Computazione in Programmazione Logica e Prolog

Alberto Martelli

Intelligenza Artificiale e Laboratorio

Programmazione logica

Una *computazione* in programmazione logica corrisponde alla dimostrazione di una formula (goal) a partire dal programma logico applicando il principio di risoluzione con una particolare strategia detta **SLD** (risoluzione **L**ineare per clausole **D**efinite con funzione di **S**elezione)

La Risoluzione è un metodo di prova di teoremi che si applica a formule in forma di clausole e si basa su un'unica regola di inferenza.

Richiami su Risoluzione

Consideriamo per il momento clausole prive di variabili. Siano C_1 e C_2 due clausole del tipo:

$$C_1 = A_1 \vee \ldots \vee A_n$$
$$C_2 = B_1 \vee \ldots \vee B_m$$

Se ci sono in C_1 e C_2 due letterali A_i e B_j tali che $A_i = \neg B_j$, allora si può derivare da C_1 e C_2 la clausola

$$A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \ldots \vee A_n \vee B_1 \vee \ldots \vee B_{j-1} \vee B_{j+1} \ldots \vee B_m$$

detta **risolvente**.

Il risolvente di C_1 e C_2 è conseguenza logica di $C_1 \cup C_2$.

Richiami su Risoluzione

Dato un insieme di formule H e una formula F la dimostrazione che F segue logicamente da H viene fatta per refutazione, ossia dimostrando che $H \cup \neg F$ è inconsistente.

L'algoritmo generale di Risoluzione parte dall'insieme delle clausole ottenuta da H e $\neg F$ risolvendo ad ogni passo due clausole e aggiungendo il risolvente all'insieme delle clausole, finché non ottiene la clausola vuota ($\square$).

Per ridurre il numero dei risolventi generati, sono state proposte diverse strategie.

Risoluzione nella logica del prim'ordine

In questo caso le clausole possono contenere delle variabili. Consideriamo le clausole C_1 e C_2 che non hanno variabili in comune.

$$C_1 = A_1 \vee \ldots A_i \vee \ldots \vee A_n$$
$$C_2 = B_1 \vee \ldots B_j \vee \ldots \vee B_m$$

Supponiamo che $A_i = p(t_1, \ldots, t_k)$, $B_j = \neg p(t'_1, \ldots, t'_k)$ e $p(t_1, \ldots, t_k)$ e $p(t'_1, \ldots, t'_k)$ unifichino con MGU (unificatore più generale) θ (vedi slide su unificazione).

Allora il risolvente di C_1 e C_2 sarà:

$$[A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \ldots \vee A_n \vee B_1 \vee \ldots \vee B_{j-1} \vee B_{j+1} \ldots \vee B_m]\theta$$

Programmazione logica e Risoluzione

Nel caso delle clausole di Horn, un passo di risoluzione può essere riformulato come segue.

Sia C una clausola di Horn

$$A \leftarrow A_1, \ldots, A_n$$

e G un goal

$$\leftarrow B_1, \ldots, B_m$$

dove G e C non hanno variabili in comune.

Sia θ un MGU di A e B_i, per $1 \leq i \leq m$.

Allora il goal

$$\leftarrow [B_1, \ldots, B_{i-1}, A_1, \ldots, A_n, B_{i+1}, \ldots, B_m]\theta$$

è il risolvente di G e C.

Derivazione SLD

Una **derivazione SLD** per un goal G_0 da un insieme di clausole definite P è

- una sequenza di clausole goal $G_0, \ldots, G_n$,
- una sequenza di **varianti*** di clausole di P $C_1, \ldots, C_n$,
- una sequenza di MGU $\theta_1, \ldots, \theta_n$ tali che G_{i+1} è derivato da G_i e da C_{i+1} attraverso la sostituzione θ_{i+1}.

Ci sono tre possibili tipi di derivazioni:

- successo, se $G_n = \square$ (ovvero $G_n = \leftarrow$)
- fallimento finito, se non è possibile derivare da G_n alcun risolvente e $G_n \neq \square$
- fallimento infinito, se è sempre possibile derivare nuovi risolventi.

*clausole con variabili rinominate

Strategia SLD

La strategia di risoluzione SLD è **corretta** e **completa** per le clausole di Horn.

La strategia SLD ha due forme di non determinismo

- regola di calcolo per selezionare ad ogni passo l'atomo B_i del goal da unificare con la testa di una clausola,
- scelta di quale clausola utilizzare ad ogni passo

Regola di calcolo

Una regola di calcolo è una funzione che ha come dominio l'insieme dei goal e che per ogni goal seleziona un suo atomo.

Una regola di calcolo non influenza correttezza e completezza del dimostratore.

