Lezione18

Table of contents

Semantica di PROLOG$_0$

Vista la sintassi, dobbiamo ora capire quale è il significato di un programma.
La semantica non è costruibile finché un punto d'innesco non è disponibile: il goal. Preso un programma e il goal, questi generano calcolo, nel tentativo di trovare quei valori alle variabili che rendano vero l'obbiettivo.

Piuttosto che vedere tutta la logica di 1^{o} ordine per comprendere la relazione tra programma e goal, applicheremo un approccio operazionale, dove la descrizione del significato del programma è reso dalla funzione non deterministica calcolata.

Il goal viene detto *soddisfatto* dal programma, se la funzione che andremo a costruire σ dello specifico programma, sarà in grado di calcolare almeno 1 risultato; viceversa se non lo è, il programma si dice *insoddisfacibile*. Ci sono casi in cui, il calcolo potrebbe impiegare tempo significativo, magari non terminare in tempo aspettato: solo perché così è, non vuole dire che il problema non ha soluzione; per come si sta muovendo l'esecutore, si entra in calcolo non terminabile.

Preso un programma π, $\mathbf{head}_P(\pi)$ è la prima clausola.
Questo permette di dire "considerando la prima clausola...", per fare ragionamenti.
Il resto del programma $\mathbf{rest}_P(\pi)$ è il programma con la prima clausola tolta; se almeno una clausola rimane, quello che rimane è un altro programma; tolte tutte le clausole, il programma assume in nome di $\bot$, che specifica l'arrivo in fondo al testo.

Come nel programma, il goal ν ha una testa $\mathbf{head}_G(\nu)$ e il resto $\mathbf{rest}_G(\nu)$; se congiunti non rimangono perché la computazione è terminata, restituiamo $\bot$.

Per evitare che 2 variabili abbiano lo stesso nome in clausole differenti, e quindi essere considerate la stessa variabile, una funzione $(.)'$ ci è fornita. Dal punto di vista logico, ogni clausola definisce le sue variabili con i suoi quantificatori, come se mettessimo un $\forall x$ davanti: siccome non vogliamo questo comportamento che inevitabilmente trova una x che sistemi tutte le clausole, abbiamo una funzione di ridenominazione. Il risultato di un'operazione simile è l'insieme delle nuove *varibili fresh*. Vediamo un esempio usando il fatto $h(X, Y, X)$:

$$(h(X, Y, X).)' = h(X1, Y1, X1).$$

con ipotesi che le variabili fresh non siano mai state usate in tutto il calcolo fatto.

La funzione non deterministica σ ha implicitamente a che fare con un programma P: per ogni strada che gli è possibile, calcola una sostituzione, dove è possibile. σ^π è una funzione per casi, ipotizzando che questi siano mutualmente esclusivi (entriamo solo in una).

$$\sigma^\pi(\nu) = \begin{cases} \sigma^\pi_G(g, \pi) & \text{se } g = \mathbf{head}_G(\nu) \wedge \bot = \mathbf{rest}_G(\nu) \\ \theta \circ \sigma^\pi(\text{:- } r\theta.) & \text{se } g = \mathbf{head}_G(\nu) \wedge r = \mathbf{rest}_G(\nu) \wedge r \neq \bot \wedge \ \theta = \sigma^\pi_G(g, \pi) \end{cases}$$

- Se il resto del goal ν è $\bot$ ($\mathbf{rest}_G(\nu) = \bot$), allora vuole dire che il goal è formato da un unico congiunto. Se formato da unico congiunto, andremo a costruire un σ_G che calcoli il valore di σ.
- Se un goal ha congiunto iniziale e resto ($\mathbf{head}_G(\nu) \wedge \mathbf{rest}_G(\nu)$), chiamiamo σ_G sulla testa G, cercando una sostituzione che la soddisfi; trovata una sostituzione, applichiamo la sostituzione al resto e andiamo avanti come se questo fosse un nuovo goal; il risultato sarà una sostituzione componibile θ con $\sigma^\pi(\text{:- } r\theta.)$.

Dato un goal formato unicamente da atomo o atomo(argomenti), σ_G^π cerca una sostituzione: l'esecutore PROLOG parte dalla cima del programma e una dopo l'altra considera le clausole; se la clausola è fatto e si riesce a usare questo per un goal, allora lì finisce, altrimenti se la clausola è regola e si riesce a utilizzare la testa per risolvere il goal, il corpo deve essere risolto.

$$\sigma_G^\pi(g,\pi) = \begin{cases} \sigma_F(g,c) & \text{se} \quad c = \mathtt{head}_P(\pi) \land c = h. \\ \sigma_R^\pi(g,c) & \text{se} \quad c = \mathtt{head}_P(\pi) \land c = h \text{ :- } b. \\ \sigma_G^\pi(g,r) & \text{se} \quad r = \mathtt{rest}_P(\pi) \land r \neq \bot \end{cases}$$

Per prima cosa, tutte le clausole vengono prese; σ_G lo si applica a congiunto e a programma.

