Outline

1. Answer Set Programming
   - Introduction
   - Normal Logic Programs
   - Modeling
   - Disjunctive Logic Programs
   - Nested Logic Programs
   - Propositional Theories
   - Computational Complexity
2. Extensions
   - Strong Negation
   - Choice Rules
   - Cardinality Constraints
   - Cardinality Rules
   - Weight Constraints (and more)
   - Aggregates
3. Bibliography

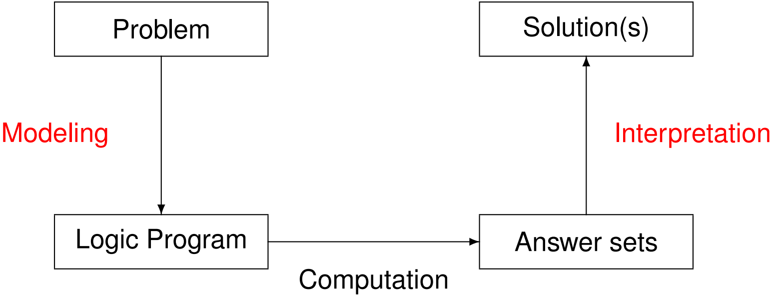
Outline

1 Answer Set Programming
- Introduction
- Normal Logic Programs
- Modeling
- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity

2 Extensions
- Strong Negation
- Choice Rules
- Cardinality Constraints
- Cardinality Rules
- Weight Constraints (and more)
- Aggregates

3 Bibliography

Modeling and Interpreting



Problems as Logic Programs

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

3-colorability of graphs

Problem

Problem instance *A graph* (V, E).

Problem class *Assign each vertex in* V *one of* 3 *colors such that no two vertexes in* V *connected by an edge in* E *have the same color.*

Solution

C(I)	*vertex(1)*	$\leftarrow$	*vertex(2)*	$\leftarrow$	*vertex(3)*	$\leftarrow$
	edge(1,2)	$\leftarrow$	*edge(2,3)*	$\leftarrow$	*edge(3,1)*	$\leftarrow$
C(P)	*colored(V,r)*	$\leftarrow$	*not colored(V,b), not colored(V,g),vertex(V)*			
	colored(V,b)	$\leftarrow$	*not colored(V,r), not colored(V,g),vertex(V)*			
	colored(V,g)	$\leftarrow$	*not colored(V,r), not colored(V,b),vertex(V)*			
		$\leftarrow$	*edge(V,U), colored(V,C), colored(U,C), color(C)*			
AS's	{ *colored(1,r), colored(2,b), colored(3,g), othercolor(1,g), ..., vertex(1),..., edge(1,2), ...* }, ...					

n-colorability of graphs (with $n = 3$)

Problem

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.*

Solution

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

Basic Methodology

ASP Basic Methodology

Generate and Test (or: Guess and Check) approach.

- Generator: Generate potential candidate answer sets (typically through non-deterministic constructs)
- Tester: Eliminate non-valid Candidates (typically through integrity constraints)

In a Nutshell...

$$\textit{Logic Program} = \textit{Data} + \textit{Generator} + \textit{Tester}\ \ [+\textit{Optimizer}]$$

Satisfiability

Problem

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.*

Solution

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

Generator	***Tester***	***Answer set***
$a \leftarrow not\ a'$	$\leftarrow not\ a, b$	$A_1 = \{a,b\}$
$a' \leftarrow not\ a$	$\leftarrow a, not\ b$	$A_2 = \{a',b'\}$
$b \leftarrow not\ b'$		
$b' \leftarrow not\ b$		

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

Hamiltonian Path

Problem

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

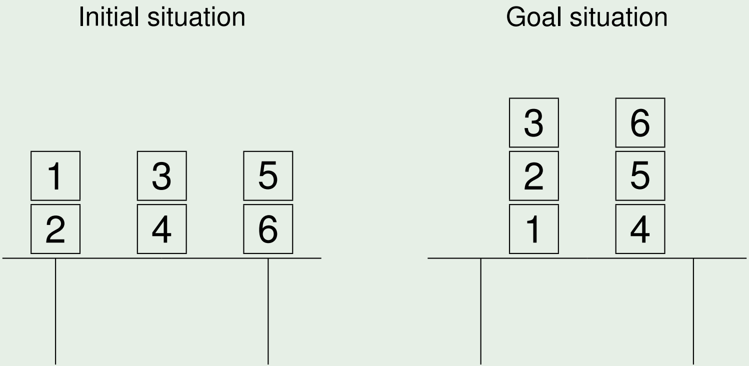
Problem class — *Find a path in* (V, E) *starting at* v *and visiting all other vertices in* V *exactly once.*

