# Answer Set Solving in Practice

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

Potassco

# ASP modulo theories: Overview

# Motivation

- Input ASP = DB+KRR+LP+SAT
- Output ASPmT = DB+KRR+LP+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 $\text{ASPmT} = \text{DB}+\text{KRR}+\text{LP}+\text{S}$
- ASP solving *ground* | *solve*
  - ➥ **logic programs with elusive theory atoms**
- Application areas

  Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc

# Motivation

- Input ASP = DB+KRR+LP+SAT
- Output ASPmT = DB+KRR+LP+SMT
- ASP solving *ground* | *solve*

  ➥ **logic programs with elusive theory atoms**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc

# Motivation

- Input ASP = DB+KRR+LP+SAT
- Output ASPmT = 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 ASP = DB+KRR+LP+SAT
- Output ASPmT = (DB+KRR+LP+SAT)mT
- ASP solving *ground* | *solve*
  - ➥ **logic programs with elusive theory atoms**
- Application areas

  Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc

# Motivation

- Input  ASP = DB+KRR+LP+SAT
- Output  ASPmT = (DB+KRR+LP+SAT)mT
- ASP solving  *ground* | *solve*
  - ➥ **logic programs with elusive theory atoms**
- Application areas

  Agents, Assisted Living, Robotics, Planning, Scheduling, Bio- and Cheminformatics, etc

## Motivation

- Input ASP = DB+KRR+LP+SAT
- Output ASPmT = (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 ASP = DB+KRR+LP+SAT
- Output ASPmT = (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

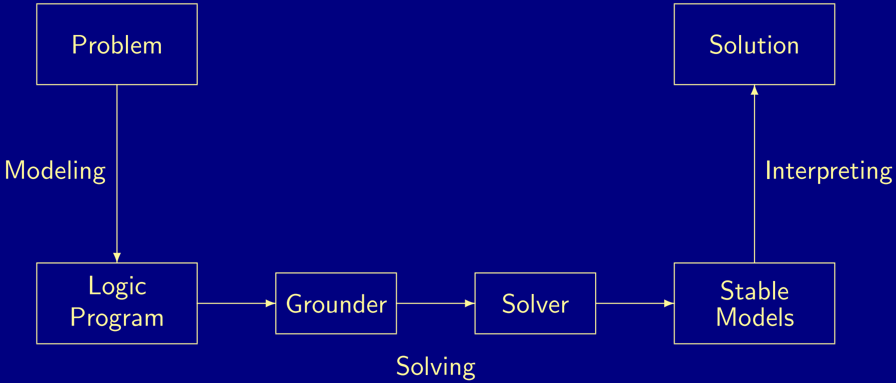
- Input ASP = DB+KRR+LP+SAT
- Output ASPmT = (DB+KRR+LP+SAT)mT
- ASP solving modulo theories *ground % theories* | *solve % theories*

  ➥ **logic programs with elusive theory atoms**

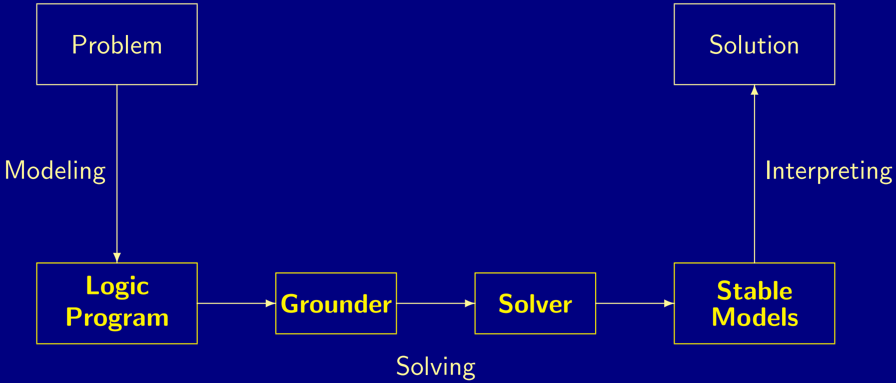
- Application areas

  Agents, Assisted Living, Robotics, Planning, Scheduling,
  Bio- and Cheminformatics, etc