- Se la testa del programma, una clausola, è uguale a un fatto ($c = h.$), calcoliamo σ_F.
- Se la testa del programma è una regola, con testa h e corpo b separati da :-, allora utilizziamo σ_R per calcolare il risultato di σ_G. Se la prima lavora sui fatti F, la seconda lavora sulle regole R.
- (Non deterministico) Se il resto del programma effettivamente contiene una clausola soltanto, allora induttivamente chiamiamo σ_G.

Con le prime 2 regole diciamo cosa succede se la testa del programma è un fatto o una regola; con l'ultima andiamo a dire "se ci sono ancora clausole ($r \neq \bot$), allora applica ricorsivamente σ_G". Dette le possibilità, vediamo cosa può fare ciascuna funzione.

1. $$\sigma_F(g,f) = \theta \quad \text{se} \quad f' = h'. \land \theta = \mathtt{mgu}(\{g,h'\}) \land \theta \neq \bot$$

   f è la clausola fatto su cui stiamo lavorando; la variante fresh f' viene calcolata cambiando quindi il nome di tutte le sue variabili, ottenendo un termine h'..
   g è un termine goal che combinato con h', calcola $\mathtt{mgu}$: se esiste un unificatore generale, allora il risultato c'è. Se il risultato non esiste ($\bot$), proviamo a usare la terza strada (perché le prime due non possiamo, sono mutualmente esclusive) e se nemmeno in quel caso abbiamo fortuna, allora il programma è finito.

2. $$\sigma_R^\pi(g,r) = \theta \circ \sigma^\pi(\text{:- } b'\theta.) \quad \text{se} \quad r' = h' \text{ :- } b' \land \theta = \mathtt{mgu}(\{g,h'\}) \land \theta \neq \bot$$

   Per prima cosa notiamo che r deve essere una regola.
   La variante fresh viene calcolata (h' :- b'); se abbiamo un risultato, cerchiamo se con h' riusciamo a trovare una sostituzione che ci permette di unificare il goal; se troviamo sostituzione, questa si chiamerà σ

Soluzione di un programma PROLOG$_0^+$

Dato $\pi = p(a).\, p(b).$, verifichiamo se il goal $\nu = \text{ :- } p(b).$ è soddisfacibile. Un goal è soddisfatto se ci viene calcolata una sostituzione, anche se vuota (quando applicata lascia il termine inalterato).

Visto come calcolare $\sigma^\pi(\nu)$, sappiamo che ci sono 2 strade possibili: se prendiamo la 1ª strada, scriveremo $\overset{1}{=}$, mentre scriveremo $\overset{2}{=}$ per la 2ª. Ci sono casi in cui le due strade sono mutualmente esclusive, ma non importa.

La prima cosa da fare è spezzare i congiunti. Siccome il nostro goal è formato da unico congiunto, il resto è $\bot$; possiamo prendere soltanto la strada 1.
Questa ci dice che dobbiamo calcolare σ_G^π con primo argomento la testa del goal e secondo il programma.

σ_G^π come sappiamo ha 3 strade:

- applicare se abbiamo fatto;
- applicare se abbiamo regola;
- applicabile se oltre a una clausola ne abbiamo un'altra.

Siccome $p(a).$ è un fatto, prendiamo la prima strada, che prevede di applicare σ_F al goal e alla prima clausola del programma. Dato il fatto che la variabile fresh rimane la stessa, perché non ci sono variabili in primo luogo, e dato il fatto che non c'é $\texttt{mgu}$, allora la soluzione è $\perp$. La strada è interrotta e non possiamo andare avanti: torniamo indietro.

La seconda strada è quella che prevede che la testa del programma che stiamo elaborando, sia una regola: la testa del programma però non è una regola ($p(a).$ è un fatto), quindi interrompiamo e torniamo indietro.

La terza strada dice di togliere la testa del programma e lavorare sul sotto programma ottenuto: $p(a).$ viene rimosso, lasciando $p(b).$. Eseguendo la funzione σ_F, scopriamo che l'unica sostituzione possibile è il set vuoto.

PROLOG restituerà `false`.

$$\begin{aligned}
&\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(b), \pi) \overset{1}{=} \sigma_F(p(b), p(a).) = \times \quad \texttt{mgu}(\{p(b), p(a)\}) = \perp \\
&\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(b), \pi) \overset{2}{=} \times \quad \text{non applicabile} \\
&\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(b), \pi) \overset{3}{=} \sigma^\pi_G(p(b), p(b).) \overset{1}{=} \sigma_F(p(b), p(b).) = \texttt{mgu}(\{p(b), p(b)\}) = \emptyset
\end{aligned}$$

Soluzione di altro programma $\text{PROLOG}_0{}^+$

In questo esempio, il goal che ci viene chiesto di verificare è ν :- $p(c).$, mentre il problema rimane lo stesso $\pi = p(a).\, p(b).$.

