Answer Set Programming

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

Roadmap

- Introduction
- Modeling
- Language Extensions
- Operators, Algorithms, and Systems
- Applications

Resources

- Course material
  - http://www.cs.uni-potsdam.de/wv/lehre
  - http://moodle.haiti.cs.uni-potsdam.de
  - http://www.cs.uni-potsdam.de/~torsten/asp
- Systems
  - clasp http://potassco.sourceforge.net
  - dlv http://www.dbai.tuwien.ac.at/proj/dlv
  - smodels http://www.tcs.hut.fi/Software/smodels
  - gringo http://potassco.sourceforge.net
    - clingo
    - iclingo
  - lparse http://www.tcs.hut.fi/Software/smodels
  - asparagus http://asparagus.cs.uni-potsdam.de

Literature

Books	[4], [59]
Surveys	[52], [2], [37]
Articles	[39], [40], [6], [67], [60], [51], [38], etc.

Motivation
Overview

1 Objective

2 Answer Set Programming

3 Historic Roots

4 Problem Solving

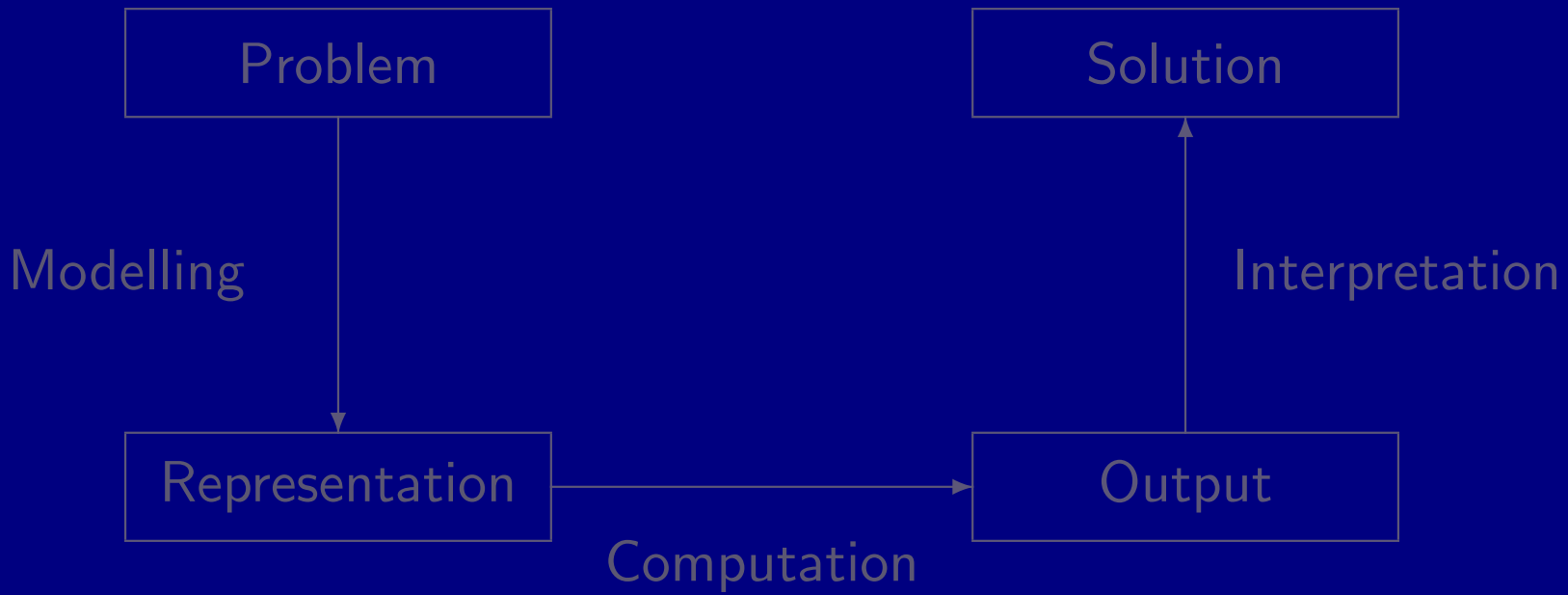
5 Applications

Goal: Declarative problem solving

- *"What is the problem?"*

  instead of

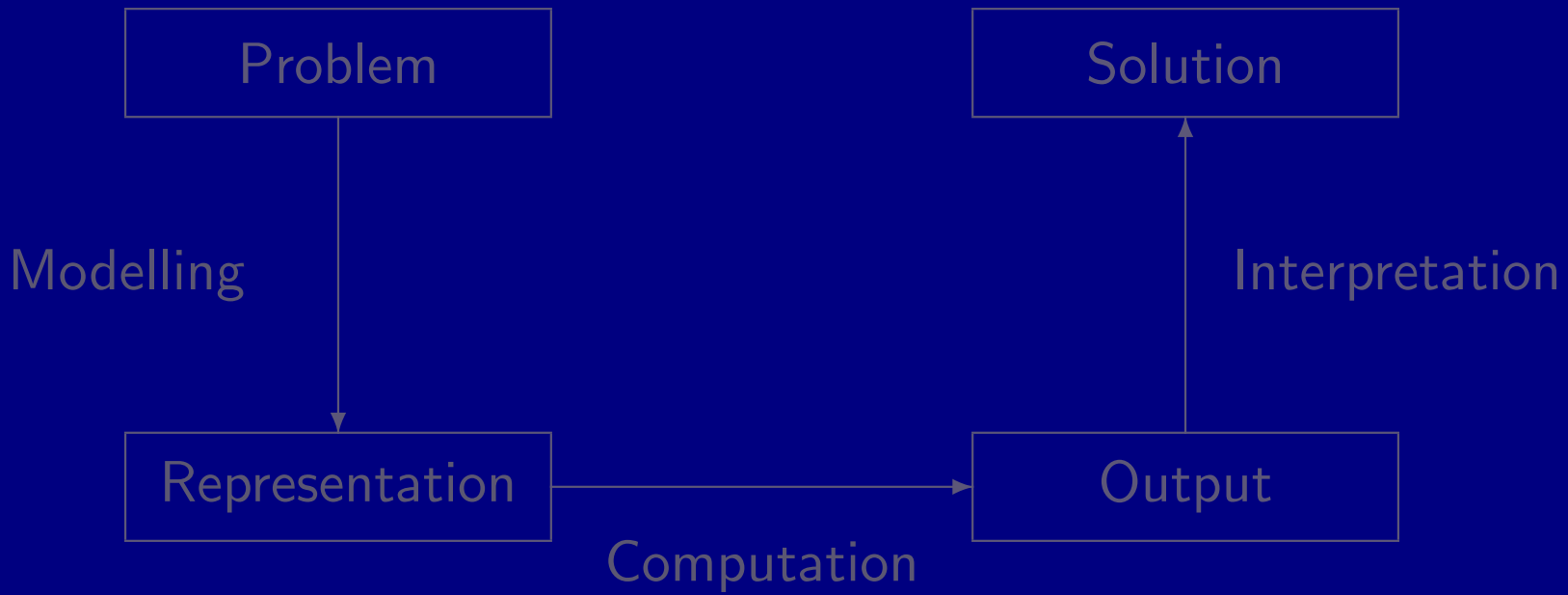
- *"How to solve the problem?"*



Goal: Declarative problem solving

- *"What is the problem?"*

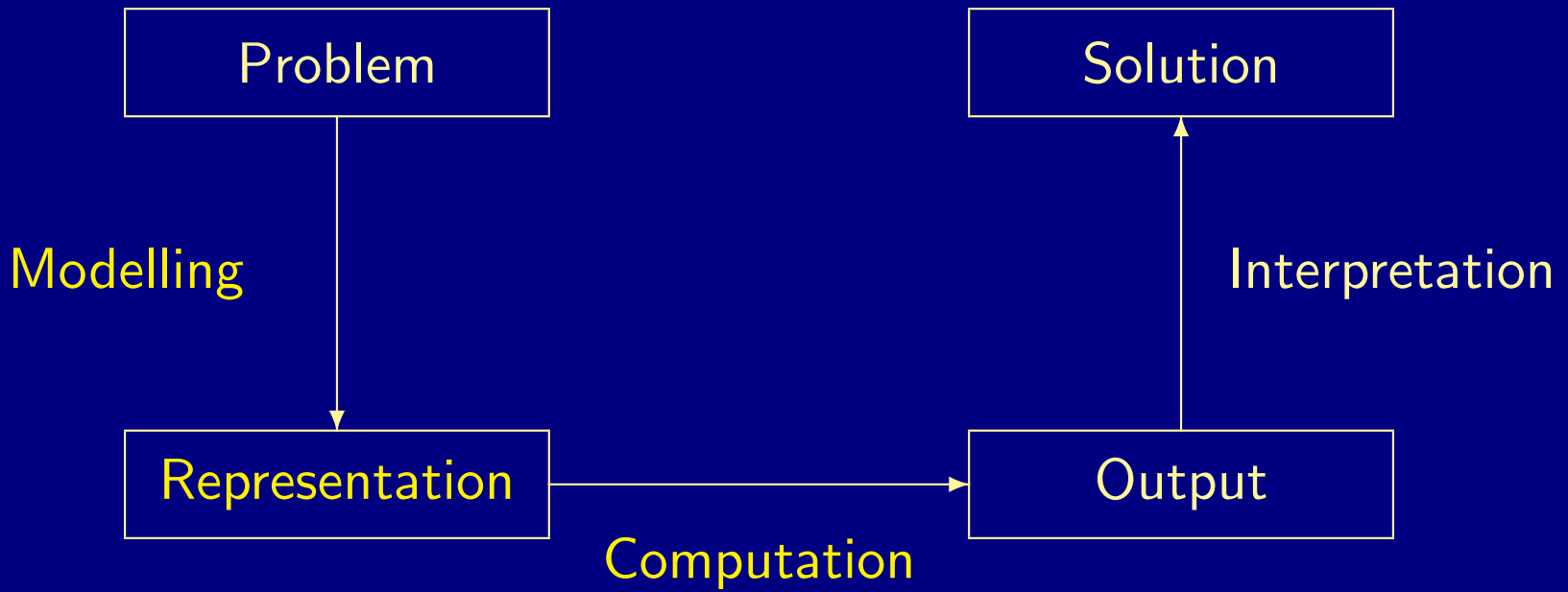
  instead of
- *"How to solve the problem?"*



Goal: Declarative problem solving

- *"What is the problem?"*

  instead of
- *"How to solve the problem?"*



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

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

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})
  (over finite domains; otherwise Turing-complete)
- allows for using powerful off-the-shelf systems
  (nowadays capable of dealing with millions of variables)

Take Home Lesson

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

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})
  (over finite domains; otherwise Turing-complete)
- allows for using powerful off-the-shelf systems
  (nowadays capable of dealing with millions of variables)

Take Home Lesson

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

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})
  (over finite domains; otherwise Turing-complete)
- allows for using powerful off-the-shelf systems
  (nowadays capable of dealing with millions of variables)

Take Home Lesson

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

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})
  (over finite domains; otherwise Turing-complete)
- allows for using powerful off-the-shelf systems
  (nowadays capable of dealing with millions of variables)

Take Home Lesson

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

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})
  (over finite domains; otherwise Turing-complete)
- allows for using powerful off-the-shelf systems
  (nowadays capable of dealing with millions of variables)

Take Home Lesson

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

Root: Logic Programming with negation

- Algorithm = Logic + Control [48]
- 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

Prolog: Programming in logic

[58]

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

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

Prolog: Programming in logic

[58]

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

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

Prolog: Programming in logic

[58]

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

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

Prolog: Programming in logic

[58]

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

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

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 [11] by

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

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 [11] by

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

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 [11] by

$$\forall x(x = a \leftrightarrow info(x)) \wedge \forall x(\neg info(x) \leftrightarrow ask(x))$$
$$\iff \quad info(a) \wedge \forall x(x \neq a \leftrightarrow ask(x))$$

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 [11] by

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

Prolog (ctd)

Clark's completion is sometimes too syntactical.
Consider

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

Or even more complex yet analogous situations.

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 (e.g. ASP)

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

Automated planning, Kautz and Selman [46]

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

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 (e.g. ASP)

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

Automated planning, Kautz and Selman [46]

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

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 (e.g. ASP)

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

Automated planning, Kautz and Selman [46]

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

Model-based Problem Solving

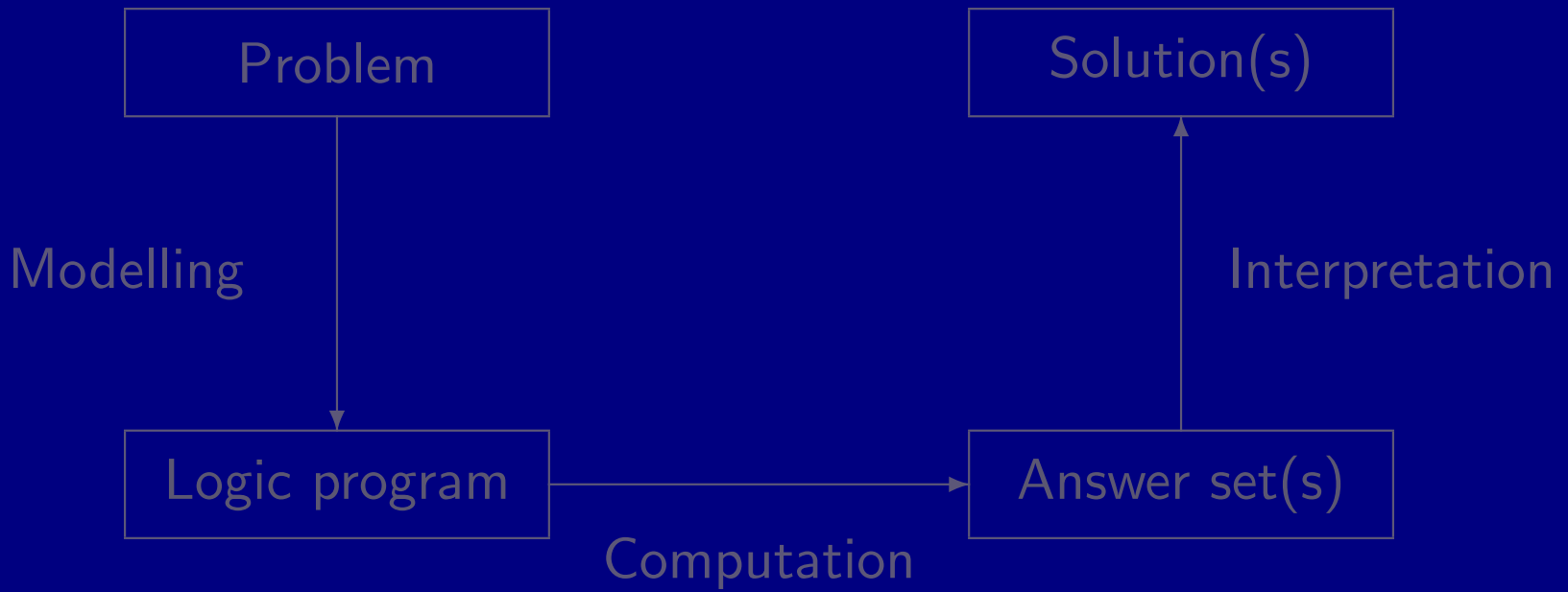
Specification	Associated Structures
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models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

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
...	

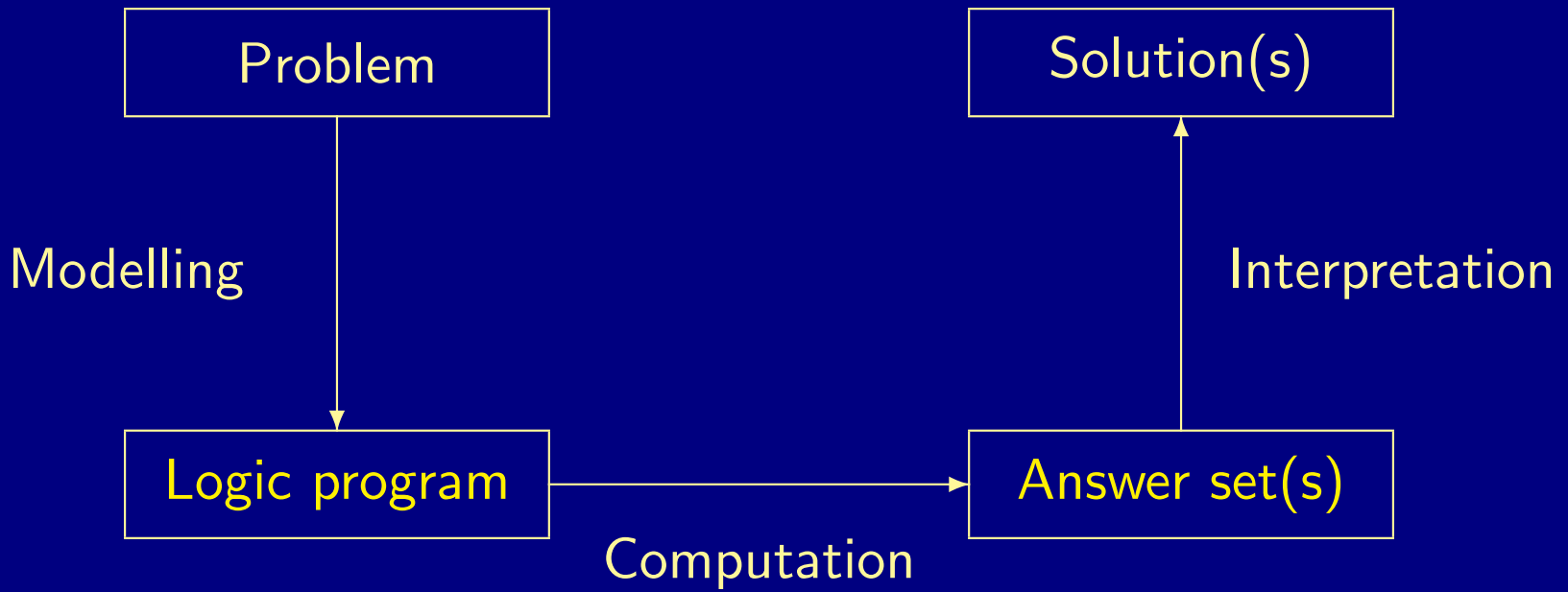
ASP as High-level Language

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



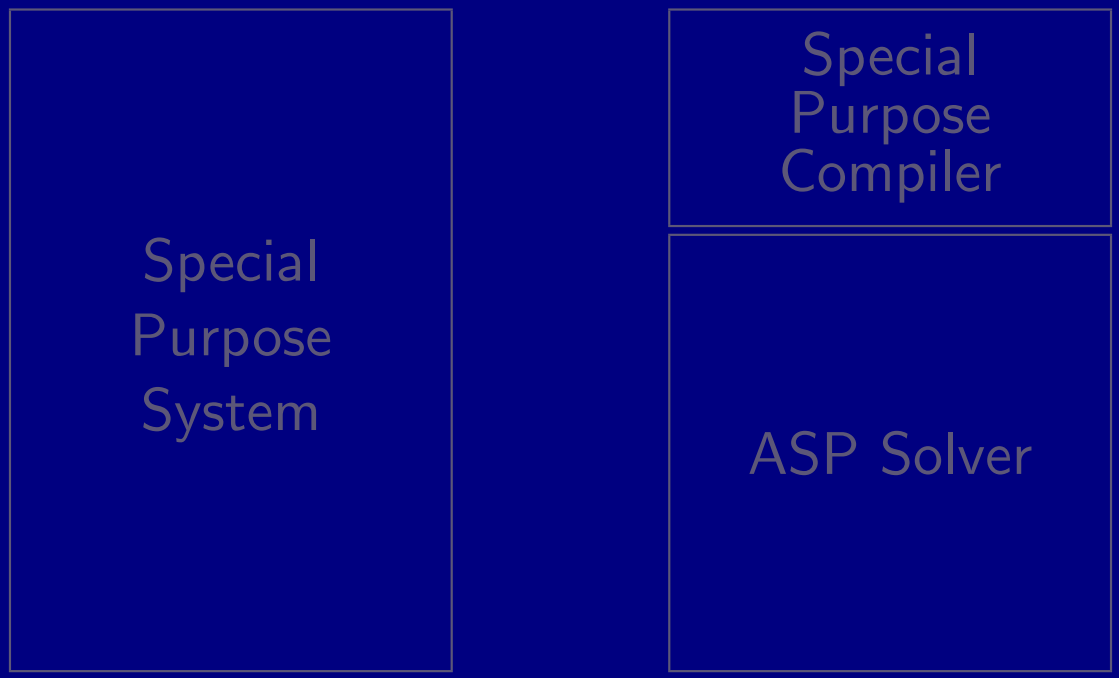
ASP as High-level Language

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



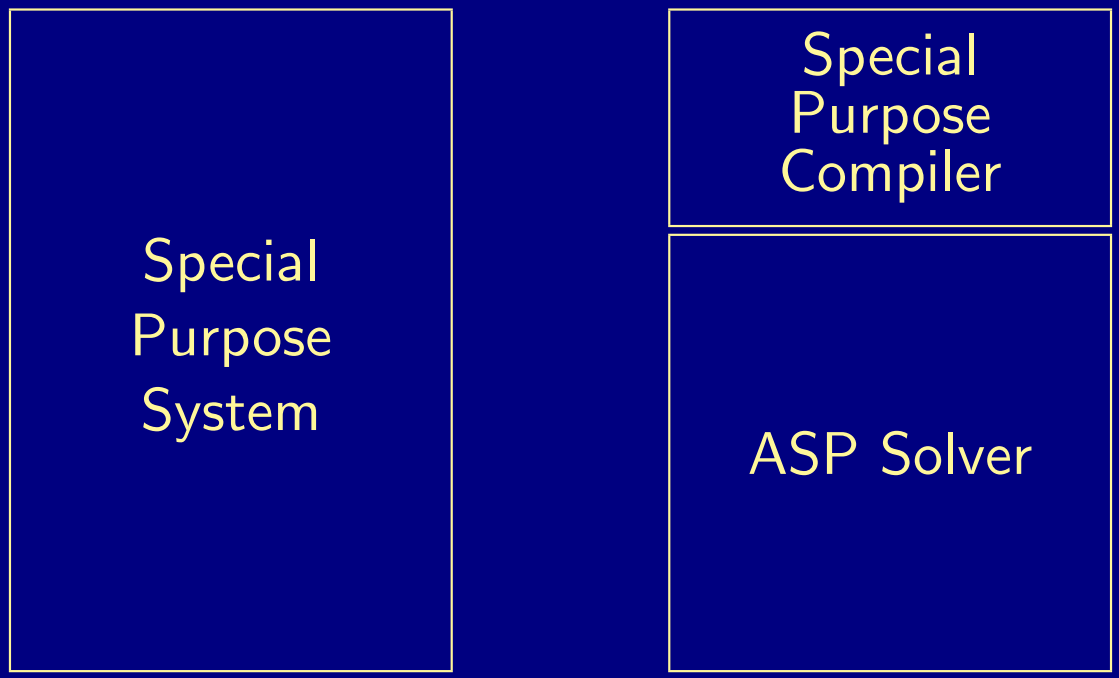
ASP as Low-level Language

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



ASP as Low-level Language

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



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 [69]

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 [69]

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 [69]

ASP as High-level Language

A very simple Configuration Example

$$
\begin{aligned}
1\ \{\ \mathit{disk}(\mathit{ide}), \mathit{disk}(\mathit{scsi}), \mathit{disk}(\mathit{fw})\ \}\ 2 &\leftarrow \mathit{computer}(X)\\
\mathit{keyboard}(\mathit{us})\ ;\ \mathit{keyboard}(\mathit{german}) &\leftarrow \mathit{computer}(X)\\
\mathit{controller}(\mathit{scsi}) &\leftarrow \mathit{disk}(\mathit{scsi})\\
\mathit{monitor}(\mathit{15in}) &\leftarrow \mathit{not\ other\text{-}monitor}(\mathit{15in})\\
\mathit{other\text{-}monitor}(Y) &\leftarrow \mathit{monitor}(X), X \neq Y\\
&\leftarrow \mathit{monitor}(\mathit{21in}), \mathit{graphics}(\mathit{evil})\\
\\
\mathit{computer}(\mathit{desktop}) &\leftarrow\\
&\leftarrow \mathit{controller}(\mathit{scsi})
\end{aligned}
$$

E.g. [75] describes a Debian Linux configurator

ASP as High-level Language

A very simple Configuration Example

$$
\begin{aligned}
1\ \{\ \mathrm{disk(ide)}, \mathrm{disk(scsi)}, \mathrm{disk(fw)}\ \}\ 2 &\leftarrow \mathrm{computer(X)} \\
\mathrm{keyboard(us)}\ ;\ \mathrm{keyboard(german)} &\leftarrow \mathrm{computer(X)} \\
\mathrm{controller(scsi)} &\leftarrow \mathrm{disk(scsi)} \\
\mathrm{monitor(15in)} &\leftarrow \mathrm{not\ other\text{-}monitor(15in)} \\
\mathrm{other\text{-}monitor(Y)} &\leftarrow \mathrm{monitor(X)}, \mathrm{X} \neq \mathrm{Y} \\
&\leftarrow \mathrm{monitor(21in)}, \mathrm{graphics(evil)} \\
\\
\mathrm{computer(desktop)} &\leftarrow \\
&\leftarrow \mathrm{controller(scsi)}
\end{aligned}
$$

E.g. [75] describes a Debian Linux configurator

ASP as Low-level Language

Action description language $\mathcal{A}$ [40]

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

is translated into

ASP Program

```
% 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. [43] describes pathways in metabolic networks.

ASP as Low-level Language

Action description language $\mathcal{A}$ [40]

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

is translated into

ASP Program

```
% 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. [43] describes pathways in metabolic networks.

ASP as Low-level Language

Action description language $\mathcal{A}$ [40]

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

is translated into

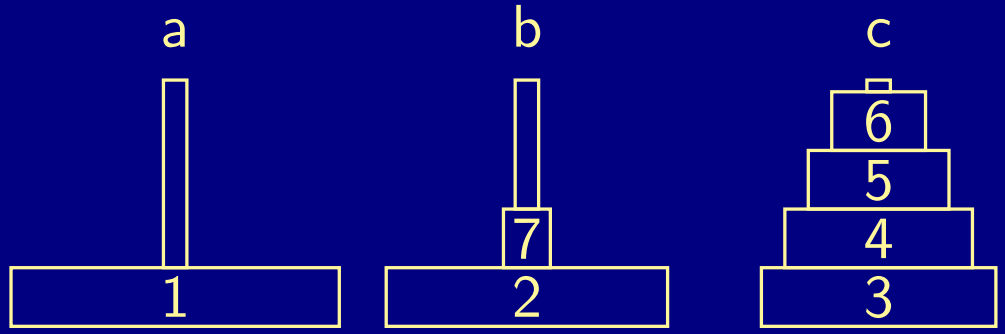
ASP Program

```
% 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.
```

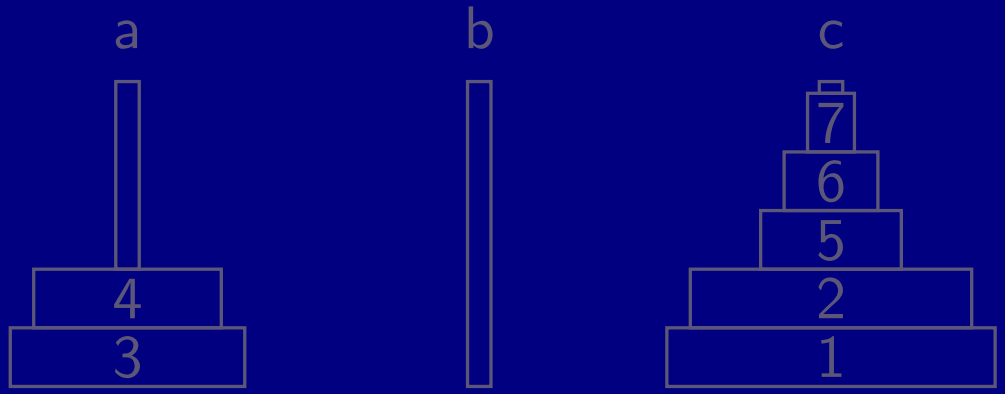
E.g. [43] describes pathways in metabolic networks.

An instance of Towers of Hanoi

Init:



Goal:

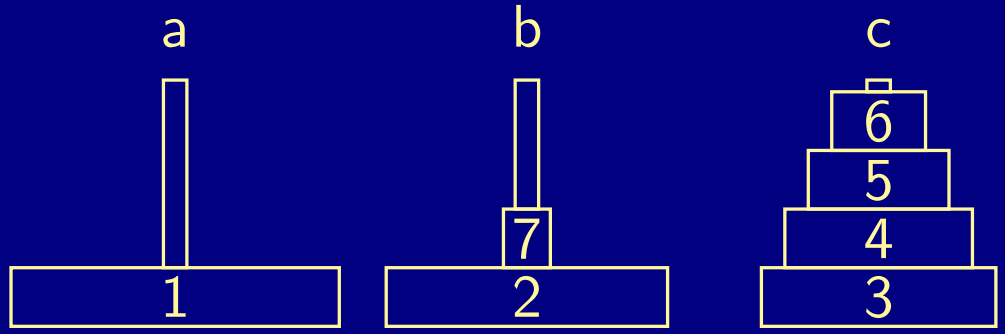



```
peg(a;b;c).
init_on(1,a).
init_on(2;7,b).
init_on(3;4;5;6,c).
```

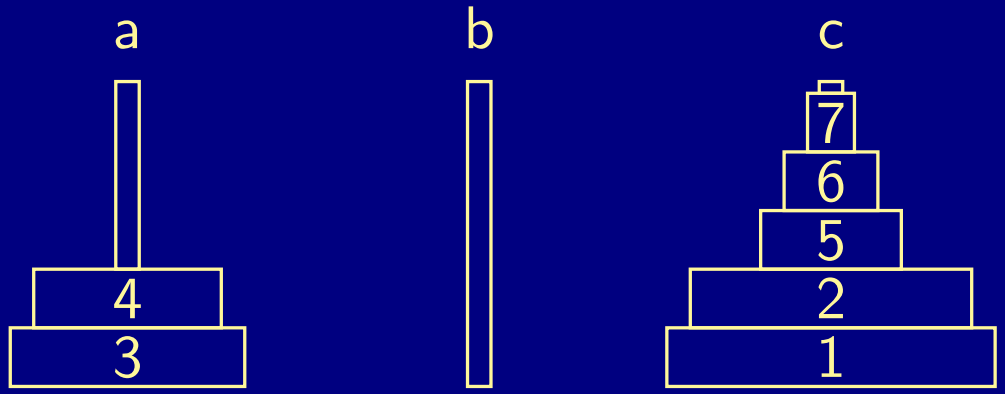
```
disk(1..7).
goal_on(3;4,a).
goal_on(1;2;5;6;7,c).
moves(70).
```

An instance of Towers of Hanoi

Init:



Goal:




```
peg(a;b;c).
init_on(1,a).
init_on(2;7,b).
init_on(3;4;5;6,c).
```

```
disk(1..7).
goal_on(3;4,a).
goal_on(1;2;5;6;7,c).
moves(70).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

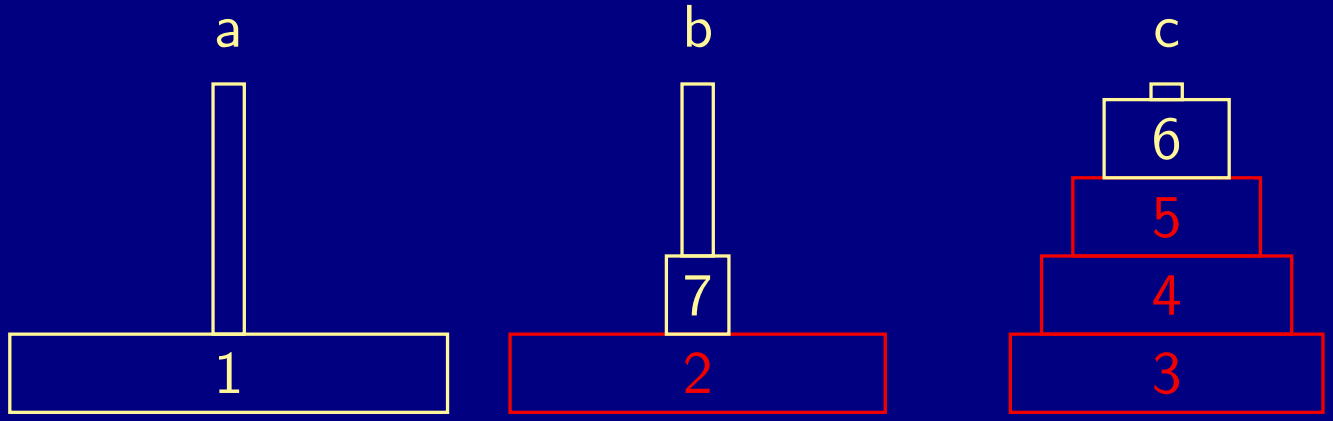
1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

An encoding of Towers of Hanoi




```
blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

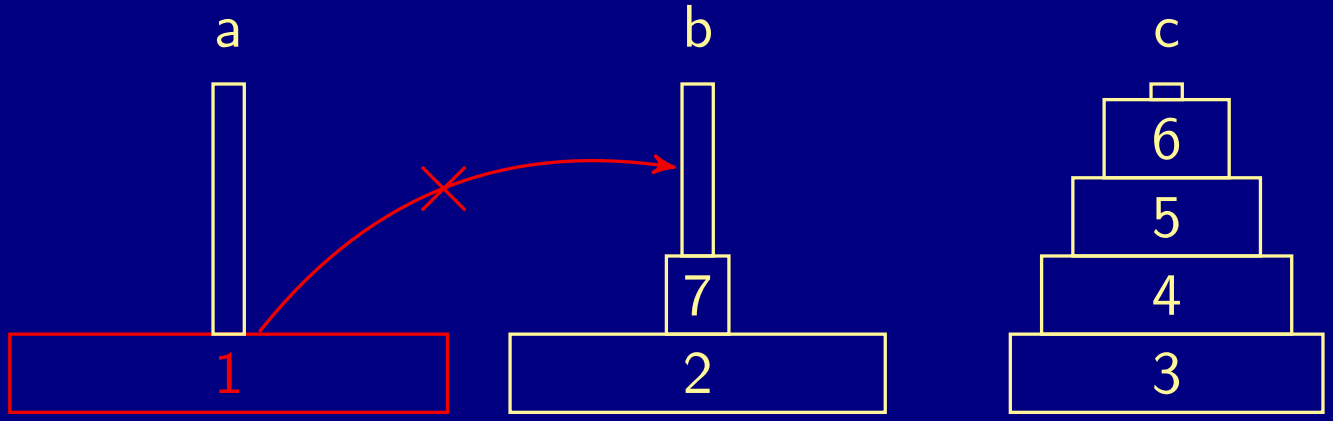
1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

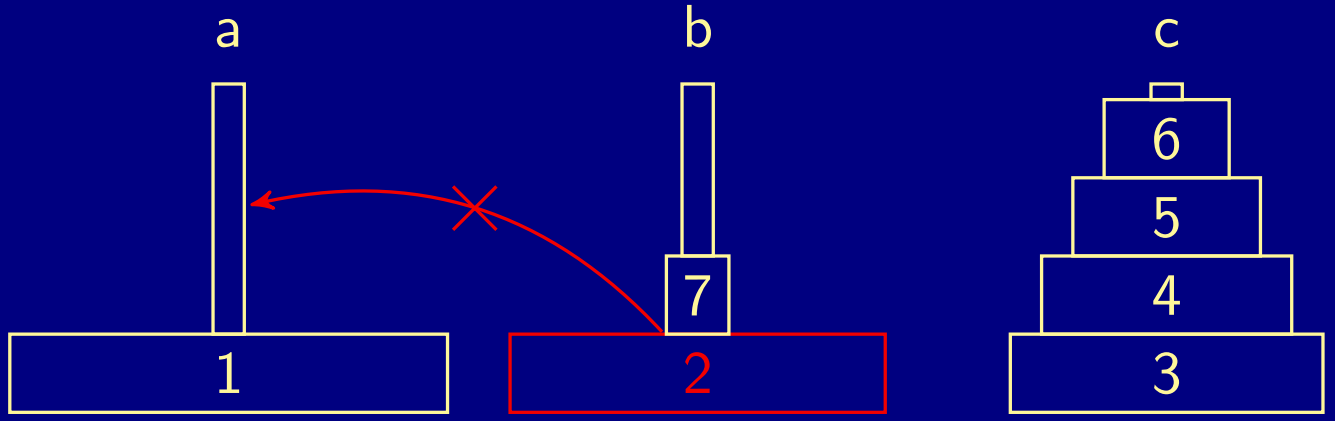
:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

An encoding of Towers of Hanoi




```
:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
```

An encoding of Towers of Hanoi




```
:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

An encoding of Towers of Hanoi

```
on(D,P,0)    :- init_on(D,P).

1 { move(D,P,T) : disk(D) : peg(P) } 1 :- moves(M), T = 1..M.
move(D,T)    :- move(D,_,T).

on(D,P,T)    :- move(D,P,T).
on(D,P,T+1)  :- on(D,P,T), not move(D,T+1), not moves(T).

blocked(D-1,P,T+1) :- on(D,P,T), not moves(T).
blocked(D-1,P,T)   :- blocked(D,P,T), disk(D).

:- move(D,P,T), blocked(D-1,P,T).
:- move(D,T), on(D,P,T-1), blocked(D,P,T).
:- not 1 { on(D,P,T) } 1, disk(D), moves(M), T = 1..M.
:- goal_on(D,P), not on(D,P,M), moves(M).
```

Let it run!

```
torsten@raz > gringo toh_instance.lp toh_encoding.lp | clasp --stats
clasp version 1.3.5
Reading from stdin
Solving...
Answer: 1
peg(a) peg(c) peg(b) init_on(1,a) init_on(2,b) ...
move(6,a,1)  move(7,a,2)  move(5,b,3)  move(7,c,4)
move(6,b,5)  move(7,b,6)  move(4,a,7)  move(7,a,8) ...
move(2,c,63) move(7,c,64) move(6,b,65) move(7,b,66)
move(5,c,67) move(7,a,68) move(6,c,69) move(7,c,70)
move(7,70) move(6,69) move(7,68) move(5,67) move(7,66) ...
SATISFIABLE

Models       : 1+
Time         : 3.280s (Solving: 3.23s 1st Model: 3.23s Unsat: 0.00s)
Choices      : 130907
Conflicts    : 35738
Restarts     : 12
```

Introduction
Overview

6 Syntax

7 Semantics

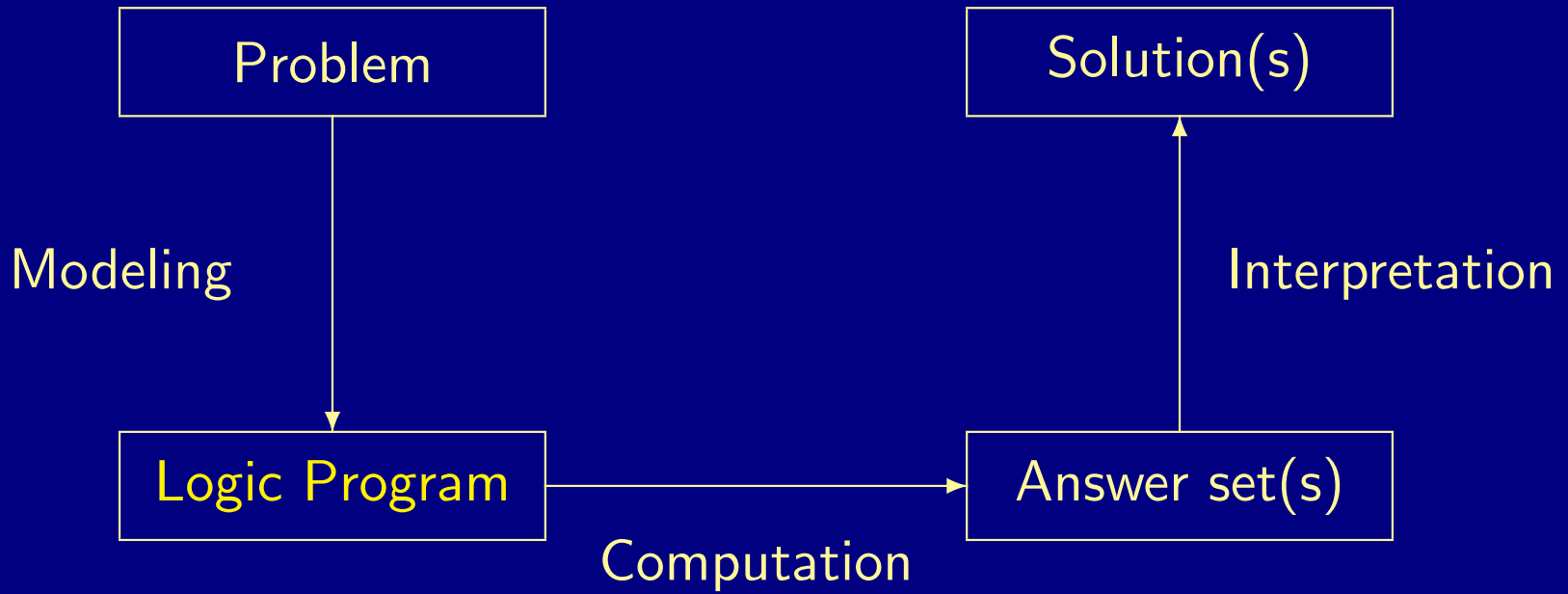
8 Examples

9 Variables and Grounding

10 Integrity Constraints

11 Computation

Problem solving in ASP: Syntax



Normal logic programs

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

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  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

$$\begin{aligned} \mathit{head}(r) &= A_0 \\ \mathit{body}(r) &= \{A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\} \\ \mathit{body}^+(r) &= \{A_1, \ldots, A_m\} \\ \mathit{body}^-(r) &= \{A_{m+1}, \ldots, A_n\} \end{aligned}$$

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

Normal logic programs

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

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  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

$$\begin{aligned} \mathit{head}(r) &= A_0 \\ \mathit{body}(r) &= \{A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\} \\ \mathit{body}^+(r) &= \{A_1, \ldots, A_m\} \\ \mathit{body}^-(r) &= \{A_{m+1}, \ldots, A_n\} \end{aligned}$$

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

Normal logic programs

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

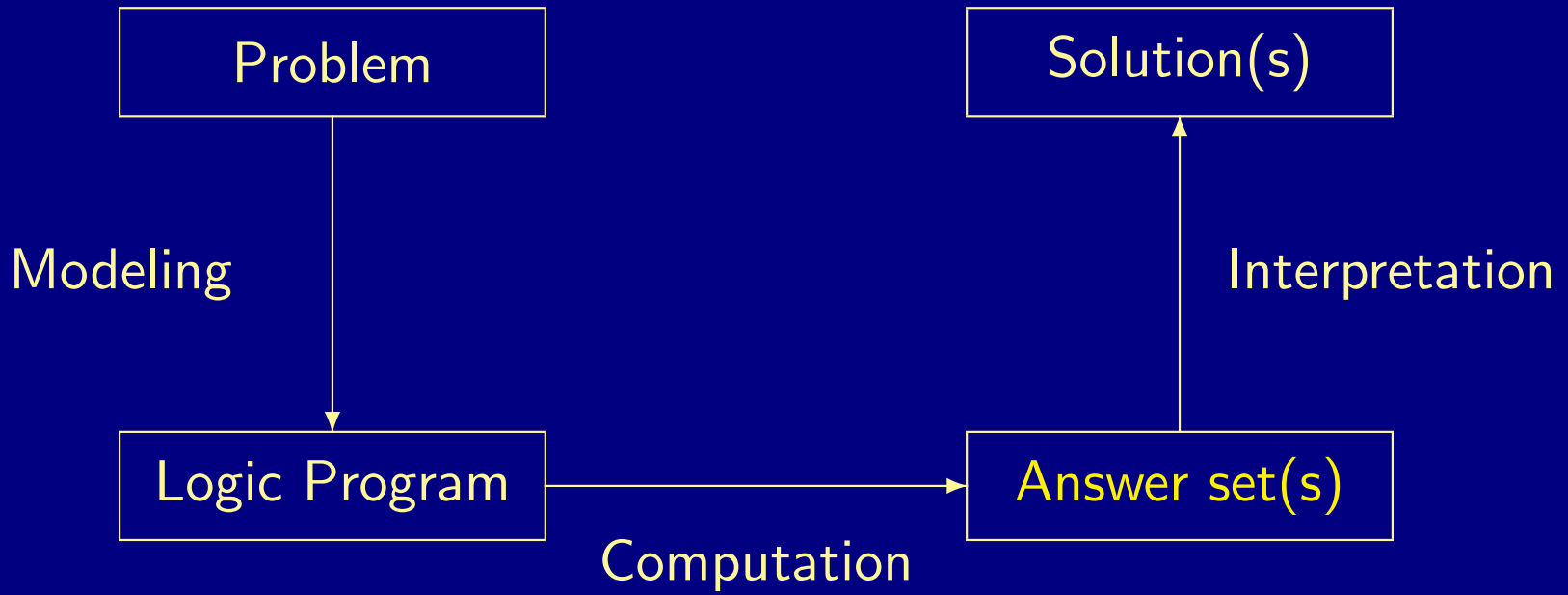
$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  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

$$\begin{aligned} \mathit{head}(r) &= A_0 \\ \mathit{body}(r) &= \{A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\} \\ \mathit{body}^+(r) &= \{A_1, \ldots, A_m\} \\ \mathit{body}^-(r) &= \{A_{m+1}, \ldots, A_n\} \end{aligned}$$

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

Problem solving in ASP: Semantics



Answer set: Formal Definition

Positive programs

- A set of atoms X is closed under a positive program Π iff for any $r \in \Pi$, $\mathit{head}(r) \in X$ whenever $\mathit{body}^+(r) \subseteq X$.
  - X corresponds to a model of Π (seen as a formula).
- The smallest set of atoms which is closed under a positive program Π is denoted by $Cn(\Pi)$.
  - $Cn(\Pi)$ corresponds to the $\subseteq$-smallest model of Π (ditto).
- The set $Cn(\Pi)$ of atoms is the answer set of a *positive* program Π.

Answer set: Formal Definition

Positive programs

- A set of atoms X is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \in X$ whenever $body^+(r) \subseteq X$.
  - ➥ X corresponds to a model of Π (seen as a formula).
- The smallest set of atoms which is closed under a positive program Π is denoted by $Cn(\Pi)$.
  - ➥ $Cn(\Pi)$ corresponds to the $\subseteq$-smallest model of Π (ditto).
- The set $Cn(\Pi)$ of atoms is the answer set of a *positive* program Π.

Answer set: Formal Definition

Positive programs

- A set of atoms X is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \in X$ whenever $body^+(r) \subseteq X$.
  - ➥ X corresponds to a model of Π (seen as a formula).
- The smallest set of atoms which is closed under a positive program Π is denoted by $Cn(\Pi)$.
  - ➥ $Cn(\Pi)$ corresponds to the $\subseteq$-smallest model of Π (ditto).
- The set $Cn(\Pi)$ of atoms is the answer set of a *positive* program Π.

Answer set: Formal Definition

Positive programs

- A set of atoms X is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \in X$ whenever $body^+(r) \subseteq X$.
  - ➥ X corresponds to a model of Π (seen as a formula).
- The smallest set of atoms which is closed under a positive program Π is denoted by $Cn(\Pi)$.
  - ➥ $Cn(\Pi)$ corresponds to the $\subseteq$-smallest model of Π (ditto).
- The set $Cn(\Pi)$ of atoms is the answer set of a *positive* program Π.

Some "logical" remarks

- Positive rules are also referred to as definite clauses.
  - Definite clauses are disjunctions with exactly one positive atom:

$$A_0 \vee \neg A_1 \vee \cdots \vee \neg A_m$$

  - 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, $Cn(\Pi)$ corresponds to the smallest model of the set of definite clauses corresponding to Π.

Some "logical" remarks

- Positive rules are also referred to as definite clauses.
  - Definite clauses are disjunctions with exactly one positive atom:

$$A_0 \vee \neg A_1 \vee \cdots \vee \neg A_m$$

  - 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, $Cn(\Pi)$ corresponds to the smallest model of the set of definite clauses corresponding to Π.

Some "logical" remarks

- Positive rules are also referred to as definite clauses.
  - Definite clauses are disjunctions with exactly one positive atom:

$$A_0 \vee \neg A_1 \vee \cdots \vee \neg A_m$$

  - 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, $Cn(\Pi)$ corresponds to the smallest model of the set of definite clauses corresponding to Π.

(Rough) notational convention

We sometimes use the following notation interchangeably in order to stress the respective view:

	if	and	or	negation as failure	classical negation
source code	`:-`	`,`	`|`	`not`	`-`
logic program	$\leftarrow$	,	;	$not/\sim$	$\neg$
formula	$\rightarrow$	$\wedge$	$\vee$	$\sim/(\neg)$	$\neg$

Answer set: Basic idea

Consider the logical formula Φ and its three (classical) models:

Φ $\quad q \wedge (q \wedge \neg r \rightarrow p)$

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

This formula has one answer set:

$\{p, q\}$

Π

$$q \leftarrow$$
$$p \leftarrow q, \mathit{not}\ r$$

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

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Answer set: Basic idea

Consider the logical formula Φ and its three (classical) models:

Φ $\boxed{q \wedge (q \wedge \neg r \rightarrow p)}$

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

This formula has one answer set:

$\{p, q\}$

Π

$$\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q,\ not\ r \end{array}$$

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

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Answer set: Basic idea

Consider the logical formula Φ and its three (classical) models:

Φ $\quad q \wedge (q \wedge \neg r \rightarrow p)$

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

This formula has one answer set:

$\{p, q\}$

Π

$$q \leftarrow$$
$$p \leftarrow q, \; not \; r$$

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

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Answer set: Basic idea

Consider the logical formula Φ and its three (classical) models:

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

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

This formula has one answer set:

$$\Pi \quad \begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ not\ r \end{array}$$

$$\{p, q\}$$

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

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Answer set: Basic idea

Consider the logical formula Φ and its three (classical) models:

Φ $\quad q \wedge (q \wedge \neg r \rightarrow p)$

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

This formula has one answer set:

$\{p, q\}$

Π

$$q \leftarrow$$
$$p \leftarrow q, \; not \; r$$

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

(rooted in intuitionistic logics HT (Heyting, 1930) and G3 (Gödel, 1932))

Answer set: Formal Definition

Normal programs [39]

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

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

- A set X of atoms is an answer set of a program Π if $Cn(\Pi^X) = X$.
  Recall: $Cn(\Pi^X)$ 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*"*

Answer set: Formal Definition

Normal programs [39]

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

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

- A set X of atoms is an answer set of a program Π if $Cn(\Pi^X) = X$.
  Recall: $Cn(\Pi^X)$ 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*"*

Answer set: Formal Definition

Normal programs [39]

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

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

- A set X of atoms is an answer set of a program Π if $Cn(\Pi^X) = X$.
  Recall: $Cn(\Pi^X)$ 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"*

Answer set: Formal Definition

Normal programs [39]

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

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

- A set X of atoms is an answer set of a program Π if $Cn(\Pi^X) = X$.
  Recall: $Cn(\Pi^X)$ 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"*

Answer set: Formal Definition

Normal programs [39]

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

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

- A set X of atoms is an answer set of a program Π if $Cn(\Pi^X) = X$.
  Recall: $Cn(\Pi^X)$ 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*"*

A closer look at Π^X

In other words, given a set X of atoms 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.

A first example

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

X	Π^X	$Cn(\Pi^X)$
$\emptyset$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p\}$	$p \leftarrow p$	$\emptyset$
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p, q\}$	$p \leftarrow p$	$\emptyset$

A first example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✗
$\{p\}$	$p \leftarrow p$	$\emptyset$	✗
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✓
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✗

A first example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

A first example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

A first example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

A first example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

A second example

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

X	Π^X	$Cn(\Pi^X)$
$\emptyset$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$
$\{p\}$	$p \leftarrow$	$\{p\}$
$\{q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		$\emptyset$

A second example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✗
$\{p\}$	$p \leftarrow$	$\{p\}$	✓
$\{q\}$	$q \leftarrow$	$\{q\}$	✓
$\{p, q\}$		$\emptyset$	✗

A second example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\}$	$p \leftarrow$	$\{p\}$	✔
$\{q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

A second example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\}$	$p \leftarrow$	$\{p\}$	✔
$\{q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

A second example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\}$	$p \leftarrow$	$\{p\}$	✔
$\{q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

A second example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\}$	$p \leftarrow$	$\{p\}$	✔
$\{q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

A third example

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

X	Π^X	$Cn(\Pi^X)$
$\emptyset$	$p \leftarrow$	$\{p\}$
$\{p\}$		$\emptyset$

A third example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$	$\{p\}$	✗
$\{p\}$		$\emptyset$	✗

A third example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	✘

A third example

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

X	Π^X	$Cn(\Pi^X)$	
$\emptyset$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	✘

Answer set: Some properties

- A program may have zero, one, or multiple answer sets!
- If X is an answer set of a logic program Π, then X is a model of Π (seen as a formula).
- If X and Y are answer sets of a *normal* program Π, then $X \not\subset Y$.

Answer set: Some properties

- A program may have zero, one, or multiple answer sets!
- If X is an answer set of a logic program Π, then X is a model of Π (seen as a formula).
- If X and Y are answer sets of a *normal* program Π, then $X \not\subset Y$.

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

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

- 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.

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

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

- 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.

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

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

- 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.

The second example revisited

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

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

The second example revisited

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

X	Π_X	*"logically"*	$\min_{\subseteq}(\Pi_X)$	
$\emptyset$	$p \leftarrow not\ q$ $q \leftarrow not\ p$	$p \vee q$	$\{p\}, \{q\}$	✘
$\{p\}$	$p \leftarrow not\ q$	$p \vee q$	$\{p\}, \{q\}$	✔
$\{q\}$	$q \leftarrow not\ p$	$p \vee q$	$\{p\}, \{q\}$	✔
$\{p, q\}$		$\top$	$\emptyset$	✘

A closer look at Cn

Inductive characterization

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

- The immediate consequence operator T_Π is defined as follows:

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

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

Theorem

For any positive program Π, we have

- $Cn(\Pi) = \bigcup_{i \geq 0} T_\Pi^i \emptyset$,
- *$X \subseteq Y$ implies $T_\Pi X \subseteq T_\Pi Y$,*
- *$Cn(\Pi)$ is the smallest fixpoint of T_Π.*

A closer look at Cn

Inductive characterization

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

- The immediate consequence operator T_Π is defined as follows:

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

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

Theorem

For any positive program Π, we have

- $Cn(\Pi) = \bigcup_{i \geq 0} T_\Pi^i \emptyset$,
- *$X \subseteq Y$ implies $T_\Pi X \subseteq T_\Pi Y$,*
- *$Cn(\Pi)$ is the smallest fixpoint of T_Π.*

A closer look at Cn

Inductive characterization

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

- The immediate consequence operator T_Π is defined as follows:

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

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

Theorem

For any positive program Π, we have

- $Cn(\Pi) = \bigcup_{i \geq 0} T_\Pi^i \emptyset$,
- $X \subseteq Y$ *implies* $T_\Pi X \subseteq T_\Pi Y$,
- $Cn(\Pi)$ *is the smallest fixpoint of* T_Π.

Let's iterate T_Π

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

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

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 $X \subseteq \{p,q,r,t,s\}$.

Let's iterate T_Π

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

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

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 $X \subseteq \{p, q, r, t, s\}$.

Let's iterate T_Π

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

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

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 $X \subseteq \{p,q,r,t,s\}$.

Programs with Variables

Let Π be a logic program.

- Herbranduniverse U^Π: Set of constants in Π
- Herbrandbase B^Π: Set of (variable-free) atoms constructible from U^Π
  ☞ We usually denote this as $\mathcal{A}$, and call it alphabet.
- Ground Instances of $r \in \Pi$: Set of variable-free rules obtained by replacing all variables in r by elements from U^Π:

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

  where $var(r)$ stands for the set of all variables occurring in r;
  θ is a (ground) substitution.
- Ground Instantiation of Π:

$$ground(\Pi) = \bigcup_{r \in \Pi} ground(r)$$

Programs with Variables

Let Π be a logic program.

- Herbranduniverse U^Π: Set of constants in Π
- Herbrandbase B^Π: Set of (variable-free) atoms constructible from U^Π
  ☞ We usually denote this as $\mathcal{A}$, and call it alphabet.
- Ground Instances of $r \in \Pi$: Set of variable-free rules obtained by replacing all variables in r by elements from U^Π:

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

  where $var(r)$ stands for the set of all variables occurring in r;
  θ is a (ground) substitution.
- Ground Instantiation of Π:

$$ground(\Pi) = \bigcup_{r \in \Pi} ground(r)$$

Programs with Variables

Let Π be a logic program.

- Herbranduniverse U^Π: Set of constants in Π
- Herbrandbase B^Π: Set of (variable-free) atoms constructible from U^Π
  ☞ We usually denote this as $\mathcal{A}$, and call it alphabet.
- Ground Instances of $r \in \Pi$: Set of variable-free rules obtained by replacing all variables in r by elements from U^Π:

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

  where $var(r)$ stands for the set of all variables occurring in r;
  θ is a (ground) substitution.
- Ground Instantiation of Π:

$$ground(\Pi) = \bigcup_{r \in \Pi} ground(r)$$

Programs with Variables

Let Π be a logic program.

- Herbranduniverse U^Π: Set of constants in Π
- Herbrandbase B^Π: Set of (variable-free) atoms constructible from U^Π
  ☞ We usually denote this as $\mathcal{A}$, and call it alphabet.
- Ground Instances of $r \in \Pi$: Set of variable-free rules obtained by replacing all variables in r by elements from U^Π:

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

  where $var(r)$ stands for the set of all variables occurring in r;
  θ is a (ground) substitution.
- Ground Instantiation of Π:

$$ground(\Pi) = \bigcup_{r \in \Pi} ground(r)$$

An example

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

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

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

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

☞ Intelligent Grounding aims at reducing the ground instantiation.

An example

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

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

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

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

☞ Intelligent Grounding aims at reducing the ground instantiation.

An example

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

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

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

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

☞ Intelligent Grounding aims at reducing the ground instantiation.

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 $Cn(ground(\Pi)^X) = X$.

Programs with Integrity Constraints

- Purpose: Integrity constraints eliminate unwanted solution candidates
- Syntax: An integrity constraints is of the form

$$\leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  where $n \geq m \geq 1$, and each A_i $(1 \leq i \leq n)$ is a atom.
- Example: $\leftarrow \mathrm{monitor}(21\mathrm{in}), \mathrm{graphics}(\mathrm{evil})$
- Implementation: For a new symbol x, map

$$\begin{array}{rrcl} & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \\ \mapsto \quad x & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n, \mathit{not}\ x \end{array}$$

- Another example: $\Pi = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p\}$
  versus $\Pi' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow\ p\}$
  versus $\Pi'' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow \mathit{not}\ p\}$

Programs with Integrity Constraints

- Purpose: Integrity constraints eliminate unwanted solution candidates
- Syntax: An integrity constraints is of the form

$$\leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  where $n \geq m \geq 1$, and each A_i $(1 \leq i \leq n)$ is a atom.
- Example: $\leftarrow$ monitor(21in), graphics(evil)
- Implementation: For a new symbol x, map

$$\begin{array}{rcl} & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \\ \mapsto \quad x & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n, \mathit{not}\ x \end{array}$$

- Another example: $\Pi = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p\}$
  versus $\Pi' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow\ p\}$
  versus $\Pi'' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow \mathit{not}\ p\}$

Programs with Integrity Constraints

- Purpose: Integrity constraints eliminate unwanted solution candidates
- Syntax: An integrity constraints is of the form

$$\leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  where $n \geq m \geq 1$, and each A_i $(1 \leq i \leq n)$ is a atom.
- Example: $\leftarrow \mathrm{monitor}(21\mathrm{in}), \mathrm{graphics}(\mathrm{evil})$
- Implementation: For a new symbol x, map

$$\begin{array}{rrcl} & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \\ \mapsto \quad x & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n, \mathit{not}\ x \end{array}$$

- Another example: $\Pi = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p\}$
  versus $\Pi' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow\ p\}$
  versus $\Pi'' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow \mathit{not}\ p\}$

Programs with Integrity Constraints

- Purpose: Integrity constraints eliminate unwanted solution candidates
- Syntax: An integrity constraints is of the form

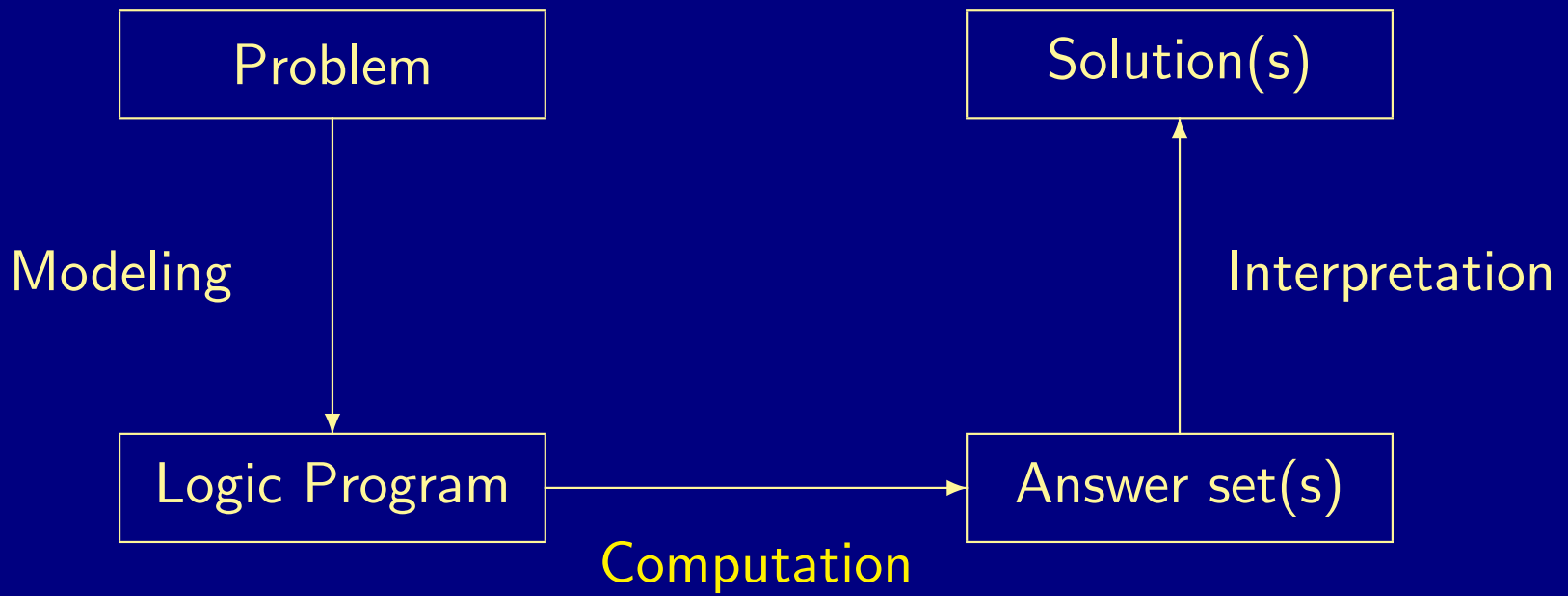
$$\leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

  where $n \geq m \geq 1$, and each A_i $(1 \leq i \leq n)$ is a atom.
- Example: $\leftarrow \mathrm{monitor}(21\mathrm{in}), \mathrm{graphics}(\mathrm{evil})$
- Implementation: For a new symbol x, map

$$\begin{array}{rrcl} & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n \\ \mapsto \quad x & \leftarrow & A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n, \mathit{not}\ x \end{array}$$

- Another example: $\Pi = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p\}$
  versus $\Pi' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow\ p\}$
  versus $\Pi'' = \{p \leftarrow \mathit{not}\ q,\ q \leftarrow \mathit{not}\ p,\ \leftarrow \mathit{not}\ p\}$

Problem solving in ASP: Computation



Standard Computation Scheme

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

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: $\mathcal{O}(2^{|\mathcal{A}|})$

Standard Computation Scheme

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

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:

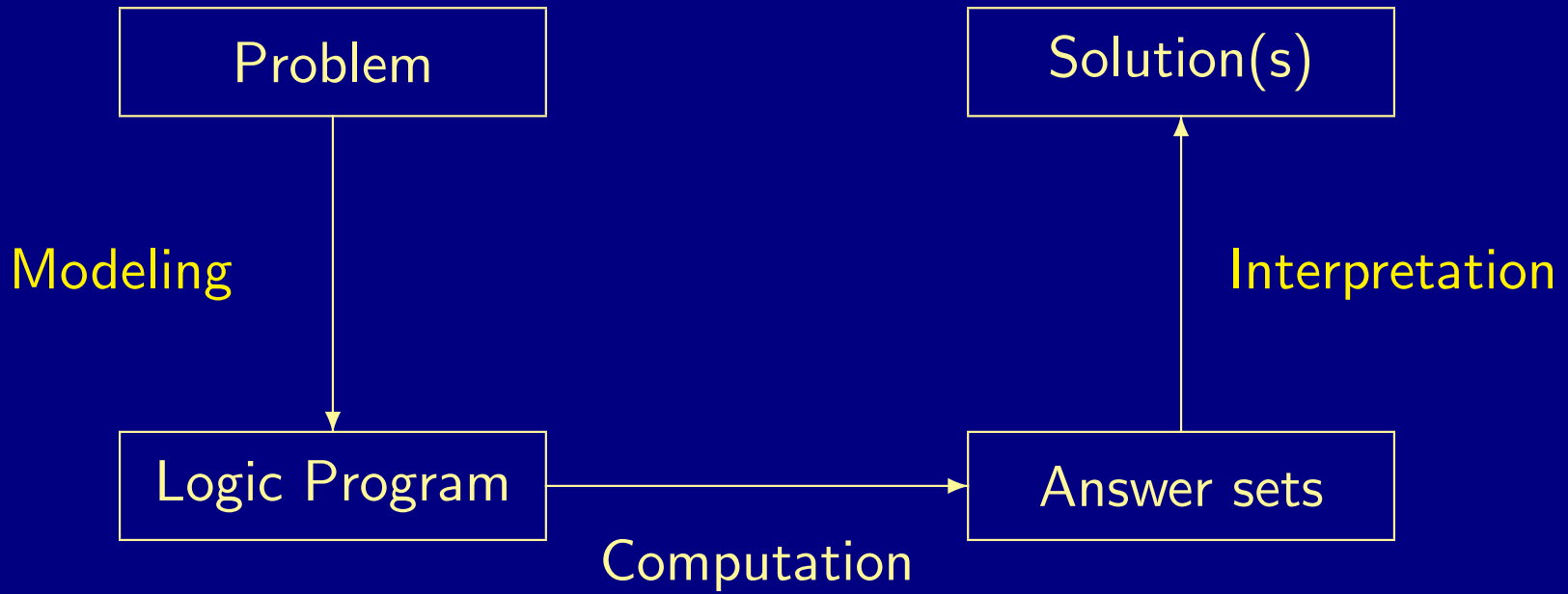
- (X, Y) is supposed to be a 3-valued model such that $X \subseteq Z$ and $Y \cap Z = \emptyset$ for any answer set Z of Π.
- Key operations: *propagation*$_\Pi(X, Y)$ and **'select** $A \in \mathcal{A} \setminus (X \cup Y)$**'**
- Worst case complexity: $\mathcal{O}(2^{|\mathcal{A}|})$

Modeling
Overview

12 Problems as Logic Programs
- Coloring Graphs

13 Methodology
- Satisfiability
- Queens
- Hamiltonian Path
- Planning

Modeling and Interpreting



Problem $\longmapsto$ 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 $C(P) \cup C(I)$.

n-colorability of graphs

Problem instance

A graph (V, E).

Problem class

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

3–colorability of graphs

C(I)	vertex(1) ←	edge(1,2) ←	
	vertex(2) ←	edge(2,3) ←	
	vertex(3) ←	edge(3,1) ←	
C(P)	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)	
Answer set	{ colored(1,r), colored(2,b), colored(3,g), . . . }		

n-colorability of graphs

with $n = 3$

C(I)	vertex(1) ← edge(1,2) ← vertex(2) ← edge(2,3) ← vertex(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,C'), C≠C', vertex(V), color(C) ← edge(V,U), colored(V,C), colored(U,C)
Answer set	{ colored(1,r), colored(2,b), colored(3,g), . . . }

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).
```

```
:- edge(V,U), colored(V,C), colored(U,C).
```