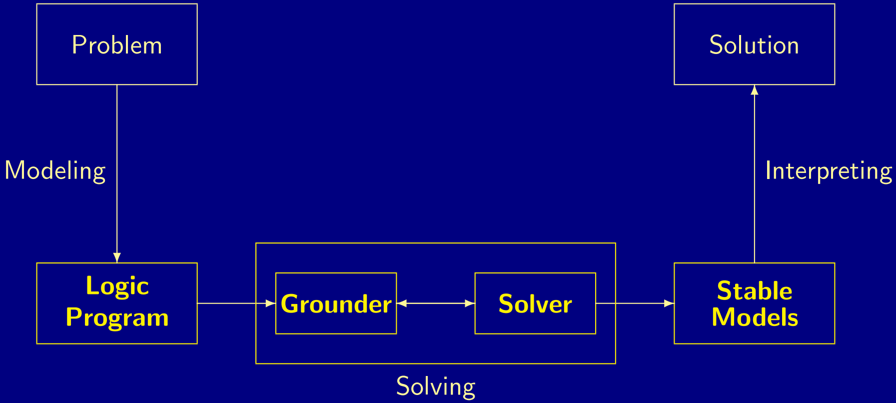
# ASP solving process



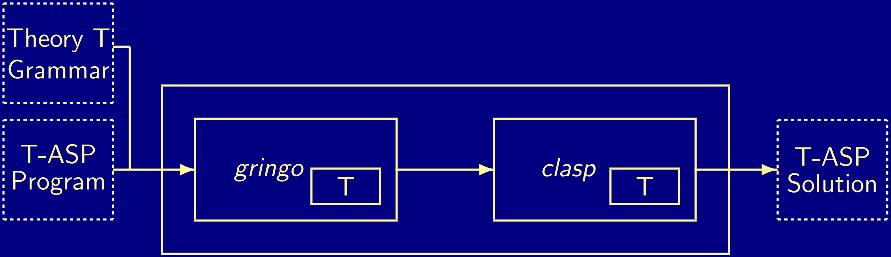
# ASP solving process modulo theories



# ASP solving process modulo theories

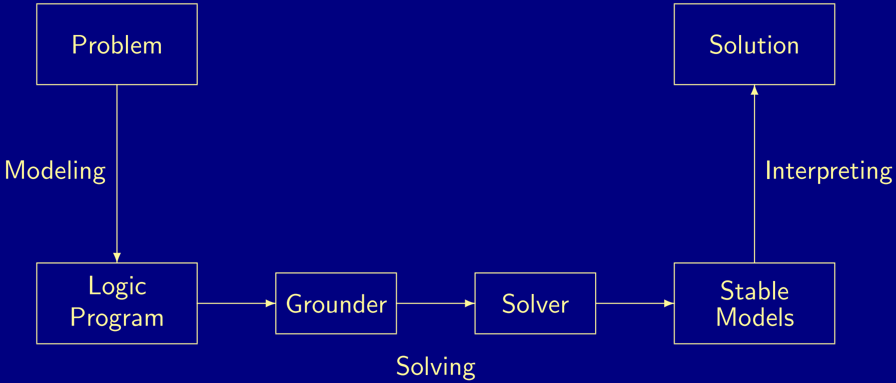


# *clingo*'s approach

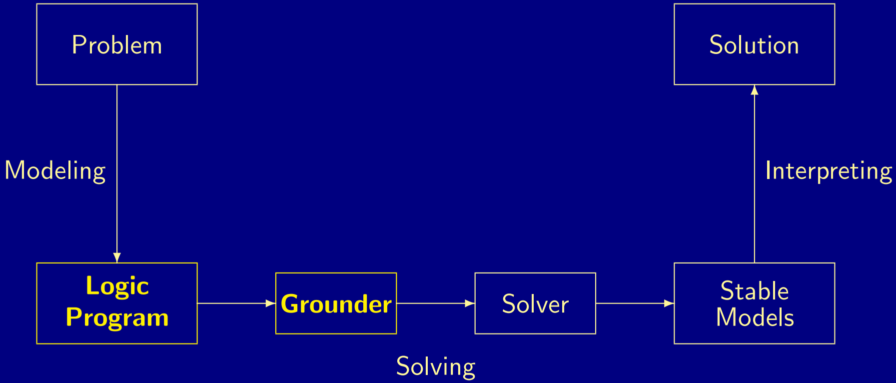


# Outline

# ASP solving process modulo theories



# ASP solving process modulo theories



# Linear constraints

```
#theory csp {
    linear_term {
      + : 5, unary;
      - : 5, unary;
      * : 4, binary, left;
      + : 3, binary, left;
      - : 3, binary, left
    };

    dom_term {
      + : 5, unary;
      - : 5, unary;
      .. : 1, binary, left
    };

    show_term {
      / : 1, binary, left
    };

    minimize_term {
      + : 5, unary;
      - : 5, unary;
      * : 4, binary, left;
      + : 3, binary, left;
      - : 3, 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
}.
```

## send+more=money

$$\begin{array}{ccccc} & s & e & n & d \\ + & m & o & r & e \\ \hline m & o & n & e & y \end{array}$$

Each letter corresponds exactly to one digit and all variables have to be pairwise distinct

$$\begin{array}{ccccc} & 9 & 5 & 6 & 7 \\ + & 1 & 0 & 8 & 5 \\ \hline 1 & 0 & 6 & 5 & 2 \end{array}$$

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

| | s | e | n | d |
|---|---|---|---|---|
| + | m | o | r | e |
| m | o | n | e | y |

Each letter corresponds exactly to one digit and all variables have to be pairwise distinct

| | 9 | 5 | 6 | 7 |
|---|---|---|---|---|
| + | 1 | 0 | 8 | 5 |
| 1 | 0 | 6 | 5 | 2 |

The example has exactly one solution

$$\{ s \mapsto 9, e \mapsto 5, n \mapsto 6, d \mapsto 7, m \mapsto 1, o \mapsto 0, r \mapsto 8, y \mapsto 2 \}$$

Potassco

# 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) }.
```

# 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) }.
```

# 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) }.
```

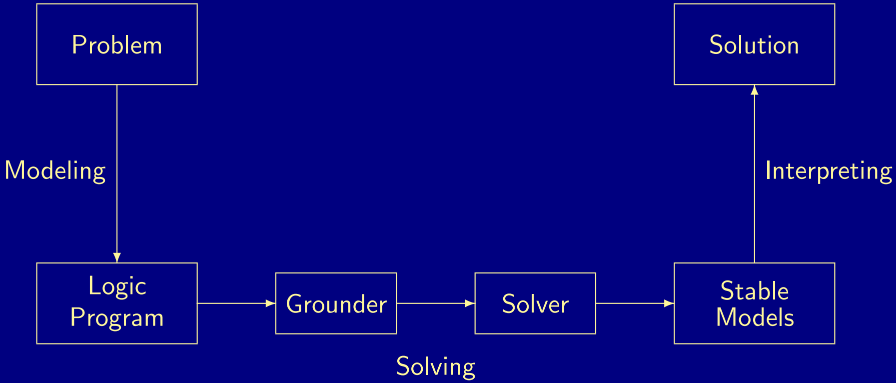
Potassco

# send+more=money

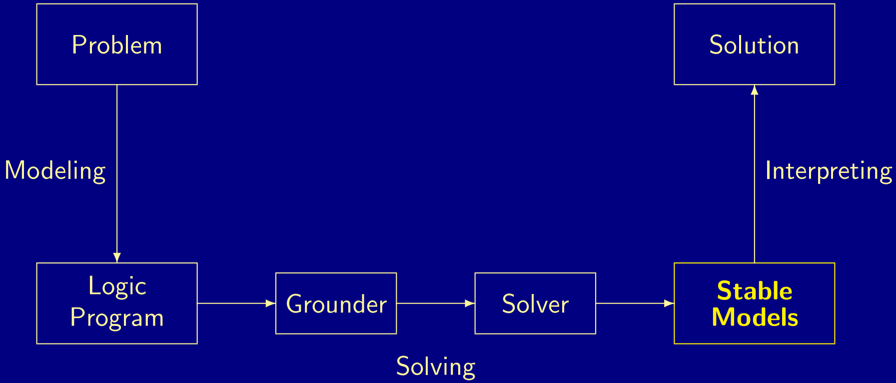
```
digit(1,3,s). digit(2,3,m). digit(sum,4,m).
digit(1,2,e). digit(2,2,o). digit(sum,3,o).
digit(1,1,n). digit(2,1,r). digit(sum,2,n).
digit(1,0,d). digit(2,0,e). digit(sum,1,e).
                            digit(sum,0,y).
base(10).
exp(0). exp(1). exp(2). exp(3). exp(4).

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}=

&sum{  1000*s;   100*e;   10*n;   1*d;
       1000*m;   100*o;   10*r;   1*e;
    -10000*m; -1000*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

# ASP solving process modulo theories



# ASP solving process modulo theories



# 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 $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\mathbb{T}$-stable model of a program $P$ if there is some $\mathbb{T}$-solution $\mathcal{S}$ such that $X$ is a (regular) stable model of the program

$$
\begin{aligned}
P \cup \{a \leftarrow \mid a \in (\mathcal{T}_e \setminus \mathit{head}(P)) \cap \mathcal{S}\}\\
\cup \{\leftarrow \sim a \mid a \in (\mathcal{T}_e \cap \mathit{head}(P)) \cap \mathcal{S}\}\\
\cup \{\{a\} \leftarrow \mid a \in (\mathcal{T}_i \setminus \mathit{head}(P)) \cap \mathcal{S}\}\\
\cup \{\leftarrow a \mid a \in (\mathcal{T} \cap \mathit{head}(P)) \setminus \mathcal{S}\}
\end{aligned}
$$

