

Answer Set Programming

João Leite

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, Portugal
`jleite@fct.unl.pt`

November 2018

- 1 Answer Set Programming
 - Introduction
 - Normal Logic Programs
 - Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity
- 2 Bibliography

- 1 Answer Set Programming
 - Introduction
 - Normal Logic Programs
 - Modeling

- Disjunctive Logic Programs
 - Nested Logic Programs
 - Propositional Theories
 - Computational Complexity
- 2 Bibliography

1 Answer Set Programming

- Introduction
- Normal Logic Programs
- Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity

2 Bibliography

Some Historical Remarks

- In the 1950's, John McCarthy expressed the need to use logic-based languages for representing and reasoning about knowledge
- First attempts used classical logic of the predicate calculus (First Order Logic)
 - well defined semantics
 - well understood inference mechanism
 - expressive power capable of representing mathematical knowledge
- However, common-sense reasoning is inherently non-monotonic, leading to the development of non-monotonic logics (late 1970's and 1980's)
 - circumscription
 - default logic
 - non-monotonic modal logics

Some Historical Remarks

- Also in the 1970's, others were investigating the idea of combining logic as a representation language with the theory of automated deduction.
- Kowalski and Colmerauer et al. defined and implemented the first PROLOG interpreter, based on a model theoretic, fixpoint and operational semantics for the Horn-clause fragment.
- The beginning of the paradigm of **Logic Programming**
- Formal foundations of LP during late 1970's:
 - least model semantics (van Emden and Kowalski)
 - first PROLOG compiler (Warren)
 - program completion (Clark)
 - closed world assumption (Reiter) - leading to negation-as-finite-failure in PROLOG

Some Historical Remarks

Logic Programming introduced Declarative Programming in Computer Science.

- Procedural Language: specify how
- Declarative Language: specify what

Algorithm = Logic + Control (Kowalski, 1979)

Features of Prolog (Colmerauer, Kowalski)

- 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

Some Historical Remarks

- Prolog is only **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

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

Some Historical Remarks

- Prolog offers **negation as failure** via operator **not**.
- But, for instance,

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

cannot be captured by

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

- but by appeal to **Clark's completion** by

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

The idea of completion

- In LP one uses “if” but means “iff” [Clark78]

```
naturalN(0).  
naturalN(s(N)) :- naturalN(N).
```

- This doesn't imply that -1 is not a natural number!
- With this program we mean:

$$naturalN(x) \Leftrightarrow \forall x (x = 0 \vee \exists y (x = s(y) \wedge naturalN(y)))$$

- This is the idea of Clark's completion:
 - Syntactically transform if's into iff's
 - Use classical logic in the transformed theory to provide the semantics of the program

Completion Semantics

Definition (Program Completion)

The completion of P is the theory $comp(P)$ obtained by:

- Replace $p(\vec{t}) \leftarrow \varphi$ by $p(\vec{x}) \leftarrow \vec{x} = \vec{t}, \varphi$
- Replace $p(\vec{x}) \leftarrow \varphi$ by $p(\vec{x}) \leftarrow \exists \vec{y} \varphi$, where \vec{y} are the original variables of the rule
- Merge all rules with the same head into a single one $p(\vec{x}) \leftarrow \varphi_1 \vee \dots \vee \varphi_n$
- For every $q(\vec{x})$ without rules, add $q(\vec{x}) \leftarrow \perp$
- Replace $p(\vec{x}) \leftarrow \varphi$ by $\forall \vec{x} (p(\vec{x}) \Leftrightarrow \varphi)$

Definition (Completion Semantics)

The completion semantics of P is given by the semantics of $comp(P)$ where \neg is interpreted as classical negation.

- Though completion's definition is not that simple, the idea behind it is quite simple
- Also, it defines a non-classical semantics by means of classical inference on a transformed theory

- By adopting completion, procedurally we have:
 - not is “negation as finite failure”

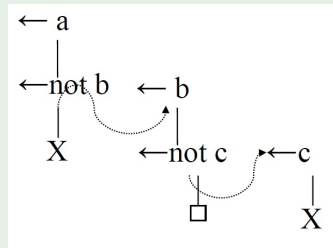
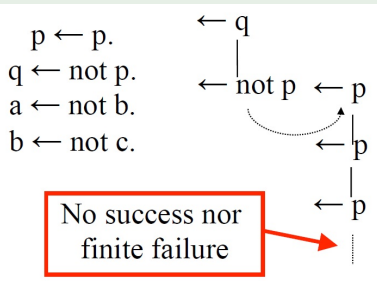
Definition (SLDNF Proof Procedure)

