# La programmazione logica come metodologia per la codifica e la risoluzione di rompicapi

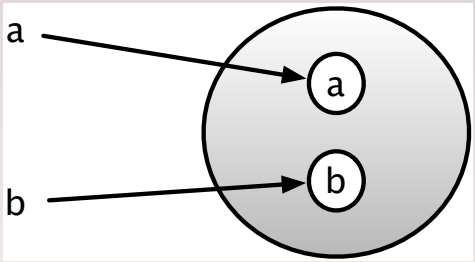
Agostino Dovier

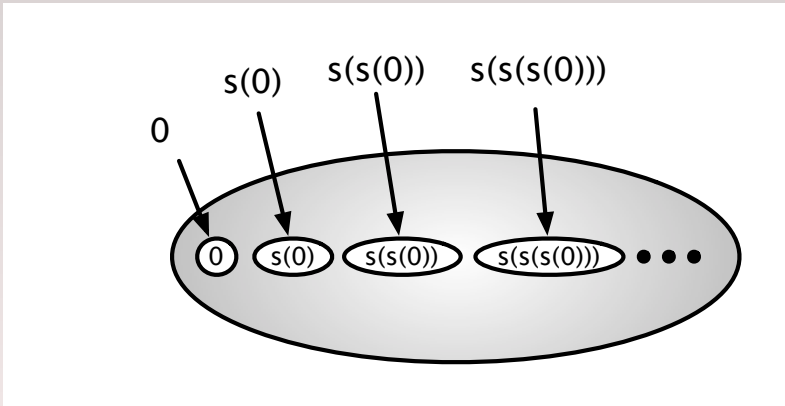
Dipartimento di Matematica e Informatica
Università di Udine

20 Novembre 2015

# Herbrand Interpretations

Let us consider the set of all ground terms that can be built with constant and function symbols in a program $P$.
This set can be used as a Universe for interpretations (the Herbrand Universe or $H_P$).
Ground terms are interpreted as themselves

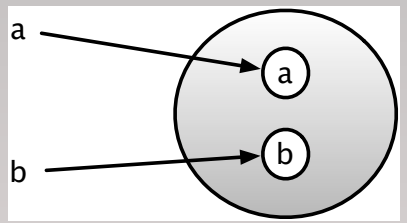


# Herbrand Models

Interpretations on the Herbrand Universe can be (or not) models (Herbrand models)

```
p(a).
q(b).
r(X) ← p(X).
```

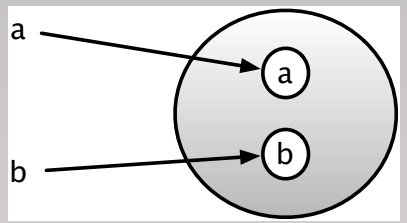



Now, $\overline{a} = a$ and $\overline{b} = b$. Let use denote with $P, Q, R$ the interpretations of the predicate symbols $p, q, r$.

# Herbrand Models

Interpretations on the Herbrand Universe can be (or not) models (Herbrand models)

```
p(a).
q(b).
r(X) ← p(X).
```



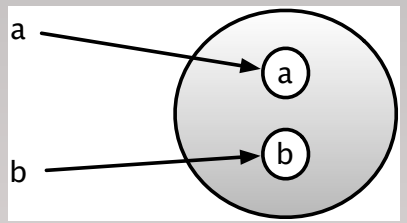

Now, $\overline{a} = a$ and $\overline{b} = b$. Let use denote with $P, Q, R$ the interpretations of the predicate symbols $p, q, r$.

1. $P = \{\overline{a}\}, Q = \{\overline{b}\}, R = \{\overline{a}, \overline{b}\}$ is a model.
2. $P = \{\overline{a}, \overline{b}\}, Q = \{\overline{b}\}, R = \{\overline{a}\}$ is NOT a model.

# Herbrand Models

Interpretations on the Herbrand Universe can be (or not) models (Herbrand models)

```
p(a).
q(b).
r(X) ← p(X).
```




Now, $\overline{a} = a$ and $\overline{b} = b$. Let use denote with $P, Q, R$ the interpretations of the predicate symbols $p, q, r$.

1. $P = \{\overline{a}\}, Q = \{\overline{b}\}, R = \{\overline{a}, \overline{b}\}$ is a model.
2. $P = \{\overline{a}, \overline{b}\}, Q = \{\overline{b}\}, R = \{\overline{a}\}$ is NOT a model.

Herbrand interpretations and models can be represented uniquely by set of atoms:

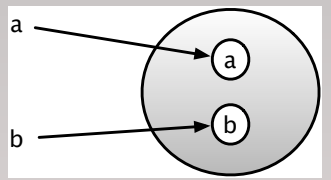
1. $\{p(a), q(b), r(a), r(b)\}$ (model)
2. $\{p(a), p(b), q(b), r(a)\}$ (not a model)

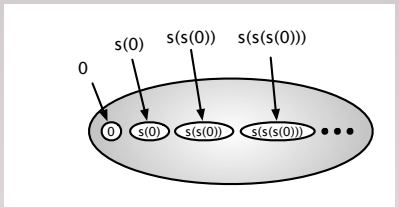
# A lattice of interpretations

- Given a program $P$, the corresponding Herbrand Universe $H_P$ is determined uniquely

```
p(a).        q(b).
r(X) ← p(X).

nat(0).
nat(s(X)) ← nat(X).
```




- Let $B_P = \{p(t_1, \ldots, t_n) : p$ is a predicate symbol in $P$, $t_i$s are ground terms made with constant and function symbols in $P\}$ $B_P$ is called the Herbrand base.
- Any subset of $B_P$ uniquely determines an Herbrand Interpretation (some of them can be models)
- $(\wp(B_P), \subseteq)$ forms a complete lattice

# The fundamental Theorem

A *clause* is a formula of the form $\forall \vec{X}(A_0 \vee \cdots \vee A_n)$ where $A_i$s are positive or negative literals built on the variables $\vec{X}$.
Observe that $A_0 \vee \neg A_1 \vee \cdots \vee \neg A_n$ is $A_0 \leftarrow A_1 \wedge \cdots \wedge A_n$.
The notions given for "programs" in the previous slides apply to conjunction of clauses as well.
If $T$ is a conjunction of clauses: $H_T$ denotes the Herbrand Universe and $B_T$ the Herbrand Base.