# 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 $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\mathbb{T}$-stable model of a program $P$ if there is some $\mathbb{T}$-solution $\mathcal{S}$ such that $X$ is a (regular) stable model of the program

$$\begin{aligned} P & \cup \{a \leftarrow \mid a \in (\mathcal{T}_e \setminus head(P)) \cap \mathcal{S}\} \\ & \cup \{\leftarrow {\sim}a \mid a \in (\mathcal{T}_e \cap head(P)) \cap \mathcal{S}\} \\ & \cup \{\{a\} \leftarrow \mid a \in (\mathcal{T}_i \setminus head(P)) \cap \mathcal{S}\} \\ & \cup \{\leftarrow a \mid a \in (\mathcal{T} \cap head(P)) \setminus \mathcal{S}\} \end{aligned}$$

Potassco

# 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 $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\mathbb{T}$-stable model of a program $P$ if there is some $\mathbb{T}$-solution $\mathcal{S}$ such that $X$ is a (regular) stable model of the program

$$
\begin{aligned}
P & \cup \{a \leftarrow \mid a \in (\mathcal{T}_e \setminus head(P)) \cap \mathcal{S}\} \\
& \cup \{\leftarrow {\sim}a \mid a \in (\mathcal{T}_e \cap head(P)) \cap \mathcal{S}\} \\
& \cup \{\{a\} \leftarrow \mid a \in (\mathcal{T}_i \setminus head(P)) \cap \mathcal{S}\} \\
& \cup \{\leftarrow a \mid a \in (\mathcal{T} \cap head(P)) \setminus \mathcal{S}\}
\end{aligned}
$$

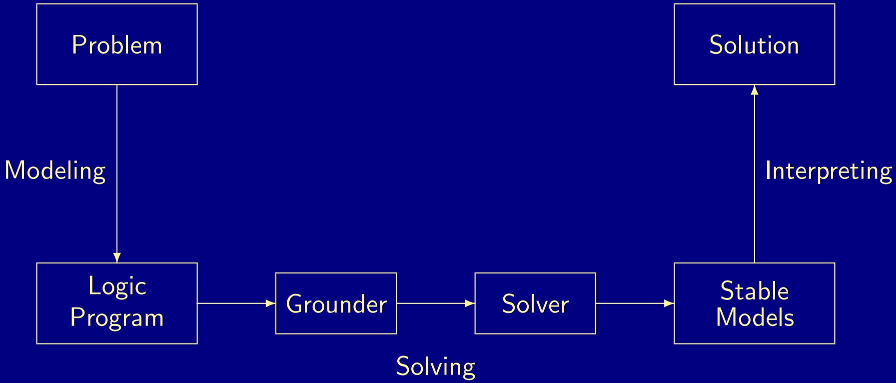
# ASP modulo theories

- 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, $\mathcal{T}_e$, or
  - non-strict only implying the associated constraint, $\mathcal{T}_i$.
- Informally, a set $X \subseteq \mathcal{A} \cup \mathcal{T}$ of atoms is a $\mathbb{T}$-stable model of a program $P$ if there is some $\mathbb{T}$-solution $\mathcal{S}$ such that $X$ is a (regular) stable model of the program

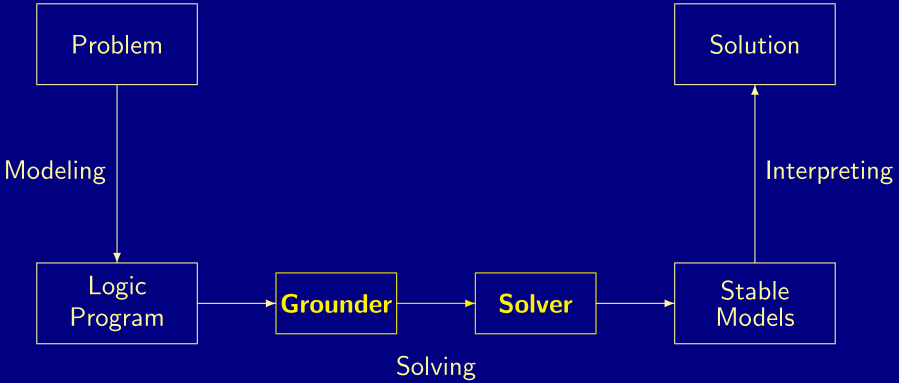
$$
\begin{aligned}
P \cup \{a \leftarrow {} & \mid a \in (\mathcal{T}_e \setminus \mathit{head}(P)) \cap \mathcal{S}\} \\
\cup \{\leftarrow {\sim}a & \mid a \in (\mathcal{T}_e \cap \mathit{head}(P)) \cap \mathcal{S}\} \\
\cup \{\{a\} \leftarrow {} & \mid a \in (\mathcal{T}_i \setminus \mathit{head}(P)) \cap \mathcal{S}\} \\
\cup \{\leftarrow a & \mid a \in (\mathcal{T} \cap \mathit{head}(P)) \setminus \mathcal{S}\}
\end{aligned}
$$

# Outline

# ASP solving process modulo theories



# ASP solving process modulo theories



# *aspif* 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
```

# *aspif* 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
```

# *aspif* overview

- Rule statements
- Minimize statements
- Projection statements
- Output statements
- External statements
- Assumption statements
- Heuristic statements
- Edge statements
- Theory terms and atoms
- Comments

# *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 }.
```

```
asp 1 0 0
1 0 1 1 0 0
1 0 1 2 0 0
1 0 1 3 0 0
1 0 1 4 0 0
1 0 1 5 0 0
1 0 1 6 0 0
4 7 task(1) 0
4 7 task(2) 0
4 15 duration(1,200) 0
4 15 duration(2,400) 0
9 0 1 200
9 0 3 400
9 0 6 1
9 0 11 2
9 1 0 4 diff
9 1 2 2 <=
9 1 4 1 -
9 1 5 3 end
9 1 8 3 beg
9 2 7 5 1 6
9 2 9 8 1 6
9 2 10 4 2 7 9
9 2 12 5 1 11
9 2 13 8 1 11
9 2 14 4 2 12 13
9 4 0 1 10 0
9 4 1 1 14 0
9 6 5 0 1 0 2 1
9 6 6 0 1 1 2 3
0
```

# *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 }.
```

```
asp 1 0 0
1 0 1 1 0 0
1 0 1 2 0 0
1 0 1 3 0 0
1 0 1 4 0 0
1 0 1 5 0 0
1 0 1 6 0 0
4 7 task(1) 0
4 7 task(2) 0
4 15 duration(1,200) 0
4 15 duration(2,400) 0
9 0 1 200
9 0 3 400
9 0 6 1
9 0 11 2
9 1 0 4 diff
9 1 2 2 <=
9 1 4 1 -
9 1 5 3 end
9 1 8 3 beg
9 2 7 5 1 6
9 2 9 8 1 6
9 2 10 4 2 7 9
9 2 12 5 1 11
9 2 13 8 1 11
9 2 14 4 2 12 13
9 4 0 1 10 0
9 4 1 1 14 0
9 6 5 0 1 0 2 1
9 6 6 0 1 1 2 3
0
```