In SLDNF proceed as in SLD. To prove not A :

- If there is a finite derivation for A , fail not A
- If, after any finite number of steps, all derivations for A fail, remove not A from the resolvent (i.e. succeed not A)
- SLDNF can be efficiently implemented (cf. Prolog)

SLDNF example

Example



According to completion:

- $\text{comp}(P) \models \{\text{not } a, b, \text{not } c\}$
- $\text{comp}(P) \not\models p, \text{comp}(P) \not\models \text{not } p$
- $\text{comp}(P) \not\models q, \text{comp}(P) \not\models \text{not } q$

Problems with completion

- Some consistent programs may become inconsistent:

$$p \leftarrow \text{not } p \text{ *becomes* } p \Leftrightarrow \text{not } p$$

- Does not correctly deal with deductive closures

```
edge(a,b).    edge(c,d).    edge(d,c).  
reachable(a).  
reachable(A) ← edge(A,B), reachable(B).
```

- Completion doesn't conclude *not reachable(c)*, due to the circularity caused by *edge(c, d)* and *edge(d, c)*
- Circularity is a procedural concept, not a declarative one

Some Historical Remarks

- Clark's completion has other problems:
- For example:

```
bird(tweety).  
fly(X) :- bird(X), not abnormal(X).  
abnormal(X) :- irregular(X).  
irregular(X) :- abnormal(X).
```

...does not allow the conclusion that tweety flies.

- Or even more complex yet analogous situations.
- An explanation would be: "the rules for abnormal and irregular cause a loop".
 - But looping is a procedural concept, not a declarative one, and should be rejected when defining declarative semantics

Some Historical Remarks

- While the Logic Programming community was developing PROLOG into a full fledged Programming Language...
- Some devoted their time to the development of appropriate semantics for logic programs with negation.
- The 1980's and early 1990's saw "the war of the semantics", mainly focusing on the meaning of programs like:

`a :- not b.`

`a :- not a.`

`b :- not a.`

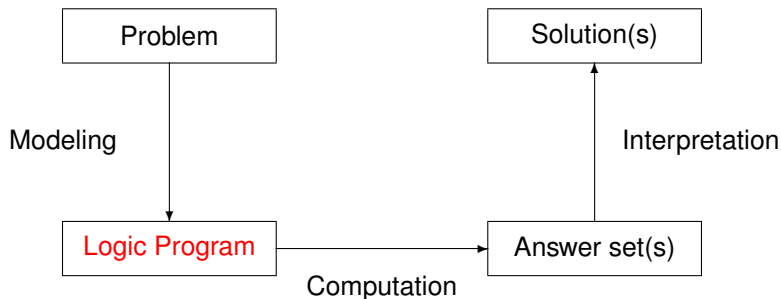
- Great Schism: **Single model vs multiple model semantics**

- Due to its declarative nature, LP has become a prime candidate for Knowledge Representation and Reasoning
- This has been more noticeable since its relations to other NMR formalisms were established
- For this usage of LP, a precise declarative semantics was in order.
- To date:
 - Well-Founded Semantics by van Gelder et al. (1991).
 - Stable Model Semantics by Gelfond & Lifschitz (1988,1991).

- 1 Answer Set Programming
 - Introduction
 - Normal Logic Programs
 - Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity
- 2 Bibliography

Problem solving in ASP: Syntax



Normal Logic Programs: Syntax

Definition (Rule)

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

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

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

Definition (Logic Program)

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

Notation

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

Normal Logic Programs: Syntax

Definition (Positive Logic Program)

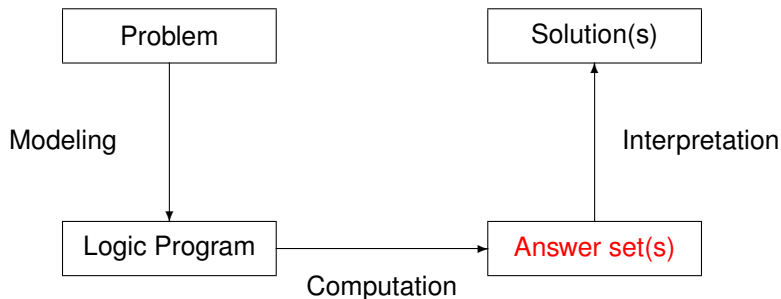
A program is called positive if $body^-(r) = \emptyset$ for all its rules.