# The fundamental Theorem

A *clause* is a formula of the form $\forall \vec{X}(A_0 \vee \cdots \vee A_n)$ where $A_i$s are positive or negative literals built on the variables $\vec{X}$.
Observe that $A_0 \vee \neg A_1 \vee \cdots \vee \neg A_n$ is $A_0 \leftarrow A_1 \wedge \cdots \wedge A_n$.
The notions given for "programs" in the previous slides apply to conjunction of clauses as well.
If $T$ is a conjunction of clauses: $H_T$ denotes the Herbrand Universe and $B_T$ the Herbrand Base.

## Theorem

*Let $T$ be a conjunction of clauses. Then $T$ has a model if and only if $T$ has an Herbrand model.*

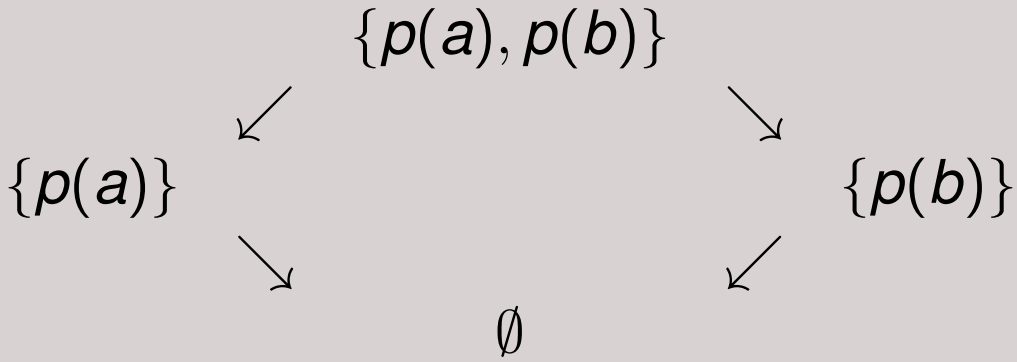
## Corollary

*Let $T$ be a conjunction of clauses and $A \in B_T$ be a ground atom. Then $T \models A$ if and only if $A$ is true in all Herbrand models of $A$.*

# Non Horn Clauses

## Example

Let $T = p(a) \vee p(b)$. There are 4 Herbrand interpretations:

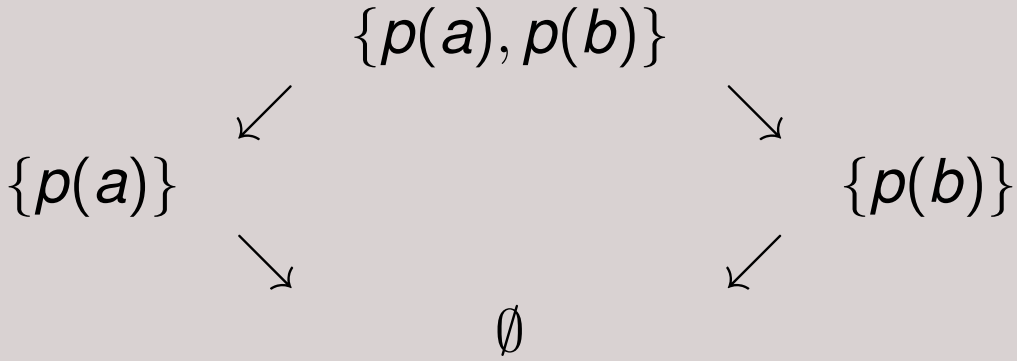



3 of them are models. There is no $A$ such that $T \models A$.

# Non Horn Clauses

## Example

Let $T = p(a) \vee p(b)$. There are 4 Herbrand interpretations:




3 of them are models. There is no $A$ such that $T \models A$.

## Example

Let $T$ be
$(p(a) \vee p(b)) \wedge (\neg p(a) \vee p(b)) \wedge (p(a) \vee \neg p(b)) \wedge (\neg p(a) \vee \neg p(b))$.
Same 4 interpretations as above. No one of them is a model.

# The fundamental Theorem (2)

Definite clauses have exactly one positive literals. The rule:

$$p(A) \leftarrow q(A, B), r(B).$$

is the clause

$$\forall A \forall B \, (p(A) \vee \neg q(A, B) \vee \neg r(B))$$

Programs are conjunctions of definite clauses.

# The fundamental Theorem (2)

Definite clauses have exactly one positive literals. The rule:

$$p(A) \leftarrow q(A, B), r(B).$$

is the clause

$$\forall A \forall B \, (p(A) \vee \neg q(A, B) \vee \neg r(B))$$

Programs are conjunctions of definite clauses.

**Theorem**

*Let $P$ be a (definite clause) program. Then $P$ admits a (unique) minimum Herbrand model $M_P$ ($M_P$ is the semantics of $P$). (i.e., if $I$ is a Herbrand model of $P$, then $M_P \subseteq I$).*

**Corollary**

*Let $P$ be a (definite clause) program and $A \in B_T$ be a ground atom. Then $T \models A$ if and only if $A \in M_P$.*

# Computing $M_P$

1. Top-Down: using SLD resolution:
   Query the SLD interpreter with the goal $:- A$
2. Bottom-Up: using the $T_P$ (immediate consequence) operator.

$$T_P(I) = \{a : a \leftarrow b_1, \ldots, b_n \in \text{ground}(P), \{b_1, \ldots, b_n\} \subseteq I\}$$

# Computing $M_P$

## Example

Let $P$ be the program:

```
r(a).
r(b).
p(a).
q(X) :- r(X),p(X).
```

Then:

$$\begin{aligned} T_P(\emptyset) &= \{r(a), r(b), p(a)\} \\ T_P(\{r(a), r(b), p(a)\}) &= \{q(a), r(a), r(b), p(a)\} \\ T_P(\{q(a), r(a), r(b), p(a)\}) &= \{q(a), r(a), r(b), p(a)\} \quad \Leftarrow \text{ Fixpoint!} \end{aligned}$$