Let it run!

```
torsten@raz > gringo 3color.lp | clasp --stats 0

clasp version 1.3.5
Reading from stdin
Solving...
Answer: 1
vertex(1) vertex(2) vertex(3)
edge(1,2) edge(2,3) edge(3,1)
color(r) color(b) color(g)
colored(3,g) othercolor(3,b) othercolor(3,r)
othercolor(2,g) colored(2,b) othercolor(2,r)
othercolor(1,g) othercolor(1,b) colored(1,r)
```

Let it run!

```
torsten@raz > gringo 3color.lp | clasp --stats 0

clasp version 1.3.5
Reading from stdin
Solving...
Answer: 1
vertex(1) vertex(2) vertex(3)
edge(1,2) edge(2,3) edge(3,1)
color(r) color(b) color(g)
colored(3,g) othercolor(3,b) othercolor(3,r)
othercolor(2,g) colored(2,b) othercolor(2,r)
othercolor(1,g) othercolor(1,b) colored(1,r)
```

Here's the rest!

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

And finally some statistics!

```
Models       : 6
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
Choices      : 5
Conflicts    : 0
Restarts     : 0

Atoms        : 28
Rules        : 45     (1: 45)
Bodies       : 28
Equivalences: 41     (Atom=Atom: 8 Body=Body: 9 Other: 24)
Tight        : Yes

Variables    : 21     (Eliminated:    0)
Constraints : 42     (Binary: 78.6% Ternary: 21.4% Other:  0.0%)
Lemmas       : 0      (Binary:  0.0% Ternary:  0.0% Other:  0.0%)
  Conflicts : 0      (Average Length: 0.0)
  Loops      : 0      (Average Length: 0.0)
  Deleted    : 0
```

Basic Methodology

Generate and Test (or: Guess and Check) approach

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

Tester Eliminate invalid candidates
(typically through integrity constraints)

Nutshell

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

Basic Methodology

Generate and Test (or: Guess and Check) approach

Generator	Generate potential answer set candidates (typically through non-deterministic constructs)
Tester	Eliminate invalid candidates (typically through integrity constraints)

Nutshell

$$\text{Logic program} = \text{Data} + \text{Generator} + \text{Tester} \quad (+ \text{Optimizer})$$

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.

Satisfiability

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

Generator

$$
\begin{array}{lcl}
\mathrm{a} & \leftarrow & \text{not } \mathrm{a}' \\
\mathrm{a}' & \leftarrow & \text{not a} \\
\mathrm{b} & \leftarrow & \text{not } \mathrm{b}' \\
\mathrm{b}' & \leftarrow & \text{not b}
\end{array}
$$

Tester

$$
\begin{array}{lcl}
\leftarrow & & \text{not a, b} \\
\leftarrow & & \text{a, not b}
\end{array}
$$

Answer set

$$
\begin{array}{lcl}
\mathrm{X}_1 & = & \{\mathrm{a},\mathrm{b}\} \\
\mathrm{X}_2 & = & \{\mathrm{a}',\mathrm{b}'\}
\end{array}
$$

Sneak Preview: Generator with a choice rule: $\{\mathrm{a},\mathrm{b}\} \leftarrow$

Satisfiability

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

Generator

$$
\begin{array}{lcl}
\mathrm{a} & \leftarrow & \text{not } \mathrm{a}' \\
\mathrm{a}' & \leftarrow & \text{not a} \\
\mathrm{b} & \leftarrow & \text{not } \mathrm{b}' \\
\mathrm{b}' & \leftarrow & \text{not b}
\end{array}
$$

Tester

$$
\begin{array}{lcl}
\leftarrow & & \text{not a, b} \\
\leftarrow & & \text{a, not b}
\end{array}
$$

Answer set

$$
\begin{array}{lcl}
\mathrm{X}_1 & = & \{\mathrm{a},\mathrm{b}\} \\
\mathrm{X}_2 & = & \{\mathrm{a}',\mathrm{b}'\}
\end{array}
$$

Sneak Preview: Generator with a choice rule: $\{\mathrm{a},\mathrm{b}\} \leftarrow$

Satisfiability

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

Generator

$$
\begin{array}{lcl}
\mathrm{a} & \leftarrow & \text{not } \mathrm{a}' \\
\mathrm{a}' & \leftarrow & \text{not a} \\
\mathrm{b} & \leftarrow & \text{not } \mathrm{b}' \\
\mathrm{b}' & \leftarrow & \text{not b}
\end{array}
$$

Tester

$$
\begin{array}{lcl}
\leftarrow & & \text{not a, b} \\
\leftarrow & & \text{a, not b}
\end{array}
$$

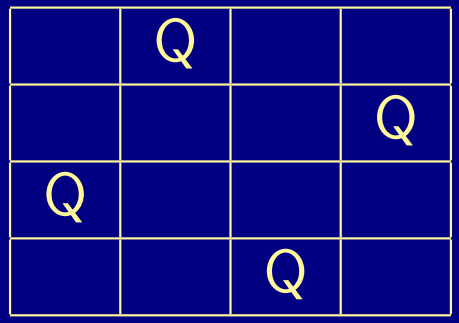
Answer set

$$
\begin{array}{lcl}
\mathrm{X}_1 & = & \{\mathrm{a},\mathrm{b}\} \\
\mathrm{X}_2 & = & \{\mathrm{a}',\mathrm{b}'\}
\end{array}
$$

Sneak Preview: Generator with a choice rule: $\{\mathrm{a},\mathrm{b}\} \leftarrow$

n-Queens Problem

A solution to $n = 4$:



n-Queens in ASP

- $q(X, Y)$ gives the legal position of a queen
- $\neg q(X, Y)$ is an independent auxiliary atom

$$
\begin{array}{rcl}
q(X,Y) & \leftarrow & not\ \neg q(X,Y) \\
\neg q(X,Y) & \leftarrow & not\ \ q(X,Y) \\
 & & \\
 & \leftarrow & q(X,Y), q(X',Y), X < X' \\
 & \leftarrow & q(X,Y), q(X,Y'), Y < Y' \\
 & \leftarrow & q(X,Y), q(X',Y'), X < X', X' - X = |Y' - Y| \\
 & & \\
hasq(X) & \leftarrow & q(X,Y) \\
 & \leftarrow & not\ hasq(X)
\end{array}
$$

n-Queens in ASP

- $q(X, Y)$ gives the legal position of a queen
- $\neg q(X, Y)$ is an independent auxiliary atom

$$
\begin{array}{rcl}
q(X,Y) & \leftarrow & \mathit{not}\ \neg q(X,Y) \\
\neg q(X,Y) & \leftarrow & \mathit{not}\ \ q(X,Y)
\end{array}
$$

$$
\begin{array}{rcl}
 & \leftarrow & q(X,Y), q(X',Y), X < X' \\
 & \leftarrow & q(X,Y), q(X,Y'), Y < Y' \\
 & \leftarrow & q(X,Y), q(X',Y'), X < X', X' - X = |Y' - Y|
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{hasq}(X) & \leftarrow & q(X,Y) \\
 & \leftarrow & \mathit{not}\ \mathit{hasq}(X)
\end{array}
$$

n-Queens in ASP

- $q(X, Y)$ gives the legal position of a queen
- $\neg q(X, Y)$ is an independent auxiliary atom

$$
\begin{array}{rcl}
q(X,Y) & \leftarrow & \mathit{not}\ \neg q(X,Y) \\
\neg q(X,Y) & \leftarrow & \mathit{not}\ \ q(X,Y) \\
 & & \\
 & \leftarrow & q(X,Y), q(X',Y), X < X' \\
 & \leftarrow & q(X,Y), q(X,Y'), Y < Y' \\
 & \leftarrow & q(X,Y), q(X',Y'), X < X', X' - X = |Y' - Y| \\
 & & \\
\mathit{hasq}(X) & \leftarrow & q(X,Y) \\
 & \leftarrow & \mathit{not}\ \mathit{hasq}(X)
\end{array}
$$

n-Queens in ASP

- $q(X, Y)$ gives the legal position of a queen
- $\neg q(X, Y)$ is an independent auxiliary atom

$$
\begin{array}{rcl}
q(X,Y) & \leftarrow & \mathit{not}\ \neg q(X,Y) \\
\neg q(X,Y) & \leftarrow & \mathit{not}\ \ q(X,Y) \\
 & & \\
 & \leftarrow & q(X,Y), q(X',Y), X < X' \\
 & \leftarrow & q(X,Y), q(X,Y'), Y < Y' \\
 & \leftarrow & q(X,Y), q(X',Y'), X < X', X' - X = |Y' - Y| \\
 & & \\
\mathit{hasq}(X) & \leftarrow & q(X,Y) \\
 & \leftarrow & \mathit{not}\ \mathit{hasq}(X)
\end{array}
$$

n-Queens

(in the gringo language)

```
d(1..queens).

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

:- q(X,Y), q(X1,Y),  X < X1.
:- q(X,Y), q(X,Y1),  Y < Y1.
:- q(X,Y), q(X1,Y1), X < X1, X1 - X == |Y1 - Y|.

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

n-Queens

(in the gringo language)

```
d(1..queens).

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

:- q(X,Y), q(X1,Y),  X < X1.
:- q(X,Y), q(X,Y1),  Y < Y1.
:- q(X,Y), q(X1,Y1), X < X1, X1 - X == |Y1 - Y|.

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

n-Queens

(in the gringo language)

```
d(1..queens).

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

:- q(X,Y), q(X1,Y),  X < X1.
:- q(X,Y), q(X,Y1),  Y < Y1.
:- q(X,Y), q(X1,Y1), X < X1, X1 - X == |Y1 - Y|.

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

n-Queens

(in the gringo language)

```
d(1..queens).

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

:- q(X,Y), q(X1,Y),  X < X1.
:- q(X,Y), q(X,Y1),  Y < Y1.
:- q(X,Y), q(X1,Y1), X < X1, X1 - X == |Y1 - Y|.

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

n-Queens in ASP (ctd)

Alternatives to

$$q(X,Y) \leftarrow not\ \neg q(X,Y)$$
$$\neg q(X,Y) \leftarrow not\ q(X,Y)$$

in

- disjunctive ASP [41]

  $$q(X,Y)\ ;\neg q(X,Y) \leftarrow$$

- nested ASP [55]

  $$q(X,Y)\ ;\ not\ q(X,Y) \leftarrow$$

- normal ASP with cardinality constraints [73]

  $$1\ \{\ q(X,Y)\ \}\ 1 \leftarrow d(X)$$
  $$1\ \{\ q(X,Y)\ \}\ 1 \leftarrow d(Y)$$

n-Queens in ASP (ctd)

Alternatives to

$$q(X,Y) \leftarrow not\ \neg q(X,Y)$$
$$\neg q(X,Y) \leftarrow not\ q(X,Y)$$

in

- disjunctive ASP [41]

  $$q(X,Y) ; \neg q(X,Y) \leftarrow$$

- nested ASP [55]

  $$q(X,Y) ; not\ q(X,Y) \leftarrow$$

- normal ASP with cardinality constraints [73]

  $$1 \{ q(X,Y) \} 1 \leftarrow d(X)$$
  $$1 \{ q(X,Y) \} 1 \leftarrow d(Y)$$

n-Queens (ctd)

(with cardinality constraints)

$$
\begin{aligned}
d(1..n) &\leftarrow \\
1\,\{\, q(X,Y) \,\}\,1 &\leftarrow d(X) \\
1\,\{\, q(X,Y) \,\}\,1 &\leftarrow d(Y) \\
&\leftarrow q(X,Y), q(X',Y'), X < X', X' - X = |Y' - Y|
\end{aligned}
$$

n-Queens (ctd)

(in the gringo language with cardinality constraints)

```
d(1..queens).

1 { q(X,Y) : d(Y) } 1 :- d(X).
1 { q(X,Y) : d(X) } 1 :- d(Y).

:- q(X,Y), q(X1,Y1), X < X1, X1 - X == |Y1 - Y|.
```

And the Performance . . . ?

```
torsten@raz > gringo -c queens=100 queens2.lp | clasp --stats
Answer: 1
d(1) d(2) d(3) d(4) d(5) ... d(96) d(97) d(98) d(99) d(100)
q(100,62) q(99,79) q(98,29) q(97,60) ... q(4,69) q(3,52) q(2,64) q(1,71)
SATISFIABLE

Models       : 1+
Time         : 3.420s (Solving: 0.57s 1st Model: 0.57s Unsat: 0.00s)
Choices      : 1256
Conflicts    : 818
Restarts     : 4

Atoms        : 10701
Rules        : 658000 (1: 657400 2: 400 3: 200)

Variables    : 667100 (Eliminated:    0)
Constraints  : 657100 (Binary: 99.9% Ternary:  0.0% Other:  0.1%)
Lemmas       : 818    (Binary:  0.0% Ternary:  0.0% Other:100.0%)
  Conflicts  : 818    (Average Length: 610.1)
  Loops      : 0      (Average Length: 0.0)
  Deleted    : 0
```

Stop! It's DEMO time!

Hamiltonian Path

Problem instance

A directed graph (V, E) and a starting vertex $v \in V$.

Problem class

Find a path in (V, E) starting at v and visiting all other vertices in V exactly once.

- Predicates: *vertex/1*, *arc/2*, *start/1*

Hamiltonian Path

Problem instance

A directed graph (V, E) and a starting vertex $v \in V$.

Problem class

Find a path in (V, E) starting at v and visiting all other vertices in V exactly once.

- Predicates: *vertex/1*, *arc/2*, *start/1*

Generator

for candidate paths

Normal logic programs

$$inPath(X,Y) \leftarrow arc(X,Y),\ not\ outPath(X,Y)$$
$$outPath(X,Y) \leftarrow arc(X,Y),\ not\ inPath(X,Y)$$

Disjunctive logic programs

$$inPath(X,Y)\ ;\ outPath(X,Y) \leftarrow arc(X,Y)$$

Nested logic programs

$$inPath(X,Y)\ ;\ not\ inPath(X,Y) \leftarrow arc(X,Y)$$

Logic programs with cardinality constraints

$$1\ \{inPath(X,Y) : arc(X,Y)\}\ 1 \leftarrow vertex(Y), not\ start(Y)$$
$$\{inPath(X,Y) : arc(X,Y)\}\ 1 \leftarrow vertex(X)$$

Generator

for candidate paths

Normal logic programs

$$inPath(X,Y) \leftarrow arc(X,Y),\ not\ outPath(X,Y)$$
$$outPath(X,Y) \leftarrow arc(X,Y),\ not\ inPath(X,Y)$$

Disjunctive logic programs

$$inPath(X,Y)\ ;\ outPath(X,Y) \leftarrow arc(X,Y)$$

Nested logic programs

$$inPath(X,Y)\ ;\ not\ inPath(X,Y) \leftarrow arc(X,Y)$$

Logic programs with cardinality constraints

$$1\ \{inPath(X,Y) : arc(X,Y)\}\ 1 \leftarrow vertex(Y), not\ start(Y)$$
$$\{inPath(X,Y) : arc(X,Y)\}\ 1 \leftarrow vertex(X)$$

Generator

for candidate paths

Normal logic programs

$$\begin{aligned} inPath(X,Y) &\leftarrow arc(X,Y),\ not\ outPath(X,Y) \\ outPath(X,Y) &\leftarrow arc(X,Y),\ not\ inPath(X,Y) \end{aligned}$$

Disjunctive logic programs

$$inPath(X,Y)\ ;\ outPath(X,Y) \leftarrow arc(X,Y)$$

Nested logic programs

$$inPath(X,Y)\ ;\ not\ inPath(X,Y) \leftarrow arc(X,Y)$$

Logic programs with cardinality constraints

$$\begin{aligned} 1\ \{inPath(X,Y) : arc(X,Y)\}\ 1 &\leftarrow vertex(Y), not\ start(Y) \\ \{inPath(X,Y) : arc(X,Y)\}\ 1 &\leftarrow vertex(X) \end{aligned}$$

Generator

for candidate paths

Normal logic programs

$$\begin{aligned} inPath(X,Y) &\leftarrow arc(X,Y),\ not\ outPath(X,Y) \\ outPath(X,Y) &\leftarrow arc(X,Y),\ not\ inPath(X,Y) \end{aligned}$$

Disjunctive logic programs

$$inPath(X,Y)\ ;\ outPath(X,Y) \leftarrow arc(X,Y)$$

Nested logic programs

$$inPath(X,Y)\ ;\ not\ inPath(X,Y) \leftarrow arc(X,Y)$$

Logic programs with cardinality constraints

$$\begin{aligned} 1\ \{inPath(X,Y) : arc(X,Y)\}\ 1 &\leftarrow vertex(Y), not\ start(Y) \\ \{inPath(X,Y) : arc(X,Y)\}\ 1 &\leftarrow vertex(X) \end{aligned}$$

Tester

eliminating invalid candidate paths

- Eliminate candidates having vertices visited more than once

$$\leftarrow \quad inPath(X,Y),\ inPath(Z,Y),\ X < Z$$
$$\leftarrow \quad inPath(X,Y),\ inPath(X,Z),\ Y < Z$$

(obsolete when using above cardinality constraints)

- Eliminate candidates having vertices never visited

$$reached(X) \leftarrow start(X)$$
$$reached(X) \leftarrow reached(Y),\ inPath(Y,X)$$
$$\leftarrow vertex(X),\ not\ reached(X)$$

Tester

eliminating invalid candidate paths

- Eliminate candidates having vertices visited more than once

$$\leftarrow \quad inPath(X,Y),\ inPath(Z,Y),\ X<Z$$
$$\leftarrow \quad inPath(X,Y),\ inPath(X,Z),\ Y<Z$$

(obsolete when using above cardinality constraints)

- Eliminate candidates having vertices never visited

$$reached(X) \leftarrow start(X)$$
$$reached(X) \leftarrow reached(Y),\ inPath(Y,X)$$
$$\leftarrow vertex(X),\ not\ reached(X)$$

Try for yourself: Hamiltonian Cycle

Problem instance

A directed graph (V, E).

Problem class

Find a cycle in (V, E) visiting all vertices in V exactly once.

Making Mr Hamilton a Traveling Salesman

Problem instance

A directed graph (V, E) and a cost function on arcs.

Problem class

Find a cycle in (V, E) visiting all vertices in V exactly once such that the sum of costs associated with the arcs is minimal.

- Predicates: $vertex/1$, $arc/2$, $start/1$, and $cost/3$, eg. $cost(3, 4, 30)$
- Optimization

$$\#minimize\ [inPath(X, Y) : cost(X, Y, C) = C].$$

Making Mr Hamilton a Traveling Salesman

Problem instance

A directed graph (V, E) and a cost function on arcs.

Problem class

Find a cycle in (V, E) visiting all vertices in V exactly once such that the sum of costs associated with the arcs is minimal.

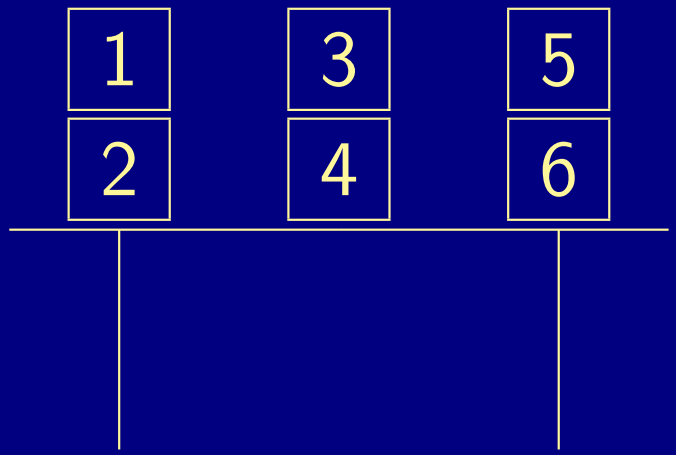
- Predicates: $vertex/1$, $arc/2$, $start/1$, and $cost/3$, eg. $cost(3, 4, 30)$
- Optimization

$$\#minimize\ [inPath(X, Y) : cost(X, Y, C) = C].$$

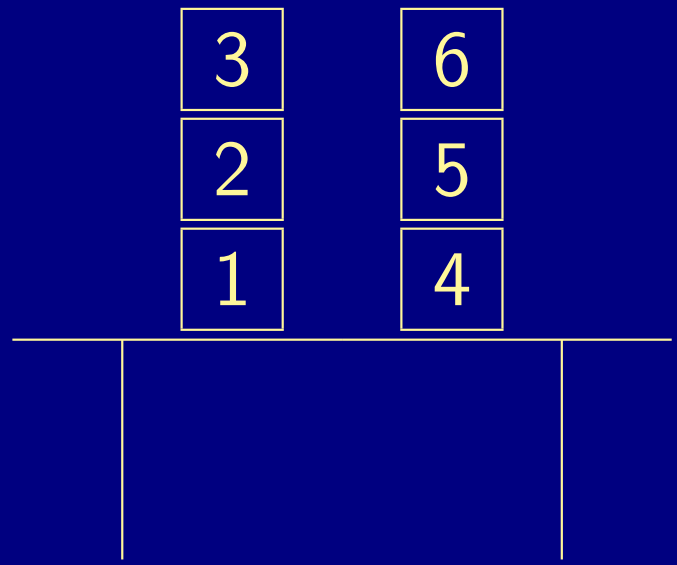
Planning

in the Blocksworld [51]

Initial situation



Goal situation



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).
```

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).
```

Planning in the Blocksworld I

GENERATE

```
time(0..lasttime).

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

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

Planning in the Blocksworld II

DEFINE

```
% blocks that move somewhere
move(B,T)   :- move(B,_,T).

% effect of moving a block
on(B,L,T+1) :- move(B,L,T).

% inertia
on(B,L,T+1) :- on(B,L,T), not move(B,T), T < lasttime.
```

Planning in the Blocksworld III

TEST

```
% a block must be at exactly one location
:- not 1 { on(B,L,T) } 1, block(B), time(T).

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

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

% a block can't be moved onto a block that is moved also
:- move(B,B1,T), move(B1,T).
```

Planning in the Blocksworld IV

DISPLAY

```
% hide everything but the moves
#hide.
#show move/3.
```

The Plan

```
torsten@raz > gringo blocks.lp | clasp --stats
Answer: 1
move(6,5,2) move(3,2,2) move(5,4,1) move(2,1,1) move(3,table,0) move(1,table,0)
SATISFIABLE

Models       : 1+
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
Choices      : 2
Conflicts    : 0
Restarts     : 0

Atoms        : 336
Rules        : 596    (1: 536 2: 57 3: 3)

