), not negq(X,Y).
negq(X,Y) :- d(X), d(Y), not q(X,Y).

:- d(X), d(Y), d(X1), q(X,Y), q(X1,Y), X1 != X.
:- d(X), d(Y), d(Y1), q(X,Y), q(X,Y1), Y1 != Y.
:- d(X), d(Y), d(X1), d(Y1), q(X,Y), q(X1,Y1),
   X != X1, Y != Y1, abs(X - X1) == abs(Y - Y1).

:- d(X), not hasq(X).
hasq(X) :- d(X), d(Y), q(X,Y).

d(1..queens).
```

55

n-Queens in answer set programming

(in *disjunctive* logic programming)

```
q(X,Y) ∨ ¬q(X,Y) ←
    ← q(X,Y), q(X',Y), X ≠ X'
    ← q(X,Y), q(X,Y'), Y ≠ Y'
    ← q(X,Y), q(X',Y'), |X - X'| = |Y - Y'|, X ≠ X', Y ≠ Y'
    ← not hasq(X)
hasq(X) ← q(X,Y)
```

56

n-Queens (ctd)

(in the `smodels` language with cardinality constraints)

```
1 { q(X,Y) } 1 ← d(X)
1 { q(X,Y) } 1 ← d(Y)
    ← q(X,Y), q(X',Y'), |X - X'| = |Y - Y'|, X ≠ X', Y ≠ Y'
d(1) ←
  ⋮
d(n) ←
```

58

n-Queens in answer set programming

(in *nested* logic programming)

```
q(X,Y) ∨ not q(X,Y) ←
    ← q(X,Y), q(X',Y), X ≠ X'
    ← q(X,Y), q(X,Y'), Y ≠ Y'
    ← q(X,Y), q(X',Y'), |X - X'| = |Y - Y'|, X ≠ X', Y ≠ Y'
    ← not hasq(X)
hasq(X) ← q(X,Y)
```

57

n-Queens (ctd)

(in the `smodels` language with cardinality constraints)

```
1 { q(X,Y) : d(Y) } 1 :- d(X).
1 { q(X,Y) : d(X) } 1 :- d(Y).
:- d(X), d(Y), d(X1), d(Y1), q(X,Y), q(X1,Y1), X!=X1, Y!=Y1,
   abs(X-X1) == abs(Y-Y1).
d(1..queens).
```

59

And the Performance ...?

```
torsten@craz > lparse -c queens=20 queens2.lp | smodels
smodels version 2.27. Reading...done
Answer: 1
Stable Model: d(1) ... d(20) q(1,16) q(2,13) q(3,6) q(4,3)
q(5,15) q(6,19) q(7,1) q(8,4) q(9,9) q(10,11) q(11,8) q(12,10) q(13,17) q(14,2)
q(15,20) q(16,18) q(17,7) q(18,5) q(19,14) q(20,12)
True
Duration: 13.031
Number of choice points: 1471
Number of wrong choices: 1464
Number of atoms: 501
Number of rules: 10100
Number of picked atoms: 304305
Number of forced atoms: 14604
Number of truth assignments: 3111768
Size of searchspace (removed): 400 (0)
```

Stop! It's demo time!

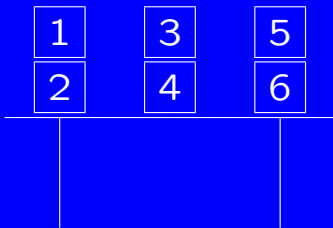
60

61

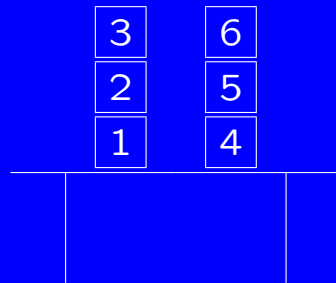
Planning

in the Blocksworld

Initial situation



Goal situation



62

Initial Situation

```
const grippers=2.
const lasttime=3.
```

```
block(1..6).
```

```
% DEFINE
on(1,2,0).
on(2,table,0).
on(3,4,0).
on(4,table,0).
on(5,6,0).
on(6,table,0).
```

63

Goal Situation

```
% TEST
:- not on(3,2,lasttime).
:- not on(2,1,lasttime).
:- not on(1,table,lasttime).
:- not on(6,5,lasttime).
:- not on(5,4,lasttime).
:- not on(4,table,lasttime).
```

64

Planning in the Blocksworld I

GENERATE

```
time(0..lasttime).

location(B) :- block(B).
location(table).

% GENERATE
{ move(B,L,T) : block(B) : location(L) } grippers :- time(T),
                                                    T<lasttime.
```

65

Planning in the Blocksworld II

DEFINE

```
% effect of moving a block
on(B,L,T+1) :- move(B,L,T),
               block(B), location(L), time(T), T<lasttime.

% inertia
on(B,L,T+1) :- on(B,L,T), not neg_on(B,L,T+1),
               location(L), block(B), time(T), T<lasttime.

% uniqueness of location
neg_on(B,L1,T) :- on(B,L,T), L!=L1,
                  block(B), location(L), location(L1), time(T).
```

66

Planning in the Blocksworld III

TEST

```
% neg_on is the negation of on
:- on(B,L,T), neg_on(B,L,T),
   block(B), location(L), time(T).

% two blocks cannot be on top of the same block
:- 2 { on(B1,B,T) : block(B1) },
   block(B), time(T).

% a block can't be moved unless it is clear
:- move(B,L,T), on(B1,B,T),
   block(B), block(B1), location(L), time(T), T<lasttime.

% a block can't be moved onto a block that is being moved also
:- move(B,B1,T), move(B1,L,T),
   block(B), block(B1), location(L), time(T), T<lasttime.
```

67

The Plan