# Computing $M_P$

## Example

Let $P$ be the program:

```
nat(0).
nat(s(X)) :- nat(X).
```

Then:

$$
\begin{array}{rcl}
T_P(\emptyset) & = & \{nat(0)\} \\
T_P(\{nat(0)\}) & = & \{nat(0), nat(s(0))\} \\
T_P(\{nat(0), nat(s(0))\}) & = & \{nat(0), nat(s(0)), nat(s(s(0)))\} \\
\vdots & \vdots & \vdots \\
T_P(\{nat(0), nat(s(0)), nat(s(s(0))), \ldots\}) & = & \{nat(0), nat(s(0)), nat(s(s(0))), \ldots\} \\
 & \Uparrow & \text{Fixpoint!}
\end{array}
$$

# Computing $M_P$

$T_P$ is monotone: $I \subseteq J \rightarrow T_P(I) \subseteq T_P(J)$ and
upward continuous: if $I_0 \subseteq I_1 \subseteq I_2 \cdots$ then $T_P(\bigcup_{i \geq 0} I_i) = \bigcup_{i \geq 0} T_P(I_i)$

Let us define

$$\begin{aligned} T_P \uparrow 0 &= \emptyset \\ T_P \uparrow n+1 &= T_P(T_P \uparrow n) \\ T_P \uparrow \omega &= \bigcup_{n \geq 0} T_P \uparrow n \end{aligned}$$

# Computing $M_P$

$T_P$ is monotone: $I \subseteq J \rightarrow T_P(I) \subseteq T_P(J)$ and
upward continuous: if $I_0 \subseteq I_1 \subseteq I_2 \cdots$ then $T_P(\bigcup_{i \geq 0} I_i) = \bigcup_{i \geq 0} T_P(I_i)$

Let us define

$$\begin{aligned} T_P \uparrow 0 &= \emptyset \\ T_P \uparrow n+1 &= T_P(T_P \uparrow n) \\ T_P \uparrow \omega &= \bigcup_{n \geq 0} T_P \uparrow n \end{aligned}$$

**Theorem**

*If $P$ is a definite clause program, then $T_P \uparrow \omega = M_P = T_P(T_P \uparrow \omega)$.*

# Programs with negation

A rule of the following kind can be used

$$A_0 \leftarrow B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n$$

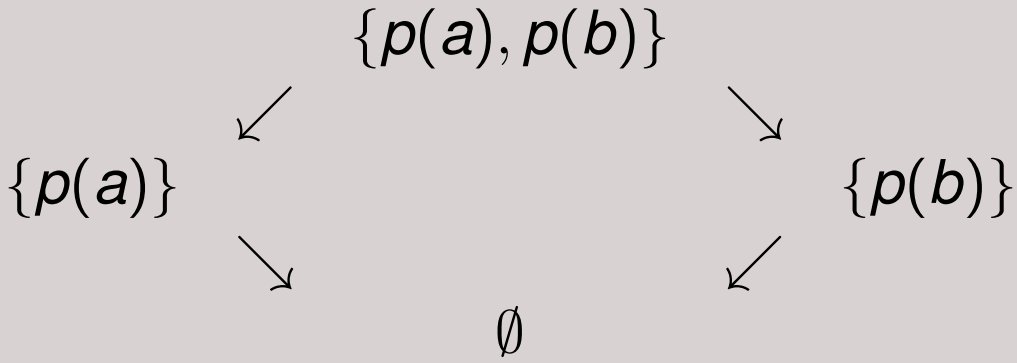
($A_i, B_j, C_k$ positive atoms). If $n > 0$ it is a clause with more than one positive literal.
An extended $T_P$ can be defined:

$$T_P(I) = \left\{ a : \begin{array}{l} a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n \in \text{ground}(P), \\ \{b_1, \ldots, b_m\} \subseteq I, \\ \{c_1, \ldots, c_n\} \cap I = \emptyset \end{array} \right\}$$

# Programs with negation

## Example

Let $P = p(a) \leftarrow \texttt{not}\ p(b)$ (it is the theory: $T = p(a) \vee p(b)$).
There are 4 Herbrand interpretations:



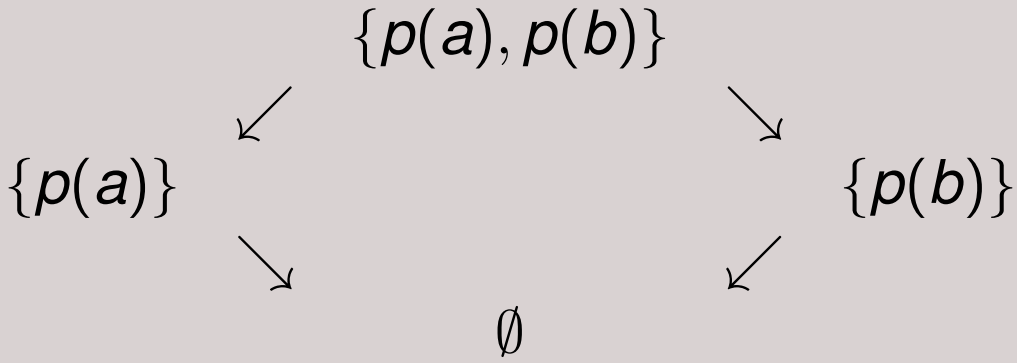

3 of them are models. There is no $A$ such that $T \models A$.

Moreover, $T_P(\emptyset) = \{p(a)\}$, $T_P(\{p(b)\}) = \emptyset$: this is not monotone.

# Programs with negation

## Example

Let $P = p(a) \leftarrow \texttt{not}\ p(b)$ (it is the theory: $T = p(a) \vee p(b)$).
There are 4 Herbrand interpretations:




3 of them are models. There is no $A$ such that $T \models A$.

Moreover, $T_P(\emptyset) = \{p(a)\}$, $T_P(\{p(b)\}) = \emptyset$: this is not monotone.

We need other techniques for reasoning on the semantics of programs with Negation (stable model semantics).