Variables    : 501    (Eliminated:    0)
Constraints  : 411    (Binary: 77.6% Ternary: 11.7% Other: 10.7%)
Lemmas       : 0      (Binary:  0.0% Ternary:  0.0% Other:  0.0%)
  Conflicts  : 0      (Average Length: 0.0)
  Loops      : 0      (Average Length: 0.0)
  Deleted    : 0
```

Disjunctive logic programs
Overview

14 Syntax

15 Semantics

16 Examples

Disjunctive logic programs

[40]

- A disjunctive rule, r, is an ordered pair of the form

$$A_1 \; ; \ldots ; A_m \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o,$$

  where $o \geq n \geq m \geq 0$, and each A_i $(0 \leq i \leq o)$ is an atom.
- A disjunctive logic program is a finite set of disjunctive rules.
- (Generalized) Notation

$$\begin{aligned} \mathit{head}(r) &= \{A_1, \ldots, A_m\} \\ \mathit{body}(r) &= \{A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o\} \\ \mathit{body}^+(r) &= \{A_{m+1}, \ldots, A_n\} \\ \mathit{body}^-(r) &= \{A_{n+1}, \ldots, A_o\} \end{aligned}$$

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

Answer sets

- Positive programs:
  - A set X of atoms is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \cap X \neq \emptyset$ whenever $body^+(r) \subseteq X$.
    - ➥ X corresponds to a model of Π (seen as a formula).
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program Π is denoted by $\min_{\subseteq}(\Pi)$.
    - ➥ $\min_{\subseteq}(\Pi)$ corresponds to the $\subseteq$-minimal models of Π (ditto).
- Disjunctive programs:
  - The reduct, Π^X, of a disjunctive program Π relative to a set X of atoms is defined by

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

  - A set X of atoms is an answer set of a disjunctive program Π if $X \in \min_{\subseteq}(\Pi^X)$.
- FYI: The alternative definition on Page 99 is applicable as well.

Answer sets

- Positive programs:
  - A set X of atoms is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \cap X \neq \emptyset$ whenever $body^+(r) \subseteq X$.
    - ➥ X corresponds to a model of Π (seen as a formula).
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program Π is denoted by $\min_{\subseteq}(\Pi)$.
    - ➥ $\min_{\subseteq}(\Pi)$ corresponds to the $\subseteq$-minimal models of Π (ditto).
- Disjunctive programs:
  - The reduct, Π^X, of a disjunctive program Π relative to a set X of atoms is defined by

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

  - A set X of atoms is an answer set of a disjunctive program Π if $X \in \min_{\subseteq}(\Pi^X)$.
- FYI: The alternative definition on Page 99 is applicable as well.

Answer sets

- Positive programs:
  - A set X of atoms is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \cap X \neq \emptyset$ whenever $body^+(r) \subseteq X$.
    - ➥ X corresponds to a model of Π (seen as a formula).
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program Π is denoted by $\min_{\subseteq}(\Pi)$.
    - ➥ $\min_{\subseteq}(\Pi)$ corresponds to the $\subseteq$-minimal models of Π (ditto).
- Disjunctive programs:
  - The reduct, Π^X, of a disjunctive program Π relative to a set X of atoms is defined by

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

  - A set X of atoms is an answer set of a disjunctive program Π if $X \in \min_{\subseteq}(\Pi^X)$.
- FYI: The alternative definition on Page 99 is applicable as well.

Answer sets

- Positive programs:
  - A set X of atoms is closed under a positive program Π iff for any $r \in \Pi$, $head(r) \cap X \neq \emptyset$ whenever $body^+(r) \subseteq X$.
    - ➥ X corresponds to a model of Π (seen as a formula).
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program Π is denoted by $\min_{\subseteq}(\Pi)$.
    - ➥ $\min_{\subseteq}(\Pi)$ corresponds to the $\subseteq$-minimal models of Π (ditto).
- Disjunctive programs:
  - The reduct, Π^X, of a disjunctive program Π relative to a set X of atoms is defined by

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

  - A set X of atoms is an answer set of a disjunctive program Π if $X \in \min_{\subseteq}(\Pi^X)$.
- FYI: The alternative definition on Page 99 is applicable as well.

A "positive" example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b\,;c & \leftarrow & a \end{array} \right\}$$

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under Π.
- We have $\min_{\subseteq}(\Pi) = \{\ \{a, b\},\ \{a, c\}\ \}$.

A "positive" example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b\,;c & \leftarrow & a \end{array} \right\}$$

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under Π.
- We have $\min_{\subseteq}(\Pi) = \{\ \{a, b\},\ \{a, c\}\ \}$.

A "positive" example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b\,;c & \leftarrow & a \end{array} \right\}$$

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under Π.
- We have $\min_{\subseteq}(\Pi) = \{\ \{a, b\},\ \{a, c\}\ \}$.

3-colorability revisited

C(I)	vertex(1) $\leftarrow$ edge(1,2) $\leftarrow$ vertex(2) $\leftarrow$ edge(2,3) $\leftarrow$ vertex(3) $\leftarrow$ edge(3,1) $\leftarrow$
C(P)	colored(V,r); colored(V,b); colored(V,g) $\leftarrow$ vertex(V) $\leftarrow$ edge(V,U), colored(V,C), colored(U,C)
Answer set	$\{$ colored(1,r), colored(2,b), colored(3,g), $\ldots\}$

More Examples

- $\Pi_1 = \{a\,;b\,;c \leftarrow\}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$ has answer sets $\{b\}$ and $\{c\}$.
- $\Pi_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$ has answer set $\{b, c\}$.
- $\Pi_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow not\ a, not\ c\ ,\ a\,;c \leftarrow not\ b\}$
  has answer sets $\{a\}$ and $\{b\}$.

More Examples

- $\Pi_1 = \{a\,;b\,;c \leftarrow\}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$ has answer sets $\{b\}$ and $\{c\}$.
- $\Pi_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$ has answer set $\{b, c\}$.
- $\Pi_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow not\ a, not\ c\ ,\ a\,;c \leftarrow not\ b\}$
  has answer sets $\{a\}$ and $\{b\}$.

More Examples

- $\Pi_1 = \{a\ ;\ b\ ;\ c \leftarrow\}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{a\ ;\ b\ ;\ c \leftarrow\ ,\ \leftarrow a\}$ has answer sets $\{b\}$ and $\{c\}$.
- $\Pi_3 = \{a\ ;\ b\ ;\ c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$ has answer set $\{b, c\}$.
- $\Pi_4 = \{a\ ;\ b \leftarrow c\ ,\ b \leftarrow not\ a, not\ c\ ,\ a\ ;\ c \leftarrow not\ b\}$ has answer sets $\{a\}$ and $\{b\}$.

More Examples

- $\Pi_1 = \{a\ ;\ b\ ;\ c \leftarrow\}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{a\ ;\ b\ ;\ c \leftarrow\ ,\ \leftarrow a\}$ has answer sets $\{b\}$ and $\{c\}$.
- $\Pi_3 = \{a\ ;\ b\ ;\ c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$ has answer set $\{b, c\}$.
- $\Pi_4 = \{a\ ;\ b \leftarrow c\ ,\ b \leftarrow not\ a, not\ c\ ,\ a\ ;\ c \leftarrow not\ b\}$
  has answer sets $\{a\}$ and $\{b\}$.

More Examples

- $\Pi_1 = \{a\ ;\ b\ ;\ c \leftarrow\}$ has answer sets $\{a\}$, $\{b\}$, and $\{c\}$.
- $\Pi_2 = \{a\ ;\ b\ ;\ c \leftarrow\ ,\ \leftarrow a\}$ has answer sets $\{b\}$ and $\{c\}$.
- $\Pi_3 = \{a\ ;\ b\ ;\ c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$ has answer set $\{b, c\}$.
- $\Pi_4 = \{a\ ;\ b \leftarrow c\ ,\ b \leftarrow not\ a, not\ c\ ,\ a\ ;\ c \leftarrow not\ b\}$
  has answer sets $\{a\}$ and $\{b\}$.

An example with variables

$$\Pi = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X)\,;c(Y) & \leftarrow & a(X,Y), \mathit{not}\ c(Y) \end{array} \right\}$$

$$\mathit{ground}(\Pi) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \mathit{not}\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \mathit{not}\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \mathit{not}\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \mathit{not}\ c(2) \end{array} \right\}$$

For every answer set X of Π, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$.

An example with variables

$$\Pi = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X)\,;c(Y) & \leftarrow & a(X,Y), \mathit{not}\ c(Y) \end{array} \right\}$$

$$\mathit{ground}(\Pi) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \mathit{not}\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \mathit{not}\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \mathit{not}\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \mathit{not}\ c(2) \end{array} \right\}$$

For every answer set X of Π, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$.

An example with variables

$$\Pi = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X)\,;c(Y) & \leftarrow & a(X,Y), \mathit{not}\ c(Y) \end{array} \right\}$$

$$\mathit{ground}(\Pi) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \mathit{not}\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \mathit{not}\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \mathit{not}\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \mathit{not}\ c(2) \end{array} \right\}$$

For every answer set X of Π, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ b(1)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2),\ b(1)\},\ \{a(1,2),\ c(2)\}\ \}$.
- X is an answer set of Π because $X \in \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ b(1)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2),\ b(1)\},\ \{a(1,2),\ c(2)\}\ \}$.
- X is an answer set of Π because $X \in \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) \,;\, c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1) \,;\, c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2) \,;\, c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2) \,;\, c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ b(1)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2),\ b(1)\},\ \{a(1,2),\ c(2)\}\ \}$.
- X is an answer set of Π because $X \in \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ b(1)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2),\ b(1)\},\ \{a(1,2),\ c(2)\}\ \}$.
- X is an answer set of Π because $X \in \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ b(1)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2),\ b(1)\},\ \{a(1,2),\ c(2)\}\ \}$.
- X is an answer set of Π because $X \in \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ c(2)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2)\}\ \}$.
- X is no answer set of Π because $X \notin \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ c(2)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2)\}\ \}$.
- X is no answer set of Π because $X \notin \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) \,;\, c(1) & \leftarrow & a(1,1), \mathit{not}\ c(1) \\ b(1) \,;\, c(2) & \leftarrow & a(1,2), \mathit{not}\ c(2) \\ b(2) \,;\, c(1) & \leftarrow & a(2,1), \mathit{not}\ c(1) \\ b(2) \,;\, c(2) & \leftarrow & a(2,2), \mathit{not}\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ c(2)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2)\}\ \}$.
- X is no answer set of Π because $X \notin \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ c(2)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2)\}\ \}$.
- X is no answer set of Π because $X \notin \min_{\subseteq}(ground(\Pi)^X)$.

An example with variables

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) \,;\, c(1) & \leftarrow & a(1,1), not\ c(1) \\ b(1) \,;\, c(2) & \leftarrow & a(1,2), not\ c(2) \\ b(2) \,;\, c(1) & \leftarrow & a(2,1), not\ c(1) \\ b(2) \,;\, c(2) & \leftarrow & a(2,2), not\ c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2),\ c(2)\}$.
- We get $\min_{\subseteq}(ground(\Pi)^X) = \{\ \{a(1,2)\}\ \}$.
- X is no answer set of Π because $X \notin \min_{\subseteq}(ground(\Pi)^X)$.

Nested logic programs
Overview

17 Syntax

18 Semantics

19 Examples

20 Implementation

Nested logic programs

[55]

- Formulas are formed from
  - propositional atoms and
  - $\top$ and $\bot$

  using
  - negation-as-failure (*not*),
  - conjunction (,), and
  - disjunction (;).
- A nested rule, r, is an ordered pair of the form $F \leftarrow G$ where F and G are formulas.
- A nested program is a finite set of rules.
- Notation: $head(r) = F$ and $body(r) = G$.

Nested logic programs

[55]

- Formulas are formed from
  - propositional atoms and
  - $\top$ and $\bot$

  using
  - negation-as-failure (*not*),
  - conjunction (,), and
  - disjunction (;).
- A nested rule, r, is an ordered pair of the form $F \leftarrow G$ where F and G are formulas.
- A nested program is a finite set of rules.
- Notation: $head(r) = F$ and $body(r) = G$.

Nested logic programs

[55]

- Formulas are formed from
  - propositional atoms and
  - $\top$ and $\bot$

  using
  - negation-as-failure (*not*),
  - conjunction (,), and
  - disjunction (;).
- A nested rule, r, is an ordered pair of the form $F \leftarrow G$ where F and G are formulas.
- A nested program is a finite set of rules.
- Notation: $head(r) = F$ and $body(r) = G$.

Satisfaction relation

- The satisfaction relation $X \models F$ between a set of atoms and a formula F is defined recursively as follows:
  - $X \models F$ if $F \in X$ for an atom F,
  - $X \models \top$,
  - $X \not\models \bot$,
  - $X \models (F, G)$ if $X \models F$ and $X \models G$,
  - $X \models (F; G)$ if $X \models F$ or $X \models G$,
  - $X \models \mathit{not}\ F$ if $X \not\models F$.
- A set X of atoms satisfies a nested program Π, written $X \models \Pi$, iff for any $r \in \Pi$, $X \models \mathit{head}(r)$ whenever $X \models \mathit{body}(r)$.
- The set of all $\subseteq$-minimal sets of atoms satisfying program Π is denoted by $\min_{\subseteq}(\Pi)$.

Satisfaction relation

- The satisfaction relation $X \models F$ between a set of atoms and a formula F is defined recursively as follows:
  - $X \models F$ if $F \in X$ for an atom F,
  - $X \models \top$,
  - $X \not\models \bot$,
  - $X \models (F, G)$ if $X \models F$ and $X \models G$,
  - $X \models (F; G)$ if $X \models F$ or $X \models G$,
  - $X \models \mathit{not}\ F$ if $X \not\models F$.
- A set X of atoms satisfies a nested program Π, written $X \models \Pi$, iff for any $r \in \Pi$, $X \models \mathit{head}(r)$ whenever $X \models \mathit{body}(r)$.
- The set of all $\subseteq$-minimal sets of atoms satisfying program Π is denoted by $\min_{\subseteq}(\Pi)$.

Satisfaction relation

- The satisfaction relation $X \models F$ between a set of atoms and a formula F is defined recursively as follows:
  - $X \models F$ if $F \in X$ for an atom F,
  - $X \models \top$,
  - $X \not\models \bot$,
  - $X \models (F, G)$ if $X \models F$ and $X \models G$,
  - $X \models (F; G)$ if $X \models F$ or $X \models G$,
  - $X \models \mathit{not}\ F$ if $X \not\models F$.
- A set X of atoms satisfies a nested program Π, written $X \models \Pi$, iff for any $r \in \Pi$, $X \models \mathit{head}(r)$ whenever $X \models \mathit{body}(r)$.
- The set of all $\subseteq$-minimal sets of atoms satisfying program Π is denoted by $\min_{\subseteq}(\Pi)$.

Reduct

- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = F$ if F is an atom or $\top$ or $\bot$,
  - $(F, G)^X = (F^X, G^X)$,
  - $(F; G)^X = (F^X; G^X)$,
  - $(not\ F)^X = \begin{cases} \bot & \text{if } X \models F \\ \top & \text{otherwise} \end{cases}$
- The reduct, Π^X, of a nested program Π relative to a set X of atoms is defined by

$$\Pi^X = \{head(r)^X \leftarrow body(r)^X \mid r \in \Pi\}.$$

- A set X of atoms is an answer set of a nested program Π if $X \in \min_{\subseteq}(\Pi^X)$.

Reduct

- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = F$ if F is an atom or $\top$ or $\bot$,
  - $(F, G)^X = (F^X, G^X)$,
  - $(F; G)^X = (F^X; G^X)$,
  - $(not\ F)^X = \begin{cases} \bot & \text{if } X \models F \\ \top & \text{otherwise} \end{cases}$
- The reduct, Π^X, of a nested program Π relative to a set X of atoms is defined by

$$\Pi^X = \{head(r)^X \leftarrow body(r)^X \mid r \in \Pi\}.$$

- A set X of atoms is an answer set of a nested program Π if $X \in \min_{\subseteq}(\Pi^X)$.

Reduct

- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = F$ if F is an atom or $\top$ or $\bot$,
  - $(F, G)^X = (F^X, G^X)$,
  - $(F; G)^X = (F^X; G^X)$,
  - $(not\ F)^X = \begin{cases} \bot & \text{if } X \models F \\ \top & \text{otherwise} \end{cases}$
- The reduct, Π^X, of a nested program Π relative to a set X of atoms is defined by

$$\Pi^X = \{head(r)^X \leftarrow body(r)^X \mid r \in \Pi\}.$$

- A set X of atoms is an answer set of a nested program Π if $X \in \min_{\subseteq}(\Pi^X)$.

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$

  For $X = \emptyset$, we get

  $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$

  $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$.

  For $X = \{p\}$, we get

  $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$

  $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$.

  $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$.

  In general,

  $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$

  $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$

  $not\ not\ not\ F$ is equivalent to $not\ F$

  ➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$.
  - For $X = \{p\}$, we get

    $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$

    $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$.
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$.

  In general,

  $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$

  $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$

  $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$.
  - For $X = \{p\}$, we get

    $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$

    $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$.
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$

  $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ;\top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get

    $\Pi_1^{\{p\}} = \{(p\ ;\bot) \leftarrow \top\}$

    $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$.
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ;\top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ;\bot) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$.
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ;\top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ;\bot) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$.
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$.
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$

  For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$.

  For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$.
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$.
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$.
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$. ✔
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$. ✔
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^{\emptyset} = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^{\emptyset} = \{p \leftarrow \bot\}$ and $\min_{\subseteq}(\Pi_2^{\emptyset}) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_{\subseteq}(\Pi_2^{\{p\}}) = \{\{p\}\}$. ✔
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Two examples

- $\Pi_1 = \{(p\ ;\ not\ p) \leftarrow \top\}$
  - For $X = \emptyset$, we get
    - $\Pi_1^\emptyset = \{(p\ ; \top) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get
    - $\Pi_1^{\{p\}} = \{(p\ ; \bot) \leftarrow \top\}$
    - $\min_\subseteq(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✔
- $\Pi_2 = \{p \leftarrow not\ not\ p\}$
  - For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \bot\}$ and $\min_\subseteq(\Pi_2^\emptyset) = \{\emptyset\}$. ✔
  - For $X = \{p\}$, we get $\Pi_2^{\{p\}} = \{p \leftarrow \top\}$ and $\min_\subseteq(\Pi_2^{\{p\}}) = \{\{p\}\}$. ✔
- In general,
  - $F \leftarrow G,\ not\ not\ H$ is equivalent to $F\ ;\ not\ H \leftarrow G$
  - $F\ ;\ not\ not\ G \leftarrow H$ is equivalent to $F \leftarrow H, not\ G$
  - $not\ not\ not\ F$ is equivalent to $not\ F$

➥ Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932)

Some more examples

$$
\begin{aligned}
\Pi_3 &= \{p \leftarrow (q, r)\,;(not\ q, not\ s)\} \\
\Pi_4 &= \{(p\,;not\ p), (q\,;not\ q), (r\,;not\ r) \leftarrow \top\} \\
\Pi_5 &= \{(p\,;not\ p), (q\,;not\ q), (r\,;not\ r) \leftarrow \top,\ \bot \leftarrow p, q\}
\end{aligned}
$$

Implementation

nlp `http://www.cs.uni-potsdam.de/~torsten/nlp`

Propositional Theories
Overview

21 Syntax

22 Semantics

23 Examples

24 Relationship with Logic Programs

Propositional theories

- Formulas are formed from
  - propositional atoms and
  - $\bot$

  using
  - conjunction ($\wedge$),
  - disjunction ($\vee$), and
  - implication ($\rightarrow$).
- Notation

$$\begin{aligned} \top &= (\bot \rightarrow \bot) \\ \sim F &= (F \rightarrow \bot) \qquad (\text{or: } \textit{not } F) \end{aligned}$$

A propositional theory is a finite set of formulas.

Propositional theories

- Formulas are formed from
  - propositional atoms and
  - $\bot$

  using
  - conjunction ($\wedge$),
  - disjunction ($\vee$), and
  - implication ($\rightarrow$).
- Notation

$$\begin{aligned} \top &= (\bot \rightarrow \bot) \\ \sim F &= (F \rightarrow \bot) \qquad (\text{or: } \textit{not } F) \end{aligned}$$

- A propositional theory is a finite set of formulas.

Propositional theories

- Formulas are formed from
  - propositional atoms and
  - $\bot$

  using
  - conjunction ($\wedge$),
  - disjunction ($\vee$), and
  - implication ($\rightarrow$).
- Notation

$$\begin{aligned} \top &= (\bot \rightarrow \bot) \\ \sim F &= (F \rightarrow \bot) \qquad (\text{or: } \textit{not } F) \end{aligned}$$

- A propositional theory is a finite set of formulas.

Reduct

[21]

- The satisfaction relation $X \models F$ between a set X of atoms and a (set of) formula(s) F is defined as in propositional logic.
- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = \bot$ if $X \not\models F$
  - $F^X = F$ if $F \in X$
  - $F^X = (G^X \circ H^X)$ if $X \models F$ and $F = (G \circ H)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - ➥ If $F = {\sim}G = (G \rightarrow \bot)$, then $F^X = (\bot \rightarrow \bot) = \top$, if $X \not\models G$, and $F^X = \bot$, otherwise.
- The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

$$\mathcal{F}^X = \{F^X \mid F \in \mathcal{F}\}.$$

Reduct

[21]

- The satisfaction relation $X \models F$ between a set X of atoms and a (set of) formula(s) F is defined as in propositional logic.
- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = \bot$ if $X \not\models F$
  - $F^X = F$ if $F \in X$
  - $F^X = (G^X \circ H^X)$ if $X \models F$ and $F = (G \circ H)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - ➥ If $F = {\sim}G = (G \rightarrow \bot)$, then $F^X = (\bot \rightarrow \bot) = \top$, if $X \not\models G$, and $F^X = \bot$, otherwise.
- The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

$$\mathcal{F}^X = \{F^X \mid F \in \mathcal{F}\}.$$

Reduct

[21]

- The satisfaction relation $X \models F$ between a set X of atoms and a (set of) formula(s) F is defined as in propositional logic.
- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = \bot$ if $X \not\models F$
  - $F^X = F$ if $F \in X$
  - $F^X = (G^X \circ H^X)$ if $X \models F$ and $F = (G \circ H)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$

  ➥ If $F = {\sim}G = (G \rightarrow \bot)$,
  then $F^X = (\bot \rightarrow \bot) = \top$, if $X \not\models G$, and $F^X = \bot$, otherwise.
- The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

$$\mathcal{F}^X = \{F^X \mid F \in \mathcal{F}\}.$$

Reduct

[21]

- The satisfaction relation $X \models F$ between a set X of atoms and a (set of) formula(s) F is defined as in propositional logic.
- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = \bot$ if $X \not\models F$
  - $F^X = F$ if $F \in X$
  - $F^X = (G^X \circ H^X)$ if $X \models F$ and $F = (G \circ H)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$

  ➥ If $F = {\sim}G = (G \rightarrow \bot)$, then $F^X = (\bot \rightarrow \bot) = \top$, if $X \not\models G$, and $F^X = \bot$, otherwise.
- The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

$$\mathcal{F}^X = \{F^X \mid F \in \mathcal{F}\}.$$

Reduct

[21]

- The satisfaction relation $X \models F$ between a set X of atoms and a (set of) formula(s) F is defined as in propositional logic.
- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = \bot$ if $X \not\models F$
  - $F^X = F$ if $F \in X$
  - $F^X = (G^X \circ H^X)$ if $X \models F$ and $F = (G \circ H)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$

  ➥ If $F = \sim G = (G \rightarrow \bot)$, then $F^X = (\bot \rightarrow \bot) = \top$, if $X \not\models G$, and $F^X = \bot$, otherwise.
- The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

$$\mathcal{F}^X = \{F^X \mid F \in \mathcal{F}\}.$$

Reduct

[21]

- The satisfaction relation $X \models F$ between a set X of atoms and a (set of) formula(s) F is defined as in propositional logic.
- The reduct, F^X, of a formula F relative to a set X of atoms is defined recursively as follows:
  - $F^X = \bot$ if $X \not\models F$
  - $F^X = F$ if $F \in X$
  - $F^X = (G^X \circ H^X)$ if $X \models F$ and $F = (G \circ H)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$

  ➥ If $F = {\sim}G = (G \rightarrow \bot)$, then $F^X = (\bot \rightarrow \bot) = \top$, if $X \not\models G$, and $F^X = \bot$, otherwise.
- The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

$$\mathcal{F}^X = \{F^X \mid F \in \mathcal{F}\}.$$

Answer sets

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\mathcal{F}$ is denoted by $\min_{\subseteq}(\mathcal{F})$.
- A set X of atoms is an answer set of a propositional theory $\mathcal{F}$ if $X \in \min_{\subseteq}(\mathcal{F}^X)$.
- If X is an answer set of $\mathcal{F}$, then
  - $X \models \mathcal{F}$ and
  - $\min_{\subseteq}(\mathcal{F}^X) = \{X\}$.
    In general, this does not imply $X \in \min_{\subseteq}(\mathcal{F})$!

Answer sets

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\mathcal{F}$ is denoted by $\min_{\subseteq}(\mathcal{F})$.
- A set X of atoms is an answer set of a propositional theory $\mathcal{F}$ if $X \in \min_{\subseteq}(\mathcal{F}^X)$.
- If X is an answer set of $\mathcal{F}$, then
  - $X \models \mathcal{F}$ and
  - $\min_{\subseteq}(\mathcal{F}^X) = \{X\}$.
  - In general, this does not imply $X \in \min_{\subseteq}(\mathcal{F})$!

Answer sets

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\mathcal{F}$ is denoted by $\min_{\subseteq}(\mathcal{F})$.
- A set X of atoms is an answer set of a propositional theory $\mathcal{F}$ if $X \in \min_{\subseteq}(\mathcal{F}^X)$.
- If X is an answer set of $\mathcal{F}$, then
  - $X \models \mathcal{F}$ and
  - $\min_{\subseteq}(\mathcal{F}^X) = \{X\}$.

  ☞ In general, this does not imply $X \in \min_{\subseteq}(\mathcal{F})$!

Answer sets

- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\mathcal{F}$ is denoted by $\min_{\subseteq}(\mathcal{F})$.
- A set X of atoms is an answer set of a propositional theory $\mathcal{F}$ if $X \in \min_{\subseteq}(\mathcal{F}^X)$.
- If X is an answer set of $\mathcal{F}$, then
  - $X \models \mathcal{F}$ and
  - $\min_{\subseteq}(\mathcal{F}^X) = \{X\}$.

  ☞ In general, this does not imply $X \in \min_{\subseteq}(\mathcal{F})$!

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$.
    For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$.

$\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$

For $X = \emptyset$, we get
$\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$.
For $X = \{p\}$, we get
$\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
For $X = \{q, r\}$, we get
$\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$.

$\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$

- For $X = \emptyset$, we get
  $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$.
- For $X = \{p\}$, we get
  $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
- For $X = \{q, r\}$, we get
  $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$.

$\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$

For $X = \emptyset$, we get
$\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$.
For $X = \{p\}$, we get
$\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
For $X = \{q, r\}$, we get
$\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

$\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$

For $X = \emptyset$, we get
$\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$.
For $X = \{p\}$, we get
$\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
For $X = \{q, r\}$, we get
$\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$.
  - For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
  - For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$.
    For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
    For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✘
    For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
    For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✘
  - For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$.
    For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✘
  - For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$. ✘
  - For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✘
  - For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$. ✘
  - For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$.

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✘
  - For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$. ✘
  - For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$. ✔

Two examples

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\mathcal{F}_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\{p,q,r\}}) = \{\emptyset\}$. ✘
  - For $X = \emptyset$, we get
    $\mathcal{F}_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✔

- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\mathcal{F}_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✘
  - For $X = \{p\}$, we get
    $\mathcal{F}_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$. ✘
  - For $X = \{q, r\}$, we get
    $\mathcal{F}_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$. ✔

Relationship with logic programs

- The translation, $\tau[(F \leftarrow G)]$, of a (nested) rule $(F \leftarrow G)$ is defined recursively as follows:
  - $\tau[(F \leftarrow G)] = (\tau[G] \rightarrow \tau[F])$,
  - $\tau[\bot] = \bot$,
  - $\tau[\top] = \top$,
  - $\tau[F] = F$ if F is an atom,
  - $\tau[not\ F] = {\sim}\tau[F]$,
  - $\tau[(F, G)] = (\tau[F] \wedge \tau[G])$,
  - $\tau[(F; G)] = (\tau[F] \vee \tau[G])$.

The translation of a logic program Π is $\tau[\Pi] = \{\tau[r] \mid r \in \Pi\}$.

Given a logic program Π and a set X of atoms,
X is an answer set of Π iff X is an answer set of $\tau[\Pi]$.

Relationship with logic programs

- The translation, $\tau[(F \leftarrow G)]$, of a (nested) rule $(F \leftarrow G)$ is defined recursively as follows:
  - $\tau[(F \leftarrow G)] = (\tau[G] \rightarrow \tau[F])$,
  - $\tau[\bot] = \bot$,
  - $\tau[\top] = \top$,
  - $\tau[F] = F$ if F is an atom,
  - $\tau[not\ F] = {\sim}\tau[F]$,
  - $\tau[(F, G)] = (\tau[F] \wedge \tau[G])$,
  - $\tau[(F; G)] = (\tau[F] \vee \tau[G])$.

- The translation of a logic program Π is $\tau[\Pi] = \{\tau[r] \mid r \in \Pi\}$.

  Given a logic program Π and a set X of atoms,
  X is an answer set of Π iff X is an answer set of $\tau[\Pi]$.

Relationship with logic programs

- The translation, $\tau[(F \leftarrow G)]$, of a (nested) rule $(F \leftarrow G)$ is defined recursively as follows:
  - $\tau[(F \leftarrow G)] = (\tau[G] \rightarrow \tau[F])$,
  - $\tau[\bot] = \bot$,
  - $\tau[\top] = \top$,
  - $\tau[F] = F$ if F is an atom,
  - $\tau[not\ F] = {\sim}\tau[F]$,
  - $\tau[(F, G)] = (\tau[F] \wedge \tau[G])$,
  - $\tau[(F; G)] = (\tau[F] \vee \tau[G])$.

- The translation of a logic program Π is $\tau[\Pi] = \{\tau[r] \mid r \in \Pi\}$.
  - Given a logic program Π and a set X of atoms, X is an answer set of Π iff X is an answer set of $\tau[\Pi]$.

Relationship with logic programs

- The translation, $\tau[(F \leftarrow G)]$, of a (nested) rule $(F \leftarrow G)$ is defined recursively as follows:
  - $\tau[(F \leftarrow G)] = (\tau[G] \rightarrow \tau[F])$,
  - $\tau[\bot] = \bot$,
  - $\tau[\top] = \top$,
  - $\tau[F] = F$ if F is an atom,
  - $\tau[not\ F] = {\sim}\tau[F]$,
  - $\tau[(F, G)] = (\tau[F] \wedge \tau[G])$,
  - $\tau[(F; G)] = (\tau[F] \vee \tau[G])$.

- The translation of a logic program Π is $\tau[\Pi] = \{\tau[r] \mid r \in \Pi\}$.
  - ➥ Given a logic program Π and a set X of atoms, X is an answer set of Π iff X is an answer set of $\tau[\Pi]$.

Logic programs as propositional theories

- The normal logic program $\Pi = \{p \leftarrow not\ q,\ q \leftarrow not\ p\}$ corresponds to $\tau[\Pi] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$.
  - Answer sets: $\{p\}$ and $\{q\}$

The disjunctive logic program $\Pi = \{p\ ;\ q \leftarrow\}$ corresponds to $\tau[\Pi] = \{\top \rightarrow p \vee q\}$.

Answer sets: $\{p\}$ and $\{q\}$

The nested logic program $\Pi = \{p \leftarrow not\ not\ p\}$ corresponds to $\tau[\Pi] = \{\sim\sim p \rightarrow p\}$.

Answer sets: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $\Pi = \{p \leftarrow not\ q,\ q \leftarrow not\ p\}$ corresponds to $\tau[\Pi] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

The disjunctive logic program $\Pi = \{p\ ;\ q \leftarrow\}$ corresponds to $\tau[\Pi] = \{\top \rightarrow p \vee q\}$.

Answer sets: $\{p\}$ and $\{q\}$

The nested logic program $\Pi = \{p \leftarrow not\ not\ p\}$ corresponds to $\tau[\Pi] = \{\sim\sim p \rightarrow p\}$.

Answer sets: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $\Pi = \{p \leftarrow not\ q,\ q \leftarrow not\ p\}$ corresponds to $\tau[\Pi] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

- The disjunctive logic program $\Pi = \{p\ ;\ q \leftarrow\}$ corresponds to $\tau[\Pi] = \{\top \rightarrow p \vee q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

- The nested logic program $\Pi = \{p \leftarrow not\ not\ p\}$ corresponds to $\tau[\Pi] = \{\sim\sim p \rightarrow p\}$.
  - Answer sets: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $\Pi = \{p \leftarrow not\ q,\ q \leftarrow not\ p\}$ corresponds to $\tau[\Pi] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

- The disjunctive logic program $\Pi = \{p\ ;\ q \leftarrow\}$ corresponds to $\tau[\Pi] = \{\top \rightarrow p \vee q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

- The nested logic program $\Pi = \{p \leftarrow not\ not\ p\}$ corresponds to $\tau[\Pi] = \{\sim\sim p \rightarrow p\}$.
  - Answer sets: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $\Pi = \{p \leftarrow not\ q,\ q \leftarrow not\ p\}$ corresponds to $\tau[\Pi] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$
- The disjunctive logic program $\Pi = \{p\ ;\ q \leftarrow\}$ corresponds to $\tau[\Pi] = \{\top \rightarrow p \vee q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$
- The nested logic program $\Pi = \{p \leftarrow not\ not\ p\}$ corresponds to $\tau[\Pi] = \{\sim\sim p \rightarrow p\}$.
  - ➥ Answer sets: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $\Pi = \{p \leftarrow not\ q,\ q \leftarrow not\ p\}$ corresponds to $\tau[\Pi] = \{\sim q \rightarrow p,\ \sim p \rightarrow q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

- The disjunctive logic program $\Pi = \{p\ ;\ q \leftarrow\}$ corresponds to $\tau[\Pi] = \{\top \rightarrow p \vee q\}$.
  - ➥ Answer sets: $\{p\}$ and $\{q\}$

- The nested logic program $\Pi = \{p \leftarrow not\ not\ p\}$ corresponds to $\tau[\Pi] = \{\sim\sim p \rightarrow p\}$.
  - ➥ Answer sets: $\emptyset$ and $\{p\}$

Classical Negation
Overview

25 Syntax

26 Semantics

27 Examples

Syntax

Status quo

- In logic programs *not* (or $\sim$) denotes default negation.

Generalization

- We allow classical negation for atoms (only!).
  - ➥ Logic programs in "negation normal form."
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg A \mid A \in \mathcal{A}\}$.
  - ☞ We assume $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$.
- The atoms A and $\neg A$ are complementary.
  - ➥ $\neg A$ is the classical negation of A, and vice versa.

Syntax

Status quo

- In logic programs *not* (or $\sim$) denotes default negation.

Generalization

- We allow classical negation for atoms (only!).
  - ➥ Logic programs in "negation normal form."
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg A \mid A \in \mathcal{A}\}$.
  - ☞ We assume $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$.
- The atoms A and $\neg A$ are complementary.
  - ➥ $\neg A$ is the classical negation of A, and vice versa.

Syntax

Status quo

- In logic programs *not* (or $\sim$) denotes default negation.

Generalization

- We allow classical negation for atoms (only!).
  - ➥ Logic programs in "negation normal form."
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{\neg A \mid A \in \mathcal{A}\}$.
  - ☞ We assume $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$.
- The atoms A and $\neg A$ are complementary.
  - ➥ $\neg A$ is the classical negation of A, and vice versa.

Semantics

- A set X of atoms is consistent, if $X \cap \{\neg A \mid A \in (\mathcal{A} \cap X)\} = \emptyset$, and inconsistent, otherwise.
- A set X of atoms is an answer set of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ if X is an answer set of $\Pi \cup \{B \leftarrow A, \neg A \mid A \in \mathcal{A}, B \in (\mathcal{A} \cup \overline{\mathcal{A}})\}$
  - ➥ The only inconsistent answer set (candidate) is $X = \mathcal{A} \cup \overline{\mathcal{A}}$.
- For a normal or disjunctive logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All answer sets of Π are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only answer set of Π.

Semantics

- A set X of atoms is consistent, if $X \cap \{\neg A \mid A \in (\mathcal{A} \cap X)\} = \emptyset$, and inconsistent, otherwise.
- A set X of atoms is an answer set of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ if X is an answer set of $\Pi \cup \{B \leftarrow A, \neg A \mid A \in \mathcal{A}, B \in (\mathcal{A} \cup \overline{\mathcal{A}})\}$
  - ➥ The only inconsistent answer set (candidate) is $X = \mathcal{A} \cup \overline{\mathcal{A}}$.
- For a normal or disjunctive logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All answer sets of Π are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only answer set of Π.

Semantics

- A set X of atoms is consistent, if $X \cap \{\neg A \mid A \in (\mathcal{A} \cap X)\} = \emptyset$, and inconsistent, otherwise.
- A set X of atoms is an answer set of a logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$ if X is an answer set of $\Pi \cup \{B \leftarrow A, \neg A \mid A \in \mathcal{A}, B \in (\mathcal{A} \cup \overline{\mathcal{A}})\}$
  - ➥ The only inconsistent answer set (candidate) is $X = \mathcal{A} \cup \overline{\mathcal{A}}$.
- For a normal or disjunctive logic program Π over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All answer sets of Π are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only answer set of Π.

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

To cross or not to cross. . . ?

- $\Pi_1 = \{cross \leftarrow not\ train\}$
  - Answer set: $\{cross\}$
- $\Pi_2 = \{cross \leftarrow \neg train\}$
  - Answer set: $\emptyset$
- $\Pi_3 = \{cross \leftarrow \neg train,\ \neg train \leftarrow\}$
  - Answer set: $\{cross, \neg train\}$
- $\Pi_4 = \{cross \leftarrow \neg train,\ \neg train \leftarrow,\ \neg cross \leftarrow\}$
  - Answer set: $\{cross, \neg cross, train, \neg train\}$
- $\Pi_5 = \{cross \leftarrow \neg train,\ \neg train \leftarrow not\ train,\ \neg cross \leftarrow\}$
  - No answer set

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$

  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$
  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$
  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$

  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ; q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{q\}$
- $\Pi = \{p\ ; \mathit{not}\ p \leftarrow \top,\ \neg p\ ; \mathit{not}\ q \leftarrow \top,\ q\ ; \mathit{not}\ q \leftarrow \top\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$
  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$

  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$

  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$

  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$

  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Example

- $\Pi = \{p \leftarrow,\ \neg p \leftarrow,\ q \leftarrow \mathit{not}\ r\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{p, \neg p, q, \neg q, r, \neg r\}$
- $\Pi = \{p\ ;\ q \leftarrow,\ r \leftarrow p,\ \neg r \leftarrow p\ \}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q, r\}\}$
  Answer set: $\{q\}$
- $\Pi = \{p\ ;\ \mathit{not}\ p \leftarrow \top,\ \neg p\ ;\ \mathit{not}\ q \leftarrow \top,\ q\ ;\ \mathit{not}\ q \leftarrow \top\}$
  $\Pi' = \Pi \cup \{A \leftarrow (B, \neg B),\ \neg A \leftarrow (B, \neg B) \mid A, B \in \{p, q\}\}$
  Answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$, $\{\neg p, q\}$, and $\{p, \neg p, q, \neg q\}$

Complexity

[13]

Let A be an atom and X be a set of atoms.

- For a positive normal logic program Π:
  - Deciding whether X is the answer set of Π is **P**-complete.
  - Deciding whether A is in the answer set of Π is **P**-complete.
- For a normal logic program Π:
  - Deciding whether X is an answer set of Π is **P**-complete.
  - Deciding whether A is in an answer set of Π is **NP**-complete.

Complexity

[13]

Let A be an atom and X be a set of atoms.

- For a positive normal logic program Π:
  - Deciding whether X is the answer set of Π is **P**-complete.
  - Deciding whether A is in the answer set of Π is **P**-complete.
- For a normal logic program Π:
  - Deciding whether X is an answer set of Π is **P**-complete.
  - Deciding whether A is in an answer set of Π is **NP**-complete.

Complexity

[13]

Let A be an atom and X be a set of atoms.

- For a positive normal logic program Π:
  - Deciding whether X is the answer set of Π is **P**-complete.
  - Deciding whether A is in the answer set of Π is **P**-complete.
- For a normal logic program Π:
  - Deciding whether X is an answer set of Π is **P**-complete.
  - Deciding whether A is in an answer set of Π is **NP**-complete.

Complexity

[13]

Let A be an atom and X be a set of atoms.

- For a positive normal logic program Π:
  - Deciding whether X is the answer set of Π is **P**-complete.
  - Deciding whether A is in the answer set of Π is **P**-complete.
- For a normal logic program Π:
  - Deciding whether X is an answer set of Π is **P**-complete.
  - Deciding whether A is in an answer set of Π is **NP**-complete.

Complexity

[13]

Let A be an atom and X be a set of atoms.

- For a positive normal logic program Π:
  - Deciding whether X is the answer set of Π is **P**-complete.
  - Deciding whether A is in the answer set of Π is **P**-complete.
- For a normal logic program Π:
  - Deciding whether X is an answer set of Π is **P**-complete.
  - Deciding whether A is in an answer set of Π is **NP**-complete.

Complexity (ctd)

- For a positive disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a nested logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a propositional theory $\mathcal{F}$:
  - Deciding whether X is an answer set of $\mathcal{F}$ is **co-NP**-complete.
  - Deciding whether A is in an answer set of $\mathcal{F}$ is $\mathbf{NP}^{\mathbf{NP}}$-complete.

Complexity (ctd)

- For a positive disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a nested logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a propositional theory $\mathcal{F}$:
  - Deciding whether X is an answer set of $\mathcal{F}$ is **co-NP**-complete.
  - Deciding whether A is in an answer set of $\mathcal{F}$ is $\mathbf{NP}^{\mathbf{NP}}$-complete.

Complexity (ctd)

- For a positive disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a nested logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP}^{\mathbf{NP}}$-complete.
- For a propositional theory $\mathcal{F}$:
  - Deciding whether X is an answer set of $\mathcal{F}$ is **co-NP**-complete.
  - Deciding whether A is in an answer set of $\mathcal{F}$ is $\mathbf{NP}^{\mathbf{NP}}$-complete.

Complexity (ctd)

- For a positive disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP^{NP}}$-complete.
- For a disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP^{NP}}$-complete.
- For a nested logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP^{NP}}$-complete.
- For a propositional theory $\mathcal{F}$:
  - Deciding whether X is an answer set of $\mathcal{F}$ is **co-NP**-complete.
  - Deciding whether A is in an answer set of $\mathcal{F}$ is $\mathbf{NP^{NP}}$-complete.

Complexity (ctd)

- For a positive disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP^{NP}}$-complete.
- For a disjunctive logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP^{NP}}$-complete.
- For a nested logic program Π:
  - Deciding whether X is an answer set of Π is **co-NP**-complete.
  - Deciding whether A is in an answer set of Π is $\mathbf{NP^{NP}}$-complete.
- For a propositional theory $\mathcal{F}$:
  - Deciding whether X is an answer set of $\mathcal{F}$ is **co-NP**-complete.
  - Deciding whether A is in an answer set of $\mathcal{F}$ is $\mathbf{NP^{NP}}$-complete.

Language Extensions
Overview

28 Motivation

29 Choice Rules

30 Cardinality Constraints

31 Cardinality Rules

32 Weight Constraints (and more)

33 Semantics by Embedding

34 Modeling Practice

Language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, eg. classical negation.
- This translation might also be used for implementing the language extension. When is this feasible?

Language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, eg. classical negation.
- This translation might also be used for implementing the language extension. When is this feasible?

Language extensions

- The expressiveness of a language can be enhanced by introducing new constructs.
- To this end, we must address the following issues:
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- A way of providing semantics is to furnish a translation removing the new constructs, eg. classical negation.
- This translation might also be used for implementing the language extension. When is this feasible?

Choice rules

[73]

Idea Choices over subsets.

Syntax

$$\{A_1, \ldots, A_m\} \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o,$$

Informal meaning If the body is satisfied in an answer set, then any subset of $\{A_1, \ldots, A_m\}$ can be included in the answer set.

Example The program $\Pi = \{\ \{a\} \leftarrow b,\ \ b \leftarrow\}$ has two answer sets: $\{b\}$ and $\{a, b\}$.

Implementation lparse/gringo + smodels/cmodels/nomore/clasp

Embedding in normal logic programs

- A choice rule of form

$$\{A_1, \ldots, A_m\} \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o$$

can be translated into $2m + 1$ rules

$$\begin{array}{rcl} A & \leftarrow & A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o \\ A_1 & \leftarrow & A, \mathit{not}\ \overline{A_1} \\ \overline{A_1} & \leftarrow & \mathit{not}\ A_1 \end{array} \quad \ldots \quad \begin{array}{rcl} A_m & \leftarrow & A, \mathit{not}\ \overline{A_m} \\ \overline{A_m} & \leftarrow & \mathit{not}\ A_m \end{array}$$

by introducing new atoms $A, \overline{A_1}, \ldots, \overline{A_m}$.

Cardinality constraints

Syntax A (positive) cardinality constraint is of the form $l\ \{A_1, \ldots, A_m\}\ u$

Informal meaning A cardinality constraint is satisfied in an answer set X, if the number of atoms from $\{A_1, \ldots, A_m\}$ satisfied in X is between l and u (inclusive).
More formally, if $l \leq |\{A_1, \ldots, A_m\} \cap X| \leq u$.

Conditions $l\ \{A_1 : B_1, \ldots, A_m : B_m\}\ u$
where $B_1, \ldots, B_m$ are used for restricting instantiations of variables occurring in $A_1, \ldots, A_m$.

Example 2 {hd(a),. . . ,hd(m)} 4

Implementation lparse/gringo + smodels/cmodels/nomore/clasp

n-colorability revisited

with $n = 3$

C(I)	vertex(1) ← edge(1,2) ← vertex(2) ← edge(2,3) ← vertex(3) ← edge(3,1) ←
C(P)	color(r) ← color(b) ← color(g) ← 1 {colored(V,C) : color(C)} 1 ← vertex(V) ← edge(V,U),color(C), colored(V,C),colored(U,C)
Answer set	{ colored(1,r), colored(2,b), colored(3,g), . . . }

Cardinality rules

[73]

Idea Control cardinality of subsets.

Syntax

$$A_0 \leftarrow l\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\}$$

Informal meaning If at least l elements of the "body" are true in an answer set, then add A_0 to the answer set.

➥ l is a lower bound on the "body"

Example The program $\Pi = \{\ a \leftarrow 1\{b, c\},\ b \leftarrow\}$ has one answer set: $\{a, b\}$.

Implementation lparse/gringo + smodels/cmodels/nomore/clasp

☞ gringo distinguishes sets and multi-sets!

Embedding in normal logic programs (ctd)

- Replace each cardinality rule

$$A_0 \leftarrow l\ \{A_1, \ldots, A_m\} \qquad \text{by} \qquad A_0 \leftarrow cc(A_1, l)$$

where atom $cc(A_i, j)$ represents the fact that at least j of the atoms in $\{A_i, \ldots, A_m\}$, that is, of the atoms that have an equal or greater index than i, are in a particular answer set.

- The definition of $cc(A_i, j)$ is given by the rules

$$\begin{aligned} cc(A_i, j+1) &\leftarrow cc(A_{i+1}, j), A_i \\ cc(A_i, j) &\leftarrow cc(A_{i+1}, j) \\ cc(A_{m+1}, 0) &\leftarrow \end{aligned}$$

- What about space complexity?

Embedding in normal logic programs (ctd)

- Replace each cardinality rule

$$A_0 \leftarrow l\ \{A_1, \ldots, A_m\} \qquad \text{by} \qquad A_0 \leftarrow cc(A_1, l)$$

where atom $cc(A_i, j)$ represents the fact that at least j of the atoms in $\{A_i, \ldots, A_m\}$, that is, of the atoms that have an equal or greater index than i, are in a particular answer set.

- The definition of $cc(A_i, j)$ is given by the rules

$$\begin{aligned} cc(A_i, j+1) &\leftarrow cc(A_{i+1}, j), A_i \\ cc(A_i, j) &\leftarrow cc(A_{i+1}, j) \\ cc(A_{m+1}, 0) &\leftarrow \end{aligned}$$

- What about space complexity?

. . . and vice versa

- A normal rule

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n,$$

can be represented by the cardinality rule

$$A_0 \leftarrow n + m\ \{A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\}.$$

Cardinality rules with upper bounds

- A rule of the form

$$A_0 \leftarrow l\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\}\ u$$

stands for

$$\begin{aligned} A_0 &\leftarrow B, not\ C \\ B &\leftarrow l\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\} \\ C &\leftarrow u+1\ \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\} \end{aligned}$$

Cardinality constraints as heads

- A rule of the form

$$I\ \{A_1, \ldots, A_m\}\ u \leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o,$$

stands for

$$\begin{aligned} B &\leftarrow A_{m+1}, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_o \\ \{A_1, \ldots, A_m\} &\leftarrow B \\ C &\leftarrow I\ \{A_1, \ldots, A_m\}\ u \\ &\leftarrow B, \mathit{not}\ C \end{aligned}$$

Full-fledged cardinality rules

- A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \ldots, l_n\ S_n\ u_n$$

stands for $0 \leq i \leq n$

$$\begin{aligned} B_i &\leftarrow l_i\ S_i \\ C_i &\leftarrow u_i + 1\ S_i \\ A &\leftarrow B_1, \ldots, B_n, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \\ &\leftarrow A, \mathit{not}\ B_0 \\ &\leftarrow A, C_0 \\ S_0 \cap \mathcal{A} &\leftarrow A \end{aligned}$$

where $\mathcal{A}$ is the underlying alphabet.

Full-fledged cardinality rules

- A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \ldots, l_n\ S_n\ u_n$$

stands for $0 \leq i \leq n$

$$\begin{aligned} B_i &\leftarrow l_i\ S_i \\ C_i &\leftarrow u_i + 1\ S_i \\ A &\leftarrow B_1, \ldots, B_n, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \\ &\leftarrow A, \mathit{not}\ B_0 \\ &\leftarrow A, C_0 \\ S_0 \cap \mathcal{A} &\leftarrow A \end{aligned}$$

where $\mathcal{A}$ is the underlying alphabet.

Weight constraints

[73]

Syntax $l\ [A_1 = w_1, \ldots, A_m = w_m,$
$not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n]\ u$

Informal meaning A weight constraint is satisfied in an answer set X, if

$$l \leq \left(\sum_{1 \leq i \leq m, A_i \in X} w_i + \sum_{m < i \leq n, A_i \notin X} w_i \right) \leq u\ .$$

➥ Generalization of cardinality constraints.

Example 80 [hd(a)=50,. . . ,hd(m)=100] 400

Implementation lparse/gringo + smodels/cmodels/nomore/clasp

☞ gringo distinguishes sets and multi-sets!

Optimization statements

[73]

Idea Compute optimal answer sets by minimizing or maximizing a weighted sum of given atoms, respectively.

Syntax

- $\mathit{minimize}\ [A_1 = w_1, \ldots, A_m = w_m, \mathit{not}\ A_{m+1} = w_{m+1}, \ldots, \mathit{not}\ A_n = w_n]$
- $\mathit{maximize}\ [A_1 = w_1, \ldots, A_m = w_m, \mathit{not}\ A_{m+1} = w_{m+1}, \ldots, \mathit{not}\ A_n = w_n]$

Several optimization statements are interpreted lexicographically.

Example

- minimize [hd(a)=30,. . . ,hd(m)=50]
- minimize [road(X,Y) : length(X,Y,L) = L]

Implementation lparse/gringo + smodels/cmodels/nomore/clasp

Weak integrity constraints

[50]

Syntax $:\sim A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n\ [w : l]$

Informal meaning

1. minimize the sum of weights of violated constraints in the highest level;
2. minimize the sum of weights of violated constraints in the next lower level;
3. etc

Implementation dlv

Embedding in nested logic programs

[22]

For formulas $F, F_1, \ldots, F_n$, define

- $\leftarrow F$ as $\bot \leftarrow F$
- $\{F_1 \ldots, F_n\}^c$ as $(F_1 \,;\, not\ F_1), \ldots, (F_n \,;\, not\ F_n)$
  - ↪ corresponds to choice over subsets (see above)

For example,

$$\Pi = \left\{ \begin{array}{lll} \{p,q\}^c & \leftarrow & \\ & \leftarrow & p,q \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p \,;\, not\ p), (q \,;\, not\ q) & \leftarrow & \\ & \bot \ \leftarrow & p,q \end{array} \right\}$$

Embedding in nested logic programs

[22]

For formulas $F, F_1, \ldots, F_n$, define

- $\leftarrow F$ as $\bot \leftarrow F$
- $\{F_1 \ldots, F_n\}^c$ as $(F_1 \,;\, not\ F_1), \ldots, (F_n \,;\, not\ F_n)$
  - ↪ corresponds to choice over subsets (see above)

For example,

$$\Pi = \left\{ \begin{array}{lll} \{p,q\}^c & \leftarrow & \\ & \leftarrow & p,q \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p \,;\, not\ p), (q \,;\, not\ q) & \leftarrow & \\ & \bot \leftarrow & p,q \end{array} \right\}$$

Embedding in nested logic programs

[22]

For formulas $F, F_1, \ldots, F_n$, define

- $\leftarrow F$ as $\bot \leftarrow F$
- $\{F_1 \ldots, F_n\}^c$ as $(F_1 \,;\, not\ F_1), \ldots, (F_n \,;\, not\ F_n)$
  - ↪ corresponds to choice over subsets (see above)

For example,

$$\Pi = \left\{ \begin{array}{lll} \{p,q\}^c & \leftarrow & \\ & \leftarrow & p,q \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p \,;\, not\ p), (q \,;\, not\ q) & \leftarrow & \\ & \bot \leftarrow & p,q \end{array} \right\}$$

Embedding in nested logic programs (ctd)

For formulas $F_1, \ldots, F_n$, define

- $l\ \{F_1, \ldots, F_n\}$ as $\bigvee_{1 \leq i_1 < \cdots < i_l \leq n} (F_{i_1}, \ldots, F_{i_l})$
- $\{F_1, \ldots, F_n\}\ u$ as $not\ (u + 1\ \{F_1, \ldots, F_n\})$
- $l\ \{F_1, \ldots, F_n\}\ u$ as $(l\ \{F_1, \ldots, F_n\}),\ (\{F_1, \ldots, F_n\}\ u)$

Embedding in nested logic programs (ctd)

For formulas $F_1, \ldots, F_n$, define

- $l\ \{F_1, \ldots, F_n\}$ as $\bigvee_{1 \leq i_1 < \cdots < i_l \leq n} (F_{i_1}, \ldots, F_{i_l})$
- $\{F_1, \ldots, F_n\}\ u$ as $not\ (u + 1\ \{F_1, \ldots, F_n\})$
- $l\ \{F_1, \ldots, F_n\}\ u$ as $(l\ \{F_1, \ldots, F_n\}), (\{F_1, \ldots, F_n\}\ u)$

Embedding in nested logic programs (ctd)

For formulas $F_1, \ldots, F_n$, define

- $l\ \{F_1, \ldots, F_n\}$ as $\bigvee_{1 \leq i_1 < \cdots < i_l \leq n} (F_{i_1}, \ldots, F_{i_l})$
- $\{F_1, \ldots, F_n\}\ u$ as $not\ (u + 1\ \{F_1, \ldots, F_n\})$
- $l\ \{F_1, \ldots, F_n\}\ u$ as $(l\ \{F_1, \ldots, F_n\}),\ (\{F_1, \ldots, F_n\}\ u)$

An example

$$\Pi = \left\{ \begin{array}{lll} \{p,q,r\}^c & \leftarrow & \\ & \leftarrow & 2\ \{p,q,r\} \\ & \leftarrow & \{p,q,r\}\ 0 \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p\ ;not\ p),(q\ ;not\ q),(r\ ;not\ r) & \leftarrow & \\ \bot & \leftarrow & (p,q)\ ;(p,r)\ ;(q,r) \\ \bot & \leftarrow & not\ (p\ ;q\ ;r) \end{array} \right\}$$

Answer sets ? $\{p\}$, $\{q\}$, and $\{r\}$!

An example

$$\Pi = \left\{ \begin{array}{lll} \{p,q,r\}^c & \leftarrow & \\ & \leftarrow & 2\ \{p,q,r\} \\ & \leftarrow & \{p,q,r\}\ 0 \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p\ ;not\ p),(q\ ;not\ q),(r\ ;not\ r) & \leftarrow & \\ \bot & \leftarrow & (p,q)\ ;(p,r)\ ;(q,r) \\ \bot & \leftarrow & not\ (p\ ;q\ ;r) \end{array} \right\}$$

Answer sets ? $\{p\}$, $\{q\}$, and $\{r\}$!

An example

$$\Pi = \left\{ \begin{array}{lll} \{p,q,r\}^c & \leftarrow & \\ & \leftarrow & 2\ \{p,q,r\} \\ & \leftarrow & \{p,q,r\}\ 0 \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p\ ;not\ p),(q\ ;not\ q),(r\ ;not\ r) & \leftarrow & \\ \bot & \leftarrow & (p,q)\ ;(p,r)\ ;(q,r) \\ \bot & \leftarrow & not\ (p\ ;q\ ;r) \end{array} \right\}$$

Answer sets ? $\{p\}$, $\{q\}$, and $\{r\}$!

An example

$$\Pi = \left\{ \begin{array}{lll} \{p,q,r\}^c & \leftarrow & \\ & \leftarrow & 2\ \{p,q,r\} \\ & \leftarrow & \{p,q,r\}\ 0 \end{array} \right\} \text{ stands for}$$

$$\left\{ \begin{array}{lll} (p\ ;not\ p),(q\ ;not\ q),(r\ ;not\ r) & \leftarrow & \\ \bot & \leftarrow & (p,q)\ ;(p,r)\ ;(q,r) \\ \bot & \leftarrow & not\ (p\ ;q\ ;r) \end{array} \right\}$$

Answer sets ? $\{p\}$, $\{q\}$, and $\{r\}$!

Embedding in nested logic programs (ctd)

For formulas $F_1, \ldots, F_n$, define

- $l\ \{F_1, \ldots, F_n\}^c$ as $(\{F_1, \ldots, F_n\}^c, l\ \{F_1, \ldots, F_n\}\ )$
- $\{F_1, \ldots, F_n\}^c\ u$ as $(\{F_1, \ldots, F_n\}^c,\ \{F_1, \ldots, F_n\}\ u)$
- $l\ \{F_1, \ldots, F_n\}^c\ u$ as $(\{F_1, \ldots, F_n\}^c, l\ \{F_1, \ldots, F_n\}\ u)$
  - ➥ corresponds to cardinality constraints (see above)

The program

$$1\ \{p, q, r\}^c\ 1 \leftarrow$$

has the same answer sets as the previous one.

Embedding in nested logic programs (ctd)

For formulas $F_1, \ldots, F_n$, define

- $I\ \{F_1, \ldots, F_n\}^c$ as $(\{F_1, \ldots, F_n\}^c, I\ \{F_1, \ldots, F_n\}\ \ )$
- $\ \ \{F_1, \ldots, F_n\}^c\ u$ as $(\{F_1, \ldots, F_n\}^c,\ \ \{F_1, \ldots, F_n\}\ u)$
- $I\ \{F_1, \ldots, F_n\}^c\ u$ as $(\{F_1, \ldots, F_n\}^c, I\ \{F_1, \ldots, F_n\}\ u)$
  - ➥ corresponds to cardinality constraints (see above)

The program

$$1\ \{p, q, r\}^c\ 1 \leftarrow$$

has the same answer sets as the previous one.

Embedding in nested logic programs (ctd)

For formulas $F_1, \ldots, F_n$, define

- $l\ \{F_1, \ldots, F_n\}^c$ as $(\{F_1, \ldots, F_n\}^c, l\ \{F_1, \ldots, F_n\}\ )$
- $\{F_1, \ldots, F_n\}^c\ u$ as $(\{F_1, \ldots, F_n\}^c,\ \{F_1, \ldots, F_n\}\ u)$
- $l\ \{F_1, \ldots, F_n\}^c\ u$ as $(\{F_1, \ldots, F_n\}^c, l\ \{F_1, \ldots, F_n\}\ u)$
  - ➥ corresponds to cardinality constraints (see above)

The program

$$1\ \{p, q, r\}^c\ 1 \leftarrow$$

has the same answer sets as the previous one.

A closer look

$1\ \{p, q, r\}\ 2 \leftarrow$ versus $1\ \{p, q, r\}^c\ 2 \leftarrow$

$$
\begin{aligned}
&1\ \{p, q, r\}\ 2 \leftarrow \\
&\quad = \quad (1\ \{p, q, r\}), (\{p, q, r\}\ 2) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), not\ (3\ \{p, q, r\}) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), not\ (\ p, q, r\ ) \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&1\ \{p, q, r\}^c\ 2 \leftarrow \\
&\quad = \quad 1\ \{p, q, r\}\ 2,\ \{p, q, r\}^c \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), not\ (\ p, q, r\ ), ((p\ ;not\ p), (q\ ;not\ q), (r\ ;not\ r)) \leftarrow
\end{aligned}
$$

A closer look

$1\ \{p, q, r\}\ 2 \leftarrow$ versus $1\ \{p, q, r\}^c\ 2 \leftarrow$

$$
\begin{aligned}
&1\ \{p, q, r\}\ 2 \leftarrow \\
&\quad = \quad (1\ \{p, q, r\}), (\{p, q, r\}\ 2) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \mathit{not}\ (3\ \{p, q, r\}) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \mathit{not}\ (\ p, q, r\ ) \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&1\ \{p, q, r\}^c\ 2 \leftarrow \\
&\quad = \quad 1\ \{p, q, r\}\ 2,\ \{p, q, r\}^c \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \mathit{not}\ (\ p, q, r\ ), ((p\ ;\mathit{not}\ p), (q\ ;\mathit{not}\ q), (r\ ;\mathit{not}\ r)) \leftarrow
\end{aligned}
$$

A closer look

$1\ \{p, q, r\}\ 2 \leftarrow$ versus $1\ \{p, q, r\}^c\ 2 \leftarrow$

$$
\begin{aligned}
&1\ \{p, q, r\}\ 2 \leftarrow \\
&\quad = \quad (1\ \{p, q, r\}), (\{p, q, r\}\ 2) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \mathit{not}\ (3\ \{p, q, r\}) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \mathit{not}\ (\ p, q, r\ ) \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&1\ \{p, q, r\}^c\ 2 \leftarrow \\
&\quad = \quad 1\ \{p, q, r\}\ 2,\ \{p, q, r\}^c \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \mathit{not}\ (\ p, q, r\ ), ((p\ ;\mathit{not}\ p), (q\ ;\mathit{not}\ q), (r\ ;\mathit{not}\ r)) \leftarrow
\end{aligned}
$$

A closer look (ctd)

$$
\begin{aligned}
&(1\ \{p,q,r\}\ 2 \leftarrow)^X \\
&\quad = \quad (1\ \{p,q,r\}\ 2)^X \leftarrow \\
&\quad = \quad (p\ ;q\ ;r)^X, (\mathit{not}\ (\ p,q,r\ ))^X \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (\mathit{not}\ (\ p,q,r\ ))^X \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&(1\ \{p,q,r\}^c\ 2 \leftarrow)^X \\
&\quad = \quad (1\ \{p,q,r\}^c\ 2)^X \leftarrow \\
&\quad = \quad (1\ \{p,q,r\}\ 2,\ \{p,q,r\}^c)^X \leftarrow \\
&\quad = \quad (1\ \{p,q,r\}\ 2)^X,\ (\{p,q,r\}^c)^X \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (\mathit{not}\ (\ p,q,r\ ))^X, \\
&\qquad\quad ((p\ ;(\mathit{not}\ p)^X), (q\ ;(\mathit{not}\ q)^X), (r\ ;(\mathit{not}\ r)^X)) \leftarrow
\end{aligned}
$$

A closer look (ctd)

$$
\begin{aligned}
&(1\ \{p,q,r\}\ 2 \leftarrow)^X \\
&\quad = \quad (1\ \{p,q,r\}\ 2)^X \leftarrow \\
&\quad = \quad (p\ ;q\ ;r)^X, (not\ (\ p,q,r\ ))^X \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p,q,r\ ))^X \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&(1\ \{p,q,r\}^c\ 2 \leftarrow)^X \\
&\quad = \quad (1\ \{p,q,r\}^c\ 2)^X \leftarrow \\
&\quad = \quad (1\ \{p,q,r\}\ 2,\ \{p,q,r\}^c)^X \leftarrow \\
&\quad = \quad (1\ \{p,q,r\}\ 2)^X,\ (\{p,q,r\}^c)^X \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p,q,r\ ))^X, \\
&\qquad\quad ((p\ ;(not\ p)^X), (q\ ;(not\ q)^X), (r\ ;(not\ r)^X)) \leftarrow
\end{aligned}
$$

A closer look (ctd)

Consider $X = \{p, q\}$

$$
\begin{aligned}
&(1\ \{p, q, r\}\ 2 \leftarrow)^{\{p,q\}} \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p, q, r\ ))^{\{p,q\}} \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \top \leftarrow \\
&\quad = \quad (p\ ;q\ ;r) \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&(1\ \{p, q, r\}^c\ 2 \leftarrow)^{\{p,q\}} \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p, q, r\ ))^{\{p,q\}}, \\
&\qquad ((p\ ;(not\ p)^{\{p,q\}}), (q\ ;(not\ q)^{\{p,q\}}), (r\ ;(not\ r)^{\{p,q\}})) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \top, ((p\ ;\bot), (q\ ;\bot), (r\ ;\top)) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (p, q) \leftarrow \\
&\quad = \quad (p, q) \leftarrow
\end{aligned}
$$

A closer look (ctd)

Consider $X = \{p, q\}$

$$
\begin{aligned}
&(1\ \{p,q,r\}\ 2 \leftarrow)^{\{p,q\}} \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p,q,r\ ))^{\{p,q\}} \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \top \leftarrow \\
&\quad = \quad (p\ ;q\ ;r) \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&(1\ \{p,q,r\}^c\ 2 \leftarrow)^{\{p,q\}} \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p,q,r\ ))^{\{p,q\}}, \\
&\qquad\quad ((p\ ;(not\ p)^{\{p,q\}}), (q\ ;(not\ q)^{\{p,q\}}), (r\ ;(not\ r)^{\{p,q\}})) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \top, ((p\ ;\bot), (q\ ;\bot), (r\ ;\top)) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (p,q) \leftarrow \\
&\quad = \quad (p,q) \leftarrow
\end{aligned}
$$

A closer look (ctd)

Consider $X = \{p, q\}$

$$
\begin{aligned}
&(1\ \{p,q,r\}\ 2 \leftarrow)^{\{p,q\}} \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p,q,r\ ))^{\{p,q\}} \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \top \leftarrow \\
&\quad = \quad (p\ ;q\ ;r) \leftarrow
\end{aligned}
$$

$$
\begin{aligned}
&(1\ \{p,q,r\}^c\ 2 \leftarrow)^{\{p,q\}} \\
&\quad = \quad (p\ ;q\ ;r), (not\ (\ p,q,r\ ))^{\{p,q\}}, \\
&\qquad\quad ((p\ ;(not\ p)^{\{p,q\}}), (q\ ;(not\ q)^{\{p,q\}}), (r\ ;(not\ r)^{\{p,q\}})) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), \top, ((p\ ;\bot), (q\ ;\bot), (r\ ;\top)) \leftarrow \\
&\quad = \quad (p\ ;q\ ;r), (p,q) \leftarrow \\
&\quad = \quad (p,q) \leftarrow
\end{aligned}
$$

Usage in `lparse` and `gringo`

In `lparse` and `gringo`,

- rule bodies may include
  - $l\ \{F_1, \ldots, F_n\}$
  - $\{F_1, \ldots, F_n\}\ u$
  - $l\ \{F_1, \ldots, F_n\}\ u$
- rule heads may include
  - $\{F_1, \ldots, F_n\}^c$
  - $l\ \{F_1, \ldots, F_n\}^c$
  - $\{F_1, \ldots, F_n\}^c\ u$
  - $l\ \{F_1, \ldots, F_n\}^c\ u$
  - but dropping superscript ^{c}
- For instance, '`{p,q,r} :- {p,q,r} 2`' stands for $\{p, q, r\}^c \leftarrow \{p, q, r\}\ 2$.

Usage in `lparse` and `gringo`

In `lparse` and `gringo`,

- rule bodies may include
  - $l\ \{F_1, \ldots, F_n\}$
  - $\{F_1, \ldots, F_n\}\ u$
  - $l\ \{F_1, \ldots, F_n\}\ u$
- rule heads may include
  - $\{F_1, \ldots, F_n\}^c$
  - $l\ \{F_1, \ldots, F_n\}^c$
  - $\{F_1, \ldots, F_n\}^c\ u$
  - $l\ \{F_1, \ldots, F_n\}^c\ u$
  - but dropping superscript ^{c}
- For instance, '`{p,q,r} :- {p,q,r} 2`' stands for $\{p, q, r\}^c \leftarrow \{p, q, r\}\ 2$.

Usage in `lparse` and `gringo`

In `lparse` and `gringo`,

- rule bodies may include
  - $l\ \{F_1, \ldots, F_n\}$
  - $\{F_1, \ldots, F_n\}\ u$
  - $l\ \{F_1, \ldots, F_n\}\ u$
- rule heads may include
  - $\{F_1, \ldots, F_n\}^c$
  - $l\ \{F_1, \ldots, F_n\}^c$
  - $\{F_1, \ldots, F_n\}^c\ u$
  - $l\ \{F_1, \ldots, F_n\}^c\ u$
  - but dropping superscript ^{c}
- For instance, '`{p,q,r} :- {p,q,r} 2`' stands for $\{p, q, r\}^c \leftarrow \{p, q, r\}\ 2$.

Usage in `lparse` and `gringo`

In `lparse` and `gringo`,

- rule bodies may include
  - $l\ \{F_1, \ldots, F_n\}$
  - $\{F_1, \ldots, F_n\}\ u$
  - $l\ \{F_1, \ldots, F_n\}\ u$
- rule heads may include
  - $\{F_1, \ldots, F_n\}^c$
  - $l\ \{F_1, \ldots, F_n\}^c$
  - $\{F_1, \ldots, F_n\}^c\ u$
  - $l\ \{F_1, \ldots, F_n\}^c\ u$
  - but dropping superscript ^{c}
- For instance, '`{p,q,r} :- {p,q,r} 2`' stands for $\{p, q, r\}^c \leftarrow \{p, q, r\}\ 2$.

Conditional literals in `lparse` and `gringo`

- We often want to encode the contents of a (multi-)set rather than enumerating each of the elements.
- To support this, `lparse` and `gringo` allow for conditional literals.

Syntax

$$A_0 : A_1 : \ldots : A_m : \mathit{not}\ A_{m+1} : \ldots : \mathit{not}\ A_n$$

Informal meaning

List all ground instances of A_0 such that corresponding instances of $A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$ are true.

Example

`gringo` instantiates the program:

```
p(1). p(2). p(3).  q(2).
{r(X) : p(X) : not q(X)}.
```

to:

```
p(1). p(2). p(3).  q(2).
{r(1), r(3)}.
```

Conditional literals in `lparse` and `gringo`

- We often want to encode the contents of a (multi-)set rather than enumerating each of the elements.
- To support this, `lparse` and `gringo` allow for conditional literals.

Syntax

$$A_0 : A_1 : \ldots : A_m : \mathit{not}\ A_{m+1} : \ldots : \mathit{not}\ A_n$$

Informal meaning

List all ground instances of A_0 such that corresponding instances of $A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$ are true.

Example

`gringo` instantiates the program:

```
p(1). p(2). p(3).  q(2).
{r(X) : p(X) : not q(X)}.
```

to:

```
p(1). p(2). p(3).  q(2).
{r(1), r(3)}.
```

Domain predicates in lparse and gringo

- The predicates of literals on the right-hand side of a colon (:) must be defined from facts without any negative recursion.
- Such domain predicates are fully evaluated by lparse and gringo.

Example

```
p(1). p(2).
q(X) :- p(X), not p(X+1).
q(X) :- p(X), q(X+1).
r(X) :- p(X), not r(X+1).
```

- p/1 and q/1 are domain predicates because none of them negatively depends on itself.
- r/1 is not a domain predicate because it is defined in terms of `not r(X+1)`.

See lparse and gringo documentations for further details.

Domain predicates in lparse and gringo

- The predicates of literals on the right-hand side of a colon (:) must be defined from facts without any negative recursion.
- Such domain predicates are fully evaluated by lparse and gringo.

Example

```
p(1). p(2).
q(X) :- p(X), not p(X+1).
q(X) :- p(X), q(X+1).
r(X) :- p(X), not r(X+1).
```

- p/1 and q/1 are domain predicates because none of them negatively depends on itself.
- r/1 is not a domain predicate because it is defined in terms of `not r(X+1)`.

See lparse and gringo documentations for further details.

Domain predicates in lparse and gringo

- The predicates of literals on the right-hand side of a colon (:) must be defined from facts without any negative recursion.
- Such domain predicates are fully evaluated by lparse and gringo.

Example