```
torsten@hoedic 538 > lparse blocks.lp | smodels
smodels version 2.25. Reading...done
Answer: 1
Stable Model: move(1,table,0) move(3,table,0)
              move(2,1,1)    move(5,4,1)
              move(3,2,2)    move(6,5,2)
Duration: 0.050
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 507
Number of rules: 3026
Number of picked atoms: 24
Number of forced atoms: 13
Number of truth assignments: 944
Size of searchspace (removed): 0 (0)
```

68

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 answer set computations as derivations in an inference system

➔ *Tableau based proof system for ASP solving*

206

Tableau calculi for analyzing ASP-solving

- Motivation
- Tableau calculi
- Tableau calculi for ASP

205

Tableau calculi

- Traditionally, tableau calculi are used for
 - automated theorem proving and
 - proof theoretical analysisin classical as well as non-classical logics.
- **General idea:** Given an input, prove some property by decomposition. Decomposition is done by applying rules.
- For details, see *Handbook of Tableau Methods*, Kluwer, 1999.

207

Tableau calculi: 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 $\gamma_1, \dots, \gamma_m$ can be extended by applying *tableau rules* of form:

$$\frac{\gamma_1, \dots, \gamma_m}{\alpha_1}$$

$$\vdots$$

$$\alpha_n$$

- Rules of the former format append entries $\alpha_1, \dots, \alpha_n$ to the branch.
- Rules of the latter format create multiple sub-branches for β_1, \dots, β_n .

208

Tableau calculus: Example

A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from \neg , \wedge , and \vee , consists of rules:

$$\frac{\neg\neg\alpha}{\alpha}$$

$$\frac{\alpha_1 \wedge \alpha_2}{\alpha_1}$$

$$\frac{\alpha_1 \wedge \alpha_2}{\alpha_2}$$

$$\frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- All rules are semantically valid, interpreting entries in a branch as connected via “*and*” and distinct (sub-)branches as connected via “*or*”.
- A propositional formula φ is unsatisfiable iff there is a tableau with φ as the root node such that
 1. all other entries can be produced by tableau rules and
 2. every branch contains some formulas α and $\neg\alpha$.

209

Tableau calculus: Example (ctd)

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	[φ]	
(2)	a	[1]	
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]	
(4)	$\neg b \wedge (\neg a \vee b)$	[3]	(9) $\neg\neg\neg a$ [3]
(5)	$\neg b$	[4]	(10) $\neg a$ [9]
(6)	$\neg a \vee b$	[4]	
(7)	$\neg a$	[6]	(8) b [6]

All three branches of the tableau are contradictory (cf. 2, 5, 7, 8, 10).

➔ $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable.

210

Tableaux and ASP: The idea

- 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 an answer set.
- An *entire tableau* represents a traversal of the *search space*.

211

Tableau calculi for ASP: Specific definitions

- A (signed) *tableau* for a logic program Π is a binary tree such that
 - the root node of the tree consists of the rules in Π ;
 - 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 rules (given next).
- 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 program Π , atoms of Π in $atom(\Pi)$ and bodies of Π in $Body(\Pi)$ can occur in signed literals.

212

More concepts

- A *tableau calculus* is a set of tableau rules.
- A branch in a tableau is *contradictory*, if it contains Tv and Fv for some variable v .
- A branch in a tableau is *total* for a program Π , if it contains either Tv or Fv for each $v \in atom(\Pi) \cup Body(\Pi)$.
- 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 contradictory 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 Π , if every branch in the tableau is contradictory.

214

Tableau rules for ASP at a glance

(FTB)	$\frac{p \leftarrow l_1, \dots, l_n}{\frac{tl_1, \dots, tl_n}{T\{l_1, \dots, l_n\}}}$	(BFB)	$\frac{F\{l_1, \dots, l_i, \dots, l_n\}}{\frac{tl_1, \dots, tl_{i-1}, tl_{i+1}, \dots, tl_n}{fl_i}}$
(FTA)	$\frac{p \leftarrow l_1, \dots, l_n}{\frac{T\{l_1, \dots, l_n\}}{Tp}}$	(BFA)	$\frac{p \leftarrow l_1, \dots, l_n}{\frac{Fp}{F\{l_1, \dots, l_n\}}}$
(FFB)	$\frac{p \leftarrow l_1, \dots, l_i, \dots, l_n}{\frac{fl_i}{F\{l_1, \dots, l_i, \dots, l_n\}}}$	(BTB)	$\frac{T\{l_1, \dots, l_i, \dots, l_n\}}{tl_i}$
(FFA)	$\frac{FB_1, \dots, FB_m}{Fp} \quad (\S)$	(BTA)	$\frac{Tp}{\frac{FB_1, \dots, FB_{i-1}, FB_{i+1}, \dots, FB_m}{TB_i}} \quad (\S)$
(WFN)	$\frac{FB_1, \dots, FB_m}{Fp} \quad (\dagger)$	(WFJ)	$\frac{Tp}{\frac{FB_1, \dots, FB_{i-1}, FB_{i+1}, \dots, FB_m}{TB_i}} \quad (\dagger)$
(FL)	$\frac{FB_1, \dots, FB_m}{Fp} \quad (\ddagger)$	(BL)	$\frac{Tp}{\frac{FB_1, \dots, FB_{i-1}, FB_{i+1}, \dots, FB_m}{TB_i}} \quad (\ddagger)$
		(Cut[X])	$\frac{}{Tv \mid Fv} \quad (\# [X])$

213

Example

Consider the program

$$\Pi = \left\{ \begin{array}{l} a \leftarrow \\ c \leftarrow \text{not } b, \text{not } d \\ d \leftarrow a, \text{not } c \end{array} \right\}$$

having two answer sets $\{a, c\}$ and $\{a, d\}$.

215

(Previewed) Example

	$a \leftarrow$		
	$c \leftarrow not\ b, not\ d$		
	$d \leftarrow a, not\ c$		
(FTB)	$T\emptyset$		
(FTA)	Ta		
(FFA)	Fb		
(Cut[atom(II)])	Tc	Fc	
(BTA)	$T\{not\ b, not\ d\}$	(BFA)	$F\{not\ b, not\ d\}$
(BTB)	Fd	(BFB)	Td
(FFB)	$F\{a, not\ c\}$	(FTB)	$T\{a, not\ c\}$

Recall answer sets $\{a, c\}$ and $\{a, d\}$.

216

Tableau rules: Auxiliary definitions

- The application of rules makes use of two conjugation functions, t and f .
- For a body literal l , define:

$$tl = \begin{cases} Tl & \text{if } l \text{ is an atom} \\ Fp & \text{if } l = not\ p \text{ for an atom } p \end{cases}$$

$$fl = \begin{cases} Tp & \text{if } l = not\ p \text{ for an atom } p \\ Fl & \text{if } l \text{ is an atom} \end{cases}$$

- Examples: $tp = Tp$ $fp = Fp$ $tnot\ p = Fp$ $fnot\ p = Tp$

217

Tableau rules: Auxiliary definitions (ctd)

- Some tableau rules require conditions for their application. Such conditions are specified as *provisos*:

$$\frac{\text{prerequisites}}{\text{consequence}} \quad (\text{proviso}) \quad \text{proviso: some condition(s)}$$

☞ All tableau rules given in the sequel are answer set preserving.

218

Forward True Body (FTB)

Prerequisites All of a body's literals are *true*.

Consequence The body is *true*.

Tableau Rule FTB

$$\frac{\begin{array}{c} p \leftarrow l_1, \dots, l_n \\ tl_1, \dots, tl_n \end{array}}{T\{l_1, \dots, l_n\}}$$

Example

$$\frac{\begin{array}{c} a \leftarrow b, not\ c \\ Tb \\ Fc \end{array}}{T\{b, not\ c\}}$$

219

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

$$\frac{F\{l_1, \dots, l_i, \dots, l_n\} \quad tl_1, \dots, tl_{i-1}, tl_{i+1}, \dots, tl_n}{fl_i}$$

Examples

$$\frac{F\{b, \text{not } c\} \quad Tb}{Tc}$$

$$\frac{F\{b, \text{not } c\} \quad Fc}{Fb}$$

220

Forward False Body (FFB)

Prerequisites Some literal of a body is *false*.

Consequence The body is *false*.

Tableau Rule FFB

$$\frac{p \leftarrow l_1, \dots, l_i, \dots, l_n \quad fl_i}{F\{l_1, \dots, l_i, \dots, l_n\}}$$

Examples

$$\frac{a \leftarrow b, \text{not } c \quad Fb}{F\{b, \text{not } c\}}$$

$$\frac{a \leftarrow b, \text{not } c \quad Tc}{F\{b, \text{not } c\}}$$

221

Backward True Body (BTB)

Prerequisites A body is *true*.

Consequence The body's literals are *true*.

Tableau Rule BTB

$$\frac{T\{l_1, \dots, l_i, \dots, l_n\}}{tl_i}$$

Examples

$$\frac{T\{b, \text{not } c\}}{Tb}$$

$$\frac{T\{b, \text{not } c\}}{Fc}$$

222

Reviewing tableau rules for bodies

- Rules FTB, BFB, FFB, and BTB are body-oriented.
- For a body $B = \{l_1, \dots, l_n\}$, consider the equivalence:

$$B \equiv l_1 \wedge \dots \wedge l_n$$

$$\text{FTB: } l_1 \wedge \dots \wedge l_n \rightarrow B \iff \bar{l}_1 \vee \dots \vee \bar{l}_n \vee B$$

$$\text{BFB: } \bar{B} \rightarrow \bar{l}_1 \vee \dots \vee \bar{l}_n \iff \bar{l}_1 \vee \dots \vee \bar{l}_n \vee B$$

$$\text{FFB: } \bar{l}_1 \vee \dots \vee \bar{l}_n \rightarrow \bar{B} \iff (l_1 \vee \bar{B}) \wedge \dots \wedge (l_n \vee \bar{B})$$

$$\text{BTB: } B \rightarrow l_1 \wedge \dots \wedge l_n \iff (l_1 \vee \bar{B}) \wedge \dots \wedge (l_n \vee \bar{B})$$

- ➔ Rules FTB+BFB and FFB+BTB are two sides of the same coin.

223

Forward True Atom (FTA)

Prerequisites Some of an atom's bodies is *true*.

Consequence The atom is *true*.

Tableau Rule FTA

$$\frac{p \leftarrow l_1, \dots, l_n \quad T\{l_1, \dots, l_n\}}{Tp}$$

Examples

$$\frac{a \leftarrow b, \text{not } c \quad T\{b, \text{not } c\}}{Ta}$$

$$\frac{a \leftarrow d, \text{not } e \quad T\{d, \text{not } e\}}{Ta}$$

224

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

$$\frac{p \leftarrow l_1, \dots, l_n \quad Fp}{F\{l_1, \dots, l_n\}}$$

Examples

$$\frac{a \leftarrow b, \text{not } c \quad Fa}{F\{b, \text{not } c\}}$$

$$\frac{a \leftarrow d, \text{not } e \quad Fa}{F\{d, \text{not } e\}}$$

225

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

$$\frac{FB_1, \dots, FB_m \quad (body(p) = \{B_1, \dots, B_m\})}{Fp}$$

Example

$$\frac{F\{b, \text{not } c\} \quad F\{d, \text{not } e\}}{Fa} \quad (body(a) = \{\{b, \text{not } c\}, \{d, \text{not } e\}\})$$

226

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

$$\frac{Tp \quad FB_1, \dots, FB_{i-1}, FB_{i+1}, \dots, FB_m \quad (body(p) = \{B_1, \dots, B_m\})}{TB_i}$$

Examples

$$\frac{Ta \quad F\{b, \text{not } c\}}{T\{d, \text{not } e\}} (*) \quad \frac{Ta \quad F\{d, \text{not } e\}}{T\{b, \text{not } c\}} (*)$$

(*): $body(a) = \{\{b, \text{not } c\}, \{d, \text{not } e\}\}$

227

Reviewing tableau rules for atoms

- Rules FTA, BFA, FFA, and BTA are atom-oriented.
