Esame del corso di Verifica di Programmi Prolog Reali

Soluzione dell'esercizio 6 della "Prolog Competition" dell'ILPS'94

Marco Comini e **Enea Zaffanella**
Dipartimento di Informatica
Corso Italia, 40 – 56125 Pisa
{ comini, zaffanel }@di.unipi.it

14 marzo 1996

Sommario

Mostriamo un programma per risolvere l'esercizio 6 della "Prolog Competition" dell'ILPS'94 e ne mostriamo la correttezza parziale e la terminazione.

1 Il Programma

Assunzioni Preliminari. Per semplificare la notazione rispetto al testo rappresentiamo ogni casella (x, y) rimossa con il fatto $goto(x, y, \mathtt{b})$ anziché con l'assenza di un fatto tipo $goto(x, y, m)$ (con $m \in \{\, \mathtt{l}, \mathtt{r}, \mathtt{u}, \mathtt{d}, \mathtt{f} \,\}$).

Inoltre indeboliamo la specifica del testo dell'esercizio eliminando alcune ipotesi. Siccome non ci interessa garantire l'unicità del percorso corretto, ammettiamo che due caselle adiacenti sul cammino siano bianche e che eventualmente più di due caselle adiacenti ad una casella bianca appartengano ad un percorso corretto. Assumiamo che siano invece soddisfatte tutte le altre ipotesi del testo.

Il programma che risolve questo problema è il seguente.

```
path(Path) :-
    path([(1,1)], Path).

path([(C,R)|SoFar], Path) :-
    goto(C, R, f),
    reverse([(C,R)|SoFar], Path).
path([(C,R)|SoFar], Path) :-
    size(Size),
    getMove((C,R), Move),
    nextPos((C,R), Move, Size, (NextC,NextR)),
    noCycle([(C,R)|SoFar], (NextC,NextR)),
    path([(NextC,NextR),(C,R)|SoFar], Path).
```

```
getMove((C,R), Move) :-
    goto(C, R, Move).
getMove((C,R), Move) :-
    goto(C, R, b),
    guessMove(Move).

guessMove(l).
guessMove(d).
guessMove(r).
guessMove(u).

nextPos((C,R), u, Size, (C,NextR)) :-
    R < Size,
    NextR is R + 1.

nextPos((C,R), d, Size, (C,NextR)) :-
    R > 1,
    NextR is R - 1.

nextPos((C,R), r, Size, (NextC,R)) :-
    C < Size,
    NextC is C + 1.

nextPos((C,R), l, Size, (NextC,R)) :-
    C > 1,
    NextC is C - 1.

noCycle([], Next).
noCycle([X|Rest], Next) :-
    Next \== X,
    noCycle(Rest, Next).

reverse(Xs, Ys) :-
    reverse(Xs, [], Ys).

reverse([], Acc, Acc).
reverse([First|Rest], Acc, Ys) :-
    reverse(Rest, [First|Acc], Ys).
```

A questo programma vanno aggiunte le clausole che definiscono la particolare mappa, e cioè un unico fatto $size(n)$ che definisce la misura della mappa e un insieme di fatti del tipo $goto(x, y, m)$ ($m \in \{\, \mathtt{l}, \mathtt{r}, \mathtt{u}, \mathtt{d}, \mathtt{f}, \mathtt{b} \,\}$).

Il nostro programma fornisce (in backtracking) tutti i percorsi corretti contenuti nella mappa. In particolare, nel caso in cui la mappa soddisfi anche le due ipotesi rimosse dalla specifica originale, si ottiene l'unico percorso corretto.

Esempio 1.1 Aggiungendo al programma sui seguenti fatti

```
size(5).
```

```
goto(1, 1, b).  goto(2, 1, d).  goto(3, 1, d).
goto(4, 1, f).  goto(5, 1, l).

goto(1, 2, u).  goto(2, 2, r).  goto(3, 2, b).
goto(4, 2, l).  goto(5, 2, b).

goto(1, 3, b).  goto(2, 3, d).  goto(3, 3, u).
goto(4, 3, d).  goto(5, 3, d).

goto(1, 4, d).  goto(2, 4, u).  goto(3, 4, l).
goto(4, 4, d).  goto(5, 4, b).

goto(1, 5, d).  goto(2, 5, b).  goto(3, 5, r).
goto(4, 5, b).  goto(5, 5, d).
```