Notation

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

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

Problem solving in ASP: Semantics



Positive Logic Programs: Semantics

Definition (Closure)

A set of atoms X is **closed under** a positive program Π iff for any $r \in \Pi$, $\text{head}(r) \in X$ whenever $\text{body}^+(r) \subseteq X$.

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

Definition ($Cn(\Pi)$)

The **least** (smallest) set of atoms which is closed under a positive program Π is denoted by $Cn(\Pi)$.

- $Cn(\Pi)$ corresponds to the \subseteq -least model of Π (seen as a formula).

Definition (Answer Set of a Positive Logic Program)

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:

$$A_0 \vee \neg A_1 \vee \dots \vee \neg A_m$$

- A set of definite clauses has a (unique) smallest model.
- **Horn clauses** are clauses with **at most** one positive atom.
 - Every definite clause is a Horn clause but not vice versa.
 - A set of Horn clauses has a smallest model or none.
- This smallest model is the intended semantics of a set of Horn clauses.
 - 👉 Given a positive program Π , $Cn(\Pi)$ corresponds to the smallest model of the set of definite clauses corresponding to Π .

Another “logical” remark

Answer sets versus (minimal) models

- Program $\{a \leftarrow \text{not } b\}$ has answer set $\{a\}$.
 - Clause $a \vee b$ (being equivalent to $a \leftarrow \neg b$)
 - has models $\{a\}$, $\{b\}$, and $\{a, b\}$,
 - among which $\{a\}$ and $\{b\}$ are minimal.
-  The negation-as-failure operator *not* makes a difference!

Normal Logic Programs: Semantics

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))

Example

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

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

This formula has one answer set:

$\{p, q\}$

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

$$\Pi \quad \begin{array}{l} q \leftarrow \\ p \leftarrow q, \text{ not } r \end{array}$$

Answer set: Basic idea

For instance, interpreting

$$\left\{ \begin{array}{l} b \leftarrow \\ a \leftarrow b, \text{ not } c \end{array} \right\} \quad \text{as} \quad b \wedge (b \wedge \neg c \rightarrow a), \text{ that is, } b \wedge (a \vee c),$$

we obtain

- 3 models: $\{a, b\}$, $\{b, c\}$, and $\{a, b, c\}$,
- 2 minimal models: $\{a, b\}$ and $\{b, c\}$, and
- 1 stable model: $\{a, b\}$ ✗ \iff answer set

Informally, a set of atoms X is an answer set of a logic program Π

- if X is a minimal (classical) model of Π ¹ and
- if all atoms in X are *justified* by some rule in Π .

¹That is, interpreting ' \leftarrow ', ' $,$ ', and '*not*' as in classical logic.

Normal Logic Programs: Semantics

Definition (GL-Reduct (Gelfond and Lifschitz 1988))

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

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

Intuitively, given a set of atoms X from Π , Π^X is obtained from Π by:

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

Definition (Answer Set of a Normal Logic Program)

A set X of atoms is an **answer set** of a program Π iff $\text{Cn}(\Pi^X) = X$.

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

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

Normal Logic Programs: Examples

Example (First Example)

Consider the program

$$\Pi = \{p \leftarrow p. \quad q \leftarrow \text{not } p.\}$$

| 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 | ✗ |

Normal Logic Programs: Examples

Example (Even Loop)

Consider the program

$$\Pi = \{p \leftarrow \text{not } q. \quad q \leftarrow \text{not } p.\}$$

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

Normal Logic Programs: Examples

Example (Odd Loop)

Consider the program

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

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

Answer Sets: Properties

Property

If X is an answer set of a logic program Π , then X is a model of Π (seen as a formula).

Property (Minimality)

Every answer set X of Π is a minimal model of Π (wrt. \subseteq).

Property (Supportedness)

If X is an answer sets of a logic program Π , and $p \in X$, then $\exists r \in \Pi$ such that $\text{head}(r) = p$ and $\text{body}^-(r) \cap X = \emptyset$ and $\text{body}^+(r) \subseteq X$.