# Up or Down?

Let us consider $P$:

```
r(0) :- r(0).          p(0) :- q(X)
q(s(X)) :- q(X)
```

We have that: $T_P \uparrow \omega = \emptyset$. Let us compute instead

$$
\begin{array}{lcll}
T_P \downarrow 0 & = & \mathcal{B}_P & = \{r(0), r(s(0)), r(s(s(0))), r(s(s(s(0))), \ldots \\
 & & & p(0), p(s(0)), p(s(s(0))), p(s(s(s(0))), \ldots \\
 & & & q(0), q(s(0)), q(s(s(0)), q(s(s(s(0)))), \ldots\} \\
T_P \downarrow 1 & = & T_P(T_P \downarrow 0) & = \{r(0), p(0), q(s(0)), q(s(s(0)), q(s(s(s(0)))), \ldots\} \\
T_P \downarrow 2 & = & T_P(T_P \downarrow 1) & = \{r(0), p(0), q(s(s(0)), q(s(s(s(0)))), \ldots\} \\
T_P \downarrow 3 & = & T_P(T_P \downarrow 2) & = \{r(0)p(0), q(s(s(s(0)))), \ldots\} \\
\vdots & \vdots & \vdots & \vdots \\
T_P \downarrow \omega & = & \bigcap_{i \geq 0} T_P \downarrow i & = \{r(0), p(0)\}
\end{array}
$$

The latter is not a fixpoint: $T_P(\{r(0), p(0)\}) = \{r(0)\}$. This is a fixpoint (the greatest fixpoint): a transfinite number of applications is needed.

# Summary

- The semantics of definite clause logic programming is based on the minimum Herbrand model $M_P$. It is the set of logical consequences. It is computable, i.e. it is a recursively enumerable set. You can compute it top down by SLD resolution or bottom up by $T_P \uparrow \omega$ (the least fixpoint). It is recursive (PTIME) if there are not function symbols in $P$.

# Summary

- The semantics of definite clause logic programming is based on the minimum Herbrand model $M_P$. It is the set of logical consequences. It is computable, i.e. it is a recursively enumerable set. You can compute it top down by SLD resolution or bottom up by $T_P \uparrow \omega$ (the least fixpoint). It is recursive (PTIME) if there are not function symbols in $P$.
- Focusing on definite clause logic programming one can be interested in the greatest fixpoint of $T_P$ for coinductive reasoning (Coinductive Logic Programming). This set is not computable (it is a productive set) but can be under approximated.

# Summary

- The semantics of definite clause logic programming is based on the minimum Herbrand model $M_P$. It is the set of logical consequences. It is computable, i.e. it is a recursively enumerable set. You can compute it top down by SLD resolution or bottom up by $T_P \uparrow \omega$ (the least fixpoint). It is recursive (PTIME) if there are not function symbols in $P$.
- Focusing on definite clause logic programming one can be interested in the greatest fixpoint of $T_P$ for coinductive reasoning (Coinductive Logic Programming). This set is not computable (it is a productive set) but can be under approximated.
- As long as you consider negation in clause bodies, $M_P$ can not exists, $T_P$ is no longer monotone, the notion of logical consequence is no longer the appropriate one (next lesson). If no function symbols are used it is recursive (establishing existence is NP or $\Sigma_2^P$ according to head syntax).

Let us consider programs consisting of rules of the kind:

$$H \leftarrow B_1, \ldots, B_m, \texttt{not}\ B_{m+1}, \ldots, \texttt{not}\ B_n \tag{1}$$

where $H, B_1, \ldots, B_n$ are atoms, $n \geq 0$, $m \geq 0$ is said an (ASP) rule.

These programs are called *general programs*.

Let us try to understand the semantics of those programs (we already know that a minimum model is no longer guaranteed).

# Semantica dei Programmi Generali

## Completamento di programmi generali

Dato un programma *P*:

```
r(a, c).
r(a, d).
q(X) :- r(X,Y), not s(Y).
p(a) :- not p(b).
p(b) :- not p(a).
```

# Semantica dei Programmi Generali

## Completamento di programmi generali

Lo normalizzo ottenendo *norm*(*P*):

```
r(X1,X2) :- X1=a, X2=c.
r(X1,X2) :- X1=a, X2=d.
q(X1) :- r(X1,Y), not s(Y).
p(X1) :- X1=a, not p(b).
p(X1) :- X1=b, not p(a).
```

# Semantica dei Programmi Generali

## Completamento di programmi generali

Posso raccogliere le teste uguali e mettere dei se e solo se, e le quantificazioni esplicite ottenendo *iff*($P$):

$$\texttt{r}(X_1,X_2) \leftrightarrow (X_1=\texttt{a} \wedge X_2=\texttt{c}) \vee (X_1=\texttt{a} \wedge X_2=\texttt{d})$$
$$\texttt{q}(X_1) \leftrightarrow \exists\ \texttt{Y}\ (\texttt{r}(X_1,\texttt{Y}) \wedge \neg\ \texttt{s}(\texttt{Y}))$$
$$\texttt{p}(X_1) \leftrightarrow (X_1=\texttt{a} \wedge \neg\ \texttt{p}(\texttt{b})) \vee (X_1=\texttt{b} \wedge \neg\ \texttt{p}(\texttt{a}))$$
$$\texttt{s}(X_1) \leftrightarrow \texttt{false}$$

# Semantica dei Programmi Generali

## Completamento di programmi generali

Il completamento di $P$ è dunque:

$$\texttt{r}(X_1,X_2) \leftrightarrow (X_1=\texttt{a} \wedge X_2=\texttt{c}) \vee (X_1=\texttt{a} \wedge X_2=\texttt{d})$$
$$\texttt{q}(X_1) \leftrightarrow \exists\ \texttt{Y}\ (\texttt{r}(X_1,\texttt{Y}) \wedge \neg\ \texttt{s}(\texttt{Y}))$$
$$\texttt{p}(X_1) \leftrightarrow (X_1=\texttt{a} \wedge \neg\ \texttt{p}(\texttt{b})) \vee (X_1=\texttt{b} \wedge \neg\ \texttt{p}(\texttt{a}))$$
$$\texttt{s}(X_1) \leftrightarrow \texttt{false}$$

(ad essere precisi, più i *freeness axioms*)

# Semantica dei Programmi Generali

## Modelli di Herbrand del completamento

$$r(X_1,X_2) \leftrightarrow (X_1=a \wedge X_2=c) \vee (X_1=a \wedge X_2=d)$$
$$q(X_1) \leftrightarrow \exists\ Y\ (r(X_1,Y) \wedge \neg\ s(Y))$$
$$p(X_1) \leftrightarrow (X_1=a \wedge \neg\ p(b)) \vee (X_1=b \wedge \neg\ p(a))$$
$$s(X_1) \leftrightarrow \mathtt{false}$$

Gli atomi da considerare sono 28:

| | | | |
|---|---|---|---|
| s(a) | s(b) | s(c) | s(d) |
| p(a) | p(b) | p(c) | p(d) |
| q(a) | q(b) | q(c) | q(d) |
| r(a,a) | r(a,b) | r(a,c) | r(a,d) |
| r(b,a) | r(b,b) | r(b,c) | r(b,d) |
| r(c,a) | r(c,b) | r(c,c) | r(c,d) |
| r(d,a) | r(d,b) | r(d,c) | r(d,d) |

Quali stanno in tutti i modelli del completamento?