che corrispondono alla mappa

	↓		→		↓
	↓	↑	←	↓	
		↓	↑	↓	↓
	↑	→		←	
$\Longrightarrow$		↓	↓	•	←

otteniamo come output

```
| ?- path(X).

X = [(1,1), (1,2), (1,3), (2,3), (2,2), (3,2), (3,3),
(3,4), (2,4), (2,5), (3,5), (4,5), (5,5), (5,4), (5,3),
(5,2), (5,1), (4,1)] ?  ;

no
```

che corrisponde all'unico cammino corretto

		→	→	→	↓
		↑	←		↓
	→	↓	↑		↓
	↑	→	↑		↓
$\Longrightarrow$	↑			•	←

■

Esempio 1.2 Modificando l'Esempio 1.1 con i fatti `goto(1, 2, b)` e `goto(3, 4, b)` al posto di `goto(1, 2, u)` e `goto(3, 4, l)` che corrispondono alla mappa

	↓		→		↓
	↓	↑	·	↓	
		↓	↑	↓	↓
	·	→		←	
$\Longrightarrow$		↓	↓	•	←

otteniamo invece come output

```
| ?- path(X).

X = [(1,1), (1,2), (2,2), (3,2), (3,3), (3,4), (2,4),
(2,5), (3,5), (4,5), (5,5), (5,4), (5, 3), (5,2), (5,1),
(4,1)] ?  ;

X = [(1,1), (1,2), (2,2), (3,2), (3,3), (3,4), (3,5),
(4,5), (5,5), (5,4), (5,3), (5,2), (5, 1), (4,1)] ?  ;

X = [(1,1), (1,2), (1,3), (2,3), (2,2), (3,2), (3,3),
(3,4), (2,4), (2,5), (3,5), (4,5), (5, 5), (5,4), (5,3),
(5,2), (5,1), (4,1)] ?  ;

X = [(1,1), (1,2), (1,3), (2,3), (2,2), (3,2), (3,3),
(3,4), (3,5), (4,5), (5,5), (5,4), (5, 3), (5,2), (5,1),
(4,1)] ?  ;

no
```

■

2 Analisi del programma

2.1 Modi Input/Output, Tipi, Occur-Check Freeness e Errori a Run-time

Definiamo i tipi

$$\begin{aligned} Pos &= \{\, (x, y) \mid x, y \in Int \,\} \\ Move &= \{\, \mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r}, \mathtt{f}, \mathtt{b} \,\} \\ List &= \text{liste di elementi qualsiasi} \\ PosList &= \text{liste di elementi di } Pos \\ Gae &= \text{Ground arithmetic expressions} \end{aligned}$$

Date le seguenti definizioni di modi e di tipo per i predicati del programma

path(Out:PosList)
path(In:PosList, Out:PosList)
getMove(In:Pos, Out:Move)
guessMove(Out:Move)
nextPos(In:Pos, In:Move, In:Int, Out:Pos)
noCycle(In:PosList, In:Pos)

reverse(In:List, Out:List)
reverse(In:List, In:List, Out:List)
size(Out:Int)
goto(Out:Int, Out:Int, Out:Move)
\==(In:Int, In:Int)
>(In:Int, In:Int)
<(In:Int, In:Int)
is(Out:Int, In:Gae)

si dimostra facilmente che il programma (rispetto al goal $path(Path)$) è ben modato e ben tipato.

Proposizione 2.1 *Programma e goal sono "occur-check free".*

Dimostrazione. Il programma e il goal sono ben modati. Inoltre tutte le teste di clausole di P sono "output linear". Possiamo quindi applicare il Corollario 4.5 di [1] da cui discende la tesi. ■

Proposizione 2.2 *Programma e goal sono liberi da errori run-time.*

Dimostrazione. Le chiamate a predicati built-in che vengono fatte dal programma nel risolvere il goal rispettano le modalità e i tipi ammessi. ■

2.2 Correttezza Parziale e Terminazione

Introduciamo ora un po' di terminologia e notazioni che ci serviranno per dimostrare la correttezza parziale del programma. Con lettere minuscole denotiamo variabili a valori nell'Universo di Herbrand. Con λ denotiamo la lista vuota, mentre se x è una lista, con $e \cdot x$ denotiamo la lista ottenuta aggiungendo e in testa ad x. Date due liste x, y con $\text{concat}(x, y)$ denotiamo la lista che si ottiene concatenando x e y. Se $x = e_1 \dots e_n$ è una lista, con $\text{reverse}(x)$ denotiamo la lista $e_n \dots e_1$, $|x|$ denota la lunghezza n di x, mentre $\text{prefix}(x)$ denota l'insieme $\{ e_1 \dots e_i \mid i \leq n \}$. Inoltre diciamo che $e \in x$ se e occorre in x.

