Constraint Programming & Planning

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, Aprile 2011

Sommario

Semantica della negazione

- Closed World Assumption
- Negation as Failure
- Implementazione della NaF
- Regola di Herbrand
- Confronti tra le tre regole

Semantica della negazione

Iniziamo vedendo alcuni approcci:

- la *Closed World Assumption* (CWA),
- la *Negation as (finite) failure* (NaF), e
- la *regola di Herbrand*.

Nel descrivere questi approcci tratteremo il caso di un programma definito P interrogato tramite un goal generale $\leftarrow$ `not` A.
Considereremo il caso in cui A è ground.
Vedremo poi di estendere il discorso a P generale.

Semantica della negazione

Iniziamo vedendo alcuni approcci:

- la *Closed World Assumption* (CWA),
- la *Negation as (finite) failure* (NaF), e
- la *regola di Herbrand*.

Nel descrivere questi approcci tratteremo il caso di un programma definito P interrogato tramite un goal generale $\leftarrow \texttt{not}\ \ A$.

Considereremo il caso in cui A è ground.

Vedremo poi di estendere il discorso a P generale.

SEMANTICA DELLA NEGAZIONE

Iniziamo vedendo alcuni approcci:

- la *Closed World Assumption* (CWA),
- la *Negation as (finite) failure* (NaF), e
- la *regola di Herbrand*.

Nel descrivere questi approcci tratteremo il caso di un programma definito P interrogato tramite un goal generale $\leftarrow \texttt{not}\ \ A$.
Considereremo il caso in cui A è ground.
Vedremo poi di estendere il discorso a P generale.

Semantica della negazione

CWA

- Il punto di partenza è la domanda:

  "Quando si deve fornire risposta positiva ad un goal della forma: $\leftarrow$ `not` A *?"*

- La regola Closed World Assumption (in breve, CWA) prevede di rispondere *yes* quando non esiste alcuna SLD-derivazione per il goal $\leftarrow A$ dal programma P.
- Ovvero, per i risultati di equivalenza, quando vale che

$$A \notin T_P \uparrow \omega.$$

- [Reiter'78] l'ha definito per le basi di dati deduttive.
- Non è però applicabile ai programmi definiti con simboli di funzione, in quanto la proprietà è indecidibile.

Semantica dei Programmi Generali

NaF

- La regola di Negazione per fallimento finito (NaF) dice di rispondere *yes* al goal $\leftarrow$ `not` A quando l'SLD-albero per $\leftarrow A$ è un albero di fallimento finito [Clark 78].
- Ovvero un albero contenente tutte le SLD derivazioni (fissate le regole di selezione e di computazione) da un goal, tale che tutte le foglie sono fallimentari ed è finito

Semantica dei Programmi Generali

NaF

- La regola di Negazione per fallimento finito (NaF) dice di rispondere *yes* al goal $\leftarrow$ `not` A quando l'SLD-albero per $\leftarrow A$ è un albero di fallimento finito [Clark 78].
- Ovvero un albero contenente tutte le SLD derivazioni (fissate le regole di selezione e di computazione) da un goal, tale che tutte le foglie sono fallimentari ed è finito

Semantica dei Programmi Generali

NaF

- NaF: *yes* a $\leftarrow$ `not` A se l'SLD-albero per $\leftarrow A$ è un albero di fallimento finito.
- È una approssimazione della CWA. Consideriamo un programma P e un goal ground $\leftarrow A$. Possono verificarsi tre possibilità:
  - esiste una SLD-derivazione di successo. In questo caso abbiamo dimostrato A e quindi non possiamo inferire `not` A.
  - Esiste un SLD-albero di fallimento finito per $\leftarrow A$. In questo caso possiamo inferire `not` A perchè siamo sicuri che non sia possibile dimostrare A.
  - Non troviamo una SLD-derivazione di successo e l'SLD-albero è infinito. In questo caso non possiamo ottenere una risposta in tempo finito, perchè prima di concludere che vale `not` A dovremmo visitare tutto l'albero.

