Answer Set Programming

Elena Mastria
Department of Mathematics and Computer Science, University of Calabria, Italy
elena.mastria@unical.it

15 November 2022

Outline

Introduction

Idea

Design a representation of the world (limited to the small part relevant to the problem at hand) by means of a logic theory; then, solve a problem via automated reasoning on its basis.

Answer Set Programming (ASP)

- **Declarative** programming language for **K**nowledge **R**epresentation and **R**easoning [GL91, EGM97, MT99, Nie99, EFLP00, BET11]
- Logic paradigm based on rules

Answer Set Programming (ASP)

- **Declarative** programming language for **K**nowledge **R**epresentation and **R**easoning [GL91, EGM97, MT99, Nie99, EFLP00, BET11]
- Logic paradigm based on rules

Standard Procedural Programming

- Need for a solving method/algorithm
- Define instructions to be executed “step by step”
- Tell the machine WHAT to do, HOW to solve the problem

ASP-based Declarative Approach

- Specify the features of the desired solution
- NO need for algorithm design
- Just provide the problem specification in form of a logic program
- **Models** will represent solutions
- You just need an ASP solver that computes such models

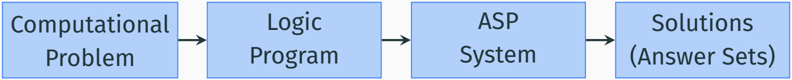
Answer Set Programming (ASP)

- **Declarative** programming language for **K**nowledge **R**epresentation and **R**easoning [GL91, EGM97, MT99, Nie99, EFLP00, BET11]
- Logic paradigm based on rules
- Able to deal with **incomplete knowledge**
- Able to model **non-monotonic reasoning**
  - does not store consequences
  - what deduced up to a certain point can be invalidated after the acquisition of new knowledge

Answer Set Programming (ASP)

- **Declarative** programming language for **K**nowledge **R**epresentation and **R**easoning [GL91, EGM97, MT99, Nie99, EFLP00, BET11]
- Logic paradigm based on rules
- Able to deal with **incomplete knowledge**
- Able to model **non-monotonic reasoning**
- Successfully employed in both academy and industry

Knowledge Representation and Reasoning with ASP



The basic construct of ASP is the one of **rule**:

$$\underbrace{Head}_{\textbf{disjunction}} \text{ :- } \underbrace{Body}_{\textbf{conjunction}} .$$

- Interpreted according to common sense principles
- Roughly, its intuitive semantics corresponds to an implication

```
parent("James Potter","Harry Potter").
son(X,Y) :- parent(Y,X). ⇒ son("Harry Potter","James Potter")
```

ASP Syntax and Semantics

Core Syntax

- An **ASP logic program** is a (finite) set of **rules** of form:

$$\underbrace{a_1 \mid \ldots \mid a_n}_{\textbf{head atoms}} \text{ :- } \underbrace{b_1, \ldots, b_k, \texttt{not } b_{k+1}, \ldots, \texttt{not } b_m.}_{\textbf{body literals}}$$

- A **literal** is either **positive** b or **negative** $\texttt{not } b$ where b is an **atom**
- An **atom** has form $p(t_1, \ldots, t_n)$ (typical FO signature), where:
  - p is a **predicate** of arity n
  - $t_1, \ldots, t_n$ are **terms**
- A **term** is either a **variable** or a **constant**

$\rightarrow \texttt{saga(}\underbrace{\texttt{"Harry Potter"}}_{\texttt{string constant}}\texttt{)}$

$\rightarrow \texttt{numer\_of\_siblings(}\underbrace{\texttt{"Ron"}}_{\texttt{string constant}}\texttt{, }\underbrace{\texttt{6}}_{\texttt{numeric constant}}\texttt{)}$

$\rightarrow \texttt{parent(}\underbrace{\texttt{Y,X}}_{\texttt{variables}}\texttt{)}$

$\rightarrow \texttt{horcrux(}\underbrace{\texttt{ring}}_{\texttt{symbolic constant}}\texttt{)}$