Data $E \in Gae$, con $\text{eval}(E)$ denotiamo il termine in Int che corrisponde alla valutazione di E.

D'ora in poi N denoterà la dimensione della mappa definita dal fatto `size(N)` $\in P$.

Definizione 2.3 *Una coppia (c, r) è* ammissibile *se $1 \leq c, r \leq N$. L'insieme delle coppie ammissibili è $CA \subseteq Pos$.*

Assumiamo inoltre che sia definita una funzione $goto : CA \longrightarrow Move$ che ad ogni coppia associa la corrispondente mossa presente sulla scacchiera.

Definiamo una relazione $\xrightarrow{TA}$ per mettere in corrispondenza coppie (ammissibili) del percorso sotto esame. $\xrightarrow{TA}$ è data dall'unione delle relazioni $\xrightarrow[\mathtt{u}]{TA}$, $\xrightarrow[\mathtt{d}]{TA}$, $\xrightarrow[\mathtt{r}]{TA}$ e $\xrightarrow[\mathtt{l}]{TA}$ definite come segue.

$$\frac{(c, r) \in CA,\ (goto(c, r, \mathtt{u}) \in P \vee goto(c, r, \mathtt{b}) \in P),\ (c, r+1) \in CA}{(c, r) \xrightarrow[\mathtt{u}]{TA} (c, r+1)}$$

$$\frac{(c,r)\in CA,\ (goto(c,r,\mathtt{d})\in P \vee goto(c,r,\mathtt{b})\in P),\ (c,r-1)\in CA}{(c,r)\xrightarrow[\mathtt{d}]{TA}(c,r-1)}$$

$$\frac{(c,r)\in CA,\ (goto(c,r,\mathtt{r})\in P \vee goto(c,r,\mathtt{b})\in P),\ (c+1,r)\in CA}{(c,r)\xrightarrow[\mathtt{r}]{TA}(c+1,r)}$$

$$\frac{(c,r)\in CA,\ (goto(c,r,\mathtt{l})\in P \vee goto(c,r,b)\in P),\ (c-1,r)\in CA}{(c,r)\xrightarrow[\mathtt{l}]{TA}(c-1,r)}$$

Definizione 2.4 *Un* percorso parziale *è una lista* $\sigma_1\cdots\sigma_n$ *senza ripetizioni di coppie ammissibili tale che* $\sigma_1=(1,1)$ *e* $\forall i\in\{1,\ldots,n-1\}\ \sigma_i\xrightarrow{TA}\sigma_{i+1}$. $PPD\subseteq PosList$ *è l'insieme dei percorsi parziali. Inoltre*

$$PPR=\{\,x\mid \mathrm{reverse}(x)\in PPD\}$$

Un percorso parziale è detto percorso finale *se, posto che* (c,r) *sia il suo ultimo elemento, abbiamo* $goto(c,r)=\mathtt{f}$. *L'insieme dei percorsi finali è denotato* PF.

Lemma 2.5 *Se* $x\in PPR$ *allora* $|x|\le N^2$.

Dimostrazione. x è una lista di elementi di CA senza ripetizioni e $\mathrm{card}(CA)=N^2$. ∎

Per definizione, tutte le clausole unitarie sono ricorrenti. Inoltre, gli unici predicati ricorsivi sono path/2, noCycle/2, reverse/3 e non si hanno predicati mutuamente ricorsivi. Nel seguito ometteremo la dimostrazione di terminazione quando risulti essere banale.

Definiamo i seguenti level mapping:

$$\begin{aligned}
|path(x)| &= 2N^2\\
|path(x,y)| &= 2N^2-|x|\\
|noCycle(x,p)| &= |x|\\
|reverse(x,y)| &= |x|\\
|reverse(x,y,z)| &= |x|
\end{aligned}$$

Gli atomi $getMove(x,m)$, $guessMove(m)$, $nextPos(x,m,y)$, $goto(c,r,m)$ e $size(n)$ che i built-in (`is`, `>`, `<`, `\==`) sono invece mappati a zero.