- For an atom p such that $body(p) = \{B_1, \dots, B_m\}$, consider the equivalence:

$$p \equiv B_1 \vee \dots \vee B_m$$

$$\mathbf{FTA:} \quad B_1 \vee \dots \vee B_m \rightarrow p \iff (\overline{B_1} \vee p) \wedge \dots \wedge (\overline{B_m} \vee p)$$

$$\mathbf{BFA:} \quad \overline{p} \rightarrow \overline{B_1} \wedge \dots \wedge \overline{B_m} \iff (\overline{B_1} \vee p) \wedge \dots \wedge (\overline{B_m} \vee p)$$

$$\mathbf{FFA:} \quad \overline{B_1} \wedge \dots \wedge \overline{B_m} \rightarrow \overline{p} \iff B_1 \vee \dots \vee B_m \vee \overline{p}$$

$$\mathbf{BTA:} \quad p \rightarrow B_1 \vee \dots \vee B_m \iff B_1 \vee \dots \vee B_m \vee \overline{p}$$

- As with body-oriented rules, FTA+BFA and FFA+BTA are two sides of the same coin.

228

Preliminaries for unfounded sets

Let Π be a logic program.

- For a $\Pi' \subseteq \Pi$, define the *greatest unfounded set*, denoted $GUS(\Pi')$, of Π with respect to Π' as:

$$GUS(\Pi') = atom(\Pi) \setminus Cn((\Pi')^0)$$

- The rule-based definition of the greatest unfounded set is more flexible than the atom-based one.

- For a loop $L \in Loop(\Pi)$, define

$$EB(L) = \{body(r) \mid r \in \Pi, head(r) \in L, body^+(r) \cap L = \emptyset\}$$

as the *external bodies* of L .

229

Well-Founded Negation (WFN)

Prerequisites An atom is in the greatest unfounded set with respect to bodies that are *false*.

Consequence The atom is *false*.

Tableau Rule WFN

$$\frac{FB_1, \dots, FB_m}{Fp} \quad (p \in GUS(\{r \in \Pi \mid body(r) \notin \{B_1, \dots, B_m\}\}))$$

Examples

$$\frac{a \leftarrow not\ b}{Fa} \quad (*)$$

$$\frac{a \leftarrow a \quad a \leftarrow not\ b}{Fa} \quad (*)$$

(*): $a \in GUS(\Pi \setminus \{a \leftarrow not\ b\})$

230

Well-Founded Justification (WFJ)

Prerequisites A *true* atom is in the greatest unfounded set with respect to bodies that are *false* if one particular body becomes *false*.

Consequence The respective body is *true*.

Tableau Rule WFJ

$$\frac{Tp \quad FB_1, \dots, FB_{i-1}, FB_{i+1}, \dots, FB_m}{TB_i} \quad (p \in GUS(\{r \in \Pi \mid body(r) \notin \{B_1, \dots, B_m\}\}))$$

Examples

$$\frac{a \leftarrow not\ b}{Ta} \quad (*)$$

$$\frac{a \leftarrow a \quad a \leftarrow not\ b}{T\{not\ b\}} \quad (*)$$

(*): $a \in GUS(\Pi \setminus \{a \leftarrow not\ b\})$

231

Reviewing well-founded tableau rules

Respective tableau rules ensure non-circular support for *true* atoms.

The declarative background is:

$$\overline{B_1} \wedge \cdots \wedge \overline{B_m} \rightarrow \bigwedge_{p \in GUS(\{r \in \Pi \mid \text{body}(r) \notin \{B_1, \dots, B_m\}\})} \overline{p} \iff \bigvee_{p \in GUS(\{r \in \Pi \mid \text{body}(r) \notin \{B_1, \dots, B_m\}\})} p \rightarrow B_1 \vee \cdots \vee B_m$$

Observe that

1. WFN subsumes forwarding *false* atoms by FFA,
2. WFJ subsumes backwarding *true* atoms by BTA,
3. genuine ASP-solvers (*nomore++*, *smodels*) apply WFN, but not WFJ, and
4. SAT-based solvers (*assat*, *cmodels*) focus unfounded sets on loops.

☞ Loops are the “interesting” unfounded sets.

232

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

$$\frac{Tp \quad FB_1, \dots, FB_{i-1}, FB_{i+1}, \dots, FB_m}{TB_i} \quad (p \in L, L \in \text{Loop}(\Pi), EB(L) = \{B_1, \dots, B_m\})$$

Example

$$\frac{a \leftarrow a \quad a \leftarrow \text{not } b \quad Ta}{T\{\text{not } b\}} \quad (EB(\{a\}) = \{\{\text{not } b\}\})$$

234

Forward Loop (FL)

Prerequisites The external bodies of a loop are *false*.

Consequence The atoms in the loop are *false*.

Tableau Rule FL

$$\frac{FB_1, \dots, FB_m}{Fp} \quad (p \in L, L \in \text{Loop}(\Pi), EB(L) = \{B_1, \dots, B_m\})$$

Example

$$\frac{a \leftarrow a \quad a \leftarrow \text{not } b \quad F\{\text{not } b\}}{Fa} \quad (EB(\{a\}) = \{\{\text{not } b\}\})$$

233

Reviewing tableau rules for loops

Respective tableau rules ensure non-circular support for *true* atoms.

The declarative background is provided by loop formulas:

$$\neg(\bigvee_{B \in EB(L)} B) \rightarrow \bigwedge_{p \in L} \overline{p} \iff \bigvee_{p \in L} p \rightarrow \bigvee_{B \in EB(L)} B$$

Iteration of FL, FFA, and FFB is as powerful as WFN (and FFB).

☞ Inferences based on loops focus unfounded sets to necessary parts.

ToDo: Implementation of BL in ASP-solvers

235

Case analysis by Cut

Up to now, all tableau rules are deterministic.
That is, rules extend a single branch but cannot create sub-branches.

☞ Closing a branch, generally, leads to a partial assignment.

Case analysis is done by $Cut[\mathcal{C}]$ where $\mathcal{C} \subseteq atom(\Pi) \cup Body(\Pi)$.

Tableau Rule $Cut[\mathcal{C}]$

$$\frac{}{Tv \mid Fv} \quad (v \in \mathcal{C})$$

Examples $Cut[\mathcal{C}]$

$$\frac{a \leftarrow not \ b \quad b \leftarrow not \ a}{Ta \mid Fa} \quad (\mathcal{C} = atom(\Pi)) \qquad \frac{a \leftarrow not \ b \quad b \leftarrow not \ a}{T\{not \ b\} \mid F\{not \ b\}} \quad (\mathcal{C} = Body(\Pi))$$

236

Tableau calculi behind ASP-solvers

ASP-solvers combine propagation with case analysis. The basic strategy is to close a branch by propagation and to then create sub-branches.

We obtain the following tableau calculi characterizing ASP-solvers:

$$\begin{aligned} \mathcal{T}_{cmodels-1} &= \mathcal{T}_{completion} \cup \{Cut[atom(\Pi) \cup Body(\Pi)]\} \\ \mathcal{T}_{assat} &= \mathcal{T}_{completion} \cup \{FL\} \cup \{Cut[atom(\Pi) \cup Body(\Pi)]\} \\ \mathcal{T}_{smodels} &= \mathcal{T}_{completion} \cup \{WFN\} \cup \{Cut[atom(\Pi)]\} \\ \mathcal{T}_{noMoRe} &= \mathcal{T}_{completion} \cup \{WFN\} \cup \{Cut[Body(\Pi)]\} \\ \mathcal{T}_{nomore++} &= \mathcal{T}_{completion} \cup \{WFN\} \cup \{Cut[atom(\Pi) \cup Body(\Pi)]\} \end{aligned}$$

Note that *assat* applies FL only to total branches (but records loop formulas).

Genuine ASP-solvers, *smodels*, *noMoRe*, and *nomore++*, differ only in their Cut rules. Their strategies are all sound and complete, but yield different *proof complexity*.

238

Well-known tableau calculi

Fitting's operator Φ applies forward propagation without sophisticated unfounded set checks. We have:

$$\mathcal{T}_{\Phi} = \{FTB, FTA, FFB, FFA\}$$

Well-founded operator Ω replaces negation of single atoms with negation of unfounded sets. We have:

$$\mathcal{T}_{\Omega} = \{FTB, FTA, FFB, WFN\}$$

Unit propagation on a program's completion can be characterized by forward and backward propagation. We have:

$$\mathcal{T}_{completion} = \{FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA\}$$

237

Proof complexity

The notion of *proof complexity* is used for describing the relative efficiency of different proof systems.

It compares proof systems based on *minimal refutations*.

☞ Proof complexity does not depend on 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 greater than minimal refutations of \mathcal{T}' .

The size of tableaux is simply the number of their entries.

Note that we do not need to know the precise number of entries, counting Cut applications is sufficient.

239

$\mathcal{T}_{\text{models}}$ VERSUS $\mathcal{T}_{\text{noMoRe}}$

Recall that $\mathcal{T}_{\text{models}}$ restricts Cut to $\text{atom}(\Pi)$ and $\mathcal{T}_{\text{noMoRe}}$ to $\text{Body}(\Pi)$.

Are both approaches similar or is one of them superior to the other?

Let $\{\Pi_a^n\}$, $\{\Pi_b^n\}$, and $\{\Pi_c^n\}$ be infinite families of programs as follows:

$$\Pi_a^n = \left\{ \begin{array}{l} x \leftarrow \text{not } x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad \Pi_b^n = \left\{ \begin{array}{ll} x \leftarrow c_1, \dots, c_n, \text{not } x & \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad \Pi_c^n = \left\{ \begin{array}{l} a_1 \leftarrow \text{not } b_1 \\ b_1 \leftarrow \text{not } a_1 \\ \vdots \\ a_n \leftarrow \text{not } b_n \\ b_n \leftarrow \text{not } a_n \end{array} \right\}$$

In minimal refutations for $\Pi_a^n \cup \Pi_c^n$, the number of applications of $\text{Cut}[\text{Body}(\Pi_a^n \cup \Pi_c^n)]$ with $\mathcal{T}_{\text{noMoRe}}$ is linear in n , whereas $\mathcal{T}_{\text{models}}$ requires exponentially many applications of $\text{Cut}[\text{atom}(\Pi_a^n \cup \Pi_c^n)]$.

Vice versa, minimal refutations for $\Pi_b^n \cup \Pi_c^n$ require linearly many applications of $\text{Cut}[\text{atom}(\Pi_b^n \cup \Pi_c^n)]$ with $\mathcal{T}_{\text{models}}$ and exponentially many applications of $\text{Cut}[\text{Body}(\Pi_b^n \cup \Pi_c^n)]$ with $\mathcal{T}_{\text{noMoRe}}$.

240

Conclusion & Outlook

Tableau methods are well-suited to underlay ASP-solving approaches with a proof-theoretical fundament.

We can uniformly describe and compare inference patterns, an existing implementation is not categorical for this.

Unfounded set inference as implemented in current ASP-solvers is not yet optimal.

Branching on atoms and bodies is mandatory in powerful ASP-solvers. Further study and development of effective heuristics is needed.

The from the declarative point of view most consequential approach, say

$$\mathcal{T}_{\text{future}} = \mathcal{T}_{\text{completion}} \cup \{FL, BL, \text{Cut}[\text{atom}(\Pi) \cup \text{Body}(\Pi)]\},$$

is to be implemented and augmented with advanced techniques, ASP-specific or adopted from related areas.

242

Relative efficiency

As witnessed by $\{\Pi_a^n \cup \Pi_c^n\}$ and $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, $\mathcal{T}_{\text{models}}$ and $\mathcal{T}_{\text{noMoRe}}$ do not polynomially simulate one another.