Quali in nessuno?

# Semantica dei Programmi Generali

## Modelli di Herbrand del completamento

$$r(X_1,X_2) \leftrightarrow (X_1=a \wedge X_2=c) \vee (X_1=a \wedge X_2=d)$$
$$q(X_1) \leftrightarrow \exists\ Y\ (r(X_1,Y) \wedge \neg\ s(Y))$$
$$p(X_1) \leftrightarrow (X_1=a \wedge \neg\ p(b)) \vee (X_1=b \wedge \neg\ p(a))$$
$$s(X_1) \leftrightarrow \mathtt{false}$$

| | | | |
|---|---|---|---|
| s(a) F | s(b) F | s(c) F | s(d) F |
| p(a) | p(b) | p(c) | p(d) |
| q(a) | q(b) | q(c) | q(d) |
| r(a,a) | r(a,b) | r(a,c) | r(a,d) |
| r(b,a) | r(b,b) | r(b,c) | r(b,d) |
| r(c,a) | r(c,b) | r(c,c) | r(c,d) |
| r(d,a) | r(d,b) | r(d,c) | r(d,d) |

Studiamo s

# Semantica dei Programmi Generali

## Modelli di Herbrand del completamento

$$r(X_1,X_2) \leftrightarrow (X_1=a \wedge X_2=c) \vee (X_1=a \wedge X_2=d)$$
$$q(X_1) \leftrightarrow \exists\ Y\ (r(X_1,Y) \wedge \neg\ s(Y))$$
$$p(X_1) \leftrightarrow (X_1=a \wedge \neg\ p(b)) \vee (X_1=b \wedge \neg\ p(a))$$
$$s(X_1) \leftrightarrow \mathtt{false}$$

| | | | |
|---|---|---|---|
| s(a) F | s(b) F | s(c) F | s(d) F |
| p(a) | p(b) | p(c) | p(d) |
| q(a) | q(b) | q(c) | q(d) |
| r(a,a) F | r(a,b) F | r(a,c) T | r(a,d) T |
| r(b,a) F | r(b,b) F | r(b,c) F | r(b,d) F |
| r(c,a) F | r(c,b) F | r(c,c) F | r(c,d) F |
| r(d,a) F | r(d,b) F | r(d,c) F | r(d,d) F |

Studiamo r.

# Semantica dei Programmi Generali

## Modelli di Herbrand del completamento

$$r(X_1,X_2) \leftrightarrow (X_1=a \wedge X_2=c) \vee (X_1=a \wedge X_2=d)$$
$$q(X_1) \leftrightarrow \exists\ Y\ (r(X_1,Y) \wedge \neg\ s(Y))$$
$$p(X_1) \leftrightarrow (X_1=a \wedge \neg\ p(b)) \vee (X_1=b \wedge \neg\ p(a))$$
$$s(X_1) \leftrightarrow \text{false}$$

| | | | |
|---|---|---|---|
| s(a) F | s(b) F | s(c) F | s(d) F |
| p(a) | p(b) | p(c) | p(d) |
| q(a) T | q(b) F | q(c) F | q(d) F |
| r(a,a) F | r(a,b) F | r(a,c) T | r(a,d) T |
| r(b,a) F | r(b,b) F | r(b,c) F | r(b,d) F |
| r(c,a) F | r(c,b) F | r(c,c) F | r(c,d) F |
| r(d,a) F | r(d,b) F | r(d,c) F | r(d,d) F |

Studiamo q.

# Semantica dei Programmi Generali

## Modelli di Herbrand del completamento

$$r(X_1,X_2) \leftrightarrow (X_1=a \wedge X_2=c) \vee (X_1=a \wedge X_2=d)$$
$$q(X_1) \leftrightarrow \exists\ Y\ (r(X_1,Y) \wedge \neg\ s(Y))$$
$$p(X_1) \leftrightarrow (X_1=a \wedge \neg\ p(b)) \vee (X_1=b \wedge \neg\ p(a))$$
$$s(X_1) \leftrightarrow \text{false}$$

| | | | |
|---|---|---|---|
| s(a) F | s(b) F | s(c) F | s(d) F |
| p(a) ? | p(b) ? | p(c) F | p(d) F |
| q(a) T | q(b) F | q(c) F | q(d) F |
| r(a,a) F | r(a,b) F | r(a,c) T | r(a,d) T |
| r(b,a) F | r(b,b) F | r(b,c) F | r(b,d) F |
| r(c,a) F | r(c,b) F | r(c,c) F | r(c,d) F |
| r(d,a) F | r(d,b) F | r(d,c) F | r(d,d) F |

Studiamo p.

# Semantica dei Programmi Generali

## Modelli di Herbrand del completamento

- Si nota che vi sono tre Ground di atomi:
  - Quelli veri in tutti i modelli di Herbrand del completamento.
  - Quelli falsi in tutti i modelli di Herbrand del completamento.
  - Quelli veri in alcuni e falsi in altri.
- Ciò suggerisce di determinare l'insieme dei $I^+$ sempre veri e quello $I^-$ dei sempre falsi (semantica di Fitting).
- Tali insiemi possono essere calcolati (talvolta in tutto o talvolta in parte) usando la nozione di *modello well-founded*.
- Il modello well-founded è una coppia $\langle I^+, I^- \rangle$ ed è unico e calcolabile in tempo polinomiale sul programma ground.
- In realtà, se $I^+ \cup I^- \neq B_P$ non è un vero modello !
- Se lo è sarà anche l'unico modello stabile (vedremo cos'è un modello stabile)

# Semantica dei Programmi Generali

## Grounding

- La nozione di modello well-founded e quella che vedremo di modello stabile sono fondate sulla nozione di programma ground
- Dato un programma generale $P$, il corrispondente programma ground $ground(P)$ è l'insieme di tutte le istanze ground su $\mathcal{H}_P$ delle clausole di $P$.

