1. Programmi Prolog che simulano programmi su macchine a registri

Definizione 1. *Una macchina ad r registri è una sequenza finita di istruzioni $I_1, \ldots, I_t$ che* operano *su una sequenza di r registri contenente i numeri $x_1, \ldots, x_r$, dove ogni istruzione I_m per $m < t$ è del tipo:*
sostituisci x_k con $x_k + 1$
oppure
se $x_k \neq 0$ allora sostituisci x_k con $x_k - 1$ e vai all'istruzione I_j.
L'istruzione I_t è sempre l'istruzione di STOP.
Il programma viene eseguito sequenzialmente a meno di isrtuzioni in altro senso.

Dato un programma $I = I_1, \ldots, I_t$ per una macchina a registri con r registri , l'input $n_1, \ldots, n_r$ e un indice $j \in \{1, \ldots, t\}$ definiremo un programma Prolog $\mathcal{P}[I, (n_1, \ldots, n_r), j]$ nel linguaggio predicativo $L = \{0, S, P_1, \ldots, P_t\}$, dove P_i sono simboli predicativi $r-ari$: ad ogni istruzione I_m di I corrisponderà una o più clausole del programma $\mathcal{P} = \mathcal{P}[I, (n_1, \ldots, n_r), j]$, nel seguente modo.

Se I_m è l'istruzione: *sostituisci x_k con $x_k + 1$*, allora $\mathcal{P}$ conterrà la clausola di programma

$$P_{m+1}(x_1, \ldots, x_{k-1}, S(y), \ldots, x_r) : -P_m(x_1, \ldots, x_{k-1}, y, \ldots, x_r);$$

se invece I_m è l'istruzione: *se $x_k \neq 0$ allora sostituisci x_k con $x_k - 1$ e vai all'istruzione I_j*, allora $\mathcal{P}$ conterrà le clausole di programma
$P_{m+1}(x_1, \ldots, x_{k-1}, 0, \ldots, x_r) : -P_m(x_1, \ldots, x_{k-1}, 0, \ldots, x_r)$
e
$P_j(x_1, \ldots, x_{k-1}, y, \ldots, x_r) : -P_m(x_1, \ldots, x_{k-1}, S(y), \ldots, x_r)$.

Infine, $\mathcal{P}$ conterrà il fatto

$$P_j(\overline{n}_1, \ldots, \overline{n}_r).$$

Teorema 1.

I termina sull'input $(n_1, \ldots, n_r)$ partendo dall'istruzione I_j con output $(m_1, \ldots, m_r)$

$$\Updownarrow$$

$$\mathcal{P}[I, (\overline{n_1}, \ldots, \overline{n_r}), j] \models P_t(\overline{m_1}, \ldots, \overline{m_r}).$$

Dimostrazione. ($\Rightarrow$) Per induzione sull'altezza h della computazione

$$(n_1, \ldots, n_r), I_{i_1} = I_j, (n'_1, \ldots, n'_r), I_{i_2}, \ldots, (m_1, \ldots, m_r), I_{i_h}$$

Se $h = 1$ allora $I_j = I_t$, $(n_1, \ldots, n_r) = (m_1, \ldots, m_r)$, e $P_t(\overline{m_1}, \ldots, \overline{m_r}) = P_j(\overline{n_1}, \ldots, \overline{n_r})$ è conseguenza logica di $\mathcal{P}[I, (\overline{n_1}, \ldots, \overline{n_r}), j]$. Nel passo induttivo, consideriamo il programma I che, fornito di input $(n'_1, \ldots, n'_r)$ a partire dall'istruzione I_{i_2} produce la computazione $(n'_1, \ldots, n'_r), I_{i_2}, \ldots, (m_1, \ldots, m_r), I_t$; per induzione sappiamo che

$$\mathcal{P}[I, (n'_1, \ldots, n'_r), i_2] \models P_t(\overline{m_1}, \ldots, \overline{m_r}).$$

Otterremo la conclusione cercata se dimostriamo che, qualsiasi sia l'istruzione I_j si ha

$$\mathcal{P}[I, (n_1, \ldots, n_r), j] \models P_{i_2}(\overline{n'_1}, \ldots, \overline{n'_r}).$$

Se I_j è l'istruzione *sostituisci* x_k *con* $x_k + 1$, allora $i_2 = j + 1$, $(n'_1, \ldots, n_k, \ldots, n'_r) = (n_1, \ldots, n_k + 1, \ldots, n_r)$ ed il programma $\mathcal{P}[I, (n_1, \ldots, n_r), j]$ conterrà la regola

$$P_{j+1}(x_1, \ldots, x_{k-1}, S(y), \ldots, x_r) : -P_j(x_1, \ldots, x_{k-1}, y, \ldots, x_r);$$

in particolare, si avrà

$$\mathcal{P}[I, (n_1, \ldots, n_r), j] \models P_{j+1}(\overline{n'_1}, \ldots, \overline{n'_r}),$$

che è quanto volevamo dimostrare.
Il caso in cui l'istruzione I_j è "*se* $x_k \neq 0$ *allora sostituisci* x_k *con* $x_k - 1$ *e vai all'istruzione* I_h viene lasciato come esercizio.