Definiamo ora Pre e $Post$ per i predicati del programma.

$$path(T)\begin{cases} Pre=\{\,path(x)\,\}\\ Post=\{\,path(x)\mid x\in PF\,\}\end{cases}$$

$path(T_1,T_2)$

$$\begin{cases} Pre=\{\,path(t_1,t_2)\mid t_1\in PPR\}\\ Post=\{\,path(t_1,t_2)\mid t_1\in PPR, t_2\in PF, \mathrm{reverse}(t_1)\in \mathrm{prefix}(t_2)\,\}\end{cases}$$

$getMove(C, M)$

$$\begin{cases} Pre = \{\, getMove(p, m) \,\} \\ Post = \{\, getMove((c, r), m) \mid \\ \qquad\qquad (c, r) \in CA, \\ \qquad\qquad m = \text{goto}(c, r) \vee (\text{goto}(c, r) = \mathtt{b} \wedge m \in \{\, \mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r} \,\}) \,\} \end{cases}$$

$$guessMove(M) \begin{cases} Pre = \{\, guessMove(m) \,\} \\ Post = \{\, guessMove(m) \mid m \in \{\, \mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r} \,\} \,\} \end{cases}$$

$nextPos(C_1, M, Size, C_2)$

$$\begin{cases} Pre = \{\, nextPos(c_1, m, N, c_2) \mid c_1 \in CA \,\} \\ Post = \{\, nextPos(c_1, m, N, c_2) \mid c_1, c_2 \in CA, c_1 \xrightarrow[\mathtt{m}]{TA} c_2 \,\} \end{cases}$$

$$noCycle(L, E) \begin{cases} Pre = \{\, noCycle(x, e) \,\} \\ Post = \{\, noCycle(l, e) \mid l \in List, e \notin l \,\} \end{cases}$$

$$reverse(Xs, Ys) \begin{cases} Pre = \{\, reverse(x, y) \,\} \\ Post = \{\, reverse(x, y) \mid x, y \in List, y = \text{reverse}(x) \,\} \end{cases}$$

$reverse(Xs, Acc, Ys)$

$$\begin{cases} Pre = \{\, reverse(x, y, z) \mid y \in List \,\} \\ Post = \{\, reverse(x, y, z) \mid x, y, z \in List, z = \text{concat}(\text{reverse}(x), y) \,\} \end{cases}$$

$$size(N) \begin{cases} Pre = \{\, size(n) \,\} \\ Post = \{\, size(N) \,\} \end{cases}$$

$$goto(C, R, M) \begin{cases} Pre = \{\, goto(c, r, m) \,\} \\ Post = \{\, goto(c, r, m) \mid (c, r) \in CA, m \in Move \,\} \end{cases}$$

$$> (X, Y) \begin{cases} Pre = \{\, > (x, y) \,\} \\ Post = \{\, > (x, y) \mid x, y \in Int, x > y \,\} \end{cases}$$

$$< (X, Y) \begin{cases} Pre = \{\, < (x, y) \,\} \\ Post = \{\, < (x, y) \mid x, y \in Int, x < y \,\} \end{cases}$$

$$is(X, Y) \begin{cases} Pre = \{\, is(x, y) \,\} \\ Post = \{\, is(x, y) \mid x \in Int, y \in Gae, x = \text{eval}(y) \,\} \end{cases}$$

$$\backslash{==}\, (X, Y) \begin{cases} Pre = \{\, \backslash{==}\, (x, y) \,\} \\ Post = \{\, \backslash{==}\, (x, y) \mid x \neq y \,\} \end{cases}$$

Teorema 2.6 *Il programma è parzialmente corretto e termina*[1].

[1] Bisogna sempre avere almeno un teorema nella vita.

Dimostrazione. Per goto/3, guessMove/1, size/1 la dimostrazione è immediata. I built-in sono parzialmente corretti per definizione. Forniamo ora i "proof outlines" per le restanti clausole [2].

$$
\begin{array}{l}
\{\, path(x) \,\} \\
path(Path) :- \qquad \langle t_0 = 2N^2 \rangle \\
\qquad \{\, path((1,1)\cdot\lambda, x) \,\} \qquad \%\%(1,1)\cdot\lambda \in PPR\%\% \\
\quad path([(1,1)], Path). \qquad \langle t_1 = 2N^2 - |[(1,1)]| = 2N^2 - 1 \rangle \\
\qquad \{\, path((1,1)\cdot\lambda, x) \mid x \in PF \,\} \\
\{\, path(x) \mid x \in PF \,\}
\end{array}
$$