Any refutation of $\mathcal{T}_{\text{models}}$ or $\mathcal{T}_{\text{noMoRe}}$ is a refutation of $\mathcal{T}_{\text{nomore}^{++}}$ (but not vice versa).

It follows that

- both $\mathcal{T}_{\text{models}}$ and $\mathcal{T}_{\text{noMoRe}}$ are polynomially simulated by $\mathcal{T}_{\text{nomore}^{++}}$ and
- $\mathcal{T}_{\text{nomore}^{++}}$ is not polynomially simulated by either $\mathcal{T}_{\text{models}}$ or $\mathcal{T}_{\text{noMoRe}}$.

→ The proof system obtained with $\text{Cut}[\text{atom}(\Pi) \cup \text{Body}(\Pi)]$ is exponentially stronger than with $\text{Cut}[\text{atom}(\Pi)]$ and $\text{Cut}[\text{Body}(\Pi)]$!

241

✉ If you have any ideas for a project or diploma thesis, don't hesitate to ask us ;-)

243

$\mathcal{T}_{\text{cmmodels-1}}$: Example tableau

	(r_1)	$a \leftarrow \text{not } b$		(r_2)	$b \leftarrow d, \text{not } a$		(r_3)	$c \leftarrow b, d$
	(r_4)	$c \leftarrow g$		(r_5)	$d \leftarrow c$		(r_6)	$d \leftarrow g$
	(r_7)	$e \leftarrow f, \text{not } c$		(r_8)	$f \leftarrow \text{not } g$		(r_9)	$g \leftarrow \text{not } a, \text{not } f$
(1)	Ta	[Cut]	(16)	Fa	[Cut]			
(2)	$T\{\text{not } b\}$	[BTA: $r_1, 1$]	(17)	$F\{\text{not } b\}$	[BFA: $r_1, 16$]			
(3)	Fb	[BTB: 2]	(18)	Tb	[BFB: 17]			
(4)	$F\{d, \text{not } a\}$	[BFA: $r_2, 3$]	(19)	$T\{d, \text{not } a\}$	[BTA: $r_2, 18$]			
(5)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 1$]	(20)	Td	[BTB: 19]			
(6)	Fg	[FFA: $r_9, 5$]	(21)	$T\{b, d\}$	[FTB: $r_3, 18, 20$]			
(7)	$T\{\text{not } g\}$	[FTB: $r_8, 6$]	(22)	Tc	[FTA: $r_3, 21$]			
(8)	Tf	[FTA: $r_8, 7$]	(23)	$F\{f, \text{not } c\}$	[FFB: $r_7, 22$]			
(9)	$F\{b, d\}$	[FFB: $r_3, 3$]	(24)	Fe	[FFA: $r_7, 23$]			
(10)	$F\{g\}$	[FFB: $r_4, r_6, 6$]	(25)	$T\{c\}$	[FTB: $r_5, 22$]			
(11)	Fc	[FFA: $r_3, r_4, 9, 10$]	(26)	Tf	[Cut]	(31)	Ff	[Cut]
(12)	$F\{c\}$	[FFB: $r_5, 11$]	(27)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 26$]	(32)	$T\{\text{not } a, \text{not } f\}$	[FTB: $r_9, 16, 31$]
(13)	Fd	[FFA: $r_5, r_6, 10, 12$]	(28)	Fg	[FFA: $r_9, 27$]	(33)	Tg	[FTA: $r_9, 32$]
(14)	$T\{f, \text{not } c\}$	[FTB: $r_7, 8, 11$]	(29)	$F\{g\}$	[FFB: $r_4, r_6, 28$]	(34)	$T\{g\}$	[FTB: $r_4, r_6, 33$]
(15)	Te	[FTA: $r_7, 14$]	(30)	$T\{\text{not } g\}$	[FTB: $r_8, 28$]	(35)	$F\{\text{not } g\}$	[FFB: $r_8, 33$]

244

$\mathcal{T}_{\text{assat}}$: Example tableau

	(r_1)	$a \leftarrow \text{not } b$		(r_2)	$b \leftarrow d, \text{not } a$		(r_3)	$c \leftarrow b, d$
	(r_4)	$c \leftarrow g$		(r_5)	$d \leftarrow c$		(r_6)	$d \leftarrow g$
	(r_7)	$e \leftarrow f, \text{not } c$		(r_8)	$f \leftarrow \text{not } g$		(r_9)	$g \leftarrow \text{not } a, \text{not } f$
(1)	Ta	[Cut]	(16)	Fa	[Cut]			
(2)	$T\{\text{not } b\}$	[BTA: $r_1, 1$]	(17)	$F\{\text{not } b\}$	[BFA: $r_1, 16$]			
(3)	Fb	[BTB: 2]	(18)	Tb	[BFB: 17]			
(4)	$F\{d, \text{not } a\}$	[BFA: $r_2, 3$]	(19)	$T\{d, \text{not } a\}$	[BTA: $r_2, 18$]			
(5)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 1$]	(20)	Td	[BTB: 19]			
(6)	Fg	[FFA: $r_9, 5$]	(21)	$T\{b, d\}$	[FTB: $r_3, 18, 20$]			
(7)	$T\{\text{not } g\}$	[FTB: $r_8, 6$]	(22)	Tc	[FTA: $r_3, 21$]			
(8)	Tf	[FTA: $r_8, 7$]	(23)	$F\{f, \text{not } c\}$	[FFB: $r_7, 22$]			
(9)	$F\{b, d\}$	[FFB: $r_3, 3$]	(24)	Fe	[FFA: $r_7, 23$]			
(10)	$F\{g\}$	[FFB: $r_4, r_6, 6$]	(25)	$T\{c\}$	[FTB: $r_5, 22$]			
(11)	Fc	[FFA: $r_3, r_4, 9, 10$]	(26)	Tf	[Cut]	(32)	Ff	[Cut]
(12)	$F\{c\}$	[FFB: $r_5, 11$]	(27)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 26$]	(33)	$T\{\text{not } a, \text{not } f\}$	[FTB: $r_9, 16, 32$]
(13)	Fd	[FFA: $r_5, r_6, 10, 12$]	(28)	Fg	[FFA: $r_9, 27$]	(34)	Tg	[FTA: $r_9, 33$]
(14)	$T\{f, \text{not } c\}$	[FTB: $r_7, 8, 11$]	(29)	$F\{g\}$	[FFB: $r_4, r_6, 28$]	(35)	$T\{g\}$	[FTB: $r_4, r_6, 34$]
(15)	Te	[FTA: $r_7, 14$]	(30)	$T\{\text{not } g\}$	[FTB: $r_8, 28$]	(36)	$F\{\text{not } g\}$	[FFB: $r_8, 34$]
			(31)	Fc	[FL: $\{c, d\}, 29$]			

245

$\mathcal{T}_{\text{smodels}}$: Example tableau

	(r_1)	$a \leftarrow \text{not } b$		(r_2)	$b \leftarrow d, \text{not } a$		(r_3)	$c \leftarrow b, d$
	(r_4)	$c \leftarrow g$		(r_5)	$d \leftarrow c$		(r_6)	$d \leftarrow g$
	(r_7)	$e \leftarrow f, \text{not } c$		(r_8)	$f \leftarrow \text{not } g$		(r_9)	$g \leftarrow \text{not } a, \text{not } f$
(1)	Ta	[Cut]	(16)	Fa	[Cut]			
(2)	$T\{\text{not } b\}$	[BTA: $r_1, 1$]	(17)	$F\{\text{not } b\}$	[BFA: $r_1, 16$]			
(3)	Fb	[BTB: 2]	(18)	Tb	[BFB: 17]			
(4)	$F\{d, \text{not } a\}$	[BFA: $r_2, 3$]	(19)	$T\{d, \text{not } a\}$	[BTA: $r_2, 18$]			
(5)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 1$]	(20)	Td	[BTB: 19]			
(6)	Fg	[FFA: $r_9, 5$]	(21)	$T\{b, d\}$	[FTB: $r_3, 18, 20$]			
(7)	$T\{\text{not } g\}$	[FTB: $r_8, 6$]	(22)	Tc	[FTA: $r_3, 21$]			
(8)	Tf	[FTA: $r_8, 7$]	(23)	$F\{f, \text{not } c\}$	[FFB: $r_7, 22$]			
(9)	$F\{b, d\}$	[FFB: $r_3, 3$]	(24)	Fe	[FFA: $r_7, 23$]			
(10)	$F\{g\}$	[FFB: $r_4, r_6, 6$]	(25)	$T\{c\}$	[FTB: $r_5, 22$]			
(11)	Fc	[FFA: $r_3, r_4, 9, 10$]	(26)	Tf	[Cut]	(29)	Ff	[Cut]
(12)	$F\{c\}$	[FFB: $r_5, 11$]	(27)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 26$]	(30)	$T\{\text{not } a, \text{not } f\}$	[FTB: $r_9, 16, 29$]
(13)	Fd	[FFA: $r_5, r_6, 10, 12$]	(28)	Fc	[WFN: 27]	(31)	Tg	[FTA: $r_9, 30$]
(14)	$T\{f, \text{not } c\}$	[FTB: $r_7, 8, 11$]				(32)	$T\{g\}$	[FTB: $r_4, r_6, 31$]
(15)	Te	[FTA: $r_7, 14$]				(33)	$F\{\text{not } g\}$	[FFB: $r_8, 31$]

246

$\mathcal{T}_{\text{noMore}}$: Example tableau

	(r_1)	$a \leftarrow \text{not } b$		(r_2)	$b \leftarrow d, \text{not } a$		(r_3)	$c \leftarrow b, d$
	(r_4)	$c \leftarrow g$		(r_5)	$d \leftarrow c$		(r_6)	$d \leftarrow g$
	(r_7)	$e \leftarrow f, \text{not } c$		(r_8)	$f \leftarrow \text{not } g$		(r_9)	$g \leftarrow \text{not } a, \text{not } f$
(1)	$T\{\text{not } b\}$	[Cut]	(16)	$F\{\text{not } b\}$	[Cut]			
(2)	Ta	[FTA: $r_1, 1$]	(17)	Fa	[FFA: $r_1, 16$]			
(3)	Fb	[BTB: 1]	(18)	Tb	[BFB: 16]			
(4)	$F\{d, \text{not } a\}$	[BFA: $r_2, 3$]	(19)	$T\{d, \text{not } a\}$	[BTA: $r_2, 18$]			
(5)	$F\{\text{not } a, \text{not } f\}$	[FFB: $r_9, 2$]	(20)	Td	[BTB: 19]			
(6)	Fg	[FFA: $r_9, 5$]	(21)	$T\{b, d\}$	[FTB: $r_3, 18, 20$]			
(7)	$T\{\text{not } g\}$	[FTB: $r_8, 6$]	(22)	Tc	[FTA: $r_3, 21$]			
(8)	Tf	[FTA: $r_8, 7$]	(23)	$F\{f, \text{not } c\}$	[FFB: $r_7, 22$]			
(9)	$F\{b, d\}$	[FFB: $r_3, 3$]	(24)	Fe	[FFA: $r_7, 23$]			
(10)	$F\{g\}$	[FFB: $r_4, r_6, 6$]	(25)	$T\{c\}$	[FTB: $r_5, 22$]			
(11)	Fc	[FFA: $r_3, r_4, 9, 10$]	(26)	$T\{\text{not } g\}$	[Cut]	(30)	$F\{\text{not } g\}$	[Cut]
(12)	$F\{c\}$	[FFB: $r_5, 11$]	(27)	Fg	[BTB: 26]	(31)	Tg	[BFB: 30]
(13)	Fd	[FFA: $r_5, r_6, 10, 12$]	(28)	$F\{g\}$	[FFB: $r_4, r_6, 27$]	(32)	$T\{g\}$	[FTB: $r_4, r_6, 31$]
(14)	$T\{f, \text{not } c\}$	[FTB: $r_7, 8, 11$]	(29)	Fc	[WFN: 28]	(33)	Ff	[FFA: $r_8, 30$]
(15)	Te	[FTA: $r_7, 14$]				(34)	$T\{\text{not } a, \text{not } f\}$	[FTB: $r_9, 17, 33$]

247

$\mathcal{T}_{\text{nomore}++}$: Example tableau

$(r_1) \quad a \leftarrow \text{not } b$ $(r_4) \quad c \leftarrow g$ $(r_7) \quad e \leftarrow f, \text{not } c$	$(r_2) \quad b \leftarrow d, \text{not } a$ $(r_5) \quad d \leftarrow c$ $(r_8) \quad f \leftarrow \text{not } g$	$(r_3) \quad c \leftarrow b, d$ $(r_6) \quad d \leftarrow g$ $(r_9) \quad g \leftarrow \text{not } a, \text{not } f$
$(1) \quad T a \quad [\text{Cut}]$ $(2) \quad T\{\text{not } b\} \quad [\text{BTA}: r_1, 1]$ $(3) \quad F b \quad [\text{BTB}: 2]$ $(4) \quad F\{d, \text{not } a\} \quad [\text{BFA}: r_2, 3]$ $(5) \quad F\{\text{not } a, \text{not } f\} \quad [\text{FFB}: r_9, 1]$ $(6) \quad F g \quad [\text{FFA}: r_9, 5]$ $(7) \quad T\{\text{not } g\} \quad [\text{FTB}: r_8, 6]$ $(8) \quad T f \quad [\text{FTA}: r_8, 7]$ $(9) \quad F\{b, d\} \quad [\text{FFB}: r_3, 3]$ $(10) \quad F\{g\} \quad [\text{FFB}: r_4, r_6, 6]$ $(11) \quad F c \quad [\text{FFA}: r_3, r_4, 9, 10]$ $(12) \quad F\{c\} \quad [\text{FFB}: r_5, 11]$ $(13) \quad F d \quad [\text{FFA}: r_5, r_6, 10, 12]$ $(14) \quad T\{f, \text{not } c\} \quad [\text{FTB}: r_7, 8, 11]$ $(15) \quad T e \quad [\text{FTA}: r_7, 14]$	$(16) \quad F a \quad [\text{Cut}]$ $(17) \quad F\{\text{not } b\} \quad [\text{BFA}: r_1, 16]$ $(18) \quad T b \quad [\text{BFB}: 17]$ $(19) \quad T\{d, \text{not } a\} \quad [\text{BTA}: r_2, 18]$ $(20) \quad T d \quad [\text{BTB}: 19]$ $(21) \quad T\{b, d\} \quad [\text{FTB}: r_3, 18, 20]$ $(22) \quad T c \quad [\text{FTA}: r_3, 21]$ $(23) \quad F\{f, \text{not } c\} \quad [\text{FFB}: r_7, 22]$ $(24) \quad F e \quad [\text{FFA}: r_7, 23]$ $(25) \quad T\{c\} \quad [\text{FTB}: r_5, 22]$ $(26) \quad T\{\text{not } g\} \quad [\text{Cut}]$ $(27) \quad F g \quad [\text{BTB}: 26]$ $(28) \quad F\{g\} \quad [\text{FFB}: r_4, r_6, 27]$ $(29) \quad F c \quad [\text{WFN}: 28]$	$(30) \quad F\{\text{not } g\} \quad [\text{Cut}]$ $(31) \quad T g \quad [\text{BFB}: 30]$ $(32) \quad T\{g\} \quad [\text{FTB}: r_4, r_6, 31]$ $(33) \quad F f \quad [\text{FFA}: r_8, 30]$ $(34) \quad T\{\text{not } a, \text{not } f\} \quad [\text{FTB}: r_9, 16, 33]$

248

$\mathcal{T}_{\text{future}}$: Example tableau

$(r_1) \quad a \leftarrow \text{not } b$ $(r_4) \quad c \leftarrow g$ $(r_7) \quad e \leftarrow f, \text{not } c$	$(r_2) \quad b \leftarrow d, \text{not } a$ $(r_5) \quad d \leftarrow c$ $(r_8) \quad f \leftarrow \text{not } g$	$(r_3) \quad c \leftarrow b, d$ $(r_6) \quad d \leftarrow g$ $(r_9) \quad g \leftarrow \text{not } a, \text{not } f$