ASP Syntax

Core Syntax

- An **ASP logic program** is a (finite) set of **rules** of form:

$$\underbrace{a_1 \mid \ldots \mid a_n}_{\textbf{head atoms}} \texttt{:-} \underbrace{b_1, \ldots, b_k, \texttt{not}\ b_{k+1}, \ldots, \texttt{not}\ b_m}_{\textbf{body literals}}.$$

  - A rule with no literal in its body is a **fact**
  - A rule with no atom in its head is a **strong constraint**
- A program (rule/literal/atom) with no variables is **ground**

```
% fact
saga("Harry Potter").
% (disjunctive) rule
like(X) | dislike(X) :- saga(X).
```

$$\Downarrow$$

```
like("Harry Potter") | dislike("Harry Potter").
```

Core Syntax

- An **ASP logic program** is a (finite) set of **rules** of form:

$$\underbrace{a_1 \mid \ldots \mid a_n}_{\text{head atoms}} \text{ :- } \underbrace{b_1, \ldots, b_k, \texttt{not}\ b_{k+1}, \ldots, \texttt{not}\ b_m}_{\text{body literals}}.$$

  - A rule with no literal in its body is a **fact**
  - A rule with no atom in its head is a **strong constraint**
- A program (rule/literal/atom) with no variables is **ground**

```
%strong constraint
:- pronounce("You-Know-Who").
```

Linguistic Extensions

- Many extensions:
  - new term types: functional terms, arithmetic terms
  - new atom and literal types: aggregate literals, built-in atoms
  - new rule types: weak constraints, choice rules, queries
- Standard input language: **ASP-Core-2** [CFG+12]

```
novel("Harry Potter and the Philosopher's Stone").
novel("Harry Potter and the Chamber of Secrets").
novel("Harry Potter and the Prisoner of Azkaban").
novel("Harry Potter and the Goblet of Fire").
novel("Harry Potter and the Order of the Phoenix").
novel("Harry Potter and the Half-Blood Prince").
novel("Harry Potter and the Deathly Hallows").
num_of_novels(N) :- #count{X:novel(X)}=N. ⇒ num_of_novels(7).
is_not_harry_potter_saga :- num_of_novels(N), N<7.
```

ASP Semantics

ASP semantics is based the concept of answer set [GL91]

Answer sets are only defined for ground programs

- For every non-ground program, a semantically equivalent ground program can be defied
  - **instantiation**, **grounding** process
- A program with variables is just a shorthand for its ground instantiation!

Program Instantiation

Given an ASP program $\mathcal{P}$

- **Herbrand Universe** ($U_{\mathcal{P}}$): set of constants occurring in program $\mathcal{P}$

```
like_herry_potter(X) | dislike_herry_potter(X):- person(X).
person("Pina").  person("Ugo").
```

$\Rightarrow U_{\mathcal{P}} = \{$"Pina", "Ugo"$\}$

Program Instantiation

Given an ASP program $\mathcal{P}$

- Herbrand Universe ($U_{\mathcal{P}}$): set of constants occurring in program $\mathcal{P}$
- **Herbrand Base** ($B_{\mathcal{P}}$): set of ground atoms constructible from $U_{\mathcal{P}}$ and predicates in $\mathcal{P}$

```
like_herry_potter(X) | dislike_herry_potter(X):- person(X).
person("Pina").  person("Ugo").
```

$\Rightarrow U_{\mathcal{P}} = \{$"Pina","Ugo"$\}$

$\Rightarrow B_{\mathcal{P}} = \{$dislike_herry_potter("Pina"),
dislike_herry_potter("Ugo"),
like_herry_potter("Pina"),
like_herry_potter("Ugo"),
person("Pina"), person("Ugo")$\}$

Program Instantiation

Given an ASP program $\mathcal{P}$

- Herbrand Universe ($U_{\mathcal{P}}$): set of constants occurring in program $\mathcal{P}$
- Herbrand Base ($B_{\mathcal{P}}$): set of ground atoms constructible from $U_{\mathcal{P}}$ and predicates in $\mathcal{P}$
- **Instantiation** $ground(\mathcal{P})$: set of the ground instances of rules in $\mathcal{P}$
  - for each rule $r \in \mathcal{P}$: replace each variable in r by a constant in $U_{\mathcal{P}}$

```
like_herry_potter(X) | dislike_herry_potter(X):- person(X).
person("Pina").  person("Ugo").
```

$$\Downarrow$$

```
like_herry_potter("Pina")|dislike_herry_potter("Pina") :- person("Pina").
like_herry_potter("Ugo")|dislike_herry_potter("Ugo") :- person("Ugo").
person("Pina").  person("Ugo").
```