La σ^π_G ha come al solito 3 strade. Di queste, in modo identico all'esempio sopra, non possiamo applicare le prime due.

La terza strada prevede di scartare la testa e lavorare sul resto: scandiamo il programma dall'alto al basso. Il resto del programma è $p(b).$, quindi cerchiamo di soddisfare questo usando $p(c).$. Per farlo usiamo σ^π_G con 3 strade possibili, di nuovo:

- per il primo caso cerchiamo un unificatore unico tra i due, ma non ci riusciamo perché non ci sono variabili, restituiamo $\perp$;
- per la seconda strada si prevede che la testa sia una regola, ma quì abbiamo un fatto;
- per la terza strada si prevede che esista un resto del programma, togliendo la prima clausola, che però in questo caso se così facessimo, otterremmo $\perp$ e quindi non possiamo applicare nemmeno questa.

Per tutti gli altri casi, nulla è applicabile per le stesse motivazioni viste sopra. Nessun tipo di risultato viene prodotto.

PROLOG stamperà `false`.

$$\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(c),\pi) \overset{1}{=} \sigma_F(p(c),p(a).) = \times \quad \mathtt{mgu}(\{p(c),p(a)\}) = \bot$$

$$\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(c),\pi) \overset{2}{=} \times \quad \text{non applicabile}$$

$$\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(c),\pi) \overset{3}{=} \sigma^\pi_G(p(c),p(b).) \overset{1}{=} \sigma_F(p(c),p(b).) = \times \quad \mathtt{mgu}(\{p(c),p(b)\}) = \bot$$

$$\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(c),\pi) \overset{3}{=} \sigma^\pi_G(p(c),p(b).) \overset{2,3}{=} \times \quad \text{non applicabili}$$

$$\sigma^\pi(\nu) \overset{2}{=} \times \quad \text{non applicabile}$$

Soluzione di programma PROLOG$_0$$^+$ con variabile

In questo caso abbiamo delle varaibili. Il programma da cui partiamo $\pi = p(a).\,p(b).$ ha due clausole con lo scopo di soddisfare il goal ν :- $p(X)$..

Per soddisfare il goal calcoliamo $\sigma^\pi(\nu)$ che ricordiamo ha 2 strade possibili:

- la 1ª la prendiamo per elaborare la testa del goal;
- la 2ª la prendiamo se esiste un resto del goal.

In questo caso il resto del goal non è presente, quindi prendiamo solo la 1ª strada. Diciamo che il valore di $\sigma^\pi(\nu)$ è uguale al valore di σ^π_G applicato all'unico congiunto del goal presente e all'intero programma. La σ^π_G ha sempre quei 3 casi:

- la testa è un fatto;
- la testa è una regola;
- scarta la testa e lavora sul resto.

Il 1º caso è trattabile, il 2º no, il 3º anche lui trattabile: per mantenere l'ordine di scansione non deterministica, partiamo dalla prima possibilità (sempre così bisogna fare).

σ_F prevede che esista $\mathtt{mgu}$ e se così non è, quella strada non è più percorribile. Tecnicamente, la ricerca dell'unificatore non lo facciamo su $p(X)$, ma su una sua variante fresh: potenzialmente il goal contiene una X e il fatto contiene una X, ma quella del goal non è quella del fatto; se quello su cui andiamo a unificare non condivide variabili, possiamo fare finta non ci siano variabili fresh, ma se così non è allora devono essere diverse.

L'unica sostituzione che mappa $p(X)$ con $p(a)$ è X in a, la più piccola (potremmo usare l'algoritmo di Martelli-Montanari per esserne sicuri).

PROLOG fornirà risultato `X=a`.

$$\sigma^\pi(\nu) \overset{1}{=} \sigma^\pi_G(p(X),\pi) \overset{1}{=} \sigma_F(p(X),p(a).) = \mathtt{mgu}(\{p(X),p(a)\}) = \{a/X\}$$

Cliccando `Next` nell'esecutore, vedremo le altre possibilità.

Sappiamo già che la 2ª strada non è intraprendibile dall'inizio, quindi soltanto la 3ª è percorribile. Viene cercata una soluzione per il goal $p(X)$ usando il programma $p(b)$. Per farlo calcoliamo σ^π_G, con le sue 3 possibilità: soltanto la 1ª useremo, siccome la testa è un fatto.

σ_F va ad applicarsi a fatto $p(b)$, calcolando $\mathtt{mgu}$ scopriamo che il caso è uguale a quello di prima, dove l'unica mappatura più semplice è quella X in b.

