Answer Set Programming

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

15 November 2022
1. Introduction

2. ASP Syntax and Semantics

3. Knowledge Representation and Reasoning with ASP
Introduction
<table>
<thead>
<tr>
<th>Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>
Answer Set Programming (ASP)

- **Declarative** programming language for Knowledge **R**epresentation and **R**easoning [GL91, EGM97, MT99, Nie99, EFLP00, BET11]
- Logic paradigm based on rules
**Answer Set Programming (ASP)**

- **Declarative** programming language for **Knowledge Representation and Reasoning** [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 **Knowledge Representation and Reasoning** [GL91, EGM97, MT99, Nie99, EFLPoo, 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 **Knowledge Representation and Reasoning** [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

Computational Problem → Logic Program → ASP System → Solutions (Answer Sets)
The basic construct of ASP is the one of rule:

\[
\begin{align*}
\text{Head} & \quad : \quad \text{Body} \\
\text{disjunction} & \quad \text{conjunction}
\end{align*}
\]

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

\[
\begin{align*}
\text{parent("James Potter","Harry Potter").} \\
\text{son(X,Y) :- parent(Y,X).} \Rightarrow \text{son("Harry Potter","James Potter").}
\end{align*}
\]
ASP Syntax and Semantics
Core Syntax

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

\[ a_1 | \ldots | a_n := b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m. \]

  - head atoms
  - body literals

- A **literal** is either **positive** \( b \) or **negative** \( \text{not } b \) 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**

\[
\begin{align*}
\rightarrow & \text{saga( "Harry Potter" )} & \rightarrow & \text{numer_of_siblings( "Ron", 6 )} \\
& \text{string constant} & & \text{string numeric constant constant} \\
\rightarrow & \text{parent( Y, X )} & \rightarrow & \text{horcrux( ring )} \\
& \text{variables} & & \text{symbolic constant}
\end{align*}
\]
An **ASP logic program** is a (finite) set of **rules** of form:

\[ a_1 \mid \ldots \mid a_n \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m. \]