Interpretations

Given an ASP program $\mathcal{P}$ the **Herbrand Interpretation** I for $\mathcal{P}$ is a consistent subset of $B_{\mathcal{P}}$

- an atom a is true w.r.t. I if $a \in I$; otherwise it is false
- a literal `not` a is true w.r.t. I if $a \notin I$; otherwise it is false
- I is consistent if, for each atom a, $\{a, \neg a\} \not\subseteq I$

$$\Downarrow$$

Models

Given an interpretation I:

- I is a model for $\mathcal{P}$ if, for every rule $r \in \mathcal{P}$, whenever the body of r is true w.r.t. I, the head of r is also True w.r.t. I
- I is a *minimal* model for $\mathcal{P}$ if no model N s.t. $N \subset I$ exists

ASP Semantics

```
like_herry_potter(X) | dislike_herry_potter(X):- person(X).
person("Pina").  person("Ugo").
```

$\Downarrow$

```
like_herry_potter("Pina")|dislike_herry_potter("Pina") :- person("Pina").
like_herry_potter("Ugo")|dislike_herry_potter("Ugo") :- person("Ugo").
person("Pina").  person("Ugo").
```

```
I1 = {person("Ugo")} ⇒ (not a model)
I2 = {dislike_herry_potter("Ugo"), person("Ugo"),
      like_herry_potter("Pina"), dislike_herry_potter("Pina"),
      person("Pina")} ⇒ (model, non minimal)
I3 = {dislike_herry_potter("Ugo"), person("Ugo"), person("Pina"),
      like_herry_potter("Pina")} ⇒ (model, minimal)
...
```

ASP Semantics

Reduct

The **reduct** or of a program $\mathcal{P}$ w.r.t. an interpretation I is the program $\mathcal{P}^{\mathcal{I}}$, obtained from $\mathcal{P}$ by

1. deleting all rules with a negative literal false w.r.t. I
2. deleting the negative literals from the bodies of the remaining rules

Answer Set

Given an interpretation I for a program $\mathcal{P}$, I is an **answer set** for $\mathcal{P}$ if it is a minimal model for $\mathcal{P}^{\mathcal{I}}$

$\mathcal{P}$

```
a :- d, not b.
b :- not d.
d.
```

I = { a, d }

a :- d, ~~not b~~.
~~b :- not d.~~
d.

$\mathcal{P}^{\mathcal{I}}$

```
a :- d.
d.
```

$\Downarrow$

I is a minimal model of $\mathcal{P}^{\mathcal{I}}$ and therefore it is an answer set of $\mathcal{P}$

Knowledge Representation and Reasoning with ASP

A Practical Example

Vertex Cover

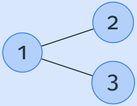
Given an undirected graph $G = (V, E)$ select $S \subseteq V$ such that all edges are covered (i.e. for every edge $(a, b) \in E$ either $a \in S$ or $b \in S$).

A Practical Example

Vertex Cover

Given an undirected graph $G = (V, E)$ select $S \subseteq V$ such that all edges are covered (i.e. for every edge $(a, b) \in E$ either $a \in S$ or $b \in S$).

1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).$ $\Rightarrow$ **Facts**
2: $inS(X) \mid outS(X)$:- $node(X).$ $\Rightarrow$ **Disjunctive Rule**
3: :- $edge(X,Y), \texttt{not}\ inS(X), \texttt{not}\ inS(Y).$ $\Rightarrow$ **Strong Constraint**

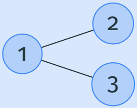


A Practical Example

Vertex Cover

Given an undirected graph $G = (V, E)$ select $S \subseteq V$ such that all edges are covered (i.e. for every edge $(a, b) \in E$ either $a \in S$ or $b \in S$).