Alberi SLD

Data una regola di calcolo, è possibile rappresentare tutte le derivazioni con un **albero SLD**:

- ciascun nodo è un goal
- la radice è il goal G_0
- ogni nodo $\leftarrow A_1, \ldots, A_m, \ldots, A_k$, dove A_m è l'atomo selezionato dalla regola di calcolo, ha un figlio per ogni clausola $A \leftarrow B_1, \ldots, B_q$ tale che A e A_m sono unificabili con MGU θ. Il nodo figlio è etichettato con il goal $\leftarrow [A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k]\theta$. Il ramo dal padre al figlio è etichettato con θ e e la clausola selezionata.

Esempio

Si consideri il programma:

```
sum(0,X,X).                          CL1
sum(s(W),Y,s(K)):- sum(W,Y,K).       CL2
```

e il goal:

```
?- sum(W,0,0), sum(W,0,K).
```

Le prossime due slide mostrano l'albero SLD per le regole di calcolo leftmost e rightmost.

Regola di calcolo leftmost

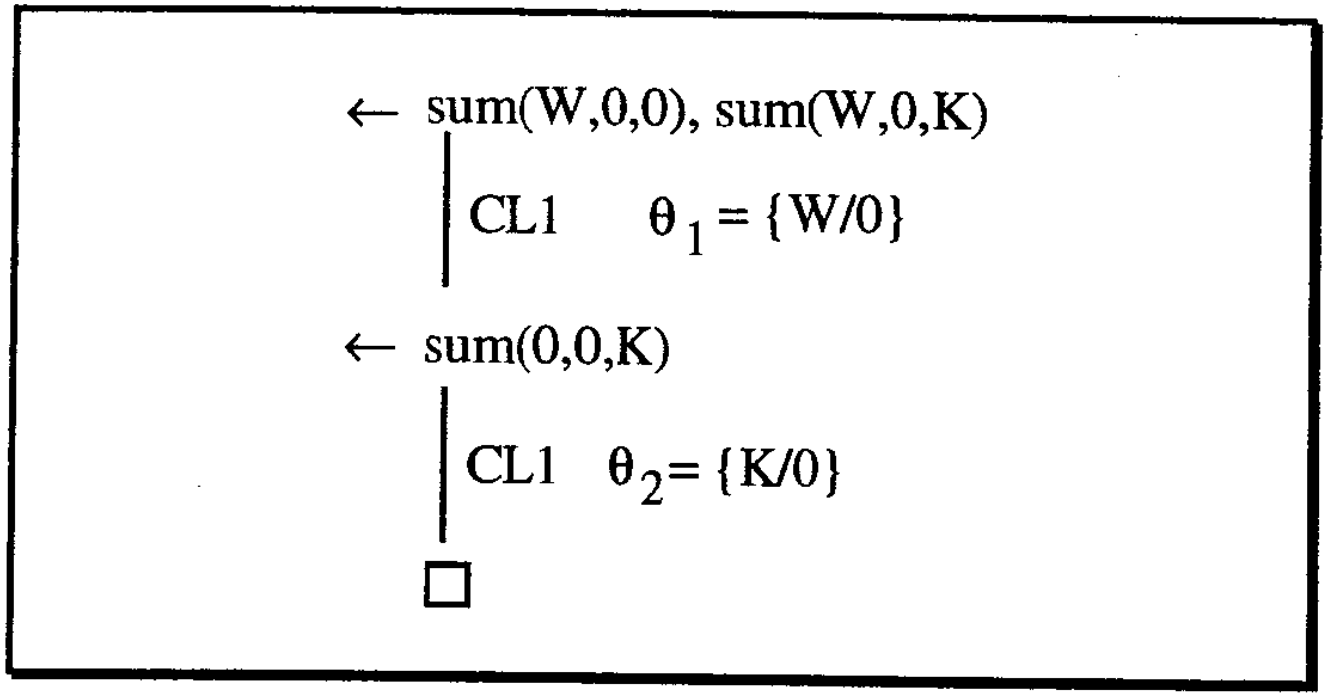

Figura 3.2: Albero SLD con regola di calcolo *left-most*

Regola di calcolo rightmost

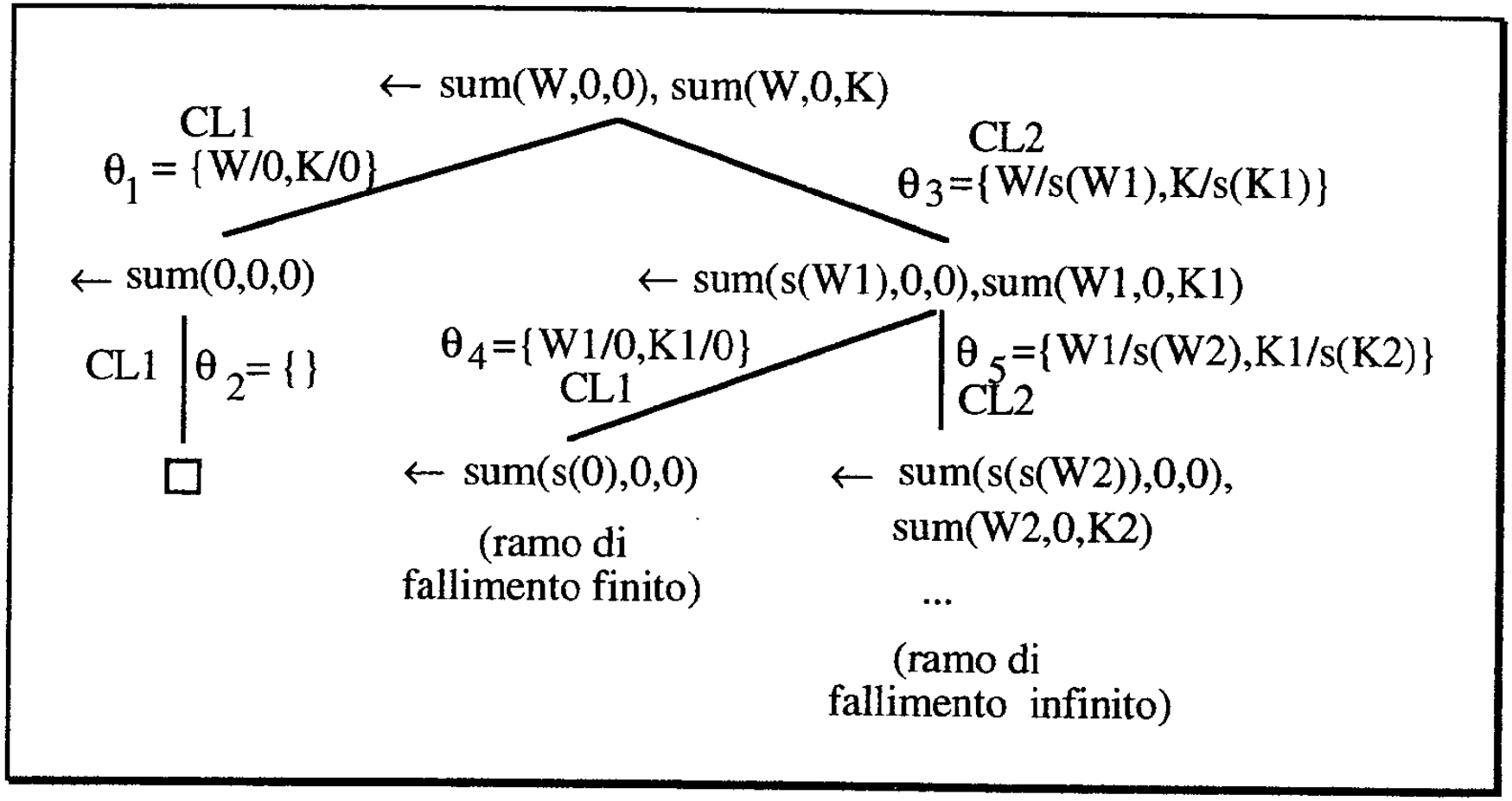

Figura 3.3: Albero SLD con regola di calcolo *right-most*

Esecuzione di un programma Prolog

Una computazione in Prolog corrisponde a una dimostrazione mediante risoluzione SLD.
Le scelte fatte dal Prolog sono:

- La regola di computazione è la regola leftmost,
- le clausole sono considerate nell'ordine in cui sono scritte nel programma,
- la strategia di ricerca usata è la ricerca in profondità con backtracking.

La strategia di ricerca non è completa. Infatti, se un ramo di derivazione di successo si trova a destra di un ramo infinito, quando l'interprete del Prolog entra nel ramo infinito non ne esce più e quindi non trova la derivazione di successo.

Prolog e unificazione

Il Prolog fornisce un predicato infisso built-in "=" che esegue l'unificazione dei suoi due operandi.
Ad esempio il goal

```
?- f(X,Y) = f(a,h(Z)), Z = b.
```

dà come risultato

```
X = a,
Y = h(b),
Z = b.
```

Tuttavia, per ragioni di efficienza, l'algoritmo di unificazione del Prolog non fa l'occur check per cui, ad esempio, considera come corretta l'unificazione di X con f(X).