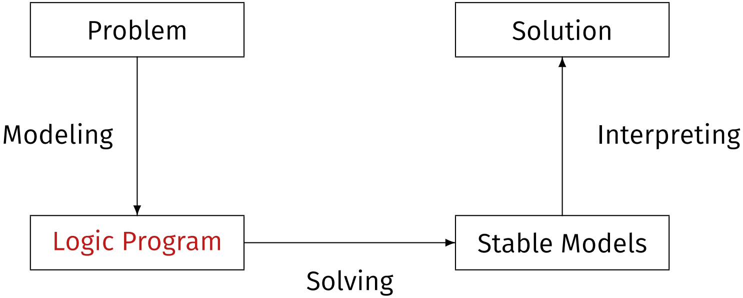
Programming Paradigms: Logic Programming

Syntax and Semantics

Víctor Gutiérrez Basulto

07.10.2019

Problem solving in ASP: Syntax



Syntax

- Assume a vocabulary Σ compromised of nonempty finite sets of
  - constants (e.g. *frankfurt*)
  - variables (e.g. X)
  - predicate symbols (e.g. *connected*)
- A term is either a variable or a constant

- An atom is an expression of the form $p(t_1, \ldots, t_n)$ where
  - p is a predicate of arity $n \geq 0$ from Σ, and
  - $t_1, \ldots t_n$ are terms

  (e.g. *connected(frankfurt)*)

- A term or an atom is ground if it contains no variable

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, 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 $0 \leq i \leq n$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, 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 $0 \leq i \leq n$

- If $n = 0$, the rule is a fact (written shortly a_0).

- Notation

$$\begin{array}{rcl} head(r) & = & a_0 \\ body(r) & = & \{a_1, \dots, a_m, \sim a_{m+1}, \dots, \sim a_n\} \\ body(r)^+ & = & \{a_1, \dots, a_m\} \\ body(r)^- & = & \{a_{m+1}, \dots, a_n\} \\ atom(P) & = & \bigcup_{r \in P} \left(\{head(r)\} \cup body(r)^+ \cup body(r)^-\right) \\ body(P) & = & \{body(r) \mid r \in P\} \end{array}$$

- A program P is positive if $body(r)^- = \emptyset$ for all $r \in P$

Example

- Ground rule: *"If Frankfurt is a hub airport, and there is a link between Frankfurt and Hamburg, then Hamburg is a connected airport"*

$$connected(hamburg) \leftarrow hub\_airport(frankfurt), link(frankfurt, hamburg)$$

Example

- Ground rule: *"If Frankfurt is a hub airport, and there is a link between Frankfurt and Hamburg, then Hamburg is a connected airport"*

$$connected(hamburg) \leftarrow hub\_airport(frankfurt),\ link(frankfurt,\ hamburg)$$

- Non-Ground rule: *"All airports with a link to a hub airport are connected"*

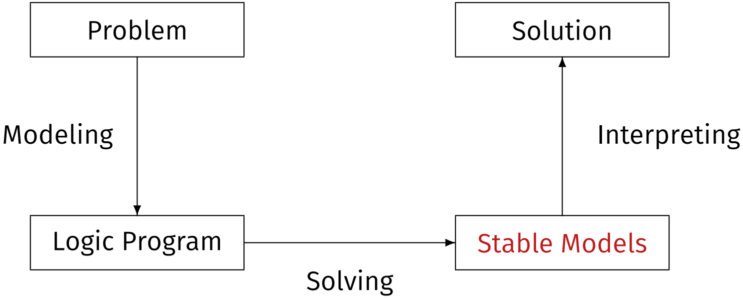
$$connected(X) \leftarrow hub\_airport(Y),\ link(X,Y)$$

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
source code		`:-`	`,`	`;`		`not`
logic program		$\leftarrow$	$,$	$;$		$\sim$

Problem solving in ASP: Semantics



Semantics: Positive Programs

- Semantics of a program P is given in terms of sets of ground atoms, which can be formed using predicates and terms in P. These sets are called interpretations.
  - Intuitively this set denotes which ground atoms are true in a given scenario

Semantics: Positive Programs

- Semantics of a program P is given in terms of sets of ground atoms, which can be formed using predicates and terms in P. These sets are called interpretations.
    - Intuitively this set denotes which ground atoms are true in a given scenario
- Which interpretations capture the intended meaning?
    - Those that "respect" the rules. These sets are called models.