Answer Sets: Alternative Definition

Definition (Modified-Reduct (Faber et al. 2004))

The **modified reduct**, Π_X , of a program Π relative to a set of atoms X is given by

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

Intuitively, given a set of atoms X from Π , Π_X (dubbed the set of generating rules of X wrt. Π) is obtained from Π by:

- 1 deleting each rule having a body literal that is false w.r.t. X .

Definition (Answer Set of a Normal Logic Program - Alternative)

A set X of atoms is an **answer set** of a program Π iff $X \in \min_{\subseteq}(\Pi_X)$, where $\min_{\subseteq}(\Pi)$ is the set of minimal models of a program Π (wrt. \subseteq).

Theorem (Soundness and completeness of the Alternative Definition)

$$X \in \min_{\subseteq}(\Pi_X) \quad \text{iff} \quad \text{Cn}(\Pi^X) = X$$

Example: Even Loop Revisited

Example (Even Loop)

Consider the program

$$\Pi = \{p \leftarrow \text{not } q. \quad q \leftarrow \text{not } p.\}$$

| X | Π_X | $\min_{\subseteq}(\Pi_X)$ |
|-------------|--|---------------------------|
| \emptyset | $p \leftarrow \text{not } q$ $q \leftarrow \text{not } p$ | $\{p\}, \{q\}$ ✗ |
| $\{p\}$ | $p \leftarrow \text{not } q$ | $\{p\}, \{q\}$ ✓ |
| $\{q\}$ | $q \leftarrow \text{not } p$ | $\{p\}, \{q\}$ ✓ |
| $\{p, q\}$ | | \emptyset ✗ |

A closer look at C_n

Definition (Immediate Consequence Operator)

Let Π be a positive program and X a set of atoms. The **immediate consequence operator** T_Π is defined as follows:

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

Let $T_\Pi^0(X) = X$ and $T_\Pi^i(X) = T_\Pi(T_\Pi^{i-1}(X))$.

Further let $T_\Pi \uparrow^\omega = \bigcup_{i=0}^\infty T_\Pi^i(\emptyset)$.

Theorem

Let Π be a positive program. Then:

- $C_n(\Pi) = T_\Pi \uparrow^\omega$.
- $X \subseteq Y$ implies $T_\Pi(X) \subseteq T_\Pi(Y)$.
- $C_n(\Pi)$ is the least fixpoint of T_Π .

Immediate Consequence Operator: Example

Example

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

$$T_{\Pi}^0(X) = \emptyset$$

$$T_{\Pi}^1(X) = \{p, q\} = T_{\Pi}(T_{\Pi}^0(X)) = T_{\Pi}(\emptyset)$$

$$T_{\Pi}^2(X) = \{p, q, r\} = T_{\Pi}(T_{\Pi}^1(X)) = T_{\Pi}(\{p, q\})$$

$$T_{\Pi}^3(X) = \{p, q, r, t\} = T_{\Pi}(T_{\Pi}^2(X)) = T_{\Pi}(\{p, q, r\})$$

$$T_{\Pi}^4(X) = \{p, q, r, t, s\} = T_{\Pi}(T_{\Pi}^3(X)) = T_{\Pi}(\{p, q, r, t\})$$

$$T_{\Pi}^5(X) = \{p, q, r, t, s\} = T_{\Pi}(T_{\Pi}^4(X)) = T_{\Pi}(\{p, q, r, t, s\})$$

$$T_{\Pi}^6(X) = \{p, q, r, t, s\} = T_{\Pi}(T_{\Pi}^5(X)) = T_{\Pi}(\{p, q, r, t, s\})$$

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

Logic Programs with Variables

Definition (Alphabet)

Let Π be a logic program.

- **Herbrand Universe** U^Π : Set of constants in Π .
- **Herbrand Base** B^Π : Set of (variable-free) atoms constructible from U^Π .
We usually denote B^Π as \mathcal{A} and call it **Alphabet**

Definition (Grounding of a rule)

Let Π be a logic program (with variables). The ground instantiation of a rule $r \in \Pi$ is the set of variable-free rules obtained by replacing all variables in r by elements from U^Π :

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

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

Logic Programs with Variables

Definition (Grounding of a Program)

Let Π be a logic program (with variables). The **Ground Instantiation** of a program Π is the set of all ground instantiations of its rules

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

Definition (Answer Set a Logic Program with Variables)

Let Π be a normal logic program with variables. A set of ground atoms X (i.e. $X \subseteq B^\Pi$) is an answer set of Π iff X is an answer set of $\text{ground}(\Pi)$, i.e. iff

$$\text{Cn}(\text{ground}(\Pi)^X) = X$$

Logic Programs with Variables: Example

Example

Consider the program:

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

We have:

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

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

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

Logic Programs with Variables: Example

Example

Consider the program:

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

We have:

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

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

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

- Intelligent grounding!

