

Hannes Strass

(based on slides by Martin Gebser & Torsten Schaub (CC-BY 3.0))

Faculty of Computer Science, Institute of Artificial Intelligence, Computational Logic Group

ASP: Syntax and Semantics

Lecture 10, 15th Dec 2025 // Foundations of Logic Programming, WS 2025/26

Previously ...

- The immediate consequence operator T_P for a normal logic program P characterizes the **supported models** of P (= the models of $comp(P)$).
- The **stratification** of a program P partitions the program in layers (strata) such that predicates in one layer only **negatively/positively** depend on predicates in **strictly lower/lower or equal** layers.
- Every **stratified** logic program P has an intended **standard model** M_P .
- A program is **locally stratified** iff its ground instantiation is stratified.
- Locally stratified programs allow for a unique **perfect model**.
- A normal program P may have zero or more **well-supported models**.

Well-supported models are also known as *stable models*.

Logic Programming Semantics

LPs \ Model(s)	Least Herbrand	Standard	Perfect	Stable (Well-Supported)
Definite	defined, exists, unique			
Stratified		defined, exists, unique		
Locally Stratified			defined, exists, unique	
Normal				defined

Overview

Motivation: ASP vs. Prolog and SAT

ASP Syntax

Semantics

Variables

Motivation: ASP vs. Prolog and SAT

KR's Shift of Paradigm

Theorem Proving based approach (e.g. Prolog)

1. Provide a representation of the problem
2. A solution is given by a **derivation** of a query

Model Generation based approach (e.g. SATisfiability testing)

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

LP-style Playing with Blocks

Prolog program

```
on(a,b) . on(b,c) .
```

```
above(X,Y) :- on(X,Y) .
```

```
above(X,Y) :- on(X,Z) , above(Z,Y) .
```

Prolog queries (testing entailment)

```
?- above(a,c) .
```

yes

```
?- above(c,a) .
```

no

LP-style Playing with Blocks

Shuffled Prolog program

```
on(a,b). on(b,c).
```

```
above(X,Y) :- above(X,Z), on(Z,Y).
```

```
above(X,Y) :- on(X,Y).
```

Prolog queries (answered via SLD resolution)

```
?- above(a,c).
```

Fatal Error: local stack overflow.

KR's Shift of Paradigm

Theorem Proving based approach (e.g. Prolog)

1. Provide a representation of the problem
2. A solution is given by a **derivation** of a query

Model Generation based approach (e.g. SATisfiability testing)

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

SAT-style Playing with Blocks

Formula

$$\begin{aligned} & on(a, b) \\ \wedge & on(b, c) \\ \wedge & (on(X, Y) \rightarrow above(X, Y)) \\ \wedge & (on(X, Z) \wedge above(Z, Y) \rightarrow above(X, Y)) \end{aligned}$$

Herbrand model (among 426)

$$\left\{ \begin{array}{ccccc} on(a, b), & on(b, c), & on(a, c), & on(b, b), & \\ above(a, b), & above(b, c), & above(a, c), & above(b, b), & above(c, b) \end{array} \right\}$$

KR's Shift of Paradigm

Theorem Proving based approach (e.g. Prolog)

1. Provide a representation of the problem
2. A solution is given by a **derivation** of a query

Model Generation based approach (e.g. SATisfiability testing)

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

➡ **Answer Set Programming (ASP)**

ASP-style Playing with Blocks

Logic program

```
on(a,b) . on(b,c) .
```

```
above(X,Y) :- on(X,Y) .
```

```
above(X,Y) :- on(X,Z) , above(Z,Y) .
```

Logic program (shuffled)

```
on(a,b) . on(b,c) .
```

```
above(X,Y) :- above(Z,Y) , on(X,Z) .
```

```
above(X,Y) :- on(X,Y) .
```

Stable Herbrand model (and no others)

```
{ on(a,b) , on(b,c) , above(b,c) , above(a,b) , above(a,c) }
```

ASP versus LP

ASP	Prolog
Model generation	Entailment proving
Bottom-up	Top-down
Modelling language	Programming language
Rule-based format	
Instantiation Flat terms	Unification Nested terms
(Turing +) $NP^{(NP)}$	Turing

ASP versus SAT

ASP	SAT
Model generation	
Bottom-up	
Constructive Logic	Classical Logic
Closed (and open) world reasoning	Open world reasoning
Modelling language	—
Complex reasoning modes	Satisfiability testing
Satisfiability	Satisfiability
Enumeration/Projection	—
Intersection/Union	—
Optimization	—
(Turing +) $NP^{(NP)}$	NP

What Is ASP Good For?