$(1) \quad T a \quad [\text{Cut}]$ $(2) \quad T\{\text{not } b\} \quad [\text{BTA}: r_1, 1]$ $(3) \quad F b \quad [\text{BTB}: 2]$ $(4) \quad F\{d, \text{not } a\} \quad [\text{BFA}: r_2, 3]$ $(5) \quad F\{\text{not } a, \text{not } f\} \quad [\text{FFB}: r_9, 1]$ $(6) \quad F g \quad [\text{FFA}: r_9, 5]$ $(7) \quad T\{\text{not } g\} \quad [\text{FTB}: r_8, 6]$ $(8) \quad T f \quad [\text{FTA}: r_8, 7]$ $(9) \quad F\{b, d\} \quad [\text{FFB}: r_3, 3]$ $(10) \quad F\{g\} \quad [\text{FFB}: r_4, r_6, 6]$ $(11) \quad F c \quad [\text{FFA}: r_3, r_4, 9, 10]$ $(12) \quad F\{c\} \quad [\text{FFB}: r_5, 11]$ $(13) \quad F d \quad [\text{FFA}: r_5, r_6, 10, 12]$ $(14) \quad T\{f, \text{not } c\} \quad [\text{FTB}: r_7, 8, 11]$ $(15) \quad T e \quad [\text{FTA}: r_7, 14]$	$(16) \quad F a \quad [\text{Cut}]$ $(17) \quad F\{\text{not } b\} \quad [\text{BFA}: r_1, 16]$ $(18) \quad T b \quad [\text{BFB}: 17]$ $(19) \quad T\{d, \text{not } a\} \quad [\text{BTA}: r_2, 18]$ $(20) \quad T d \quad [\text{BTB}: 19]$ $(21) \quad T\{b, d\} \quad [\text{FTB}: r_3, 18, 20]$ $(22) \quad T c \quad [\text{FTA}: r_3, 21]$ $(23) \quad F\{f, \text{not } c\} \quad [\text{FFB}: r_7, 22]$ $(24) \quad F e \quad [\text{FFA}: r_7, 23]$ $(25) \quad T\{c\} \quad [\text{FTB}: r_5, 22]$ $(26) \quad T\{g\} \quad [\text{BL}: \{c, d\}, 20]$ $(27) \quad T g \quad [\text{BTB}: 26]$ $(28) \quad T\{\text{not } a, \text{not } f\} \quad [\text{BTA}: r_9, 27]$ $(29) \quad F\{\text{not } g\} \quad [\text{FFB}: r_8, 27]$ $(30) \quad F f \quad [\text{FFA}: r_8, 29]$	$(30) \quad F\{\text{not } g\} \quad [\text{Cut}]$ $(31) \quad T g \quad [\text{BFB}: 30]$ $(32) \quad T\{g\} \quad [\text{FTB}: r_4, r_6, 31]$ $(33) \quad F f \quad [\text{FFA}: r_8, 30]$ $(34) \quad T\{\text{not } a, \text{not } f\} \quad [\text{FTB}: r_9, 16, 33]$

249

Conflict-Driven Answer Set Solving

- Motivation
- Assignments
- Nogoods
- Conflict Learning
- Algorithms
- Experiments

250

Motivation

Goal New approach to computing answer sets of logic programs, based on concepts from

- Constraint Processing (CSP) and
- Satisfiability Checking (SAT)

Idea View inferences in Answer Set Programming (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 CSP and SAT
- highly competitive implementation

251

Assignments

- An *assignment* A over $dom(A) = atom(\Pi) \cup body(\Pi)$ is a sequence

$$(\sigma_1, \dots, \sigma_n)$$

of *signed literals* σ_i of form $\mathbf{T}p$ or $\mathbf{F}p$ for $p \in dom(A)$ and $1 \leq i \leq n$;

☞ $\mathbf{T}p$ expresses that p is *true* and $\mathbf{F}p$ that it is *false*.

- The complement, $\bar{\sigma}$, of a literal σ is defined as $\overline{\mathbf{T}p} = \mathbf{F}p$ and $\overline{\mathbf{F}p} = \mathbf{T}p$.
- $A \circ B$ denotes the concatenation of assignments A and B .
- We sometimes identify an assignment with the set of its literals. Given this, we access true and false propositions in A via

$$A^{\mathbf{T}} = \{p \in dom(A) \mid \mathbf{T}p \in A\} \quad \text{and} \quad A^{\mathbf{F}} = \{p \in dom(A) \mid \mathbf{F}p \in A\} .$$

252

Nogoods of Logic Programs

via a program's Clark completion

The Clark completion of a logic program Π can be defined as follows:

$$\{p_\beta \leftrightarrow p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n \mid \beta \in body(\Pi), \beta = \{p_1, \dots, p_m, not\ p_{m+1}, \dots, not\ p_n\}\}$$

$$\cup \{p \leftrightarrow p_{\beta_1} \vee \dots \vee p_{\beta_k} \mid p \in atom(\Pi), body(p) = \{\beta_1, \dots, \beta_k\}\} .$$

where $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$.

254

Nogoods, Solutions, and Unit Propagation

- A *nogood* is a set $\{\sigma_1, \dots, \sigma_n\}$ of signed literals, expressing a *constraint* violated by any assignment containing $\sigma_1, \dots, \sigma_n$.
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ and $A^{\mathbf{T}} \cap A^{\mathbf{F}} = \emptyset$ is a *solution* for a set Δ of nogoods, if $\delta \not\subseteq A$ for all $\delta \in \Delta$.
- For a nogood δ , a literal $\sigma \in \delta$, and an assignment A , we say that $\bar{\sigma}$ is *unit-resulting* for δ wrt A , if
 - $\delta \setminus A = \{\sigma\}$ and
 - $\bar{\sigma} \notin A$.
- For a set Δ 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 Δ .

253

Nogoods of Logic Programs (ctd)

via a program's Clark completion

Let $\beta \in body(\Pi)$ be a body in some logic program Π .

The equivalence

$$p_\beta \leftrightarrow p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n$$

can be decomposed into two implications.

- We get

$$p_\beta \rightarrow p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n ,$$

which is equivalent to the conjunction of

$$\neg p_\beta \vee p_1, \dots, \neg p_\beta \vee p_m, \neg p_\beta \vee \neg p_{m+1}, \dots, \text{and } \neg p_\beta \vee \neg p_n .$$

This set of clauses expresses the following set of nogoods:

$$\Delta(\beta) = \{\{\mathbf{T}\beta, \mathbf{F}p_1\}, \dots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \dots, \{\mathbf{T}\beta, \mathbf{T}p_n\}\} .$$

255

2. The converse of the previous implication, viz.

$$p_\beta \leftarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n ,$$

gives rise to the nogood

$$\delta(\beta) = \{ \mathbf{F}\beta, \mathbf{T}p_1, \dots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \dots, \mathbf{F}p_n \} .$$

Intuitively, $\delta(\beta)$ is a constraint enforcing the truth of body β , or the falsity of a contained literal.

256

Nogoods of Logic Programs

Atom-oriented nogoods

For an atom p where $body(p) = \{\beta_1, \dots, \beta_k\}$, define

$$\begin{aligned} \delta(p) &= \{ \mathbf{T}p, \mathbf{F}\beta_1, \dots, \mathbf{F}\beta_k \} \\ \Delta(p) &= \{ \{ \mathbf{F}p, \mathbf{T}\beta_1 \}, \dots, \{ \mathbf{F}p, \mathbf{T}\beta_k \} \} \end{aligned}$$

For example, for atom x with $body(x) = \{ \{y\}, \{not\ z\} \}$, we obtain

$\begin{array}{l} x \leftarrow y \\ x \leftarrow not\ z \end{array}$	$\begin{aligned} \delta(x) &= \{ \mathbf{T}x, \mathbf{F}\{y\}, \mathbf{F}\{not\ z\} \} \\ \Delta(x) &= \{ \{ \mathbf{F}x, \mathbf{T}\{y\} \}, \{ \mathbf{F}x, \mathbf{T}\{not\ z\} \} \} \end{aligned}$
--	---

For nogood $\delta(x) = \{ \mathbf{T}x, \mathbf{F}\{y\}, \mathbf{F}\{not\ z\} \}$, the signed literal

- $\mathbf{F}x$ is unit-resulting wrt assignment $(\mathbf{F}\{y\}, \mathbf{F}\{not\ z\})$ and
- $\mathbf{T}\{not\ z\}$ is unit-resulting wrt assignment $(\mathbf{T}x, \mathbf{F}\{y\})$

258

Nogoods of Logic Programs (ctd)

via a program's Clark completion

Proceeding analogously with the atom-based equivalences, viz.

$$p \leftrightarrow p_{\beta_1} \vee \cdots \vee p_{\beta_k}$$

we obtain for an atom $p \in atom(\Pi)$ along with its bodies

$body(p) = \{\beta_1, \dots, \beta_k\}$ the nogoods

$$\Delta(p) = \{ \{ \mathbf{F}p, \mathbf{T}\beta_1 \}, \dots, \{ \mathbf{F}p, \mathbf{T}\beta_k \} \} \quad \text{and} \quad \delta(p) = \{ \mathbf{T}p, \mathbf{F}\beta_1, \dots, \mathbf{F}\beta_k \} .$$

257

Nogoods of Logic Programs

body-oriented nogoods

For a body $\beta = \{p_1, \dots, p_m, not\ p_{m+1}, \dots, not\ p_n\}$, define

$$\begin{aligned} \delta(\beta) &= \{ \mathbf{F}\beta, \mathbf{T}p_1, \dots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \dots, \mathbf{F}p_n \} \\ \Delta(\beta) &= \{ \{ \mathbf{T}\beta, \mathbf{F}p_1 \}, \dots, \{ \mathbf{T}\beta, \mathbf{F}p_m \}, \{ \mathbf{T}\beta, \mathbf{T}p_{m+1} \}, \dots, \{ \mathbf{T}\beta, \mathbf{T}p_n \} \} \end{aligned}$$

For example, for body $\{x, not\ y\}$, we obtain

$\begin{array}{l} \cdots \leftarrow x, not\ y \\ \vdots \\ \cdots \leftarrow x, not\ y \end{array}$	$\begin{aligned} \delta(\{x, not\ y\}) &= \{ \mathbf{F}\{x, not\ y\}, \mathbf{T}x, \mathbf{F}y \} \\ \Delta(\{x, not\ y\}) &= \{ \{ \mathbf{T}\{x, not\ y\}, \mathbf{F}x \}, \{ \mathbf{T}\{x, not\ y\}, \mathbf{T}y \} \} \end{aligned}$
---	---

For nogood $\delta(\{x, not\ y\}) = \{ \mathbf{F}\{x, not\ y\}, \mathbf{T}x, \mathbf{F}y \}$, the signed literal

- $\mathbf{T}\{x, not\ y\}$ is unit-resulting wrt assignment $(\mathbf{T}x, \mathbf{F}y)$ and
- $\mathbf{T}y$ is unit-resulting wrt assignment $(\mathbf{F}\{x, not\ y\}, \mathbf{T}x)$.

259

Characterization of answer sets

for tight logic programs

Theorem Let Π be a tight logic program and

$$\Delta_{\Pi} = \{\delta(p) \mid p \in \text{atom}(\Pi)\} \cup \{\delta \in \Delta(p) \mid p \in \text{atom}(\Pi)\} \\ \cup \{\delta(\beta) \mid \beta \in \text{body}(\Pi)\} \cup \{\delta \in \Delta(\beta) \mid \beta \in \text{body}(\Pi)\}$$

Then, $X \subseteq \text{atom}(\Pi)$ is an answer set of Π

iff

$X = A^T \cap \text{atom}(\Pi)$ for a (unique) solution A for Δ_{Π} .

☞ The set of nogoods Δ_{Π} captures inferences from the (program and its) Clark completion.

260

- Inferences from nogood $\delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \dots, \mathbf{F}\beta_k\}$ correspond to those from Tableau rules

FFA: $(\neg p_{\beta_1} \wedge \dots \wedge \neg p_{\beta_m}) \rightarrow \neg p$

BTA: $p \rightarrow (p_{\beta_1} \vee \dots \vee p_{\beta_m})$

both being equivalent to $p_{\beta_1} \vee \dots \vee p_{\beta_m} \vee \neg p$

262

Atom-oriented Nogoods and Tableau Rules

- Tableau rules FTA, BFA, FFA, and BTA are atom-oriented.
- For an atom p such that $\text{body}(p) = \{\beta_1, \dots, \beta_m\}$, consider the equivalence:

$$p \leftrightarrow p_{\beta_1} \vee \dots \vee p_{\beta_k}$$

- Inferences from nogoods $\Delta(p) = \{\{\mathbf{F}p, \mathbf{T}\beta_1\}, \dots, \{\mathbf{F}p, \mathbf{T}\beta_k\}\}$ correspond to those from Tableau rules

FTA: $(p_{\beta_1} \vee \dots \vee p_{\beta_m}) \rightarrow p$

BFA: $\neg p \rightarrow (\neg p_{\beta_1} \wedge \dots \wedge \neg p_{\beta_m})$