Semantica dei Programmi Generali

NaF

- NaF: *yes* a $\leftarrow$ `not` A se l'SLD-albero per $\leftarrow A$ è un albero di fallimento finito.
- È una approssimazione della CWA. Consideriamo un programma P e un goal ground $\leftarrow A$. Possono verificarsi tre possibilità:
  - esiste una SLD-derivazione di successo. In questo caso abbiamo dimostrato A e quindi non possiamo inferire `not` A.
  - Esiste un SLD-albero di fallimento finito per $\leftarrow A$. In questo caso possiamo inferire `not` A perchè siamo sicuri che non sia possibile dimostrare A.
  - Non troviamo una SLD-derivazione di successo e l'SLD-albero è infinito. In questo caso non possiamo ottenere una risposta in tempo finito, perchè prima di concludere che vale `not` A dovremmo visitare tutto l'albero.

Semantica dei Programmi Generali

NaF

- NaF: *yes* a $\leftarrow$ `not` A se l'SLD-albero per $\leftarrow A$ è un albero di fallimento finito.
- È una approssimazione della CWA. Consideriamo un programma P e un goal ground $\leftarrow A$. Possono verificarsi tre possibilità:
  - esiste una SLD-derivazione di successo. In questo caso abbiamo dimostrato A e quindi non possiamo inferire `not` A.
  - Esiste un SLD-albero di fallimento finito per $\leftarrow A$. In questo caso possiamo inferire `not` A perchè siamo sicuri che non sia possibile dimostrare A.
  - Non troviamo una SLD-derivazione di successo e l'SLD-albero è infinito. In questo caso non possiamo ottenere una risposta in tempo finito, perchè prima di concludere che vale `not` A dovremmo visitare tutto l'albero.

SEMANTICA DEI PROGRAMMI GENERALI

NAF

- NaF: *yes* a $\leftarrow$ `not` A se l'SLD-albero per $\leftarrow A$ è un albero di fallimento finito.
- È una approssimazione della CWA. Consideriamo un programma P e un goal ground $\leftarrow A$. Possono verificarsi tre possibilità:
  - esiste una SLD-derivazione di successo. In questo caso abbiamo dimostrato A e quindi non possiamo inferire `not` A.
  - Esiste un SLD-albero di fallimento finito per $\leftarrow A$. In questo caso possiamo inferire `not` A perchè siamo sicuri che non sia possibile dimostrare A.
  - Non troviamo una SLD-derivazione di successo e l'SLD-albero è infinito. In questo caso non possiamo ottenere una risposta in tempo finito, perchè prima di concludere che vale `not` A dovremmo visitare tutto l'albero.

Consideriamo il programma definito

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

congiuntamente al goal `:- not r(b).`

- La regola CWA risponde *yes*, perché $r(b)$ non appartiene al modello minimo del programma che è $M_P = \{p(a), p(b), q(a), r(a)\}$.
- NaF opera come segue: viene iniziata una computazione *ausiliaria* per il goal `?- r(b)`. Tale computazione ausiliaria genera un SLD-albero finito e fornisce la risposta *no*. Quindi la risposta al goal generale iniziale sarà *yes*.

Aggiungiamo una tautologia al programma di prima:

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

e consideriamo sempre il goal `:- not r(b).`

- La regola CWA risponde *yes*, perché $r(b) \notin M_P = \{p(a), p(b), q(a), r(a)\}$.
- La computazione *ausiliaria* per il goal `?- r(b)` ora genera un SLD-albero (senza foglie di successo) infinito.
  Quindi non è in grado di fornire risposta *yes* al goal.