# *aspif* theory example

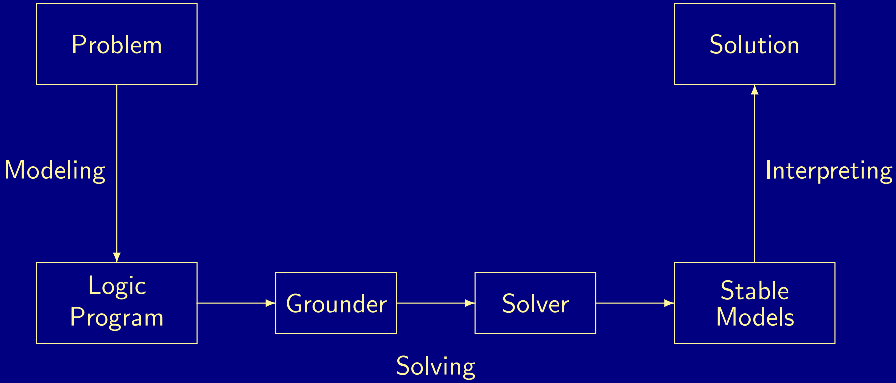
```
task(1).
task(2).

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!

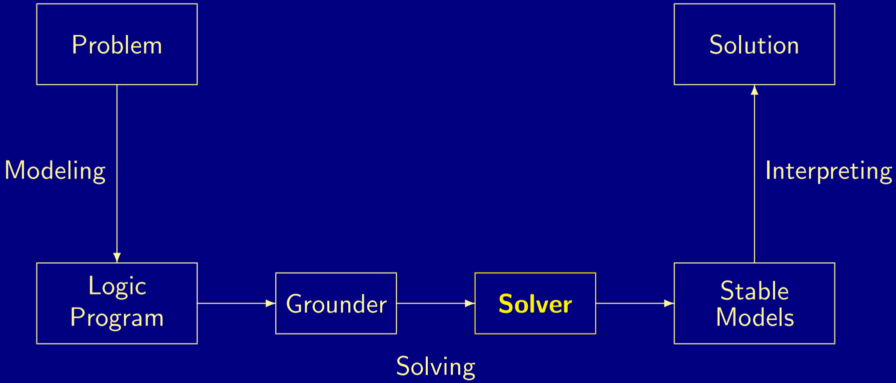
```
asp 1 0 0
1 0 1 1 0 0
1 0 1 2 0 0
1 0 1 3 0 0
1 0 1 4 0 0
1 0 1 5 0 0
1 0 1 6 0 0
4 7 task(1) 0
4 7 task(2) 0
4 15 duration(1,200) 0
4 15 duration(2,400) 0
9 0 1 200
9 0 3 400
9 0 6 1
9 0 11 2
9 1 0 4 diff
9 1 2 2 <=
9 1 4 1 -
9 1 5 3 end
9 1 8 3 beg
9 2 7 5 1 6
9 2 9 8 1 6
9 2 10 4 2 7 9
9 2 12 5 1 11
9 2 13 8 1 11
9 2 14 4 2 12 13
9 4 0 1 10 0
9 4 1 1 14 0
9 6 5 0 1 0 2 1
9 6 6 0 1 1 2 3
0
```

# Outline

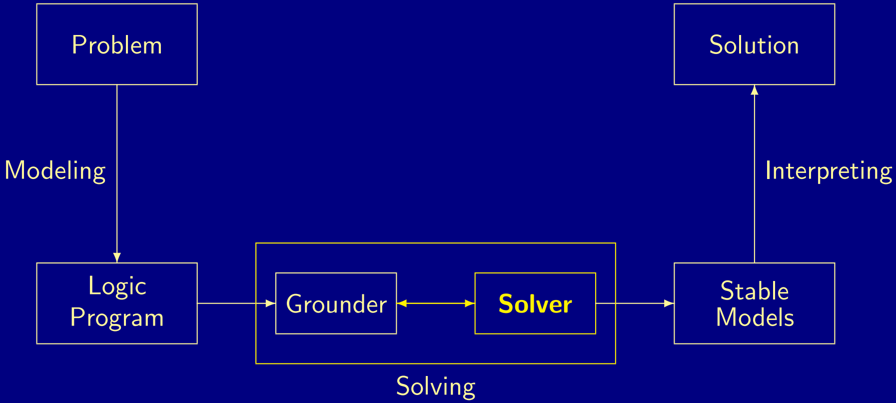
# ASP solving process modulo theories



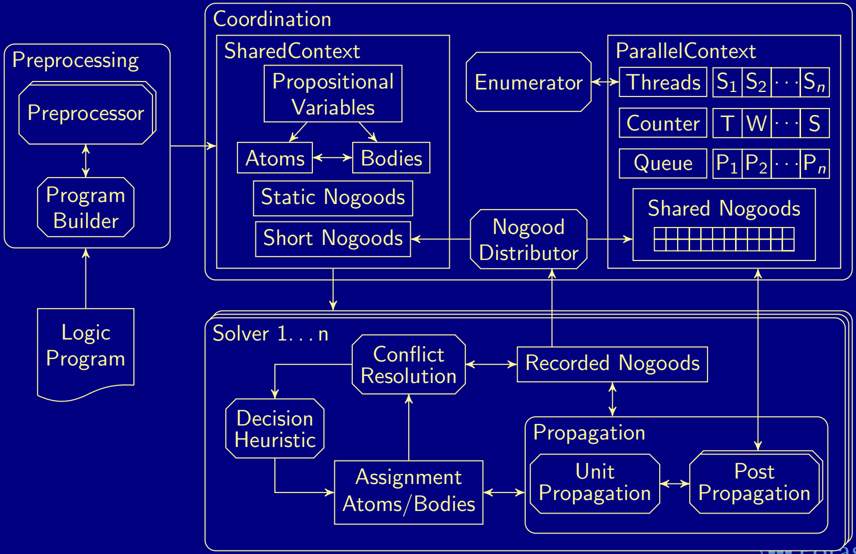
# ASP solving process modulo theories



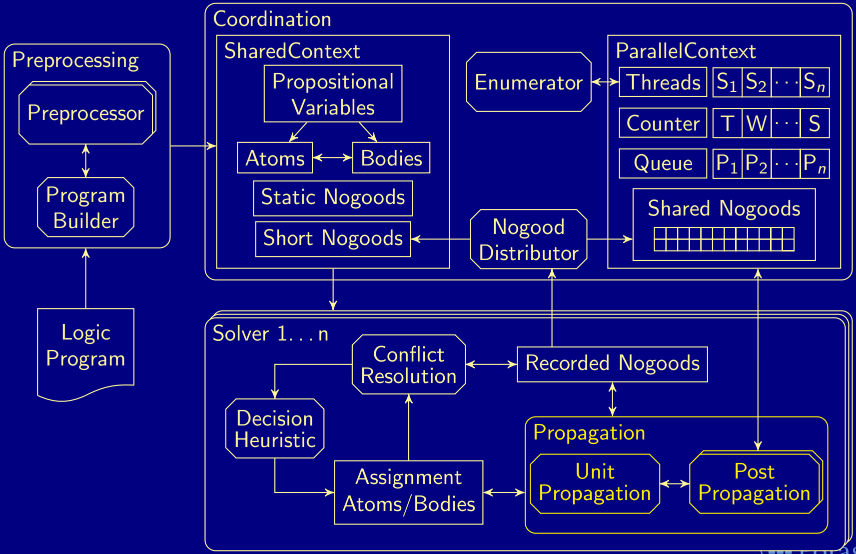
# ASP solving process modulo theories



# Architecture of *clasp*



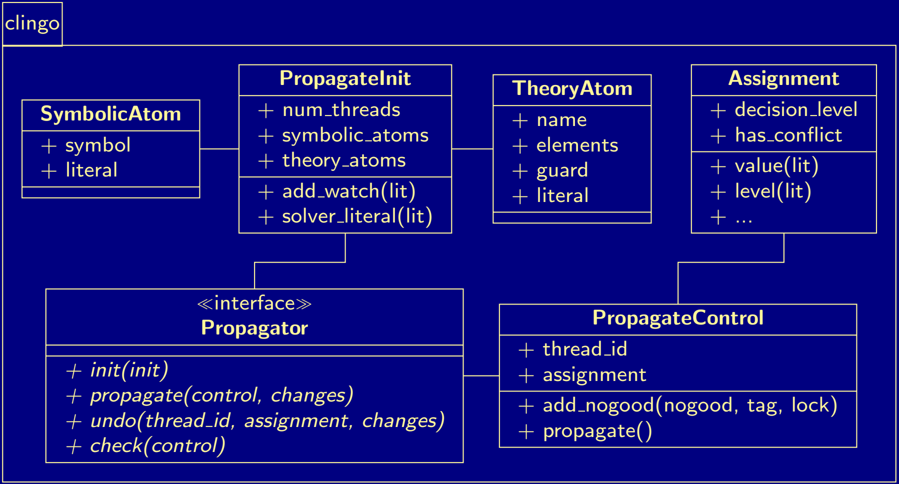
# Architecture of *clasp*



# Conflict-driven constraint learning modulo theories