both being equivalent to $(\neg p_{\beta_1} \vee p) \wedge \dots \wedge (\neg p_{\beta_m} \vee p)$

261

body-oriented Nogoods and Tableau Rules

- Tableau rules FTB, BFB, FFB, and BTB are body-oriented.
- For a body $\beta = \{p_1, \dots, p_m, \neg p_{m+1}, \dots, \neg p_n\}$, consider the equivalence:

$$p_{\beta} \leftrightarrow p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n$$

- Inferences from nogood $\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \dots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \dots, \mathbf{F}p_n\}$ correspond to those from Tableau rules

FTB: $(p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n) \rightarrow p_{\beta}$

BFB: $\neg p_{\beta} \rightarrow (\neg p_1 \vee \dots \vee \neg p_m \vee p_{m+1} \vee \dots \vee p_n)$

both being equivalent to $(\neg p_1 \vee \dots \vee \neg p_m \vee p_{m+1} \vee \dots \vee p_n \vee p_{\beta})$

263

- Inferences from nogoods

$$\Delta(\beta) = \{ \{ \mathbf{T}\beta, \mathbf{F}p_1 \}, \dots, \{ \mathbf{T}\beta, \mathbf{F}p_m \}, \{ \mathbf{T}\beta, \mathbf{T}p_{m+1} \}, \dots, \{ \mathbf{T}\beta, \mathbf{T}p_n \} \}$$

correspond to those from Tableau rules

$$\mathbf{FFB}: (\neg p_1 \vee \dots \vee \neg p_m \vee p_{m+1} \vee \dots \vee p_n) \rightarrow \neg p_\beta$$

$$\mathbf{BTB}: p_\beta \rightarrow (p_1 \wedge \dots \wedge p_m \wedge \neg p_{m+1} \wedge \dots \wedge \neg p_n)$$

both being equivalent to $(p_1 \vee \neg p_\beta) \wedge \dots \wedge (\neg p_{m+1} \vee \neg p_\beta)$

264

Nogoods of Logic Programs

Loop nogoods

For a program Π and some $U \subseteq \text{atom}(\Pi)$, define the *loop nogood* of an atom $p \in U$ as

$$\lambda(p, U) = \{ \mathbf{F}\beta_1, \dots, \mathbf{F}\beta_k, \mathbf{T}p \}$$

$$\begin{aligned} \text{where } \{ \beta_1, \dots, \beta_k \} &= \text{body}(ES_\Pi(L)) \\ &= \{ \text{body}(r) \mid r \in \Pi, \text{head}(r) \in U, \text{body}(r) \cap U = \emptyset \}. \end{aligned}$$

In all, we get the following set of loop nogoods for a program Π :

$$\Lambda_\Pi = \bigcup_{U \subseteq \text{atom}(\Pi), U \neq \emptyset} \{ \lambda(p, U) \mid p \in U \}$$

- The set of loop nogoods Λ_Π denies cyclically supported sets of atoms.

266

Nogoods of Logic Programs

via Loop formulas (cf. Slide 169)

Let Π be a normal logic program.

- For $L \subseteq \text{atom}(\Pi)$, define the *external support* of L for Π as

$$ES_\Pi(L) = \{ r \in \Pi \mid \text{head}(r) \in L, \text{body}(r) \cap L = \emptyset \}.$$

- The *loop formula* of L for Π is

$$\begin{aligned} LF_\Pi(L) &= \left(\bigwedge_{r \in ES_\Pi(L)} \neg \left(\bigwedge_{A \in \text{body}^+(r)} A \wedge \bigwedge_{A \in \text{body}^-(r)} \neg A \right) \right) \rightarrow \bigwedge_{A \in L} \neg A \\ &= \left(\bigwedge_{r \in ES_\Pi(L)} \neg \text{body}(r) \right) \rightarrow \bigwedge_{A \in L} \neg A \\ &= \bigwedge_{A \in L} \left(\bigvee_{r \in ES_\Pi(L)} \text{body}(r) \vee \neg A \right) \end{aligned}$$

- The loop formula of a set L of atoms enforces all elements of L to be false, if L is not *externally supported*.

265

Example

Consider

$$\Pi = \left\{ \begin{array}{ll} x \leftarrow \text{not } y & u \leftarrow x \\ y \leftarrow \text{not } x & u \leftarrow v \\ & v \leftarrow u, y \end{array} \right\}$$

For u in the set $\{u, v\}$, we obtain the loop nogood

$$\lambda(u, \{u, v\}) = \{ \mathbf{F}\{x\}, \mathbf{T}u \}$$

267

Characterization of answer sets

Theorem Let Π be a logic program, and let Δ_Π and Λ_Π be as given above.

Then, $X \subseteq \text{atom}(\Pi)$ is an answer set of Π iff

$X = A^T \cap \text{atom}(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- The nogoods in $\Delta_\Pi \cup \Lambda_\Pi$ describe a set of constraints that must principally be checked for computing answer sets.
- The size of Δ_Π is linear in $\text{atom}(\Pi) \times \text{body}(\Pi)$,
- the one of Λ_Π is exponential in $\text{atom}(\Pi)$.

268

Example

Consider

$$\Pi = \left\{ \begin{array}{ll} x \leftarrow \text{not } y & u \leftarrow x \\ y \leftarrow \text{not } x & u \leftarrow v \\ & v \leftarrow u, y \end{array} \right\}$$

Assignment	Nogood	Type
$F\{\text{not } y\}$	by	choice
Ty	by $\{F\{\text{not } y\}, Fy\}$	$\delta(\{\text{not } y\})$
$T\{\text{not } x\}$	by $\{Ty, F\{\text{not } x\}\}$	$\delta(y)$
Fx	by $\{T\{\text{not } x\}, Tx\}$	$\Delta(\{\text{not } x\})$
$F\{x\}$	by $\{Fx, T\{x\}\}$	$\Delta(\{x\})$
Fv	by $\{F\{x\}, Tu\}$	$\lambda(u, \{u, v\})$
$F\{v\}$	by $\{Fv, T\{v\}\}$	$\Delta(v)$
Fv	by $\{F\{v\}, Tv\}$	$\delta(\{v\})$
$F\{u, y\}$	by $\{Fv, T\{u, y\}\}$	$\Delta(v)$

269

Conflicting Assignment

- The sequence

$$A = (Tx, Ta, Fb, Tc, Ty, Td, Fe, Tf, Tz, Th, Ti, Fj, Fk)$$

constitutes an assignment.

- Tx , Ty , and Tz are *decision literals* (σ_{dl}) assigned by $\text{Select}(\Pi, \nabla, A)$ in Algorithm 2.
- dl indicates the corresponding *decision level*.
- Ta , Fb , Tc , Td , Fe , Tf , Th , Ti , Fj , and Fk are assigned by unit propagation.

Such literals are referred to as *implied literals* ($\bar{\sigma}$).

- The Nogood $\{Fk, Ti, Th, Tf, Fe, Td, Fb\}$ is contained in Assignment A ; disqualifying A as a solution.
- A nogood δ is a *conflict nogood* for an assignment A , if $\delta \subseteq A$.

dl	σ_{dl}	$\bar{\sigma}$
1	Tx	
1		Ta
1		Fb
1		Tc
2	Ty	
2		Td
2		Fe
2		Tf
3	Tz	
3		Th
3		Ti
3		Fj
3		Fk

270

Implied signed literal

- Each implied literal, $\bar{\sigma}$, is unit-resulting for some nogood $\delta \in \Delta_\Pi \cup \nabla$ wrt Assignment A (cf. Algorithm 1).

☞ We keep track of the nogoods implying unit-resulting literals in Algorithm 1.

- We have for each nogood δ implying $\bar{\sigma}$ that
 - $\bar{\sigma}$ is in the assignment and
 - all other literals $\delta \setminus \{\bar{\sigma}\}$ are situated in the assignment *before* $\bar{\sigma}$

➔ Consequently, we can “resolve away” any implied literal in the conflict nogood by applying resolution with an implying nogood.

dl	σ_{dl}	$\bar{\sigma}$	δ
1	Tx		
1		Ta	...
1		Fb	...
1		Tc	...
2	Ty		
2		Td	$\{Fd, Fb, Ty\}$
2		Fe	...
2		Tf	...
3	Tz		
3		Th	$\{Fh, Fb, Fe, Tz\}$
3		Ti	$\{Fi, Th, Td, Tx\}$
3		Fj	$\{Tj, Th, Fb\}$
3		Fk	$\{Tk, Ti, Th, Fe, Fb\}$

271

Example: Resolution

Consider conflict nogood

$$\{Fk, Ti, Th, Tf, Fe, Td, Fb\} .$$

$$\frac{\{Fk, Ti, Th, Tf, Fe, Td, Fb\} \quad \{Fd, Fb, Ty\}}{\{Fk, Ti, Th, Tf, Fe, Fb, Ty\}}$$

$$\frac{\{Fk, Ti, Th, Tf, Fe, Td, Fb\} \quad \{Fh, Fb, Fe, Tz\}}{\{Fk, Ti, Tf, Fe, Td, Fb, Tz\}}$$

$$\vdots$$

$$\frac{\{Fk, Ti, Th, Tf, Fe, Td, Fb\} \quad \{Tk, Ti, Th, Fe, Fb\}}{\{Ti, Th, Tf, Fe, Td, Fb\}}$$

$$\{Ti, Th, Tf, Fe, Td, Fb\}$$

☞ All resolvents are also conflict nogoods!

272

dl	σ_{dl}	$\bar{\sigma}$	δ
1	Tx		
1		Ta	...
1		Fb	...
1		Tc	...
2	Ty		
2		Td	$\{Fd, Fb, Ty\}$
2		Fe	...
2		Tf	...
3	Tz		
3		Th	$\{Fh, Fb, Fe, Tz\}$
3		Ti	$\{Fi, Th, Td, Tx\}$
3		Fj	$\{Tj, Th, Fb\}$
3		Fk	$\{Tk, Ti, Th, Fe, Fb\}$

Conflict nogoods

- Every implied literal in every conflict nogood can be resolved upon to generate a new conflict nogood.
 - Various choices exist for generating a conflict nogood.
- The (almost) standard strategy is the *FirstUIP* scheme; cf. Algorithm 3.
- A literal is a *unique implication point* (UIP) in a conflict nogood, if it is the only literal belonging to the current decision level.
 - The FirstUIP strategy stops when finding the first UIP.
 - The LastUIP strategy stops with the decision literal, always being a UIP.

273

Example: FirstUIP scheme

$$\frac{\{Fk, Ti, Th, Tf, Fe, Td, Fb\} \quad \{Tk, Ti, Th, Fe, Fb\}}{\{Ti, Th, Tf, Fe, Td, Fb\}}$$

$$\{Ti, Th, Tf, Fe, Td, Fb\}$$

$$\frac{\{Ti, Th, Tf, Fe, Td, Fb\} \quad \{Fi, Th, Td, Tx\}}{\{Th, Tf, Fe, Td, Fb, Tx\}}$$

$$\{Th, Tf, Fe, Td, Fb, Tx\}$$

☞ Yellow literals belong to decision level 3.

- Th is the first UIP.
- The complement of Th , viz. Fh , is implied at the greatest decision level of the remaining literals, viz. 2 in $\{Tf, Fe, Td, Fb, Tx\}$.
- The FirstUIP scheme takes advantage of this and backjumps to the decision level at which the complement of the UIP is implied by the conflict nogood.

274

Example: FirstUIP Backjumping

dl	σ_{dl}	$\bar{\sigma}$	δ
1	Tx		
1		Ta	...
1		Fb	...
1		Tc	...
2	Ty		
2		Td	$\{Fd, Fb, Ty\}$
2		Fe	...
2		Tf	...
3	Tz		
3		Th	$\{Fh, Fb, Fe, Tz\}$
3		Ti	$\{Fi, Th, Td, Tx\}$
3		Fj	$\{Tj, Th, Fb\}$
3		Fk	$\{Tk, Ti, Th, Fe, Fb\}$
X			$\{Fk, Ti, Th, Tf, Fe, Td, Fb\}$

Applying the FirstUIP scheme makes us generate the conflict nogood $\{Th, Tf, Fe, Td, Fb, Tx\}$ backjump to level 2, and resume unit propagation.

dl	σ_{dl}	$\bar{\sigma}$	δ
1	Tx		
1		Ta	...
1		Fb	...
1		Tc	...
2	Ty		
2		Td	$\{Fd, Fb, Ty\}$
2		Fe	...
2		Tf	...
2		Fh	$\{Th, Tf, Fe, Td, Fb, Tx\}$

275

Algorithms: Overview

Parameters

- Π — given logic program (along with its set Δ_{Π} of static nogoods)
- ∇ — set of dynamic (=loop and conflict) nogoods
- A — an assignment over $atom(\Pi) \cup body(\Pi)$

Algorithm 1: NogoodPropagation relies on

LocalPropagation(Π, ∇, A) returns an assignments obtained by *unit propagation* on $\Delta_{\Pi} \cup \nabla$

UnfoundedSet(Π, A) returns small loop-encompassing unfounded sets giving rise to unit propagation (if there are any)

Algorithm 2: CDNL-ASP computes an answer set of Π , if one exists, by *conflict driven nogood learning* with First-UIP scheme

Algorithm 3: ConflictAnalysis determines conflict nogoods

276

Algorithm 1: NogoodPropagation

Input : A program Π , a set ∇ of nogoods, and an assignment A .
Output : An extended assignment and set of nogoods.