1: $node(1).\ node(2).\ node(3).\ edge(1, 2).\ edge(1, 3).$ $\Rightarrow$ **Facts**
2: $inS(X) \mid outS(X)$:- $node(X).$ $\Rightarrow$ **Disjunctive Rule**
3: :- $edge(X, Y)$, not $inS(X)$, not $inS(Y)$. $\Rightarrow$ **Strong Constraint**



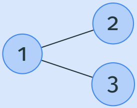
1: $node(1).\ node(2).\ node(3).\ edge(1, 2).\ edge(1, 3).$
2: $inS(1) \mid outS(1)$:- $node(1).$
3: $inS(2) \mid outS(2)$:- $node(2).$
4: $inS(3) \mid outS(3)$:- $node(3).$
5: :- $edge(1, 1)$, not $inS(1)$, not $inS(1)$.
6: :- $edge(1, 2)$, not $inS(1)$, not $inS(2)$.
7: :- $edge(1, 3)$, not $inS(1)$, not $inS(3)$.
8: :- $edge(2, 1)$, not $inS(2)$, not $inS(1)$.
9: :- $edge(2, 2)$, not $inS(2)$, not $inS(2)$.
10: :- $edge(2, 3)$, not $inS(3)$, not $inS(3)$.
11: :- $edge(3, 1)$, not $inS(3)$, not $inS(1)$.
12: :- $edge(3, 2)$, not $inS(3)$, not $inS(2)$.
13: :- $edge(3, 3)$, not $inS(3)$, not $inS(3)$.

A Practical Example

Vertex Cover

Given an undirected graph $G = (V, E)$ select $S \subseteq V$ such that all edges are covered (i.e. for every edge $(a, b) \in E$ either $a \in S$ or $b \in S$).

1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).\ \Rightarrow$ **Facts**
2: $inS(X) \mid outS(X)$:- $node(X).\ \Rightarrow$ **Disjunctive Rule**
3: :- $edge(X, Y), \texttt{not}\ inS(X), \texttt{not}\ inS(Y).\ \Rightarrow$ **Strong Constraint**



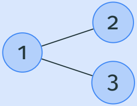
1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).$
2: $inS(1) \mid outS(1)$:- $\cancel{node(1)}.$
3: $inS(2) \mid outS(2)$:- $\cancel{node(2)}.$
4: $inS(3) \mid outS(3)$:- $\cancel{node(3)}.$
5: $\cancel{:\!-\ edge(1,1), \texttt{not}\ inS(1), \texttt{not}\ inS(1).}$
6: :- $\cancel{edge(1,2)}, \texttt{not}\ inS(1), \texttt{not}\ inS(2).$
7: :- $\cancel{edge(1,3)}, \texttt{not}\ inS(1), \texttt{not}\ inS(3).$
8: $\cancel{:\!-\ edge(2,1), \texttt{not}\ inS(2), \texttt{not}\ inS(1).}$
9: $\cancel{:\!-\ edge(2,2), \texttt{not}\ inS(2), \texttt{not}\ inS(2).}$
10: $\cancel{:\!-\ edge(2,3), \texttt{not}\ inS(3), \texttt{not}\ inS(3).}$
11: $\cancel{:\!-\ edge(3,1), \texttt{not}\ inS(3), \texttt{not}\ inS(1).}$
12: $\cancel{:\!-\ edge(3,2), \texttt{not}\ inS(3), \texttt{not}\ inS(2).}$
13: $\cancel{:\!-\ edge(3,3), \texttt{not}\ inS(3), \texttt{not}\ inS(3).}$

A Practical Example

Vertex Cover

Given an undirected graph $G = (V, E)$ select $S \subseteq V$ such that all edges are covered (i.e. for every edge $(a, b) \in E$ either $a \in S$ or $b \in S$).