```
(I)  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 δ ∈ Δ_T is violated for T ∈ 𝕋 then record δ   // theory propagator's check
              else return variable assignment                       // 𝕋-stable model found
           else
(P)           propagate theories T ∈ 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



# The *dot* propagator

```
#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.
```

# Outline

# Difference logic propagation

| Problem | # | ASP | | ASP modulo *DL* (stateless) | | | | ASP modulo *DL* (stateful) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | defined | | external | | defined | | external | |
| | | T | TO | T | TO | T | TO | T | TO | T | TO |
| Flow shop | 120 | 569 | 110 | 283 | 40 | 382 | 70 | **177** | **30** | 281 | 50 |
| Job shop | 80 | 600 | 80 | 600 | 80 | 600 | 80 | **37** | **0** | 43 | **0** |
| Open shop | 60 | 405 | 40 | 214 | 20 | 213 | 20 | **2** | **0** | **2** | **0** |
| Total | 260 | 525 | 230 | 366 | 140 | 398 | 170 | **72** | **30** | 109 | 50 |

- 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

Potassco

# Difference logic propagation

| Problem | # | ASP | | ASP modulo *DL* (stateless) | | | | ASP modulo *DL* (stateful) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | defined | | external | | defined | | external | |
| | | T | TO | T | TO | T | TO | T | TO | T | TO |
| Flow shop | 120 | 569 | 110 | 283 | 40 | 382 | 70 | **177** | **30** | 281 | 50 |
| Job shop | 80 | 600 | 80 | 600 | 80 | 600 | 80 | **37** | **0** | 43 | **0** |
| Open shop | 60 | 405 | 40 | 214 | 20 | 213 | 20 | **2** | **0** | **2** | **0** |
| Total | 260 | 525 | 230 | 366 | 140 | 398 | 170 | **72** | **30** | 109 | 50 |

- 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

Potassco

# Difference logic propagation

| | | ASP | | ASP modulo *DL* (stateless) | | | | ASP modulo *DL* (stateful) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | defined | | external | | defined | | external | |
| Problem | # | T | TO | T | TO | T | TO | T | TO | T | TO |
| Flow shop | 120 | 569 | 110 | 283 | 40 | 382 | 70 | **177** | **30** | 281 | 50 |
| Job shop | 80 | 600 | 80 | 600 | 80 | 600 | 80 | **37** | **0** | 43 | **0** |
| Open shop | 60 | 405 | 40 | 214 | 20 | 213 | 20 | **2** | **0** | **2** | **0** |
| Total | 260 | 525 | 230 | 366 | 140 | 398 | 170 | **72** | **30** | 109 | 50 |

- 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

Potassco

# Difference logic propagation

| Problem | # | ASP T | ASP TO | ASP modulo *DL* (stateless) defined T | defined TO | external T | external TO | ASP modulo *DL* (stateful) defined T | defined TO | external T | external TO |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Flow shop | 120 | 569 | 110 | 283 | 40 | 382 | 70 | **177** | **30** | 281 | 50 |
| Job shop | 80 | 600 | 80 | 600 | 80 | 600 | 80 | **37** | **0** | 43 | **0** |
| Open shop | 60 | 405 | 40 | 214 | 20 | 213 | 20 | **2** | **0** | **2** | **0** |
| Total | 260 | 525 | 230 | 366 | 140 | 398 | 170 | **72** | **30** | 109 | 50 |

- 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

Potassco

# Difference logic propagation

| Problem | # | ASP |  | ASP modulo *DL* (stateless) |  |  |  | ASP modulo *DL* (stateful) |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  | defined |  | external |  | defined |  | external |  |
|  |  | T | TO | T | TO | T | TO | T | TO | T | TO |
| Flow shop | 120 | 569 | 110 | 283 | 40 | 382 | 70 | **177** | **30** | 281 | 50 |
| Job shop | 80 | 600 | 80 | 600 | 80 | 600 | 80 | **37** | **0** | 43 | **0** |
| Open shop | 60 | 405 | 40 | 214 | 20 | 213 | 20 | **2** | **0** | **2** | **0** |
| Total | 260 | 525 | 230 | 366 | 140 | 398 | 170 | **72** | **30** | 109 | 50 |

- 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

Potassco

# Outline

## Builtin acyclicity checking

- Edge statement

$$\#\texttt{edge}\ (u,v)\ :\ l_1,\dots,l_n. \tag{3}$$

- 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

$$(\{u,v \mid X \models l_1,\dots,l_n,(3)\in P\},\{(u,v)\mid X\models l_1,\dots,l_n,(3)\in P\})$$

  is acyclic

# Builtin acyclicity checking

- Edge statement

$$\#\text{edge }(u,v) \ : \ l_1,\dots,l_n. \tag{3}$$

- 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

     $$(\{u,v \mid X \models l_1,\dots,l_n,(3) \in P\},\{(u,v) \mid X \models l_1,\dots,l_n,(3) \in P\})$$

     is acyclic

# Outline

# Constraint Satisfaction Problem

- 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 $dom(v) \in D$;
  - 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 $R \subseteq dom(v_1) \times \cdots \times dom(v_k)$

# Constraint Satisfaction Problem

- 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 $dom(v) \in D$;
  - 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 $S = (v_1, \dots, v_k)$, we have $R \subseteq dom(v_1) \times \cdots \times dom(v_k)$

# Constraint Satisfaction Problem

- 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 $dom(v) \in D$;
  - 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 $R \subseteq dom(v_1) \times \cdots \times dom(v_k)$