```
p(1). p(2).
q(X) :- p(X), not p(X+1).
q(X) :- p(X), q(X+1).
r(X) :- p(X), not r(X+1).
```

- p/1 and q/1 are domain predicates because none of them negatively depends on itself.
- r/1 is not a domain predicate because it is defined in terms of `not r(X+1)`.

See lparse and gringo documentations for further details.

Normal form in `lparse` and `gringo`

- Consider a logic program consisting of
  - normal rules
  - choice rules
  - cardinality rules
- Such a format is obtained by `lparse` or `gringo`

  and directly implemented by `smodels` and `clasp`.

Aggregates
Overview

35 Motivation

36 Syntax

37 Semantics

Motivation

- Aggregates provide a general way to obtain a single value from a collection of input values given as a set, a bag, or a list.
- Popular aggregate (functions):
  - Average
  - Count
  - Maximum
  - Minimum
  - Sum
- Cardinality and Weight constraints rely on Count and Sum aggregates.

Syntax

- An aggregate has the form:

$$F\,\langle A_1 = w_1, \ldots, A_m = w_m, \mathit{not}\ A_{m+1} = w_{m+1}, \ldots, \mathit{not}\ A_n = w_n\rangle \prec k$$

  where
  - F stands for a function mapping multi-sets of $\mathbb{Z}$ to $\mathbb{Z} \cup \{+\infty, -\infty\}$,
  - $\prec$ stands for a relation between $\mathbb{Z} \cup \{+\infty, -\infty\}$ and $\mathbb{Z}$,
  - k an integer,
  - A_i is an atom, and
  - w_i are integers

  for $1 \leq i \leq n$.
- For instance, $\mathit{sum}\,\langle \mathit{hd}(a) = 30, \ldots, \mathit{hd}(m) = 50\rangle \leq 300$

Semantics

[21]

- A (positive) aggregate $F\ \langle A_1 = w_1, \ldots, A_n = w_n \rangle \prec k$ can be represented by the formula:

$$\bigwedge_{I \subseteq \{1,\ldots,n\}, F\langle w_i | i \in I\rangle \not\prec k} \left(\bigwedge_{i \in I} A_i \rightarrow \bigvee_{i \in \bar{I}} A_i \right)$$

  where $\bar{I} = \{1, \ldots, n\} \setminus I$ and $\not\prec$ is the complement of $\prec$.
- Then, $F\ \langle A_1 = w_1, \ldots, A_n = w_n \rangle \prec k$ is true in X iff the above formula is true in X.

An example

- Consider $sum\langle p = 1, q = 1\rangle \neq 1$
  ➥ that is, $A_1 = p$, $A_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\langle w_i \mid i \in I\rangle$	$\sum\langle w_i \mid i \in I\rangle$	$\sum\langle w_i \mid i \in I\rangle = 1$
$\emptyset$	$\langle\rangle$	0	*false*
$\{1\}$	$\langle 1\rangle$	1	*true*
$\{2\}$	$\langle 1\rangle$	1	*true*
$\{1,2\}$	$\langle 1,1\rangle$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\langle p = 1, q = 1\rangle = 1$.

An example

- Consider $sum\langle p = 1, q = 1\rangle \neq 1$
  ➥ that is, $A_1 = p$, $A_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\langle w_i \mid i \in I\rangle$	$\sum\langle w_i \mid i \in I\rangle$	$\sum\langle w_i \mid i \in I\rangle = 1$
$\emptyset$	$\langle\rangle$	0	*false*
$\{1\}$	$\langle 1\rangle$	1	*true*
$\{2\}$	$\langle 1\rangle$	1	*true*
$\{1,2\}$	$\langle 1,1\rangle$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\langle p = 1, q = 1\rangle = 1$.

An example

- Consider $sum\langle p = 1, q = 1\rangle \neq 1$
  ➥ that is, $A_1 = p$, $A_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\langle w_i \mid i \in I\rangle$	$\sum\langle w_i \mid i \in I\rangle$	$\sum\langle w_i \mid i \in I\rangle = 1$
$\emptyset$	$\langle\rangle$	0	*false*
$\{1\}$	$\langle 1\rangle$	1	*true*
$\{2\}$	$\langle 1\rangle$	1	*true*
$\{1,2\}$	$\langle 1,1\rangle$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\langle p = 1, q = 1\rangle = 1$.

Monotonicity

Monotone aggregates

- For instance,
  - $body^+(r)$
  - $sum\langle p = 1, q = 1\rangle > 1$ amounts to $q \wedge p$
- We get a simpler characterization:
  $\bigwedge_{I \subseteq \{1,\ldots,n\}, F\langle w_i | i \in I\rangle \not< k} \bigvee_{i \in \bar{I}} A_i$

Anti-monotone aggregates

- For instance,
  - $body^-(r)$
  - $sum\langle p = 1, q = 1\rangle < 1$ amounts to $\neg p \wedge \neg q$
- We get a simpler characterization:
  $\bigwedge_{I \subseteq \{1,\ldots,n\}, F\langle w_i | i \in I\rangle \not< k} \neg \bigwedge_{i \in I} A_i$

Non-monotone aggregates

- For instance, $sum\langle p = 1, q = 1\rangle \neq 1$ is non-monotone.

Monotonicity

Monotone aggregates

- For instance,
  - $body^+(r)$
  - $sum\langle p = 1, q = 1\rangle > 1$ amounts to $q \wedge p$
- We get a simpler characterization:
  $\bigwedge_{I \subseteq \{1,\ldots,n\}, F\langle w_i | i \in I\rangle \not\prec k} \bigvee_{i \in \bar{I}} A_i$

Anti-monotone aggregates

- For instance,
  - $body^-(r)$
  - $sum\langle p = 1, q = 1\rangle < 1$ amounts to $\neg p \wedge \neg q$
- We get a simpler characterization:
  $\bigwedge_{I \subseteq \{1,\ldots,n\}, F\langle w_i | i \in I\rangle \not\prec k} \neg \bigwedge_{i \in I} A_i$

Non-monotone aggregates

- For instance, $sum\langle p = 1, q = 1\rangle \neq 1$ is non-monotone.

The smodels approach
Overview

38 Motivation

39 Approximation

40 Partial Interpretations

41 Basic Algorithms

(Towards) the smodels approach [73]

- Wanted:
  - An efficient procedure to compute answer sets
- The smodels approach:
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a 3-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**select**)
  - ☞ Heuristic choices are made on atoms

(Towards) the smodels approach [73]

- Wanted:
  - An efficient procedure to compute answer sets
- The smodels approach:
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a 3-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**select**)
  - ☞ Heuristic choices are made on atoms

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$X \subseteq Y$ implies $\Pi^Y \subseteq \Pi^X$ implies $Cn(\Pi^Y) \subseteq Cn(\Pi^X)$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$X \subseteq Y$ implies $\Pi^Y \subseteq \Pi^X$ implies $Cn(\Pi^Y) \subseteq Cn(\Pi^X)$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets

First Idea Approximate an answer set X by two sets of atoms L and U such that $L \subseteq X \subseteq U$.

- L and U constitute lower and upper bounds on X.
- L and $(\mathcal{A} \setminus U)$ describe a 3-valued model of the program.

Observation

$$X \subseteq Y \text{ implies } \Pi^Y \subseteq \Pi^X \text{ implies } Cn(\Pi^Y) \subseteq Cn(\Pi^X)$$

Properties Let X be an answer set of normal logic program Π.

- If $L \subseteq X$, then $X \subseteq Cn(\Pi^L)$.
- If $X \subseteq U$, then $Cn(\Pi^U) \subseteq X$.
- If $L \subseteq X \subseteq U$, then $L \cup Cn(\Pi^U) \subseteq X \subseteq U \cap Cn(\Pi^L)$.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

Approximating answer sets (ctd)

Second Idea

Iterate

- Replace L by $L \cup Cn(\Pi^U)$
- Replace U by $U \cap Cn(\Pi^L)$

until L and U do not change anymore.

Observations

- At each iteration step
  - L becomes larger (or equal)
  - U becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every answer set X of Π
- If $L \not\subseteq U$, then Π has no answer set!
- If $L = U$, then L is an answer set of Π.

The simplistic expand algorithm

expand(L, U)

repeat

$L' \leftarrow L$
$U' \leftarrow U$
$L \leftarrow L' \cup Cn(\Pi^{U'})$
$U \leftarrow U' \cap Cn(\Pi^{L'})$
if $L \not\subseteq U$ **then return**

until $L = L'$ and $U = U'$

☞ Π is a global parameter!

Let's expand!

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

	L'	$Cn(\Pi^{U'})$	L	U'	$Cn(\Pi^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a,b,c,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
2	$\{a\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
3	$\{a,b\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$

- We have $\{a,b\} \subseteq X$ and $(\mathcal{A} \setminus \{a,b,d,e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every answer set X of Π.

Let's expand!

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

	L'	$Cn(\Pi^{U'})$	L	U'	$Cn(\Pi^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a,b,c,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
2	$\{a\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$
3	$\{a,b\}$	$\{a,b\}$	$\{a,b\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$	$\{a,b,d,e\}$

- We have $\{a,b\} \subseteq X$ and $(\mathcal{A} \setminus \{a,b,d,e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every answer set X of Π.

The simplistic expand algorithm (ctd)

expand

- tightens the approximation on answer sets
- is answer set preserving

Let's expand with d !

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

	L'	$Cn(\Pi^{U'})$	L	U'	$Cn(\Pi^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

➥ $\{a, b, d\}$ is an answer set X of Π.

Let's expand with d !

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

	L'	$Cn(\Pi^{U'})$	L	U'	$Cn(\Pi^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

➥ $\{a, b, d\}$ is an answer set X of Π.

Let's expand with "*not d*" !

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

	L'	$Cn(\Pi^{U'})$	L	U'	$Cn(\Pi^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

➥ $\{a, b, e\}$ is an answer set X of Π.

Let's expand with "*not d*" !

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

	L'	$Cn(\Pi^{U'})$	L	U'	$Cn(\Pi^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

➥ $\{a, b, e\}$ is an answer set X of Π.

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation of a logic program Π maps atoms on truth values: $\{true, false, unknown\}$.

Representation $\langle T, F \rangle$, where

- T is the set of all *true* atoms and
- F is the set of all *false* atoms.
- Truth of atoms in $atom(\Pi) \setminus (T \cup F)$ is *unknown*.

☞ By $atom(\Pi)$, we denote the set of atoms occuring in Π.

Properties

- $\langle T, F \rangle$ is conflicting iff $T \cap F \neq \emptyset$.
- $\langle T, F \rangle$ is total iff $T \cup F = atom(\Pi)$ and $T \cap F = \emptyset$.

Definition For $\langle T_1, F_1 \rangle$ and $\langle T_2, F_2 \rangle$, define:

- $\langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
- $\langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle$

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation of a logic program Π maps atoms on truth values: $\{true, false, unknown\}$.

Representation $\langle T, F \rangle$, where

- T is the set of all *true* atoms and
- F is the set of all *false* atoms.
- Truth of atoms in $atom(\Pi) \setminus (T \cup F)$ is *unknown*.

☞ By $atom(\Pi)$, we denote the set of atoms occuring in Π.

Properties

- $\langle T, F \rangle$ is conflicting iff $T \cap F \neq \emptyset$.
- $\langle T, F \rangle$ is total iff $T \cup F = atom(\Pi)$ and $T \cap F = \emptyset$.

Definition For $\langle T_1, F_1 \rangle$ and $\langle T_2, F_2 \rangle$, define:

- $\langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
- $\langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle$

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation of a logic program Π maps atoms on truth values: $\{true, false, unknown\}$.

Representation $\langle T, F\rangle$, where

- T is the set of all *true* atoms and
- F is the set of all *false* atoms.
- Truth of atoms in $atom(\Pi) \setminus (T \cup F)$ is *unknown*.

☞ By $atom(\Pi)$, we denote the set of atoms occuring in Π.

Properties

- $\langle T, F\rangle$ is conflicting iff $T \cap F \neq \emptyset$.
- $\langle T, F\rangle$ is total iff $T \cup F = atom(\Pi)$ and $T \cap F = \emptyset$.

Definition For $\langle T_1, F_1\rangle$ and $\langle T_2, F_2\rangle$, define:

- $\langle T_1, F_1\rangle \sqsubseteq \langle T_2, F_2\rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
- $\langle T_1, F_1\rangle \sqcup \langle T_2, F_2\rangle = \langle T_1 \cup T_2, F_1 \cup F_2\rangle$

The **smodels** (decision) algorithm

Global: Normal logic program Π

smodels($\langle T, F \rangle$**)**

$\langle T, F \rangle \leftarrow$ **expand(**$\langle T, F \rangle$**)**
if $\langle T, F \rangle$ is conflicting **then return**
else if $\langle T, F \rangle$ is total **then exit with** T
else
$A \leftarrow$ **select(**$atom(\Pi) \setminus (T \cup F)$**)**
smodels($\langle T \cup \{A\}, F \rangle$**)**
smodels($\langle T, F \cup \{A\} \rangle$**)**

Call: **smodels(**$\langle \emptyset, \emptyset \rangle$**)**

The **smodels** (decision) algorithm

Global: Normal logic program Π

smodels($\langle T, F \rangle$**)**

$\langle T, F \rangle \leftarrow$ **expand(**$\langle T, F \rangle$**)**
if $\langle T, F \rangle$ is conflicting **then return**
else if $\langle T, F \rangle$ is total **then exit with** T
else
$A \leftarrow$ **select(**$atom(\Pi) \setminus (T \cup F)$**)**
smodels($\langle T \cup \{A\}, F \rangle$**)**
smodels($\langle T, F \cup \{A\} \rangle$**)**

Call: **smodels(**$\langle \emptyset, \emptyset \rangle$**)**

Deterministic consequences via **expand**

Global: Normal logic program Π

expand($\langle T, F \rangle$**)**

repeat

$\langle T, F \rangle \leftarrow$ **atleast(**$\langle T, F \rangle$**)**

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

else

$F' \leftarrow F$

$F \leftarrow F \cup$ **atmost(**$\langle T, F \rangle$**)**

until $F = F'$

return $\langle T, F \rangle$

- **atleast(**$\langle T, F \rangle$**)** derives deterministic consequences from Clark's completion
- **atmost(**$\langle T, F \rangle$**)** derives deterministic consequences from unfounded sets

Deterministic consequences via **expand**

Global: Normal logic program Π

expand($\langle T, F \rangle$**)**

repeat

$\langle T, F \rangle \leftarrow$ **atleast(**$\langle T, F \rangle$**)**

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

else

$F' \leftarrow F$

$F \leftarrow F \cup$ **atmost(**$\langle T, F \rangle$**)**

until $F = F'$

return $\langle T, F \rangle$

- **atleast(**$\langle T, F \rangle$**)** derives deterministic consequences from Clark's completion
- **atmost(**$\langle T, F \rangle$**)** derives deterministic consequences from unfounded sets

A glimpse at **atleast**($\langle T, F \rangle$)

repeat

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

$\langle T', F' \rangle \leftarrow \langle T, F \rangle$

case of

$r \in \Pi$ **such that** $head(r) \notin T$ **and** $body^+(r) \subseteq T, body^-(r) \subseteq F$**:**
$T \leftarrow T \cup \{head(r)\}$

$A \in (atom(\Pi) \setminus F)$ **such that for all** $r \in \Pi$: $head(r) \neq A$ **or** $(body^+(r) \cap F) \cup (body^-(r) \cap T) \neq \emptyset$**:**
$F \leftarrow F \cup \{A\}$

$head(r) \in F, r \in \Pi$ **such that** $body^+(r) \cap body^-(r) = \emptyset$ **and** $(body^+(r) \setminus T) \cup (body^-(r) \setminus F) = \{A\}$**:**
if $A \in body^+(r)$ **then** $F \leftarrow F \cup \{A\}$ **else** $T \leftarrow T \cup \{A\}$

$(A = head(r)) \in T, r \in \Pi$ **such that** $body^+(r) \not\subseteq T$ **or** $body^-(r) \not\subseteq F$ **and for all** $r' \in \Pi \setminus \{r\}$: $head(r') \neq A$ **or** $(body^+(r') \cap F) \cup (body^-(r') \cap T) \neq \emptyset$**:**
$T \leftarrow T \cup body^+(r)$
$F \leftarrow F \cup body^-(r)$

until $\langle T, F \rangle = \langle T', F' \rangle$

return $\langle T, F \rangle$

A glimpse at **atleast**($\langle T, F \rangle$)

repeat

  if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

  $\langle T', F' \rangle \leftarrow \langle T, F \rangle$

  case of

  $r \in \Pi$ **such that** $head(r) \notin T$ **and** $body^+(r) \subseteq T, body^-(r) \subseteq F$**:**

    $T \leftarrow T \cup \{head(r)\}$

  $A \in (atom(\Pi) \setminus F)$ **such that for all** $r \in \Pi$**:** $head(r) \neq A$ **or** $(body^+(r) \cap F) \cup (body^-(r) \cap T) \neq \emptyset$**:**

    $F \leftarrow F \cup \{A\}$

  $head(r) \in F, r \in \Pi$ **such that** $body^+(r) \cap body^-(r) = \emptyset$ **and** $(body^+(r) \setminus T) \cup (body^-(r) \setminus F) = \{A\}$**:**

    if $A \in body^+(r)$ **then** $F \leftarrow F \cup \{A\}$ **else** $T \leftarrow T \cup \{A\}$

  $(A = head(r)) \in T, r \in \Pi$ **such that** $body^+(r) \not\subseteq T$ **or** $body^-(r) \not\subseteq F$ **and for all** $r' \in \Pi \setminus \{r\}$**:** $head(r') \neq A$ **or** $(body^+(r') \cap F) \cup (body^-(r') \cap T) \neq \emptyset$**:**

    $T \leftarrow T \cup body^+(r)$

    $F \leftarrow F \cup body^-(r)$

until $\langle T, F \rangle = \langle T', F' \rangle$

return $\langle T, F \rangle$

A glimpse at **atmost($\langle T, F \rangle$)**

return $\mathbf{U}_{\Pi}\langle T, F \rangle$

A glimpse at **atmost**($\langle T, F \rangle$)

return $\mathbf{U}_{\sqcap}\langle T, F \rangle$

Completion
Overview

42 Supported Models

43 Fitting Operator

44 Implementation via smodels

45 Tightness

Completion

[11]

Let Π be a normal logic program,
and recall that $atom(\Pi)$ denotes the set of atoms occurring in Π.
The completion of Π is defined as follows:

$$Comp(body(r)) = \bigwedge_{A \in body^+(r)} A \wedge \bigwedge_{A \in body^-(r)} \neg A$$

$$Comp(\Pi) = \{A \leftrightarrow \bigvee_{r \in \Pi, head(r)=A} Comp(body(r)) \mid A \in atom(\Pi)\}$$

- Every answer set of Π is a model of $Comp(\Pi)$, but not vice versa.
- Models of $Comp(\Pi)$ are called the supported models of Π.
- ☞ In other words, every answer set of Π is a supported model of Π.
- ☞ By definition, every supported model of Π is also a model of Π.

Completion [11]

Let Π be a normal logic program,
and recall that $atom(\Pi)$ denotes the set of atoms occurring in Π.
The completion of Π is defined as follows:

$$Comp(body(r)) = \bigwedge_{A \in body^+(r)} A \wedge \bigwedge_{A \in body^-(r)} \neg A$$

$$Comp(\Pi) = \{A \leftrightarrow \bigvee_{r \in \Pi, head(r)=A} Comp(body(r)) \mid A \in atom(\Pi)\}$$

- Every answer set of Π is a model of $Comp(\Pi)$, but not vice versa.
- Models of $Comp(\Pi)$ are called the supported models of Π.
- ☞ In other words, every answer set of Π is a supported model of Π.
- ☞ By definition, every supported model of Π is also a model of Π.

Completion [11]

Let Π be a normal logic program,
and recall that $atom(\Pi)$ denotes the set of atoms occurring in Π.
The completion of Π is defined as follows:

$$Comp(body(r)) = \bigwedge_{A \in body^+(r)} A \wedge \bigwedge_{A \in body^-(r)} \neg A$$

$$Comp(\Pi) = \{A \leftrightarrow \bigvee_{r \in \Pi, head(r)=A} Comp(body(r)) \mid A \in atom(\Pi)\}$$

- Every answer set of Π is a model of $Comp(\Pi)$, but not vice versa.
- Models of $Comp(\Pi)$ are called the supported models of Π.
- ☞ In other words, every answer set of Π is a supported model of Π.
- ☞ By definition, every supported model of Π is also a model of Π.

Completion

[11]

Let Π be a normal logic program,
and recall that $atom(\Pi)$ denotes the set of atoms occurring in Π.
The completion of Π is defined as follows:

$$Comp(body(r)) = \bigwedge_{A \in body^+(r)} A \wedge \bigwedge_{A \in body^-(r)} \neg A$$

$$Comp(\Pi) = \{A \leftrightarrow \bigvee_{r \in \Pi, head(r)=A} Comp(body(r)) \mid A \in atom(\Pi)\}$$

- Every answer set of Π is a model of $Comp(\Pi)$, but not vice versa.
- Models of $Comp(\Pi)$ are called the supported models of Π.
- ☞ In other words, every answer set of Π is a supported model of Π.
- ☞ By definition, every supported model of Π is also a model of Π.

Completion [11]

Let Π be a normal logic program,
and recall that $atom(\Pi)$ denotes the set of atoms occurring in Π.
The completion of Π is defined as follows:

$$Comp(body(r)) = \bigwedge_{A \in body^+(r)} A \wedge \bigwedge_{A \in body^-(r)} \neg A$$

$$Comp(\Pi) = \{A \leftrightarrow \bigvee_{r \in \Pi, head(r)=A} Comp(body(r)) \mid A \in atom(\Pi)\}$$

- Every answer set of Π is a model of $Comp(\Pi)$, but not vice versa.
- Models of $Comp(\Pi)$ are called the supported models of Π.
- ☞ In other words, every answer set of Π is a supported model of Π.
- ☞ By definition, every supported model of Π is also a model of Π.

Completion [11]

Let Π be a normal logic program,
and recall that $atom(\Pi)$ denotes the set of atoms occurring in Π.
The completion of Π is defined as follows:

$$Comp(body(r)) = \bigwedge_{A \in body^+(r)} A \wedge \bigwedge_{A \in body^-(r)} \neg A$$

$$Comp(\Pi) = \{A \leftrightarrow \bigvee_{r \in \Pi, head(r)=A} Comp(body(r)) \mid A \in atom(\Pi)\}$$

- Every answer set of Π is a model of $Comp(\Pi)$, but not vice versa.
- Models of $Comp(\Pi)$ are called the supported models of Π.

☞ In other words, every answer set of Π is a supported model of Π.

☞ By definition, every supported model of Π is also a model of Π.

A first example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b & \leftarrow & a \\ c & \leftarrow & b \\ c & \leftarrow & d \\ d & \leftarrow & c, e \end{array} \right\} \qquad Comp(\Pi) = \left\{ \begin{array}{lcl} a & \leftrightarrow & \top \\ b & \leftrightarrow & a \\ c & \leftrightarrow & (b \vee d) \\ d & \leftrightarrow & (c \wedge e) \\ e & \leftrightarrow & \bot \end{array} \right\}$$

- The supported model of Π is $\{a, b, c\}$.
- The answer set of Π is $\{a, b, c\}$.

A first example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b & \leftarrow & a \\ c & \leftarrow & b \\ c & \leftarrow & d \\ d & \leftarrow & c, e \end{array} \right\} \qquad Comp(\Pi) = \left\{ \begin{array}{lcl} a & \leftrightarrow & \top \\ b & \leftrightarrow & a \\ c & \leftrightarrow & (b \vee d) \\ d & \leftrightarrow & (c \wedge e) \\ e & \leftrightarrow & \bot \end{array} \right\}$$

- The supported model of Π is $\{a, b, c\}$.
- The answer set of Π is $\{a, b, c\}$.

A first example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & \\ b & \leftarrow & a \\ c & \leftarrow & b \\ c & \leftarrow & d \\ d & \leftarrow & c, e \end{array} \right\} \qquad Comp(\Pi) = \left\{ \begin{array}{lcl} a & \leftrightarrow & \top \\ b & \leftrightarrow & a \\ c & \leftrightarrow & (b \vee d) \\ d & \leftrightarrow & (c \wedge e) \\ e & \leftrightarrow & \bot \end{array} \right\}$$

- The supported model of Π is $\{a, b, c\}$.
- The answer set of Π is $\{a, b, c\}$.

A second example

$$\Pi = \left\{ \begin{array}{lcl} q & \leftarrow & not\ p \\ p & \leftarrow & not\ q, not\ x \end{array} \right\} \quad Comp(\Pi) = \left\{ \begin{array}{lcl} q & \leftrightarrow & \neg p \\ p & \leftrightarrow & (\neg q \wedge \neg x) \\ x & \leftrightarrow & \bot \end{array} \right\}$$

- The supported models of Π are $\{p\}$ and $\{q\}$.
- The answer sets of Π are $\{p\}$ and $\{q\}$.

A second example

$$\Pi = \left\{ \begin{array}{lcl} q & \leftarrow & not\ p \\ p & \leftarrow & not\ q, not\ x \end{array} \right\} \quad Comp(\Pi) = \left\{ \begin{array}{lcl} q & \leftrightarrow & \neg p \\ p & \leftrightarrow & (\neg q \wedge \neg x) \\ x & \leftrightarrow & \bot \end{array} \right\}$$

- The supported models of Π are $\{p\}$ and $\{q\}$.
- The answer sets of Π are $\{p\}$ and $\{q\}$.

A second example

$$\Pi = \left\{ \begin{array}{lcl} q & \leftarrow & not\ p \\ p & \leftarrow & not\ q, not\ x \end{array} \right\} \quad Comp(\Pi) = \left\{ \begin{array}{lcl} q & \leftrightarrow & \neg p \\ p & \leftrightarrow & (\neg q \wedge \neg x) \\ x & \leftrightarrow & \bot \end{array} \right\}$$

- The supported models of Π are $\{p\}$ and $\{q\}$.
- The answer sets of Π are $\{p\}$ and $\{q\}$.

A third example

$$\Pi = \{\ p \leftarrow p\ \} \qquad \mathit{Comp}(\Pi) = \{\ p \leftrightarrow p\ \}$$

- The supported models of Π are $\emptyset$ and $\{p\}$.
- The answer set of Π is $\emptyset$!

A third example

$$\Pi = \{\ p \leftarrow p\ \} \qquad \mathit{Comp}(\Pi) = \{\ p \leftrightarrow p\ \}$$

- The supported models of Π are $\emptyset$ and $\{p\}$.
- The answer set of Π is $\emptyset$!

A third example

$$\Pi = \{\ p \leftarrow p\ \} \qquad Comp(\Pi) = \{\ p \leftrightarrow p\ \}$$

- The supported models of Π are $\emptyset$ and $\{p\}$.
- The answer set of Π is $\emptyset$!

Fitting operator: Basic idea

Idea Extend T_Π to normal logic programs.

Logical background Completion

- The head atom of a rule must be *true* if the rule's body is *true*.
- An atom must be *false* if the body of each rule having it as head is *false*.

Fitting operator: Basic idea

Idea Extend T_Π to normal logic programs.

Logical background Completion

- The head atom of a rule must be *true* if the rule's body is *true*.
- An atom must be *false* if the body of each rule having it as head is *false*.

Fitting operator: Definition

[23]

Let Π be a normal logic program.

Define

$$\mathbf{\Phi}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{F}_\Pi\langle T, F\rangle\rangle$$

where

$$\mathbf{T}_\Pi\langle T, F\rangle = \{head(r) \mid r \in \Pi, body^+(r) \subseteq T, body^-(r) \subseteq F\}$$

$$\mathbf{F}_\Pi\langle T, F\rangle = \{A \in atom(\Pi) \mid body^+(r) \cap F \neq \emptyset \text{ or } body^-(r) \cap T \neq \emptyset \text{ for each } r \in \Pi \text{ such that } head(r) = A\}$$

Fitting operator: Definition

[23]

Let Π be a normal logic program.

Define

$$\mathbf{\Phi}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{F}_\Pi\langle T, F\rangle\rangle$$

where

$$\mathbf{T}_\Pi\langle T, F\rangle = \{head(r) \mid r \in \Pi, body^+(r) \subseteq T, body^-(r) \subseteq F\}$$

$$\mathbf{F}_\Pi\langle T, F\rangle = \{A \in atom(\Pi) \mid body^+(r) \cap F \neq \emptyset \text{ or } body^-(r) \cap T \neq \emptyset \text{ for each } r \in \Pi \text{ such that } head(r) = A\}$$

Fitting operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Phi}_{\Pi_1}$ on $\langle\{a\},\{d\}\rangle$:

$$\begin{aligned} \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{d\}\rangle &= \langle\{a,c\},\{b\}\rangle \\ \mathbf{\Phi}_{\Pi_1}\langle\{a,c\},\{b\}\rangle &= \langle\{a\},\{b,d\}\rangle \\ \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b,d\}\rangle &= \langle\{a,c\},\{b\}\rangle \\ &\vdots \end{aligned}$$

Fitting operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Phi}_{\Pi_1}$ on $\langle \{a\}, \{d\} \rangle$:

$$\begin{array}{rcl} \mathbf{\Phi}_{\Pi_1} \langle \{a\}, \{d\} \rangle & = & \langle \{a, c\}, \{b\} \rangle \\ \mathbf{\Phi}_{\Pi_1} \langle \{a, c\}, \{b\} \rangle & = & \langle \{a\}, \{b, d\} \rangle \\ \mathbf{\Phi}_{\Pi_1} \langle \{a\}, \{b, d\} \rangle & = & \langle \{a, c\}, \{b\} \rangle \\ & \vdots & \end{array}$$

Fitting operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Phi}_{\Pi_1}$ on $\langle \{a\}, \{d\} \rangle$:

$$\begin{array}{rcl} \mathbf{\Phi}_{\Pi_1}\langle \{a\}, \{d\} \rangle & = & \langle \{a, c\}, \{b\} \rangle \\ \mathbf{\Phi}_{\Pi_1}\langle \{a, c\}, \{b\} \rangle & = & \langle \{a\}, \{b, d\} \rangle \\ \mathbf{\Phi}_{\Pi_1}\langle \{a\}, \{b, d\} \rangle & = & \langle \{a, c\}, \{b\} \rangle \\ & \vdots & \end{array}$$

Fitting operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Phi}_{\Pi_1}$ on $\langle\{a\},\{d\}\rangle$:

$$\begin{aligned} \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{d\}\rangle &= \langle\{a,c\},\{b\}\rangle \\ \mathbf{\Phi}_{\Pi_1}\langle\{a,c\},\{b\}\rangle &= \langle\{a\},\{b,d\}\rangle \\ \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b,d\}\rangle &= \langle\{a,c\},\{b\}\rangle \\ &\vdots \end{aligned}$$

Fitting operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Phi}_{\Pi_1}$ on $\langle\{a\},\{d\}\rangle$:

$$\begin{array}{rcl} \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{d\}\rangle & = & \langle\{a,c\},\{b\}\rangle \\ \mathbf{\Phi}_{\Pi_1}\langle\{a,c\},\{b\}\rangle & = & \langle\{a\},\{b,d\}\rangle \\ \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b,d\}\rangle & = & \langle\{a,c\},\{b\}\rangle \\ & \vdots & \end{array}$$

Fitting semantics

Define the iterative variant of $\mathbf{\Phi}_\Pi$ analogously to T_Π:

$$\mathbf{\Phi}^0_\Pi\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Phi}^{i+1}_\Pi\langle T, F\rangle = \mathbf{\Phi}_\Pi\mathbf{\Phi}^i_\Pi\langle T, F\rangle$$

Define the Fitting semantics of a normal logic program Π as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Phi}^i_\Pi\langle\emptyset, \emptyset\rangle$$

Fitting semantics

Define the iterative variant of $\mathbf{\Phi}_\Pi$ analogously to T_Π:

$$\mathbf{\Phi}^0_\Pi\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Phi}^{i+1}_\Pi\langle T, F\rangle = \mathbf{\Phi}_\Pi\mathbf{\Phi}^i_\Pi\langle T, F\rangle$$

Define the Fitting semantics of a normal logic program Π as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Phi}^i_\Pi\langle\emptyset, \emptyset\rangle$$

Fitting semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \Phi^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \Phi^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Phi_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \Phi^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Phi_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b\}\rangle \\ \Phi^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Phi_{\Pi_1}\langle\{a\},\{b\}\rangle & = & \langle\{a\},\{b\}\rangle \\ \bigsqcup_{i\geq 0}\Phi^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b\}\rangle & & \end{array}$$

Fitting semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \mathbf{\Phi}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \mathbf{\Phi}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b\}\rangle \\ \mathbf{\Phi}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b\}\rangle & = & \langle\{a\},\{b\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Phi}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b\}\rangle & & \end{array}$$

Fitting semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \mathbf{\Phi}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \mathbf{\Phi}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b\}\rangle \\ \mathbf{\Phi}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b\}\rangle & = & \langle\{a\},\{b\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Phi}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b\}\rangle & & \end{array}$$

Fitting semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \mathbf{\Phi}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \mathbf{\Phi}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b\}\rangle \\ \mathbf{\Phi}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b\}\rangle & = & \langle\{a\},\{b\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Phi}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b\}\rangle & & \end{array}$$

Fitting semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \mathbf{\Phi}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \mathbf{\Phi}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b\}\rangle \\ \mathbf{\Phi}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b\}\rangle & = & \langle\{a\},\{b\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Phi}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b\}\rangle & & \end{array}$$

Fitting semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \mathbf{\Phi}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \mathbf{\Phi}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b\}\rangle \\ \mathbf{\Phi}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}_{\Pi_1}\langle\{a\},\{b\}\rangle & = & \langle\{a\},\{b\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Phi}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b\}\rangle & & \end{array}$$

Fitting semantics: Properties

Let Π be a normal logic program.

- $\mathbf{\Phi}_\Pi\langle\emptyset,\emptyset\rangle$ is monotonic.
  That is, $\mathbf{\Phi}^i_\Pi\langle\emptyset,\emptyset\rangle \sqsubseteq \mathbf{\Phi}^{i+1}_\Pi\langle\emptyset,\emptyset\rangle$.
- The Fitting semantics of Π is
  - not conflicting,
  - and generally not total.

Fitting fixpoints

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
Define $\langle T, F \rangle$ as a Fitting fixpoint of Π if $\mathbf{\Phi}_\Pi \langle T, F \rangle = \langle T, F \rangle$.

- The Fitting semantics is the $\sqsubseteq$-least Fitting fixpoint of Π.
- Any other Fitting fixpoint extends the Fitting semantics.
- Total Fitting fixpoints correspond to supported models.

Fitting fixpoints

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
Define $\langle T, F \rangle$ as a Fitting fixpoint of Π if $\mathbf{\Phi}_\Pi \langle T, F \rangle = \langle T, F \rangle$.

- The Fitting semantics is the $\sqsubseteq$-least Fitting fixpoint of Π.
- Any other Fitting fixpoint extends the Fitting semantics.
- Total Fitting fixpoints correspond to supported models.

Fitting fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has three total Fitting fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$
3. $\langle \{a, c, e\}, \{b, d\} \rangle$

Π_1 has three supported models, two of them are answer sets.

Fitting fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has three total Fitting fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$
3. $\langle \{a, c, e\}, \{b, d\} \rangle$

Π_1 has three supported models, two of them are answer sets.

Fitting fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has three total Fitting fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$
3. $\langle \{a, c, e\}, \{b, d\} \rangle$

Π_1 has three supported models, two of them are answer sets.

Fitting fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has three total Fitting fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$
3. $\langle \{a, c, e\}, \{b, d\} \rangle$

Π_1 has three supported models, two of them are answer sets.

Fitting fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has three total Fitting fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$
3. $\langle \{a, c, e\}, \{b, d\} \rangle$

Π_1 has three supported models, two of them are answer sets.

Fitting fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has three total Fitting fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$
3. $\langle \{a, c, e\}, \{b, d\} \rangle$

Π_1 has three supported models, two of them are answer sets.

Properties of Fitting operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Phi}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Phi}_\Pi$ is answer set preserving.

  $\mathbf{\Phi}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

However, $\mathbf{\Phi}_\Pi$ is still insufficient, because total fixpoints correspond to supported models, not necessarily answer sets.

☞ The problem is the same as with program completion.

The missing piece is non-circularity of derivations !

Properties of Fitting operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Phi}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Phi}_\Pi$ is answer set preserving.

  $\mathbf{\Phi}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

However, $\mathbf{\Phi}_\Pi$ is still insufficient, because total fixpoints correspond to supported models, not necessarily answer sets.

☞ The problem is the same as with program completion.

The missing piece is non-circularity of derivations !

Properties of Fitting operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Phi}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Phi}_\Pi$ is answer set preserving.
  - ➥ $\mathbf{\Phi}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

However, $\mathbf{\Phi}_\Pi$ is still insufficient, because total fixpoints correspond to supported models, not necessarily answer sets.

- ☞ The problem is the same as with program completion.

The missing piece is non-circularity of derivations !

Properties of Fitting operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Phi}_\Pi\langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Phi}_\Pi$ is answer set preserving.
  - ➥ $\mathbf{\Phi}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

However, $\mathbf{\Phi}_\Pi$ is still insufficient, because total fixpoints correspond to supported models, not necessarily answer sets.

- ☞ The problem is the same as with program completion.

The missing piece is non-circularity of derivations !

Properties of Fitting operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Phi}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Phi}_\Pi$ is answer set preserving.
  - ➥ $\mathbf{\Phi}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

However, $\mathbf{\Phi}_\Pi$ is still insufficient, because total fixpoints correspond to supported models, not necessarily answer sets.

- ☞ The problem is the same as with program completion.

The missing piece is non-circularity of derivations !

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

$$\begin{array}{lcl} \Phi_\Pi^0\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \\ \Phi_\Pi^1\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \end{array}$$

That is, Fitting semantics cannot assign *false* to a and b, although they can never become *true* !

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

$$\begin{array}{lcl} \Phi_\Pi^0\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \\ \Phi_\Pi^1\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \end{array}$$

That is, Fitting semantics cannot assign *false* to a and b, although they can never become *true* !

Rebuilding **atleast**($\langle T, F \rangle$)

repeat from Fitting operator

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

$\langle T', F' \rangle \leftarrow \langle T, F \rangle$

case of

$r \in \Pi$ **such that** $head(r) \notin T$ **and**
$body^+(r) \subseteq T, body^-(r) \subseteq F$**:**

$$T \leftarrow T \cup \{head(r)\}$$

$A \in (atom(\Pi) \setminus F)$ **such that for all** $r \in \Pi$:
$head(r) \neq A$ **or** $(body^+(r) \cap F) \cup (body^-(r) \cap T) \neq \emptyset$**:**

$$F \leftarrow F \cup \{A\}$$

until $\langle T, F \rangle = \langle T', F' \rangle$

return $\langle T, F \rangle$

Rebuilding **atleast**($\langle T, F \rangle$)

repeat from Fitting operator

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

$\langle T', F' \rangle \leftarrow \langle T, F \rangle$

case of

$r \in \Pi$ **such that** $head(r) \notin T$ **and**
$body^+(r) \subseteq T, body^-(r) \subseteq F$**:**

$$T \leftarrow T \cup \{head(r)\}$$

$A \in (atom(\Pi) \setminus F)$ **such that for all** $r \in \Pi$:
$head(r) \neq A$ **or** $(body^+(r) \cap F) \cup (body^-(r) \cap T) \neq \emptyset$**:**

$$F \leftarrow F \cup \{A\}$$

until $\langle T, F \rangle = \langle T', F' \rangle$

return $\langle T, F \rangle$

Relationship with Fitting semantics

Let Π be a normal logic program.

- $\mathbf{atleast}(\langle \emptyset, \emptyset \rangle) = \bigsqcup_{i \geq 0} \mathbf{\Phi}^i_\Pi \langle \emptyset, \emptyset \rangle$

What about supported models?
Consider:

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow b & b \leftarrow not\ c & c \leftarrow not\ b \\ d \leftarrow e & e \leftarrow not\ f & f \leftarrow not\ e \end{array} \right\}$$

$\mathbf{atleast}(\langle \{a\}, \{d\} \rangle) = \langle \{a\}, \{d\} \rangle$

The only supported model X of Π such that $a \in X$ and $d \notin X$ is $\{a, b, f\}$!

We can enhance $\mathbf{atleast}(\langle T, F \rangle)$ by backward propagation !

Relationship with Fitting semantics

Let Π be a normal logic program.

- $\mathbf{atleast}(\langle \emptyset, \emptyset \rangle) = \bigsqcup_{i \geq 0} \mathbf{\Phi}^i_\Pi \langle \emptyset, \emptyset \rangle$

What about supported models?

Consider:

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow b & b \leftarrow not\ c & c \leftarrow not\ b \\ d \leftarrow e & e \leftarrow not\ f & f \leftarrow not\ e \end{array} \right\}$$

$\mathbf{atleast}(\langle \{a\}, \{d\} \rangle) = \langle \{a\}, \{d\} \rangle$

The only supported model X of Π such that $a \in X$ and $d \notin X$ is $\{a, b, f\}$!

We can enhance $\mathbf{atleast}(\langle T, F \rangle)$ by backward propagation !

Relationship with Fitting semantics

Let Π be a normal logic program.

- $\textbf{atleast}(\langle \emptyset, \emptyset \rangle) = \bigsqcup_{i \geq 0} \mathbf{\Phi}^i_\Pi \langle \emptyset, \emptyset \rangle$

What about supported models?
Consider:

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow b & b \leftarrow not\ c & c \leftarrow not\ b \\ d \leftarrow e & e \leftarrow not\ f & f \leftarrow not\ e \end{array} \right\}$$

- $\textbf{atleast}(\langle \{a\}, \{d\} \rangle) = \langle \{a\}, \{d\} \rangle$
- The only supported model X of Π such that $a \in X$ and $d \notin X$ is $\{a, b, f\}$!

We can enhance $\textbf{atleast}(\langle T, F \rangle)$ by backward propagation !

Relationship with Fitting semantics

Let Π be a normal logic program.

- $\textbf{atleast}(\langle \emptyset, \emptyset \rangle) = \bigsqcup_{i \geq 0} \mathbf{\Phi}^i_\Pi \langle \emptyset, \emptyset \rangle$

What about supported models?
Consider:

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow b & b \leftarrow not\ c & c \leftarrow not\ b \\ d \leftarrow e & e \leftarrow not\ f & f \leftarrow not\ e \end{array} \right\}$$

- $\textbf{atleast}(\langle \{a\}, \{d\} \rangle) = \langle \{a\}, \{d\} \rangle$
- The only supported model X of Π such that $a \in X$ and $d \notin X$ is $\{a, b, f\}$!

We can enhance $\textbf{atleast}(\langle T, F \rangle)$ by backward propagation !

Relationship with Fitting semantics

Let Π be a normal logic program.

- $\textbf{atleast}(\langle \emptyset, \emptyset \rangle) = \bigsqcup_{i \geq 0} \mathbf{\Phi}^i_\Pi \langle \emptyset, \emptyset \rangle$

What about supported models?
Consider:

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow b & b \leftarrow not\ c & c \leftarrow not\ b \\ d \leftarrow e & e \leftarrow not\ f & f \leftarrow not\ e \end{array} \right\}$$

- $\textbf{atleast}(\langle \{a\}, \{d\} \rangle) = \langle \{a\}, \{d\} \rangle$
- The only supported model X of Π such that $a \in X$ and $d \notin X$ is $\{a, b, f\}$!

We can enhance $\textbf{atleast}(\langle T, F \rangle)$ by backward propagation !

Relationship with Fitting semantics

Let Π be a normal logic program.

- $\textbf{atleast}(\langle \emptyset, \emptyset \rangle) = \bigsqcup_{i \geq 0} \mathbf{\Phi}^i_\Pi \langle \emptyset, \emptyset \rangle$

What about supported models?
Consider:

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow b & b \leftarrow not\ c & c \leftarrow not\ b \\ d \leftarrow e & e \leftarrow not\ f & f \leftarrow not\ e \end{array} \right\}$$

- $\textbf{atleast}(\langle \{a\}, \{d\} \rangle) = \langle \{a\}, \{d\} \rangle$
- The only supported model X of Π such that $a \in X$ and $d \notin X$ is $\{a, b, f\}$!

We can enhance $\textbf{atleast}(\langle T, F \rangle)$ by backward propagation !

Rebuilding **atleast**($\langle T, F \rangle$)

repeat from supported models

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

$\langle T', F' \rangle \leftarrow \langle T, F \rangle$

case of

$r \in \Pi$ **such that** $head(r) \notin T$ **and**
$body^+(r) \subseteq T, body^-(r) \subseteq F$**:**
$T \leftarrow T \cup \{head(r)\}$

$A \in (atom(\Pi) \setminus F)$ **such that for all** $r \in \Pi$:
$head(r) \neq A$ **or** $(body^+(r) \cap F) \cup (body^-(r) \cap T) \neq \emptyset$**:**
$F \leftarrow F \cup \{A\}$

$head(r) \in F, r \in \Pi$ **such that** $body^+(r) \cap body^-(r) = \emptyset$ **and**
$(body^+(r) \setminus T) \cup (body^-(r) \setminus F) = \{A\}$**:**
if $A \in body^+(r)$ **then** $F \leftarrow F \cup \{A\}$ **else** $T \leftarrow T \cup \{A\}$

$(A = head(r)) \in T, r \in \Pi$ **such that** $body^+(r) \not\subseteq T$ **or** $body^-(r) \not\subseteq F$ **and**
for all $r' \in \Pi \setminus \{r\}$: $head(r') \neq A$ **or** $(body^+(r') \cap F) \cup (body^-(r') \cap T) \neq \emptyset$**:**
$T \leftarrow T \cup body^+(r)$
$F \leftarrow F \cup body^-(r)$

until $\langle T, F \rangle = \langle T', F' \rangle$

return $\langle T, F \rangle$

Rebuilding **atleast**($\langle T, F \rangle$)

repeat from supported models

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

$\langle T', F' \rangle \leftarrow \langle T, F \rangle$

case of

$r \in \Pi$ **such that** $head(r) \notin T$ **and**
$body^+(r) \subseteq T, body^-(r) \subseteq F$**:**
$T \leftarrow T \cup \{head(r)\}$

$A \in (atom(\Pi) \setminus F)$ **such that for all** $r \in \Pi$:
$head(r) \neq A$ **or** $(body^+(r) \cap F) \cup (body^-(r) \cap T) \neq \emptyset$**:**
$F \leftarrow F \cup \{A\}$

$head(r) \in F, r \in \Pi$ **such that** $body^+(r) \cap body^-(r) = \emptyset$ **and**
$(body^+(r) \setminus T) \cup (body^-(r) \setminus F) = \{A\}$**:**
if $A \in body^+(r)$ **then** $F \leftarrow F \cup \{A\}$ **else** $T \leftarrow T \cup \{A\}$

$(A = head(r)) \in T, r \in \Pi$ **such that** $body^+(r) \not\subseteq T$ **or** $body^-(r) \not\subseteq F$ **and**
for all $r' \in \Pi \setminus \{r\}$: $head(r') \neq A$ **or** $(body^+(r') \cap F) \cup (body^-(r') \cap T) \neq \emptyset$**:**
$T \leftarrow T \cup body^+(r)$
$F \leftarrow F \cup body^-(r)$

until $\langle T, F \rangle = \langle T', F' \rangle$

return $\langle T, F \rangle$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b\}\rangle$
select	$\langle \{a\}, \{b\}\rangle$	$\langle \{a, e\}, \{b\}\rangle$
expand	$\langle \{a, e\}, \{b\}\rangle$	$\langle \{a, c, e\}, \{b, d\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset \rangle$	
expand	$\langle \emptyset, \emptyset \rangle$	$\langle \{a\}, \{b\} \rangle$
select	$\langle \{a\}, \{b\} \rangle$	$\langle \{a, e\}, \{b\} \rangle$
expand	$\langle \{a, e\}, \{b\} \rangle$	$\langle \{a, c, e\}, \{b, d\} \rangle$
smodels	$\langle \emptyset, \emptyset \rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset \rangle$	
expand	$\langle \emptyset, \emptyset \rangle$	$\langle \{a\}, \{b\} \rangle$
select	$\langle \{a\}, \{b\} \rangle$	$\langle \{a, e\}, \{b\} \rangle$
expand	$\langle \{a, e\}, \{b\} \rangle$	$\langle \{a, c, e\}, \{b, d\} \rangle$
smodels	$\langle \emptyset, \emptyset \rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b\}\rangle$
select	$\langle \{a\}, \{b\}\rangle$	$\langle \{a, e\}, \{b\}\rangle$
expand	$\langle \{a, e\}, \{b\}\rangle$	$\langle \{a, c, e\}, \{b, d\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b\}\rangle$
select	$\langle \{a\}, \{b\}\rangle$	$\langle \{a, e\}, \{b\}\rangle$
expand	$\langle \{a, e\}, \{b\}\rangle$	$\langle \{a, c, e\}, \{b, d\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b\}\rangle$
select	$\langle \{a\}, \{b\}\rangle$	$\langle \{a, e\}, \{b\}\rangle$
expand	$\langle \{a, e\}, \{b\}\rangle$	$\langle \{a, c, e\}, \{b, d\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b\}\rangle$
select	$\langle \{a\}, \{b\}\rangle$	$\langle \{a, e\}, \{b\}\rangle$
expand	$\langle \{a, e\}, \{b\}\rangle$	$\langle \{a, c, e\}, \{b, d\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c, e\}$

Relationship with supported models

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **atleast**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is a supported model of Π

Assuming **atmost**$(\langle T, F\rangle) = \emptyset$ for all $\langle T, F\rangle$,
we can apply **smodels** to compute supported models !

Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b\}\rangle$
select	$\langle \{a\}, \{b\}\rangle$	$\langle \{a, e\}, \{b\}\rangle$
expand	$\langle \{a, e\}, \{b\}\rangle$	$\langle \{a, c, e\}, \{b, d\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c, e\}$

(Non-)cyclic derivations

- Cyclic derivations are causing the mismatch between supported models and answer sets.
- Atoms in an answer set can be "derived" from a program in a finite number of steps.
- Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps.
  - ☞ But they do not contradict the completion of a program.

(Non-)cyclic derivations

- Cyclic derivations are causing the mismatch between supported models and answer sets.
- Atoms in an answer set can be "derived" from a program in a finite number of steps.
- Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps.
  - ☞ But they do not contradict the completion of a program.

Non-cyclic derivations

Let X be an answer set of normal logic program Π.

- For every atom $A \in X$, there is a finite sequence of positive rules

$$\langle r_1, \ldots, r_n \rangle$$

such that

1. $head(r_1) = A$,
2. $body^+(r_i) \subseteq \{head(r_j) \mid i < j \leq n\}$ for $1 \leq i \leq n$,
3. $r_i \in \Pi^X$ for $1 \leq i \leq n$.

- That is, each atom of X has a non-cyclic derivation from Π^X.
- Is a derivable from program $\{a \leftarrow b,\ b \leftarrow a\}$?

Non-cyclic derivations

Let X be an answer set of normal logic program Π.

- For every atom $A \in X$, there is a finite sequence of positive rules

  $$\langle r_1, \ldots, r_n \rangle$$

  such that
  1. $head(r_1) = A$,
  2. $body^+(r_i) \subseteq \{head(r_j) \mid i < j \leq n\}$ for $1 \leq i \leq n$,
  3. $r_i \in \Pi^X$ for $1 \leq i \leq n$.
- That is, each atom of X has a non-cyclic derivation from Π^X.
- Is a derivable from program $\{a \leftarrow b,\ b \leftarrow a\}$?

Non-cyclic derivations

Let X be an answer set of normal logic program Π.

- For every atom $A \in X$, there is a finite sequence of positive rules

$$\langle r_1, \ldots, r_n \rangle$$

such that

1. $head(r_1) = A$,
2. $body^+(r_i) \subseteq \{head(r_j) \mid i < j \leq n\}$ for $1 \leq i \leq n$,
3. $r_i \in \Pi^X$ for $1 \leq i \leq n$.

- That is, each atom of X has a non-cyclic derivation from Π^X.
- Is a derivable from program $\{a \leftarrow b,\ b \leftarrow a\}$?

Positive atom dependency graph

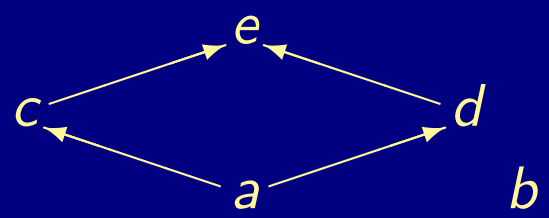
Let Π be a normal logic program.

The positive atom dependency graph of Π is a directed graph $G(\Pi) = (V, E)$ such that

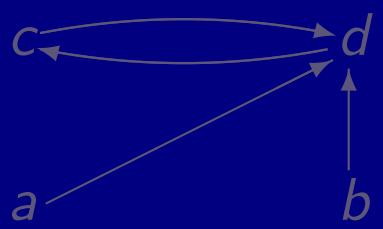
1. $V = atom(\Pi)$ and
2. $E = \{(p, q) \mid r \in \Pi, p \in body^+(r), head(r) = q\}$.

Examples

$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$

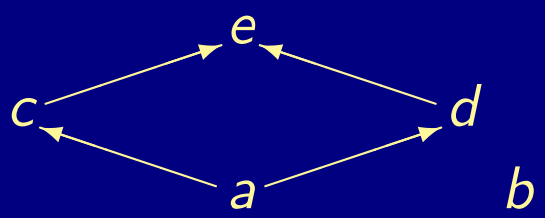


$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$

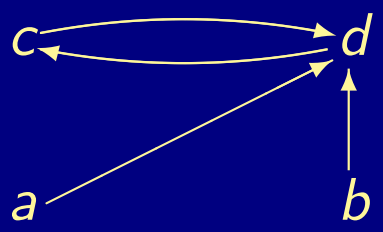


Examples

$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$



Tight programs

[20]

- A normal logic program Π is tight iff $G(\Pi)$ is acyclic.
- For example, Π_2 is tight, whereas Π_3 is not.
- If a normal logic program Π is tight, then

  X is an answer set of Π iff X is a model of $\mathit{Comp}(\Pi)$.

  That is, for tight programs, answer sets and supported models coincide.
- Also, for tight programs, Φ_Π is sufficient for propagation.

Tight programs

[20]

- A normal logic program Π is tight iff $G(\Pi)$ is acyclic.
- For example, Π_2 is tight, whereas Π_3 is not.
- If a normal logic program Π is tight, then

  X is an answer set of Π iff X is a model of $\mathit{Comp}(\Pi)$.

  That is, for tight programs, answer sets and supported models coincide.
- Also, for tight programs, Φ_Π is sufficient for propagation.

Tight programs

[20]

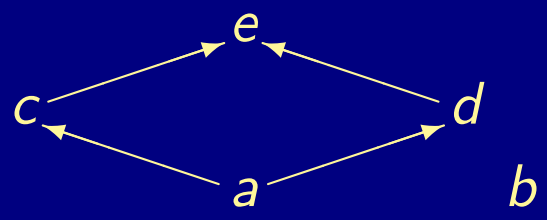
- A normal logic program Π is tight iff $G(\Pi)$ is acyclic.
- For example, Π_2 is tight, whereas Π_3 is not.
- If a normal logic program Π is tight, then

  X is an answer set of Π iff X is a model of $\mathit{Comp}(\Pi)$.

  That is, for tight programs, answer sets and supported models coincide.
- Also, for tight programs, $\mathbf{\Phi}_\Pi$ is sufficient for propagation.

(Non-)tight programs: Examples

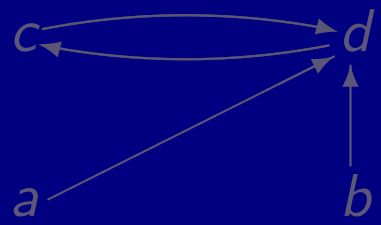
$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



Answer sets: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

Supported models: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$

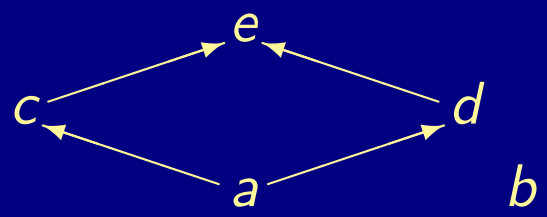


Answer sets: $\{\{a\}, \{b, c, d\}\}$

Supported models: $\{\{a\}, \{b, c, d\}, \{a, c, d\}\}$

(Non-)tight programs: Examples

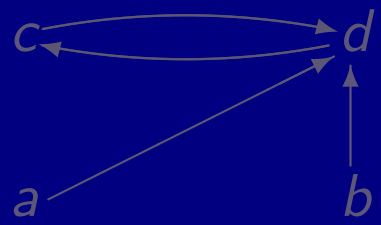
$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



Answer sets: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

Supported models: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$

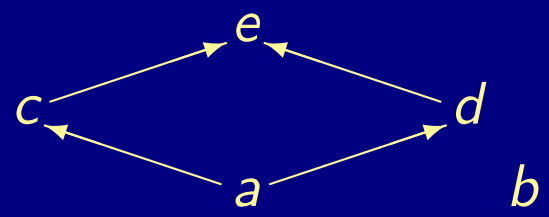


Answer sets: $\{\{a\}, \{b, c, d\}\}$

Supported models: $\{\{a\}, \{b, c, d\}, \{a, c, d\}\}$

(Non-)tight programs: Examples

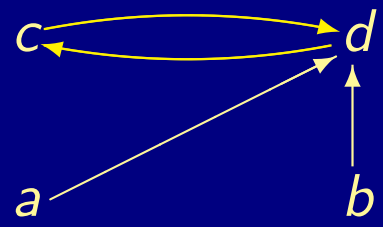
$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



Answer sets: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

Supported models: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$

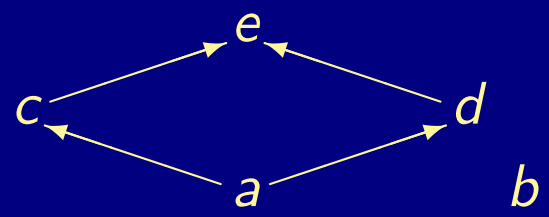


Answer sets: $\{\{a\}, \{b, c, d\}\}$

Supported models: $\{\{a\}, \{b, c, d\}, \{a, c, d\}\}$

(Non-)tight programs: Examples

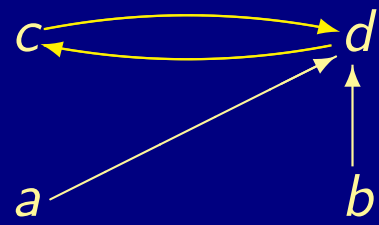
$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



Answer sets: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

Supported models: $\{\{a, c\}, \{a, d, e\}, \{b\}\}$

$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$



Answer sets: $\{\{a\}, \{b, c, d\}\}$

Supported models: $\{\{a\}, \{b, c, d\}, \{a, c, d\}\}$

Unfounded Sets
Overview

46 Definitions

47 Well-Founded Operator

48 Implementation via smodels

49 Loops and Loop Formulas

Unfounded sets

[77]

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
A set $U \subseteq atom(\Pi)$ is an unfounded set of Π with respect to $\langle T, F \rangle$ if,
for each rule $r \in \Pi$, we have

1. $head(r) \notin U$,
2. $body^+(r) \cap F \neq \emptyset$ or $body^-(r) \cap T \neq \emptyset$, or
3. $body^+(r) \cap U \neq \emptyset$.

- Intuitively, $\langle T, F \rangle$ is what we already know about Π.
- Rules satisfying Condition 1 or 2 are not usable for further derivations.
- Condition 3 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true.

Unfounded sets

[77]

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
A set $U \subseteq atom(\Pi)$ is an unfounded set of Π with respect to $\langle T, F \rangle$ if, for each rule $r \in \Pi$, we have

1. $head(r) \notin U$,
2. $body^+(r) \cap F \neq \emptyset$ or $body^-(r) \cap T \neq \emptyset$, or
3. $body^+(r) \cap U \neq \emptyset$.

- Intuitively, $\langle T, F \rangle$ is what we already know about Π.
- Rules satisfying Condition 1 or 2 are not usable for further derivations.
- Condition 3 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true.

Unfounded sets

[77]

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
A set $U \subseteq atom(\Pi)$ is an unfounded set of Π with respect to $\langle T, F \rangle$ if, for each rule $r \in \Pi$, we have

1. $head(r) \notin U$,
2. $body^+(r) \cap F \neq \emptyset$ or $body^-(r) \cap T \neq \emptyset$, or
3. $body^+(r) \cap U \neq \emptyset$.

- Intuitively, $\langle T, F \rangle$ is what we already know about Π.
- Rules satisfying Condition 1 or 2 are not usable for further derivations.
- Condition 3 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true.

Unfounded sets

[77]

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
A set $U \subseteq atom(\Pi)$ is an unfounded set of Π with respect to $\langle T, F \rangle$ if, for each rule $r \in \Pi$, we have

1. $head(r) \notin U$,
2. $body^+(r) \cap F \neq \emptyset$ or $body^-(r) \cap T \neq \emptyset$, or
3. $body^+(r) \cap U \neq \emptyset$.

- Intuitively, $\langle T, F \rangle$ is what we already know about Π.
- Rules satisfying Condition 1 or 2 are not usable for further derivations.
- Condition 3 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true.

Unfounded sets

[77]

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
A set $U \subseteq atom(\Pi)$ is an unfounded set of Π with respect to $\langle T, F \rangle$ if, for each rule $r \in \Pi$, we have

1. $head(r) \notin U$,
2. $body^+(r) \cap F \neq \emptyset$ or $body^-(r) \cap T \neq \emptyset$, or
3. $body^+(r) \cap U \neq \emptyset$.

- Intuitively, $\langle T, F \rangle$ is what we already know about Π.
- Rules satisfying Condition 1 or 2 are not usable for further derivations.
- Condition 3 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true.

Unfounded sets

[77]

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
A set $U \subseteq atom(\Pi)$ is an unfounded set of Π with respect to $\langle T, F \rangle$ if, for each rule $r \in \Pi$, we have

1. $head(r) \notin U$,
2. $body^+(r) \cap F \neq \emptyset$ or $body^-(r) \cap T \neq \emptyset$, or
3. $body^+(r) \cap U \neq \emptyset$.

- Intuitively, $\langle T, F \rangle$ is what we already know about Π.
- Rules satisfying Condition 1 or 2 are not usable for further derivations.
- Condition 3 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Example

$$\Pi = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition).
- $\{a\}$ is not an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a\}$ is an unfounded set of Π wrt $\langle \emptyset, \{b\} \rangle$.
- $\{a\}$ is not an unfounded set of Π wrt $\langle \{b\}, \emptyset \rangle$.

  ➥ Analogously for $\{b\}$.
- $\{a, b\}$ is an unfounded set of Π wrt $\langle \emptyset, \emptyset \rangle$.
- $\{a, b\}$ is an unfounded set of Π wrt any partial interpretation.

Greatest unfounded sets

Observation The union of two unfounded sets is an unfounded set.

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
The greatest unfounded set of Π with respect to $\langle T, F \rangle$, denoted by $\mathbf{U}_\Pi\langle T, F \rangle$, is the union of all unfounded sets of Π with respect to $\langle T, F \rangle$.

- Alternatively, we may define

$$\mathbf{U}_\Pi\langle T, F \rangle = atom(\Pi) \setminus Cn(\{r \in \Pi \mid body^+(r) \cap F = \emptyset\}^T).$$

- Observe that $Cn(\{r \in \Pi \mid body^+(r) \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from Π wrt $\langle T, F \rangle$.

Greatest unfounded sets

Observation The union of two unfounded sets is an unfounded set.

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.
The greatest unfounded set of Π with respect to $\langle T, F\rangle$, denoted by $\mathbf{U}_\Pi\langle T, F\rangle$, is the union of all unfounded sets of Π with respect to $\langle T, F\rangle$.

- Alternatively, we may define

$$\mathbf{U}_\Pi\langle T, F\rangle = atom(\Pi) \setminus Cn(\{r \in \Pi \mid body^+(r) \cap F = \emptyset\}^T).$$

- Observe that $Cn(\{r \in \Pi \mid body^+(r) \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from Π wrt $\langle T, F\rangle$.

Greatest unfounded sets

Observation The union of two unfounded sets is an unfounded set.

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.
The greatest unfounded set of Π with respect to $\langle T, F\rangle$, denoted by $\mathbf{U}_\Pi\langle T, F\rangle$, is the union of all unfounded sets of Π with respect to $\langle T, F\rangle$.

- Alternatively, we may define

$$\mathbf{U}_\Pi\langle T, F\rangle = \mathit{atom}(\Pi) \setminus \mathit{Cn}(\{r \in \Pi \mid \mathit{body}^+(r) \cap F = \emptyset\}^T).$$

- Observe that $\mathit{Cn}(\{r \in \Pi \mid \mathit{body}^+(r) \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from Π wrt $\langle T, F\rangle$.

Well-founded operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

Observation Condition 2 (in the definition of an unfounded set) corresponds to set $\mathbf{F}_\Pi\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_\Pi\langle T, F\rangle$.

Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_\Pi$.
That is,

- keep definition of $\mathbf{T}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ and
- replace $\mathbf{F}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ by $\mathbf{U}_\Pi\langle T, F\rangle$.

In words, an atom must be *false*
if it belongs to the greatest unfounded set.

Definition $\mathbf{\Omega}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{U}_\Pi\langle T, F\rangle\rangle$

Property $\mathbf{\Phi}_\Pi\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_\Pi\langle T, F\rangle$

Well-founded operator

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.

Observation Condition 2 (in the definition of an unfounded set) corresponds to set $\mathbf{F}_\Pi\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_\Pi\langle T, F\rangle$.

Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_\Pi$.
That is,

- keep definition of $\mathbf{T}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ and
- replace $\mathbf{F}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ by $\mathbf{U}_\Pi\langle T, F\rangle$.

In words, an atom must be *false*
if it belongs to the greatest unfounded set.

Definition $\mathbf{\Omega}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{U}_\Pi\langle T, F\rangle\rangle$

Property $\mathbf{\Phi}_\Pi\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_\Pi\langle T, F\rangle$

Well-founded operator

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.

Observation Condition 2 (in the definition of an unfounded set) corresponds to set $\mathbf{F}_\Pi\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_\Pi\langle T, F\rangle$.

Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_\Pi$.
That is,

- keep definition of $\mathbf{T}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ and
- replace $\mathbf{F}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ by $\mathbf{U}_\Pi\langle T, F\rangle$.

In words, an atom must be *false*
if it belongs to the greatest unfounded set.

Definition $\mathbf{\Omega}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{U}_\Pi\langle T, F\rangle\rangle$

Property $\mathbf{\Phi}_\Pi\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_\Pi\langle T, F\rangle$

Well-founded operator

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.

Observation Condition 2 (in the definition of an unfounded set) corresponds to set $\mathbf{F}_\Pi\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_\Pi\langle T, F\rangle$.

Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_\Pi$.
That is,

- keep definition of $\mathbf{T}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ and
- replace $\mathbf{F}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ by $\mathbf{U}_\Pi\langle T, F\rangle$.

In words, an atom must be *false*
if it belongs to the greatest unfounded set.

Definition $\mathbf{\Omega}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{U}_\Pi\langle T, F\rangle\rangle$

Property $\mathbf{\Phi}_\Pi\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_\Pi\langle T, F\rangle$

Well-founded operator

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.

Observation Condition 2 (in the definition of an unfounded set) corresponds to set $\mathbf{F}_\Pi\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_\Pi\langle T, F\rangle$.

Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_\Pi$.
That is,

- keep definition of $\mathbf{T}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ and
- replace $\mathbf{F}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ by $\mathbf{U}_\Pi\langle T, F\rangle$.

In words, an atom must be *false*
if it belongs to the greatest unfounded set.

Definition $\mathbf{\Omega}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{U}_\Pi\langle T, F\rangle\rangle$

Property $\mathbf{\Phi}_\Pi\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_\Pi\langle T, F\rangle$

Well-founded operator

Let Π be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation.

Observation Condition 2 (in the definition of an unfounded set) corresponds to set $\mathbf{F}_\Pi\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_\Pi\langle T, F\rangle$.

Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_\Pi$.
That is,

- keep definition of $\mathbf{T}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ and
- replace $\mathbf{F}_\Pi\langle T, F\rangle$ from $\mathbf{\Phi}_\Pi\langle T, F\rangle$ by $\mathbf{U}_\Pi\langle T, F\rangle$.

In words, an atom must be *false*
if it belongs to the greatest unfounded set.

Definition $\mathbf{\Omega}_\Pi\langle T, F\rangle = \langle \mathbf{T}_\Pi\langle T, F\rangle, \mathbf{U}_\Pi\langle T, F\rangle\rangle$

Property $\mathbf{\Phi}_\Pi\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_\Pi\langle T, F\rangle$

Well-founded operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Omega}_{\Pi_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{aligned} \mathbf{\Omega}_{\Pi_1}\langle \{c\}, \emptyset \rangle &= \langle \{a\}, \{d\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{d\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a, c\}, \{b, e\} \rangle &= \langle \{a\}, \{b, d, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{b, d, e\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ &\vdots \end{aligned}$$

Well-founded operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Omega}_{\Pi_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{aligned} \mathbf{\Omega}_{\Pi_1}\langle \{c\}, \emptyset \rangle &= \langle \{a\}, \{d\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{d\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a, c\}, \{b, e\} \rangle &= \langle \{a\}, \{b, d, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{b, d, e\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ &\vdots \end{aligned}$$

Well-founded operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Omega}_{\Pi_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{aligned} \mathbf{\Omega}_{\Pi_1}\langle \{c\}, \emptyset \rangle &= \langle \{a\}, \{d\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{d\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a, c\}, \{b, e\} \rangle &= \langle \{a\}, \{b, d, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{b, d, e\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ &\vdots \end{aligned}$$

Well-founded operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Omega}_{\Pi_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{array}{rcl} \mathbf{\Omega}_{\Pi_1}\langle \{c\}, \emptyset \rangle & = & \langle \{a\}, \{d\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{d\} \rangle & = & \langle \{a, c\}, \{b, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a, c\}, \{b, e\} \rangle & = & \langle \{a\}, \{b, d, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1}\langle \{a\}, \{b, d, e\} \rangle & = & \langle \{a, c\}, \{b, e\} \rangle \\ & \vdots & \end{array}$$

Well-founded operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Omega}_{\Pi_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{aligned} \mathbf{\Omega}_{\Pi_1} \langle \{c\}, \emptyset \rangle &= \langle \{a\}, \{d\} \rangle \\ \mathbf{\Omega}_{\Pi_1} \langle \{a\}, \{d\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1} \langle \{a, c\}, \{b, e\} \rangle &= \langle \{a\}, \{b, d, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1} \langle \{a\}, \{b, d, e\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ &\vdots \end{aligned}$$

Well-founded operator: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Let's iterate $\mathbf{\Omega}_{\Pi_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{aligned} \mathbf{\Omega}_{\Pi_1} \langle \{c\}, \emptyset \rangle &= \langle \{a\}, \{d\} \rangle \\ \mathbf{\Omega}_{\Pi_1} \langle \{a\}, \{d\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1} \langle \{a, c\}, \{b, e\} \rangle &= \langle \{a\}, \{b, d, e\} \rangle \\ \mathbf{\Omega}_{\Pi_1} \langle \{a\}, \{b, d, e\} \rangle &= \langle \{a, c\}, \{b, e\} \rangle \\ &\vdots \end{aligned}$$

Well-founded semantics

Define the iterative variant of $\mathbf{\Omega}_\Pi$ analogously to $\mathbf{\Phi}_\Pi$:

$$\mathbf{\Omega}^0_\Pi\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Omega}^{i+1}_\Pi\langle T, F\rangle = \mathbf{\Omega}_\Pi\mathbf{\Omega}^i_\Pi\langle T, F\rangle$$

Define the well-founded semantics of a normal logic program Π as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Omega}^i_\Pi\langle\emptyset, \emptyset\rangle$$

Well-founded semantics

Define the iterative variant of $\mathbf{\Omega}_\Pi$ analogously to $\mathbf{\Phi}_\Pi$:

$$\mathbf{\Omega}_\Pi^0\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Omega}_\Pi^{i+1}\langle T, F\rangle = \mathbf{\Omega}_\Pi\mathbf{\Omega}_\Pi^i\langle T, F\rangle$$

Define the well-founded semantics of a normal logic program Π as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Omega}_\Pi^i\langle\emptyset, \emptyset\rangle$$

Well-founded semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \Omega^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \Omega^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \Omega^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle \\ \Omega^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\{b,e\}\rangle & = & \langle\{a\},\{b,e\}\rangle \\ \bigsqcup_{i\geq 0}\Omega^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle & & \end{array}$$

Well-founded semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcll} \Omega^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & \langle\emptyset,\emptyset\rangle \\ \Omega^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\emptyset,\emptyset\rangle & = \langle\{a\},\emptyset\rangle \\ \Omega^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\emptyset\rangle & = \langle\{a\},\{b,e\}\rangle \\ \Omega^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\{b,e\}\rangle & = \langle\{a\},\{b,e\}\rangle \\ \bigsqcup_{i\geq 0}\Omega^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle & \end{array}$$

Well-founded semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcll} \Omega^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & \langle\emptyset,\emptyset\rangle \\ \Omega^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\emptyset,\emptyset\rangle & = \langle\{a\},\emptyset\rangle \\ \Omega^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\emptyset\rangle & = \langle\{a\},\{b,e\}\rangle \\ \Omega^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\{b,e\}\rangle & = \langle\{a\},\{b,e\}\rangle \\ \bigsqcup_{i\geq 0}\Omega^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle & \end{array}$$

Well-founded semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcll} \mathbf{\Omega}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Omega}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = \langle\{a\},\emptyset\rangle \\ \mathbf{\Omega}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}_{\Pi_1}\langle\{a\},\emptyset\rangle & = \langle\{a\},\{b,e\}\rangle \\ \mathbf{\Omega}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}_{\Pi_1}\langle\{a\},\{b,e\}\rangle & = \langle\{a\},\{b,e\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Omega}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle & \end{array}$$

Well-founded semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcll} \mathbf{\Omega}^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Omega}^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}_{\Pi_1}\langle\emptyset,\emptyset\rangle & = \langle\{a\},\emptyset\rangle \\ \mathbf{\Omega}^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}_{\Pi_1}\langle\{a\},\emptyset\rangle & = \langle\{a\},\{b,e\}\rangle \\ \mathbf{\Omega}^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}_{\Pi_1}\langle\{a\},\{b,e\}\rangle & = \langle\{a\},\{b,e\}\rangle \\ \bigsqcup_{i\geq 0}\mathbf{\Omega}^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle & \end{array}$$

Well-founded semantics: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

$$\begin{array}{rcccl} \Omega^0_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \Omega^1_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\emptyset\rangle \\ \Omega^2_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle \\ \Omega^3_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \Omega_{\Pi_1}\langle\{a\},\{b,e\}\rangle & = & \langle\{a\},\{b,e\}\rangle \\ \bigsqcup_{i\geq 0}\Omega^i_{\Pi_1}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e\}\rangle & & \end{array}$$

Well-founded semantics: Properties

Let Π be a normal logic program.

- $\mathbf{\Omega}_\Pi\langle\emptyset,\emptyset\rangle$ is monotonic.
  That is, $\mathbf{\Omega}^i_\Pi\langle\emptyset,\emptyset\rangle \sqsubseteq \mathbf{\Omega}^{i+1}_\Pi\langle\emptyset,\emptyset\rangle$.
- The well-founded semantics of Π is
  - not conflicting,
  - and generally not total.
- We have $\bigsqcup_{i\geq 0} \mathbf{\Phi}^i_\Pi\langle\emptyset,\emptyset\rangle \sqsubseteq \bigsqcup_{i\geq 0} \mathbf{\Omega}^i_\Pi\langle\emptyset,\emptyset\rangle$.

Well-founded fixpoints

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
Define $\langle T, F \rangle$ as a well-founded fixpoint of Π if $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T, F \rangle$.

- The well-founded semantics is the $\sqsubseteq$-least well-founded fixpoint of Π.
- Any other well-founded fixpoint extends the well-founded semantics.
- Total well-founded fixpoints correspond to answer sets.

Well-founded fixpoints

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.
Define $\langle T, F \rangle$ as a well-founded fixpoint of Π if $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T, F \rangle$.

- The well-founded semantics is the $\sqsubseteq$-least well-founded fixpoint of Π.
- Any other well-founded fixpoint extends the well-founded semantics.
- Total well-founded fixpoints correspond to answer sets.

Well-founded fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has two total well-founded fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$

Both of them represent answer sets.

Well-founded fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has two total well-founded fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$

Both of them represent answer sets.

Well-founded fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has two total well-founded fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$

Both of them represent answer sets.

Well-founded fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has two total well-founded fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$

Both of them represent answer sets.

Well-founded fixpoints: Example

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Π_1 has two total well-founded fixpoints:

1. $\langle \{a, c\}, \{b, d, e\} \rangle$
2. $\langle \{a, d\}, \{b, c, e\} \rangle$

Both of them represent answer sets.

Properties of well-founded operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Omega}_\Pi$ is answer set preserving.

  $\mathbf{\Omega}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

Unlike $\mathbf{\Phi}_\Pi$, operator $\mathbf{\Omega}_\Pi$ is sufficient for propagation because total fixpoints correspond to answer sets.

☞ In addition to $\mathbf{\Omega}_\Pi$, most ASP-solvers apply backward propagation (cf. Page 455), originating from program completion (although this is unnecessary from a formal point of view).

Properties of well-founded operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Omega}_\Pi$ is answer set preserving.

  $\mathbf{\Omega}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

Unlike $\mathbf{\Phi}_\Pi$, operator $\mathbf{\Omega}_\Pi$ is sufficient for propagation because total fixpoints correspond to answer sets.

☞ In addition to $\mathbf{\Omega}_\Pi$, most ASP-solvers apply backward propagation (cf. Page 455), originating from program completion (although this is unnecessary from a formal point of view).

Properties of well-founded operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Omega}_\Pi$ is answer set preserving.
  - ➥ $\mathbf{\Omega}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

Unlike $\mathbf{\Phi}_\Pi$, operator $\mathbf{\Omega}_\Pi$ is sufficient for propagation because total fixpoints correspond to answer sets.

- ☞ In addition to $\mathbf{\Omega}_\Pi$, most ASP-solvers apply backward propagation (cf. Page 455), originating from program completion (although this is unnecessary from a formal point of view).

Properties of well-founded operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Omega}_\Pi$ is answer set preserving.
  - ➥ $\mathbf{\Omega}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

Unlike $\mathbf{\Phi}_\Pi$, operator $\mathbf{\Omega}_\Pi$ is sufficient for propagation because total fixpoints correspond to answer sets.

- ☞ In addition to $\mathbf{\Omega}_\Pi$, most ASP-solvers apply backward propagation (cf. Page 455), originating from program completion (although this is unnecessary from a formal point of view).

Properties of well-founded operator

Let Π be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation.

- Let $\mathbf{\Omega}_\Pi \langle T, F \rangle = \langle T', F' \rangle$.
  If X is an answer set of Π such that $T \subseteq X$ and $X \cap F = \emptyset$,
  then $T' \subseteq X$ and $X \cap F' = \emptyset$.
- That is, $\mathbf{\Omega}_\Pi$ is answer set preserving.
  - ➥ $\mathbf{\Omega}_\Pi$ can be used for approximating answer sets and so for propagation in ASP-solvers.

Unlike $\mathbf{\Phi}_\Pi$, operator $\mathbf{\Omega}_\Pi$ is sufficient for propagation because total fixpoints correspond to answer sets.

- ☞ In addition to $\mathbf{\Omega}_\Pi$, most ASP-solvers apply backward propagation (cf. Page 455), originating from program completion (although this is unnecessary from a formal point of view).

Rebuilding **atmost**($\langle T, F \rangle$)

from (greatest) unfounded sets

return $\mathbf{U}_\Pi\langle T, F \rangle$

Rebuilding **atmost**($\langle T, F \rangle$)

from (greatest) unfounded sets

return $\mathbf{U}_\Pi\langle T, F \rangle$

Recalling **expand**

Global: Normal logic program Π

expand($\langle T, F \rangle$**)**

repeat

$\langle T, F \rangle \leftarrow$ **atleast(**$\langle T, F \rangle$**)**

if $\langle T, F \rangle$ is conflicting **then return** $\langle T, F \rangle$

else

$F' \leftarrow F$

$F \leftarrow F \cup$ **atmost(**$\langle T, F \rangle$**)**

until $F = F'$

return $\langle T, F \rangle$

- **atleast(**$\langle T, F \rangle$**)** derives deterministic consequences from Clark's completion
- **atmost(**$\langle T, F \rangle$**)** derives deterministic consequences from unfounded sets

Relationship with well-founded semantics

Let Π be a normal logic program.

- **expand**$(\langle\emptyset,\emptyset\rangle) = \bigsqcup_{i\geq 0}\mathbf{\Omega}^i_\Pi\langle\emptyset,\emptyset\rangle$

☞ That is, **expand** is basically an implementation of well-founded semantics !

☞ Additional backward propagation in **atleast** prunes the search space further !

Relationship with well-founded semantics

Let Π be a normal logic program.

- **expand(**$\langle \emptyset, \emptyset \rangle$**)** $= \bigsqcup_{i \geq 0} \mathbf{\Omega}^i_\Pi \langle \emptyset, \emptyset \rangle$

☞ That is, **expand** is basically an implementation of well-founded semantics !

☞ Additional backward propagation in **atleast** prunes the search space further !

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Relationship with answer sets

Let Π be a normal logic program and $\langle T, F\rangle$ a total interpretation.

- **expand**$(\langle T, F\rangle) = \langle T, F\rangle$ iff T is an answer set of Π

Given **atmost**$(\langle T, F\rangle) = \mathbf{U}_\Pi\langle T, F\rangle$,
we can apply **smodels** to compute answer sets !
Reconsider:

$$\Pi_1 = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, not\ d & e \leftarrow b \\ b \leftarrow not\ a & d \leftarrow not\ c, not\ e & e \leftarrow e \end{array} \right\}$$

Call	Interpretation	Result
smodels	$\langle \emptyset, \emptyset\rangle$	
expand	$\langle \emptyset, \emptyset\rangle$	$\langle \{a\}, \{b, e\}\rangle$
select	$\langle \{a\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, e\}\rangle$
expand	$\langle \{a, c\}, \{b, e\}\rangle$	$\langle \{a, c\}, \{b, d, e\}\rangle$
smodels	$\langle \emptyset, \emptyset\rangle$	$\{a, c\}$

Additional remarks on smodels [73]

The smodels implementation also features:

- Extended rules
  - Cardinality constraints
  - Weight constraints
- Optimiziation via *minimize* and *maximize*
- Efficient counter-based propagation
- Lazy implementation of **atmost** based on "source pointers"
- Failed-literal detection, also called lookahead, for stronger propagation

Characterizing non-cyclic derivations

An alternative approach

Question Is there a propositional formula $F(\Pi)$ such that the models of $F(\Pi)$ correspond to the answer sets of Π ?

- ☞ If we consider the completion of a program, $Comp(\Pi)$, then the problem boils down to eliminating the circular support of atoms that are true in the supported models of Π.

Idea Add formulas to $Comp(\Pi)$ that prohibit circular support of sets of atoms.

- ☞ Circular support between atoms p and q is possible if p has a path to q and q has a path to p in a program's positive atom dependency graph.

Characterizing non-cyclic derivations

An alternative approach

Question Is there a propositional formula $F(\Pi)$ such that the models of $F(\Pi)$ correspond to the answer sets of Π ?

- ☞ If we consider the completion of a program, $Comp(\Pi)$, then the problem boils down to eliminating the circular support of atoms that are true in the supported models of Π.

Idea Add formulas to $Comp(\Pi)$ that prohibit circular support of sets of atoms.

- ☞ Circular support between atoms p and q is possible if p has a path to q and q has a path to p in a program's positive atom dependency graph.

Characterizing non-cyclic derivations

An alternative approach

Question Is there a propositional formula $F(\Pi)$ such that the models of $F(\Pi)$ correspond to the answer sets of Π ?

- ☞ If we consider the completion of a program, $Comp(\Pi)$, then the problem boils down to eliminating the circular support of atoms that are true in the supported models of Π.

Idea Add formulas to $Comp(\Pi)$ that prohibit circular support of sets of atoms.

- ☞ Circular support between atoms p and q is possible if p has a path to q and q has a path to p in a program's positive atom dependency graph.

Characterizing non-cyclic derivations

An alternative approach

Question Is there a propositional formula $F(\Pi)$ such that the models of $F(\Pi)$ correspond to the answer sets of Π ?

- ☞ If we consider the completion of a program, $Comp(\Pi)$, then the problem boils down to eliminating the circular support of atoms that are true in the supported models of Π.

Idea Add formulas to $Comp(\Pi)$ that prohibit circular support of sets of atoms.

- ☞ Circular support between atoms p and q is possible if p has a path to q and q has a path to p in a program's positive atom dependency graph.

Loops

[57]

Let Π be a normal logic program, and
let $G(\Pi) = (atom(\Pi), E)$ be the positive atom dependency graph of Π.

- A set $\emptyset \subset L \subseteq atom(\Pi)$ is a loop of Π if it induces a non-trivial strongly connected subgraph of $G(\Pi)$.
- That is, each pair of atoms in L is connected by a path of non-zero length in $(L, E \cap (L \times L))$.
- We denote the set of all loops of Π by $Loop(\Pi)$.

Observation Program Π is tight iff $Loop(\Pi) = \emptyset$.

Loops

[57]

Let Π be a normal logic program, and
let $G(\Pi) = (atom(\Pi), E)$ be the positive atom dependency graph of Π.

- A set $\emptyset \subset L \subseteq atom(\Pi)$ is a loop of Π if it induces a non-trivial strongly connected subgraph of $G(\Pi)$.
- That is, each pair of atoms in L is connected by a path of non-zero length in $(L, E \cap (L \times L))$.
- We denote the set of all loops of Π by $Loop(\Pi)$.

Observation Program Π is tight iff $Loop(\Pi) = \emptyset$.

Loop formulas

Let Π be a normal logic program.

- For $L \subseteq atom(\Pi)$, define the external supports of L for Π as

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

- The (disjunctive) loop formula of L for Π is

$$\begin{aligned} LF_\Pi(L) &= \left(\bigvee_{A \in L} A\right) \rightarrow \left(\bigvee_{r \in ES_\Pi(L)} Comp(body(r))\right) \\ &\equiv \left(\bigwedge_{r \in ES_\Pi(L)} \neg Comp(body(r))\right) \rightarrow \left(\bigwedge_{A \in L} \neg A\right). \end{aligned}$$

  - ☞ The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported.

- Define

$$LF(\Pi) = \{\ LF_\Pi(L) \mid L \in Loop(\Pi)\ \}.$$

Loop formulas

Let Π be a normal logic program.

- For $L \subseteq atom(\Pi)$, define the external supports of L for Π as

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

- The (disjunctive) loop formula of L for Π is

$$\begin{aligned} LF_\Pi(L) &= \left(\bigvee_{A \in L} A\right) \rightarrow \left(\bigvee_{r \in ES_\Pi(L)} Comp(body(r))\right) \\ &\equiv \left(\bigwedge_{r \in ES_\Pi(L)} \neg Comp(body(r))\right) \rightarrow \left(\bigwedge_{A \in L} \neg A\right). \end{aligned}$$

  - ☞ The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported.

- Define

$$LF(\Pi) = \{\ LF_\Pi(L) \mid L \in Loop(\Pi)\ \}.$$

Loop formulas

Let Π be a normal logic program.

- For $L \subseteq atom(\Pi)$, define the external supports of L for Π as

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

- The (disjunctive) loop formula of L for Π is

$$\begin{aligned} LF_\Pi(L) &= \left(\bigvee_{A \in L} A\right) \rightarrow \left(\bigvee_{r \in ES_\Pi(L)} Comp(body(r))\right) \\ &\equiv \left(\bigwedge_{r \in ES_\Pi(L)} \neg Comp(body(r))\right) \rightarrow \left(\bigwedge_{A \in L} \neg A\right). \end{aligned}$$

  - ☞ The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported.

- Define

$$LF(\Pi) = \{\ LF_\Pi(L) \mid L \in Loop(\Pi)\ \}.$$

Loop formulas

Let Π be a normal logic program.

- For $L \subseteq atom(\Pi)$, define the external supports of L for Π as

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

- The (disjunctive) loop formula of L for Π is

$$\begin{aligned} LF_\Pi(L) &= \left(\bigvee_{A \in L} A\right) \rightarrow \left(\bigvee_{r \in ES_\Pi(L)} Comp(body(r))\right) \\ &\equiv \left(\bigwedge_{r \in ES_\Pi(L)} \neg Comp(body(r))\right) \rightarrow \left(\bigwedge_{A \in L} \neg A\right). \end{aligned}$$

  - ☞ The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported.

- Define

$$LF(\Pi) = \{\ LF_\Pi(L) \mid L \in Loop(\Pi)\ \}.$$

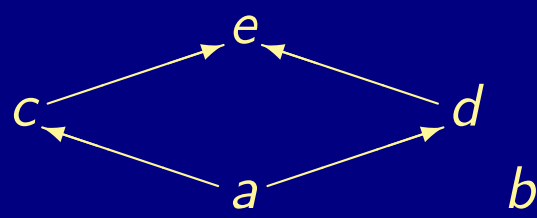
Lin-Zhao Theorem

Theorem

Let Π *be a normal logic program and* $X \subseteq atom(\Pi)$.
Then, X *is an answer set of* Π *iff* $X \models Comp(\Pi) \cup LF(\Pi)$.

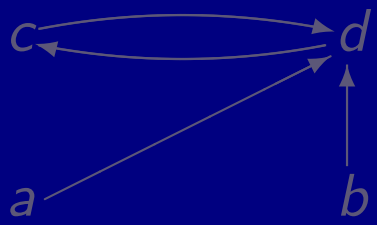
Loops and loop formulas: Examples

$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



$Loop(\Pi_2) = \emptyset$
$LF(\Pi_2) = \emptyset$

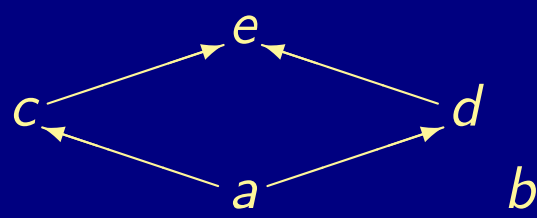
$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$



$Loop(\Pi_3) = \{\{c, d\}\}$
$LF(\Pi_3) = \{(c \vee d) \rightarrow (\neg a \vee (a \wedge b))\}$

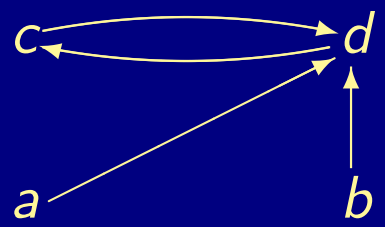
Loops and loop formulas: Examples

$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



$Loop(\Pi_2) = \emptyset$
$LF(\Pi_2) = \emptyset$

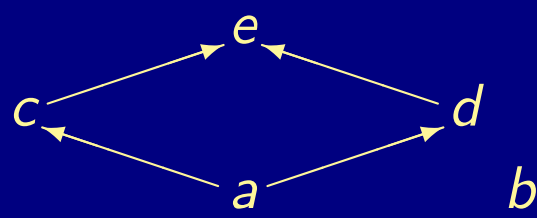
$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$



$Loop(\Pi_3) = \{\{c, d\}\}$
$LF(\Pi_3) = \{(c \vee d) \rightarrow (\neg a \vee (a \wedge b))\}$

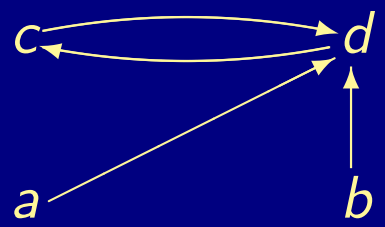
Loops and loop formulas: Examples

$$\Pi_2 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow a, not\ d & d \leftarrow a, not\ c \\ e \leftarrow c, not\ a & e \leftarrow d, not\ b \end{array} \right\}$$



$Loop(\Pi_2) = \emptyset$
$LF(\Pi_2) = \emptyset$

$$\Pi_3 = \left\{ \begin{array}{ll} a \leftarrow not\ b & b \leftarrow not\ a \\ c \leftarrow not\ a & c \leftarrow d \\ d \leftarrow a, b & d \leftarrow c \end{array} \right\}$$



$Loop(\Pi_3) = \{\{c, d\}\}$
$LF(\Pi_3) = \{(c \vee d) \rightarrow (\neg a \vee (a \wedge b))\}$

Loops and loop formulas: Properties

[49]

Let X be a supported model of normal logic program Π.

Then, X is an answer set of Π iff

- $X \models \{ LF_\Pi(U) \mid U \subseteq atom(\Pi) \}$;
- $X \models \{ LF_\Pi(U) \mid U \subseteq X \}$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi) \}$, that is, $X \models LF(\Pi)$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi), L \subseteq X \}$.
  - ➥ If X is not an answer set of Π,
    then there is a loop $L \subseteq X \setminus Cn(\Pi^X)$ such that $X \not\models LF_\Pi(L)$.

Loops and loop formulas: Properties

[49]

Let X be a supported model of normal logic program Π.

Then, X is an answer set of Π iff

- $X \models \{ LF_\Pi(U) \mid U \subseteq atom(\Pi) \}$;
- $X \models \{ LF_\Pi(U) \mid U \subseteq X \}$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi) \}$, that is, $X \models LF(\Pi)$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi), L \subseteq X \}$.
  - ➥ If X is not an answer set of Π,
    then there is a loop $L \subseteq X \setminus Cn(\Pi^X)$ such that $X \not\models LF_\Pi(L)$.

Loops and loop formulas: Properties

[49]

Let X be a supported model of normal logic program Π.

Then, X is an answer set of Π iff

- $X \models \{ LF_\Pi(U) \mid U \subseteq atom(\Pi) \}$;
- $X \models \{ LF_\Pi(U) \mid U \subseteq X \}$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi) \}$, that is, $X \models LF(\Pi)$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi), L \subseteq X \}$.
  - ➥ If X is not an answer set of Π,
    then there is a loop $L \subseteq X \setminus Cn(\Pi^X)$ such that $X \not\models LF_\Pi(L)$.

Loops and loop formulas: Properties

[49]

Let X be a supported model of normal logic program Π.

Then, X is an answer set of Π iff

- $X \models \{ LF_\Pi(U) \mid U \subseteq atom(\Pi) \}$;
- $X \models \{ LF_\Pi(U) \mid U \subseteq X \}$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi) \}$, that is, $X \models LF(\Pi)$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi), L \subseteq X \}$.
  - ➥ If X is not an answer set of Π,
    then there is a loop $L \subseteq X \setminus Cn(\Pi^X)$ such that $X \not\models LF_\Pi(L)$.

Loops and loop formulas: Properties

[49]

Let X be a supported model of normal logic program Π.

Then, X is an answer set of Π iff

- $X \models \{ LF_\Pi(U) \mid U \subseteq atom(\Pi) \}$;
- $X \models \{ LF_\Pi(U) \mid U \subseteq X \}$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi) \}$, that is, $X \models LF(\Pi)$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi), L \subseteq X \}$.
  - ➥ If X is not an answer set of Π,
    then there is a loop $L \subseteq X \setminus Cn(\Pi^X)$ such that $X \not\models LF_\Pi(L)$.

Loops and loop formulas: Properties

[49]

Let X be a supported model of normal logic program Π.

Then, X is an answer set of Π iff

- $X \models \{ LF_\Pi(U) \mid U \subseteq atom(\Pi) \}$;
- $X \models \{ LF_\Pi(U) \mid U \subseteq X \}$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi) \}$, that is, $X \models LF(\Pi)$;
- $X \models \{ LF_\Pi(L) \mid L \in Loop(\Pi), L \subseteq X \}$.
  - ➥ If X is not an answer set of Π,
    then there is a loop $L \subseteq X \setminus Cn(\Pi^X)$ such that $X \not\models LF_\Pi(L)$.

Loops and loop formulas: Properties (ctd)

[54]

If $\mathcal{P} \not\subseteq \mathcal{NC}^1/poly$,[1] then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program Π, both of the following conditions hold:

1. The propositional variables in $\mathcal{T}[\Pi]$ are a subset of $atom(\Pi)$.
2. The size of $\mathcal{T}[\Pi]$ is polynomial in the size of Π.

☞ Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case).

Observations

- Translation $Comp(\Pi) \cup LF(\Pi)$ preserves the vocabulary of Π.
- The number of loops in $Loop(\Pi)$ may be exponential in $|atom(\Pi)|$.

[1] A conjecture from the theory of complexity that is widely believed to be true.

Loops and loop formulas: Properties (ctd)

[54]

If $\mathcal{P} \not\subseteq \mathcal{NC}^1/poly$,[1] then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program Π, both of the following conditions hold:

1. The propositional variables in $\mathcal{T}[\Pi]$ are a subset of $atom(\Pi)$.
2. The size of $\mathcal{T}[\Pi]$ is polynomial in the size of Π.
   - ☞ Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case).

Observations

- Translation $Comp(\Pi) \cup LF(\Pi)$ preserves the vocabulary of Π.
- The number of loops in $Loop(\Pi)$ may be exponential in $|atom(\Pi)|$.

[1] A conjecture from the theory of complexity that is widely believed to be true.

Loops and loop formulas: Properties (ctd)

[54]

If $\mathcal{P} \not\subseteq \mathcal{NC}^1/poly$,[1] then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program Π, both of the following conditions hold:

1. The propositional variables in $\mathcal{T}[\Pi]$ are a subset of $atom(\Pi)$.
2. The size of $\mathcal{T}[\Pi]$ is polynomial in the size of Π.
   - ☞ Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case).

Observations

- Translation $Comp(\Pi) \cup LF(\Pi)$ preserves the vocabulary of Π.
- The number of loops in $Loop(\Pi)$ may be exponential in $|atom(\Pi)|$.

[1] A conjecture from the theory of complexity that is widely believed to be true.

Tableau Calculi
Overview

50 Motivation

51 Tableau Methods

52 Tableau Calculi for ASP
- Definitions
- Tableau Rules for Clark's Completion
- Tableau Rules for Unfounded Sets
- Tableau Rules for Case Analysis
- Particular Tableau Calculi
- Relative Efficiency
- Example Tableaux

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 analyzing ASP-solving

Motivation

Goal Analyze computations in ASP-solvers

Wanted A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP-solvers

Idea View answer set computations as derivations in an inference system

➥ Tableau-based proof system for analyzing ASP-solving

Motivation

Goal Analyze computations in ASP-solvers

Wanted A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP-solvers

Idea View answer set computations as derivations in an inference system

➥ Tableau-based proof system for analyzing ASP-solving

Motivation

Goal Analyze computations in ASP-solvers

Wanted A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP-solvers

Idea View answer set computations as derivations in an inference system

➥ Tableau-based proof system for analyzing ASP-solving

Tableau calculi

- Traditionally, tableau calculi are used for
  - automated theorem proving and
  - proof theoretical analysis

  in classical as well as non-classical logics.
- **General idea:** Given an input, prove some property by decomposition. Decomposition is done by applying deduction rules.
- For details, see [12].

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 $\gamma_1, \ldots, \gamma_m$ can be extended by applying tableau rules of form:

$$\frac{\gamma_1, \ldots, \gamma_m}{\begin{array}{c}\alpha_1 \\ \vdots \\ \alpha_n\end{array}} \qquad \frac{\gamma_1, \ldots, \gamma_m}{\beta_1 \mid \ldots \mid \beta_n}$$

- Rules of the former format append entries $\alpha_1, \ldots, \alpha_n$ to the branch.
- Rules of the latter format create multiple sub-branches for $\beta_1, \ldots, \beta_n$.

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 $\gamma_1, \ldots, \gamma_m$ can be extended by applying tableau rules of form:

$$\frac{\gamma_1, \ldots, \gamma_m}{\begin{array}{c}\alpha_1 \\ \vdots \\ \alpha_n\end{array}} \qquad \frac{\gamma_1, \ldots, \gamma_m}{\beta_1 \mid \ldots \mid \beta_n}$$

- Rules of the former format append entries $\alpha_1, \ldots, \alpha_n$ to the branch.
- Rules of the latter format create multiple sub-branches for $\beta_1, \ldots, \beta_n$.

Tableau calculus: Example

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

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- 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 (composed from $\neg$, $\wedge$, and $\vee$) 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 $\neg\alpha$.

Tableau calculus: Example

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

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- 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 (composed from $\neg$, $\wedge$, and $\vee$) 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 $\neg\alpha$.

Tableau calculus: Example

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

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- 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 (composed from $\neg$, $\wedge$, and $\vee$) 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 $\neg\alpha$.

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(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.

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

Tableau calculus: Example (ctd)

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

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

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

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.

Tableaux and 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 $\mathbf{T}v$ or $\mathbf{F}v$, called signed literals, where v is a variable,
  - generated by extending a tableau using deduction rules (given below).
- An entry $\mathbf{T}v$ ($\mathbf{F}v$) reflects that variable v is *true* (*false*) in a corresponding variable assignment.
  - ➥ A set of signed literals constitutes a partial assignment.
- For a normal logic program Π,
  - atoms of Π in $atom(\Pi)$ and
  - bodies of Π in $body(\Pi) = \{body(r) \mid r \in \Pi\}$

  can occur as variables in signed literals.

Tableaux and 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 $\mathbf{T}v$ or $\mathbf{F}v$, called signed literals, where v is a variable,
  - generated by extending a tableau using deduction rules (given below).
- An entry $\mathbf{T}v$ ($\mathbf{F}v$) reflects that variable v is *true* (*false*) in a corresponding variable assignment.
  - ➥ A set of signed literals constitutes a partial assignment.
- For a normal logic program Π,
  - atoms of Π in $atom(\Pi)$ and
  - bodies of Π in $body(\Pi) = \{body(r) \mid r \in \Pi\}$

  can occur as variables in signed literals.

Tableaux and 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 $\mathbf{T}v$ or $\mathbf{F}v$, called signed literals, where v is a variable,
  - generated by extending a tableau using deduction rules (given below).
- An entry $\mathbf{T}v$ ($\mathbf{F}v$) reflects that variable v is *true* (*false*) in a corresponding variable assignment.
  - ➥ A set of signed literals constitutes a partial assignment.
- For a normal logic program Π,
  - atoms of Π in $atom(\Pi)$ and
  - bodies of Π in $body(\Pi) = \{body(r) \mid r \in \Pi\}$

  can occur as variables in signed literals.

Tableau rules for ASP at a glance [35]

$$\text{(FTB)}\ \frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}} \qquad \text{(BFB)}\ \frac{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

$$\text{(FTA)}\ \frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{T}\{l_1, \ldots, l_n\}}{\mathbf{T}p} \qquad \text{(BFA)}\ \frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{F}p}{\mathbf{F}\{l_1, \ldots, l_n\}}$$

$$\text{(FFB)}\ \frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}} \qquad \text{(BTB)}\ \frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

$$\text{(FFA)}\ \frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \qquad (\S) \qquad \text{(BTA)}\ \frac{\mathbf{T}p \quad \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m}{\mathbf{T}B_i}$$

$$\text{(WFN)}\ \frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \qquad (\dagger) \qquad \text{(WFJ)}\ \frac{\mathbf{T}p \quad \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m}{\mathbf{T}B_i}$$

$$\text{(FL)}\ \frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \qquad (\ddagger) \qquad \text{(BL)}\ \frac{\mathbf{T}p \quad \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m}{\mathbf{T}B_i}$$

$$\text{(Cut}[X]\text{)}\ \frac{}{\mathbf{T}v \mid \mathbf{F}v} \qquad (\sharp[X])$$

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

More concepts

- A tableau calculus is a set of tableau rules.
- A branch in a tableau is conflicting,
  if it contains both $\mathbf{T}v$ and $\mathbf{F}v$ for some variable v.
- A branch in a tableau is total for a program Π,
  if it contains either $\mathbf{T}v$ or $\mathbf{F}v$ for each $v \in \mathit{atom}(\Pi) \cup \mathit{body}(\Pi)$.
- 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 conflicting or both total and closed.
- A tableau is complete,
  if all its branches are complete.
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program Π,
  if every branch in the tableau is conflicting.

Example

Consider the program

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

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

(Previewed) Example

$a \leftarrow$
$c \leftarrow not\ b, not\ d$
$d \leftarrow a, not\ c$

(FTB) $\mathbf{T}\emptyset$
(FTA) $\mathbf{T}a$
(FFA) $\mathbf{F}b$
(Cut[$atom(\Pi)$]) $\mathbf{T}c$ | $\mathbf{F}c$

(BTA)	$\mathbf{T}\{not\ b, not\ d\}$	
(BTB)	$\mathbf{F}d$	
(FFB)	$\mathbf{F}\{a, not\ c\}$	
		(BFA) $\mathbf{F}\{not\ b, not\ d$
		(BFB) $\mathbf{T}d$
		(FTB) $\mathbf{T}\{a, not\ c\}$

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

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow not\ b, not\ d$$
$$d \leftarrow a, not\ c$$

(FTB)		$\mathbf{T}\emptyset$	
(FTA)		$\mathbf{T}a$	
(FFA)		$\mathbf{F}b$	
(Cut[$atom(\Pi)$])	$\mathbf{T}c$		$\mathbf{F}c$
(BTA)	$\mathbf{T}\{not\ b, not\ d\}$	(BFA)	$\mathbf{F}\{not\ b, not\ d$
(BTB)	$\mathbf{F}d$	(BFB)	$\mathbf{T}d$
(FFB)	$\mathbf{F}\{a, not\ c\}$	(FTB)	$\mathbf{T}\{a, not\ c\}$

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

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow not\ b, not\ d$$
$$d \leftarrow a, not\ c$$

(FTB) $\mathbf{T}\emptyset$
(FTA) $\mathbf{T}a$
(FFA) $\mathbf{F}b$
(Cut[$atom(\Pi)$]) $\mathbf{T}c$ | $\mathbf{F}c$

(BTA) $\mathbf{T}\{not\ b, not\ d\}$
(BTB) $\mathbf{F}d$
(FFB) $\mathbf{F}\{a, not\ c\}$

(BFA) $\mathbf{F}\{not\ b, not\ d$
(BFB) $\mathbf{T}d$
(FTB) $\mathbf{T}\{a, not\ c\}$

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

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow not\ b, not\ d$$
$$d \leftarrow a, not\ c$$

$\mathbf{T}\emptyset$ (FTB)
$\mathbf{T}a$ (FTA)
$\mathbf{F}b$ (FFA)
$\mathbf{T}c$ (Cut[$atom(\Pi)$])

(BTA)	$\mathbf{T}\{not\ b, not\ d\}$	$\mathbf{F}\{not\ b, not\ d\}$ (BFA)
(BTB)	$\mathbf{F}d$	$\mathbf{T}d$ (BFB)
(FFB)	$\mathbf{F}\{a, not\ c\}$	$\mathbf{T}\{a, not\ c\}$ (FTB)

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

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow not\ b, not\ d$$
$$d \leftarrow a, not\ c$$

(FTB)		$\mathbf{T}\emptyset$		
(FTA)		$\mathbf{T}a$		
(FFA)		$\mathbf{F}b$		
(Cut[$atom(\Pi)$])		$\mathbf{T}c$		$\mathbf{F}c$
	(BTA)	$\mathbf{T}\{not\ b, not\ d\}$	(BFA)	$\mathbf{F}\{not\ b, not\ d$
	(BTB)	$\mathbf{F}d$	(BFB)	$\mathbf{T}d$
	(FFB)	$\mathbf{F}\{a, not\ c\}$	(FTB)	$\mathbf{T}\{a, not\ c\}$

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

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow not\ b, not\ d$$
$$d \leftarrow a, not\ c$$

(FTB) $\mathbf{T}\emptyset$
(FTA) $\mathbf{T}a$
(FFA) $\mathbf{F}b$
(Cut[$atom(\Pi)$]) $\mathbf{T}c$ | $\mathbf{F}c$

(BTA) $\mathbf{T}\{not\ b, not\ d\}$
(BTB) $\mathbf{F}d$
(FFB) $\mathbf{F}\{a, not\ c\}$

(BFA) $\mathbf{F}\{not\ b, not\ d\}$
(BFB) $\mathbf{T}d$
(FTB) $\mathbf{T}\{a, not\ c\}$

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

Tableau rules: Auxiliary definitions

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

$$\mathbf{t}l = \begin{cases} \mathbf{T}l & \text{if } l \text{ is an atom} \\ \mathbf{F}p & \text{if } l = \mathit{not}\ p \text{ for an atom } p \end{cases}$$

$$\mathbf{f}l = \begin{cases} \mathbf{F}l & \text{if } l \text{ is an atom} \\ \mathbf{T}p & \text{if } l = \mathit{not}\ p \text{ for an atom } p \end{cases}$$

Examples

$$\mathbf{t}p = \mathbf{T}p \qquad \mathbf{f}p = \mathbf{F}p \qquad \mathbf{t}\mathit{not}\ p = \mathbf{F}p \qquad \mathbf{f}\mathit{not}\ p = \mathbf{T}p$$

Tableau rules: Auxiliary definitions

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

$$\mathbf{t}l = \begin{cases} \mathbf{T}l & \text{if } l \text{ is an atom} \\ \mathbf{F}p & \text{if } l = not\ p \text{ for an atom } p \end{cases}$$

$$\mathbf{f}l = \begin{cases} \mathbf{F}l & \text{if } l \text{ is an atom} \\ \mathbf{T}p & \text{if } l = not\ p \text{ for an atom } p \end{cases}$$

Examples

$$\mathbf{t}p = \mathbf{T}p \qquad \mathbf{f}p = \mathbf{F}p \qquad \mathbf{t}not\ p = \mathbf{F}p \qquad \mathbf{f}not\ p = \mathbf{T}p$$

Tableau rules: Auxiliary definitions (ctd)

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

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

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

Forward True Body (FTB)

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

Consequence The body is *true*.

Tableau Rule FTB

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}}$$

Example

$$\frac{a \leftarrow b, not\ c \quad \mathbf{T}b \quad \mathbf{F}c}{\mathbf{T}\{b, not\ c\}}$$

Forward True Body (FTB)

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

Consequence The body is *true*.

Tableau Rule FTB

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}}$$

Example

$$\frac{a \leftarrow b, not\ c \quad \mathbf{T}b \quad \mathbf{F}c}{\mathbf{T}\{b, not\ c\}}$$

Backward False Body (BFB)

Prerequisites A body is *false*, and all its literals except for one are *true*.

Consequence The residual body literal is *false*.

Tableau Rule BFB

$$\frac{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

Examples

$$\frac{\mathbf{F}\{b, \mathit{not}\ c\} \quad \mathbf{T}b}{\mathbf{T}c} \qquad \frac{\mathbf{F}\{b, \mathit{not}\ c\} \quad \mathbf{F}c}{\mathbf{F}b}$$

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

$$\frac{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

Examples

$$\frac{\mathbf{F}\{b, not\ c\} \quad \mathbf{T}b}{\mathbf{T}c} \qquad \frac{\mathbf{F}\{b, not\ c\} \quad \mathbf{F}c}{\mathbf{F}b}$$

Forward False Body (FFB)

Prerequisites Some literal of a body is *false*.

Consequence The body is *false*.

Tableau Rule FFB

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}\, l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

Examples

$$\frac{a \leftarrow b, not\ c \quad \mathbf{F}\, b}{\mathbf{F}\{b, not\ c\}} \qquad \frac{a \leftarrow b, not\ c \quad \mathbf{T}\, c}{\mathbf{F}\{b, not\ c\}}$$

Forward False Body (FFB)

Prerequisites Some literal of a body is *false*.

Consequence The body is *false*.

Tableau Rule FFB

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}\, l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

Examples

$$\frac{a \leftarrow b, \mathit{not}\ c \quad \mathbf{F}\, b}{\mathbf{F}\{b, \mathit{not}\ c\}} \qquad \frac{a \leftarrow b, \mathit{not}\ c \quad \mathbf{T}\, c}{\mathbf{F}\{b, \mathit{not}\ c\}}$$

Backward True Body (BTB)

Prerequisites A body is *true*.

Consequence The body's literals are *true*.

Tableau Rule BTB

$$\frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

Examples

$$\frac{\mathbf{T}\{b, not\ c\}}{\mathbf{T}b} \qquad \frac{\mathbf{T}\{b, not\ c\}}{\mathbf{F}c}$$

Backward True Body (BTB)

Prerequisites A body is *true*.

Consequence The body's literals are *true*.

Tableau Rule BTB

$$\frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

Examples

$$\frac{\mathbf{T}\{b, not\ c\}}{\mathbf{T}b} \qquad \frac{\mathbf{T}\{b, not\ c\}}{\mathbf{F}c}$$

Reviewing tableau rules for bodies

Consider rule body $B = \{l_1, \ldots, l_n\}$.

- Rules FTB and BFB amount to implication:

$$l_1 \wedge \cdots \wedge l_n \rightarrow B$$

- Rules FFB and BTB amount to implication:

$$B \rightarrow l_1 \wedge \cdots \wedge l_n$$

☞ Together they yield:

$$B \equiv l_1 \wedge \cdots \wedge l_n$$

Reviewing tableau rules for bodies

Consider rule body $B = \{l_1, \ldots, l_n\}$.

- Rules FTB and BFB amount to implication:

$$l_1 \wedge \cdots \wedge l_n \rightarrow B$$

- Rules FFB and BTB amount to implication:

$$B \rightarrow l_1 \wedge \cdots \wedge l_n$$

☞ Together they yield:

$$B \equiv l_1 \wedge \cdots \wedge l_n$$

Forward True Atom (FTA)

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

Consequence The atom is *true*.

Tableau Rule FTA

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{T}\{l_1, \ldots, l_n\}}{\mathbf{T}p}$$

Examples

$$\frac{a \leftarrow b, not\ c \quad \mathbf{T}\{b, not\ c\}}{\mathbf{T}a} \qquad \frac{a \leftarrow d, not\ e \quad \mathbf{T}\{d, not\ e\}}{\mathbf{T}a}$$

Forward True Atom (FTA)

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

Consequence The atom is *true*.

Tableau Rule FTA

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{T}\{l_1, \ldots, l_n\}}{\mathbf{T}p}$$

Examples

$$\frac{a \leftarrow b, not\ c \quad \mathbf{T}\{b, not\ c\}}{\mathbf{T}a} \qquad \frac{a \leftarrow d, not\ e \quad \mathbf{T}\{d, not\ e\}}{\mathbf{T}a}$$

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

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{F}p}{\mathbf{F}\{l_1, \ldots, l_n\}}$$

Examples

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

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

Backward False Atom (BFA)

Prerequisites An atom is *false*.

Consequence The bodies of all rules with the atom as head are *false*.

Tableau Rule BFA

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{F}p}{\mathbf{F}\{l_1, \ldots, l_n\}}$$

Examples

$$\frac{a \leftarrow b, not\ c \quad \mathbf{F}a}{\mathbf{F}\{b, not\ c\}} \qquad \frac{a \leftarrow d, not\ e \quad \mathbf{F}a}{\mathbf{F}\{d, not\ e\}}$$

Forward False Atom (FFA)

Prerequisites For some atom, the bodies of all rules with the atom as head are *false*.

Consequence The atom is *false*.

Tableau Rule FFA

$$\frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \; (body(p) = \{B_1, \ldots, B_m\})$$

☞ For an atom p occurring in a logic program Π, we let $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$.

Example

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

Forward False Atom (FFA)

Prerequisites For some atom, the bodies of all rules with the atom as head are *false*.

Consequence The atom is *false*.

Tableau Rule FFA

$$\frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \; (body(p) = \{B_1, \ldots, B_m\})$$

☞ For an atom p occurring in a logic program Π, we let $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$.

Example

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

Backward True Atom (BTA)

Prerequisites An atom is *true*, and the bodies of all rules with the atom as head except for one are *false*.

Consequence The residual body is *true*.

Tableau Rule BTA

$$\frac{\begin{array}{c}\mathbf{T}p \\ \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m\end{array}}{\mathbf{T}B_i} \; (body(p) = \{B_1, \ldots, B_m\})$$

Examples

$$\frac{\begin{array}{c}\mathbf{T}a \\ \mathbf{F}\{b, not\ c\}\end{array}}{\mathbf{T}\{d, not\ e\}} \; (*) \qquad \frac{\begin{array}{c}\mathbf{T}a \\ \mathbf{F}\{d, not\ e\}\end{array}}{\mathbf{T}\{b, not\ c\}} \; (*)$$

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

Backward True Atom (BTA)

Prerequisites An atom is *true*, and the bodies of all rules with the atom as head except for one are *false*.

Consequence The residual body is *true*.

Tableau Rule BTA

$$\frac{\begin{array}{c}\mathbf{T}p \\ \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m\end{array}}{\mathbf{T}B_i} \; (body(p) = \{B_1, \ldots, B_m\})$$

Examples

$$\frac{\begin{array}{c}\mathbf{T}a \\ \mathbf{F}\{b, not\ c\}\end{array}}{\mathbf{T}\{d, not\ e\}} \; (*) \qquad \frac{\begin{array}{c}\mathbf{T}a \\ \mathbf{F}\{d, not\ e\}\end{array}}{\mathbf{T}\{b, not\ c\}} \; (*)$$

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

Reviewing tableau rules for atoms

Consider an atom p such that $body(p) = \{B_1, \ldots, B_m\}$.

- Rules FTA and BFA amount to implication:

$$B_1 \vee \cdots \vee B_m \rightarrow p$$

- Rules FFA and BTA amount to implication:

$$p \rightarrow B_1 \vee \cdots \vee B_m$$

☞ Together they yield:

$$p \equiv B_1 \vee \cdots \vee B_m$$

Reviewing tableau rules for atoms

Consider an atom p such that $body(p) = \{B_1, \ldots, B_m\}$.

- Rules FTA and BFA amount to implication:

$$B_1 \vee \cdots \vee B_m \rightarrow p$$

- Rules FFA and BTA amount to implication:

$$p \rightarrow B_1 \vee \cdots \vee B_m$$

☞ Together they yield:

$$p \equiv B_1 \vee \cdots \vee B_m$$

Relationship with Clark's completion

Let Π be a normal logic program.

The eight tableau rules introduced so far essentially provide:

- (straightforward) inferences from $\mathit{Comp}(\Pi)$ (cf. Page 399)
- inferences via **atleast** (cf. Page 455)

Given the same partial assignment (of atoms),

- any literal derived by **atleast** is also derived by tableau rules,
  while the converse does not hold in general.

Relationship with Clark's completion

Let Π be a normal logic program.

The eight tableau rules introduced so far essentially provide:

- (straightforward) inferences from $Comp(\Pi)$ (cf. Page 399)
- inferences via **atleast** (cf. Page 455)

Given the same partial assignment (of atoms),

- any literal derived by **atleast** is also derived by tableau rules, while the converse does not hold in general.

Relationship with Clark's completion

Let Π be a normal logic program.

The eight tableau rules introduced so far essentially provide:

- (straightforward) inferences from $Comp(\Pi)$ (cf. Page 399)
- inferences via **atleast** (cf. Page 455)

Given the same partial assignment (of atoms),

- any literal derived by **atleast** is also derived by tableau rules,
- while the converse does not hold in general.

Relationship with Clark's completion

Let Π be a normal logic program.

The eight tableau rules introduced so far essentially provide:

- (straightforward) inferences from $Comp(\Pi)$ (cf. Page 399)
- inferences via **atleast** (cf. Page 455)

Given the same partial assignment (of atoms),

- any literal derived by **atleast** is also derived by tableau rules,
- while the converse does not hold in general.

Preliminaries for unfounded sets

Let Π be a normal logic program.

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

$$GUS(\Pi') = atom(\Pi) \setminus Cn((\Pi')^{\emptyset})$$

- 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.

Preliminaries for unfounded sets

Let Π be a normal logic program.

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

$$GUS(\Pi') = atom(\Pi) \setminus Cn((\Pi')^{\emptyset})$$

- 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.

Well-Founded Negation (WFN)

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

Consequence The atom is *false*.

Tableau Rule WFN

$$\frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \; (p \in GUS(\{r \in \Pi \mid body(r) \notin \{B_1, \ldots, B_m\}\}))$$

Examples

$$\frac{a \leftarrow not\ b \qquad \mathbf{F}\{not\ b\}}{\mathbf{F}a}\ (*) \qquad\qquad \frac{a \leftarrow a \qquad a \leftarrow not\ b \qquad \mathbf{F}\{not\ b\}}{\mathbf{F}a}\ (*)$$

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

Well-Founded Negation (WFN)

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

Consequence The atom is *false*.

Tableau Rule WFN

$$\frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \; (p \in GUS(\{r \in \Pi \mid body(r) \notin \{B_1, \ldots, B_m\}\}))$$

Examples

$$\frac{\begin{array}{c} a \leftarrow not\ b \\ \mathbf{F}\{not\ b\} \end{array}}{\mathbf{F}a} \; (*) \qquad \frac{\begin{array}{c} a \leftarrow a \\ a \leftarrow not\ b \\ \mathbf{F}\{not\ b\} \end{array}}{\mathbf{F}a} \; (*)$$

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

Well-Founded Justification (WFJ)

Prerequisites A *true* atom is in the greatest unfounded set with respect to rules whose bodies are *false* if a particular body is made *false*.

Consequence The respective body is *true*.

Tableau Rule WFJ

$$\frac{\mathbf{T}p \qquad \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m}{\mathbf{T}B_i} \; (p \in GUS(\{r \in \Pi \mid body(r) \notin \{B_1, \ldots, B_m\}\}))$$

Examples

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

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

Well-Founded Justification (WFJ)

Prerequisites A *true* atom is in the greatest unfounded set with respect to rules whose bodies are *false* if a particular body is made *false*.

Consequence The respective body is *true*.

Tableau Rule WFJ

$$\frac{\begin{array}{c}\mathbf{T}p \\ \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m\end{array}}{\mathbf{T}B_i} \; (p \in GUS(\{r \in \Pi \mid body(r) \notin \{B_1, \ldots, B_m\}\}))$$

Examples

$$\frac{\begin{array}{c}a \leftarrow not\ b \\ \mathbf{T}a\end{array}}{\mathbf{T}\{not\ b\}}\ (*) \qquad \frac{\begin{array}{c}a \leftarrow a \\ a \leftarrow not\ b \\ \mathbf{T}a\end{array}}{\mathbf{T}\{not\ b\}}\ (*)$$

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

Reviewing well-founded tableau rules

Tableau rules WFN and WFJ ensure non-circular support for *true* atoms.
Note that

1. WFN subsumes falsifying atoms via FFA,
2. WFJ can be viewed as "backward propagation" for unfounded sets,
3. WFJ subsumes backward propagation of *true* atoms via BTA.

Reviewing well-founded tableau rules

Tableau rules WFN and WFJ ensure non-circular support for *true* atoms. Note that

1. WFN subsumes falsifying atoms via FFA,
2. WFJ can be viewed as "backward propagation" for unfounded sets,
3. WFJ subsumes backward propagation of *true* atoms via BTA.

Relationship with well-founded operator

Let Π be a normal logic program, $\langle T, F \rangle$ a partial interpretation, and $\Pi' = \{r \in \Pi \mid body^+(r) \cap F = \emptyset, body^-(r) \cap T = \emptyset\}$.
Then the following conditions are equivalent:

1. $p \in \mathbf{U}_\Pi\langle T, F \rangle$; (cf. Page 495)
2. $p \in \mathbf{atmost}(\langle T, F \rangle)$; (cf. Page 531)
3. $p \in GUS(\Pi')$.
   - ➥ Well-founded operator, **atmost**, and WFN coincide.

☞ In contrast to the former, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable.

Relationship with well-founded operator

Let Π be a normal logic program, $\langle T, F\rangle$ a partial interpretation, and $\Pi' = \{r \in \Pi \mid body^+(r) \cap F = \emptyset, body^-(r) \cap T = \emptyset\}$.
Then the following conditions are equivalent:

1. $p \in \mathbf{U}_\Pi\langle T, F\rangle$; (cf. Page 495)
2. $p \in \mathbf{atmost}(\langle T, F\rangle)$; (cf. Page 531)
3. $p \in GUS(\Pi')$.

➥ Well-founded operator, **atmost**, and WFN coincide.

☞ In contrast to the former, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable.

Relationship with well-founded operator

Let Π be a normal logic program, $\langle T, F\rangle$ a partial interpretation, and $\Pi' = \{r \in \Pi \mid body^+(r) \cap F = \emptyset, body^-(r) \cap T = \emptyset\}$.
Then the following conditions are equivalent:

1. $p \in \mathbf{U}_\Pi\langle T, F\rangle$; (cf. Page 495)
2. $p \in$ **atmost(**$\langle T, F\rangle$**)**; (cf. Page 531)
3. $p \in GUS(\Pi')$.
   - ➥ Well-founded operator, **atmost**, and WFN coincide.

☞ In contrast to the former, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable.

Forward Loop (FL)

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

Consequence The atoms in the loop are *false*.

Tableau Rule FL

$$\frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \; (p \in L, L \in \mathit{Loop}(\Pi), \mathit{EB}(L) = \{B_1, \ldots, B_m\})$$

Example

$$\frac{\begin{array}{c} a \leftarrow a \\ a \leftarrow \mathit{not}\ b \\ \mathbf{F}\{\mathit{not}\ b\} \end{array}}{\mathbf{F}a} \; (\mathit{EB}(\{a\}) = \{\{\mathit{not}\ b\}\})$$

Forward Loop (FL)

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

Consequence The atoms in the loop are *false*.

Tableau Rule FL

$$\frac{\mathbf{F}B_1, \ldots, \mathbf{F}B_m}{\mathbf{F}p} \; (p \in L, L \in \mathit{Loop}(\Pi), \mathit{EB}(L) = \{B_1, \ldots, B_m\})$$

Example

$$\frac{\begin{array}{c} a \leftarrow a \\ a \leftarrow \mathit{not}\ b \\ \mathbf{F}\{\mathit{not}\ b\} \end{array}}{\mathbf{F}a} \; (\mathit{EB}(\{a\}) = \{\{\mathit{not}\ b\}\})$$

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

$$\frac{\begin{array}{c}\mathbf{T}p \\ \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m\end{array}}{\mathbf{T}B_i} \; (p \in L, L \in \mathit{Loop}(\Pi), \mathit{EB}(L) = \{B_1, \ldots, B_m\})$$

Example

$$\frac{\begin{array}{c}a \leftarrow a \\ a \leftarrow \mathit{not}\ b \\ \mathbf{T}a\end{array}}{\mathbf{T}\{\mathit{not}\ b\}} \; (\mathit{EB}(\{a\}) = \{\{\mathit{not}\ b\}\})$$

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

$$\frac{\begin{array}{c}\mathbf{T}p \\ \mathbf{F}B_1, \ldots, \mathbf{F}B_{i-1}, \mathbf{F}B_{i+1}, \ldots, \mathbf{F}B_m\end{array}}{\mathbf{T}B_i} \; (p \in L, L \in \mathit{Loop}(\Pi), \mathit{EB}(L) = \{B_1, \ldots, B_m\})$$

Example

$$\frac{\begin{array}{c}a \leftarrow a \\ a \leftarrow \mathit{not}\ b \\ \mathbf{T}a\end{array}}{\mathbf{T}\{\mathit{not}\ b\}} \; (\mathit{EB}(\{a\}) = \{\{\mathit{not}\ b\}\})$$

Reviewing tableau rules for loops

Tableau rules FL and BL ensure non-circular support for *true* atoms.
For a loop L such that $EB(L) = \{B_1, \ldots, B_m\}$,
they amount to implication:

$$\bigvee_{p \in L} p \rightarrow B_1 \vee \cdots \vee B_m$$

Comparison to well-founded tableau rules yields:

- FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
- BL cannot simulate inferences via WFJ.

Reviewing tableau rules for loops

Tableau rules FL and BL ensure non-circular support for *true* atoms.
For a loop L such that $EB(L) = \{B_1, \ldots, B_m\}$,
they amount to implication:

$$\bigvee_{p \in L} p \rightarrow B_1 \vee \cdots \vee B_m$$

Comparison to well-founded tableau rules yields:

- FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
- BL cannot simulate inferences via WFJ.

Relationship with loop formulas

Tableau rules FL and BL essentially provide:

- (straightforward) inferences from loop formulas (cf. Page 551)
  - ☞ But impractical to precompute exponentially many loop formulas !
- an application of the Lin-Zhao Theorem (cf. Page 555)

In practice, ASP-solvers such as `smodels`:

- exploit strongly connected components of positive atom dependency graphs
  - ➥ Can be viewed as an interpolation of FL.
- do not directly implement BL (and neither WFJ)
  - ➥ Probably difficult to do efficiently.
- could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
  - ➥ What about the computational cost?

Relationship with loop formulas

Tableau rules FL and BL essentially provide:

- (straightforward) inferences from loop formulas (cf. Page 551)
  - ☞ But impractical to precompute exponentially many loop formulas !
- an application of the Lin-Zhao Theorem (cf. Page 555)

In practice, ASP-solvers such as `smodels`:

- exploit strongly connected components of positive atom dependency graphs
  - ➥ Can be viewed as an interpolation of FL.
- do not directly implement BL (and neither WFJ)
  - ➥ Probably difficult to do efficiently.
- could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
  - ➥ What about the computational cost?

Relationship with loop formulas

Tableau rules FL and BL essentially provide:

- (straightforward) inferences from loop formulas (cf. Page 551)
  - ☞ But impractical to precompute exponentially many loop formulas !
- an application of the Lin-Zhao Theorem (cf. Page 555)

In practice, ASP-solvers such as `smodels`:

- exploit strongly connected components of positive atom dependency graphs
  - ➥ Can be viewed as an interpolation of FL.
- do not directly implement BL (and neither WFJ)
  - ➥ Probably difficult to do efficiently.
- could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
  - ➥ What about the computational cost?

Relationship with loop formulas

Tableau rules FL and BL essentially provide:

- (straightforward) inferences from loop formulas (cf. Page 551)
  - ☞ But impractical to precompute exponentially many loop formulas !
- an application of the Lin-Zhao Theorem (cf. Page 555)

In practice, ASP-solvers such as `smodels`:

- exploit strongly connected components of positive atom dependency graphs
  - ➥ Can be viewed as an interpolation of FL.
- do not directly implement BL (and neither WFJ)
  - ➥ Probably difficult to do efficiently.
- could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
  - ➥ What about the computational cost?

Relationship with loop formulas

Tableau rules FL and BL essentially provide:

- (straightforward) inferences from loop formulas (cf. Page 551)
  - ☞ But impractical to precompute exponentially many loop formulas !
- an application of the Lin-Zhao Theorem (cf. Page 555)

In practice, ASP-solvers such as `smodels`:

- exploit strongly connected components of positive atom dependency graphs
  - ➥ Can be viewed as an interpolation of FL.
- do not directly implement BL (and neither WFJ)
  - ➥ Probably difficult to do efficiently.
- could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
  - ➥ What about the computational cost?

Relationship with loop formulas

Tableau rules FL and BL essentially provide:

- (straightforward) inferences from loop formulas (cf. Page 551)
  - ☞ But impractical to precompute exponentially many loop formulas !
- an application of the Lin-Zhao Theorem (cf. Page 555)

In practice, ASP-solvers such as `smodels`:

- exploit strongly connected components of positive atom dependency graphs
  - ➥ Can be viewed as an interpolation of FL.
- do not directly implement BL (and neither WFJ)
  - ➥ Probably difficult to do efficiently.
- could simulate BL via FL/WFN by means of failed-literal detection (lookahead)
  - ➥ What about the computational cost?

Case analysis by *Cut*

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

☞ In general, closing a branch leads to a partial assignment.

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

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

$$\frac{}{\mathbf{T}v \mid \mathbf{F}v} \ (v \in \mathcal{C})$$

Examples $Cut[\mathcal{C}]$

$$\frac{\begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}}{\mathbf{T}a \mid \mathbf{F}a} \ (\mathcal{C} = atom(\Pi)) \qquad \frac{\begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}}{\mathbf{T}\{not\ b\} \mid \mathbf{F}\{not\ b\}} \ (\mathcal{C} = body(\Pi))$$

Case analysis by *Cut*

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

☞ In general, closing a branch leads to a partial assignment.

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

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

$$\frac{}{\mathbf{T}v \mid \mathbf{F}v} \ (v \in \mathcal{C})$$

Examples $Cut[\mathcal{C}]$

$$\frac{\begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}}{\mathbf{T}a \mid \mathbf{F}a} \ (\mathcal{C} = atom(\Pi)) \qquad \frac{\begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}}{\mathbf{T}\{not\ b\} \mid \mathbf{F}\{not\ b\}} \ (\mathcal{C} = body(\Pi))$$

Case analysis by *Cut*

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

☞ In general, closing a branch leads to a partial assignment.

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

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

$$\frac{}{\mathbf{T}v \mid \mathbf{F}v} \ (v \in \mathcal{C})$$

Examples $Cut[\mathcal{C}]$

$$\frac{\begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}}{\mathbf{T}a \mid \mathbf{F}a} \ (\mathcal{C} = atom(\Pi)) \qquad \frac{\begin{array}{c} a \leftarrow not\ b \\ b \leftarrow not\ a \end{array}}{\mathbf{T}\{not\ b\} \mid \mathbf{F}\{not\ b\}} \ (\mathcal{C} = body(\Pi))$$

Well-known tableau calculi

Fitting's operator $\mathbf{\Phi}$ applies forward propagation without sophisticated unfounded set checks. We have:

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

Well-founded operator $\mathbf{\Omega}$ replaces negation of single atoms with negation of unfounded sets. We have:

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

"Local" propagation via a program's completion can be determined by elementary inferences on atoms and rule bodies. We have:

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

Well-known tableau calculi

Fitting's operator $\mathbf{\Phi}$ applies forward propagation without sophisticated unfounded set checks. We have:

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

Well-founded operator $\mathbf{\Omega}$ replaces negation of single atoms with negation of unfounded sets. We have:

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

"Local" propagation via a program's completion can be determined by elementary inferences on atoms and rule bodies. We have:

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

Well-known tableau calculi

Fitting's operator $\mathbf{\Phi}$ applies forward propagation without sophisticated unfounded set checks. We have:

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

Well-founded operator $\mathbf{\Omega}$ replaces negation of single atoms with negation of unfounded sets. We have:

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

"Local" propagation via a program's completion can be determined by elementary inferences on atoms and rule bodies. We have:

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

Tableau calculi characterizing ASP-solvers

ASP-solvers combine propagation with case analysis.
We obtain the following tableau calculi characterizing [3, 57, 42, 73, 50, 47, 1]:

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

- SAT-based ASP-solvers, `assat` and `cmodels`, incrementally add loop formulas to a program's completion.
- Genuine ASP-solvers, `smodels`, `dlv`, `noMoRe`, and `nomore++`, essentially differ only in their *Cut* rules.

Tableau calculi characterizing ASP-solvers

ASP-solvers combine propagation with case analysis.
We obtain the following tableau calculi characterizing [3, 57, 42, 73, 50, 47, 1]:

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

- SAT-based ASP-solvers, `assat` and `cmodels`, incrementally add loop formulas to a program's completion.
- Genuine ASP-solvers, `smodels`, `dlv`, `noMoRe`, and `nomore++`, essentially differ only in their *Cut* rules.

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 larger than minimal refutations of $\mathcal{T}'$.
The size of tableaux is simply the number of their entries.

☞ We do not need to know the precise number of entries:
Counting required *Cut* applications is sufficient !

Proof complexity

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 larger than minimal refutations of $\mathcal{T}'$.
The size of tableaux is simply the number of their entries.

☞ We do not need to know the precise number of entries:
Counting required *Cut* applications is sufficient !

Proof complexity

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 larger than minimal refutations of $\mathcal{T}'$.
The size of tableaux is simply the number of their entries.

☞ We do not need to know the precise number of entries:
Counting required *Cut* applications is sufficient !

Proof complexity

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 larger than minimal refutations of $\mathcal{T}'$.
The size of tableaux is simply the number of their entries.

☞ We do not need to know the precise number of entries:
Counting required *Cut* applications is sufficient !

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

Recall that $\mathcal{T}_{smodels}$ restricts Cut to $atom(\Pi)$ and $\mathcal{T}_{noMoRe}$ to $body(\Pi)$.
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:

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

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

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

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

Recall that $\mathcal{T}_{smodels}$ restricts Cut to $atom(\Pi)$ and $\mathcal{T}_{noMoRe}$ to $body(\Pi)$.
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:

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

In minimal refutations for $\Pi_a^n \cup \Pi_c^n$, the number of applications of $Cut[body(\Pi_a^n \cup \Pi_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(\Pi_a^n \cup \Pi_c^n)]$.
Vice versa, minimal refutations for $\Pi_b^n \cup \Pi_c^n$ require linearly many applications of $Cut[atom(\Pi_b^n \cup \Pi_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(\Pi_b^n \cup \Pi_c^n)]$ with $\mathcal{T}_{noMoRe}$.

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

Recall that $\mathcal{T}_{smodels}$ restricts Cut to $atom(\Pi)$ and $\mathcal{T}_{noMoRe}$ to $body(\Pi)$.
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:

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

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

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

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

Recall that $\mathcal{T}_{smodels}$ restricts Cut to $atom(\Pi)$ and $\mathcal{T}_{noMoRe}$ to $body(\Pi)$.
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:

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

In minimal refutations for $\Pi_a^n \cup \Pi_c^n$, the number of applications of $Cut[body(\Pi_a^n \cup \Pi_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(\Pi_a^n \cup \Pi_c^n)]$.
Vice versa, minimal refutations for $\Pi_b^n \cup \Pi_c^n$ require linearly many applications of $Cut[atom(\Pi_b^n \cup \Pi_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(\Pi_b^n \cup \Pi_c^n)]$ with $\mathcal{T}_{noMoRe}$.

Relative efficiency

As witnessed by $\{\Pi_a^n \cup \Pi_c^n\}$ and $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another.
Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa).

It follows that

- both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
- $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$.

➥ The proof system obtained with $Cut[atom(\Pi) \cup body(\Pi)]$ is exponentially stronger than the ones with either $Cut[atom(\Pi)]$ or $Cut[body(\Pi)]$!

☞ Case analyses (at least) on atoms and bodies are mandatory in powerful ASP-solvers.

Relative efficiency

As witnessed by $\{\Pi_a^n \cup \Pi_c^n\}$ and $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another.
Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa).

It follows that

- both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
- $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$.

➥ The proof system obtained with $Cut[atom(\Pi) \cup body(\Pi)]$ is exponentially stronger than the ones with either $Cut[atom(\Pi)]$ or $Cut[body(\Pi)]$!

☞ Case analyses (at least) on atoms and bodies are mandatory in powerful ASP-solvers.

Relative efficiency

As witnessed by $\{\Pi_a^n \cup \Pi_c^n\}$ and $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another.
Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa).

It follows that

- both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
- $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$.

➥ The proof system obtained with $Cut[atom(\Pi) \cup body(\Pi)]$ is exponentially stronger than the ones with either $Cut[atom(\Pi)]$ or $Cut[body(\Pi)]$!

☞ Case analyses (at least) on atoms and bodies are mandatory in powerful ASP-solvers.

Relative efficiency

As witnessed by $\{\Pi_a^n \cup \Pi_c^n\}$ and $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another.
Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa).

It follows that

- both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
- $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$.

➥ The proof system obtained with $Cut[atom(\Pi) \cup body(\Pi)]$ is exponentially stronger than the ones with either $Cut[atom(\Pi)]$ or $Cut[body(\Pi)]$!

☞ Case analyses (at least) on atoms and bodies are mandatory in powerful ASP-solvers.

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

(r_1) $a \leftarrow not\ b$
(r_2) $b \leftarrow d, not\ a$
(r_3) $c \leftarrow b, d$
(r_4) $c \leftarrow g$
(r_5) $d \leftarrow c$
(r_6) $d \leftarrow g$
(r_7) $e \leftarrow f, not\ c$
(r_8) $f \leftarrow not\ g$
(r_9) $g \leftarrow not\ a, not\ f$

(1)	$\mathbf{T}a$	$[Cut]$
(2)	$\mathbf{T}\{not\ b\}$	$[BTA: r_1, 1]$
(3)	$\mathbf{F}b$	$[BTB: 2]$
(4)	$\mathbf{F}\{d, not\ a\}$	$[BFA: r_2, 3]$
(5)	$\mathbf{F}\{not\ a, not\ f\}$	$[FFB: r_9, 1]$
(6)	$\mathbf{F}g$	$[FFA: r_9, 5]$
(7)	$\mathbf{T}\{not\ g\}$	$[FTB: r_8, 6]$
(8)	$\mathbf{T}f$	$[FTA: r_8, 7]$
(9)	$\mathbf{F}\{b, d\}$	$[FFB: r_3, 3]$
(10)	$\mathbf{F}\{g\}$	$[FFB: r_4, r_6, 6]$
(11)	$\mathbf{F}c$	$[FFA: r_3, r_4, 9, 10]$
(12)	$\mathbf{F}\{c\}$	$[FFB: r_5, 11]$
(13)	$\mathbf{F}d$	$[FFA: r_5, r_6, 10, 12]$
(14)	$\mathbf{T}\{f, not\ c\}$	$[FTB: r_7, 8, 11]$
(15)	$\mathbf{T}e$	$[FTA: r_7, 14]$

(16)	$\mathbf{F}a$	$[Cut]$
(17)	$\mathbf{F}\{not\ b\}$	$[BFA: r_1, 16]$
(18)	$\mathbf{T}b$	$[BFB: 17]$
(19)	$\mathbf{T}\{d, not\ a\}$	$[BTA: r_2, 18]$
(20)	$\mathbf{T}d$	$[BTB: 19]$
(21)	$\mathbf{T}\{b, d\}$	$[FTB: r_3, 18, 20]$
(22)	$\mathbf{T}c$	$[FTA: r_3, 21]$
(23)	$\mathbf{F}\{f, not\ c\}$	$[FFB: r_7, 22]$
(24)	$\mathbf{F}e$	$[FFA: r_7, 23]$
(25)	$\mathbf{T}\{c\}$	$[FTB: r_5, 22]$

(26)	$\mathbf{T}f$	$[Cut]$
(27)	$\mathbf{F}\{not\ a, not\ f\}$	$[FFB: r_9, 26]$
(28)	$\mathbf{F}c$	$[WFN: 27]$

(29)	$\mathbf{F}f$	$[Cut]$
(30)	$\mathbf{T}\{not\ a, not\ f\}$	$[FTB: r_9,$
(31)	$\mathbf{T}g$	$[FTA: r_9,$
(32)	$\mathbf{T}\{g\}$	$[FTB: r_4,$
(33)	$\mathbf{F}\{not\ g\}$	$[FFB: r_8,$

$\mathcal{T}_{noMoRe}$: Example tableau

(r_1) $a \leftarrow not\ b$
(r_2) $b \leftarrow d, not\ a$
(r_3) $c \leftarrow b, d$
(r_4) $c \leftarrow g$
(r_5) $d \leftarrow c$
(r_6) $d \leftarrow g$
(r_7) $e \leftarrow f, not\ c$
(r_8) $f \leftarrow not\ g$
(r_9) $g \leftarrow not\ a, not\ f$

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

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

(r_1) $a \leftarrow not\ b$
(r_2) $b \leftarrow d, not\ a$
(r_3) $c \leftarrow b, d$
(r_4) $c \leftarrow g$
(r_5) $d \leftarrow c$
(r_6) $d \leftarrow g$
(r_7) $e \leftarrow f, not\ c$
(r_8) $f \leftarrow not\ g$
(r_9) $g \leftarrow not\ a, not\ f$

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

Conflict-Driven Answer Set Solving
Overview

53 Motivation

54 Boolean Constraints

55 Nogoods from Logic Programs
- Nogoods from Clark's Completion
- Nogoods from Loop Formulas

56 Conflict-Driven Nogood Learning
- CDNL-ASP Algorithm
- Nogood Propagation
- Conflict Analysis

57 Implementation via clasp

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

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

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

Assignments

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

$$(\sigma_1, \ldots, \sigma_n)$$

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, $\overline{\sigma}$, 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.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals. Given this, we access *true* and *false* propositions in A via

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

Assignments

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

  $$(\sigma_1, \ldots, \sigma_n)$$

  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, $\overline{\sigma}$, 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.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals. Given this, we access *true* and *false* propositions in A via

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

Assignments

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

  $$(\sigma_1, \ldots, \sigma_n)$$

  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, $\overline{\sigma}$, 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.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals. Given this, we access *true* and *false* propositions in A via

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

Assignments

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

  $$(\sigma_1, \ldots, \sigma_n)$$

  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, $\overline{\sigma}$, 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.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals. Given this, we access *true* and *false* propositions in A via

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

Assignments

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

  $$(\sigma_1, \ldots, \sigma_n)$$

  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, $\overline{\sigma}$, 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.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals. Given this, we access *true* and *false* propositions in A via

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

Assignments

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

  $$(\sigma_1, \ldots, \sigma_n)$$

  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, $\overline{\sigma}$, 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.
- Given $A = (\sigma_1, \ldots, \sigma_{k-1}, \sigma_k, \ldots, \sigma_n)$, we let $A[\sigma_k] = (\sigma_1, \ldots, \sigma_{k-1})$.
- We sometimes identify an assignment with the set of its literals. Given this, we access *true* and *false* propositions in A via

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

Nogoods, Solutions, and Unit Propagation

- A nogood is a set $\{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_n$.
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ 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 $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$.
- 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.

Nogoods, Solutions, and Unit Propagation

- A nogood is a set $\{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_n$.
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ 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 $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$.
- 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.

Nogoods, Solutions, and Unit Propagation

- A nogood is a set $\{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_n$.
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ 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 $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$.
- 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.

Nogoods, Solutions, and Unit Propagation

- A nogood is a set $\{\sigma_1, \ldots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment containing $\sigma_1, \ldots, \sigma_n$.
- An assignment A such that $A^{\mathbf{T}} \cup A^{\mathbf{F}} = dom(A)$ 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 $\overline{\sigma}$ is unit-resulting for δ wrt A, if
  1. $\delta \setminus A = \{\sigma\}$ and
  2. $\overline{\sigma} \notin A$.
- 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.

Nogoods from logic programs

via Clark's completion

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

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

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

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

Nogoods from logic programs (ctd)

via Clark's completion

Let $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$ be a body.
The equivalence

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

can be decomposed into two implications.

1. We get

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

which is equivalent to the conjunction of

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

This set of clauses expresses the following set of nogoods:

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

Nogoods from logic programs (ctd)

via Clark's completion

Let $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$ be a body.
The equivalence

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

can be decomposed into two implications.

1. We get

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

which is equivalent to the conjunction of

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

This set of clauses expresses the following set of nogoods:

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

Nogoods from logic programs (ctd)

via Clark's completion

Let $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$ be a body.
The equivalence

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

can be decomposed into two implications.

2. The converse of the previous implication, viz.

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

gives rise to the nogood

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

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

Nogoods from logic programs (ctd)

via Clark's completion

Let $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$ be a body.
The equivalence

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

can be decomposed into two implications.

2. The converse of the previous implication, viz.

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

gives rise to the nogood

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

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

Nogoods from logic programs (ctd)

via Clark's 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 $p \in \mathit{atom}(\Pi)$ along with its bodies $\mathit{body}(p) = \{\beta_1, \ldots, \beta_k\}$ the nogoods

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

Nogoods from logic programs

atom-oriented nogoods

For an atom p where $body(p) = \{\beta_1, \ldots, \beta_k\}$, recall that

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

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

$$\begin{array}{|lcl|} \hline x & \leftarrow & y \\ x & \leftarrow & not\ z \\ \hline \end{array} \qquad \begin{aligned} \delta(x) &= \{\mathbf{T}x, \mathbf{F}\{y\}, \mathbf{F}\{not\ z\}\} \\ \Delta(x) &= \{\{\mathbf{F}x, \mathbf{T}\{y\}\}, \{\mathbf{F}x, \mathbf{T}\{not\ z\}\}\} \end{aligned}$$

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 $(\mathbf{T}x, \mathbf{F}\{y\})$.

Nogoods from logic programs

atom-oriented nogoods

For an atom p where $body(p) = \{\beta_1, \ldots, \beta_k\}$, recall that

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

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

x	$\leftarrow$	y
x	$\leftarrow$	$not\ z$

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

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 $(\mathbf{T}x, \mathbf{F}\{y\})$.

Nogoods from logic programs

atom-oriented nogoods

For an atom p where $body(p) = \{\beta_1, \ldots, \beta_k\}$, recall that

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

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

$$\boxed{\begin{array}{lcl} x & \leftarrow & y \\ x & \leftarrow & not\ z \end{array}}$$

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

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 $(\mathbf{T}x, \mathbf{F}\{y\})$.

Nogoods from logic programs

atom-oriented nogoods

For an atom p where $body(p) = \{\beta_1, \ldots, \beta_k\}$, recall that

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

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

x	$\leftarrow$	y
x	$\leftarrow$	$not\ z$

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

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 $(\mathbf{T}x, \mathbf{F}\{y\})$.

Nogoods from logic programs

atom-oriented nogoods

For an atom p where $body(p) = \{\beta_1, \ldots, \beta_k\}$, recall that

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

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

x	$\leftarrow$	y
x	$\leftarrow$	$not\ z$

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

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 $(\mathbf{T}x, \mathbf{F}\{y\})$.

Nogoods from logic programs

body-oriented nogoods

For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, recall that

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

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

$$\begin{array}{|c|}\hline \ldots \leftarrow x, not\ y \\ \vdots \\ \ldots \leftarrow x, not\ y \\ \hline\end{array}$$

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

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 $(\mathbf{F}\{x, not\ y\}, \mathbf{T}x)$.

Nogoods from logic programs

body-oriented nogoods

For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, recall that

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

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

$$\begin{array}{|c|} \hline \ldots \leftarrow x, not\ y \\ \vdots \\ \ldots \leftarrow x, not\ y \\ \hline \end{array}$$

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

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 $(\mathbf{F}\{x, not\ y\}, \mathbf{T}x)$.

Nogoods from logic programs

body-oriented nogoods

For a body $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$, recall that

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

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

$$\begin{array}{|l|} \hline \ldots \leftarrow x, \mathit{not}\ y \\ \vdots \\ \ldots \leftarrow x, \mathit{not}\ y \\ \hline \end{array}$$

$$\begin{aligned} \delta(\{x, \mathit{not}\ y\}) &= \{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\} \\ \Delta(\{x, \mathit{not}\ y\}) &= \{\{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{T} \end{aligned}$$

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

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

Nogoods from logic programs

body-oriented nogoods

For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, recall that

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

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

$$\begin{array}{|c|}\hline \ldots \leftarrow x, not\ y \\ \vdots \\ \ldots \leftarrow x, not\ y \\ \hline \end{array}$$

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

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 $(\mathbf{F}\{x, not\ y\}, \mathbf{T}x)$.

Nogoods from logic programs

body-oriented nogoods

For a body $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$, recall that

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

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

$$\boxed{\begin{array}{c} \ldots \leftarrow x, \mathit{not}\ y \\ \vdots \\ \ldots \leftarrow x, \mathit{not}\ y \end{array}}$$

$$\begin{aligned} \delta(\{x, \mathit{not}\ y\}) &= \{\mathbf{F}\{x, \mathit{not}\ y\}, \mathbf{T}x, \mathbf{F}y\} \\ \Delta(\{x, \mathit{not}\ y\}) &= \{\{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{F}x\}, \{\mathbf{T}\{x, \mathit{not}\ y\}, \mathbf{T} \end{aligned}$$

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

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

Characterization of answer sets

for tight logic programs

Let Π be a logic program and

$$\begin{aligned}\Delta_\Pi &= \{\delta(p) \mid p \in \mathit{atom}(\Pi)\} \cup \{\delta \in \Delta(p) \mid p \in \mathit{atom}(\Pi)\} \\ &\cup \{\delta(\beta) \mid \beta \in \mathit{body}(\Pi)\} \cup \{\delta \in \Delta(\beta) \mid \beta \in \mathit{body}(\Pi)\}\ .\end{aligned}$$

Theorem

Let Π be a tight logic program. Then,
$X \subseteq \mathit{atom}(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap \mathit{atom}(\Pi)$ for a (unique) solution A for Δ_Π.

☞ The set Δ_Π of nogoods captures inferences from (program Π and) Clark's completion.

Characterization of answer sets

for tight logic programs

Let Π be a logic program and

$$\begin{aligned}\Delta_\Pi &= \{\delta(p) \mid p \in \mathit{atom}(\Pi)\} \cup \{\delta \in \Delta(p) \mid p \in \mathit{atom}(\Pi)\} \\ &\cup \{\delta(\beta) \mid \beta \in \mathit{body}(\Pi)\} \cup \{\delta \in \Delta(\beta) \mid \beta \in \mathit{body}(\Pi)\}\ .\end{aligned}$$

Theorem

Let Π be a tight logic program. Then,
$X \subseteq \mathit{atom}(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap \mathit{atom}(\Pi)$ for a (unique) solution A for Δ_Π.

☞ The set Δ_Π of nogoods captures inferences from (program Π and) Clark's completion.

Characterization of answer sets

for tight logic programs

Let Π be a logic program and

$$\begin{aligned}\Delta_\Pi &= \{\delta(p) \mid p \in \mathit{atom}(\Pi)\} \cup \{\delta \in \Delta(p) \mid p \in \mathit{atom}(\Pi)\} \\ &\cup \{\delta(\beta) \mid \beta \in \mathit{body}(\Pi)\} \cup \{\delta \in \Delta(\beta) \mid \beta \in \mathit{body}(\Pi)\}\ .\end{aligned}$$

Theorem

Let Π be a tight logic program. Then,
$X \subseteq \mathit{atom}(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap \mathit{atom}(\Pi)$ for a (unique) solution A for Δ_Π.

- ☞ The set Δ_Π of nogoods captures inferences from (program Π and) Clark's completion.

Atom-oriented nogoods and tableau rules

- Tableau rules FTA, BFA, FFA, and BTA are atom-oriented.
- For an atom p such that $body(p) = \{\beta_1, \ldots, \beta_k\}$, consider the equivalence: $p \leftrightarrow p_{\beta_1} \vee \cdots \vee p_{\beta_k}$
- Inferences from nogoods $\Delta(p) = \{\{\mathbf{F}p, \mathbf{T}\beta_1\}, \ldots, \{\mathbf{F}p, \mathbf{T}\beta_k\}\}$ correspond to those from tableau rules FTA and BFA:

$$\frac{p \leftarrow \beta \quad \mathbf{T}\beta}{\mathbf{T}p} \qquad \frac{p \leftarrow \beta \quad \mathbf{F}p}{\mathbf{F}\beta}$$

- Inferences from nogood $\delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$ correspond to those from tableau rules FFA and BTA:

$$\frac{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k}{\mathbf{F}p} \qquad \frac{\mathbf{T}p \quad \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_{i-1}, \mathbf{F}\beta_{i+1}, \ldots, \mathbf{F}\beta_k}{\mathbf{T}\beta_i}$$

Atom-oriented nogoods and tableau rules

- Tableau rules FTA, BFA, FFA, and BTA are atom-oriented.
- For an atom p such that $body(p) = \{\beta_1, \ldots, \beta_k\}$, consider the equivalence: $p \leftrightarrow p_{\beta_1} \vee \cdots \vee p_{\beta_k}$

- Inferences from nogoods $\Delta(p) = \{\{\mathbf{F}p, \mathbf{T}\beta_1\}, \ldots, \{\mathbf{F}p, \mathbf{T}\beta_k\}\}$ correspond to those from tableau rules FTA and BFA:

$$\frac{p \leftarrow \beta \quad \mathbf{T}\beta}{\mathbf{T}p} \qquad \frac{p \leftarrow \beta \quad \mathbf{F}p}{\mathbf{F}\beta}$$

- Inferences from nogood $\delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$ correspond to those from tableau rules FFA and BTA:

$$\frac{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k}{\mathbf{F}p} \qquad \frac{\mathbf{T}p \quad \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_{i-1}, \mathbf{F}\beta_{i+1}, \ldots, \mathbf{F}\beta_k}{\mathbf{T}\beta_i}$$

Atom-oriented nogoods and tableau rules

- Tableau rules FTA, BFA, FFA, and BTA are atom-oriented.
- For an atom p such that $body(p) = \{\beta_1, \ldots, \beta_k\}$, consider the equivalence: $p \leftrightarrow p_{\beta_1} \vee \cdots \vee p_{\beta_k}$

- Inferences from nogoods $\Delta(p) = \{\{\mathbf{F}p, \mathbf{T}\beta_1\}, \ldots, \{\mathbf{F}p, \mathbf{T}\beta_k\}\}$ correspond to those from tableau rules FTA and BFA:

$$\frac{p \leftarrow \beta \quad \mathbf{T}\beta}{\mathbf{T}p} \qquad \frac{p \leftarrow \beta \quad \mathbf{F}p}{\mathbf{F}\beta}$$

- Inferences from nogood $\delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$ correspond to those from tableau rules FFA and BTA:

$$\frac{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k}{\mathbf{F}p} \qquad \frac{\mathbf{T}p \quad \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_{i-1}, \mathbf{F}\beta_{i+1}, \ldots, \mathbf{F}\beta_k}{\mathbf{T}\beta_i}$$

Atom-oriented nogoods and tableau rules

- Tableau rules FTA, BFA, FFA, and BTA are atom-oriented.
- For an atom p such that $body(p) = \{\beta_1, \ldots, \beta_k\}$, consider the equivalence: $p \leftrightarrow p_{\beta_1} \vee \cdots \vee p_{\beta_k}$

- Inferences from nogoods $\Delta(p) = \{\{\mathbf{F}p, \mathbf{T}\beta_1\}, \ldots, \{\mathbf{F}p, \mathbf{T}\beta_k\}\}$ correspond to those from tableau rules FTA and BFA:

$$\frac{p \leftarrow \beta \quad \mathbf{T}\beta}{\mathbf{T}p} \qquad \frac{p \leftarrow \beta \quad \mathbf{F}p}{\mathbf{F}\beta}$$

- Inferences from nogood $\delta(p) = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$ correspond to those from tableau rules FFA and BTA:

$$\frac{\mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k}{\mathbf{F}p} \qquad \frac{\mathbf{T}p \quad \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_{i-1}, \mathbf{F}\beta_{i+1}, \ldots, \mathbf{F}\beta_k}{\mathbf{T}\beta_i}$$

Body-oriented nogoods and tableau rules

- Tableau rules FTB, BFB, FFB, and BTB are body-oriented.
- For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} = \{l_1, \ldots, l_n\}$, consider the equivalence: $p_\beta \leftrightarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$
- Inferences from nogood $\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$ correspond to those from tableau rules FTB and BFB:

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}} \qquad \frac{\mathbf{F}\{l_1, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

- Inferences from nogoods $\Delta(\beta) = \{\{\mathbf{T}\beta, \mathbf{F}p_1\}, \ldots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \ldots, \{\mathbf{T}\beta, \mathbf{T}p_n\}\}$ correspond to those from tableau rules FFB and BTB:

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}} \qquad \frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

Body-oriented nogoods and tableau rules

- Tableau rules FTB, BFB, FFB, and BTB are body-oriented.
- For a body $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\} = \{l_1, \ldots, l_n\}$, consider the equivalence: $p_\beta \leftrightarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$
- Inferences from nogood $\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$ correspond to those from tableau rules FTB and BFB:

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}} \qquad \frac{\mathbf{F}\{l_1, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

- Inferences from nogoods $\Delta(\beta) = \{\{\mathbf{T}\beta, \mathbf{F}p_1\}, \ldots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \ldots, \{\mathbf{T}\beta, \mathbf{T}p_n\}\}$ correspond to those from tableau rules FFB and BTB:

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}} \qquad \frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

Body-oriented nogoods and tableau rules

- Tableau rules FTB, BFB, FFB, and BTB are body-oriented.
- For a body $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} = \{l_1, \ldots, l_n\}$, consider the equivalence: $p_\beta \leftrightarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$
- Inferences from nogood $\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$ correspond to those from tableau rules FTB and BFB:

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}} \qquad \frac{\mathbf{F}\{l_1, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

- Inferences from nogoods $\Delta(\beta) = \{\{\mathbf{T}\beta, \mathbf{F}p_1\}, \ldots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \ldots, \{\mathbf{T}\beta, \mathbf{T}p_n\}\}$ correspond to those from tableau rules FFB and BTB:

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}} \qquad \frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

Body-oriented nogoods and tableau rules

- Tableau rules FTB, BFB, FFB, and BTB are body-oriented.
- For a body $\beta = \{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\} = \{l_1, \ldots, l_n\}$, consider the equivalence: $p_\beta \leftrightarrow p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$
- Inferences from nogood $\delta(\beta) = \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$ correspond to those from tableau rules FTB and BFB:

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_n}{\mathbf{T}\{l_1, \ldots, l_n\}} \qquad \frac{\mathbf{F}\{l_1, \ldots, l_n\} \quad \mathbf{t}l_1, \ldots, \mathbf{t}l_{i-1}, \mathbf{t}l_{i+1}, \ldots, \mathbf{t}l_n}{\mathbf{f}l_i}$$

- Inferences from nogoods $\Delta(\beta) = \{\{\mathbf{T}\beta, \mathbf{F}p_1\}, \ldots, \{\mathbf{T}\beta, \mathbf{F}p_m\}, \{\mathbf{T}\beta, \mathbf{T}p_{m+1}\}, \ldots, \{\mathbf{T}\beta, \mathbf{T}p_n\}\}$ correspond to those from tableau rules FFB and BTB:

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \mathbf{f}l_i}{\mathbf{F}\{l_1, \ldots, l_i, \ldots, l_n\}} \qquad \frac{\mathbf{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\mathbf{t}l_i}$$

Nogoods from logic programs

via loop formulas (cf. Page 551)

Let Π be a normal logic program and recall that:

- For $L \subseteq atom(\Pi)$, the external supports of L for Π are
$$ES_\Pi(L) = \{r \in \Pi \mid head(r) \in L, body^+(r) \cap L = \emptyset\}.$$
- The (disjunctive) loop formula of L for Π is
$$\begin{aligned} LF_\Pi(L) &= \left(\bigvee_{A \in L} A\right) \rightarrow \left(\bigvee_{r \in ES_\Pi(L)} Comp(body(r))\right) \\ &\equiv \left(\bigwedge_{r \in ES_\Pi(L)} \neg Comp(body(r))\right) \rightarrow \left(\bigwedge_{A \in L} \neg A\right). \end{aligned}$$
  - ☞ The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported.
- The external bodies of L for Π are
$$\begin{aligned} EB(L) &= \{body(r) \mid r \in \Pi, head(r) \in L, body^+(r) \cap L = \emptyset\} \\ &= \{body(r) \mid r \in ES_\Pi(L)\}. \end{aligned}$$

Nogoods from logic programs

loop nogoods

For a logic program Π and some $\emptyset \subset U \subseteq atom(\Pi)$, define the loop nogood of an atom $p \in U$ as

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

where $EB(U) = \{\beta_1, \ldots, \beta_k\}$.

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

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

☞ The set Λ_Π of loop nogoods denies cyclic support among *true* atoms.

Nogoods from logic programs

loop nogoods

For a logic program Π and some $\emptyset \subset U \subseteq atom(\Pi)$, define the loop nogood of an atom $p \in U$ as

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

where $EB(U) = \{\beta_1, \ldots, \beta_k\}$.

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

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

☞ The set Λ_Π of loop nogoods denies cyclic support among *true* atoms.

Nogoods from logic programs

loop nogoods

For a logic program Π and some $\emptyset \subset U \subseteq atom(\Pi)$, define the loop nogood of an atom $p \in U$ as

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

where $EB(U) = \{\beta_1, \ldots, \beta_k\}$.

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

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

☞ The set Λ_Π of loop nogoods denies cyclic support among *true* atoms.

Example

Consider

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

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

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

Similarly for v in $\{u, v\}$, we get:

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

Example

Consider

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

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

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

Similarly for v in $\{u, v\}$, we get:

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

Example

Consider

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

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

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

Similarly for v in $\{u, v\}$, we get:

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

Characterization of answer sets

For a logic program Π,
let Δ_Π and Λ_Π as defined on Page 711 and Page 723, respectively.

Theorem

Let Π be a logic program. Then,
$X \subseteq atom(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- Nogoods in Λ_Π augment Δ_Π with conditions checking for unfounded sets, in particular, those being loops. While $|\Delta_\Pi|$ is linear in the size of Π, Λ_Π may contain exponentially many (non-redundant) loop nogoods !

Characterization of answer sets

For a logic program Π,
let Δ_Π and Λ_Π as defined on Page 711 and Page 723, respectively.

Theorem

Let Π be a logic program. Then,
$X \subseteq atom(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- Nogoods in Λ_Π augment Δ_Π with conditions checking for unfounded sets, in particular, those being loops. While $|\Delta_\Pi|$ is linear in the size of Π, Λ_Π may contain exponentially many (non-redundant) loop nogoods !

Characterization of answer sets

For a logic program Π,
let Δ_Π and Λ_Π as defined on Page 711 and Page 723, respectively.

Theorem

Let Π be a logic program. Then,
$X \subseteq atom(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- Nogoods in Λ_Π augment Δ_Π with conditions checking for unfounded sets, in particular, those being loops.
- While $|\Delta_\Pi|$ is linear in the size of Π, Λ_Π may contain exponentially many (non-redundant) loop nogoods !

Characterization of answer sets

For a logic program Π,
let Δ_Π and Λ_Π as defined on Page 711 and Page 723, respectively.

Theorem

Let Π be a logic program. Then,
$X \subseteq atom(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- Nogoods in Λ_Π augment Δ_Π with conditions checking for unfounded sets, in particular, those being loops.
- While $|\Delta_\Pi|$ is linear in the size of Π, Λ_Π may contain exponentially many (non-redundant) loop nogoods !

Characterization of answer sets

For a logic program Π,
let Δ_Π and Λ_Π as defined on Page 711 and Page 723, respectively.

Theorem

Let Π be a logic program. Then,
$X \subseteq atom(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- Nogoods in Λ_Π augment Δ_Π with conditions checking for unfounded sets, in particular, those being loops.
- While $|\Delta_\Pi|$ is linear in the size of Π, Λ_Π may contain exponentially many (non-redundant) loop nogoods !

Characterization of answer sets

For a logic program Π,
let Δ_Π and Λ_Π as defined on Page 711 and Page 723, respectively.

Theorem

Let Π be a logic program. Then,
$X \subseteq atom(\Pi)$ is an answer set of Π iff
$X = A^{\mathsf{T}} \cap atom(\Pi)$ for a (unique) solution A for $\Delta_\Pi \cup \Lambda_\Pi$.

Some remarks

- Nogoods in Λ_Π augment Δ_Π with conditions checking for unfounded sets, in particular, those being loops.
- While $|\Delta_\Pi|$ is linear in the size of Π, Λ_Π may contain exponentially many (non-redundant) loop nogoods !

Conflict-driven search

Boolean constraint solving algorithms pioneered for SAT led to:

Traditional approach

- (Unit) propagation
- Exhaustive (chronological) backtracking
- ☞ DPLL [15, 14]

State of the art

- (Unit) propagation
- Conflict analysis (via resolution)
- Learning + Backjumping + Assertion
- ☞ CDCL [79, 61]

Idea

- ➥ Apply CDCL-style search in ASP solving !

Conflict-driven search

Boolean constraint solving algorithms pioneered for SAT led to:

Traditional approach

- (Unit) propagation
- Exhaustive (chronological) backtracking

☞ DPLL [15, 14]

State of the art

- (Unit) propagation
- Conflict analysis (via resolution)
- Learning + Backjumping + Assertion

☞ CDCL [79, 61]

Idea

➥ Apply CDCL-style search in ASP solving !

Conflict-driven search

Boolean constraint solving algorithms pioneered for SAT led to:

Traditional approach

- (Unit) propagation
- Exhaustive (chronological) backtracking

☞ DPLL [15, 14]

State of the art

- (Unit) propagation
- Conflict analysis (via resolution)
- Learning + Backjumping + Assertion

☞ CDCL [79, 61]

Idea

➥ Apply CDCL-style search in ASP solving !

Outline of CDNL-ASP algorithm

[30]

- Keep track of deterministic consequences by unit propagation on:
  - Clark's completion $[\Delta_\Pi]$
  - Loop nogoods, determined and recorded on demand $[\Lambda_\Pi]$
    - ☞ Dedicated unfounded set detection !
  - Dynamic nogoods, derived from conflicts and unfounded sets $[\nabla]$
- When a nogood in $\Delta_\Pi \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution until reaching the First Unique Implication Point (First-UIP) [62]
  - Learn the derived conflict nogood δ
  - Backjump to the earliest (heuristic) choice such that the complement of the First-UIP is unit-resulting for δ
  - Assert the complement of the First-UIP and proceed (by unit propagation)
- Terminate when either:
  - Finding an answer set (a solution for $\Delta_\Pi \cup \Lambda_\Pi$)
  - Deriving a conflict independently of (heuristic) choices

Outline of CDNL-ASP algorithm

[30]

- Keep track of deterministic consequences by unit propagation on:
  - Clark's completion $[\Delta_\Pi]$
  - Loop nogoods, determined and recorded on demand $[\Lambda_\Pi]$
    - ☞ Dedicated unfounded set detection !
  - Dynamic nogoods, derived from conflicts and unfounded sets $[\nabla]$
- When a nogood in $\Delta_\Pi \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution until reaching the First Unique Implication Point (First-UIP) [62]
  - Learn the derived conflict nogood δ
  - Backjump to the earliest (heuristic) choice such that the complement of the First-UIP is unit-resulting for δ
  - Assert the complement of the First-UIP and proceed (by unit propagation)
- Terminate when either:
  - Finding an answer set (a solution for $\Delta_\Pi \cup \Lambda_\Pi$)
  - Deriving a conflict independently of (heuristic) choices

Outline of CDNL-ASP algorithm

[30]

- Keep track of deterministic consequences by unit propagation on:
  - Clark's completion $[\Delta_\Pi]$
  - Loop nogoods, determined and recorded on demand $[\Lambda_\Pi]$
    - ☞ Dedicated unfounded set detection !
  - Dynamic nogoods, derived from conflicts and unfounded sets $[\nabla]$
- When a nogood in $\Delta_\Pi \cup \nabla$ becomes violated:
  - Analyze the conflict by resolution until reaching the First Unique Implication Point (First-UIP) [62]
  - Learn the derived conflict nogood δ
  - Backjump to the earliest (heuristic) choice such that the complement of the First-UIP is unit-resulting for δ
  - Assert the complement of the First-UIP and proceed (by unit propagation)
- Terminate when either:
  - Finding an answer set (a solution for $\Delta_\Pi \cup \Lambda_\Pi$)
  - Deriving a conflict independently of (heuristic) choices

Algorithm 1: CDNL-ASP

Input : A logic program Π.
Output : An answer set of Π or "no answer set".

