Richiami
Programmi Definiti
Grounding
Programmi Generali
Definizioni
Modelli Stabili
ASP programming

Logic Programming, Knowledge Representation, and Non Monotonic Reasoning

Agostino Dovier

Department of Mathematics and Computer Science,
University of Udine, Italy

Udine, Fall 2011

Sommario

Richiami
- Programmi Definiti
- Grounding

Programmi Generali
- Definizioni
- Modelli Stabili
- ASP programming

Programmi Definiti

Un programma definito P è un insieme di clausole (di Horn) con esattamente un letterale positivo

$$A :- \ B_1, \ldots, B_n$$

$n \geq 0$ e $A, B_1, \ldots, B_n$ sono formule atomiche (atomi).

Programmi Definiti

- Dato P possiamo definire l'universo di Herbrand H_P (insieme dei termini ground) e la base di Herbrand B_P (insieme degli atomi ground).

$$P = \left\{ \begin{array}{rcl} q(a). & & \\ q(b). & & \\ r(b). & & \\ r(c). & & \\ p(X,Y) & :- & q(X), r(Y). \end{array} \right\}$$

- $H_P = \{a, b, c\}$
- $B_P = \{q(a), q(b), q(c), r(a), r(b), r(c), p(a,a), p(a,b), p(a,c), p(b,a), p(b,b), p(b,c), p(c,a), p(c,b), p(c,c)\}$.

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

PROGRAMMI DEFINITI

- Dato P possiamo definire l'universo di Herbrand H_P (insieme dei termini ground) e la base di Herbrand B_P (insieme degli atomi ground).

$$P = \left\{ \begin{array}{rcl} q(a). & & \\ q(b). & & \\ r(b). & & \\ r(c). & & \\ p(X,Y) & :- & q(X), r(Y). \end{array} \right\}$$

- $H_P = \{a, b, c\}$
- $B_P = \{q(a), q(b), q(c), r(a), r(b), r(c), p(a,a), p(a,b), p(a,c), p(b,a), p(b,b), p(b,c), p(c,a), p(c,b), p(c,c)\}$.

Programmi Definiti

- Dato P possiamo definire l'universo di Herbrand H_P (insieme dei termini ground) e la base di Herbrand B_P (insieme degli atomi ground).

$$P = \left\{ \begin{array}{rcl} q(a). & & \\ q(b). & & \\ r(b). & & \\ r(c). & & \\ p(X,Y) & :- & q(X), r(Y). \end{array} \right\}$$

- $H_P = \{a, b, c\}$
- $B_P = \{q(a), q(b), q(c), r(a), r(b), r(c),$ $p(a,a), p(a,b), p(a,c), p(b,a), p(b,b), p(b,c),$ $p(c,a), p(c,b), p(c,c)\}$.

Programmi Definiti

- Hanno una semantica *logica* di unico modello minimo: Dato P esiste M_P minimo modello di Herbrand.
- $P = \{q(a). q(b). r(b). r(c). p(X,Y) :- \ q(X), r(Y).\}$, $M_P = \{q(a), q(b), r(b), r(c),$ $p(a,b), p(a,c), p(b,b), p(b,c)\}$.
- Ogni atomo $A \in B_P$ in M_P è conseguenza logica di P, ($P \models A$, ovvero A vale in tutti i modelli di P ovvero $P \cup \{\neg A\}$ è insoddisfacibile).
- Hanno una semantica *denotazionale* di minimo punto fisso mediante l'operatore $T_P : \wp(B_P) \longrightarrow \wp(B_P)$:

$$\begin{array}{rcll} T_P(I) & = & \{a : & a \leftarrow b_1, \ldots, b_m \in ground(P) \\ & & & b_1 \in I, \ldots, b_m \in I\} \end{array}$$

- Hanno una semantica *operazionale* basata sulla SLD-risoluzione.

Programmi Definiti

- Hanno una semantica *logica* di unico modello minimo: Dato P esiste M_P minimo modello di Herbrand.
- $P = \{q(a).\ q(b).\ r(b).\ r(c).\ p(X,Y) :- \ q(X), r(Y).\}$, $M_P = \{q(a), q(b), r(b), r(c),$ $p(a,b), p(a,c), p(b,b), p(b,c)\}$.
- Ogni atomo $A \in B_P$ in M_P è conseguenza logica di P, ($P \models A$, ovvero A vale in tutti i modelli di P ovvero $P \cup \{\neg A\}$ è insoddisfacibile).
- Hanno una semantica *denotazionale* di minimo punto fisso mediante l'operatore $T_P : \wp(B_P) \longrightarrow \wp(B_P)$:

$$\begin{array}{rcl} T_P(I) & = & \{a : \quad a \leftarrow b_1, \ldots, b_m \in ground(P) \\ & & \qquad b_1 \in I, \ldots, b_m \in I\} \end{array}$$

- Hanno una semantica *operazionale* basata sulla SLD-risoluzione.