Consideriamo ora il programma:

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

ed il goal `?- not p(0).`

- La risposta ottenuta utilizzando CWA è *yes*, dato che $M_P = \emptyset$.
- La regola NaF invece prevede di verificare se il goal ausiliario `?- p(0)` sia dimostrabile. La computazione ausiliaria tuttavia risulta infinita. Pertanto NaF non risponderà nulla in questo caso.

SEMANTICA DEI PROGRAMMI GENERALI

SLDNF

La procedura risolutiva impiegata per implementare la NaF è chiamata SLDNF che estende la SLD-risoluzione con la negazione per fallimento finito.

DEFINITION

Sia $G = \leftarrow L_1, \ldots, L_n$ un goal generale. Sia L_i il letterale selezionato.

1. Se L_i è positivo, allora si esegue un passo standard di *SLD*-risoluzione
2. L_i è negativo (ovvero `not` A) e l'atomo A è ground, allora si inizi una SLDNF-derivazione aux per $\leftarrow A$.
   - Se tale computazione termina con *yes* allora la derivazione del goal G è di fallimento.
   - Se termina con *no* (si ottiene cioè un albero di fallimento finito), allora il goal risolvente di G è

$$G' = \leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n.$$

Consideriamo ora il programma generale P

```
p(a) :- not q(a)
```

(equivalente a $p(a) \vee q(a)$) e i goal:

1. $\leftarrow p(a). \Rightarrow yes.$
2. $\leftarrow$ `not` $p(a). \Rightarrow no;$
3. $\leftarrow q(a). \Rightarrow no.$
4. $\leftarrow$ `not` $q(a). \Rightarrow yes.$

Si noti l'asimmetria rispetto al significato logico.

Nella def di SLDNF-derivazione A è ground. Perchè?

```
(1)   uguale(X,X).
(2)   p :- not uguale(X,1), uguale(X,2).
(3)   p :- uguale(X,2), not uguale(X,1).
(4)   q :- not p
```

Consideriamo i programmi $P_1 = \{(1), (2), (4)\}$ e $P_2 = \{(1), (3), (4)\}$ logicamente equivalenti. Chiediamo `?- p` nei due casi.

- In P_1 si deriva `?- not p` e `?- q`.
- In P_2 si deriva `?- p` e `?- not q`.

SLDNF-risoluzione diventerebbe una procedura *scorretta* e *incompleta*.

Semantica dei Programmi Generali

SLDNF

- Vi sono condizioni sintattiche e di modo d'uso che garantiscono che la SLDNF-risoluzione sia corretta e completa.
- As esempio:

  > *Ogni variabile presente in un letterale negato del corpo deve essere presente in un letterale positivo del corpo o nella testa.*

- Dato un programma P e un goal generale G, diremo che la computazione per il goal G svolazza (*flounders*) se ad un certo punto della computazione si genera un goal che contiene solamente letterali negativi non ground.
- Se la computazione svolazza allora non siamo in grado di procedere nella derivazione neanche utilizzando SLDNF.

- In Prolog la negazione viene implementata nel seguente modo. In presenza di un goal della forma

  ```
  ?- not P.
  ```

  si agisce come se nel programma fossero presenti le clausole:

  ```
  not(P) :- P, !, fail.
  not(P).
  ```

- Nella prima clausola un termine nella testa diventa atomo nel corpo (meta-variable facility)
- Si noti l'uso del cut (ma ne abbiamo già visti di peggiori)
- Ultima nota: `not` viene scritto `\+` (ISO)

- La regola di Herbrand dice di inferire il goal $\leftarrow$ `not` A da un programma P quando A è falso in tutti i modelli di Herbrand del completamento $Comp(P)$ di P.
- Introduciamo il concetto di completamento su un esempio.

- La regola di Herbrand dice di inferire il goal $\leftarrow$ `not` A da un programma P quando A è falso in tutti i modelli di Herbrand del completamento $Comp(P)$ di P.
- Introduciamo il concetto di completamento su un esempio.