Semantics: Positive Programs

- Semantics of a program P is given in terms of sets of ground atoms, which can be formed using predicates and terms in P. These sets are called interpretations.
  - Intuitively this set denotes which ground atoms are true in a given scenario
- Which interpretations capture the intended meaning?
  - Those that “respect” the rules. These sets are called models.
- Do we want all models?
  - No, we want the smallest models, that is, the truth of an atoms should be “justified” by a rule.

Example

Given:

$$P_1 = \{a \leftarrow b. \quad b \leftarrow c. \quad c\},$$

truth of a in the model $I = \{a, b, c\}$ is "founded".

Given:

$$P_2 = \{a \leftarrow b. \quad b \leftarrow a. \quad c\},$$

truth of a in the model $I = \{a, b, c\}$ is not founded.

Formal Def. Semantics

Positive Programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
- The smallest (least) set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P (ditto)
- The set $Cn(P)$ of atoms is the stable model of a *positive* program P
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of rules corresponding to P

Example

- For $P_1 = \{\ a \leftarrow b. \quad b \leftarrow c. \quad c\ \}$, we have $LM(P_1) = \{a, b, c\}$.
- For $P_2 = \{\ a \leftarrow b. \quad b \leftarrow a. \quad c\ \}$, we have $LM(P_2) = \{c\}$.
- For P from above,

$$\begin{aligned} p(X,Y,Z) &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r). \\ h(X,Z') &\leftarrow p(X,Y,Z'), h(X,Y), t(Z,Z',r). \\ p(0,0,b). \qquad & h(0,0). \qquad t(a,b,r). \end{aligned}$$

we have

$$LM(P) = \{h(0,0), t(a,b,r), p(0,0,b), p(0,0,a), h(0,b)\}.$$

Negation in Logic Programs

- Why Negation?
  - Natural linguistic concept
  - Facilitates convenient, declarative descriptions (definitions).

    "Men who are not husbands are single".

Negation in Logic Programs

- Why Negation?
  - Natural linguistic concept
  - Facilitates convenient, declarative descriptions (definitions).

    "Men who are not husbands are single".

- A rule, r, is of the form

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

- A rule, r, is of the form

$$a_0 \mathtt{:-} a_1, \ldots, a_m, \mathtt{not}\ a_{m+1}, \ldots, \mathtt{not}\ a_n$$

Negation in Logic Programs

- `not` a means "negation as failure (to prove) a"
- Close World Assumption (CWA): whatever cannot be derived is false.

  Different from classical negation in First-order logic

Image: http://

```
cross :- -train.
```

Evidence of train not coming, therefore you can cross

At a rail road crossing cross the road if **no train** approaches

```
cross :- not train.
```

By default you can cross, **unless** you have evidence that a train is coming

At a rail road crossing cross the road if **no train is known** to approach

Example

$$man(dilbert).$$
$$single(X) \leftarrow man(X), not\ husband(X).$$

- Can not prove $husband(dilbert)$ from rules.
- Single intended minimal model: $\{man(dilbert), single(dilbert)\}$.

Example

$$
\begin{aligned}
man(dilbert). \\
single(X) \leftarrow man(X), not\ husband(X). \\
husband(X) \leftarrow man(X), not\ single(X).
\end{aligned}
$$

Semantics???

Problem: not a single intuitive model!

Example

$$man(dilbert).$$
$$single(X) \leftarrow man(X), not\ husband(X).$$
$$husband(X) \leftarrow man(X), not\ single(X).$$

Semantics???

Problem: not a single intuitive model!

Two intuitive

$M_1 = \{man(dilbert), single(dilbert)\}$, and

$M_2 = \{man(dilbert), husband(dilbert)\}$.

Which one to choose?

Stable Models

- We adopt the Stable Model (alias Answer Set) Semantics by Gelfond and Lifschitz (1990)

  Alternative (“justified”) Models:

$$
\begin{aligned}
M_1 &= \{man(dilbert), single(dilbert)\} \\
M_2 &= \{man(dilbert), husband(dilbert)\}
\end{aligned}
$$

- Other kind of semantics exist, e.g. well-founded semantics, considering a partial models

Example

Consider program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

- Consider $M' = \{man(dilbert)\}$.
  - Assuming that $man(dilbert)$ is true and $husband(dilbert)$ is false, by r_1 also $single(dilbert)$ should be true.
  - M' does not represent a coherent or “stable” view of the information given by P_1.