- Combinatorial search problems in the realm of P , NP , and NP^{NP} (some with substantial amount of data), like
 - Automated Planning
 - Code Optimization
 - Composition of Renaissance Music
 - Database Integration
 - Decision Support for NASA shuttle controllers
 - Model Checking
 - Product Configuration
 - Robotics
 - Systems Biology
 - System Synthesis
 - (industrial) Team-building
 - and many many more

ASP Syntax

Normal Logic Programs

Definition

Let \mathcal{A} be a set of atoms.

- A (normal) **rule** – denoted r – is of the form

$$a_0 \leftarrow a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n$$

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$.

- A (normal) **logic program** (over \mathcal{A}) – denoted P – is a finite set of rules.
- A program P is **positive** (definite) $:\iff m = n$ for all $r \in P$.

$$head(r) = a_0$$

$$body(r) = \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\}$$

$$body(r)^+ = \{a_1, \dots, a_m\}$$

$$body(r)^- = \{a_{m+1}, \dots, a_n\}$$

$$atom(P) = \bigcup_{r \in P} \left(\{head(r)\} \cup body(r)^+ \cup body(r)^- \right)$$

$$body(P) = \{body(r) \mid r \in P\}$$

Rough Notational Convention

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

	true, false	if	and	or	iff	default negation	classical negation
source code		$:-$	$,$	$ $		not	$-$
logic program		\leftarrow	$,$	$;$		\sim	\neg
formula	\top, \perp	\rightarrow	\wedge	\vee	\leftrightarrow	\sim	\neg

Semantics

Formal Definition

Stable Models of Positive Programs

Definition

- A set of atoms X is **closed under** a positive program P
: \iff for any $r \in P$, we have that $body(r)^+ \subseteq X$ implies $head(r) \in X$.
 - X corresponds to a model of P (seen as a formula)
- The **smallest** set of atoms that is closed under a positive program P is denoted by $Cn(P)$.
 - $Cn(P)$ corresponds to the \subseteq -smallest model of P
- The set $Cn(P)$ of atoms is the **stable model** of a *positive* program P .

$Cn(P)$ is the \subseteq -least fixpoint of the one-step consequence operator T_P .

Proposition

If P_1 and P_2 are positive programs with $P_1 \subseteq P_2$, then $Cn(P_1) \subseteq Cn(P_2)$.

Proof Idea: Every model of P_2 is a model of P_1 , thus satisfies all $a \in Cn(P_1)$.

Basic Idea

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

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

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

Formula ϕ has one stable model, often called **answer set**:

p	\mapsto	1
q	$\{p, q\}$	1
r	\mapsto	0

$$P_\phi \quad \begin{array}{l} q \leftarrow \\ p \leftarrow q, \sim r \end{array}$$

Informally, a set X of atoms is a **stable model** of a logic program P

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

“Justified” here means **well-founded support**.

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

Formal Definition

Stable Models of Normal Programs

Definition

Let P be a normal logic program and X be a set of atoms.

1. The (Gelfond-Lifschitz-)**reduct** of P relative to X is the positive program

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

2. A set X of atoms is a **stable model** of a program P $:\Leftrightarrow Cn(P^X) = X$.

- Note: $Cn(P^X)$ is the \subseteq -smallest (classical) model of P^X
- Note: Every atom in X is justified by an “*applying rule from P* ”

Intuitively:

- We evaluate all **negative** body atoms by X , in particular
- we **assume** all atoms $a \notin X$ to be **false**, and then
- **derive** what must be **true** under this assumption.
- If this allows us to **reconstruct** X , then X is **stable**.

A Closer Look at P^X

In other words, given a set X of atoms from P ,

P^X is obtained from P by **deleting**

1. each **rule** having $\sim a$ in its body with $a \in X$ and then
2. all **negative atoms** of the form $\sim a$ in the bodies of the remaining rules.

Note: Only **negative body literals** are evaluated w.r.t. X .

Proposition

If $X \subseteq Y$, then $P^Y \subseteq P^X$.

Proof.

- Let $r \in P^Y$. Then there exists a rule $r' \in P$ such that $r = \text{head}(r') \leftarrow \text{body}(r')^+$ and $\text{body}(r')^- \cap Y = \emptyset$.
- Due to $X \subseteq Y$ we have $\text{body}(r')^- \cap X = \emptyset$ and thus $r \in P^X$. □

A First Example

$$P = \{p \leftarrow p, q \leftarrow \sim p\}$$

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

A Second Example

$$P = \{p \leftarrow \sim q, q \leftarrow \sim p\}$$

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

A Third Example

$$P = \{p \leftarrow \sim p\}$$

\mathcal{X}	$P^{\mathcal{X}}$	$Cn(P^{\mathcal{X}})$
$\{ \}$	$p \leftarrow$	$\{p\}$ x
$\{p\}$		\emptyset x

Quiz: Stable Models

Quiz

Consider the following normal logic program P : ...