- A normal rule is **safe**, if each of its variables also occurs in some positive body literal
- A normal program is safe, if all of its rules are safe

Example

| | Safe ? |
|---|--------|
| $d(a)$ | ✓ |
| $d(c)$ | ✓ |
| $d(d)$ | ✓ |
| $p(a, b)$ | ✓ |
| $p(b, c)$ | ✓ |
| $p(c, d)$ | ✓ |
| $p(X, Z) \leftarrow p(X, Y), p(Y, Z)$ | ✓ |
| $q(a)$ | ✓ |
| $q(b)$ | ✓ |
| $q(X) \leftarrow \text{not } r(X), d(X)$ | ✓ |
| $r(X) \leftarrow \text{not } q(X), d(X)$ | ✓ |
| $s(X) \leftarrow \text{not } r(X), p(X, Y), q(Y)$ | ✓ |

Programs with Integrity Constraints: Syntax

- Integrity constraints eliminate unwanted candidate solutions

Definition (Integrity Constraint)

An integrity constraint is (a special kind of rule) of the form

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

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

Example

The integrity constraint

$$\leftarrow \text{painted}(X, C), \text{painted}(Y, C), \text{adjacent}(X, Y).$$

intuitively, would prevent the existence of answer sets in which two adjacent nodes (X and Y) are painted with the same colour (C).

Programs with Integrity Constraints: Semantics

Definition (Semantics of Integrity Constraints)

An integrity constraint of the form

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

is mapped into the rule (where x is a new atom not appearing anywhere else in the program)

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

Example

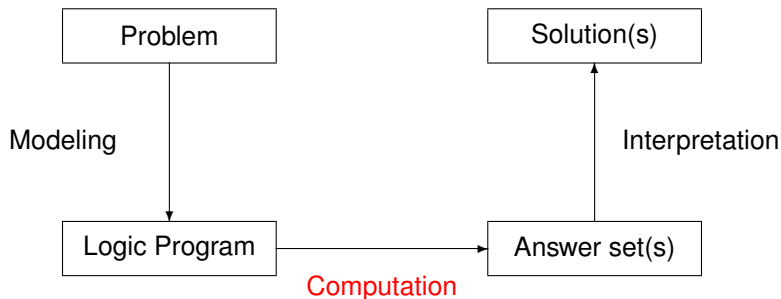
Compare the answer sets of the following logic programs:

$$\Pi = \{ p \leftarrow \text{not } q. \quad q \leftarrow \text{not } p. \}$$

$$\Pi' = \{ p \leftarrow \text{not } q. \quad q \leftarrow \text{not } p. \quad \leftarrow p. \}$$

$$\Pi'' = \{ p \leftarrow \text{not } q. \quad q \leftarrow \text{not } p. \quad \leftarrow \text{not } p. \}$$

Problem solving in ASP: Computation



Standard Computation Scheme

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

Definition ($answerset_{\Pi}(T, F)$)

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

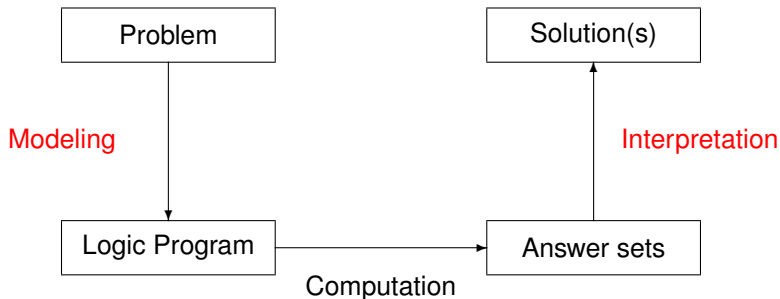
Comments:

- (T, F) is supposed to be a 3-valued model such that $T \subseteq X$ and $F \cap X = \emptyset$ for an answer set X of Π .
- Key operations: $propagation_{\Pi}(T, F)$ and '**select** $A \in \mathcal{A} \setminus (T \cup F)$ '
- Worst case complexity: $\mathcal{O}(2^{|\mathcal{A}|})$

- 1 Answer Set Programming
 - Introduction
 - Normal Logic Programs
 - Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity
- 2 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 | $\{ colored(1,r), colored(2,b), colored(3,g), \dots \}$ | | |

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

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

ASP Basic Methodology

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

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

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

In a Nutshell...