- 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**

```plaintext
% fact
saga("Harry Potter").

% (disjunctive) rule
like(X) | dislike(X) :- saga(X).

↓

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

Core Syntax

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

\[
\begin{align*}
    a_1 & \mid \ldots \mid a_n := b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m.
\end{align*}
\]

- 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 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 defined
  - 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 $\mathcal{P}$

\[
\text{like}_\text{herry}_\text{potter}(X) \mid \text{dislike}_\text{herry}_\text{potter}(X) : \neg \text{person}(X).
\text{person}("Pina"). \text{person}("Ugo").
\Rightarrow U_\mathcal{P} = \{"Pina", "Ugo"\}
\]
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").
⇒ $U_\mathcal{P}$ = {"Pina","Ugo"}
⇒ $B_\mathcal{P}$ = {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** $\text{ground}(\mathcal{P})$: 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").

⇓

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

⇓

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

↓

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)
...

E. Mastria

Answer Set Programming
ASP Semantics

**Reduct**

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

\[
\begin{align*}
a & : - d, \text{ not } b. \\
b & : - \text{ not } d. \\
d. 
\end{align*}
\]

$I = \{ a, d \}$

$\mathcal{P}^I$

\[
\begin{align*}
a & : - d, \text{ not } b. \\
b & : - \text{ not } d. \\
d. 
\end{align*}
\]

\[
\downarrow
\]

$I$ is a minimal model of $\mathcal{P}^I$ 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: \( \text{node}(1). \text{node}(2). \text{node}(3). \text{edge}(1, 2). \text{edge}(1, 3). \Rightarrow \text{Facts} \)

2: \( \text{in}S(X) \mid \text{out}S(X) :- \text{node}(X). \Rightarrow \text{Disjunctive Rule} \)

3: \( :- \text{edge}(X, Y), \text{not} \text{in}S(X), \text{not} \text{in}S(Y). \Rightarrow \text{Strong Constraint} \)
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: \textit{node}(1), \textit{node}(2), \textit{node}(3), \textit{edge}(1, 2), \textit{edge}(1, 3). \Rightarrow \textbf{Facts}
2: \textit{inS}(X) \mid \textit{outS}(X) := \textit{node}(X). \Rightarrow \textbf{Disjunctive Rule}
3: \neg \textit{edge}(X, Y), \neg \textit{inS}(X), \neg \textit{inS}(Y). \Rightarrow \textbf{Strong Constraint}

1: \textit{node}(1), \textit{node}(2), \textit{node}(3), \textit{edge}(1, 2), \textit{edge}(1, 3).
2: \textit{inS}(1) \mid \textit{outS}(1) := \textit{node}(1).
3: \textit{inS}(2) \mid \textit{outS}(2) := \textit{node}(2).
4: \textit{inS}(3) \mid \textit{outS}(3) := \textit{node}(3).
5: \neg \textit{edge}(1, 1), \neg \textit{inS}(1), \neg \textit{inS}(1).
6: \neg \textit{edge}(1, 2), \neg \textit{inS}(1), \neg \textit{inS}(2).
7: \neg \textit{edge}(1, 3), \neg \textit{inS}(1), \neg \textit{inS}(3).
8: \neg \textit{edge}(2, 1), \neg \textit{inS}(2), \neg \textit{inS}(1).
9: \neg \textit{edge}(2, 2), \neg \textit{inS}(2), \neg \textit{inS}(2).
10: \neg \textit{edge}(2, 3), \neg \textit{inS}(3), \neg \textit{inS}(3).
11: \neg \textit{edge}(3, 1), \neg \textit{inS}(3), \neg \textit{inS}(1).
12: \neg \textit{edge}(3, 2), \neg \textit{inS}(3), \neg \textit{inS}(2).
13: \neg \textit{edge}(3, 3), \neg \textit{inS}(3), \neg \textit{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) | outS(X) :- node(X). $\Rightarrow$ Disjunctive Rule
3: :- edge(X, Y), not inS(X), not inS(Y). $\Rightarrow$ Strong Constraint

E. Mastria
Answer Set Programming
**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. $inS(X) \mid outS(X) \leftarrow node(X)$. ⇒ **Disjunctive Rule**
3. $\leftarrow edge(X, Y), not\,inS(X), not\,inS(Y)$. ⇒ **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. $\leftarrow not\,inS(1), not\,inS(2)$.
6. $\leftarrow not\,inS(1), not\,inS(3)$.
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: \textit{node}(1). \textit{node}(2). \textit{node}(3). \textit{edge}(1, 2). \textit{edge}(1, 3). \Rightarrow \textbf{Facts}
2: \textit{inS}(X) \mid \textit{outS}(X) \Leftarrow \textit{node}(X). \Rightarrow \textbf{Disjunctive Rule}
3: \Leftarrow \textit{edge}(X, Y), \text{not inS}(X), \text{not inS}(Y). \Rightarrow \textbf{Strong Constraint}

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

\[ \begin{align*}
AS_1 &= Facts \cup \{inS(1), outS(2), outS(3)\} & S = \{1\} \\
AS_2 &= Facts \cup \{outS(1), inS(2), inS(3)\} & S = \{2, 3\} \\
AS_3 &= Facts \cup \{inS(1), inS(2), outS(3)\} & S = \{1, 2\} \\
AS_4 &= Facts \cup \{inS(1), outS(2), inS(3)\} & S = \{1, 3\} \\
AS_5 &= Facts \cup \{inS(1), inS(2), inS(3)\} & S = \{1, 2, 3\}
\end{align*} \]
Canonical approach to solve an ASP program $P$ over a set of facts $F$: 
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:
     \[ \text{ground}(P \cup F) \subseteq \text{ground}_{\text{Herbrand}}(P \cup F) \]
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:
     \[ \text{ground}(P \cup F) \subseteq \text{ground}_{\text{Herbrand}}(P \cup F) \]

2. **Solving** phase:
   - Generates the Answer Set(s) $\text{AS}(P \cup F)$

\[
\text{AS}(P \cup F) \equiv \text{AS} \left( \text{ground}(P \cup F) \right) \equiv \text{AS} \left( \text{ground}_{\text{Herbrand}}(P \cup F) \right)
\]
## Canonical approach: Ground & Solve

- Stand-alone grounders: LPARSE [Syr01], GRINGO [GKKS11], I-DLV [CFPZ17]
- Stand-alone solvers: CMODELS [GLMO6], SMODELS [SNS02], CLASP [GKS12], WASP [ADLR15]
- Monolithic systems: DLV [LPF⁺06, ACD⁺17], CLINGO [GKKS14]

## Other approaches

- Lazy Grounding: GASP [DDPR09], ASPERIX [LNO9, 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?
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.**

Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski.

**Answer set programming at a glance.**

Marcello Balduccini and Tomi Janhunen, editors.


Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Simona Perri, and Jessica Zangari.

**Efficiently coupling the I-DLV grounder with ASP solvers.**
References

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.**

Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari.

**T-DLV: The New Intelligent Grounder of DLV.**

Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors.


Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi.

**GASP: Answer Set Programming with Lazy Grounding.**

Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.

**Declarative Problem-Solving using the DLV System.**
Thomas Eiter, Georg Gottlob, and Heikki Mannila.

**Disjunctive Datalog.**


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

**Advances in gringo series 3.**


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

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


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

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


Michael Gelfond and Vladimir Lifschitz.

**Classical Negation in Logic Programs and Disjunctive Databases.**


Enrico Giunchiglia, Yulia Lierler, and Marco Maratea.

**Answer Set Programming Based on Propositional Satisfiability.**

Tomi Janhunen.

Some (in)translatability results for normal logic programs and propositional theories.


Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia.

**ASPeRiX, a first-order forward chaining approach for answer set computing.**


Claire Lefèvre and Pascal Nicolas.

The first version of a new asp solver: Asperix.


Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello.

**The dlv system for knowledge representation and reasoning.**


Marco Maratea, Luca Pulina, and Francesco Ricca.

A multi-engine approach to answer-set programming.

Victor W. Marek and Mirosław Truszczyński.

**Stable Models and an Alternative Logic Programming Paradigm.**

Ilkka Niemelä.

**Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.**

Patrik Simons, Ilkka Niemelä, and Timo Soininen.

**Extending and implementing the stable model semantics.**

Tommi Syrjänen.

**Omega-restricted logic programs.**

Antonius Weinzierl.

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