```
1  A ← ∅                                          // assignment over atom(Π) ∪ body(Π)
2  ∇ ← ∅                                          // set of (dynamic) nogoods
3  dl ← 0                                         // decision level
4  loop
5      (A, ∇) ← NogoodPropagation(Π, ∇, A)
6      if ε ⊆ A for some ε ∈ Δ_Π ∪ ∇ then
7          if dl = 0 then return no answer set
8          (δ, k) ← ConflictAnalysis(ε, Π, ∇, A)
9          ∇ ← ∇ ∪ {δ}                            // learning
10         A ← (A \ {σ ∈ A | k < dl(σ)})          // backjumping
11         dl ← k
12     else if A^T ∪ A^F = atom(Π) ∪ body(Π) then
13         return A^T ∩ atom(Π)                   // answer set
14     else
15         σ_d ← Select(Π, ∇, A)                  // heuristic choice of σ_d ∉ A
16         dl ← dl + 1
17         A ← A ∘ (σ_d)                          // dl(σ_d) = dl
```

Observations

- Decision level dl, initially set to 0, is used to count the number of heuristically chosen literals in assignment A.
- For a heuristically chosen literal $\sigma_d = \mathbf{T}p$ or $\sigma_d = \mathbf{F}p$, respectively, we require $p \in (atom(\Pi) \cup body(\Pi)) \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$.
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of σ, viz. the value dl had when σ was assigned.
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_\Pi \cup \nabla$.
- A conflict at decision level 0 (where A contains no heuristically chosen literals) indicates non-existence of answer sets.
- A nogood δ derived by conflict analysis is asserting, that is, some literal is unit-resulting for δ at a decision level $k < dl$.
  - ➥ After learning δ and backjumping to decision level k, at least one literal is newly derivable by unit propagation.
  - ☞ No explicit flipping of heuristically chosen literals !

Observations

- Decision level dl, initially set to 0, is used to count the number of heuristically chosen literals in assignment A.
- For a heuristically chosen literal $\sigma_d = \mathbf{T}p$ or $\sigma_d = \mathbf{F}p$, respectively, we require $p \in (atom(\Pi) \cup body(\Pi)) \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$.
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of σ, viz. the value dl had when σ was assigned.
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_\Pi \cup \nabla$.
- A conflict at decision level 0 (where A contains no heuristically chosen literals) indicates non-existence of answer sets.
- A nogood δ derived by conflict analysis is asserting, that is, some literal is unit-resulting for δ at a decision level $k < dl$.
  - ➥ After learning δ and backjumping to decision level k, at least one literal is newly derivable by unit propagation.
  - ☞ No explicit flipping of heuristically chosen literals !

Observations

- Decision level dl, initially set to 0, is used to count the number of heuristically chosen literals in assignment A.
- For a heuristically chosen literal $\sigma_d = \mathbf{T}p$ or $\sigma_d = \mathbf{F}p$, respectively, we require $p \in (atom(\Pi) \cup body(\Pi)) \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$.
- For any literal $\sigma \in A$, $dl(\sigma)$ denotes the decision level of σ, viz. the value dl had when σ was assigned.
- A conflict is detected from violation of a nogood $\varepsilon \subseteq \Delta_\Pi \cup \nabla$.
- A conflict at decision level 0 (where A contains no heuristically chosen literals) indicates non-existence of answer sets.
- A nogood δ derived by conflict analysis is asserting, that is, some literal is unit-resulting for δ at a decision level $k < dl$.
  - ➥ After learning δ and backjumping to decision level k, at least one literal is newly derivable by unit propagation.
  - ☞ No explicit flipping of heuristically chosen literals !

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example: CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
2	$\mathbf{F}\{not\ x, not\ y\}$		
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$
3	$\mathbf{F}\{not\ y\}$		
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$
		$\vdots$	$\vdots$
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$ ✘

Example (ctd): CDNL-ASP

Consider

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

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
		$\mathbf{T}x$	$\{\mathbf{T}u, \mathbf{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{x\}\} \in \Delta(v)$
		$\mathbf{F}y$	$\{\mathbf{T}y, \mathbf{F}\{not\ x\}\} = \delta(y)$
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$

Example (ctd): CDNL-ASP

Consider

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

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
		$\mathbf{T}x$	$\{\mathbf{T}u, \mathbf{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{x\}\} \in \Delta(v)$
		$\mathbf{F}y$	$\{\mathbf{T}y, \mathbf{F}\{not\ x\}\} = \delta(y)$
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$

Example (ctd): CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{lll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y \end{array} \right. \quad w \leftarrow not\ x, not\ y \}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
		$\mathbf{T}x$	$\{\mathbf{T}u, \mathbf{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{x\}\} \in \Delta(v)$
		$\mathbf{F}y$	$\{\mathbf{T}y, \mathbf{F}\{not\ x\}\} = \delta(y)$
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$

Example (ctd): CDNL-ASP

Consider

$$\Pi = \left\{ \begin{array}{lll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y \end{array} \right. \quad w \leftarrow not\ x, not\ y \}$$

dl	σ_d	$\overline{\sigma}$	δ
1	$\mathbf{T}u$		
		$\mathbf{T}x$	$\{\mathbf{T}u, \mathbf{F}x\} \in \nabla$
		$\vdots$	$\vdots$
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{x\}\} \in \Delta(v)$
		$\mathbf{F}y$	$\{\mathbf{T}y, \mathbf{F}\{not\ x\}\} = \delta(y)$
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_Π and ∇;
  - Unfounded sets $U \subseteq atom(\Pi)$.
- Note that U is unfounded if $EB(U) \subseteq A^{\mathbf{F}}$.
  - ☞ For any $p \in U$, we have $(\lambda(p, U) \setminus \{\mathbf{T}p\}) \subseteq A$.
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}}) \ .$$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of Π.
  - ➥ Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $p \in U$, adding $\lambda(p, U)$ to ∇ triggers a conflict or further derivations by unit propagation.
  - ☞ Add loop nogoods atom by atom to eventually falsify all $p \in U$.

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_Π and ∇;
  - Unfounded sets $U \subseteq atom(\Pi)$.
- Note that U is unfounded if $EB(U) \subseteq A^{\mathbf{F}}$.
  - ☞ For any $p \in U$, we have $(\lambda(p, U) \setminus \{\mathbf{T}p\}) \subseteq A$.
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}}) \ .$$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of Π.
  - ➥ Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $p \in U$, adding $\lambda(p, U)$ to ∇ triggers a conflict or further derivations by unit propagation.
  - ☞ Add loop nogoods atom by atom to eventually falsify all $p \in U$.

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_Π and ∇;
  - Unfounded sets $U \subseteq atom(\Pi)$.
- Note that U is unfounded if $EB(U) \subseteq A^{\mathbf{F}}$.
  - ☞ For any $p \in U$, we have $(\lambda(p, U) \setminus \{\mathbf{T}p\}) \subseteq A$.
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}}) \ .$$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of Π.
  - ➥ Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $p \in U$, adding $\lambda(p, U)$ to ∇ triggers a conflict or further derivations by unit propagation.
  - ☞ Add loop nogoods atom by atom to eventually falsify all $p \in U$.

Outline of NogoodPropagation

- Derive deterministic consequences via:
  - Unit propagation on Δ_Π and ∇;
  - Unfounded sets $U \subseteq atom(\Pi)$.
- Note that U is unfounded if $EB(U) \subseteq A^{\mathbf{F}}$.
  - ☞ For any $p \in U$, we have $(\lambda(p, U) \setminus \{\mathbf{T}p\}) \subseteq A$.
- An "interesting" unfounded set U satisfies:

$$\emptyset \subset U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}}) \ .$$

- Wrt a fixpoint of unit propagation, such an unfounded set contains some loop of Π.
  - ➥ Tight programs do not yield "interesting" unfounded sets !
- Given an unfounded set U and some $p \in U$, adding $\lambda(p, U)$ to ∇ triggers a conflict or further derivations by unit propagation.
  - ☞ Add loop nogoods atom by atom to eventually falsify all $p \in U$.

Algorithm 2: NogoodPropagation

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