Logic Program = Data + Generator + Tester [+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

$a \leftarrow \text{not } a'$

$a' \leftarrow \text{not } a$

$b \leftarrow \text{not } b'$

$b' \leftarrow \text{not } b$

Tester

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

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

Answer set

$A_1 = \{a, b\}$

$A_2 = \{a', b'\}$

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.

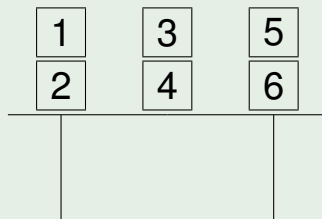
Solution

| $C(I)$ | vertex/1 | arc/2 | start/1 |
|--------|-----------------|--------------|--|
| $C(P)$ | $inPath(X, Y)$ | \leftarrow | $arc(X, Y), \text{ not } outPath(X, Y).$ |
| | $outPath(X, Y)$ | \leftarrow | $arc(X, Y), \text{ not } inPath(X, Y).$ |
| | | \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 | $vertex(X), \text{ not } reached(X).$ |
| | | \leftarrow | $inPath(Y, X), start(X).$ |

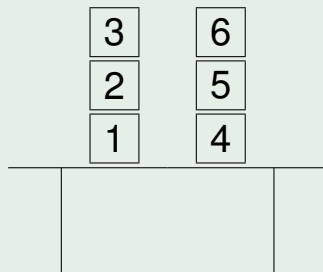
Planning in the Blocksworld

Example (Scenario)

Initial situation



Goal situation



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).
```

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) .
```

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.
```

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)

```
smodels version 2.25. Reading...done
Answer: 1
Stable Model: move(1,table,0) move(3,table,0)
               move(2,1,1)      move(5,4,1)
               move(3,2,2)      move(6,5,2)