Dato un programma *P*:

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

Lo normalizzo ottenendo *norm*(P):

```
r(X1,X2) :- X1=a, X2=c.
p(X1) :- X1=a.
p(X1) :- X1=b.
q(X1) :- p(X1), r(X1,X2).
q(X1) :- s(X1).
```

Posso raccogliere le teste uguali e introdurre dei se e solo se ($\leftrightarrow$), delle clausole per predicati non definiti, e le quantificazioni esistenziali esplicite ottenendo *iff*(P):

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

Il completamento di P è dunque definito:

$$Comp(P) = iff(P) \wedge \textbf{(F1)} \wedge \textbf{(F2)} \wedge \textbf{(F3)}$$

dove **(F1)**, **(F2)** e **(F3)** sono gli assiomi della: *Clark's Equality Theory* (CET):

(F1) $f(t_1, \cdots, t_n) = f(s_1, \cdots, s_n) \rightarrow t_1 = s_1 \wedge \cdots \wedge t_n = s_n$

(F2) $f(t_1, \cdots, t_n) \neq g(s_1, \cdots, s_m)$ se $f \neq g$

(F3) $X \neq t[X]$: ogni termine X è diverso da ogni termine t che contenga X come sottotermine proprio.

Pertanto, $Comp(P) = (F1), (F2), (F3), iff(P)$:

$\texttt{r}(X_1,X_2) \leftrightarrow (X_1=\texttt{a} \wedge X_2=\texttt{c})$

$\texttt{p}(X_1) \leftrightarrow (X_1=\texttt{a}) \vee (X_1=\texttt{b})$

$\texttt{q}(X_1) \leftrightarrow (\exists\, X_2(\texttt{p}(X_1) \wedge \texttt{r}(X_1,X_2))) \vee \texttt{s}(X_1)$

$\texttt{s}(X_1) \leftrightarrow$ false

- $p(a), p(b), r(a,c), q(a)$ sono veri in tutti i suoi modelli di Herbrand.
- Pertanto, ad esempio, $\leftarrow \texttt{not}\ \ p(a)$ è falso, $\leftarrow \texttt{not}\ \ r(a,b)$ è vero.

- CWA coinvolge un problema indecidibile.
- Anche la regola di Herbrand, essendo basata sui modelli del completamento, non appare facilmente automatizzabile.
- NaF è implementabile, ma può entrare in loop.
- Vediamo il risultato che le collega.

Theorem

Sia P un programma definito e A un atomo di $\mathcal{B}_P$. Allora:

1. *$\neg A$ è inferibile tramite CWA se e solo se $A \in \mathcal{B}_P \setminus T_P \uparrow \omega$*
2. *$\neg A$ è inferibile tramite NaF se e solo se $A \in \mathcal{B}_P \setminus T_P \downarrow \omega$.*
3. *$\neg A$ è inferibile dalla regola di Herbrand se e solo se $A \in \mathcal{B}_P \setminus gfp(T_P)$ ($gfp(T_P) = T_P \downarrow \alpha$ per qualche ordinale $\alpha \geq \omega$).*

Disegno alla lavagna!

Ricordo che il risultato vale per P definito . . . dobbiamo ora estenderlo.

THEOREM

Sia P un programma definito e A un atomo di $\mathcal{B}_P$. Allora:

1. *$\neg A$ è inferibile tramite CWA se e solo se $A \in \mathcal{B}_P \setminus T_P \uparrow \omega$*
2. *$\neg A$ è inferibile tramite NaF se e solo se $A \in \mathcal{B}_P \setminus T_P \downarrow \omega$.*
3. *$\neg A$ è inferibile dalla regola di Herbrand se e solo se $A \in \mathcal{B}_P \setminus gfp(T_P)$ ($gfp(T_P) = T_P \downarrow \alpha$ per qualche ordinale $\alpha \geq \omega$).*

Disegno alla lavagna!
Ricordo che il risultato vale per P definito . . . dobbiamo ora estenderlo.