# Semantica dei Programmi Generali

## Grounding

$P =$

```
p(a).  p(b).  q(b).  q(c).
p(X,Y) :- p(X),q(Y).
```

$ground(P) =$

```
p(a).  p(b).  q(b).  q(c).
p(a,a) :- p(a),q(a).
p(a,b) :- p(a),q(b).
p(a,c) :- p(a),q(c).
p(b,a) :- p(c),q(a).
p(b,b) :- p(c),q(b).
p(b,c) :- p(c),q(c).
p(c,a) :- p(c),q(a).
p(c,b) :- p(c),q(b).
p(c,c) :- p(c),q(c).
```

# Semantica dei Programmi Generali

## Grounding

- Quant'è costoso il grounding?
- Assumiamo che $H_P$ sia finito (e sia $|H_P| = c$).
- Siano $c_1, \ldots, c_n$ le clausole di $P$ ($|P| = n$), e siano, rispettivamente, $\alpha_1, \ldots, \alpha_n$ i numeri delle variabili occorrenti in esse.

# Semantica dei Programmi Generali

## Grounding

- Quant'è costoso il grounding?
- Assumiamo che $H_P$ sia finito (e sia $|H_P| = c$).
- Siano $c_1, \ldots, c_n$ le clausole di $P$ ($|P| = n$), e siano, rispettivamente, $\alpha_1, \ldots, \alpha_n$ i numeri delle variabili occorrenti in esse.
- Allora

$$|ground(P)| = \sum_{i=1}^{n} c^{\alpha_i}$$

# Semantica dei Programmi Generali

## Grounding

- Quant'è costoso il grounding?
- Assumiamo che $H_P$ sia finito (e sia $|H_P| = c$).
- Siano $c_1, \ldots, c_n$ le clausole di $P$ ($|P| = n$), e siano, rispettivamente, $\alpha_1, \ldots, \alpha_n$ i numeri delle variabili occorrenti in esse.
- Allora

$$|ground(P)| = \sum_{i=1}^{n} c^{\alpha_i}$$

- Se $k = \max_i\{\alpha_i\}$, abbiamo che

$$|ground(P)| \leq nc^k$$

# Semantica dei Programmi Generali

## Grounding

- Quant'è costoso il grounding?
- Assumiamo che $H_P$ sia finito (e sia $|H_P| = c$).
- Siano $c_1, \ldots, c_n$ le clausole di $P$ ($|P| = n$), e siano, rispettivamente, $\alpha_1, \ldots, \alpha_n$ i numeri delle variabili occorrenti in esse.
- Allora

$$|ground(P)| = \sum_{i=1}^{n} c^{\alpha_i}$$

- Se $k = \max_i\{\alpha_i\}$, abbiamo che

$$|ground(P)| \leq nc^k$$

- $c$ può essere definito implicitamente (p.es. $p(1..100)$). In questi casi i numeri si sentono.

# ASP solvers

## lparse

- Per rendere finito il processo di grounding e generare un programma *ground*($P$) composto da un numero finito di regole, il programma $P$ deve rispettare alcuni vincoli.
- I simboli di funzione *non sono vietati*, ma è consentito solamente un uso limitato.
- Il programma deve essere *strongly range restricted*.
- L'idea base è che il programma $P$ deve essere strutturato in modo che sia possibile (ed automatico per il solver), per ogni variabile di una regola, stabilire l'insieme di valori che essa può assumere.
- tale insieme deve essere inoltre finito.

# Semantica dei Programmi Generali

## Modelli Stabili

- Se $P$ è

```
p :- not q.
```

  Sappiamo che $P$ ha due modelli minimali: $\{p\}$ e $\{q\}$.
- Tuttavia, osserviamo che il modello $\{q\}$ non riflette il significato intuitivo che attribuiamo al programma $P$ (ovvero "se si dimostra che $q$ è falso allora vale $p$"). Infatti in $P$ non vi è nessuna informazione che fornisca una giustificazione a sostegno della verità di $q$.
- In questo caso dunque vorremmo come modello stabile $\{p\}$ (è anche well-founded).
- Se $P$ invece è:

```
p :- not q.   q :- not p.
```

  well-founded dice $I^+ = \emptyset, I^- = \emptyset$

# Semantica dei Programmi Generali

## Modelli Stabili

- Dato un programma generale $P$ e un suo modello $S$, definiamo il programma $P^S$ ridotto di $P$ rispetto a $S$ nel seguente modo:
  1. rimuovendo ogni regola il cui corpo contiene un naf-literal `not` $L$ tale che $L \in S$;
  2. rimuovendo tutti i naf-literal dai corpi delle restanti regole.
- Per costruzione $P^S$ è un programma definito. Se il suo modello minimo coincide con $S$, allora $S$ è un modello stabile (anche detto answer set) per $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

$\emptyset$ Abbiamo che $P^{\emptyset} = \{p \leftarrow a.\ \ a.\ \ b.\}$ ma $\emptyset$ non è answer set per $P^{\emptyset}$.
Quindi $\emptyset$ non è un answer set di $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

$\{a\}$ Abbiamo che $P^{\{a\}} = \{p \leftarrow a.\ \ a.\}$ ma $\{a\}$ non è answer set per $P^{\{a\}}$. Quindi neanche $\{a\}$ è un answer set di $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

$\{b\}$ Abbiamo che $P^{\{b\}} = \{p \leftarrow a.\ \ b.\}$ e $\{b\}$ è l'answer set di $P^{\{b\}}$ (ovvero, è il modello minimo). Quindi $\{b\}$ è un answer set di $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

$\{p\}$ Abbiamo che $P^{\{p\}} = \{p \leftarrow a.\ \ a.\ \ b.\}$ ma $\{p\}$ non è answer set per $P^{\{p\}}$. Quindi neanche $\{p\}$ è un answer set di $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

$\{p, a\}$ Abbiamo che $P^{\{p,a\}} = \{p \leftarrow a.\ \ a.\}$ e $\{p, a\}$ è l'answer set di $P^{\{p,a\}}$. Quindi $\{p, a\}$ è un answer set di $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

$\{a, b\}$, $\{b, p\}$ e $\{a, b, p\}$, non sono answer set di $P$ perchè includono propriamente un answer set (ad esempio $\{b\}$)