Example

Consider program P_1:

$$man(dilbert). \qquad (f_1)$$

$$single(dilbert) \leftarrow man(dilbert), not\ husband(dilbert). \qquad (r_1)$$

$$husband(dilbert) \leftarrow man(dilbert), not\ single(dilbert). \qquad (r_2)$$

- Consider $M' = \{man(dilbert)\}$.
  - Assuming that $man(dilbert)$ is true and $husband(dilbert)$ is false, by r_1 also $single(dilbert)$ should be true.
  - M' does not represent a coherent or "stable" view of the information given by P_1.
- Consider $M'' = \{man(dilbert), single(dilbert), husband(dilbert)\}$.
  - The bodies of r_1 and r_2 are not true w.r.t. M'', hence there is no evidence for $single(dilbert)$ and $husband(dilbert)$ being true.
  - M'' is not "stable" either.

Formal Definition

Stable models of normal programs

- The reduct, P^X, of a program P relative to a set X of atoms (interpretation) is defined by

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

- A set X of atoms is a stable model of a program P, if $Cn(P^X) = X$

- Remarks
  - $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of P^X
  - Each atom in X is justified by an *"applying rule from P"*
  - Set X is stable under *"applying rules from P"*

A pragmatic look at P^X

- Alternatively, 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

An Example

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

An Example

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

X		
$\{\ \ \}$		
$\{p\ \}$		
$\{\ q\}$		
$\{p, q\}$		

An Example

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

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

An Example

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

X	P^X	$Cn(P^X)$	$Cn(P^X) = X?$
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

An Example

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

X	P^X	$Cn(P^X)$	$Cn(P^X) = X?$
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

An Example

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

X	P^X	$Cn(P^X)$	$Cn(P^X) = X?$
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

An Example

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

X	P^X	$Cn(P^X)$	$Cn(P^X) = X?$
$\{\ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\}$	$p \leftarrow p$	$\emptyset$	✘
$\{q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

Some properties

- A logic program may have **zero, one, or multiple** stable models
- If X is a stable model of a logic program P,
  then X is a model of P (seen as a formula)
- If X and Y are stable models of a *normal* program P,
  then $X \not\subset Y$

Programs with variables

Programs with variables

Let P be a logic program

- Let $\mathcal{T}$ a set of variable-free terms in P (also called Herbrand universe)
- Let $\mathcal{A}$ be a set of all variable-free atoms constructible from $\mathcal{T}$ (also called alphabet or Herbrand base)

Programs with variables

Let P be a logic program

- Let $\mathcal{T}$ a set of (variable-free) terms in P
- Let $\mathcal{A}$ be a set of all variable-free atoms constructible from $\mathcal{T}$
- Ground Instances of $r \in P$: Set of variable-free rules obtained by replacing *all* variables in r by elements from $\mathcal{T}$:

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

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

Programs with variables

Let P be a logic program

- Let $\mathcal{T}$ a set of (variable-free) terms in P
- Let $\mathcal{A}$ be a set of all variable-free atoms constructible from $\mathcal{T}$
- Ground Instances of $r \in P$: Set of variable-free rules obtained by replacing *all* variables in r by elements from $\mathcal{T}$:

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

  where $var(r)$ stands for the set of all variables occurring in r; θ is a (ground) substitution
- Ground Instantiation of P: $\quad ground(P) = \bigcup_{r \in P} ground(r)$

An example

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

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

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

An example

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

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

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

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

An example

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

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

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

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

- Intelligent Grounding aims at reducing the ground instantiation

Safety

- 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

$$
\begin{aligned}
&d(a)\\
&d(c)\\
&d(d)\\
&p(a,b)\\
&p(b,c)\\
&p(c,d)\\
&p(X,Z) \leftarrow p(X,Y), p(Y,Z)\\
&q(a)\\
&q(b)\\
&q(X) \leftarrow {\sim}r(X), d(X)\\
&r(X) \leftarrow {\sim}q(X), d(X)\\
&s(X) \leftarrow {\sim}r(X), p(X,Y), q(Y)
\end{aligned}
$$

Example

Safe ?

$$
\begin{array}{l}
d(a) \\
d(c) \\
d(d) \\
p(a,b) \\
p(b,c) \\
p(c,d) \\
p(X,Z) \leftarrow p(X,Y), p(Y,Z) \\
q(a) \\
q(b) \\
q(X) \leftarrow \sim r(X), d(X) \\
r(X) \leftarrow \sim q(X), d(X) \\
s(X) \leftarrow \sim r(X), p(X,Y), q(Y)
\end{array}
$$

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Example

	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a, b)$	✔
$p(b, c)$	✔
$p(c, d)$	✔
$p(X, Z) \leftarrow p(X, Y), p(Y, Z)$	
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	
$r(X) \leftarrow \sim q(X), d(X)$	
$s(X) \leftarrow \sim r(X), p(X, Y), q(Y)$	