# Example

$$\begin{array}{ccccc} & s & e & n & d \\ + & m & o & r & e \\ \hline m & o & n & e & y \end{array}$$

Each letter corresponds exactly to one digit and all variables have to be pairwise distinct

$$V = \{s, e, n, d, m, o, r, y\}$$

$$D = \{dom(v) = \{0, \ldots, 9\} \mid v \in V\}$$

$$\begin{array}{rll} C = \{ & (\vec{V}, & allDistinct(V)), \\ & (\vec{V}, & s \times 1000 + e \times 100 + n \times 10 + d + \\ & & m \times 1000 + o \times 100 + r \times 10 + e == \\ & & m \times 10000 + o \times 1000 + n \times 100 + e \times 10 + y), \\ & ((m), m == 1) \} & \end{array}$$

# Example

$$
\begin{array}{ccccc}
 & s & e & n & d \\
+ & m & o & r & e \\
\hline
m & o & n & e & y
\end{array}
$$

Each letter corresponds exactly to one digit and all variables have to be pairwise distinct

$$V = \{s, e, n, d, m, o, r, y\}$$

$$D = \{dom(v) = \{0, \ldots, 9\} \mid v \in V\}$$

$$
\begin{aligned}
C = \{ & (\vec{V}, \quad allDistinct(V)), \\
& (\vec{V}, \quad s \times 1000 + e \times 100 + n \times 10 + d + \\
& \qquad m \times 1000 + o \times 100 + r \times 10 + e == \\
& \qquad m \times 10000 + o \times 1000 + n \times 100 + e \times 10 + y), \\
& ((m), m == 1) \}
\end{aligned}
$$

Potassco

# Example

| | | | | |
|---|---|---|---|---|
| | s | e | n | d |
| + | m | o | r | e |
| m | o | n | e | y |

Each letter corresponds exactly to one digit and all variables have to be pairwise distinct

| | | | | |
|---|---|---|---|---|
| | 9 | 5 | 6 | 7 |
| + | 1 | 0 | 8 | 5 |
| 1 | 0 | 6 | 5 | 2 |

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 $A : V \to \bigcup_{v \in V} dom(v)$ 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 Answer Set Programming

- 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 $head(r) \in \mathcal{A}$ for each $r \in P$

- Given a set of literals $B$ and some set $\mathcal{B}$ of atoms, we define
  $B|_{\mathcal{B}} = (B^+ \cap \mathcal{B}) \cup \{\sim a \mid a \in B^- \cap \mathcal{B}\}$

# Constraint Answer Set Programming

- 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 $head(r) \in \mathcal{A}$ for each $r \in P$
- Given a set of literals $B$ and some set $\mathcal{B}$ of atoms, we define
  $B|_{\mathcal{B}} = (B^+ \cap \mathcal{B}) \cup \{\sim a \mid a \in B^- \cap \mathcal{B}\}$

# Constraint Answer Set Programming

- We identify constraint atoms with constraints via a function

$$\gamma : \mathcal{C} \rightarrow C$$

- 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.

$$\gamma(x < y) \ = \ \gamma(((-y-1) \leq -(x+1)) \wedge (x \neq y))$$

- A constraint logic program $P$ is associated with a CSP as follows
  - $C[P] = \gamma(atom(P) \cap \mathcal{C})$,
  - $V[P]$ is obtained from the constraint scopes in $C[P]$,
  - $D[P]$ is provided by a declaration

# Constraint Answer Set Programming

- We identify constraint atoms with constraints via a function

  $$\gamma : \mathcal{C} \rightarrow C$$

- 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.

  $$\gamma(x < y) \;=\; \gamma(((-y - 1) \leq -(x + 1)) \wedge (x \neq y))$$

- A constraint logic program $P$ is associated with a CSP as follows
  - $C[P] = \gamma(atom(P) \cap \mathcal{C})$,
  - $V[P]$ is obtained from the constraint scopes in $C[P]$,
  - $D[P]$ is provided by a declaration

# Constraint Answer Set Programming

- We identify constraint atoms with constraints via a function

$$\gamma : \mathcal{C} \rightarrow C$$

- 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.

$$\gamma(x < y) \ = \ \gamma(((-y - 1) \leq -(x + 1)) \wedge (x \neq y))$$

- A constraint logic program $P$ is associated with a CSP as follows
  - $C[P] = \gamma(atom(P) \cap \mathcal{C})$,
  - $V[P]$ is obtained from the constraint scopes in $C[P]$,
  - $D[P]$ is provided by a declaration

Potassco

# 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

$$
\begin{aligned}
P^A \quad = \quad \{ \, head(r) \leftarrow body(r)|_{\mathcal{A}} \mid r \in P, \\
\gamma(body(r)|_{\mathcal{C}}{}^{+}) \subseteq sat_{C[P]}(A), \\
\gamma(body(r)|_{\mathcal{C}}{}^{-}) \cap sat_{C[P]}(A) = \emptyset \, \}
\end{aligned}
$$

- 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 $P^A$.

  That is, if $X$ is the $\subseteq$-smallest model of $(P^A)^X$

# Constraint Answer Set Programming

- Let $P$ be a constraint logic program over $\mathcal{A} \cup \mathcal{C}$ and let $A : V[P] \to D[P]$ be an assignment,

  define the constraint reduct of as $P$ wrt $A$ as follows

$$\begin{aligned} P^A \quad = \quad \{ \; head(r) \leftarrow body(r)|_{\mathcal{A}} \mid r \in P, & \\ \gamma(body(r)|_{\mathcal{C}}{}^{+}) \subseteq sat_{C[P]}(A), & \\ \gamma(body(r)|_{\mathcal{C}}{}^{-}) \cap sat_{C[P]}(A) = \emptyset \; \} & \end{aligned}$$

- 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 $(P^A)^X$

# Constraint Answer Set Programming

- Let $P$ be a constraint logic program over $\mathcal{A} \cup \mathcal{C}$ and let $A : V[P] \to D[P]$ be an assignment,

  define the constraint reduct of as $P$ wrt $A$ as follows

$$
\begin{aligned}
P^A \quad = \quad \{\ head(r) \leftarrow body(r)|_{\mathcal{A}} \mid r \in P,\\
\gamma(body(r)|_{\mathcal{C}}{}^{+}) \subseteq sat_{C[P]}(A),\\
\gamma(body(r)|_{\mathcal{C}}{}^{-}) \cap sat_{C[P]}(A) = \emptyset\ \}
\end{aligned}
$$

- 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 $(P^A)^X$

# 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
- *aspmt*, *dlvhex*, *ezcsp*, *gasp*, *inca*, ...

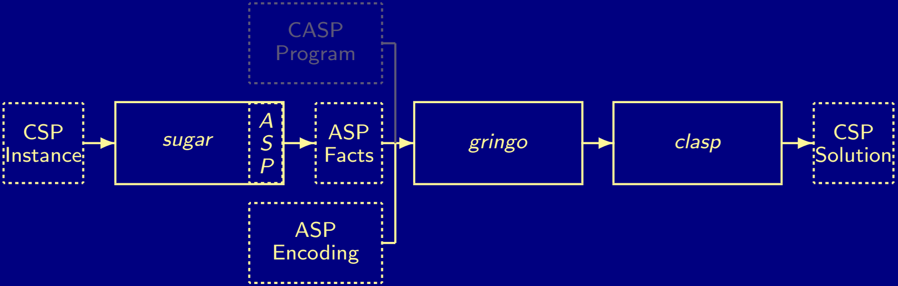
# Some Constraint Answer Set Programming (CASP) systems

- *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
- *aspmt*, *dlvhex*, *ezcsp*, *gasp*, *inca*, ...