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- a.
a :- not b.
b :- not a.
```

I candidati modelli stabili di $P$ sono i sottoinsiemi di $\mathcal{B}_P = \{a, b, p\}$.

Quindi $P$ ha due answer set distinti: $\{b\}$ e $\{p, a\}$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

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

L'insieme $S_1 = \{b, d\}$ è un answer set di $P$. Infatti $P^{S_1} = \{b.\ \ d.\}$ che ha $S_1$ come answer set.
Invece l'insieme $S_2 = \{a, d\}$ non è un answer set di $P$. Infatti $P^{S_2} = \{a.\ \ b.\ \ d.\}$ che non ha $S_2$ come answer set.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- not p,  d.
d.
```

Questo programma ammette il modello $\{p, d\}$. Ogni modello di $P$ deve contenere $d$. Q uindi abbiamo due possibili candidati ad essere modello stabile:

- $S_1 = \{d\}$: allora $P^{S_1} = \{p \leftarrow d.\ \ d.\}$ ha come modello minimo $\{d, p\}$ che è diverso da $S_1$. Quindi $S_1$ non è answer set per $P$.
- $S_2 = \{d, p\}$: allora $P^{S_2} = \{\ d.\}$ ha come answer set $\{d\}$ che è diverso da $S_2$. Quindi $S_2$ non è answer set per $P$.

# Semantica dei Programmi Generali

## Modelli Stabili

Consideriamo il programma $P$

```
p :- not p,  d.
d.
```

Questo programma ammette il modello $\{p, d\}$. Ogni modello di $P$ deve contenere $d$. Q uindi abbiamo due possibili candidati ad essere modello stabile:

- $S_1 = \{d\}$: allora $P^{S_1} = \{p \leftarrow d.\ \ d.\}$ ha come modello minimo $\{d, p\}$ che è diverso da $S_1$. Quindi $S_1$ non è answer set per $P$.
- $S_2 = \{d, p\}$: allora $P^{S_2} = \{\ d.\}$ ha come answer set $\{d\}$ che è diverso da $S_2$. Quindi $S_2$ non è answer set per $P$.

Questo Programma ha modelli, ma non ha modelli stabili.

# Semantica dei Programmi Generali

## “Vincoli”

- Pensiamo al corpo di una regola ASP come ad una giustificazione per supportare la verità della testa della regola.
- L’idea intuitiva per decidere se un modello sia o meno un answer set è la seguente: *“Un qualsiasi p è presente nell’answer set solo se è forzato ad esserlo in quanto testa di una regola con corpo vero. Tuttavia, la verità della testa p di una regola non può essere giustificata in base alla verità di* `not` *p nel corpo.”*
- Quindi la regola

```
p :- not p, d.
```

  non può supportare la verità di *p*.
- (potrebbe comunque essere supportata da una altra regola, se ci fosse)

# Semantica dei Programmi Generali

## “Vincoli”

- Dal punto di vista della ricerca di answer set, pertanto, se `p` non occorre altrove, la regola

```
p :- not p, d.
```

  equivale ad imporre che *d* sia falso.
- Potremmo pertanto far ciò scrivendo il vincolo (goal)

```
:- d
```

- Ammettiamo di usare vincoli di questo tipo. Sappiamo che sono uno zucchero sintattico.

# Semantica dei Programmi Generali

## Scelte non-deterministiche

Consideriamo il seguente programma

```
a :- not n_a.   n_a :- not a.
b :- not n_b.   n_b :- not b.
c :- not n_c.   n_c :- not c.
d :- not n_d.   n_d :- not d.
```

Gli answer set di questo programma rappresentano tutti i modi possibili di scegliere una e una sola alternativa

- tra `a` e `n_a`,
- tra `b` e `n_b`,
- tra `c` e `n_c`,
- tra `d` e `n_d`.

# Semantica dei Programmi Generali

## Complessità

**Theorem**

*Il problema di stabilire se un programma generale ground $P$ ammette modelli stabili è NP-completo.*

NP Dato $P$ ground, un possibile modello stabile $S$ conterrà necessariamente solo atomi presenti in $P$. Dunque $|S| \leq |P|$. Per verificare se $S$ sia o meno modello stabile è sufficiente calcolare $P^S$ e il modello minimo dello stesso. Entrambe le operazioni si effettuano in tempo polinomiale su $|P|$. Dunque il problema appartiene ad NP.

# Semantica dei Programmi Generali

## Complessità

**Theorem**

*Il problema di stabilire se un programma generale ground P ammette modelli stabili è NP-completo.*

**Hardness** Consideriamo un'istanza $\varphi$ di 3SAT:

$$\underbrace{(A \vee \neg B \vee C)}_{\texttt{c1}} \wedge \underbrace{(\neg A \vee B \vee \neg C)}_{\texttt{c2}}$$

e si costruisca il seguente programma $P$:

```
a :- not na.   na :- not a.
b :- not nb.   nb :- not b.
c :- not nc.   nc :- not c.
c1 :- a.   c1 :- nb.   c1 :- c.
c2 :- na.   c2 :- b.   c2 :- nc.
:- not c1.   :- not c2.
```

# Semantica dei Programmi Generali

## Riassunto

- $P$ definito: unico modello minimo (che è anche well-founded e stabile). Se $P$ ground finito si calcola in tempo polinomiale (no Turing)
- $P$ generale. Ha sempre modello $B_P$, ma non è interessante.
- $P$ generale. Ammette unico "modello" well-founded. Se $P$ ground finito si calcola in tempo polinomiale (no Turing).
  Se è totale, allora è anche unico modello stabile.
- Se invece è parziale (cicli tra negazioni), sempre se $P$ ground finito, stabilire l'esistenza di un modello stabile è NP-completo.
- Abbiamo un formalismo per il calcolo fatto apposta per i problemi in NP !!!!!!!