```
1  U ← ∅                                                    // set of unfounded atoms
2  loop
3      repeat
4          if δ ⊆ A for some δ ∈ Δ_Π ∪ ∇ then return (A, ∇)              // conflict
5          Σ ← {δ ∈ Δ_Π ∪ ∇ | (δ \ A) = {σ}, σ̄ ∉ A}                    // unit-resulting nogoods
6          if Σ ≠ ∅ then
7              let σ ∈ (δ \ A) for some δ ∈ Σ in
8                  A ← A ∘ (σ̄)          // dl(σ̄) = max({dl(ρ) | ρ ∈ (δ \ {σ})} ∪ {0})
9      until Σ = ∅
10     if Π is tight then return (A, ∇)   // no unfounded set ∅ ⊂ U ⊆ (atom(Π) \ A^F)
11     else
12         U ← (U \ A^F)
13         if U = ∅ then U ← UnfoundedSet(Π, A)
14         if U = ∅ then return (A, ∇)  // no unfounded set ∅ ⊂ U ⊆ (atom(Π) \ A^F)
15         let p ∈ U in
16             ∇ ← ∇ ∪ {λ(p, U)}    // record unit-resulting or violated loop nogood
```

Requirements for UnfoundedSet

- Implementations of UnfoundedSet must guarantee the following for a result U:
  1. $U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}})$;
  2. $EB(U) \subseteq A^{\mathbf{F}}$;
  3. $U = \emptyset$ iff there is no nonempty unfounded subset of $(atom(\Pi) \setminus A^{\mathbf{F}})$.
- Beyond that, there are various alternatives, such as:
  - Calculating the greatest unfounded set.
  - Calculating unfounded sets within strongly connected components of the positive atom dependency graph of Π.

    Usually, the latter option is implemented in ASP solvers !

Requirements for UnfoundedSet

- Implementations of UnfoundedSet must guarantee the following for a result U:
  1. $U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}})$;
  2. $EB(U) \subseteq A^{\mathbf{F}}$;
  3. $U = \emptyset$ iff there is no nonempty unfounded subset of $(atom(\Pi) \setminus A^{\mathbf{F}})$.
- Beyond that, there are various alternatives, such as:
  - Calculating the greatest unfounded set.
  - Calculating unfounded sets within strongly connected components of the positive atom dependency graph of Π.

☞ Usually, the latter option is implemented in ASP solvers !

Requirements for UnfoundedSet

- Implementations of UnfoundedSet must guarantee the following for a result U:
  1. $U \subseteq (atom(\Pi) \setminus A^{\mathbf{F}})$;
  2. $EB(U) \subseteq A^{\mathbf{F}}$;
  3. $U = \emptyset$ iff there is no nonempty unfounded subset of $(atom(\Pi) \setminus A^{\mathbf{F}})$.
- Beyond that, there are various alternatives, such as:
  - Calculating the greatest unfounded set.
  - Calculating unfounded sets within strongly connected components of the positive atom dependency graph of Π.
  - ☞ Usually, the latter option is implemented in ASP solvers !

Example: NogoodPropagation

Consider

$$\Pi = \left\{ \begin{array}{llll} x \leftarrow not\ y & u \leftarrow x, y & v \leftarrow x & w \leftarrow not\ x, not\ y \\ y \leftarrow not\ x & u \leftarrow v & v \leftarrow u, y & \end{array} \right.$$

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Outline of ConflictAnalysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_\Pi \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$.
- ☞ Note that all but the first literal assigned at dl have been unit-resulting for nogoods $\varepsilon \in \Delta_\Pi \cup \nabla$.
  - ➥ If $\sigma \in \delta$ has been unit-resulting for ε, we obtain a new violated nogood by resolving δ and ε as follows:

$$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\}) \ .$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$.
  - ☞ Iterated resolution progresses in inverse order of assignment.
- Iterated resolution stops as soon as it generates a nogood δ containing exactly one literal σ assigned at decision level dl.
  - This literal σ is called First Unique Implication Point (First-UIP).
  - ☞ All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than dl.

Outline of ConflictAnalysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_\Pi \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$.
- ☞ Note that all but the first literal assigned at dl have been unit-resulting for nogoods $\varepsilon \in \Delta_\Pi \cup \nabla$.
  - ➥ If $\sigma \in \delta$ has been unit-resulting for ε, we obtain a new violated nogood by resolving δ and ε as follows:

$$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\}) \ .$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$.
  - ☞ Iterated resolution progresses in inverse order of assignment.
- Iterated resolution stops as soon as it generates a nogood δ containing exactly one literal σ assigned at decision level dl.
  - This literal σ is called First Unique Implication Point (First-UIP).
  - ☞ All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than dl.

Outline of ConflictAnalysis

- Conflict analysis is triggered whenever some nogood $\delta \in \Delta_\Pi \cup \nabla$ becomes violated, viz. $\delta \subseteq A$, at a decision level $dl > 0$.
- ☞ Note that all but the first literal assigned at dl have been unit-resulting for nogoods $\varepsilon \in \Delta_\Pi \cup \nabla$.
  - ➥ If $\sigma \in \delta$ has been unit-resulting for ε, we obtain a new violated nogood by resolving δ and ε as follows:

$$(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\}) \ .$$

- Resolution is directed by resolving first over the literal $\sigma \in \delta$ derived last, viz. $(\delta \setminus A[\sigma]) = \{\sigma\}$.
  - ☞ Iterated resolution progresses in inverse order of assignment.
- Iterated resolution stops as soon as it generates a nogood δ containing exactly one literal σ assigned at decision level dl.
  - This literal σ is called First Unique Implication Point (First-UIP).
  - ☞ All literals in $(\delta \setminus \{\sigma\})$ are assigned at decision levels smaller than dl.

Algorithm 3: ConflictAnalysis

Input : A violated nogood δ, a logic program Π, a set ∇ of nogoods, and an assignment A.

Output : A derived nogood and a decision level.