# Some Constraint Answer Set Programming (CASP) systems

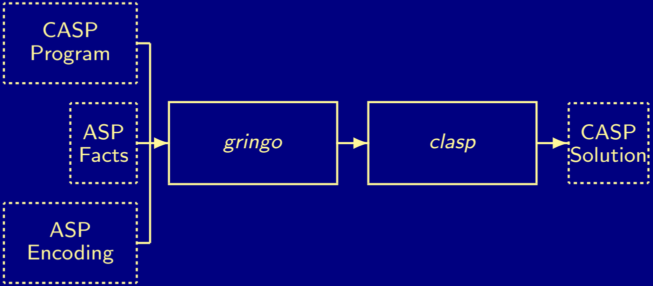
- *adsolver*
  - extension of ASP solver *smodels*
- *clingcon*
  - extension of ASP system *clingo* (viz. *gringo* and *clasp*)
  - lazy approach
- *aspartame*
  - translational approach (independent of ASP system)
  - eager approach
- *aspmt*, *dlvhex*, *ezcsp*, *gasp*, *inca*, . . .

# *aspartame*'s eager approach



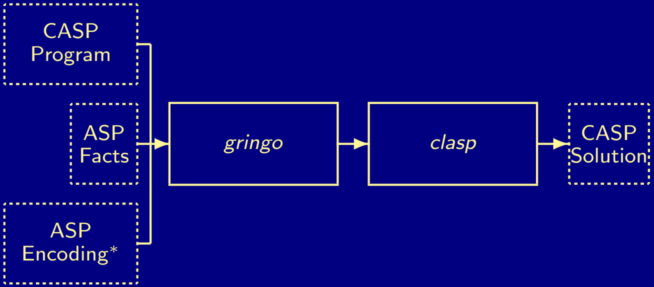

* based on order-encoding for CSPs

# *aspartame*'s eager approach



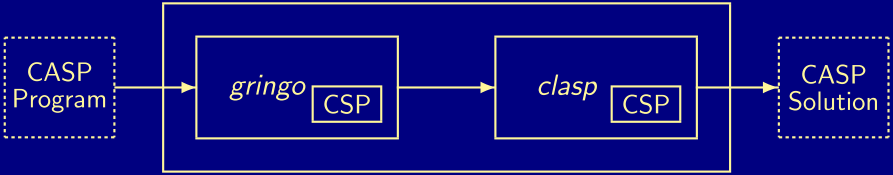

* based on order-encoding for CSPs

# *aspartame*'s eager approach



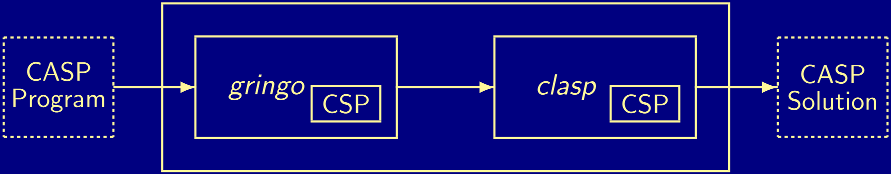

* based on order-encoding for CSPs

# *clingcon*'s lazy approach



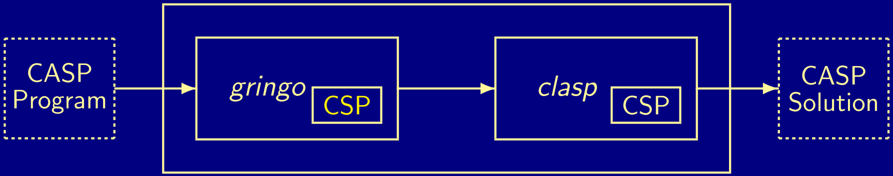

- *clingcon* 1
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

# *clingcon*'s lazy approach



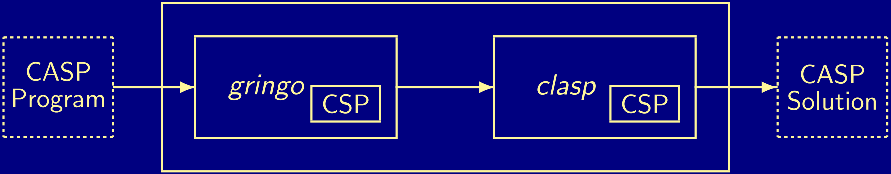

- *clingcon* 1
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

# *clingcon*'s lazy approach



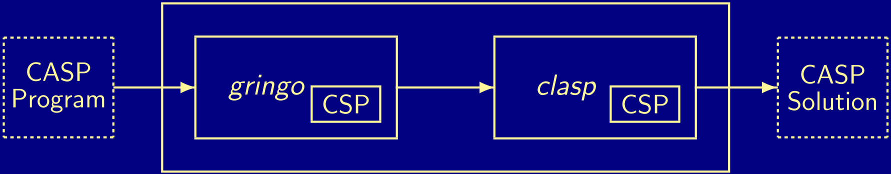

- *clingcon* 1
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

# *clingcon*'s lazy approach



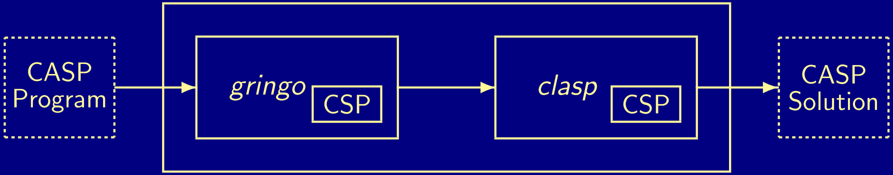

- *clingcon* 1
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

# *clingcon*'s lazy approach




- *clingcon* 1+2
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

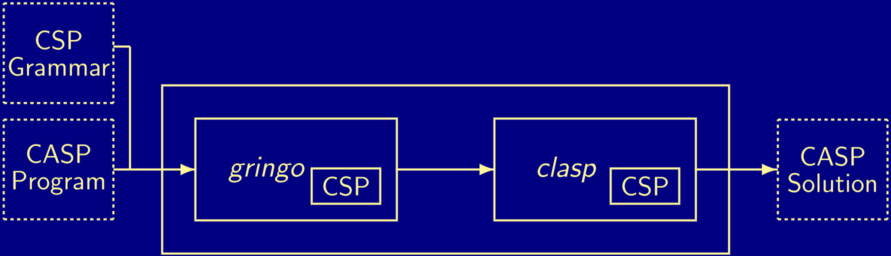
# *clingcon*'s lazy approach




- *clingcon* 1+2
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

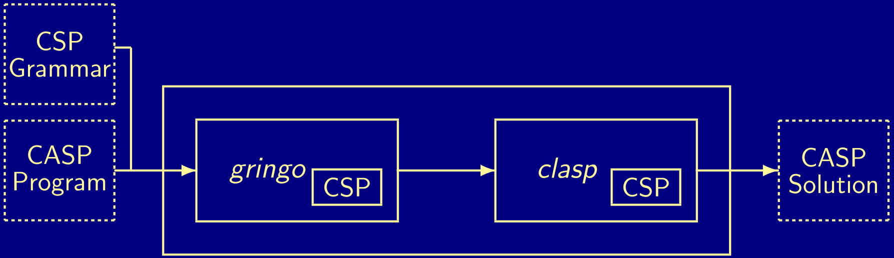
Potassco

# *clingcon*'s lazy approach



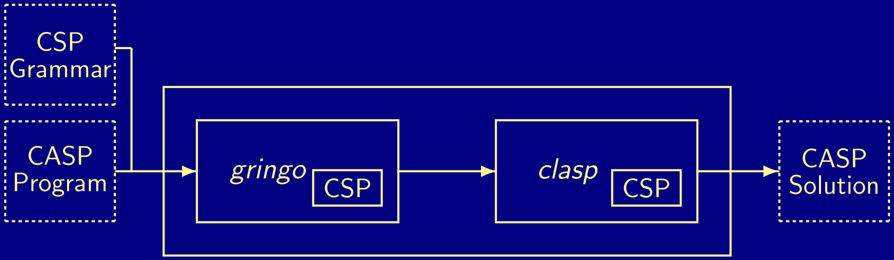

- *clingcon* 1+2
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

# *clingcon*'s lazy approach

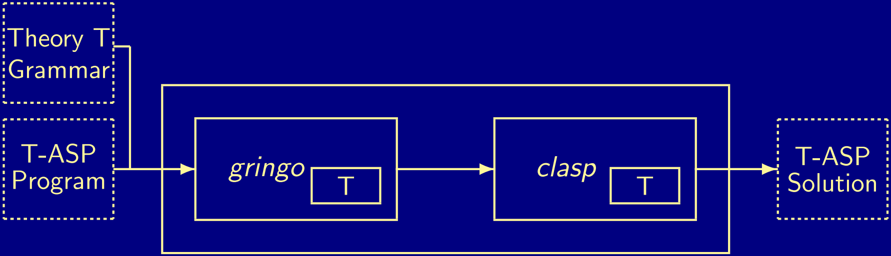



- *clingcon* 1+2
  - language extension
  - propagation via *gecode*
  - conflict minimization
- *clingcon* 3
  - language specification
  - lazy propagation*

# *clingcon*'s approach



# *clingcon* instantiates *clingo*



[1] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[2] C. Baral.
*Knowledge Representation, Reasoning and Declarative Problem Solving*.
Cambridge University Press, 2003.

[3] C. Baral, G. Brewka, and J. Schlipf, editors.
*Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[4] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
*Journal of Logic Programming*, 12:1–80, 1994.

[5] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.

In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[6] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.
In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[7] A. Biere.
PicoSAT essentials.
*Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[8] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
*Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*.

IOS Press, 2009.

[9] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
*Communications of the ACM*, 54(12):92–103, 2011.

[10] G. Brewka, I. Niemelä, and M. Truszczyński.
Answer set optimization.
In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 867–872. Morgan Kaufmann Publishers, 2003.

[11] K. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[12] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
*Handbook of Tableau Methods*.
Kluwer Academic Publishers, 1999.

[13] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[14] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
*Communications of the ACM*, 5:394–397, 1962.

[15] M. Davis and H. Putnam.
A computing procedure for quantification theory.
*Journal of the ACM*, 7:201–215, 1960.

[16] E. Di Rosa, E. Giunchiglia, and M. Maratea.
Solving satisfiability problems with preferences.
*Constraints*, 15(4):485–515, 2010.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.

Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.
In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
*Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.
*Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[23] P. Ferraris.
Answer sets for propositional theories.

In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
*Journal of Logic Programming*, 2(4):295–312, 1985.

[26] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub.
Abstract Gringo.
*Theory and Practice of Logic Programming*, 15(4-5):449–463, 2015.
Available at `http://arxiv.org/abs/1507.06576`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
*Potassco User Guide*.
University of Potsdam, second edition edition, 2015.

[28] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[29] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[30] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.