Solution

C(I)	vertex/1	arc/2	start/1
C(P)	inPath(X,Y)	$\leftarrow$	arc(X,Y), not outPath(X,Y)
	outPath(X,Y)	$\leftarrow$	arc(X,Y), not inPath(X,Y)
		$\leftarrow$	inPath(X,Y), inPath(X,Z), $Y \neq Z$
		$\leftarrow$	inPath(X,Y), inPath(Z,Y), $X \neq Z$
	reached(X)	$\leftarrow$	start(X)
	reached(X)	$\leftarrow$	reached(Y),inPath(Y,X)
		$\leftarrow$	vertex(X),not reached(X)
		$\leftarrow$	inPath(Y,X), start(X)

Planning in the Blocksworld

Example (Scenario)



Planning in the Blocksworld

Example (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).
```

Planning in the Blocksworld

Example (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

Example (Generate)

```
time(0..lasttime).

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

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

#show move/3.
```

Planning in the Blocksworld

Example (Define)

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

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

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

Planning in the Blocksworld

Example (Test)

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

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

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

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

Planning in the Blocksworld

Example (The Plan)

```
clingo blocks.lp 0
clingo version 5.4.0
Reading from blocks.lp
Solving...
Answer: 1
move(1,table,0) move(3,table,0) move(2,1,1) move(5,4,1) move(3,2,2)
move(6,5,2)
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.008s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.008s
```

Outline

1. Answer Set Programming
   - Introduction
   - Normal Logic Programs
   - Modeling
   - Disjunctive Logic Programs
   - Nested Logic Programs
   - Propositional Theories
   - Computational Complexity
2. Extensions
   - Strong Negation
   - Choice Rules
   - Cardinality Constraints
   - Cardinality Rules
   - Weight Constraints (and more)
   - Aggregates
3. Bibliography

Disjunctive Logic Programs: Syntax

Definition (Disjunctive Rule)

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.

Definition (Disjunctive Logic Program)

A disjunctive logic program is a finite set of disjunctive rules.

Notation

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

Disjunctive Logic Programs: Semantics

Definition (Positive Disjunctive Logic Programs)

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

Definition (Closure)

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).

Definition ($\min_\subseteq(\Pi)$)

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 (seen as a formula).

Disjunctive Logic Programs: Semantics

Definition (Reduct of a Disjunctive Logic Program)

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

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

Definition (Answer Set of a Disjunctive Logic Program)

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

Positive Disjunctive Logic Programs: Example

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 of graphs revisited

Problem

Problem instance A graph (V, E).

Problem class Assign each vertex in V one of 3 colors such that no two vertexes in V connected by an edge in E have the same color.

Solution

C(I)	vertex(1) ← vertex(2) ← vertex(3) ← edge(1,2) ← edge(2,3) ← edge(3,1) ←
C(P)	colored(V,r); colored(V,b); colored(V,g) ← vertex(V) ← edge(V,U), colored(V,C), colored(U,C)
AS's	{ colored(1,r), colored(2,b), colored(3,g), ... }, ...

Disjunctive Logic Programs: Examples

Example

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

Some properties

Property

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

Property

If X is a stable model of a disjunctive logic program Π, then X is a model of Π (seen as a formula)

Property

If X and Y are stable models of a disjunctive logic program Π, then $X \not\subset Y$

Property

If $A \in X$ for some stable model X of a disjunctive logic program Π, then there is a rule $r \in \Pi$ such that $body^+(r) \subseteq X$, $body^-(r) \cap X = \emptyset$, and $head(r) \cap X = \{A\}$

Disjunctive Logic Programs: Example with variables

Example

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

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

For every answer set X of Π, we have

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

Disjunctive Logic Programs: Example with variables

Example

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1) \\ b(1)\,;c(2) & \leftarrow & a(1,2) \\ b(2)\,;c(1) & \leftarrow & a(2,1) \\ b(2)\,;c(2) & \leftarrow & a(2,2) \end{array} \right\}$$

- 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 $X \in \min_{\subseteq}(ground(\Pi)^X)$.

Disjunctive Logic Programs: Example with variables

Example

$$ground(\Pi)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1) \\ & & \\ b(2)\,;c(1) & \leftarrow & a(2,1) \\ & & \end{array} \right\}$$

- 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 $X \notin \min_{\subseteq}(ground(\Pi)^X)$.

Outline

1. Answer Set Programming
   - Introduction
   - Normal Logic Programs
   - Modeling
   - Disjunctive Logic Programs
   - Nested Logic Programs
   - Propositional Theories
   - Computational Complexity