($\Leftarrow$) Sia M la seguente interpretazione del linguaggio di $\mathcal{P}[I, (n_1, \ldots, n_r), j]$: il dominio e l'interpretazione di $0, S$ sono dati dai numeri naturali con le usuali interpretazioni di 0 e S, mentre

$$(k_1, \ldots, k_r) \in P_i^M$$
$$\Updownarrow$$

$(k_1, \ldots, k_r)$ è raggiungibile partendo dall'input $(n_1, \ldots, n_r)$, iniziando la computazione con l'istruzione I_j, in modo che, una volta arrivati a $(k_1, \ldots, k_r)$, l'istruzione seguente sia I_i.

Si vede facilmente che $M \models \mathcal{P}[I, (n_1, \ldots, n_r), j]$. Quindi dall'ipotesi segue

$$M \models P_t(\overline{m_1}, \ldots, \overline{m_r}),$$

e $(m_1, \ldots, m_r)$ è l'output che si ottiene partendo da I_j su input $(n_1, \ldots, n_r)$.

Nel prossimo teorema faremo uso del risultato seguente (per una sua dimostrazione, consultare un libro di teoria della computabilità, ad esempio Cutland " Computability: An introduction to Recursive function Theory" Cambridge University Press.)

Non esiste alcun algoritmo che stabilisca, dato un programma per macchina a registri I ed $(n_1, \ldots, n_r)$, se I si ferma partendo dall'input $(n_1, \ldots, n_r)$.

Teorema 2. *(Teorema di Church)*
Il problema della soddisfacibilità per enunciati della logica predicativa è indecidibile.

Dimostrazione. Facciamo vedere come sia possibile ridurre in modo effettivo il problema della fermata di un programma di una macchina a registri su un dato input alla soddisfacibilità di un enunciato della logica predicativa.

Dato I, l'input $(n_1, \ldots, n_r)$ e un indice $j \in \{1, \ldots, r\}$ sia $\mathcal{P}[I, (n_1, \ldots, n_r), j]$ il programma Prolog costruito come nel teorema precedente.

Facciamo vedere che vale:

$$I \text{ termina sull'input } (n_1, \ldots, n_r) \text{ partendo dall'istruzione } I_1$$
$$\Updownarrow$$
$$\mathcal{P}[I, (n_1, \ldots, n_r), j] \cup \{\neg \exists \vec{x} P_t(\vec{x})\} \text{ è insoddisfacibile.}$$

(o equivalentemente: $\mathcal{P}[I, (n_1, \ldots, n_r), j] \models \exists \vec{x} P_t(\vec{x})$.)

Dalle proprietà dei programmi prolog (vedi lezione sul modello minimo, corollario 1) si ha

$$\mathcal{P}[I,(n_1,\ldots,n_r),j] \models \exists \vec{x} P_t(\vec{x}) \Leftrightarrow \exists \vec{s} \in T_c \; \mathcal{P}[I,(n_1,\ldots,n_r),j] \models P_t(\vec{s}).$$

Poiché $T_c = \{\overline{0}, \overline{1}, \ldots, \overline{m}, \ldots\}$ ne segue:

$$\mathcal{P}[I,(n_1,\ldots,n_r),j] \models \exists \vec{x} P_t(\vec{x})$$
$$\Updownarrow$$
$$\text{esistono } m_1,\ldots,m_r \text{ tali che } \mathcal{P}[I,(n_1,\ldots,n_r),j] \models P_t(\overline{m_1},\ldots,\overline{m_r}),$$

e dal teorema 1 abbiamo

$$\mathcal{P}[I,(n_1,\ldots,n_r),j] \models \exists \vec{x} P_t(\vec{x})$$
$$\Updownarrow$$
$$I \text{ termina sull'input } (n_1,\ldots,n_r).$$

2. Esercizi

(1) Siano $g_1,\ldots,g_n,h$ funzioni da $\mathbb{N}^r$ in $\mathbb{N}$, $h:\mathbb{N}^n \to \mathbb{N}$ e $f:\mathbb{N}^r \to \mathbb{N}$ definita da

$$f(x_1,\ldots,x_r) = h(g_1(x_1,\ldots,x_r),\ldots,g_n(x_1,\ldots,x_r)).$$

Supponiamo di avere simboli predicativi $p_{g_1},\ldots,p_{g_n},p_h,p_f$ per ogni funzione suddetta (dove p_{g_1} è un simbolo $r+1$-ario e cosí via).

Supponiamo che esistano programmi prolog $P_{g_1},\ldots,P_{g_n},P_f$ che definiscono le funzioni $g_1,\ldots,g_n,h$ (ad esempio nel caso di g_1 questo significa che

$$g_1(n_1,\ldots,n_r) = n \qquad \Leftrightarrow P_{g_1} \cup \{: -p_{g_1}(\overline{n_1},\ldots\overline{n_r},\overline{n})\} \text{ ha una } SLD \text{ refutazione,}$$

Utilizzare tali programmi per scrivere un programma P_f che definisca f.