On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [49], pages 250–264.

[31] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
*Answer Set Solving in Practice*.
Synthesis Lectures on Artificial Intelligence and Machine Learning.
Morgan and Claypool Publishers, 2012.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [3], pages 260–265.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [3], pages 136–148.

[34] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [74], pages 386–392.

[35] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[36] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.
In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[37] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*

*(CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[38] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [49], pages 235–249.

[39] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.
In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[40] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[41] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[42] M. Gelfond and Y. Kahl.
*Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*.
Cambridge University Press, 2014.

[43] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
*Artificial Intelligence*, 138(1-2):3–38, 2002.

[44] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.

In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[45] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[46] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.
*Journal of Automated Reasoning*, 36(4):345–377, 2006.

[47] K. Gödel.
Zum intuitionistischen Aussagenkalkül.
In *Anzeiger der Akademie der Wissenschaften in Wien*, page 65–66. 1932.

[48] A. Heyting.

Die formalen Regeln der intuitionistischen Logik.
In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. 1930.
Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

[49] P. Hill and D. Warren, editors.
*Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[50] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [74], pages 2318–2323.

[51] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
*Theory and Practice of Logic Programming*, 6(1-2):61–106, 2006.

[52] J. Lee.

A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[53] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
*ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[54] V. Lifschitz.
Answer set programming and plan generation.
*Artificial Intelligence*, 138(1-2):39–54, 2002.

[55] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[56] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?

*ACM Transactions on Computational Logic*, 7(2):261–268, 2006.

[57] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
*Artificial Intelligence*, 157(1-2):115–137, 2004.

[58] V. Marek and M. Truszczyński.
*Nonmonotonic logic: context-dependent reasoning*.
Artifical Intelligence. Springer-Verlag, 1993.

[59] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[60] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [8], chapter 4, pages 131–153.

[61] J. Marques-Silva and K. Sakallah.

GRASP: A search algorithm for propositional satisfiability.
*IEEE Transactions on Computers*, 48(5):506–521, 1999.

[62] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.
In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[63] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
*Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.

[64] D. Mitchell.
A SAT solver primer.
*Bulletin of the European Association for Theoretical Computer Science*, 85:112–133, 2005.

[65] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[66] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
*Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[67] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
*Journal of the ACM*, 53(6):937–977, 2006.

[68] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.

In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[69] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

[70] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
*Artificial Intelligence*, 138(1-2):181–234, 2002.

[71] T. Son and E. Pontelli.
Planning with preferences using logic programming.
*Theory and Practice of Logic Programming*, 6(5):559–608, 2006.

[72] T. Syrjänen.
Lparse 1.0 user's manual, 2001.

[73] A. Van Gelder, K. Ross, and J. Schlipf.

The well-founded semantics for general logic programs.
*Journal of the ACM*, 38(3):620–650, 1991.

[74] M. Veloso, editor.
*Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*. AAAI/MIT Press, 2007.

[75] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.
In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.