Some Properties

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

Proposition

1. If X is a stable model of a logic program P , then X is a model of P (seen as a formula).
2. If X and Y are distinct stable models of a logic program P , then $X \not\subseteq Y$.

Proof.

1.
 - P^X evaluates P w.r.t. all $a \in \mathcal{A} \setminus X$.
 - $X = \text{Cn}(P^X)$ is a model of P^X .
 - Thus evaluating P by X leads to true.
2.
 - Let X and Y be stable models of P and assume $X \subseteq Y$.
 - Then $P^Y \subseteq P^X$ and $\text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$.
 - Thus $Y = \text{Cn}(P^Y) \subseteq \text{Cn}(P^X) = X$.



Variables

Programs with Variables

Definition

Let P be a logic program with **first-order** atoms (built from predicates over terms, where terms are built from constant/function symbols and variables).

- Let \mathcal{T} be a set of variable-free **terms**. (also called **Herbrand universe**).
- Let \mathcal{A} be a set of (variable-free) **atoms** constructable from \mathcal{T} . (also called **Herbrand base**).
- For a rule $r \in P$ (with variables), the **ground instances** of r are the variable-free rules obtained by replacing all variables in r by elements from \mathcal{T} :

$$\text{ground}(r) := \{r\theta \mid \theta : \text{var}(r) \rightarrow \mathcal{T} \text{ and } \text{var}(r\theta) = \emptyset\}$$

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

- The **ground instantiation** of P is $\text{ground}(P) := \bigcup_{r \in P} \text{ground}(r)$.

An Example

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

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a, a), r(a, b), r(a, c), r(b, a), r(b, b), r(b, c), r(c, a), r(c, b), r(c, c), \\ t(a, a), t(a, b), t(a, c), t(b, a), t(b, b), t(b, c), t(c, a), t(c, b), t(c, c) \end{array} \right\}$$

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

Intelligent Grounding aims at reducing the ground instantiation.

Stable Models of Programs with Variables

Definition

Let P be a normal logic program with variables.

A set X of **ground** atoms is a **stable model** of P

$:\Leftrightarrow$

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

Example

The normal first-order program $P = \{ \text{even}(0) \leftarrow, \text{even}(s(X)) \leftarrow \sim \text{even}(X) \}$ has the single stable model

$$S = \{ \text{even}(0), \text{even}(s(s(0))), \text{even}(s(s(s(s(0))))), \dots \}$$

since the reduct $\text{ground}(P)^S$ is given by $\{ \text{even}(0) \leftarrow, \text{even}(s(s(0))) \leftarrow, \dots \}$.

Well-Supported Models = Stable Models

Theorem (Fages, 1991)

For any normal (first-order) logic program P , its well-supported models coincide with its stable models.

Proof Ideas.

- For X a stable model of P , define $A \prec_X B :\iff$ for some $i \in \mathbb{N}$, $A \in T_{pX} \uparrow i$ and $B \in T_{pX} \uparrow (i+1) \setminus T_{pX} \uparrow i$. Show that X is well-supported via \prec_X .
- For M a well-supported model of P via \prec , show by induction that for any atom $A \in M$, there is $i \in \mathbb{N}$ with $A \in T_{pM} \uparrow i$. For this, employ that \prec is well-founded and use the cardinality of the set $\{B \mid B \prec A\}$. □

Recall: A Herbrand interpretation $I \subseteq \mathcal{A}$ is **well-supported** $:\iff$ there is a well-founded ordering \prec on \mathcal{A} such that:

for each $A \in I$ there is a clause $A \leftarrow \vec{B} \in \text{ground}(P)$ with:

$I \models \vec{B}$, and for every positive atom $C \in \vec{B}$, we have $C \prec A$.

Conclusion

Summary

- PROLOG-based logic programming focuses on **theorem proving**.
- LP based on stable model semantics focuses on **model generation**.
- The **stable model** of a positive program is its least (Herbrand) model.
- The **stable models** of a normal logic program P are those sets X for which X is the stable model of the positive program P^X (the reduct).
- The **well-supported** model semantics equals **stable** model semantics.

Suggested action points:

- Download the solver `clingo` and try out the examples of this lecture.
- Clarify: How do stable models have justified support for true atoms?
- Show that every stable model X of a program P satisfies $X \subseteq Cn(P^\emptyset)$.