2. Extensions
   - Strong Negation
   - Choice Rules
   - Cardinality Constraints
   - Cardinality Rules
   - Weight Constraints (and more)
   - Aggregates
3. Bibliography

Nested Logic Programs: Syntax

Definition (Formulas)

Formulas are formed from propositional atoms, $\top$ and $\bot$, using negation-as-failure (*not*), conjunction (,), and disjunction (;).

Definition (Nested Rules)

A nested rule, r, is an ordered pair of the form

$$F \leftarrow G$$

where F and G are formulas.

Definition (Nested Logic Program)

A nested program is a finite set of rules.

Notation

$head(r) = F$ and $body(r) = G$.

Nested Logic Programs: Semantics

Definition (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 $X \models \mathit{body}(r)$.

Nested Logic Programs: Semantics

Definition ($\min_{\subseteq}(\Pi)$)

The set of all $\subseteq$-minimal sets of atoms satisfying program Π is denoted by $\min_{\subseteq}(\Pi)$.

Definition (Reduct of a Formula)

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 $\bot$,
- $(F, G)^X = (F^X, G^X)$,
- $(F; G)^X = (F^X; G^X)$,
- $(not\ F)^X = \begin{cases} \bot & \text{if } X \models F \\ \top & \text{otherwise} \end{cases}$

Nested Logic Programs: Semantics

Definition (Reduct of a Nested Logic Program)

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

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

Definition (Answer Set of a Nested Logic Program)

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

Nested Logic Programs: Examples

Example

- $\Pi_1 = \{(p \,;\, not\ p) \leftarrow \top\}$
  - 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 (Intuitionistic Logics HT (Heyting, 1930) and G3 (Gödel, 1932))
  - $F \leftarrow G,\ not\ not\ H$ 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 $not\ F$

Hamiltonian Paths: Generator Revisited

Example

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

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

Outline

1. Answer Set Programming
   - Introduction
   - Normal Logic Programs
   - Modeling
   - Disjunctive Logic Programs
   - Nested Logic Programs
   - Propositional Theories
   - Computational Complexity
2. Extensions
   - Strong Negation
   - Choice Rules
   - Cardinality Constraints
   - Cardinality Rules
   - Weight Constraints (and more)
   - Aggregates
3. Bibliography

Propositional Theories: Syntax

Definition (Formulas)

Formulas are formed from atoms and $\bot$ using conjunction ($\wedge$), disjunction ($\vee$), and implication ($\rightarrow$).

Notation

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

Definition (Propositional Theory)

A propositional theory is a finite set of formulas.

Propositional Theories: Semantics

Definition (Satisfaction relation)

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.

Definition (Reduct of a formula)

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.

Propositional Theories: Semantics

Definition (Reduct of a Propositional Theory)

The reduct, $\mathcal{F}^X$, of a propositional theory $\mathcal{F}$ relative to a set X of atoms is defined as

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

Definition (Satisfaction of a Propositional Theory)

A set X of atoms satisfies a propositional theory $\mathcal{F}$, written $X \models \mathcal{F}$, iff $X \models F$ for each $F \in \mathcal{F}$.

Propositional Theories: Semantics

Definition ($\min_{\subseteq}(\mathcal{F})$)

The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory $\mathcal{F}$ is denoted by $\min_{\subseteq}(\mathcal{F})$.

Definition (Answer Set of a Propositional Theory)

A set X of atoms is an answer set of a propositional theory $\mathcal{F}$ if $X \in \min_{\subseteq}(\mathcal{F}^X)$.

Proposition

If X is an answer set of $\mathcal{F}$, then $X \models \mathcal{F}$.

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

Propositional Theories: Two examples

Example

- $\mathcal{F}_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - 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 $\min_{\subseteq}(\mathcal{F}_2^{\{q,r\}}) = \{\{q, r\}\}$. ✔

Propositional Theories: Relationship with Logic Programs

Definition (Translation of a nested rule)

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,
- $\tau[not\ F] = \sim\tau[F]$,
- $\tau[(F, G)] = (\tau[F] \wedge \tau[G])$,
- $\tau[(F; G)] = (\tau[F] \vee \tau[G])$.

Definition (Translation of a nested logic program)

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

Propositional Theories: Relationship with Logic Programs

Theorem (Embedding of nested logic programs)

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 $\tau[\Pi]$.

Example

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

Outline

1. Answer Set Programming
   - Introduction
   - Normal Logic Programs
   - Modeling
   - Disjunctive Logic Programs
   - Nested Logic Programs
   - Propositional Theories
   - Computational Complexity
2. Extensions
   - Strong Negation
   - Choice Rules
   - Cardinality Constraints
   - Cardinality Rules
   - Weight Constraints (and more)
   - Aggregates
3. Bibliography

Computational Complexity

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.

Computational Complexity

- 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.