PROLOG stampa `X=b`.

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(p(X), \pi) \stackrel{2}{=} \times \quad \text{non applicabile}$$
$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(p(X), \pi) \stackrel{3}{=} \sigma_G(p(X), p(b).) \stackrel{1}{=} \sigma_F(p(X), p(b).) = \texttt{mgu}(\{p(X), p(b)\}) = \{b/X\}$$

Soluzione di programma PROLOG$_0$$^+$ con variabili

Preso il programma $\pi = p(a).\, q(X) \text{ :- } p(X).$, vogliamo verificare il goal $\nu \text{ :- } q(X)$.

Per farlo applichiamo σ al programma e al goal $q(X)$ con 3 strade possibili: soltanto la 3ª è disponibile; eliminiamo $p(X)$. Facendo così, ci rimane da trovare $\texttt{mgu}$ tra $p(a)$ e $q(X)$, che non esiste perché nessuna sostituzione le renderà mai uguali ($\perp$).

La 2ª strada di σ^π_G prevede che la testa del programma sia un regola, ma non lo è, e quindi non l'applichiamo.

La 3ª strada di σ^π_G prevede che esista resto al programma, togliendo la testa: il resto c'è ma questa è una regola, quindi ci tocca usare σ_R.

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(q(X), \pi) \stackrel{1}{=} \sigma^\pi_F(q(X), p(a).) = \times \quad \texttt{mgu}(\{q(X), p(a)\}) = \perp$$
$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(q(X), \pi) \stackrel{2}{=} \times \quad \text{non applicabile}$$
$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(q(X), \pi) \stackrel{3}{=} \sigma^\pi_G(q(X), q(X) \text{ :- } p(X).) \stackrel{1}{=} \times \quad \text{non applicabile}$$
$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(q(X), \pi) \stackrel{3}{=} \sigma^\pi_G(q(X), q(X) \text{ :- } p(X).) \stackrel{2}{=} \sigma^\pi_R(q(X), q(X) \text{ :- } p(X).)$$

σ^π_R ha una strada unica che riusciamo ad applicare se esiste un $\texttt{mgu}$ tra la testa fornita ($q(X)$) e il congiunto del goal ($q(X)$) che stiamo cercando di soddisfare. L'unificatore esiste, tra varianti fresh della X, siccome questa è presente in tutte e due le parti: questo unificatore è una sostituzione che mappa $\{X/X1\}$ con $p(X)$.

$$\sigma^\pi_R(q(X), q(X) \text{ :- } p(X).) = \{X/X1\} \circ \sigma^\pi(\text{:- } p(X1)\{X/X1\}.) = \{X/X1\} \circ \sigma^\pi(\text{:- } p(X).)$$

σ^π_G viene poi applicato al risultato di σ_R, per la quale la 1ª strada è percorribile.

PROLOG stampa `a/X` (`a/X1` non viene stampata perché è servita solo per fare calcoli).

$$\sigma^\pi(\text{:- } p(X).) \stackrel{1}{=} \sigma^\pi_G(p(X), \pi) \stackrel{1}{=} \sigma_F(p(X), p(a).) = \texttt{mgu}(\{p(X), p(a)\}) = \{a/X\}$$

Programma PROLOG$_0$$^+$ in cui la computazione non termina

Questo caso deve esserci, perché altrimenti la terminazione e tante altre cose che si appoggiano all'informatica teorica, non sarebbero vere.

Un programma ci dice $\pi = q(X) \text{ :- } q(X).$ ($q(X)$ è vera se $q(X)$ è vera). Per come PROLOG ragiona, la σ^π non è calcolabile perché si comincia a richiedere ciclicamente la stessa cosa, senza terminare. Il goal è $\nu = \text{:- } q(X).$.

Cerchiamo di calcolare la σ^π, con goal a singolo congiunto, che prevede di usare la 1ª funzione soltanto: σ^π_G. La testa del programma è una regola, usiamo σ_R.

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(q(X), \pi) \stackrel{1}{=} \times \quad \text{non applicabile}$$
$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma^\pi_G(q(X), \pi) \stackrel{2}{=} \sigma^\pi_R(q(X), q(X) \text{ :- } q(X).)$$

σ_R prevede delle sostituzioni.
Il problema delle sostituzioni per ottenere variabili fresh, è che se applicate a sostituzione di altre variabili già fresh, queste non porteranno mai a terminazione, siccome esiste sempre una sostituzione che soddisfi.

$$\sigma^\pi_R(q(X), q(X) \text{ :- } q(X).) = \{X/X1\} \circ \sigma^\pi(\text{:- } q(X1)\{X/X1\}.) = \{X/X1\} \circ \sigma^\pi(\text{:- } q(X).)$$

09/05/2023