```
1 loop
2     let σ ∈ δ such that (δ \ A[σ]) = {σ} in
3         k ← max({dl(ρ) | ρ ∈ δ \ {σ}} ∪ {0})
4         if k = dl(σ) then
5             let ε ∈ Δ_Π ∪ ∇ such that (ε \ A[σ]) = {σ̄} in
6                 δ ← (δ \ {σ}) ∪ (ε \ {σ̄})                    // resolution
7         else return (δ, k)
```

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Example: ConflictAnalysis

Consider

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

dl	σ_d	$\overline{\sigma}$	δ	
1	$\mathbf{T}u$			
2	$\mathbf{F}\{not\ x, not\ y\}$			
		$\mathbf{F}w$	$\{\mathbf{T}w, \mathbf{F}\{not\ x, not\ y\}\} = \delta(w)$	
3	$\mathbf{F}\{not\ y\}$			
		$\mathbf{F}x$	$\{\mathbf{T}x, \mathbf{F}\{not\ y\}\} = \delta(x)$	
		$\mathbf{F}\{x\}$	$\{\mathbf{T}\{x\}, \mathbf{F}x\} \in \Delta(\{x\})$	$\{\mathbf{T}u, \mathbf{F}x\}$
		$\mathbf{F}\{x, y\}$	$\{\mathbf{T}\{x, y\}, \mathbf{F}x\} \in \Delta(\{x, y\})$	$\{\mathbf{T}u, \mathbf{F}x, \mathbf{F}\{x\}\}$
		$\mathbf{T}\{not\ x\}$	$\{\mathbf{F}\{not\ x\}, \mathbf{F}x\} = \delta(\{not\ x\})$	
		$\mathbf{T}y$	$\{\mathbf{F}\{not\ y\}, \mathbf{F}y\} = \delta(\{not\ y\})$	
		$\mathbf{T}\{v\}$	$\{\mathbf{T}u, \mathbf{F}\{x, y\}, \mathbf{F}\{v\}\} = \delta(u)$	
		$\mathbf{T}\{u, y\}$	$\{\mathbf{F}\{u, y\}, \mathbf{T}u, \mathbf{T}y\} = \delta(\{u, y\})$	
		$\mathbf{T}v$	$\{\mathbf{F}v, \mathbf{T}\{u, y\}\} \in \Delta(v)$	
			$\{\mathbf{T}u, \mathbf{F}\{x\}, \mathbf{F}\{x, y\}\} = \lambda(u, \{u, v\})$	✘

Remarks

- There always is a First-UIP at which conflict analysis terminates.
- ☞ In the worst, resolution stops at the heuristically chosen literal assigned at decision level dl.
  - The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$.
  - We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$.
    - ➥ After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
      - ☞ Such a nogood δ is called asserting.
- ☞ Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

Remarks

- There always is a First-UIP at which conflict analysis terminates.
- ☞ In the worst, resolution stops at the heuristically chosen literal assigned at decision level dl.
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$.
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$.
  - ➥ After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - ☞ Such a nogood δ is called asserting.
- ☞ Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

Remarks

- There always is a First-UIP at which conflict analysis terminates.
- ☞ In the worst, resolution stops at the heuristically chosen literal assigned at decision level dl.
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$.
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$.
  - ➥ After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - ☞ Such a nogood δ is called asserting.
- ☞ Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

Remarks

- There always is a First-UIP at which conflict analysis terminates.
- ☞ In the worst, resolution stops at the heuristically chosen literal assigned at decision level dl.
- The nogood δ containing First-UIP σ is violated by A, viz. $\delta \subseteq A$.
- We have $k = max(\{dl(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) < dl$.
  - ➥ After recording δ in ∇ and backjumping to decision level k, $\overline{\sigma}$ is unit-resulting for δ !
  - ☞ Such a nogood δ is called asserting.
- ☞ Asserting nogoods direct conflict-driven search into a different region of the search space than traversed before, without explicitly flipping any heuristically chosen literal !

The clasp system [32]

- Native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - Advanced preprocessing including, e.g., equivalence reasoning
  - Lookback-based decision heuristics
  - Restart policies
  - Nogood deletion
  - Progress saving
  - Dedicated data structures for binary and ternary nogoods
  - Lazy data structures (watched literals) for long nogoods
  - Dedicated data structures for cardinality and weight constraints
  - Lazy unfounded set checking based on "source pointers"
  - Tight integration of unit propagation and unfounded set checking
  - Reasoning modes
  - ...

☞ Many of these techniques are configurable !

The clasp system [32]

- Native ASP solver combining conflict-driven search with sophisticated reasoning techniques:
  - Advanced preprocessing including, e.g., equivalence reasoning
  - Lookback-based decision heuristics
  - Restart policies
  - Nogood deletion
  - Progress saving
  - Dedicated data structures for binary and ternary nogoods
  - Lazy data structures (watched literals) for long nogoods
  - Dedicated data structures for cardinality and weight constraints
  - Lazy unfounded set checking based on "source pointers"
  - Tight integration of unit propagation and unfounded set checking
  - Reasoning modes
  - ...
- ☞ Many of these techniques are configurable !

Reasoning modes of clasp

Beyond deciding answer set existence, clasp allows for:

- Optimization
- Enumeration [without solution recording]
- Projective Enumeration [without solution recording]
- Brave and Cautious Reasoning determining the
  - union or
  - intersection

  of all answer sets by computing only linearly many of them

☞ Reasoning applicable wrt answer sets as well as supported models

Front-ends also admit clasp to solve:

- Propositional CNF formulas
- Pseudo-Boolean formulas

Find clasp at: http://potassco.sourceforge.net

Reasoning modes of clasp

Beyond deciding answer set existence, clasp allows for:

- Optimization
- Enumeration [without solution recording]
- Projective Enumeration [without solution recording]
- Brave and Cautious Reasoning determining the
  - union or
  - intersection

  of all answer sets by computing only linearly many of them

☞ Reasoning applicable wrt answer sets as well as supported models

Front-ends also admit clasp to solve:

- Propositional CNF formulas
- Pseudo-Boolean formulas

Find clasp at: http://potassco.sourceforge.net

Reasoning modes of clasp

Beyond deciding answer set existence, clasp allows for:

- Optimization
- Enumeration [without solution recording]
- Projective Enumeration [without solution recording]
- Brave and Cautious Reasoning determining the
  - union or
  - intersection

  of all answer sets by computing only linearly many of them

☞ Reasoning applicable wrt answer sets as well as supported models

Front-ends also admit clasp to solve:

- Propositional CNF formulas
- Pseudo-Boolean formulas

Find clasp at: http://potassco.sourceforge.net

Equivalence
Overview

Equivalence of Logic Programs

[53, 76]

- Motivation
- Ordinary Equivalence
- Strong Equivalence
- Uniform Equivalence

Motivation

Questions

- How to optimize logic programs?
- How to remove redundancies in automatically generated logic programs?

Difficulty

Given that ASP is nonmonotonic,
it is difficult to attribute meaning to

- program parts or
- incomplete programs

because the addition of further rules generally changes the overall semantics.

Motivation

Questions

- How to optimize logic programs?
- How to remove redundancies in automatically generated logic programs?

Difficulty Given that ASP is nonmonotonic, it is difficult to attribute meaning to

- program parts or
- incomplete programs

because the addition of further rules generally changes the overall semantics.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow not\ b\ , b \leftarrow not\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\}, \{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ , b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow not\ b\ , b \leftarrow not\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\}, \{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ , b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ ,\ b \leftarrow \mathit{not}\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\},\{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ ,\ b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\}, \{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ , b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow not\ b\ , b \leftarrow not\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\}, \{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ , b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\}, \{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ , b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Notions of Equivalence

Two logic programs Π_1 and Π_2 are

- equivalent ($\Pi_1 \equiv \Pi_2$) if $AS(\Pi_1) = AS(\Pi_2)$.
- strongly equivalent ($\Pi_1 \equiv_s \Pi_2$) if $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'.
- uniformly equivalent ($\Pi_1 \equiv_u \Pi_2$) if $AS(\Pi_1 \cup F) = AS(\Pi_2 \cup F)$ for any set F of facts.

Example $\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow not\ b\ , b \leftarrow not\ a\}$

- $\Pi_1 \equiv \Pi_2$ since $AS(\Pi_1) = \{\{a\}, \{b\}\} = AS(\Pi_2)$
- $\Pi_1 \not\equiv_s \Pi_2$, e.g. $\Pi' = \{a \leftarrow b\ , b \leftarrow a\}$
- $\Pi_1 \equiv_u \Pi_2$

Implications

- strong equivalence implies uniform equivalence and
- uniform equivalence implies (ordinary) equivalence.

Ordinary Equivalence

- Consider $\Pi_1 = \{a \leftarrow b\}$ and $\Pi_2 = \{a \leftarrow c\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Consider $\Pi_1 = \{a \leftarrow \mathit{not}\ b\}$ and $\Pi_2 = \{a \leftarrow\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Ordinary equivalence in ASP does not allow for substitution of equivalents:

  $$\Pi_1 \equiv \Pi_2 \quad \text{not implies} \quad \Pi \equiv \Pi[\Pi_1/\Pi_2],$$

  for any logic programs Π_1, Π_2, and Π.

  - The non-monotonicity of ASP makes equivalence of programs a much weaker concept than equivalence in (monotonic) classical logic.

Ordinary Equivalence

- Consider $\Pi_1 = \{a \leftarrow b\}$ and $\Pi_2 = \{a \leftarrow c\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Consider $\Pi_1 = \{a \leftarrow \mathit{not}\ b\}$ and $\Pi_2 = \{a \leftarrow\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Ordinary equivalence in ASP does not allow for substitution of equivalents:

  $$\Pi_1 \equiv \Pi_2 \quad \text{not implies} \quad \Pi \equiv \Pi[\Pi_1/\Pi_2],$$

  for any logic programs Π_1, Π_2, and Π.

  - The non-monotonicity of ASP makes equivalence of programs a much weaker concept than equivalence in (monotonic) classical logic.

Ordinary Equivalence

- Consider $\Pi_1 = \{a \leftarrow b\}$ and $\Pi_2 = \{a \leftarrow c\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Consider $\Pi_1 = \{a \leftarrow \mathit{not}\ b\}$ and $\Pi_2 = \{a \leftarrow\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Ordinary equivalence in ASP does not allow for substitution of equivalents:

  $$\Pi_1 \equiv \Pi_2 \quad \text{not implies} \quad \Pi \equiv \Pi[\Pi_1/\Pi_2],$$

  for any logic programs Π_1, Π_2, and Π.

  - The non-monotonicity of ASP makes equivalence of programs a much weaker concept than equivalence in (monotonic) classical logic.

Ordinary Equivalence

- Consider $\Pi_1 = \{a \leftarrow b\}$ and $\Pi_2 = \{a \leftarrow c\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Consider $\Pi_1 = \{a \leftarrow \mathit{not}\ b\}$ and $\Pi_2 = \{a \leftarrow\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Ordinary equivalence in ASP does not allow for substitution of equivalents:

  $$\Pi_1 \equiv \Pi_2 \quad \text{not implies} \quad \Pi \equiv \Pi[\Pi_1/\Pi_2],$$

  for any logic programs Π_1, Π_2, and Π.

  ➥ The non-monotonicity of ASP makes equivalence of programs a much weaker concept than equivalence in (monotonic) classical logic.

Ordinary Equivalence

- Consider $\Pi_1 = \{a \leftarrow b\}$ and $\Pi_2 = \{a \leftarrow c\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Consider $\Pi_1 = \{a \leftarrow \mathit{not}\ b\}$ and $\Pi_2 = \{a \leftarrow\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Ordinary equivalence in ASP does not allow for substitution of equivalents:

  $$\Pi_1 \equiv \Pi_2 \quad \text{not implies} \quad \Pi \equiv \Pi[\Pi_1/\Pi_2],$$

  for any logic programs Π_1, Π_2, and Π.

  ➥ The non-monotonicity of ASP makes equivalence of programs a much weaker concept than equivalence in (monotonic) classical logic.

Ordinary Equivalence

- Consider $\Pi_1 = \{a \leftarrow b\}$ and $\Pi_2 = \{a \leftarrow c\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Consider $\Pi_1 = \{a \leftarrow \mathit{not}\ b\}$ and $\Pi_2 = \{a \leftarrow\}$.

  $$\Pi_1 \equiv \Pi_2 \quad \text{but} \quad (\Pi_1 \cup \{b \leftarrow\}) \not\equiv (\Pi_2 \cup \{b \leftarrow\}).$$

- Ordinary equivalence in ASP does not allow for substitution of equivalents:

  $$\Pi_1 \equiv \Pi_2 \quad \text{not implies} \quad \Pi \equiv \Pi[\Pi_1/\Pi_2],$$

  for any logic programs Π_1, Π_2, and Π.

  ↪ The non-monotonicity of ASP makes equivalence of programs a much weaker concept than equivalence in (monotonic) classical logic.

Strong Equivalence

- Two logic programs Π_1 and Π_2 are strongly equivalent if

  $$(\Pi_1 \cup \Pi') \equiv (\Pi_2 \cup \Pi') \text{ for any logic programs } \Pi \ .$$

- Strong Equivalence (SE) guarantees substitution of equivalents.
- How to test strong equivalence?
  - How to avoid testing $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'?

Strong Equivalence

- Two logic programs Π_1 and Π_2 are strongly equivalent if

  $$(\Pi_1 \cup \Pi') \equiv (\Pi_2 \cup \Pi') \text{ for any logic programs } \Pi \ .$$

- Strong Equivalence (SE) guarantees substitution of equivalents.
- How to test strong equivalence?
  - ➥ How to avoid testing $AS(\Pi_1 \cup \Pi') = AS(\Pi_2 \cup \Pi')$ for any logic program Π'?

SE-models

Model-theoretic characterization of Strong Equivalence.

Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation over $\mathcal{A}$ is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$
- An SE-interpretation (X, Y) is an SE-model of Π if
  1. $Y \models \Pi$
  2. $X \models \Pi^Y$
- $SE(\Pi)$ denotes the set of all SE-models of Π

Theorem $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$

Observation If (X, X) is the unique SE-model of Π whose second component is X, then X is an answer set of Π.

SE-models

Model-theoretic characterization of Strong Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation over $\mathcal{A}$ is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$
- An SE-interpretation (X, Y) is an SE-model of Π if
  1. $Y \models \Pi$
  2. $X \models \Pi^Y$
- $SE(\Pi)$ denotes the set of all SE-models of Π

Theorem $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$

Observation If (X, X) is the unique SE-model of Π whose second component is X, then X is an answer set of Π.

SE-models

Model-theoretic characterization of Strong Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation over $\mathcal{A}$ is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$
- An SE-interpretation (X, Y) is an SE-model of Π if
  1. $Y \models \Pi$
  2. $X \models \Pi^Y$
- $SE(\Pi)$ denotes the set of all SE-models of Π

Theorem $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$

Observation If (X, X) is the unique SE-model of Π whose second component is X, then X is an answer set of Π.

SE-models

Model-theoretic characterization of Strong Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation over $\mathcal{A}$ is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$
- An SE-interpretation (X, Y) is an SE-model of Π if
  1. $Y \models \Pi$
  2. $X \models \Pi^Y$
- $SE(\Pi)$ denotes the set of all SE-models of Π

Theorem $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$

Observation If (X, X) is the unique SE-model of Π whose second component is X, then X is an answer set of Π.

SE-models

Model-theoretic characterization of Strong Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation over $\mathcal{A}$ is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$
- An SE-interpretation (X, Y) is an SE-model of Π if
  1. $Y \models \Pi$
  2. $X \models \Pi^Y$
- $SE(\Pi)$ denotes the set of all SE-models of Π

Theorem $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$

Observation If (X, X) is the unique SE-model of Π whose second component is X, then X is an answer set of Π.

SE-models

Model-theoretic characterization of Strong Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation over $\mathcal{A}$ is a pair (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$
- An SE-interpretation (X, Y) is an SE-model of Π if
  1. $Y \models \Pi$
  2. $X \models \Pi^Y$
- $SE(\Pi)$ denotes the set of all SE-models of Π

Theorem $\Pi_1 \equiv_s \Pi_2$ iff $SE(\Pi_1) = SE(\Pi_2)$

Observation If (X, X) is the unique SE-model of Π whose second component is X, then X is an answer set of Π.

Example: SE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\ \}$

We get the following SE-models over $\{a, b\}$:

$$SE(\Pi_1) = \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b\}$$

$$SE(\Pi_2) = SE(\Pi_1) \cup \{(\emptyset,\{a,b\})\}$$

We have

- $SE(\Pi_1) \neq SE(\Pi_2)$ implies $\Pi_1 \not\equiv_s \Pi_2$
- **Counterexample** Take $\Pi' = \{a \leftarrow b\ , b \leftarrow a\ \}$

Example: SE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\ \}$
We get the following SE-models over $\{a, b\}$:

$$SE(\Pi_1) = \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b\}$$

$$SE(\Pi_2) = SE(\Pi_1) \cup \{(\emptyset,\{a,b\})\}$$

We have

- $SE(\Pi_1) \neq SE(\Pi_2)$ implies $\Pi_1 \not\equiv_s \Pi_2$
- **Counterexample** Take $\Pi' = \{a \leftarrow b\ , b \leftarrow a\ \}$

Example: SE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\ \}$
We get the following SE-models over $\{a, b\}$:

$$SE(\Pi_1) = \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b\}$$

$$SE(\Pi_2) = SE(\Pi_1) \cup \{(\emptyset,\{a,b\})\}$$

We have

- $SE(\Pi_1) \neq SE(\Pi_2)$ implies $\Pi_1 \not\equiv_s \Pi_2$
- **Counterexample** Take $\Pi' = \{a \leftarrow b\ , b \leftarrow a\ \}$

Example: SE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\ \}$
We get the following SE-models over $\{a, b\}$:

$$SE(\Pi_1) = \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b\}$$

$$SE(\Pi_2) = SE(\Pi_1) \cup \{(\emptyset,\{a,b\})\}$$

We have

- $SE(\Pi_1) \neq SE(\Pi_2)$ implies $\Pi_1 \not\equiv_s \Pi_2$
- **Counterexample** Take $\Pi' = \{a \leftarrow b\ , b \leftarrow a\ \}$

Example: SE-models

$\Pi_1 = \{a \leftarrow\}$ and $\Pi_2 = \{a \leftarrow , a \leftarrow b , a \leftarrow \mathit{not}\ c\}$
We get the following SE-models over $\{a, b, c\}$:

$$\begin{aligned} SE(\Pi_1) &= \{(\{a\},\{a\}), (\{a\},\{a,b\}), (\{a\},\{a,c\}), \\ &\quad (\{a\},\{a,b,c\}), (\{a,b\},\{a,b\}), (\{a,b\},\{a,b,c\}), \\ &\quad (\{a,c\},\{a,c\}), (\{a,c\},\{a,b,c\}), (\{a,b,c\},\{a,b,c\})\} \\ SE(\Pi_2) &= SE(\Pi_1) \end{aligned}$$

Observation For rules r_1 and r_2, we have $\{r_1\} \equiv_s \{r_1, r_2\}$ whenever $SE(\{r_1\}) \subseteq SE(\{r_2\})$

Example

- $SE(\{r_1\}) \subseteq SE(\{r_2\})$ holds for any rules where $body(r_1) \subseteq body(r_2)$
  - In any program, delete a rule r_2 whenever there is some rule r_1 such that $body(r_1) \subseteq body(r_2)$.

Example: SE-models

$\Pi_1 = \{a \leftarrow\}$ and $\Pi_2 = \{a \leftarrow , a \leftarrow b , a \leftarrow \mathit{not}\ c\}$
We get the following SE-models over $\{a, b, c\}$:

$$\begin{aligned} SE(\Pi_1) &= \{(\{a\},\{a\}), (\{a\},\{a,b\}), (\{a\},\{a,c\}), \\ &\quad (\{a\},\{a,b,c\}), (\{a,b\},\{a,b\}), (\{a,b\},\{a,b,c\}), \\ &\quad (\{a,c\},\{a,c\}), (\{a,c\},\{a,b,c\}), (\{a,b,c\},\{a,b,c\})\} \\ SE(\Pi_2) &= SE(\Pi_1) \end{aligned}$$

Observation For rules r_1 and r_2, we have $\{r_1\} \equiv_s \{r_1, r_2\}$ whenever $SE(\{r_1\}) \subseteq SE(\{r_2\})$

Example

- $SE(\{r_1\}) \subseteq SE(\{r_2\})$ holds for any rules where $body(r_1) \subseteq body(r_2)$
  - In any program, delete a rule r_2 whenever there is some rule r_1 such that $body(r_1) \subseteq body(r_2)$.

Example: SE-models

$\Pi_1 = \{a \leftarrow\}$ and $\Pi_2 = \{a \leftarrow , a \leftarrow b , a \leftarrow \mathit{not}\ c\}$
We get the following SE-models over $\{a, b, c\}$:

$$\begin{aligned} SE(\Pi_1) &= \{(\{a\},\{a\}), (\{a\},\{a,b\}), (\{a\},\{a,c\}), \\ &\quad (\{a\},\{a,b,c\}), (\{a,b\},\{a,b\}), (\{a,b\},\{a,b,c\}), \\ &\quad (\{a,c\},\{a,c\}), (\{a,c\},\{a,b,c\}), (\{a,b,c\},\{a,b,c\})\} \\ SE(\Pi_2) &= SE(\Pi_1) \end{aligned}$$

Observation For rules r_1 and r_2, we have $\{r_1\} \equiv_s \{r_1, r_2\}$ whenever $SE(\{r_1\}) \subseteq SE(\{r_2\})$

Example

- $SE(\{r_1\}) \subseteq SE(\{r_2\})$ holds for any rules where $body(r_1) \subseteq body(r_2)$
  - In any program, delete a rule r_2 whenever there is some rule r_1 such that $body(r_1) \subseteq body(r_2)$.

Example: SE-models

$\Pi_1 = \{a \leftarrow\}$ and $\Pi_2 = \{a \leftarrow , a \leftarrow b , a \leftarrow \mathit{not}\ c\}$
We get the following SE-models over $\{a, b, c\}$:

$$\begin{aligned} SE(\Pi_1) &= \{(\{a\},\{a\}), (\{a\},\{a,b\}), (\{a\},\{a,c\}), \\ &\quad (\{a\},\{a,b,c\}), (\{a,b\},\{a,b\}), (\{a,b\},\{a,b,c\}), \\ &\quad (\{a,c\},\{a,c\}), (\{a,c\},\{a,b,c\}), (\{a,b,c\},\{a,b,c\})\} \\ SE(\Pi_2) &= SE(\Pi_1) \end{aligned}$$

Observation For rules r_1 and r_2, we have $\{r_1\} \equiv_s \{r_1, r_2\}$ whenever $SE(\{r_1\}) \subseteq SE(\{r_2\})$

Example

- $SE(\{r_1\}) \subseteq SE(\{r_2\})$ holds for any rules where $body(r_1) \subseteq body(r_2)$
  - In any program, delete a rule r_2 whenever there is some rule r_1 such that $body(r_1) \subseteq body(r_2)$.

Example: SE-models

$\Pi_1 = \{a \leftarrow\}$ and $\Pi_2 = \{a \leftarrow , a \leftarrow b , a \leftarrow \mathit{not}\ c\}$
We get the following SE-models over $\{a, b, c\}$:

$$\begin{aligned} SE(\Pi_1) &= \{(\{a\},\{a\}), (\{a\},\{a,b\}), (\{a\},\{a,c\}), \\ &\quad (\{a\},\{a,b,c\}), (\{a,b\},\{a,b\}), (\{a,b\},\{a,b,c\}), \\ &\quad (\{a,c\},\{a,c\}), (\{a,c\},\{a,b,c\}), (\{a,b,c\},\{a,b,c\})\} \end{aligned}$$

$$SE(\Pi_2) = SE(\Pi_1)$$

Observation For rules r_1 and r_2, we have $\{r_1\} \equiv_s \{r_1, r_2\}$ whenever $SE(\{r_1\}) \subseteq SE(\{r_2\})$

Example

- $SE(\{r_1\}) \subseteq SE(\{r_2\})$ holds for any rules where $body(r_1) \subseteq body(r_2)$
  - In any program, delete a rule r_2 whenever there is some rule r_1 such that $body(r_1) \subseteq body(r_2)$.

Example: SE-models

$\Pi_1 = \{a \leftarrow\}$ and $\Pi_2 = \{a \leftarrow , a \leftarrow b , a \leftarrow not\ c\}$
We get the following SE-models over $\{a, b, c\}$:

$$\begin{aligned} SE(\Pi_1) &= \{(\{a\},\{a\}), (\{a\},\{a,b\}), (\{a\},\{a,c\}), \\ &\quad (\{a\},\{a,b,c\}), (\{a,b\},\{a,b\}), (\{a,b\},\{a,b,c\}), \\ &\quad (\{a,c\},\{a,c\}), (\{a,c\},\{a,b,c\}), (\{a,b,c\},\{a,b,c\})\} \end{aligned}$$

$$SE(\Pi_2) = SE(\Pi_1)$$

Observation For rules r_1 and r_2, we have $\{r_1\} \equiv_s \{r_1, r_2\}$ whenever $SE(\{r_1\}) \subseteq SE(\{r_2\})$

Example

- $SE(\{r_1\}) \subseteq SE(\{r_2\})$ holds for any rules where $body(r_1) \subseteq body(r_2)$
  - ➥ In any program, delete a rule r_2 whenever there is some rule r_1 such that $body(r_1) \subseteq body(r_2)$.

Strong Equivalence

Normal versus Disjunctive logic programs

Reduct-Intersection Let Π be a normal program.
If $(U, Y) \in SE(\Pi)$ and $(V, Y) \in SE(\Pi)$, then $(U \cap V, Y) \in SE(\Pi)$.

(☞ Since for any X, Π^X is a Horn program.)

- Reduct-Intersection is not satisfied by disjunctive logic programs.
- If the SE-models of a disjunctive program do not satisfy reduct-intersection, then no strongly equivalent normal programs exists.

Strong Equivalence

Normal versus Disjunctive logic programs

Reduct-Intersection Let Π be a normal program.

If $(U, Y) \in SE(\Pi)$ and $(V, Y) \in SE(\Pi)$, then $(U \cap V, Y) \in SE(\Pi)$.

(☞ Since for any X, Π^X is a Horn program.)

- Reduct-Intersection is not satisfied by disjunctive logic programs.
- If the SE-models of a disjunctive program do not satisfy reduct-intersection, then no strongly equivalent normal programs exists.

Strong Equivalence

Normal versus Disjunctive logic programs

Reduct-Intersection Let Π be a normal program.

If $(U, Y) \in SE(\Pi)$ and $(V, Y) \in SE(\Pi)$, then $(U \cap V, Y) \in SE(\Pi)$.

(☞ Since for any X, Π^X is a Horn program.)

- Reduct-Intersection is not satisfied by disjunctive logic programs.
- If the SE-models of a disjunctive program do not satisfy reduct-intersection, then no strongly equivalent normal programs exists.

Strong Equivalence

Normal versus Disjunctive logic programs

Reduct-Intersection Let Π be a normal program.

If $(U, Y) \in SE(\Pi)$ and $(V, Y) \in SE(\Pi)$, then $(U \cap V, Y) \in SE(\Pi)$.

(☞ Since for any X, Π^X is a Horn program.)

- Reduct-Intersection is not satisfied by disjunctive logic programs.
- If the SE-models of a disjunctive program do not satisfy reduct-intersection, then no strongly equivalent normal programs exists.

Example

- Recall program $\Pi_1 = \{a \vee b \leftarrow\}$ along with

$$SE(\Pi_1) = \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b\},$$

- $SE(\Pi_1)$ is not closed under reduct-intersection, since $(\{a\},\{a,b\})$ and $(\{b\},\{a,b\})$ call for $(\emptyset,\{a,b\})$.
  - ➥ No normal program is strongly equivalent to $\{a \vee b \leftarrow\}$.

UE-models

Model-theoretic characterization of Uniform Equivalence.

Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation (X, Y) is a UE-model of Π if
  1. $(X, Y) \in SE(\Pi)$ and
  2. for each Z with $X \subset Z \subset Y$, we have $(Z, Y) \notin SE(\Pi)$
- $UE(\Pi)$ denotes the set of all UE-models of Π

Theorem $\Pi_1 \equiv_u \Pi_2$ iff $UE(\Pi_1) = UE(\Pi_2)$

Observation UE-models of a program Π are

- all SE-models (X, X) of Π,
- all further SE-models (X, Y) of Π, where $X \subset Y$ is maximal in being a model of Π^Y.

UE-models

Model-theoretic characterization of Uniform Equivalence.

Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation (X, Y) is a UE-model of Π if
  1. $(X, Y) \in SE(\Pi)$ and
  2. for each Z with $X \subset Z \subset Y$, we have $(Z, Y) \notin SE(\Pi)$
- $UE(\Pi)$ denotes the set of all UE-models of Π

Theorem $\Pi_1 \equiv_u \Pi_2$ iff $UE(\Pi_1) = UE(\Pi_2)$

Observation UE-models of a program Π are

- all SE-models (X, X) of Π,
- all further SE-models (X, Y) of Π, where $X \subset Y$ is maximal in being a model of Π^Y.

UE-models

Model-theoretic characterization of Uniform Equivalence.

Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation (X, Y) is a UE-model of Π if
  1. $(X, Y) \in SE(\Pi)$ and
  2. for each Z with $X \subset Z \subset Y$, we have $(Z, Y) \notin SE(\Pi)$
- $UE(\Pi)$ denotes the set of all UE-models of Π

Theorem $\Pi_1 \equiv_u \Pi_2$ iff $UE(\Pi_1) = UE(\Pi_2)$

Observation UE-models of a program Π are

- all SE-models (X, X) of Π,
- all further SE-models (X, Y) of Π, where $X \subset Y$ is maximal in being a model of Π^Y.

UE-models

Model-theoretic characterization of Uniform Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation (X, Y) is a UE-model of Π if
  1. $(X, Y) \in SE(\Pi)$ and
  2. for each Z with $X \subset Z \subset Y$, we have $(Z, Y) \notin SE(\Pi)$
- $UE(\Pi)$ denotes the set of all UE-models of Π

Theorem $\Pi_1 \equiv_u \Pi_2$ iff $UE(\Pi_1) = UE(\Pi_2)$

Observation UE-models of a program Π are

- all SE-models (X, X) of Π,
- all further SE-models (X, Y) of Π, where $X \subset Y$ is maximal in being a model of Π^Y.

UE-models

Model-theoretic characterization of Uniform Equivalence.
Let Π be a logic program over alphabet $\mathcal{A}$.

- An SE-interpretation (X, Y) is a UE-model of Π if
  1. $(X, Y) \in SE(\Pi)$ and
  2. for each Z with $X \subset Z \subset Y$, we have $(Z, Y) \notin SE(\Pi)$
- $UE(\Pi)$ denotes the set of all UE-models of Π

Theorem $\Pi_1 \equiv_u \Pi_2$ iff $UE(\Pi_1) = UE(\Pi_2)$

Observation UE-models of a program Π are

- all SE-models (X, X) of Π,
- all further SE-models (X, Y) of Π, where $X \subset Y$ is maximal in being a model of Π^Y.

Example: UE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\}$

$$
\begin{aligned}
UE(\Pi_1) &= SE(\Pi_1) \\
&= \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b
\end{aligned}
$$

$$
\begin{aligned}
UE(\Pi_2) &= SE(\Pi_2) \setminus \{(\emptyset,\{a,b\})\} \\
&= SE(\Pi_1)
\end{aligned}
$$

We have

- $UE(\Pi_1) = UE(\Pi_2)$ implies $\Pi_1 \equiv_u \Pi_2$ and $\Pi_1 \equiv \Pi_2$ although $SE(\Pi_1) \neq SE(\Pi_2)$.
- Note that the SE-model $(\emptyset,\{a,b\})$ is no UE-model of Π_2, since $(\{a\},\{a,b\})$ is an UE-model of Π_2.

Example: UE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\ \}$

$$
\begin{aligned}
UE(\Pi_1) &= SE(\Pi_1) \\
&= \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b
\end{aligned}
$$

$$
\begin{aligned}
UE(\Pi_2) &= SE(\Pi_2) \setminus \{(\emptyset,\{a,b\})\} \\
&= SE(\Pi_1)
\end{aligned}
$$

We have

- $UE(\Pi_1) = UE(\Pi_2)$ implies $\Pi_1 \equiv_u \Pi_2$ and $\Pi_1 \equiv \Pi_2$ although $SE(\Pi_1) \neq SE(\Pi_2)$.
- Note that the SE-model $(\emptyset,\{a,b\})$ is no UE-model of Π_2, since $(\{a\},\{a,b\})$ is an UE-model of Π_2.

Example: UE-models

$\Pi_1 = \{a \vee b \leftarrow\}$ and $\Pi_2 = \{a \leftarrow \mathit{not}\ b\ , b \leftarrow \mathit{not}\ a\ \}$

$$
\begin{aligned}
UE(\Pi_1) &= SE(\Pi_1) \\
&= \{(\{a\},\{a\}), (\{b\},\{b\}), (\{a\},\{a,b\}), (\{b\},\{a,b\}), (\{a,b
\end{aligned}
$$

$$
\begin{aligned}
UE(\Pi_2) &= SE(\Pi_2) \setminus \{(\emptyset,\{a,b\})\} \\
&= SE(\Pi_1)
\end{aligned}
$$

We have

- $UE(\Pi_1) = UE(\Pi_2)$ implies $\Pi_1 \equiv_u \Pi_2$ and $\Pi_1 \equiv \Pi_2$ although $SE(\Pi_1) \neq SE(\Pi_2)$.
- Note that the SE-model $(\emptyset,\{a,b\})$ is no UE-model of Π_2, since $(\{a\},\{a,b\})$ is an UE-model of Π_2.

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset$, $head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations

Let be Π a (disjunctive) logic program.

TAUT if $head(r) \cap body^+(r) \neq \emptyset$ then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{a \leftarrow , a \leftarrow a\} \equiv_s \{a \leftarrow\}$

RED$^-$ $r_1, r_2 \in \Pi$, $body(r_2) = \emptyset, head(r_2) \subseteq body^-(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , b \leftarrow not\ a\} \equiv_s \{a \leftarrow\}$

NONMIN $r_1, r_2 \in \Pi, head(r_2) \subseteq head(r_1), body(r_2) \subseteq body(r_1)$, then $\Pi \equiv_s \Pi \setminus \{r_1\}$ and $\Pi \equiv_u \Pi \setminus \{r_1\}$,
e.g. $\{a \leftarrow , a \leftarrow b\} \equiv_s \{a \leftarrow\}$

CONTRA $body^+(r) \cap body^-(r) \neq \emptyset$, then $\Pi \equiv_s \Pi \setminus \{r\}$ and $\Pi \equiv_u \Pi \setminus \{r\}$,
e.g. $\{b \leftarrow a, not\ a\} \equiv_s \emptyset$

Program Transformations (ctd)

WGPPE $r_1 \in \Pi$, $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid head(r_2) = a\}$, $G_a \neq \emptyset$, then $\Pi \equiv_\alpha \Pi \cup G'_a$ $(\alpha \in \{s, u\})$ holds where $G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}$,

S-IMP $r, r' \in \Pi$ such that there exists an $A \subseteq body^-(r')$ such that $head(r) \subseteq head(r') \cup A$, $body^-(r) \subseteq body^-(r') \setminus A$ and $body^+(r) \subseteq body^+(r')$ we have $\Pi \equiv_s \Pi \setminus \{r'\}$ and $\Pi \equiv_u \Pi \setminus \{r'\}$

Program Transformations (ctd)

WGPPE $r_1 \in \Pi$, $a \in body^+(r_1)$, $G_a = \{r_2 \in \Pi \mid head(r_2) = a\}$, $G_a \neq \emptyset$, then $\Pi \equiv_\alpha \Pi \cup G'_a$ ($\alpha \in \{s, u\}$) holds where $G'_a = \{head(r_1) \leftarrow (body^+(r_1) \setminus \{a\}) \cup not\ body^-(r_1) \cup body(r_2) \mid r_2 \in G_a\}$,

S-IMP $r, r' \in \Pi$ such that there exists an $A \subseteq body^-(r')$ such that $head(r) \subseteq head(r') \cup A$, $body^-(r) \subseteq body^-(r') \setminus A$ and $body^+(r) \subseteq body^+(r')$ we have $\Pi \equiv_s \Pi \setminus \{r'\}$ and $\Pi \equiv_u \Pi \setminus \{r'\}$

Modularity
Overview

58 Splitting Sets

59 Modules

60 Composition

Splitting Sets

[56]

- A splitting set for a program Π is a set U of atoms such that

  $$\text{if } \mathit{head}(r) \in U, \text{ then } \mathit{atom}(r) \subseteq U \text{ for each } r \in \Pi\ .$$

- A splitting set U divides the program into two parts, namely, the bottom of Π, defined as

  $$b_U(\Pi) = \{r \in \Pi \mid \mathit{atom}(r) \subseteq U\}\ ,$$

  and the top of Π, that is, $\Pi \setminus b_U(\Pi)$.
- If U is a splitting set of Π, then $b_U(\Pi) = \{r \in \Pi \mid \mathit{head}(r) \in U\}$.
- The bottom $b_U(\Pi)$ does not contain any head atoms from the top $\Pi \setminus b_U(\Pi)$, that is, $\mathit{atom}(b_U(\Pi)) \cap \mathit{head}(\Pi \setminus b_U(\Pi)) = \emptyset$.
- A set X of atoms is an answer set of Π iff there is an answer set Y of $b_U(\Pi)$ such that X is an answer set of $\{p \leftarrow \mid p \in Y\} \cup (\Pi \setminus b_U(\Pi))$.

Splitting Sets

[56]

- A splitting set for a program Π is a set U of atoms such that

  $$\text{if } \mathit{head}(r) \in U, \text{ then } \mathit{atom}(r) \subseteq U \text{ for each } r \in \Pi \ .$$

- A splitting set U divides the program into two parts, namely, the bottom of Π, defined as

  $$b_U(\Pi) = \{r \in \Pi \mid \mathit{atom}(r) \subseteq U\} \ ,$$

  and the top of Π, that is, $\Pi \setminus b_U(\Pi)$.
- If U is a splitting set of Π, then $b_U(\Pi) = \{r \in \Pi \mid \mathit{head}(r) \in U\}$.
- The bottom $b_U(\Pi)$ does not contain any head atoms from the top $\Pi \setminus b_U(\Pi)$, that is, $\mathit{atom}(b_U(\Pi)) \cap \mathit{head}(\Pi \setminus b_U(\Pi)) = \emptyset$.
- A set X of atoms is an answer set of Π iff there is an answer set Y of $b_U(\Pi)$ such that X is an answer set of $\{p \leftarrow \mid p \in Y\} \cup (\Pi \setminus b_U(\Pi))$.

Splitting Sets

[56]

- A splitting set for a program Π is a set U of atoms such that

  $$\text{if } \mathit{head}(r) \in U, \text{ then } \mathit{atom}(r) \subseteq U \text{ for each } r \in \Pi \ .$$

- A splitting set U divides the program into two parts, namely, the bottom of Π, defined as

  $$b_U(\Pi) = \{r \in \Pi \mid \mathit{atom}(r) \subseteq U\} \ ,$$

  and the top of Π, that is, $\Pi \setminus b_U(\Pi)$.
- If U is a splitting set of Π, then $b_U(\Pi) = \{r \in \Pi \mid \mathit{head}(r) \in U\}$.
- The bottom $b_U(\Pi)$ does not contain any head atoms from the top $\Pi \setminus b_U(\Pi)$, that is, $\mathit{atom}(b_U(\Pi)) \cap \mathit{head}(\Pi \setminus b_U(\Pi)) = \emptyset$.
- A set X of atoms is an answer set of Π iff there is an answer set Y of $b_U(\Pi)$ such that X is an answer set of $\{p \leftarrow \mid p \in Y\} \cup (\Pi \setminus b_U(\Pi))$.

Splitting Sets

[56]

- A splitting set for a program Π is a set U of atoms such that

  $$\text{if } \mathit{head}(r) \in U, \text{ then } \mathit{atom}(r) \subseteq U \text{ for each } r \in \Pi \ .$$

- A splitting set U divides the program into two parts, namely, the bottom of Π, defined as

  $$b_U(\Pi) = \{r \in \Pi \mid \mathit{atom}(r) \subseteq U\} \ ,$$

  and the top of Π, that is, $\Pi \setminus b_U(\Pi)$.
- If U is a splitting set of Π, then $b_U(\Pi) = \{r \in \Pi \mid \mathit{head}(r) \in U\}$.
- The bottom $b_U(\Pi)$ does not contain any head atoms from the top $\Pi \setminus b_U(\Pi)$, that is, $\mathit{atom}(b_U(\Pi)) \cap \mathit{head}(\Pi \setminus b_U(\Pi)) = \emptyset$.
- A set X of atoms is an answer set of Π iff there is an answer set Y of $b_U(\Pi)$ such that X is an answer set of $\{p \leftarrow \mid p \in Y\} \cup (\Pi \setminus b_U(\Pi))$.

Splitting Set Theorem

- Given two sets U, X of atoms and a program Π, define

$$\Pi^{U,X} = \left\{ r \in \Pi \;\middle|\; \begin{array}{l} (body^+(r) \cap U) \subseteq X, \\ (body^-(r) \cap U) \cap X = \emptyset \end{array} \right\}$$

$$e_U(\Pi, X) = \left\{ r' \;\middle|\; r \in \Pi^{U,X}, \begin{array}{rcl} head(r') & = & head(r), \\ body^+(r') & = & body^+(r) \setminus U, \\ body^-(r') & = & body^-(r) \setminus U \end{array} \right\}$$

  - A solution to Π wrt a splitting set U is a pair $\langle X_0, X_1 \rangle$ such that
    1. X_0 is an answer set of $b_U(\Pi)$, and
    2. X_1 is an answer set of $e_U(\Pi \setminus b_U(\Pi), X_0)$.

Theorem (Splitting Set Theorem)

Given a splitting set U for Π,
a set X of atoms is an answer set of Π iff
$X = X_0 \cup X_1$ for some solution $\langle X_0, X_1 \rangle$ to Π wrt U.

Splitting Set Theorem

- Given two sets U, X of atoms and a program Π, define

$$\Pi^{U,X} = \left\{ r \in \Pi \;\middle|\; \begin{array}{l} (body^+(r) \cap U) \subseteq X, \\ (body^-(r) \cap U) \cap X = \emptyset \end{array} \right\}$$

$$e_U(\Pi, X) = \left\{ r' \;\middle|\; r \in \Pi^{U,X}, \begin{array}{rcl} head(r') & = & head(r), \\ body^+(r') & = & body^+(r) \setminus U, \\ body^-(r') & = & body^-(r) \setminus U \end{array} \right\}$$

- A solution to Π wrt a splitting set U is a pair $\langle X_0, X_1 \rangle$ such that
  1. X_0 is an answer set of $b_U(\Pi)$, and
  2. X_1 is an answer set of $e_U(\Pi \setminus b_U(\Pi), X_0)$.

Theorem (Splitting Set Theorem)

Given a splitting set U for Π,
a set X of atoms is an answer set of Π iff
$X = X_0 \cup X_1$ for some solution $\langle X_0, X_1 \rangle$ to Π wrt U.

Splitting Set Theorem

- Given two sets U, X of atoms and a program Π, define

$$\Pi^{U,X} = \left\{ r \in \Pi \;\middle|\; \begin{array}{l} (body^+(r) \cap U) \subseteq X, \\ (body^-(r) \cap U) \cap X = \emptyset \end{array} \right\}$$

$$e_U(\Pi, X) = \left\{ r' \;\middle|\; r \in \Pi^{U,X}, \begin{array}{rcl} head(r') & = & head(r), \\ body^+(r') & = & body^+(r) \setminus U, \\ body^-(r') & = & body^-(r) \setminus U \end{array} \right\}$$

- A solution to Π wrt a splitting set U is a pair $\langle X_0, X_1 \rangle$ such that
  1. X_0 is an answer set of $b_U(\Pi)$, and
  2. X_1 is an answer set of $e_U(\Pi \setminus b_U(\Pi), X_0)$.

Theorem (Splitting Set Theorem)

Given a splitting set U for Π,
a set X of atoms is an answer set of Π iff
$X = X_0 \cup X_1$ for some solution $\langle X_0, X_1 \rangle$ to Π wrt U.

Splitting Sequences

- A sequence $\langle U_i \rangle_{i \in I}$ of splitting sets for Π such that $U_i \subset U_j$ whenever $i < j$ and $\bigcup_{i \in I} U_i = atom(\Pi)$ is a splitting sequence for Π.

- A solution to Π wrt a splitting sequence $\langle U_i \rangle_{i \in I}$ is a sequence $\langle X_i \rangle_{i \in I}$ of sets of atoms such that
  1. X_0 is an answer set of $b_{U_0}(\Pi)$, and
  2. X_{i+1} is an answer set of $e_{U_i}(b_{U_{i+1}}(\Pi) \setminus b_{U_i}(\Pi), \bigcup_{j \leq i} X_j)$.

- Note that
  1. every atom in $b_{U_0}(\Pi)$ belongs to $U_0 \cap atom(\Pi)$, and
  2. every atom in $e_{U_i}(b_{U_{i+1}}(\Pi) \setminus b_{U_i}(\Pi), \bigcup_{j \leq i} X_j)$ belongs to $(U_{i+1} \setminus U_i) \cap atom(\Pi)$.

Splitting Sequences

- A sequence $\langle U_i \rangle_{i \in I}$ of splitting sets for Π such that $U_i \subset U_j$ whenever $i < j$ and $\bigcup_{i \in I} U_i = atom(\Pi)$ is a splitting sequence for Π.

- A solution to Π wrt a splitting sequence $\langle U_i \rangle_{i \in I}$ is a sequence $\langle X_i \rangle_{i \in I}$ of sets of atoms such that
  1. X_0 is an answer set of $b_{U_0}(\Pi)$, and
  2. X_{i+1} is an answer set of $e_{U_i}(b_{U_{i+1}}(\Pi) \setminus b_{U_i}(\Pi), \bigcup_{j \leq i} X_j)$.

- Note that
  1. every atom in $b_{U_0}(\Pi)$ belongs to $U_0 \cap atom(\Pi)$, and
  2. every atom in $e_{U_i}(b_{U_{i+1}}(\Pi) \setminus b_{U_i}(\Pi), \bigcup_{j \leq i} X_j)$ belongs to $(U_{i+1} \setminus U_i) \cap atom(\Pi)$.

Splitting Sequence Theorem

- We have for a solution $\langle X_i \rangle_{i \in I}$ that

$$\begin{aligned} X_0 &\subseteq U_0 \cap atom(\Pi) \\ X_{i+1} &\subseteq (U_{i+1} \setminus U_i) \cap atom(\Pi) \end{aligned}$$

  and so $X_i \cap X_j = \emptyset$ for all distinct $i, j \in I$.

Theorem (Splitting Sequence Theorem)

Given a splitting sequence $\langle U_i \rangle_{i \in I}$ for Π,
a set X of atoms is an answer set of Π iff
$X = \bigcup_{i \in I} X_i$ for some solution $\langle X_i \rangle_{i \in I}$ to Π wrt $\langle U_i \rangle_{i \in I}$.

Splitting Sequence Theorem

- We have for a solution $\langle X_i \rangle_{i \in I}$ that

$$\begin{aligned} X_0 &\subseteq U_0 \cap atom(\Pi) \\ X_{i+1} &\subseteq (U_{i+1} \setminus U_i) \cap atom(\Pi) \end{aligned}$$

  and so $X_i \cap X_j = \emptyset$ for all distinct $i, j \in I$.

Theorem (Splitting Sequence Theorem)

Given a splitting sequence $\langle U_i \rangle_{i \in I}$ for Π,
a set X of atoms is an answer set of Π iff
$X = \bigcup_{i \in I} X_i$ for some solution $\langle X_i \rangle_{i \in I}$ to Π wrt $\langle U_i \rangle_{i \in I}$.

Modules

- Given a logic program Π and pairwisely distinct sets I, O, and H of atoms, a module $\mathbf{\Pi}$ is a quadruple (Π, I, O, H) such that
  1. $atom(\Pi) \subseteq I \cup O \cup H$ and
  2. $head(\Pi) \subseteq O \cup H$.
- The elements of I, O, and H are called input, output, and hidden atoms, respectively.
- The atoms in $I \cup O$ are to be visible and thus subject to interaction with other modules.

Modules

- Given a logic program Π and pairwisely distinct sets I, O, and H of atoms, a module $\mathbf{\Pi}$ is a quadruple (Π, I, O, H) such that
  1. $atom(\Pi) \subseteq I \cup O \cup H$ and
  2. $head(\Pi) \subseteq O \cup H$.
- The elements of I, O, and H are called input, output, and hidden atoms, respectively.
- The atoms in $I \cup O$ are to be visible and thus subject to interaction with other modules.

Answer Sets of Modules

- An answer set X of a module $\mathbf{\Pi} = (\Pi, I, O, H)$ is defined as the $\subseteq$-minimal model of

$$\Pi^X \cup \{a \leftarrow \mid a \in X \cap I\}$$

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

- Applying the Lin-Zhao Theorem on Page 555,[2] we get:

Lemma

Given a module $\mathbf{\Pi} = (\Pi, I, O, H)$ and an interpretation $X \subseteq I \cup O \cup H$, X is an answer set of $\mathbf{\Pi}$ iff $X \models (Comp(\Pi) \setminus \{A \leftrightarrow \bot \mid A \in I\}) \cup LF(\Pi)$.

[2] For a module $\mathbf{\Pi} = (\Pi, I, O, H)$, we define $atom(\Pi) = I \cup O \cup H$.

Answer Sets of Modules

- An answer set X of a module $\mathbf{\Pi} = (\Pi, I, O, H)$ is defined as the $\subseteq$-minimal model of

$$\Pi^X \cup \{a \leftarrow \mid a \in X \cap I\}$$

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

- Applying the Lin-Zhao Theorem on Page 555,[2] we get:

Lemma

Given a module $\mathbf{\Pi} = (\Pi, I, O, H)$ *and an interpretation* $X \subseteq I \cup O \cup H$, X *is an answer set of* $\mathbf{\Pi}$ *iff* $X \models (Comp(\Pi) \setminus \{A \leftrightarrow \bot \mid A \in I\}) \cup LF(\Pi)$.

[2] For a module $\mathbf{\Pi} = (\Pi, I, O, H)$, we define $atom(\Pi) = I \cup O \cup H$.

(In)dependence

Given a logic program Π, let $SCC^+(\Pi)$ denote the set of all strongly connected components in the positive atom dependency graph of Π, represented by their vertices.

- Two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ are said to be mutually dependent, if there is some $S \in SCC^+(\Pi_1 \cup \Pi_2)$ shared by Π_1 and Π_2, viz., if $S \cap (O_1 \cup H_1) \neq \emptyset$ and $S \cap (O_2 \cup H_2) \neq \emptyset$.

Lemma

Given two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ such that $H_1 \cap (I_2 \cup O_2 \cup H_2) = H_2 \cap (I_1 \cup O_1 \cup H_1) = \emptyset$, $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ are mutually dependent iff $S \cap O_1 \neq \emptyset$ and $S \cap O_2 \neq \emptyset$ for some $S \in SCC^+(\Pi_1 \cup \Pi_2)$.

(In)dependence

Given a logic program Π, let $SCC^+(\Pi)$ denote the set of all strongly connected components in the positive atom dependency graph of Π, represented by their vertices.

- Two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ are said to be mutually dependent, if there is some $S \in SCC^+(\Pi_1 \cup \Pi_2)$ shared by Π_1 and Π_2, viz., if $S \cap (O_1 \cup H_1) \neq \emptyset$ and $S \cap (O_2 \cup H_2) \neq \emptyset$.

Lemma

Given two modules $\mathbf{\Pi}_1$ *and* $\mathbf{\Pi}_2$ *such that*
$H_1 \cap (I_2 \cup O_2 \cup H_2) = H_2 \cap (I_1 \cup O_1 \cup H_1) = \emptyset$,
$\mathbf{\Pi}_1$ *and* $\mathbf{\Pi}_2$ *are mutually dependent iff*
$S \cap O_1 \neq \emptyset$ *and* $S \cap O_2 \neq \emptyset$ *for some* $S \in SCC^+(\Pi_1 \cup \Pi_2)$.

Module Join

[24]

- Define the join of two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$, written $\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2$, as

$$(\Pi_1 \cup \Pi_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2, H_1 \cup H_2)$$

provided that

1. $H_1 \cap (I_2 \cup O_2 \cup H_2) = H_2 \cap (I_1 \cup O_1 \cup H_1) = \emptyset$ and
2. $S \cap O_1 = \emptyset$ or $S \cap O_2 = \emptyset$ for all $S \in SCC^+(\Pi_1 \cup \Pi_2)$.

- The last requirement implies $O_1 \cap O_2 = \emptyset$; unlike this, $I_1 \cap I_2 \neq \emptyset$ is allowed.
- The module join imposes neither restrictions on any negative dependencies nor on positive dependencies inside modules, respectively.

Module Join

[24]

- Define the join of two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$, written $\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2$, as

$$(\Pi_1 \cup \Pi_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2, H_1 \cup H_2)$$

  provided that

  1. $H_1 \cap (I_2 \cup O_2 \cup H_2) = H_2 \cap (I_1 \cup O_1 \cup H_1) = \emptyset$ and
  2. $S \cap O_1 = \emptyset$ or $S \cap O_2 = \emptyset$ for all $S \in SCC^+(\Pi_1 \cup \Pi_2)$.

- The last requirement implies $O_1 \cap O_2 = \emptyset$; unlike this, $I_1 \cap I_2 \neq \emptyset$ is allowed.
- The module join imposes neither restrictions on any negative dependencies nor on positive dependencies inside modules, respectively.

Module Theorem

[70]

- Given two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$, define two interpretations $X_1 \subseteq (I_1 \cup O_1 \cup H_1)$ and $X_2 \subseteq (I_2 \cup O_2 \cup H_2)$ as mutually compatible if $X_1 \cap (I_2 \cup O_2) = X_2 \cap (I_1 \cup O_1)$.
- That is, whenever X_1 and X_2 agree upon the truth values of the visible atoms common to both $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$.

Theorem (Module Theorem)

Given two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ such that $\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2$ is defined, and two mutually compatible interpretations $X_1 \subseteq (I_1 \cup O_1 \cup H_1)$ and $X_2 \subseteq (I_2 \cup O_2 \cup H_2)$.
Then, we have

$$X_1 \cup X_2 \in AS(\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2) \quad \textit{iff} \quad X_1 \in AS(\mathbf{\Pi}_1) \textit{ and } X_2 \in AS(\mathbf{\Pi}_2)\ .$$

Module Theorem

[70]

- Given two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$, define two interpretations $X_1 \subseteq (I_1 \cup O_1 \cup H_1)$ and $X_2 \subseteq (I_2 \cup O_2 \cup H_2)$ as mutually compatible if $X_1 \cap (I_2 \cup O_2) = X_2 \cap (I_1 \cup O_1)$.
- That is, whenever X_1 and X_2 agree upon the truth values of the visible atoms common to both $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$.

Theorem (Module Theorem)

Given two modules $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ such that $\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2$ is defined, and two mutually compatible interpretations $X_1 \subseteq (I_1 \cup O_1 \cup H_1)$ and $X_2 \subseteq (I_2 \cup O_2 \cup H_2)$.
Then, we have

$$X_1 \cup X_2 \in AS(\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2) \quad \textit{iff} \quad X_1 \in AS(\mathbf{\Pi}_1) \textit{ and } X_2 \in AS(\mathbf{\Pi}_2) \ .$$

Example

- Consider $\mathbf{\Pi}_1 = (\Pi_1, I_1, O_1, H_1 = \emptyset)$ and $\mathbf{\Pi}_2 = (\Pi_2, I_2, O_2, H_2 = \emptyset)$, where:

$$\Pi_1 = \left\{ \begin{array}{ll} a \leftarrow b & b \leftarrow a \\ b \leftarrow z & c \leftarrow \mathit{not}\ z \end{array} \right\} \quad \Pi_2 = \left\{ \begin{array}{ll} x \leftarrow y & y \leftarrow x \\ y \leftarrow c & z \leftarrow \mathit{not}\ c \end{array} \right\}$$

$$I_1 = \{z\} \qquad I_2 = \{c\}$$

$$O_1 = \{a, b, c\} \qquad O_2 = \{x, y, z\}$$

- Each module's program contains several strongly connected components: $SCC^+(\Pi_1) = \{\{a, b\}, \{c\}, \{z\}\}$ and $SCC^+(\Pi_2) = \{\{x, y\}, \{z\}, \{c\}\}$.
- Although $\{c\}$ and $\{z\}$ belong to both $SCC^+(\Pi_1)$ and $SCC^+(\Pi_2)$, there is no strongly connected component in $SCC^+(\Pi_1 \cup \Pi_2) = SCC^+(\Pi_1) \cup SCC^+(\Pi_2)$ that contains atoms from both O_1 and O_2.
- This and the absence of hidden atoms establish that the join of $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ is well-defined, and we obtain

$$\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2 = (\Pi_1 \cup \Pi_2, \emptyset, \{a, b, c, x, y, z\}, \emptyset).$$

Example (ctd)

- Module $\mathbf{\Pi}_1$ has answer sets $\{c\}$ and $\{a, b, z\}$;
  module $\mathbf{\Pi}_2$ has answer sets $\{z\}$ and $\{x, y, c\}$.
- Module $\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2$ has answer sets $\{x, y, c\}$ and $\{a, b, z\}$.
- As regards negative dependencies, we observe that neither $\mathbf{\Pi}_1$ nor $\mathbf{\Pi}_2$ comprises any negative cycles.
  This is different in $\mathbf{\Pi}_1 \sqcup \mathbf{\Pi}_2$, where the underlying program induces an even cycle because of $c \leftarrow \mathit{not}\ z$ and $z \leftarrow \mathit{not}\ c$.

Module Theorem (ctd)

- Given that the join operation is associative, the last result generalizes to finite collections $(\mathbf{\Pi}_j)_{j\in J}$ of modules along with pairwisely compatible interpretations $(X_j)_{j\in J}$ such that $X_j \subseteq (I_j \cup O_j \cup H_j)$ for $j \in J$.
- Then, provided that $\bigsqcup_{j\in J} \mathbf{\Pi}_j$ is defined, we have that

$$\bigcup_{j\in J} X_j \in AS(\bigsqcup_{j\in J} \mathbf{\Pi}_j) \quad \text{iff} \quad X_j \in AS(\mathbf{\Pi}_j) \text{ for all } j \in J \ .$$

From splitting sets to modules

- A splitting set can be used as a basis for a module structure:
  Given a splitting set U for Π, define

$$\mathbf{\Pi} = (b_U(\Pi), \emptyset, U, \emptyset) \sqcup (\Pi \setminus b_U(\Pi), U, atom(\Pi) \setminus U, \emptyset) \ .$$

- A splitting sequence can be used as a basis for a module structure:
  Given a splitting sequence $\langle U_i \rangle_{i \in I}$ for Π, define

$$\begin{aligned} \mathbf{\Pi}_0 &= (b_{U_0}(\Pi), \emptyset, U_0, \emptyset) \\ \mathbf{\Pi}_{i+1} &= (b_{U_{i+1}}(\Pi) \setminus b_{U_i}(\Pi), U_i, (U_{i+1} \setminus U_i), \emptyset) \ . \end{aligned}$$

  (Recall that $\bigcup_{j \leq i} U_j \subset U_{i+1}$.)

From splitting sets to modules

- A splitting set can be used as a basis for a module structure:
  Given a splitting set U for Π, define

$$\mathbf{\Pi} = (b_U(\Pi), \emptyset, U, \emptyset) \sqcup (\Pi \setminus b_U(\Pi), U, atom(\Pi) \setminus U, \emptyset) \ .$$

- A splitting sequence can be used as a basis for a module structure:
  Given a splitting sequence $\langle U_i \rangle_{i \in I}$ for Π, define

$$\begin{aligned} \mathbf{\Pi}_0 &= (b_{U_0}(\Pi), \emptyset, U_0, \emptyset) \\ \mathbf{\Pi}_{i+1} &= (b_{U_{i+1}}(\Pi) \setminus b_{U_i}(\Pi), U_i, (U_{i+1} \setminus U_i), \emptyset) \ . \end{aligned}$$

  (Recall that $\bigcup_{j \leq i} U_j \subset U_{i+1}$.)

Incremental Grounding and Solving Overview

61 Motivation

62 Incremental Modularity

63 Incremental ASP Solving

64 Experiments

65 Conclusion

Motivation

Many real-world applications, having exponential state spaces, like

- bio-informatics,
- planning,
- model checking,
- etc.

have associated PSPACE-decision problems.

☞ For instance, the plan existence problem of deterministic planning is PSPACE-complete.

But the problem of whether there is a plan having a length bounded by a given polynomial is in NP.

Motivation

Many real-world applications, having exponential state spaces, like

- bio-informatics,
- planning,
- model checking,
- etc.

have associated PSPACE-decision problems.

☞ For instance, the plan existence problem of deterministic planning is PSPACE-complete.

But the problem of whether there is a plan having a length bounded by a given polynomial is in NP.

Motivation

Many real-world applications, having exponential state spaces, like

- bio-informatics,
- planning,
- model checking,
- etc.

have associated PSPACE-decision problems.

☞ For instance, the plan existence problem of deterministic planning is PSPACE-complete.

But the problem of whether there is a plan having a length bounded by a given polynomial is in NP.

Motivation

State of the Art In ASP such problems are dealt with by iterative deepening search.
That is, considering one problem instance after another by gradually increasing the bound on the solution size.

Problem This approach

- is prone to redundancies in grounding and solving, and
- cannot harness modern look-back techniques regarding conflict-driven learning and heuristics.

Goal Avoiding redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem.

Proposal An incremental approach to both grounding and solving in ASP.

Motivation

State of the Art In ASP such problems are dealt with by iterative deepening search.
That is, considering one problem instance after another by gradually increasing the bound on the solution size.

Problem This approach

- is prone to redundancies in grounding and solving, and
- cannot harness modern look-back techniques regarding conflict-driven learning and heuristics.

Goal Avoiding redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem.

Proposal An incremental approach to both grounding and solving in ASP.

Motivation

State of the Art In ASP such problems are dealt with by iterative deepening search.
That is, considering one problem instance after another by gradually increasing the bound on the solution size.

Problem This approach

- is prone to redundancies in grounding and solving, and
- cannot harness modern look-back techniques regarding conflict-driven learning and heuristics.

Goal Avoiding redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem.

Proposal An incremental approach to both grounding and solving in ASP.

Problem Specification

- A (parameterized) domain description is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers.
  We sometimes denote P and Q by $P[k]$ and $Q[k]$.
- The base program B is meant to describe static knowledge, independent of parameter k.
  The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each value of k.
- One goal is then to decide, for instance, whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

  has an answer set for some (minimum) integer $i \geq 1$.
  We write $R[i]$ rather than $R[k/i]$ whenever clear from the context.

Problem Specification

- A (parameterized) domain description is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers.
  We sometimes denote P and Q by $P[k]$ and $Q[k]$.
- The base program B is meant to describe static knowledge, independent of parameter k.
  The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each value of k.
- One goal is then to decide, for instance, whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

  has an answer set for some (minimum) integer $i \geq 1$.
  We write $R[i]$ rather than $R[k/i]$ whenever clear from the context.

Problem Specification

- A (parameterized) domain description is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers.
  We sometimes denote P and Q by $P[k]$ and $Q[k]$.
- The base program B is meant to describe static knowledge, independent of parameter k.
  The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each value of k.
- One goal is then to decide, for instance, whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has an answer set for some (minimum) integer $i \geq 1$.
We write $R[i]$ rather than $R[k/i]$ whenever clear from the context.

Problem Specification

- A (parameterized) domain description is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers.
  We sometimes denote P and Q by $P[k]$ and $Q[k]$.
- The base program B is meant to describe static knowledge, independent of parameter k.
  The role of P is to capture knowledge accumulating with increasing k, whereas Q is specific for each value of k.
- One goal is then to decide, for instance, whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

  has an answer set for some (minimum) integer $i \geq 1$.
  We write $R[i]$ rather than $R[k/i]$ whenever clear from the context.

Grounding and Solving, classically

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, classically

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, classically

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, classically

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, classically

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$, Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$, Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$,
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$,
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$,
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

Grounding and Solving, incrementally

Input A domain description $R[k] = (B, P[k], Q[k])$.

Output A non-empty set of answer sets of $R[k/i]$, for instance.

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$,
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$,
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$,
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc. until an answer set is obtained.

An Example

$$\left.\begin{array}{l} a \textbf{ causes } p \\ \textbf{exogenous } a \\ \textbf{inertial } p \end{array}\right\} \mapsto \left\{\begin{array}{rcl} B = \left\{\begin{array}{rcl} p(0) & \leftarrow & not\ \neg p(0) \\ \neg p(0) & \leftarrow & not\ p(0) \\ & \leftarrow & p(0), \neg p(0) \end{array}\right\} \\ P[k] = \left\{\begin{array}{rcl} a(k) & \leftarrow & not\ \neg a(k) \\ \neg a(k) & \leftarrow & not\ a(k) \\ p(k) & \leftarrow & a(k) \\ p(k) & \leftarrow & p(k{-}1), not\ \neg p(k) \\ \neg p(k) & \leftarrow & \neg p(k{-}1), not\ p(k) \\ & \leftarrow & p(k), \neg p(k) \\ & \leftarrow & a(k), \neg a(k) \end{array}\right\} \end{array}\right.$$

$$\left.\begin{array}{l} \neg p \textbf{ holds at } 0 \\ p \textbf{ holds at } n \\ \neg a \textbf{ occurs at } n \end{array}\right\} \mapsto \left\{ Q[k] = \left\{\begin{array}{rcl} \leftarrow & not\ \neg p(0) \\ \leftarrow & not\ p(k) \\ \leftarrow & not\ \neg a(k) \end{array}\right\}\right.$$

Module

A module $\mathbb{P}$ is a triple (P, I, O) consisting of

- a (ground) program P over $grd(\mathcal{A})$ and
- sets $I, O \subseteq grd(\mathcal{A})$ such that
  - $I \cap O = \emptyset$,
  - $atom(P) \subseteq I \cup O$, and
  - $head(P) \subseteq O$.

The elements of I and O are called input and output atoms,

- also denoted by $I(\mathbb{P})$ and $O(\mathbb{P})$, respectively;
- similarly, we refer to (ground) program P by $P(\mathbb{P})$.

Module

A module $\mathbb{P}$ is a triple (P, I, O) consisting of

- a (ground) program P over $grd(\mathcal{A})$ and
- sets $I, O \subseteq grd(\mathcal{A})$ such that
  - $I \cap O = \emptyset$,
  - $atom(P) \subseteq I \cup O$, and
  - $head(P) \subseteq O$.

The elements of I and O are called input and output atoms,

- also denoted by $I(\mathbb{P})$ and $O(\mathbb{P})$, respectively;
- similarly, we refer to (ground) program P by $P(\mathbb{P})$.

Recall: Ground Instantiation

- The ground instantiation of a program P is defined as

$$grd(P) = \{r\theta \mid r \in P, \theta : var(r) \to \mathcal{U}\} \text{ , where}$$
$$\mathcal{U} = \{t \in \mathcal{T} \mid var(t) = \emptyset\} .$$

- Analogously, $grd(\mathcal{A}) = \{a \in \mathcal{A} \mid var(a) = \emptyset\}$.

- Note that the set $\mathcal{T}$ of terms includes the natural numbers !

Recall: Ground Instantiation

- The ground instantiation of a program P is defined as

$$grd(P) = \{r\theta \mid r \in P, \theta : var(r) \to \mathcal{U}\} \text{ , where}$$
$$\mathcal{U} = \{t \in \mathcal{T} \mid var(t) = \emptyset\} .$$

- Analogously, $grd(\mathcal{A}) = \{a \in \mathcal{A} \mid var(a) = \emptyset\}$.

- Note that the set $\mathcal{T}$ of terms includes the natural numbers !

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$,

$$P|_X = \{head(r) \leftarrow body^+(r) \cup L \mid r \in P, body^+(r) \subseteq X, L = \{not\ c \mid c \in body^-(r) \cap X\}\}\ .$$

  - ☞ $P|_X$ projects the bodies of rules in P to the atoms of X.

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\, grd(P)|_Y, I, head(grd(P)|_X)\,)\ ,$$

  where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$.

- Let $\mathbb{P}(I) = (P', I, O)$. Then, we have

$$O \subseteq grd(\mathcal{A}) \quad \text{and} \quad atom(P') \subseteq I \cup O\ .$$

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$,

$$P|_X = \{head(r) \leftarrow body^+(r) \cup L \mid r \in P, body^+(r) \subseteq X, L = \{not\ c \mid c \in body^-(r) \cap X\}\}\ .$$

  ☞ $P|_X$ projects the bodies of rules in P to the atoms of X.

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\ grd(P)|_Y, I, head(grd(P)|_X)\ )\ ,$$

  where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$.

- Let $\mathbb{P}(I) = (P', I, O)$. Then, we have

$$O \subseteq grd(\mathcal{A}) \quad \text{and} \quad atom(P') \subseteq I \cup O\ .$$

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$,

$$P|_X = \{head(r) \leftarrow body^+(r) \cup L \mid r \in P, body^+(r) \subseteq X, L = \{not\ c \mid c \in body^-(r) \cap X\}\}\ .$$

  - ☞ $P|_X$ projects the bodies of rules in P to the atoms of X.

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\ grd(P)|_Y, I, head(grd(P)|_X)\ )\ ,$$

  where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$.

- Let $\mathbb{P}(I) = (P', I, O)$. Then, we have

$$O \subseteq grd(\mathcal{A}) \quad \text{and} \quad atom(P') \subseteq I \cup O\ .$$

A Simple Example

Consider

$$P[k] = \{\ p(k) \leftarrow p(Y), \mathit{not}\ p(2) \qquad p(k) \leftarrow p(2)\ \}$$

and note that $grd(P[1])$ is infinite !
For $P[1]$ and $I = \{\ p(0)\ \}$, we get the module

$$(\ grd(P[1])|_{\{p(0),p(1)\}}\ ,\ \{p(0)\}\ ,\ \{p(1)\}\ )$$

where $grd(P[1])|_{\{p(0),p(1)\}} = \{p(1) \leftarrow p(0) \qquad p(1) \leftarrow p(1)\}$.

A Simple Example

Consider

$$P[k] = \{\ p(k) \leftarrow p(Y), \mathit{not}\ p(2) \qquad p(k) \leftarrow p(2)\ \}$$

and note that $grd(P[1])$ is infinite !
For $P[1]$ and $I = \{\ p(0)\ \}$, we get the module

$$(\ grd(P[1])|_{\{p(0),p(1)\}}\ ,\ \{p(0)\}\ ,\ \{p(1)\}\ )$$

where $grd(P[1])|_{\{p(0),p(1)\}} = \{p(1) \leftarrow p(0) \qquad p(1) \leftarrow p(1)\}$.

Modular Domain Description

- Define the join of two modules $\mathbb{P}$ and $\mathbb{Q}$, $\mathbb{P} \sqcup \mathbb{Q}$, as the module

$$(P(\mathbb{P}) \cup P(\mathbb{Q}),\ I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})),\ O(\mathbb{P}) \cup O(\mathbb{Q})),$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$.
  - ☞ Recursion between two modules to be joined is disallowed.
  - ☞ Recursion is allowed within each module.
- A domain description $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$.

Modular Domain Description

- Define the join of two modules $\mathbb{P}$ and $\mathbb{Q}$, $\mathbb{P} \sqcup \mathbb{Q}$, as the module

$$(P(\mathbb{P}) \cup P(\mathbb{Q}),\ I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})),\ O(\mathbb{P}) \cup O(\mathbb{Q})),$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$.
  - ☞ Recursion between two modules to be joined is disallowed.
  - ☞ Recursion is allowed within each module.
- A domain description $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$.

Modular Domain Description

- Define the join of two modules $\mathbb{P}$ and $\mathbb{Q}$, $\mathbb{P} \sqcup \mathbb{Q}$, as the module

$$(P(\mathbb{P}) \cup P(\mathbb{Q}),\ I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})),\ O(\mathbb{P}) \cup O(\mathbb{Q})),$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$.
  - ☞ Recursion between two modules to be joined is disallowed.
  - ☞ Recursion is allowed within each module.
- A domain description $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$.

A Pragmatic Approach

A domain description $(B, P[k], Q[k])$ is modular, if

- atoms defined in B comprise dedicated predicates or 0 as argument,
- atoms defined in $P[k]$ comprise k as argument, and
- atoms defined in $Q[k]$ comprise dedicated predicates and k as argument.

The above conditions can be formalized as follows:

- $atom(grd(B)) \cap \left(\bigcup_{1 \leq i} head(grd(P[i] \cup Q[i]))\right) = \emptyset$,
- $\left(\bigcup_{1 \leq i} atom(grd(P[i]))\right) \cap \left(\bigcup_{1 \leq j} head(grd(Q[j]))\right) = \emptyset$,
- $atom(grd(P[i])) \cap \left(\bigcup_{i < j} head(grd(P[j]))\right) = \emptyset$ for all $1 \leq i$, and
- $atom(grd(Q[i])) \cap \left(\bigcup_{i < j} head(grd(Q[j]))\right) = \emptyset$ for all $1 \leq i$.

A Pragmatic Approach

A domain description $(B, P[k], Q[k])$ is modular, if

- atoms defined in B comprise dedicated predicates or 0 as argument,
- atoms defined in $P[k]$ comprise k as argument, and
- atoms defined in $Q[k]$ comprise dedicated predicates and k as argument.

The above conditions can be formalized as follows:

- $atom(grd(B)) \cap \left(\bigcup_{1 \leq i} head(grd(P[i] \cup Q[i]))\right) = \emptyset$,
- $\left(\bigcup_{1 \leq i} atom(grd(P[i]))\right) \cap \left(\bigcup_{1 \leq j} head(grd(Q[j]))\right) = \emptyset$,
- $atom(grd(P[i])) \cap \left(\bigcup_{i < j} head(grd(P[j]))\right) = \emptyset$ for all $1 \leq i$, and
- $atom(grd(Q[i])) \cap \left(\bigcup_{i < j} head(grd(Q[j]))\right) = \emptyset$ for all $1 \leq i$.

Incremental ASP Solving (made very easy)

See [26] for formal details!

Grounding For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, an incremental grounder is a partial function

$$\texttt{ground} : (P, I) \mapsto (P', O) \ ,$$

where P' is a program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$.

Solving For programs R, R' over $grd(\mathcal{A})$ and a set L of literals over $grd(\mathcal{A})$, an incremental solver is a pair of total functions

$$\texttt{add} : R \mapsto R' \quad \text{and} \quad \texttt{solve} : L \mapsto \chi,$$

where χ is a subset of the power set of $grd(\mathcal{A})$.

Incremental ASP Solving (made very easy)

See [26] for formal details!

Grounding For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, an incremental grounder is a partial function

$$\texttt{ground} : (P, I) \mapsto (P', O) \ ,$$

where P' is a program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$.

Solving For programs R, R' over $grd(\mathcal{A})$ and a set L of literals over $grd(\mathcal{A})$, an incremental solver is a pair of total functions

$$\texttt{add} : R \mapsto R' \quad \text{and} \quad \texttt{solve} : L \mapsto \chi,$$

where χ is a subset of the power set of $grd(\mathcal{A})$.

Incremental ASP Solving (made very easy)

See [26] for formal details!

Grounding For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, an incremental grounder is a partial function

$$\texttt{ground} : (P, I) \mapsto (P', O) \ ,$$

where P' is a program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$.

Solving For programs R, R' over $grd(\mathcal{A})$ and a set L of literals over $grd(\mathcal{A})$, an incremental solver is a pair of total functions

$$\texttt{add} : R \mapsto R' \quad \text{and} \quad \texttt{solve} : L \mapsto \chi,$$

where χ is a subset of the power set of $grd(\mathcal{A})$.

Algorithm 4: `isolve`

Input : A domain description $(B, P[k], Q[k])$.
Output : A nonempty set of answer sets.
Internal : A grounder GROUNDER.
Internal : A solver SOLVER.