Programmi Definiti

- Hanno una semantica *logica* di unico modello minimo: Dato P esiste M_P minimo modello di Herbrand.
- $P = \{q(a). q(b). r(b). r(c). p(X, Y) :- \ q(X), r(Y).\}$, $M_P = \{q(a), q(b), r(b), r(c), p(a, b), p(a, c), p(b, b), p(b, c)\}$.
- Ogni atomo $A \in B_P$ in M_P è conseguenza logica di P, ($P \models A$, ovvero A vale in tutti i modelli di P ovvero $P \cup \{\neg A\}$ è insoddisfacibile).
- Hanno una semantica *denotazionale* di minimo punto fisso mediante l'operatore $T_P : \wp(B_P) \longrightarrow \wp(B_P)$:

$$\begin{array}{rcll} T_P(I) & = & \{a : & a \leftarrow b_1, \ldots, b_m \in ground(P) \\ & & & b_1 \in I, \ldots, b_m \in I\} \end{array}$$

- Hanno una semantica *operazionale* basata sulla SLD-risoluzione.

PROGRAMMI DEFINITI

- Vale che: esiste SLD derivazione di successo per A se e solo se $A \in M_P = T_P \uparrow \omega(\emptyset)$ (equivalenza tra le semantiche)
- In altri termini, tutto ciò che sta in M_P è quello che intendiamo essere vero, il resto lo intendiamo falso.
- Sono Turing Completi (se si usa almeno un simbolo di funzione e uno di costante)
- Altrimenti (solo simboli di costante) diventano decidibili e sono un formalismo "povero" per il calcolo (DB deduttivi).
- Sono poco indicati per rappresentare la conoscenza (anche nel caso di presenza di simboli di funzione).

GROUNDING

- Le varie semantiche 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.

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).
```

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.

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.

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.

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.

```
studente(mark). studente(bill).
sposato(joe). sposato(mark). sposato(bob).
```

Definiamo il predicato che caratterizza le persone che sono studenti non sposati.

```
studente_single(X) :-
      studente(X),
      not sposato(X).
```

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

- La presenza del `not` nel corpo di una clausola ci fa uscire dall'insieme dei programmi definiti/clausole di Horn.
- Ad esempio,