```

1  $U \leftarrow \emptyset$  // set of unfounded atoms
2 loop
3    $A \leftarrow \text{LocalPropagation}(\Pi, \nabla, A)$  // unit propagation
4   if  $\delta \subseteq A$  for some  $\delta \in \Delta_{\Pi} \cup \nabla$  or  $\text{Tight}(\Pi)$  then
5     return  $(A, \nabla)$ 
6   else
7      $U \leftarrow U \setminus A^F$ 
8     if  $U = \emptyset$  then  $U \leftarrow \text{UnfoundedSet}(\Pi, A)$ 
9     if  $U = \emptyset$  then return  $(A, \nabla)$ 
10    else let  $p \in U$  in
11       $\nabla \leftarrow \nabla \cup \{\lambda(p, U)\}$  // addition of single loop nogood
12      if  $Tp \in A$  then return  $(A, \nabla)$ 
13      else  $A \leftarrow A \circ (Fp)$ 

```

277

Algorithm 2: CDNL-ASP

Input : A program Π .
Output : An answer set of Π .

```

1  $A \leftarrow \emptyset$  // assignment over  $atom(\Pi) \cup Body(\Pi)$ 
2  $\nabla \leftarrow \emptyset$  // set of (dynamic) nogoods
3  $dl \leftarrow 0$  // decision level
4 loop
5    $(A, \nabla) \leftarrow \text{NogoodPropagation}(\Pi, \nabla, A)$ 
6   if  $\varepsilon \subseteq A$  for some  $\varepsilon \in \Delta_{\Pi} \cup \nabla$  then
7     if  $dl = 0$  then return no answer set
8      $(\delta, \sigma_{UIP}, k) \leftarrow \text{ConflictAnalysis}(\varepsilon, \Pi, \nabla, A)$ 
9      $\nabla \leftarrow \nabla \cup \{\delta\}$ 
10     $A \leftarrow A \setminus \{\sigma \in A \mid k < dl(\sigma)\}$ 
11     $dl \leftarrow k$ 
12     $A \leftarrow A \circ (\overline{\sigma_{UIP}})$ 
13  else if  $A^T \cup A^F = atom(\Pi) \cup Body(\Pi)$  then
14    return  $A^T \cap atom(\Pi)$ 
15  else
16     $\sigma_d \leftarrow \text{Select}(\Pi, \nabla, A)$ 
17     $dl \leftarrow dl + 1$ 
18     $A \leftarrow A \circ (\sigma_d)$ 

```

278

Algorithm 3: ConflictAnalysis

Input : A violated nogood δ , a program Π , a set ∇ of nogoods, and an assignment A .
Output : A derived nogood, a UIP, and a decision level.