1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).$ ⇒ **Facts**
2: $inS(X) \mid outS(X) \text{ :- } node(X).$ ⇒ **Disjunctive Rule**
3: $\text{ :- } edge(X,Y), \texttt{not}\ inS(X), \texttt{not}\ inS(Y).$ ⇒ **Strong Constraint**



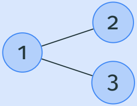
1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).$
2: $inS(1) \mid outS(1).$
3: $inS(2) \mid outS(2).$
4: $inS(3) \mid outS(3).$
5: $\text{ :- } \texttt{not}\ inS(1),\ \texttt{not}\ inS(2).$
6: $\text{ :- } \texttt{not}\ inS(1),\ \texttt{not}\ inS(3).$

A Practical Example

Vertex Cover

Given an undirected graph $G = (V, E)$ select $S \subseteq V$ such that all edges are covered (i.e. for every edge $(a, b) \in E$ either $a \in S$ or $b \in S$).

1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).$ $\Rightarrow$ **Facts**
2: $inS(X) \mid outS(X)$:- $node(X).$ $\Rightarrow$ **Disjunctive Rule**
3: :- $edge(X,Y), \texttt{not}\ inS(X), \texttt{not}\ inS(Y).$ $\Rightarrow$ **Strong Constraint**



1: $node(1).\ node(2).\ node(3).\ edge(1,2).\ edge(1,3).$
2: $inS(1) \mid outS(1).$
3: $inS(2) \mid outS(2).$
4: $inS(3) \mid outS(3).$
5: :- $\texttt{not}\ inS(1),\ \texttt{not}\ inS(2).$
6: :- $\texttt{not}\ inS(1),\ \texttt{not}\ inS(3).$

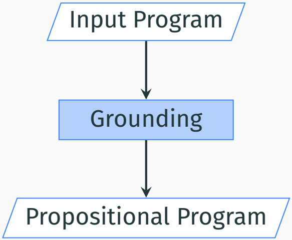
$$AS_1 = Facts \cup \{inS(1), outS(2), outS(3)\} \qquad S = \{1\}$$
$$AS_2 = Facts \cup \{outS(1), inS(2), inS(3)\} \qquad S = \{2,3\}$$
$$AS_3 = Facts \cup \{inS(1), inS(2), outS(3)\} \qquad S = \{1,2\}$$
$$AS_4 = Facts \cup \{inS(1), outS(2), inS(3)\} \qquad S = \{1,3\}$$
$$AS_5 = Facts \cup \{inS(1), inS(2), inS(3)\} \qquad S = \{1,2,3\}$$

Canonical approach to solve an ASP program P over a set of facts F:

ASP Computation

Canonical approach to solve an ASP program P over a set of facts F:

1. **Grounding** (or **Instantiation**) phase:
    - Produces a semantically equivalent ground (i.e., propositional) program: $ground(P \cup F) \subseteq ground_{Herbrand}(P \cup F)$

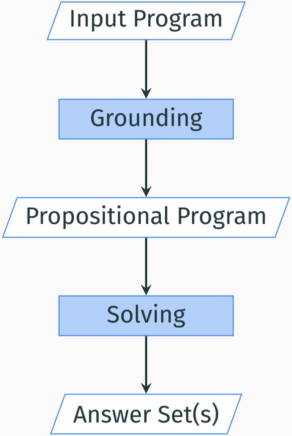


ASP Computation

Canonical approach to solve an ASP program P over a set of facts F:

1. **Grounding** (or **Instantiation**) phase:
   - Produces a semantically equivalent ground (i.e., propositional) program: $ground(P \cup F) \subseteq ground_{Herbrand}(P \cup F)$
2. **Solving** phase:
   - Generates the Answer Set(s) $AS(P \cup F)$

$$AS(P \cup F) \equiv AS(ground(P \cup F)) \equiv AS(ground_{Herbrand}(P \cup F))$$



ASP Systems

Canonical approach: Ground&Solve

- Stand-alone grounders: LPARSE [Syr01], GRINGO [GKKS11], I-DLV [CFPZ17]
- Stand-alone solvers: CMODELS [GLM06], SMODELS [SNS02], CLASP [GKS12], WASP [ADLR15]
- Monolithic systems: DLV [LPF+06, ACD+17], CLINGO [GKKS14]

Other approaches

- Lazy Grounding: GASP [DDPR09], ASPERIX [LN09, LBSG17], OMIGA [dCHM12], ALPHA [Wei17]
- Translation-based systems: LPTOSAT [Jan06]
- ML-based approaches analyse the grounding to select the best solver [MPR14, CDF+20]

Thanks for your attention :)

Questions?