```
studente_single(X) :-
     studente(X),
     not sposato(X).
```

equivale a:

studente_single(X) $\vee$ $\neg$studente(X) $\vee$sposato(X)

che non è una clausola di Horn.

Programmi Generali
Definizione

- Un *programma generale* è un programma in cui sono ammessi letterali negativi nel corpo delle clausole.
- Ovvero le regole sono del tipo

$$H \leftarrow B_1, \ldots, B_m, \texttt{not}\ \ C_1, \ldots, \texttt{not}\ \ C_n$$

  con H, B_i, C_i atomi.
- Un *goal generale* è un goal in cui sono ammessi letterali negativi.
- Le clausole non sono più clausole di Horn:

$$p(a) \leftarrow \neg q(a)$$

  è equivalente a $p(a) \vee q(a)$.
- Un programma comunque ammette sempre un modello: l'insieme di tutti gli atomi ground $\mathcal{B}_P$, soddisfacendo tutte le teste, è un modello.

Programmi Generali

Non Monotonia

- L'intersezione di modelli non è necessariamente un modello. Ad esempio, $p(a) \leftarrow \texttt{not} \;\; q(a)$ ammette (anche) i due modelli $\{p(a)\}$ e $\{q(a)\}$; tuttavia la loro intersezione non è un modello.
- Può non esistere un unico modello minimo; in generale possono esistere più modelli minimali (si veda sopra)
- Si può definire anche per i programmi generali un operatore T_P. Il modo naturale per fare ciò è:

$$\begin{array}{rcll} T_P(I) & = & \{a: & a \leftarrow b_1, \ldots, b_m, \texttt{not} \;\; c_1, \ldots, \texttt{not} \;\; c_n \in ground(P) \\ & & & b_1 \in I, \ldots, b_m \in I, \\ & & & c_1 \notin I, \ldots, c_n \notin I\} \end{array}$$

Tuttavia, $T_P(\emptyset) = \{p(a)\}$ e $T_P(\{q(a)\}) = \emptyset$:
T_P non è monotono (né continuo).
Non possiamo sfruttare il Teorema di punto fisso (Tarski) come per i programmi definiti.

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

- Quando scriviamo $p(a) :- \texttt{not}\ \ p(b)$, intendiamo davvero $p(a) \vee p(b)$?
- E' dunque lo stesso che scrivere $p(b) :- \texttt{not}\ \ p(a)$?
- Dobbiamo chiarire quali sono le conseguenze che vogliamo veramente ottenere o quantomeno ci aspettiamo da una regola generale.
- In realtà in KR sono due concetti diversi. Si parla di nagazione classica ($\neg$) e di default negation ($\texttt{not}$), anche detta negation as failure.

Programmi Generali

Negazione classica o altro?

- Quando scriviamo $p(a) :- \texttt{not}\ \ p(b)$, intendiamo davvero $p(a) \vee p(b)$?
- E' dunque lo stesso che scrivere $p(b) :- \texttt{not}\ \ p(a)$?
- Dobbiamo chiarire quali sono le conseguenze che vogliamo veramente ottenere o quantomeno ci aspettiamo da una regola generale.
- In realtà in KR sono due concetti diversi. Si parla di nagazione classica ($\neg$) e di default negation ($\texttt{not}\ \ $), anche detta negation as failure.

Semantica dei Programmi Generali

- Molte sono state le proposte per assegnare una semantica a un programma generale.
- Sotto certe condizioni (ad esempio, la stratificazione) si può estendere la nozione di semantica del modello minimo e garantire una semantica univoca.
- Ad esempio:

```
p.
q.
r :- p,q.
s :- p, not r.
t :- p, not s.
```

Il modello minimale è `{p,q,r,t}`.
E' possibile ragionare "a livelli"

SEMANTICA DEI PROGRAMMI GENERALI

MODELLI STABILI (GELFOND-LIFSCHITZ 1988)

- L'ipotesi di stratificazione è però molto restrittiva, pertanto non consideriamola in quanto segue.
- Sia P ground.
- I *modelli stabili* di un programma P vanno ricercati tra i sottoinsiemi *minimali* di $\mathcal{B}_P$ che siano modello di P.
- Caso base: Un programma generale privo di negazioni (naf-literals) è un programma definito e quindi ha un unico modello minimale (il modello minimo): quello sarà anche il suo modello stabile.

- 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 non riesco a dimostrare che q è vero 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 $I^+ = \emptyset, I^- = \emptyset$

- 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.

Consideriamo il programma P

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

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

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

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.

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.

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.

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.

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.

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 $\{b\}$)

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 $\{p, a\}$.

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

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.

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.

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

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.
- Possiamo leggerlo come: non è possibile che sia vero *d*.
- Similmente,

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

  sta per non è possibile che siano veri sia *a* che *b* e nel contempo falsi *c* e *d*.

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.
- Possiamo leggerlo come: non è possibile che sia vero *d*.
- Similmente,

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

  sta per non è possibile che siano veri sia *a* che *b* e nel contempo falsi *c* e *d*.

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.
- Possiamo leggerlo come: non è possibile che sia vero *d*.
- Similmente,

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

  sta per non è possibile che siano veri sia *a* che *b* e nel contempo falsi *c* e *d*.

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`.

In pratica `a` è ¬`n_a` e `n_a` è ¬`a`: abbiamo recuperato la negazione classica.

SEMANTICA DEI PROGRAMMI GENERALI

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)
- Esiste anche GASP!, un solver che rimanda il più possibile la fase di grounding, in ogni caso applicandola ad una clausola alla volta.
- Il sistema più veloce è sviluppato all'Università di Potsdam, dal gruppo diretto da Torsten Schaub. Si tratta di Gringo+Clasp, in breve clingo.

- Si usa un front-end (lparse, Gringo, DLV grounder) che *groundizza* il programma.
- Poi si chiama un ASP solver (smodels, cmodels, clasp, DLV)
- Esiste anche GASP!, un solver che rimanda il più possibile la fase di grounding, in ogni caso applicandola ad una clausola alla volta.
- Il sistema più veloce è sviluppato all'Università di Potsdam, dal gruppo diretto da Torsten Schaub. Si tratta di Gringo+Clasp, in breve clingo.

ASP solvers
Finitezza del grounding

- Sia P un programma. Il *dependency graph* D_P di P è un grafo etichettato così definito:
- i nodi di D_P corrispondono ai simboli di predicato presenti in P;
- se in P esiste una regola in cui p_i occorre nella testa e p_j occorre nel corpo

  $$p_i\,(\ldots) \quad \texttt{:-} \quad \ldots \quad p_j\,(\ldots) \quad \ldots$$

  allora in D_P si mette l'arco $\langle p_i, p_j, \ell\rangle$ (o $p_i \xrightarrow{\ell} p_j$).
- ℓ può essere uno o entrambi i simboli $+$, $-$ a seconda che il simbolo p_j occorra in un letterale positivo o negativo nel corpo della regola.
- Un ciclo nel grafo D_P si dice *ciclo negativo* se almeno una delle sue etichette contiene $-$.
- Un programma senza cicli negativi è stratificato.

- Sia P un programma. Il *dependency graph* D_P di P è un grafo etichettato così definito:
- i nodi di D_P corrispondono ai simboli di predicato presenti in P;
- se in P esiste una regola in cui p_i occorre nella testa e p_j occorre nel corpo

  $$p_i(\ldots) \quad :- \quad \ldots \quad p_j(\ldots) \quad \ldots$$

  allora in D_P si mette l'arco $\langle p_i, p_j, \ell \rangle$ (o $p_i \xrightarrow{\ell} p_j$).
- ℓ può essere uno o entrambi i simboli $+$, $-$ a seconda che il simbolo p_j occorra in un letterale positivo o negativo nel corpo della regola.
- Un ciclo nel grafo D_P si dice *ciclo negativo* se almeno una delle sue etichette contiene $-$.
- Un programma senza cicli negativi è stratificato.

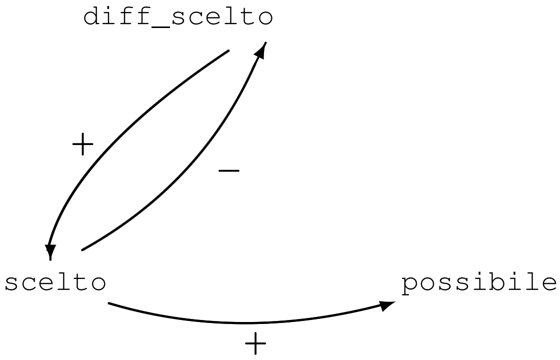
RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

- Un predicato p che occorre in P si dice *predicato di dominio* se e solo se in D_P ogni cammino che parte dal nodo p non contiene cicli negativi.
- Una regola ρ si dice *strongly range restricted* se ogni variabile che occorre in ρ occorre anche negli argomenti di un predicato di dominio nel corpo di ρ.
- Un programma P è *strongly range restricted* se tutte le sue regole sono strongly range restricted.

ASP solvers

Finitezza del grounding

```
diff_scelto(X) :- scelto(Y), X!=Y.
scelto(X) :- possibile(X), not diff_scelto(X).
possibile(a).  possibile(b).  possibile(c).
```



RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

Un Ground cardinality constraint si usa come un atomo positivo della forma

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

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 si usa come un atomo positivo della forma

n `{p(X,Y): range(X,Y) }` m

dove `range` è predicato di dominio.
C è vero in S se il numero di atomi della forma $p(X, Y)$ (tali che $range(X, Y)$) è compreso tra n e m.

In luogo delle regole

```
val(4).
val(5).
val(6).
val(7).
```

possiamo scrivere

```
val(4..7).
```

Al posto della regola

```
p:- q(6), q(7), q(8).
```

è possibile la regola

```
p:- q(6..8).
```

(e altre cose usando il ; ...)

RICHIAMI
PROGRAMMI DEFINITI
GROUNDING
PROGRAMMI GENERALI
DEFINIZIONI
MODELLI STABILI
ASP PROGRAMMING

- Funzioni built-in: `plus`, `minus`, `times`, `div`, `mod`, `lt`, `gt`, `le`, `ge`, `neq`, `abs`, `and`, `or`,
- Tali funzioni vengono valutate durante il processo di grounding. Quindi in tale momento gli argomenti delle funzioni devono essere disponibili.
- Ci sono sia l'operatore di confronto "==" che quello di assegnamento "=".
- È anche possibile per il programmatore definire proprie funzioni tramite dei programmi C o C++.