$$
\begin{array}{l}
\{\, path((c,r)\cdot x, y) \mid (c,r)\cdot x \in PPR \} \\
path([(C,R)|SoFar], Path) :- \qquad \langle t_0 = 2N^2 - |[(C,R)|SoFar]| \geq N^2 \rangle \\
\qquad \{\, goto(c, r, \mathtt{f}) \,\} \\
\quad goto(C, R, f), \qquad \langle t_1 = 0 \rangle \\
\qquad \{\, goto(c, r, \mathtt{f}) \mid (c,r) \in CA, \mathrm{goto}(c,r) = \mathtt{f} \,\} \\
\qquad \{\, reverse((c,r)\cdot x, y) \,\} \\
\quad reverse([(C,R)|SoFar], Path). \qquad \langle t_2 = |[(C,R)|SoFar]| \leq N^2 \leq t_0 \,\} \\
\qquad \{\, reverse((c,r)\cdot x, y) \mid y = \mathrm{reverse}((c,r)\cdot x) \,\} \\
\{\, path((c,r)\cdot x, y) \mid (c,r)\cdot x \in PPR, y \in PF, y = \mathrm{reverse}((c,r)\cdot x) \,\}
\end{array}
$$

$\{\, path((c,r)\cdot x, y) \mid (c,r)\cdot x \in PPR \,\}$

$path([(C,R)|SoFar], Path) :-$ $\quad \langle t_0 = 2N^2 - |[(C,R)|SoFar]| \geq N^2 \rangle$

$\quad \{\, size(n) \,\},$

$\quad size(Size),$ $\quad \langle t_1 = 0 \rangle$

$\quad \{\, size(N) \,\},$

$\quad \{\, getMove((c,r), m) \,\}$

$\quad getMove((C,R), Move),$ $\quad \langle t_2 = 0 \rangle$

$\quad \{\, getMove((c,r), m) \mid (c,r) \in CA, m = \mathrm{goto}(c,r) \vee (\mathrm{goto}(c,r) = \mathtt{b} \wedge m \in \{\, \mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r} \,\}) \,\}$

$\quad \{\, nextPos((c,r), m, N, (c',r')) \mid (c,r) \in CA \,\}$

$\quad nextPos((C,R), Move, Size, (NextC, NextR)),$ $\quad \langle t_3 = 0 \rangle$

$\quad \{\, nextPos((c,r), m, N, (c',r')) \mid (c,r), (c',r') \in CA, (c,r) \xrightarrow[\mathtt{m}]{TA} (c',r') \,\}$

$\quad \{\, noCycle((c,r)\cdot x, (c',r')) \,\}$

$\quad noCycle([(C,R)|SoFar], (NextC, NextR)),$

$\quad\quad \langle t_4 = |[(C,R)|SoFar]| \leq N^2 \leq t_0 \rangle$

$\quad \{\, noCycle((c,r)\cdot x, (c',r')) \mid (c,r)\cdot x \in List, (c',r') \notin (c,r)\cdot x \,\}$

$\quad \{\, path((c',r')\cdot(c,r)\cdot x, y) \mid (c',r')\cdot(c,r)\cdot x \in PPR \,\}$

$\quad path([(NextC, NextR), (C,R)|SoFar], Path).$

$\quad\quad \langle t_5 = 2N^2 - (|SoFar| + 2) = t_0 - 1 \rangle$

$\quad \{\, path((c',r')\cdot(c,r)\cdot x, y) \mid (c',r')\cdot(c,r)\cdot x \in PPR, y \in PF, \mathrm{reverse}((c',r')\cdot(c,r)\cdot x) \in \mathrm{prefix}(y) \,\}$

$\{\, path((c,r)\cdot x, y) \mid (c,r)\cdot x \in PPR, y \in PF, \mathrm{reverse}((c,r)\cdot x) \in \mathrm{prefix}(y) \,\}$

$\{\, getMove((c,r), m) \,\}$

$getMove((C,R), Move) :-$

$\quad \{\, goto(c, r, m) \,\}$

$\quad goto(C, R, Move).$

$\quad \{\, goto(c, r, m) \mid (c,r) \in CA, m = \mathrm{goto}(c,r) \,\}$

$\{\, getMove((c,r), m) \mid (c,r) \in CA, m = \mathrm{goto}(c,r) \,\}$

$\{\, getMove((c,r),m) \,\}$
$getMove((C,R),Move) :-$
$\quad \{\, goto(c,r,\mathtt{b}) \,\}$
$\quad goto(C,R,b),$
$\quad \{\, goto(c,r,\mathtt{b}) \mid (c,r) \in CA, \mathrm{goto}(c,r) = \mathtt{b} \,\}$
$\quad \{\, guessMove(m) \,\}$
$\quad guessMove(Move).$
$\quad \{\, guessMove(m) \mid m \in \{\, \mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r} \,\} \,\}$
$\{\, getMove((c,r),m) \mid (c,r) \in CA, \mathrm{goto}(c,r) = \mathtt{b}, m \in \{\, \mathtt{u}, \mathtt{d}, \mathtt{l}, \mathtt{r} \,\} \,\}$