Example

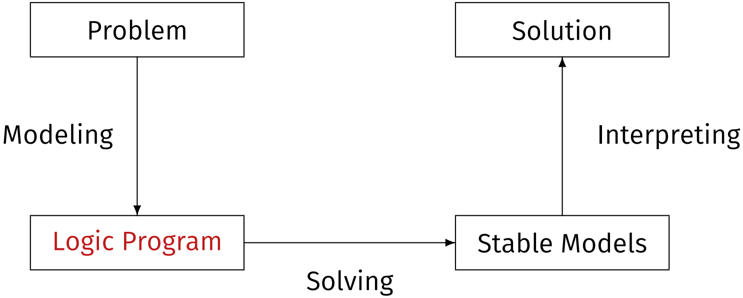
	Safe ?
$d(a)$	✔
$d(c)$	✔
$d(d)$	✔
$p(a,b)$	✔
$p(b,c)$	✔
$p(c,d)$	✔
$p(X,Z) \leftarrow p(X,Y), p(Y,Z)$	✔
$q(a)$	✔
$q(b)$	✔
$q(X) \leftarrow \sim r(X), d(X)$	✔
$r(X) \leftarrow \sim q(X), d(X)$	✔
$s(X) \leftarrow \sim r(X), p(X,Y), q(Y)$	✔

Stable models of programs with Variables

Let P be a normal logic program with variables

- A set X of (ground) atoms is a stable model of P, if $Cn(ground(P)^X) = X$

Problem solving in ASP: Extended Syntax



Language constructs

- Facts
  - `p.`
  - `p(a;b;c).` short hand for `p(a).` `p(b).` `p(c).`
  - `p(1 .. 10).` short hand for `p(1).` `p(2).` `p(3).` `etc.`
- Variables `p(X) :- q(X)`

Language constructs

- Intervals `M .. N`
  - `grid(1..S,1..S):- size(S).`
  - `grid(X,Y):- X=1..S,Y=1..S, X-Y!=0,X+Y-1!=S.`
- Conditional literals `p(X) :- q(X) : r(X)`
  - `meet:- available(X) : person(X)`
  - `on(X) : day(X):-meet.`

Language constructs

- Choice (Cardinality Constraints)
  - `1{p(1 .. 3)}2.`
  - `1 { has_property(X,C) : property(C)} 1 :- item(X).`
- Aggregates
  - `20 <= # sum {4 : course (db) ; 6 course(ai); 8 : course(project); 3 : course(xml)}`
  - `#sum { 3 : bananas; 25 : cigars; 10 : broom } <= 30`
  - `within_budget :- # sum 10 {Amount : paid(Amount)} 100.`

Language constructs

- Aggregates `#count{ ···}`
  - `many_neighbors(X):-vertex(X), #count{Y : adjacent (X,Y)} >3.`
- Disjunction `p(X) ; q(X) :- r(X)`
  - if `r(X)` then `p(X)` **or** `q(X)`
- Integrity constraints `:- q(X), p(X)`
  - `:- in_clique(X), in_clique(Y), not edge(X,Y).`
  - this constraint says: it **cannot** be the case that nodes X and Y are in a clique, *and* there is no edge between X and Y.

Language constructs

- Multi-objective oOptimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

    `#maximize { 1,X:in_clique(X), node(X) }.`

- `noisy :- hotel(X), main_street(X).`
- `#maximize { Y@1,X : hotel(X), star(X,Y) }.`
- `#minimize { Y / Z@2,X : hotel(X), cost(X,Y), star(X,Z) }.`
- `noisy.  [ 1@3 ]`

Language constructs

- Multi-objective oOptimization
  - Weak constraints `:~ q(X), p(X,C) [C@42]`
  - Statements `#minimize { C@42 : q(X), p(X,C) }`

`#maximize { 1,X:in_clique(X), node(X) }.`

- `noisy :- hotel(X), main_street(X).`
- `#maximize { Y@1,X : hotel(X), star(X,Y) }.`
- `#minimize { Y / Z@2,X : hotel(X), cost(X,Y), star(X,Z) }.`
- `noisy. [ 1@3 ]`

Language constructs

- Arithmetic Functions
  - $+$ (addition), $-$ (subtraction), $*$ (multiplication), $/$ (integer division), $\backslash$ (modulo), $**$ (exponentiation), $|\cdot|$ (absolute value).
- Comparison Predicates
  - $=$ (equal), $!=$ (not equal), $<$ (less than), $<=$ (less than or equal), $>$ (greater than), and $>=$ (greater than or equal).

Mandatory reading: Chapter 2 of of Answer Set Solving in Practice, by Gebser, Kaminski, Kaufmann, Schaub