Duration: 0.050
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 507
Number of rules: 3026
Number of picked atoms: 24
Number of forced atoms: 13
Number of truth assignments: 944
Size of searchspace (removed): 0 (0)
```

- 1 Answer Set Programming
 - Introduction
 - Normal Logic Programs
 - Modeling

- Disjunctive Logic Programs
 - Nested Logic Programs
 - Propositional Theories
 - Computational Complexity
- 2 Bibliography

Disjunctive Logic Programs: Syntax

Definition (Disjunctive Rule)

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

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

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

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

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

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

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

| | | | |
|--------|---|------------------------|------------------------|
| $C(I)$ | $vertex(1) \leftarrow$ | $vertex(2) \leftarrow$ | $vertex(3) \leftarrow$ |
| | $edge(1,2) \leftarrow$ | $edge(2,3) \leftarrow$ | $edge(3,1) \leftarrow$ |
| $C(P)$ | $colored(V,r); colored(V,b); colored(V,g) \leftarrow vertex(V)$ $\leftarrow edge(V,U), colored(V,C), colored(U,C)$ | | |
| AS's | $\{ colored(1,r), colored(2,b), colored(3,g), \dots \}$ | | |

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 $\{b, c\}$.
- $\Pi_4 = \{a ; b \leftarrow c , b \leftarrow \textit{not } a, \textit{not } c , a ; c \leftarrow \textit{not } b\}$ 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 $X \not\subseteq Y$

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 $\text{body}^+(r) \subseteq X$, $\text{body}^-(r) \cap X = \emptyset$, and $\text{head}(r) \cap X = \{A\}$

Disjunctive Logic Programs: Example with variables

Example

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

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

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

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

Disjunctive Logic Programs: Example with variables

Example

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

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

1 Answer Set Programming

- Introduction
- Normal Logic Programs
- Modeling

- Disjunctive Logic Programs

- **Nested Logic Programs**

- Propositional Theories

- Computational Complexity

2 Bibliography

Nested Logic Programs: Syntax

Definition (Formulas)

Formulas are formed from propositional atoms, \top and \perp , using negation-as-failure (*not*), conjunction (\wedge), and disjunction (\vee).

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.

Nested Logic Programs: Semantics

Notation

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

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 ,
- $X \models \top$,
- $X \not\models \perp$,
- $X \models (F, G)$ if $X \models F$ and $X \models G$,
- $X \models (F; G)$ if $X \models F$ or $X \models G$,
- $X \models \text{not } F$ if $X \not\models F$.

A set X of atoms satisfies a nested program Π , written $X \models \Pi$, iff for any $r \in \Pi$, $X \models head(r)$ whenever $X \models body(r)$.

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

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

- $\Pi_1 = \{(p ; \text{not } p) \leftarrow \top\}$
 - For $X = \emptyset$, we get
 - $\Pi_1^\emptyset = \{(p ; \top) \leftarrow \top\}$
 - $\min_{\subseteq}(\Pi_1^\emptyset) = \{\emptyset\}$. ✓
 - For $X = \{p\}$, we get
 - $\Pi_1^{\{p\}} = \{(p ; \perp) \leftarrow \top\}$
 - $\min_{\subseteq}(\Pi_1^{\{p\}}) = \{\{p\}\}$. ✓
- $\Pi_2 = \{p \leftarrow \text{not not } p\}$
 - For $X = \emptyset$, we get $\Pi_2^\emptyset = \{p \leftarrow \perp\}$ 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))
 - $F \leftarrow G, \text{not not } H$ is equivalent to $F ; \text{not } H \leftarrow G$
 - $F ; \text{not not } G \leftarrow H$ is equivalent to $F \leftarrow H, \text{not } G$
 - $\text{not not not } F$ is equivalent to $\text{not } F$

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

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

1 Answer Set Programming

- Introduction
- Normal Logic Programs
- Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- **Propositional Theories**
- Computational Complexity

2 Bibliography

Propositional Theories: Syntax

Definition (Formulas)

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

Notation

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

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:

- $F^X = \perp$ 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 $F = \sim G = (G \rightarrow \perp)$,
then $F^X = (\perp \rightarrow \perp) = \top$, if $X \not\models G$, and $F^X = \perp$, 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
 $\mathcal{F}_1^{\emptyset} = \{\perp \vee (\perp \rightarrow \perp)\}$ and $\min_{\subseteq}(\mathcal{F}_1^{\emptyset}) = \{\emptyset\}$. ✓
- $\mathcal{F}_2 = \{p \vee (\sim p \rightarrow (q \wedge r))\}$
 - For $X = \emptyset$, we get
 $\mathcal{F}_2^{\emptyset} = \{\perp\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\emptyset}) = \emptyset$. ✗
 - For $X = \{p\}$, we get
 $\mathcal{F}_2^{\{p\}} = \{p \vee (\perp \rightarrow \perp)\}$ and $\min_{\subseteq}(\mathcal{F}_2^{\{p\}}) = \{\emptyset\}$. ✗
 - For $X = \{q, r\}$, we get
 $\mathcal{F}_2^{\{q,r\}} = \{\perp \vee (\top \rightarrow (q \wedge r))\}$ 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:

- $\tau[(F \leftarrow G)] = (\tau[G] \rightarrow \tau[F])$,
- $\tau[\perp] = \perp$,
- $\tau[\top] = \top$,
- $\tau[F] = F$ if F is an atom,
- $\tau[\text{not } F] = \sim \tau[F]$,
- $\tau[(F, G)] = (\tau[F] \wedge \tau[G])$,
- $\tau[(F; G)] = (\tau[F] \vee \tau[G])$.

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

1 Answer Set Programming

- Introduction
- Normal Logic Programs
- Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity

2 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 **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 **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 **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 **NP^{NP}**-complete.