# Programmazione in ASP

## Come si calcolano in pratica i modelli stabili?

- Si usa un front-end (lparse, Gringo, DLV grounder) che *groundizza* il programma.
- Poi si chiama un ASP solver (smodels, cmodels, clasp, DLV, Yasmin–GPU)
- Esiste anche GASP!, un solver che rimanda il più possibile la fase di grounding, in ogni caso applicandola ad una clausola alla volta.
- clingo (= gringo+clasp) ci nasconde la fase di grounding

# Programmazione in ASP

## Intervalli

- Nei programmi per i database “familiari” avevamo usato dei fatti con argomento dei nomi di persona.
- Spesso nelle codifiche di problemi o giochi, vorremmo definire degli insiemi di valori.
- Ad esempio i valori per un predicato che spazi nel lato di un quadrato (o di una scacchiera) di dimensione 4:

  ```
  lato(1).   lato(2).   lato(3).   lato(4).
  ```

- Oppure, usando una abbreviazione comoda:

  ```
  lato(1..4).
  ```

- Ancora più in generale, si può usare una *costante* che viene istanziata all’esecuzione

  ```
  lato(1..n).
  ```

  (Ad esempio se vogliamo `n=4`, si eseguirà: `clingo -c n=4 nomefile.lp`)

# Programmazione in ASP

## Abbreviazione

- Se invece non siamo di fronte ad un intervallo, come:

  ```
  primo(2).   primo(3).   primo(5).   primo(7).
  ```

  possiamo (volendo) usare

  ```
  primo(2;3;5;7).
  ```

- Questo vale per i fatti o in generale nella testa di una regola.
- Nel corpo di una clausola ha un significato diverso a seconda delle versioni di gringo/clingo. Suggerisco di non usare questa abbreviazione nel corpo.

# Programmazione in ASP

## Cardinality constraint

Un *Ground cardinality constraint* si usa come un atomo positivo:

$$n\{\mathtt{L}_1,\ldots,\mathtt{L}_h,\ \mathtt{not}\ \mathtt{H}_1,\ldots,\ \mathtt{not}\ \mathtt{H}_k\}m$$

dove $\mathtt{L}_1,\ldots,\mathtt{L}_h,\mathtt{H}_1,\ldots,\mathtt{H}_k$ sono atomi e $n$ e $m$ sono numeri interi (uno o entrambi possono essere assenti). Definiamo, relativamente ad un insieme di atomi $S$ e ad un cardinality constraint $C$ il valore $val(C,S)$ come

$$val(C,S) = \mid S \cap \{\mathtt{L}_1,\ldots,\mathtt{L}_h\} \mid + (k - \mid S \cap \{\mathtt{H}_1,\ldots,\mathtt{H}_k\} \mid).$$

$C$ è vero in $S$ se $n \leqslant val(C,S) \leqslant m$.

# Programmazione in ASP

## Cardinality constraint

Un *Ground cardinality constraint* si usa come un atomo positivo:

$$n\{\mathtt{L}_1, \ldots, \mathtt{L}_h, \ \mathtt{not}\ \mathtt{H}_1, \ldots, \ \mathtt{not}\ \mathtt{H}_k\}m$$

dove $\mathtt{L}_1, \ldots, \mathtt{L}_h, \mathtt{H}_1, \ldots, \mathtt{H}_k$ sono atomi e $n$ e $m$ sono numeri interi (uno o entrambi possono essere assenti). Definiamo, relativamente ad un insieme di atomi $S$ e ad un cardinality constraint $C$ il valore $val(C, S)$ come

$$val(C, S) = \mid S \cap \{\mathtt{L}_1, \ldots, \mathtt{L}_h\} \mid + (k - \mid S \cap \{\mathtt{H}_1, \ldots, \mathtt{H}_k\} \mid).$$

$C$ è vero in $S$ se $n \leqslant val(C, S) \leqslant m$.
Un *cardinality constraint* può anche essere non ground (dobbiamo pensare alla sua versione ground per il significato.

# Programmazione in ASP

## Cardinality constraint: caso tipico

Usato come:

```
lato(1..2).
valore(1..3).
1 {assegna(X,Y,V) : valore(V) } 1 :- lato(X), lato(Y).
```

Dice che ad ogni coppia di “lati” (X,Y) viene assegnato uno ed un solo valore dal predicato assegna. In pratica diciamo che `assegna` è una funzione.

# Programmazione in ASP

## Cardinality constraint: caso tipico

Usato come:

```
lato(1..2).
valore(1..3).
1 {assegna(X,Y,V) : valore(V) } 1 :- lato(X), lato(Y).
```

Dice che ad ogni coppia di “lati” (X,Y) viene assegnato uno ed un solo valore dal predicato assegna. In pratica diciamo che `assegna` è una funzione. Equivale a

```
1 {assegna(X,Y,1),assegna(X,Y,2),assegna(X,Y,3)} 1 :-
        lato(X), lato(Y).
```

che a sua volta equivale a:

```
1 {assegna(1,1,1),assegna(1,1,2),assegna(1,1,3)} 1.
1 {assegna(1,2,1),assegna(1,2,2),assegna(1,2,3)} 1.
1 {assegna(2,1,1),assegna(2,1,2),assegna(2,1,3)} 1.
1 {assegna(2,2,1),assegna(2,2,2),assegna(2,2,3)} 1.
```

# Programmazione in ASP

## Scelta non deterministica

Usato come:

```
0 { coppiabuona(X,Y) } 1 :- lato(X), lato(Y).
```

Introduce del non determinismo. (X,Y) può essere una coppia buona o meno.
Si può anche scrivere così

```
{ coppiabuona(X,Y) } :- lato(X), lato(Y).
```

# Programmazione in ASP

## Aritmetica

- Funzioni built-in: +, -, *, “/” (div), “\” (mod), etc
- Relazioni built-in: =, !=, <, >, <=, >=, etc
- Tali funzioni vengono valutate durante il processo di grounding. Quindi in tale momento gli argomenti delle funzioni devono essere disponibili (e devono essere numeriche).
- È anche possibile per il programmatore definire proprie funzioni tramite dei programmi C o C++ “esterni”.

# Programmazione in ASP

## Esecuzione

- Supponiamo il programma $P$ stia nel file `asp.lp`.
- Il file viene processato da lparse/smodels invocando il comando

  `clingo asp.lp` $n$

  oppure semplicemente `clingo asp.lp`
- $n$ è un numero intero che indica quanti answer set (al massimo) smodels debba produrre ($n = 0$: tutti, nessun numero: 1 solo)
- Per programmi parametrici, si possono assegnare valori alle costanti (es. clingo -c n=15 asp.lp ...)
- La dichiarazione alla fine del file `#show p/2.` dice di stampare l'output del predicato `p` (che ha arità 2). In assenza della dichiarazione vengono stampati tutti gli atomi.