```
1  i ← 0
2  (P_0, O) ← GROUNDER.ground(B, ∅)
3  SOLVER.add(P_0)
4  loop
5      i ← i + 1
6      (P_i, O_i) ← GROUNDER.ground(P[i], O)
7      SOLVER.add(P_i)
8      O ← O ∪ O_i
9      (Q_i, O'_i) ← GROUNDER.ground(Q[i], O)
10     SOLVER.add(Q_i(α_i) ∪ {{α_i} ←} ∪ {← α_{i−1}})
11     χ ← SOLVER.solve({α_i})
12     if χ ≠ ∅ then return {X \ {α_i} | X ∈ χ}
```

Example Reloaded

$$\left.\begin{array}{l} a \textbf{ causes } p \\ \textbf{exogenous } a \\ \textbf{inertial } p \end{array}\right\} \mapsto \left\{ B = \left\{ \begin{array}{rcl} p(0) & \leftarrow & not\ \neg p(0) \\ \neg p(0) & \leftarrow & not\ p(0) \\ & \leftarrow & p(0), \neg p(0) \end{array}\right\}, P[k] = \left\{ \begin{array}{rcl} a(k) & \leftarrow & not\ \neg a(k) \\ \neg a(k) & \leftarrow & not\ a(k) \\ p(k) & \leftarrow & a(k) \\ p(k) & \leftarrow & p(k{-}1), not\ \neg p(k) \\ \neg p(k) & \leftarrow & \neg p(k{-}1), not\ p(k) \\ & \leftarrow & p(k), \neg p(k) \\ & \leftarrow & a(k), \neg a(k) \end{array}\right\} \right.$$

$$\left.\begin{array}{l} \neg p \textbf{ holds at } 0 \\ p \textbf{ holds at } n \\ \neg a \textbf{ occurs at } n \end{array}\right\} \mapsto \left\{ Q[k] = \left\{ \begin{array}{rcl} \leftarrow & not\ \neg p(0) \\ \leftarrow & not\ p(k) \\ \leftarrow & not\ \neg a(k) \end{array}\right\} \right.$$

Example Reloaded

i	Rules				L
0	B	$p(0)$	$\leftarrow$	$not\ \neg p(0)$	
		$\neg p(0)$	$\leftarrow$	$not\ p(0)$	
			$\leftarrow$	$p(0), \neg p(0)$	
1	$P[1]$	$a(1)$	$\leftarrow$	$not\ \neg a(1)$	
		$\neg a(1)$	$\leftarrow$	$not\ a(1)$	
		$p(1)$	$\leftarrow$	$a(1)$	
		$p(1)$	$\leftarrow$	$p(0), not\ \neg p(1)$	
		$\neg p(1)$	$\leftarrow$	$\neg p(0), not\ p(1)$	
			$\leftarrow$	$p(1), \neg p(1)$	
			$\leftarrow$	$a(1), \neg a(1)$	
	$Q[1](\alpha_1)$		$\leftarrow$	$not\ \neg p(0), \alpha_1$	α_1
			$\leftarrow$	$not\ p(1), \alpha_1$	
			$\leftarrow$	$not\ \neg a(1), \alpha_1$	
		$\{\alpha_1\}$	$\leftarrow$		
			$\leftarrow$	α_0	

Example Reloaded

i	Rules				L
0	B	$p(0)$	$\leftarrow$	$not\ \neg p(0)$	
		$\neg p(0)$	$\leftarrow$	$not\ p(0)$	
			$\leftarrow$	$p(0), \neg p(0)$	
1			$\vdots$		
2	$P[2]$	$a(2)$	$\leftarrow$	$not\ \neg a(2)$	
		$\neg a(2)$	$\leftarrow$	$not\ a(2)$	
		$p(2)$	$\leftarrow$	$a(2)$	
		$p(2)$	$\leftarrow$	$p(1), not\ \neg p(2)$	
		$\neg p(2)$	$\leftarrow$	$\neg p(1), not\ p(2)$	
			$\leftarrow$	$p(2), \neg p(2)$	
			$\leftarrow$	$a(2), \neg a(2)$	
	$Q[2](\alpha_2)$		$\leftarrow$	$not\ \neg p(0), \alpha_2$	α_2
			$\leftarrow$	$not\ p(2), \alpha_2$	
			$\leftarrow$	$not\ \neg a(2), \alpha_2$	
		$\{\alpha_2\}$	$\leftarrow$		
			$\leftarrow$	α_1	

Example with iclingo

incremental.lp

```
#base.
p(0)  :- not -p(0).
-p(0) :- not p(0).
:- p(0), -p(0).

#cumulative k.
a(k)  :- not -a(k).
-a(k) :- not a(k).
p(k)  :- a(k).
p(k)  :- p(k-1), not -p(k).
-p(k) :- -p(k-1), not p(k).
:- p(k), -p(k).
:- a(k), -a(k).

#volatile k.
:- not -p(0).
:- not p(k).
:- not -a(k).
```

```
$ iclingo -V[erbose] incremental.lp

iclingo version 2.0.2 (clasp 1.1.1)
Reading from incremental.lp...
=============== step 1 ===============
Grounding...
Preprocessing...
Solving...
=============== step 2 ===============
Grounding...
Preprocessing...
Solving...
Answer: 1
-p(0) a(1) p(1) -a(2) p(2)
=============== Summary ==============

Models       : 1
Total Steps  : 2
Time         : 0.000
```

Experiments

We consider iclingo in four settings, keeping over successive solving steps

1. learned constraints,
2. learned constraints and heuristic values,
3. heuristic values only, and
4. neither.

We compare these variants with iterative deepening search using

- clingo, the direct combination of gringo and clasp via an internal interface, as well as
- gringo and clasp via a textual interface (using the output language of lparse).

Experiments

Name	*n*	*iclingo (1)*	*iclingo (2)*	*iclingo (3)*	*iclingo (4)*	*clingo*	*gringo|clasp*
Blocks-world	20	2.61	2.61	2.62	2.62	37.09	42.41
	25	6.78	6.84	6.80	6.80	124.35	138.68
	30	15.68	15.80	15.71	15.81	330.15	362.39
	35	32.43	32.36	32.29	32.31	753.90	821.96
	40	60.99	60.75	60.71	61.04	-	-
	Σ	118.49	118.36	118.13	118.58	2445.49	2565.44
Queens	80	19.46	65.83	39.98	47.79	144.28	153.61
	90	36.72	135.19	70.81	81.70	249.13	264.21
	100	49.25	227.69	111.99	128.62	409.69	431.23
	110	64.05	424.03	176.16	201.67	636.91	669.75
	120	99.54	612.76	274.29	354.00	958.34	1003.67
	Σ	269.02	1465.50	673.23	813.78	2398.35	2522.47

Experiments

Name	*n*	*iclingo (1)*	*iclingo (2)*	*iclingo (3)*	*iclingo (4)*	*clingo*	*gringo|clasp*
Sokoban	16	243.22	287.46	320.07	334.08	376.74	384.41
	12	26.50	37.55	50.61	28.19	27.83	28.43
	16	124.26	124.44	320.97	341.94	189.48	194.12
	16	135.72	164.70	128.66	183.74	120.60	123.57
	18	140.80	145.07	233.71	275.12	236.60	242.19
	16	26.86	40.60	29.41	27.88	45.94	47.04
	17	1165.67	906.00	734.44	730.09	887.26	904.75
	14	119.95	140.11	106.40	213.22	96.26	98.10
	14	35.42	42.74	58.79	46.81	70.16	71.81
	21	286.46	200.43	600.19	777.68	278.97	285.09
	17	120.33	140.44	139.19	156.85	171.01	174.90
	14	39.09	36.21	36.00	47.48	66.12	67.43
	Σ	2464.28	2265.75	2758.44	3163.08	2566.97	2621.84

Experiments

Name	*n*	*iclingo (1)*	*iclingo (2)*	*iclingo (3)*	*iclingo (4)*	*clingo*	*gringo|clasp*
Sokoban	16	-	-	-	-	-	-
back	12	51.23	44.62	98.09	57.42	72.59	74.30
	16	264.81	201.48	265.21	359.38	296.45	302.46
	16	148.19	121.19	150.06	145.40	148.25	151.43
	18	723.07	-	-	-	1059.02	1081.34
	16	243.81	185.00	340.97	190.32	402.27	410.72
	17	599.74	714.40	1051.60	825.61	-	-
	14	149.37	126.04	164.98	191.33	170.36	173.74
	14	29.73	69.46	73.03	28.04	43.06	43.89
	21	346.56	428.43	400.81	295.69	402.78	411.70
	17	181.00	143.20	172.83	317.82	234.21	239.56
	14	15.06	58.45	39.27	17.50	59.63	60.78
	Σ	3952.57	4492.27	5156.85	4828.51	5288.62	5349.92

Experiments

Name	*n*	*iclingo (1)*	*iclingo (2)*	*iclingo (3)*	*iclingo (4)*	*clingo*	*gringo*|*clasp*
Towers	33	38.00	42.96	48.46	27.15	31.98	32.76
	34	61.40	36.78	47.09	45.95	61.77	63.39
	36	81.26	60.77	88.52	131.29	86.56	88.46
	39	223.46	155.76	184.63	204.13	216.89	222.74
	41	429.82	327.74	392.47	342.11	459.97	471.22
	Σ	833.94	624.01	761.17	750.63	857.17	878.57
Towers	33	4.62	6.42	5.68	5.80	12.59	12.79
back	34	55.79	33.42	56.27	42.39	52.80	54.00
	36	16.66	16.46	14.69	17.11	24.81	25.38
	39	27.88	25.43	28.60	32.83	46.01	46.85
	41	48.20	36.38	62.75	40.62	83.78	85.60
	Σ	153.15	118.11	167.99	138.75	219.99	224.62
	$\Sigma\Sigma$	7791.45	9084.00	9635.81	9813.33	13776.59	14162.86

Conclusion

- Tackling bounded problems in ASP, paving the way for more ambitious real-world applications.
- Module theory provides us with
  - a natural semantics for non-ground, parameterized program slices and
  - makes precise their composition by appeal to input/output interfaces.
- First experimental results indicate the computational impact of our incremental approach, *but* more needs to be done !

☞ Incremental problems differ from traditional ones !

Conclusion

- Tackling bounded problems in ASP, paving the way for more ambitious real-world applications.
- Module theory provides us with
  - a natural semantics for non-ground, parameterized program slices and
  - makes precise their composition by appeal to input/output interfaces.
- First experimental results indicate the computational impact of our incremental approach, *but* more needs to be done !
- ☞ Incremental problems differ from traditional ones !

Constraint Answer Set Programming
Overview

66 Motivation

67 Preliminaries

68 Modeling Language

69 Algorithms

70 Experiments

71 Summary

Motivation

Observation

Certain applications are more naturally modeled by mixing Boolean with non-Boolean constructs, eg., accounting for

- resources,
- fine timings, or
- functions over finite domains.

Introduction

Groundbreaking Work in ASP [7, 63, 64]

- semantics for multi-sorted, first-order language
- algorithms using DPLL-style backtracking

SAT Modulo Theories [68]

- no modelling language
- algorithms using CDCL-style backjumping and learning

Our ASP approach [34]

- propositional semantics
- algorithms using CDCL-style backjumping and learning
- use off-the-shelf CP solvers

SAT Modulo Theories

SAT Modulo Theories (SMT)

- logical formulas with respect to combinations of background theories
- real numbers, integers, lists, arrays, bit vectors ...

SAT Modulo Theories

SAT Modulo Theories (SMT)

- logical formulas with respect to combinations of background theories
- real numbers, integers, lists, arrays, bit vectors ...

SAT

$$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

SAT Modulo Theories

SAT Modulo Theories (SMT)

- logical formulas with respect to combinations of background theories
- real numbers, integers, lists, arrays, bit vectors ...

SMT

$$(\sin(x)^3 = \cos(\log(y) \cdot x) \quad \vee b \quad \vee -x^2 \geq 2.3y)$$

SAT Modulo Theories

SAT Modulo Theories (SMT)

- logical formulas with respect to combinations of background theories
- real numbers, integers, lists, arrays, bit vectors ...

SMT

$$(\quad a \quad \vee b \quad \vee -x^2 \geq 2.3y)$$

SAT Modulo Theories

SAT Modulo Theories (SMT)

- logical formulas with respect to combinations of background theories
- real numbers, integers, lists, arrays, bit vectors . . .

SMT

$$(\quad a \quad \vee b \quad \vee \quad c \quad)$$

Outline

66 Motivation

67 Preliminaries

68 Modeling Language

69 Algorithms

70 Experiments

71 Summary

Constraint Satisfaction Problem

Definition

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.
  - ☞ For $S = (v_1, \ldots, v_k)$, we have $R \subseteq dom(v_1) \times \cdots \times dom(v_k)$.

Constraint Satisfaction Problem

Definition

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.
  - ☞ For $S = (v_1, \ldots, v_k)$, we have $R \subseteq dom(v_1) \times \cdots \times dom(v_k)$.

Example

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

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

$V = \{s, e, n, d, m, o, r, y\}$ $\qquad dom(v) = 0..9$ for all $v \in V$

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

Example

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

Each letter corresponds exactly to one digit and all variables have to be pairwisely distinct.
The example has exactly one solution.

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

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

Constraint Satisfaction Problem

Notation

We use $S(c) = S$ and $R(c) = R$ to access the scope and the relation of a constraint $c = (S, R)$.

Definition

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 Satisfaction Problem

Notation

We use $S(c) = S$ and $R(c) = R$ to access the scope and the relation of a constraint $c = (S, R)$.

Definition

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

Definition

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 $r \in P$.

Auxiliary Definition

Given a set of literals B and some set $\mathcal{B}$ of atoms, we define $B|_{\mathcal{B}} = (B^+ \cap \mathcal{B}) \cup \{not\ a \mid a \in B^- \cap \mathcal{B}\}$.

Constraint Answer Set Programming

Definition

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 $r \in P$.

Auxiliary Definition

Given a set of literals B and some set $\mathcal{B}$ of atoms, we define $B|_{\mathcal{B}} = (B^+ \cap \mathcal{B}) \cup \{not\ a \mid a \in B^- \cap \mathcal{B}\}$.

Constraint Answer Set Programming

Definition

We identify constraint atoms with constraints via a function

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

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

Definition

We identify constraint atoms with constraints via a function

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

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

Definition

We identify constraint atoms with constraints via a function

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

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

Definition

Let P be a constraint logic program over $\mathcal{A} \cup \mathcal{C}$ and
let $A : V[P] \to D[P]$ be an assignment.
We define the *constraint reduct* of as P wrt A as follows.

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

Definition

A set $X \subseteq \mathcal{A}$ of (regular) atoms is a *constraint answer set* of P wrt A, if X is an answer set of P^A.

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

Constraint Answer Set Programming

Definition

Let P be a constraint logic program over $\mathcal{A} \cup \mathcal{C}$ and
let $A : V[P] \to D[P]$ be an assignment.
We define the *constraint reduct* of as P wrt A as follows.

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

Definition

A set $X \subseteq \mathcal{A}$ of (regular) atoms is a *constraint answer set* of P wrt A, if X is an answer set of P^A.

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

Constraint Answer Set Programming

Definition

Let P be a constraint logic program over $\mathcal{A} \cup \mathcal{C}$ and
let $A : V[P] \to D[P]$ be an assignment.
We define the *constraint reduct* of as P wrt A as follows.

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

Definition

A set $X \subseteq \mathcal{A}$ of (regular) atoms is a *constraint answer set* of P wrt A, if X is an answer set of P^A.

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

Outline

66 Motivation

67 Preliminaries

68 Modeling Language

69 Algorithms

70 Experiments

71 Summary

Modeling Language

Note

Although our semantics is propositional, the atoms in $\mathcal{A}$ and $\mathcal{C}$ are constructible from a multi-sorted, first-order signature given by:

- a set $\mathcal{P}_\mathcal{A} \cup \mathcal{P}_\mathcal{C}$ of *predicate symbols* such that $\mathcal{P}_\mathcal{A} \cap \mathcal{P}_\mathcal{C} = \emptyset$,
- a set $\mathcal{F}_\mathcal{A} \cup \mathcal{F}_\mathcal{C}$ of *function symbols* (including constant symbols),
- a set $\mathcal{V}_\mathcal{A}$ of *regular variable symbols*, and
- a set $\mathcal{V}_\mathcal{C} \subseteq \mathcal{T}(\mathcal{F}_\mathcal{A})$ of *constraint variable symbols*, where $\mathcal{T}(\mathcal{F}_\mathcal{A})$ denotes the set of all ground terms over $\mathcal{F}_\mathcal{A}$.

As common in ASP, the atoms in $\mathcal{A} \cup \mathcal{C}$ are obtained by a grounding process.

An Example

$$
\begin{aligned}
& time(0..t_{max}) \\
& bucket(a) \quad bucket(b) \\
1\ \{pour(B,T) : bucket(B)\}\ 1 \quad & \leftarrow \quad time(T), T < t_{max} \\
1 \leq^{\$} amt(B,T) \quad & \leftarrow \quad pour(B,T), T < t_{max} \\
amt(B,T) \leq^{\$} 3 \quad & \leftarrow \quad pour(B,T), T < t_{max} \\
amt(B,T) =^{\$} 0 \quad & \leftarrow \quad not\ pour(B,T), T < t_{max} \\
vol(B,T+1) =^{\$} vol(B,T) + amt(B,T) \quad & \leftarrow \quad time(T) < t_{max} \\
down(B,T) \quad & \leftarrow \quad vol(C,T) <^{\$} vol(B,T) \\
up(B,T) \quad & \leftarrow \quad not\ down(B,T) \\
vol(a,0) =^{\$} 0 \quad vol(b,0) =^{\$} 1 & \\
& \leftarrow \quad up(a,t_{max})
\end{aligned}
$$

An Example

- Consider the signature of our exemplary program:

$$\begin{aligned} \{B, C, T\} &\subseteq \mathcal{V}_{\mathcal{A}} \\ \{0, \ldots, t_{max}, +, a, b, amt, vol\} &\subseteq \mathcal{F}_{\mathcal{A}} \\ \{<, time, bucket, pour, up, down\} &\subseteq \mathcal{P}_{\mathcal{A}} \\ \{0, 1, 3, +\} &\subseteq \mathcal{F}_{\mathcal{C}} \\ \{=^{\$}, <^{\$}, \leq^{\$}\} &\subseteq \mathcal{P}_{\mathcal{C}} \end{aligned}$$

- With substitution $\{B \mapsto b, T \mapsto 1, t_{max} \mapsto 2\}$, we get:

$$amt(b,1) =^{\$} 0 \leftarrow not\ pour(b,1)$$

$$vol(b,2) =^{\$} vol(b,1) + amt(b,1) \leftarrow$$

- and, among others, our signature hence contains

$$\begin{aligned} \{amt(b,1), vol(b,1), vol(b,2)\} &\subseteq \mathcal{V}_{\mathcal{C}} \\ \{pour(b,1)\} &\subseteq \mathcal{A} \\ \{amt(b,1) =^{\$} 0,\ vol(b,2) =^{\$} vol(b,1) + amt(b,1)\} &\subseteq \mathcal{C} \end{aligned}$$

An Example

- Consider the signature of our exemplary program:

$$\begin{aligned} \{B, C, T\} &\subseteq \mathcal{V}_{\mathcal{A}} \\ \{0, \ldots, t_{max}, +, a, b, amt, vol\} &\subseteq \mathcal{F}_{\mathcal{A}} \\ \{<, time, bucket, pour, up, down\} &\subseteq \mathcal{P}_{\mathcal{A}} \\ \{0, 1, 3, +\} &\subseteq \mathcal{F}_{\mathcal{C}} \\ \{=^{\$}, <^{\$}, \leq^{\$}\} &\subseteq \mathcal{P}_{\mathcal{C}} \end{aligned}$$

- With substitution $\{B \mapsto b, T \mapsto 1, t_{max} \mapsto 2\}$, we get:

$$\begin{aligned} amt(b,1) =^{\$} 0 &\leftarrow not\ pour(b,1) \\ vol(b,2) =^{\$} vol(b,1) + amt(b,1) &\leftarrow \end{aligned}$$

- and, among others, our signature hence contains

$$\begin{aligned} \{amt(b,1), vol(b,1), vol(b,2)\} &\subseteq \mathcal{V}_{\mathcal{C}} \\ \{pour(b,1)\} &\subseteq \mathcal{A} \\ \{amt(b,1) =^{\$} 0,\ vol(b,2) =^{\$} vol(b,1) + amt(b,1)\} &\subseteq \mathcal{C} \end{aligned}$$

An Example

- Consider the signature of our exemplary program:

$$\begin{array}{rcl} \{B, C, T\} & \subseteq & \mathcal{V}_{\mathcal{A}} \\ \{0, \ldots, t_{max}, +, a, b, amt, vol\} & \subseteq & \mathcal{F}_{\mathcal{A}} \\ \{<, time, bucket, pour, up, down\} & \subseteq & \mathcal{P}_{\mathcal{A}} \\ \{0, 1, 3, +\} & \subseteq & \mathcal{F}_{\mathcal{C}} \\ \{=^{\$}, <^{\$}, \leq^{\$}\} & \subseteq & \mathcal{P}_{\mathcal{C}} \end{array}$$

- With substitution $\{B \mapsto b, T \mapsto 1, t_{max} \mapsto 2\}$, we get:

$$\begin{array}{rcl} amt(b,1) =^{\$} 0 & \leftarrow & not\ pour(b,1) \\ vol(b,2) =^{\$} vol(b,1) + amt(b,1) & \leftarrow & \end{array}$$

- and, among others, our signature hence contains

$$\begin{array}{rcl} \{amt(b,1), vol(b,1), vol(b,2)\} & \subseteq & \mathcal{V}_{\mathcal{C}} \\ \{pour(b,1)\} & \subseteq & \mathcal{A} \\ \{amt(b,1) =^{\$} 0,\ vol(b,2) =^{\$} vol(b,1) + amt(b,1)\} & \subseteq & \mathcal{C} \end{array}$$

An Example

- Consider the signature of our exemplary program:

$$\begin{aligned} \{B, C, T\} &\subseteq \mathcal{V}_{\mathcal{A}} \\ \{0, \ldots, t_{max}, +, a, b, amt, vol\} &\subseteq \mathcal{F}_{\mathcal{A}} \\ \{<, time, bucket, pour, up, down\} &\subseteq \mathcal{P}_{\mathcal{A}} \\ \{0, 1, 3, +\} &\subseteq \mathcal{F}_{\mathcal{C}} \\ \{=^{\$}, <^{\$}, \leq^{\$}\} &\subseteq \mathcal{P}_{\mathcal{C}} \end{aligned}$$

- With substitution $\{B \mapsto b, T \mapsto 1, t_{max} \mapsto 2\}$, we get:

$$\begin{aligned} amt(b,1) =^{\$} 0 &\leftarrow not\ pour(b,1) \\ vol(b,2) =^{\$} vol(b,1) + amt(b,1) &\leftarrow \end{aligned}$$

- and, among others, our signature hence contains

$$\begin{aligned} \{amt(b,1), vol(b,1), vol(b,2)\} &\subseteq \mathcal{V}_{\mathcal{C}} \\ \{pour(b,1)\} &\subseteq \mathcal{A} \\ \{amt(b,1) =^{\$} 0,\ vol(b,2) =^{\$} vol(b,1) + amt(b,1)\} &\subseteq \mathcal{C} \end{aligned}$$

An Example

- Consider the signature of our exemplary program:

$$\begin{aligned} \{B, C, T\} &\subseteq \mathcal{V}_{\mathcal{A}} \\ \{0, \ldots, t_{max}, +, a, b, amt, vol\} &\subseteq \mathcal{F}_{\mathcal{A}} \\ \{<, time, bucket, pour, up, down\} &\subseteq \mathcal{P}_{\mathcal{A}} \\ \{0, 1, 3, +\} &\subseteq \mathcal{F}_{\mathcal{C}} \\ \{=^{\$}, <^{\$}, \leq^{\$}\} &\subseteq \mathcal{P}_{\mathcal{C}} \end{aligned}$$

- With substitution $\{B \mapsto b, T \mapsto 1, t_{max} \mapsto 2\}$, we get:

$$\begin{aligned} amt(b,1) =^{\$} 0 &\leftarrow not\ pour(b,1) \\ vol(b,2) =^{\$} vol(b,1) + amt(b,1) &\leftarrow \end{aligned}$$

- and, among others, our signature hence contains

$$\begin{aligned} \{amt(b,1), vol(b,1), vol(b,2)\} &\subseteq \mathcal{V}_{\mathcal{C}} \\ \{pour(b,1)\} &\subseteq \mathcal{A} \\ \{amt(b,1) =^{\$} 0,\ vol(b,2) =^{\$} vol(b,1) + amt(b,1)\} &\subseteq \mathcal{C} \end{aligned}$$

An Example

For $t_{max} = 2$, our program has eleven constraint answer sets, summarized as follows

$up(a,0)$	$pour(a,0)$	$amt(a,0)$	$up(a,1)$	$pour(a,1)$	$amt(a,1)$	$up(a,2)$
T	**T**	1	**T**	**T**	1, 2, 3	**F**
T	**T**	2, 3	**F**	**T**	1, 2, 3	**F**
T	**T**	3	**F**	**F**	0	**F**
T	**F**	0	**T**	**T**	3	**F**

An Example

For $t_{max} = 2$, our program has eleven constraint answer sets, summarized as follows

$up(a,0)$	$pour(a,0)$	$amt(a,0)$	$up(a,1)$	$pour(a,1)$	$amt(a,1)$	$up(a,2)$
T	**T**	1	**T**	**T**	$1,2,3$	**F**
T	**T**	$2,3$	**F**	**T**	$1,2,3$	**F**
T	**T**	3	**F**	**F**	0	**F**
T	**F**	0	**T**	**T**	3	**F**

Outline

66 Motivation

67 Preliminaries

68 Modeling Language

69 Algorithms

70 Experiments

71 Summary

CDNL

Main Algorithm

loop

- Propagate
- **if** no conflict **then**
  - **if** partial assignment **then** Decide
  - **else return** solution
- **else if** some decisions made **then**
  - Analyze Conflict
  - Record Reason
  - Backjump
- **else exit**

CDNL

Main Algorithm

loop

- Propagate
- **if** no conflict **then**
  - **if** partial assignment **then** Decide
  - **else return** solution
- **else if** some decisions made **then**
  - Analyze Conflict
  - Record Reason
  - Backjump
- **else exit**

ASP Propagation

Propagation Algorithm

loop

  Unit-Propagation

  if conflict **then return**

  else if Unfounded-Set **then**

    Record Loop-Nogood

    if conflict **then return**

  else return

Constraint ASP Propagation

Propagation Algorithm

loop

UNIT-PROPAGATION

if conflict **then return**

else if UNFOUNDED-SET **then**

RECORD LOOP-NOGOOD

if conflict **then return**

else if CONSTRAINT-PROPAGATION **then**

if conflict **then return**

else return

CDNL

Main Algorithm

loop

- Propagate
- **if** no conflict **then**
  - **if** partial assignment **then** Decide
  - **else return** solution
- **else if** some decisions made **then**
  - Analyze Conflict
  - Record Reason
  - Backjump
- **else exit**

Constraint CDNL

Main Algorithm

loop

- Propagate
- **if** no conflict **then**
  - **if** partial assignment **then** Decide
  - **else if** CSP-Solve **then return** solution
  - **else**
    - Analyze Conflict
    - Record Reason
    - Backjump
- **else if** some decisions made **then**
  - Analyze Conflict
  - Record Reason
  - Backjump
- **else exit**

Outline

66 Motivation

67 Preliminaries

68 Modeling Language

69 Algorithms

70 Experiments

71 Summary

Example on Grounding

```
John goes to work either by car (30-40 minutes),
or by bus (at least 60 minutes).
Fred goes to work either by car (20-30 minutes),
or in a car pool (40-50 minutes).
Today John left home between 7:10 and 7:20,
and Fred arrived between 8:00 and 8:10.
We also know that John arrived at work about 10-20
minutes after Fred left home.
```

We wish to answer queries such as:

- Is the information in the story consistent?
- Is it possible that John took the bus, and Fred used the carpool?
- What are the possible times at which Fred left home?

Example on Grounding

- Number of Lines Output

maxtime	gringo	clingcon
10	626	19
100	47132	19
500	1137332	19
1000	4525082	19

NASA Advisor: 40 seconds realtime

	clingo	*adsolver*				*clingcon*				
Benchmark	5	5	7	11	13	5	7	11	13	20
3-0/025	162.84	14.74	51.42	460.57	365.37	1.19	1.97	4.21	5.99	17.84
3-0/050	173.28	31.39	108.21	471.41	—	1.26	2.32	6.80	11.85	27.36
3-0/100	175.94	448.90	188.33	—	—	1.32	2.35	10.11	12.04	38.78
3-0/125	165.64	19.78	60.07	224.60	—	1.18	1.94	4.05	10.00	133.99
5-0/025	174.12	28.78	107.41	—	—	1.28	2.90	5.87	14.27	66.55
5-0/050	163.25	13.57	42.00	204.34	497.64	1.18	1.97	4.71	10.04	241.59
5-0/100	168.16	21.50	66.10	282.36	514.08	1.20	1.98	4.13	6.45	25.32
5-0/125	174.38	32.02	104.32	429.72	—	1.34	2.95	6.39	9.70	81.17
8-0/025	177.82	41.57	140.93	—	—	1.30	2.73	11.00	12.69	222.49
8-0/050	167.72	18.83	54.76	215.43	—	1.18	1.93	4.02	7.76	457.86
8-0/100	165.55	13.72	41.03	208.74	—	1.21	2.00	5.05	6.10	26.17
8-0/125	162.29	16.81	53.40	246.64	519.59	1.20	1.99	4.15	6.69	17.82
∅	169.25	58.47	84.83	378.65	558.06	1.24	2.25	5.87	9.47	113.08

clingo standard ASP grounder and ASP solver
adsolver ASPmCSP solver based on smodels [Mellarkod and Gelfond, '08]
clingcon ASPmCSP solver based on clingo and gecode

Reason Calculation

Design Goal

Use off-the-shelf CP solvers

Problem

Off-the-shelf CP solvers do not provide reasons for conflicts

Idea

Approximate reasons in various ways

- lazy learning versus early learning
- use constraint interdependencies or not

Reason Calculation

Design Goal

Use off-the-shelf CP solvers

Problem

Off-the-shelf CP solvers do not provide reasons for conflicts

Idea

Approximate reasons in various ways

- lazy learning versus early learning
- use constraint interdependencies or not

NASA Advisor: 40 seconds realtime

clingcon	11				13				20			
Benchmark	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)
3-0/025	12.83	4.90	4.21	4.21	36.71	8.09	5.99	5.90	—	64.89	17.84	18.78
3-0/050	49.21	13.25	6.80	8.42	219.01	35.38	11.85	12.41	—	370.71	27.36	30.46
3-0/100	36.44	9.54	10.11	5.30	34.82	53.96	12.04	7.16	—	—	38.78	33.67
3-0/125	6.31	4.25	4.05	4.07	163.47	21.09	10.00	8.52	—	377.06	133.99	31.46
5-0/025	69.36	15.38	5.87	10.52	176.81	23.39	14.27	15.06	—	—	66.55	118.65
5-0/050	24.40	6.58	4.71	4.67	311.11	22.93	10.04	7.19	—	542.27	241.59	—
5-0/100	6.39	4.30	4.13	4.11	72.95	6.36	6.45	5.70	—	83.85	25.32	49.03
5-0/125	61.72	17.66	6.39	6.96	111.92	41.84	9.70	10.87	—	—	81.17	73.68
8-0/025	76.73	7.69	11.00	9.25	248.29	37.33	12.69	7.81	—	—	222.49	—
8-0/050	6.33	4.19	4.02	4.05	189.04	15.75	7.76	8.05	—	52.14	457.86	32.29
8-0/100	7.71	4.49	5.05	4.16	220.12	11.71	6.10	5.94	—	159.37	26.17	23.72
8-0/125	5.11	4.31	4.15	4.14	38.91	15.31	6.69	7.06	—	69.64	17.82	18.80
∅	30.21	8.05	5.87	5.82	151.93	24.43	9.47	8.47	—	343.33	113.08	135.88

(a) lazy learning with constraint interdependencies
(b) early learning with constraint interdependencies;
(c) lazy learning without constraint interdependencies
(d) early learning without constraint interdependencies.

Outline

66 Motivation

67 Preliminaries

68 Modeling Language

69 Algorithms

70 Experiments

71 Summary

Summary

- Our *semantic approach* relies on a propositional language rather than a multi-sorted, first-order language.
- Our *algorithmic approach* integrates CP capacities into modern ASP solvers based on conflict-driven learning and backjumping.
  - We follow the lazy approach of advanced SMT solvers by abstracting from the constraints in a specialized theory.
- We implemented our approach to constraint answer set solving within the new solver *clingcon*
  - extending ASP system *clingo*
  - with the generic CP solver *gecode*

☞ `http://potassco.sourceforge.net`


C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub.
The `nomore++` approach to answer set solving.
In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.


C. Anger, K. Konczak, T. Linke, and T. Schaub.
A glimpse of answer set programming.
Künstliche Intelligenz, 19(1):12–17, 2005.


Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft; available at `http://www.cs.utexas.edu/users/tag/cmodels.html`, 2003.


C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.

Cambridge University Press, 2003.


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.


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


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.


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.


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


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.


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


M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.

Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.


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.


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


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


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.


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.


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.


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.


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.


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.


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 One*, pages 615–664. College Publications, 2005.


M. Fitting.
A kripke-kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.


H Gaifman and E. Shapiro.
Fully abstract compositional semantics for logic programs.
In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Programming Languages (POPL'89)*, pages 134–142, 1989.


M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.
Available at `http://potassco.sourceforge.net`.


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.


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 [44], pages 250–264.


M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [5], pages 260–265.


M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [5], pages 136–148.


M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.

Conflict-driven answer set solving.
In Veloso [78], pages 386–392.


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.


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.


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.


M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [44], pages 235–249.


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.


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.


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


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


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.


M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In *Proceedings of the International Conference on Logic Programming*, pages 579–597, 1990.


M. Gelfond and V. Lifschitz.
Classical negation in logic programs and disjunctive databases.
New Generation Computing, 9:365–385, 1991.


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


S. Grell.
Investigation and analysis of new approaches for representing and reasoning about biological networks using action languages.
Diploma thesis, University of Potsdam and Max Planck Institute of Molecular Plant Physiology, 2006.


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.


J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [78], pages 2318–2323.


H. Kautz and B. Selman.
Planning as satisfiability.
In B. Neumann, editor, *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363. John Wiley & sons, 1992.


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.


R. Kowalski.
Logic for data description.

In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 77–103. Plenum Press, 1978.


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.


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.


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


V. Lifschitz.
Introduction to answer set programming.

Unpublished draft; available at
http://www.cs.utexas.edu/users/vl/papers/esslli.ps, 2004.


V. Lifschitz, D. Pearce, and A. Valverde.
Strongly equivalent logic programs.
ACM Transactions on Computational Logic, 2(4):526–541, 2001.


V. Lifschitz and A. Razborov.
Why are there so many loop formulas?
ACM Transactions on Computational Logic, 7(2):261–268, 2006.


V. Lifschitz, L. Tang, and H. Turner.
Nested expressions in logic programs.
Annals of Mathematics and Artificial Intelligence, 25(3-4):369–389, 1999.


V. Lifschitz and H. Turner.
Splitting a logic program.
In *Proceedings of the Eleventh International Conference on Logic Programming*, pages 23–37. MIT Press, 1994.


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


J. Lloyd.
Foundations of Logic Programming.
Symbolic Computation. Springer-Verlag, 2nd edition, 1987.


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


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


J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.

In Biere et al. [10], chapter 4, pages 131–153.


J. Marques-Silva and K. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.


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.


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.


D. Mitchell.
A SAT solver primer.

Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.


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.


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.


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.


M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry.
An A-prolog decision support system for the space shuttle.

In I. Ramakrishnan, editor, *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 2001.



E. Oikarinen and T. Janhunen.
Modular equivalence for normal logic programs.
In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, pages 412–416. IOS Press, 2006.



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.



L. Ryan.
Efficient algorithms for clause-learning SAT solvers.

Master's thesis, Simon Fraser University, 2004.


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


T. Syrjänen.
Lparse 1.0 user's manual.
http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.


T. Syrjänen.
A rule-based formal method for software configuration.
Research Report A55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, 1999.


H. Turner.
Strong equivalence made easy: nested expressions and weight constraints.
Theory and Practice of Logic Programming, 3(4-5):609–622, 2003.


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.



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



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, 2001.