- 1 Answer Set Programming
 - Introduction
 - Normal Logic Programs
 - Modeling

- Disjunctive Logic Programs
- Nested Logic Programs
- Propositional Theories
- Computational Complexity

- 2 Bibliography



C. Baral.

Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.



S. Brass and J. Dix.

Semantics of (disjunctive) logic programs based on partial evaluation.
Journal of Logic Programming, 40(1):1–46, 1999.



P. Cabalar and P. Ferraris.

Propositional theories are strongly equivalent to logic programs.
Theory and Practice of Logic Programming, 7(6):745–759, 2007.



K. Clark.

Negation as failure.

In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages
293–322. Plenum Press, 1978.



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.



E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.

Complexity and expressive power of logic programming.

ACM Computing Surveys, 33(3):374–425, 2001.



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.



T. Eiter, M. Fink, H. Tompits, and S. Woltran.

Simplifying logic programs under uniform and strong equivalence.
In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Artificial Intelligence*, pages 87–99. Springer-Verlag, 2004.



Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.

Recursive aggregates in disjunctive logic programs: Semantics and complexity.

In José Júlio Alferes and João Alexandre Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 200–212. Springer, 2004.



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.



Paolo Ferraris and Vladimir Lifschitz.

Mathematical foundations of answer set programming.

In Sergei N. Artëmov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods, editors, *We Will Show Them! (1)*, pages 615–664. College Publications, 2005.



Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz.

A new perspective on stable models.

In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 372–379, 2007.



Paolo Ferraris.

Answer sets for propositional theories.

In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 2005.



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>.



Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf.

The well-founded semantics for general logic programs.

J. ACM, 38(3):620–650, 1991.



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. The MIT Press, 1988.

Bibliography VI



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.



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.



R. Kowalski.

Logic for data description.

In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 77–103. Plenum Press, 1978.



Joohyung Lee, Vladimir Lifschitz, and Ravi Palla.

A reductive semantics for counting and choice in answer set programming.

In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 472–479. AAAI Press, 2008.



Joohyung Lee.

A model-theoretic counterpart of loop formulas.

In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, 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.

Bibliography VIII



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.



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, D. Pearce, and A. Valverde.

Strongly equivalent logic programs.

ACM Transactions on Computational Logic, 2(4):526–541, 2001.



Vladimir Lifschitz, David Pearce, and Agustín Valverde.

Strongly equivalent logic programs.

ACM Trans. Comput. Log., 2(4):526–541, 2001.

Bibliography IX



V. Lifschitz.

Answer set programming and plan generation.

Artificial Intelligence, 138(1-2):39–54, 2002.



Vladimir Lifschitz.

Twelve definitions of a stable model.

In Maria Garcia de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2008.



Fangzhen Lin and Yuting Zhao.

Assat: computing answer sets of a logic program by sat solvers.

Artif. Intell., 157(1-2):115–137, 2004.



Fangzhen Lin and Yi Zhou.

From answer set logic programming to circumscription via logic of gk.

In Manuela M. Veloso, editor, *IJCAI*, pages 441–446, 2007.



J. Lloyd.

Foundations of Logic Programming.

Symbolic Computation. Springer-Verlag, 2nd edition, 1987.



J. McCarthy and P. J. Hayes.

Some philosophical problems from the standpoint of artificial intelligence.
pages 26–45, 1987.



John McCarthy.

Circumscription - a form of non-monotonic reasoning.

Artif. Intell., 13(1-2):27–39, 1980.



Drew V. McDermott and Jon Doyle.

Non-monotonic logic i.

Artif. Intell., 13(1-2):41–72, 1980.



Drew V. McDermott.

Nonmonotonic logic ii: Nonmonotonic modal theories.

J. ACM, 29(1):33–57, 1982.

Bibliography XI



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.



M. Osorio, J. Navarro, and J. Arrazola.

Equivalence in answer set programming.

In A. Pettorossi, editor, *Proceedings of the Eleventh International Workshop on Logic Based Program Synthesis and Transformation (LOPSTR'01)*, volume 2372 of *Lecture Notes in Computer Science*, pages 57–75. Springer-Verlag, 2001.

Bibliography XII



David Pearce, Hans Tompits, and Stefan Woltran.

Encodings for equilibrium logic and logic programs with nested expressions.

In Pavel Brazdil and Alípio Jorge, editors, *EPIA*, volume 2258 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2001.



David Pearce.

A new logical characterisation of stable models and answer sets.

In Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusiński, editors, *NMELP*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 1996.



David Pearce.

Equilibrium logic.

Ann. Math. Artif. Intell., 47(1-2):3–41, 2006.



Raymond Reiter.

A logic for default reasoning.

Artif. Intell., 13(1-2):81–132, 1980.



Domenico Saccà and Carlo Zaniolo.

Stable models and non-determinism in logic programs with negation.
In *PODS*, pages 205–217. ACM Press, 1990.



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>.



H. Turner.

Strong equivalence made easy: nested expressions and weight constraints.
Theory and Practice of Logic Programming, 3(4-5):609–622, 2003.



Maarten H. van Emden and Robert A. Kowalski.

The semantics of predicate logic as a programming language.

J. ACM, 23(4):733–742, 1976.