References i

Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari.
The ASP system DLV2.
In Balduccini and Janhunen [BJ17], pages 215–221.

Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca.
Advances in WASP.
In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, volume 9345 of *Lecture Notes in Computer Science*, pages 40–54. Springer, 2015.

Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

Marcello Balduccini and Tomi Janhunen, editors.
***Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*. Springer, 2017.**

Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Simona Perri, and Jessica Zangari.
Efficiently coupling the I-DLV grounder with ASP solvers.
Theory Pract. Log. Program., 20(2):205–224, 2020.

Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub.
Asp-core-2: Input language format, 2012.
https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03b.pdf.

Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari.
$\mathcal{I}$-*DLV*: The New Intelligent Grounder of DLV.
Intelligenza Artificiale, 11(1):5–20, 2017.

Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors.
***Logics in Artificial Intelligence - 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings*, volume 7519 of *Lecture Notes in Computer Science*. Springer, 2012.**

Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi.
GASP: Answer Set Programming with Lazy Grounding.
Fundamenta Informaticae, 96(3):297–322, 2009.

Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Declarative Problem-Solving using the DLV System.
In *Logic-based artificial intelligence*, pages 79–103. Springer, 2000.

Thomas Eiter, Georg Gottlob, and Heikki Mannila.

Disjunctive Datalog.

ACM Transactions on Database Systems, 22(3):364–418, September 1997.

Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub.

Advances in *gringo* series 3.

In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 345–351. Springer, 2011.

M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.

***Clingo* = ASP + control: Preliminary report.**

In M. Leuschel and T. Schrijvers, editors, *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, volume arXiv:1405.3694v1, 2014. Theory and Practice of Logic Programming, Online Supplement.

M. Gebser, B. Kaufmann, and T. Schaub.

Conflict-driven answer set solving: From theory to practice.

Artificial Intelligence, 187-188:52–89, 2012.

Michael Gelfond and Vladimir Lifschitz.

Classical Negation in Logic Programs and Disjunctive Databases.

New Generation Computing, 9(3/4):365–385, 1991.

Enrico Giunchiglia, Yulia Lierler, and Marco Maratea.

Answer Set Programming Based on Propositional Satisfiability.

Journal of Automated Reasoning, 36(4):345–377, 2006.

References iv

Tomi Janhunen.
Some (in)translatability results for normal logic programs and propositional theories.
Journal of Applied Non-Classical Logics, 16(1-2):35–86, 2006.

Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia.
ASPeRiX, a first-order forward chaining approach for answer set computing.
Theory and Practice of Logic Programming, 17(3):266–310, 2017.

Claire Lefèvre and Pascal Nicolas.
The first version of a new asp solver : Asperix.
In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning — 10th International Conference (LPNMR 2009)*, volume 5753 of *Lecture Notes in Computer Science*, pages 522–527. Springer Verlag, September 2009.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello.
The dlv system for knowledge representation and reasoning.
ACM Transactions on Computational Logic (TOCL), 7(3):499–562, 2006.

Marco Maratea, Luca Pulina, and Francesco Ricca.
A multi-engine approach to answer-set programming.
Theory Pract. Log. Program., 14(6):841–868, 2014.

Victor W. Marek and Mirosław Truszczyński.

Stable Models and an Alternative Logic Programming Paradigm.
In Krzysztof R. Apt, V. Wiktor Marek, Mirosław Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

Ilkka Niemelä.

Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3–4):241–273, 1999.

Patrik Simons, Ilkka Niemelä, and Timo Soininen.

Extending and implementing the stable model semantics.
Artif. Intell., 138(1-2):181–234, 2002.

Tommi Syrjänen.

Omega-restricted logic programs.
In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2001.

Antonius Weinzierl.

Blending lazy-grounding and CDNL search for answer-set solving.
In Balduccini and Janhunen [BJ17], pages 191–204.