$\{\, nextPos((c,r),\mathtt{u},N,(c,r')) \mid (c,r) \in CA \,\}$
$nextPos((C,R),u,Size,(C,NextR)) :-$
$\quad \{\, <(r,N) \,\}$
$\quad R < Size,$
$\quad \{\, <(r,N) \mid r < N \,\}$
$\quad \{\, is(r', +(r,1)) \,\}$
$\quad NextR \ is \ R+1.$
$\quad \{\, is(r', +(r,1)) \mid r' \in Int, +(r,1) \in Gae, r' = \mathrm{eval}(+(r,1)) = r+1 \,\}$
$\{\, nextPos((c,r),\mathtt{u},N,(c,r')) \mid (c,r),(c,r') \in CA, (c,r) \xrightarrow[\mathtt{u}]{TA} (c,r') \,\}$

Per le altre tre clausole con `d`, `l`, `r` si procede analogamente.

$\{\, noCycle(\lambda,p) \,\}$
$noCycle([],Next).$
$\{\, noCycle(\lambda,p) \,\}$

$\{\, noCycle(e \cdot x,p) \,\}$
$noCycle([X|Rest],Next) :- \qquad \langle t_0 = |Rest|+1 \rangle$
$\quad \{\, \backslash{==}(e,p) \,\}$
$\quad Next \backslash{==} X, \qquad \langle t_1 = 0 \rangle$
$\quad \{\, \backslash{==}(e,p) \mid e \neq p \,\}$
$\quad \{\, noCycle(x,p) \,\}$
$\quad noCycle(Rest,Next). \qquad \langle t_2 = |Rest| = t_0 - 1 \,\rangle$
$\quad \{\, noCycle(x,p) \mid p \notin x \,\}$
$\{\, noCycle(e \cdot x,p) \mid p \notin e \cdot x \,\}$

$$
\begin{array}{l}
\{\, reverse(x,y)\,\} \\
reverse(Xs, Ys) :- \qquad \langle t_0 = |Xs| \rangle \\
\qquad \{\, reverse(x,a,y) \mid a \in List\,\} \\
\quad reverse(Xs, [\,], Ys). \qquad \langle t_1 = |Xs| = t_0 \rangle \\
\qquad \{\, reverse(x,\lambda,y) \mid x,y \in List, y = \mathrm{concat}(\mathrm{reverse}(x), \lambda)\,\} \\
\{\, reverse(x,y) \mid x,y \in List, y = \mathrm{reverse}(x)\,\}
\end{array}
$$

$$
\begin{array}{l}
\{\, reverse(\lambda,z,y) \mid z \in List\,\} \\
reverse([\,], Acc, Acc). \\
\{\, reverse(\lambda,z,y) \mid y,z \in List, y = \mathrm{concat}(\mathrm{reverse}(\lambda), z) = z\,\}
\end{array}
$$

$$
\begin{array}{l}
\{\, reverse(e \cdot x,z,y) \mid z \in List\,\} \\
reverse([First|Rest], Acc, Ys) :- \qquad \langle t_0 = |Rest| + 1 \rangle \\
\qquad \{\, reverse(x,e \cdot z,y) \mid e \cdot z \in List\,\} \\
\quad reverse(Rest, [First|Acc], Ys). \qquad \langle t_1 = |Rest| = t_0 - 1 \rangle \\
\qquad \{\, reverse(x,e \cdot z,y) \mid x,y,z \in List, y = \mathrm{concat}(\mathrm{reverse}(x), e \cdot z)\,\} \\
\{\, reverse(e \cdot x,z,y) \mid e \cdot x,y,z \in List, y = \mathrm{concat}(\mathrm{reverse}(e \cdot x), z)\,\}
\end{array}
$$

■

Riferimenti bibliografici

[1] K. R. Apt and A. Pellegrini. On the occur-check free prolog programs. Technical Report CS-R9238, CWI, October 1992.

[2] D. Pedreschi and S. Ruggieri. A case study in logic program verification: the `vanilla` metainterpreter. In *Proceedings of the GULP-PRODE'95 Joint Conference on Declarative Programming*, 1995.