```

1 let  $\sigma \in \delta$  st  $A = B \circ (\sigma) \circ B'$  and  $\delta \setminus \{\sigma\} \subseteq B$ 
2 while  $\{\rho \in \delta \mid dl(\rho) = dl(\sigma)\} \neq \{\sigma\}$  do
3   let  $\varepsilon \in \Delta_{\Pi} \cup \nabla$  st  $\bar{\sigma} \in \varepsilon$  and  $\varepsilon \setminus \{\bar{\sigma}\} \subseteq B$ 
4    $\delta \leftarrow (\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\bar{\sigma}\})$ 
5   let  $\sigma \in \delta$  st  $B = C \circ (\sigma) \circ C'$  and  $\delta \setminus \{\sigma\} \subseteq C$ 
6    $B \leftarrow C$ 
7  $k \leftarrow \max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\})$ 
8 return  $(\delta, \sigma, k)$ 

```

279

The ASP solver clasp

- combines the *high-level modeling capacities* of ASP with state-of-the-art techniques from the area of *Boolean constraint solving*
- is originally designed and optimized for *conflict-driven* ASP solving
- directly incorporates suitable *data structures*, particularly fitting *backjumping and learning*
- and much more, as you can read in the paper!

280

clasp

Origin University of Potsdam

People M. Gebser, *B. Kaufmann*, *A. Neumann*, T. Schaub

Web <http://www.cs.uni-potsdam.de/clasp>

Category Genuine conflict-driven ASP-solver

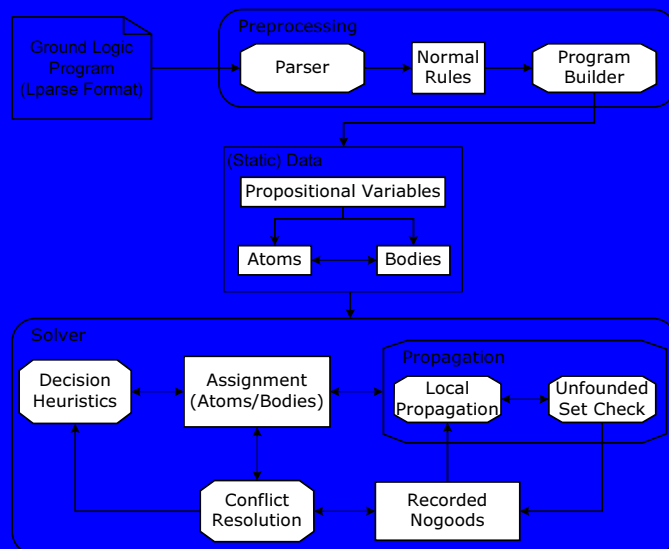
Input format lparse

Extensions

- choice rules
- cardinality constraints

281

The system architecture of clasp



282

Some features of clasp

- Conflict-driven nogood learning and backjumping (primary mode) following the *First-UIP* scheme
- Elementary data structure: *Boolean constraint*
- *Native* and *recorded* nogoods
- Different data structures are used for binary, ternary, and longer nogoods
 - ✦ maintaining two *watch lists* for each variable, storing all longer nogoods that need to be updated, depending on whether the variable becomes true or false (see Slide 285).
- *Nogood deletion* associates an *activity* with each nogood and limits the number of learned nogoods by removing nogoods whenever a threshold is reached.

283

- *Decision heuristics* use different *look-back* strategies which are conflict-oriented and so primarily influenced by conflict resolution.
- Different *restart policies* are available
- *Initial randomized runs*

284

Experiments

clasp (RC2)

clasp_a using backjumping and learning

clasp_b using systematic backtracking and lookahead (but no learning)

versus *smodels* (2.31), *assat* (2.02), *cmodels* (2.12), *smodels_{cc}* (1.08)

setting 2.2GHz PC on Linux, restricted to 900s time and 1GB RAM

measure times in seconds, taking the average of 3 runs

286

Watched literals

Observation A nogood becomes unit, if all but one of its literals become true.

Before Usage of counters following the Dowling/Gallier Algorithm.

Idea

- Assign two non-false *watched literals* per nogood.
- Only if one of the watched literals becomes false, the nogood is inspected:
 - Try to find another non-false watched literal
 - If no such literal exists, the nogood has become unit or obsolete.

Advantage No updating when backjumping!

285

Experiments computing one answer set of tight programs

nr	benchmark	<i>assat</i>	<i>cmodels</i>	<i>smodels_{cc}</i>	<i>clasp_a</i>	<i>clasp_b</i>	<i>smodels</i>
1	dp_10.formula1-i-O2-b12	0.54	11.17	9.33	0.51	299.15	228.93
2	dp_8.fsa-D-i-O2-b8	0.21	0.09	0.16	0.15	1.1	0.17
3	elevator_4-D-s-O2-b10	0.83	1.48	16.62	1.15	168	78.96
4	mmgt_3.fsa-D-i-O2-b10	3.28	5.65	45.5	3.21	66.85	331.89
5	mmgt_4.fsa-D-s-O2-b8	0.49	0.98	3.37	0.3	343.72	142.24
6	des-r3-p6-t1	1.82	1.79	1.28	0.93	—	821.1
7	des-r3-p7-t1	2	2.01	1.9	1.04	—	—
8	des-r3-p8-t1	2.28	2.69	3.54	1.26	—	—
9	des-r3-p9-t1	2.2	2.64	1.82	1.44	—	280.34
10	des-r3-p10-t1	3.3	2.96	2.4	1.62	—	—

from *bounded model checking* (1-5) and *DES cryptanalysis* (6-10)

287

Experiments computing one answer set (ctd) of non-tight programs

nr	benchmark	assat	cmodels	smodels _{cc}	clasp _a	clasp _b	smodels
11	p3_time8	0.91	0.99	17.43	2.81	15.63	30.87
12	p3_time9	1.03	1.18	23.75	3.2	21.38	18.23
13	clumpyhc12_12_08	4.3	3.73	0.29	0.2	—	53.69
14	clumpyhc12_12_10	3.66	4.48	0.26	0.22	—	0.74
15	clumpyhc14_14_08	10.73	8.86	1.59	0.25	—	—
16	clumpyhc14_14_10	27.08	43.18	0.95	0.32	—	—
17	clumpyhc16_16_08	203.05	25.87	6.23	0.46	—	—
18	clumpyhc16_16_10	23.99	18.99	1.42	1.62	—	—
19	clumpyhc18_18_08	—	—	3.13	23.73	—	—
20	clumpyhc18_18_10	—	—	1.61	1.23	—	—

from *blocksworld planning* (11-12) and *Hamiltonian cycles in clumpy graphs* (13-20)

288

Experiments computing one answer set (ctd) of non-tight programs

nr	benchmark	assat	cmodels	smodels _{cc}	clasp _a	clasp _b	smodels
21	gryzzles.6	2.16	117.69	0.23	0.11	23.02	15.46
22	gryzzles.32	2.85	3.63	0.25	0.14	106.24	11.09
23	gryzzles.36	145.74	19.31	0.72	0.55	—	—
24	gryzzles.41	3.57	2.23	0.18	0.11	10.05	25.75
25	gryzzles.47	7.76	46.87	1.23	0.33	—	—
26	yorick.51.n11.len11	36.53	104.98	36.26	6.17	107.02	211.44
27	yoshio.11.n15.len15	—	330.93	64.1	13.34	—	—
28	yoshio.16.n13.len13	84.57	111.11	52.74	8.04	126.18	303.94
29	yoshio.33.n12.len12	15.4	16	9.32	9.06	—	—
30	yoshio.50.n10.len10	22.21	13.57	14.78	1.05	2.69	94.52

from *Hamiltonian paths for the Gryzzles game* (21-25) and *Sokoban* (26-30)

289

Experiments computing all answer set of tight programs

nr	benchmark	#sol	assat	cmodels	smodels _{cc}	clasp _a	clasp _b	smodels
31	dp_10.formula1-i-O2-b12	12600	n/a	22.38	179.74	76.21	—	—
32	dp_8.fsa-D-i-O2-b8	40320	n/a	39.8	78	13.06	40.12	16.53
33	elevator_4-D-s-O2-b10	6240	n/a	14.32	40.48	6.53	171.42	97.58
34	mmgt_3.fsa-D-i-O2-b10	288	n/a	97.26	199.43	28.1	98.77	395.99
35	mmgt_4.fsa-D-s-O2-b8	1344	n/a	28.67	62.15	12.56	417.76	248
36	test12	0	0.09	0.09	0.57	0.08	2.01	0.9
37	test14	0	0.12	0.1	10.96	0.1	10.8	4.21
38	test16	0	0.16	0.12	200.4	0.14	53.56	18.56
39	test18	0	0.19	0.15	—	0.18	260.38	86.57
40	test20	0	0.23	0.18	—	0.23	—	407.35
41	test22	0	0.29	0.21	—	0.27	—	—

from *bounded model checking* (31-35) and *machine code superoptimization* (36-41)

290

Experiments computing all answer set (ctd) of non-tight programs (yet “tight on supported models”)

nr	benchmark	#sol	assat	cmodels	smodels _{cc}	clasp _a	clasp _b	smodels
42	p3_time7	0	0.75	0.92	12.23	2.29	10.61	14.06
43	p3_time8	28	n/a	16.54	18.43	2.93	25.15	53.38
44	p3_time9	3374	n/a	—	56.69	14.75	163.68	417.61
45	p4_time5	0	1.74	1.62	4.41	8.03	8.46	4.1

from *blocksworld planning*

291

Experiments computing all answer set (ctd)

of non-tight programs

nr	benchmark	#sol	assat	cmodels	smodels _{cc}	clasp _a	clasp _b	smodels
46	yorick.51.n11.len10	0	22.81	27.08	11.48	2.08	410.46	836.24
47	yorick.51.n11.len11	512	n/a	—	41.01	11.12	—	—
48	yoshio.11.n15.len14	0	185.47	162.14	38.18	12.11	—	—
49	yoshio.11.n15.len15	512	n/a	—	87.95	19.57	—	—
50	yoshio.16.n13.len12	0	57.94	38.18	30.04	3.82	—	882.44
51	yoshio.16.n13.len13	32	n/a	—	50.81	8.51	—	—
52	yoshio.33.n12.len11	0	16.21	23.69	33.43	8.81	860.09	688.21
53	yoshio.33.n12.len12	114176	n/a	—	—	622.93	—	—
54	yoshio.50.n10.len9	0	33.98	23.25	8.66	2.13	42.38	27.99
55	yoshio.50.n10.len10	384	n/a	—	17.29	4.82	172.11	94.57

from *Sokoban*