Lezione 23-24

April 16, 2012

Logic Programming Fundamentals

- Semantica: Resolution e Programmazione Deduttiva
- PROLOG: Sintassi e Semantica
- Spazio di ricerca e Nondeterminismo
- Variabili Logiche, Invertibilità e Dichiaratività
- Metodologie: Programmazione Deduttiva
- Meccanismi Addizionali: Is, if-then-else.
- Meccanismi Addizionali: cut
- Estensioni: Negation as Failure
- Data Base and Constraint Programming

Semantica: Risoluzione

Example

$\mathtt{P} \equiv \{\mathtt{add(0,0,0)}; \mathtt{add(s(X),Y,s(Z))} :\!-\mathtt{add(X,Y,Z)}\}$

$\mathtt{G} \equiv \mathtt{add(X,Y,s(0))}$ [ovvero la formula $\neg\exists(\mathtt{X},\mathtt{Y})\mathtt{add(X,Y,s(0))}$]

R(C1,L1(G)):

$\mathtt{Rename(C1)} = \mathtt{add(0,0,0)}$

$\mathtt{Unify}\{\mathtt{X}=0,\mathtt{Y}=01,\mathtt{s(0)}=0\} : \mathtt{fail}$

R(C2,L1(G)):

$\mathtt{Rename(C2)} = \mathtt{add(s(X1),Y1,s(Z1))} :\!-\mathtt{add(X1,Y1,Z1)}$

$\mathtt{Unify}\{\mathtt{X}=\mathtt{s(X1)},\mathtt{Y}=\mathtt{Y1},\mathtt{s(0)}=\mathtt{s(Z1)}\} \rightarrow_{\mathtt{mgu}} \{\mathtt{X}=\mathtt{s(X1)},\mathtt{Y}=\mathtt{Y1},\mathtt{Z1}=0\}$

$\mathtt{G1} \equiv \mathtt{X}=\mathtt{s(X1)},\mathtt{Y}=\mathtt{Y1},\mathtt{Z1}=0,\mathtt{add(X1,Y1,0)}$

R(C1,L4(G1)):

$\mathtt{Rename(C1)} = \mathtt{add(0,0,0)}$

$\mathtt{Unify}\{\mathtt{X1}=0,\mathtt{Y1}=0\} \rightarrow_{\mathtt{mgu}} \{\mathtt{X1}=0,\mathtt{Y1}=0\}$

$\mathtt{G2} \equiv \mathtt{X}=\mathtt{s(0)},\mathtt{Y}=0,\mathtt{Z1}=0,\mathtt{X1}=0,\mathtt{Y1}=0)$

La risoluzione produce un *goal risolto*: i valori di X e Y sono quelli di "falsificano" G

Altre soluzioni?: Il *risolvente* R(C2,L4(G1))?

Example

Un goal è risolto se contiene solo $\overline{\mathtt{E}}$ in forma risolta, ovvero tale che $\vdash \overline{\mathtt{E}} \longrightarrow_{\mathtt{mgu}} \overline{\mathtt{E}}$. Si dica quali proprietà sintattiche devono essere soddisfatte affichè $\overline{\mathtt{E}}$ sia in forma risolta.

Example

Una clausola è una formula logica con una particolare struttura, disgiuntiva o implicativa (come si preferisce). Questa struttura impone una correlazione tra gli atomi. Nel caso della clausola $\mathtt{add(s(X), Y, s(Z))} :\!\!-\mathtt{add(X, Y, Z)}$, quale correlazione è espressa tra l'atomo $\mathtt{add(s(X), Y, s(Z))}$ e l'atomo $\mathtt{add(X, Y, Z)}$

Example

Un sistema di riscrittura è un sistema che permette di rimpiazzare un termine con uno semanticamente equivalente. Perchè la risoluzione non è un sistema di riscrittura?

Example

$\mathtt{P} \equiv \{\mathtt{add(0,0,0)}; \mathtt{add(s(X),Y,s(Z))} :\!-\mathtt{add(X,Y,Z)}\}$
$\mathtt{G} \equiv \mathtt{add(X,Y,s(0))}$ [ovvero la formula $\neg\exists(\mathtt{X},\mathtt{Y})\mathtt{add(X,Y,s(0))}$]
R(C2,L1(G)):
$\mathtt{Rename(C2)} = \mathtt{add(s(X1),Y1,s(Z1))} :\!-\mathtt{add(X1,Y1,Z1)}$
$\mathtt{Unify}\{\mathtt{X}=\mathtt{s(X1)},\mathtt{Y}=\mathtt{Y1},\mathtt{s(0)}=\mathtt{s(Z1)}\} \rightarrow_{\mathtt{mgu}} \{\mathtt{X}=\mathtt{s(X1)},\mathtt{Y}=\mathtt{Y1},\mathtt{Z1}=\mathtt{0}\}$
$\mathtt{G1} \equiv \mathtt{X} = \mathtt{s(X1)}, \mathtt{Y} = \mathtt{Y1}, \mathtt{Z1} = \mathtt{0}, \mathtt{add(X1,Y1,0)}$

- La risoluzione conserva la soddisfacibilità non la semantica in senso stretto:

  $$\mathtt{G} \rightarrow_{\mathtt{lush}R(C2,L1(G))} \mathtt{G1}$$

  Le due formule sono diverse (non solo sintatticamente) ma se G1 è refutabile allora lo è anche G
- Quando il risolvente ha la forma $\mathtt{R(,Li(G))}$ allora il letterale $\mathtt{Li(G)}$ è un'equazione.

PROLOG: Sintassi

Table18 – PROLOG Sintassi

Struttura dei programmi

$$\mathtt{P} ::= \overline{\mathtt{C_{q1}}\,.} \;\cdots\; \overline{\mathtt{C_{qk}}\,.}$$

$$\mathtt{C_q} ::= \mathtt{H_q}[:\!-\overline{\mathtt{L}}]$$

$$\mathtt{H_q} ::= \mathtt{q}(\overline{\mathtt{T}})$$

$$\mathtt{L} ::= \mathtt{A} \mid \mathtt{not\ L} \mid \mathtt{cut} \mid \mathtt{L} \text{-}\!> \mathtt{L}; \mathtt{L} \mid \mathtt{Q} \mid \ldots$$

$$\mathtt{A} ::= \mathtt{T} = \mathtt{T} \mid \mathtt{u}(\overline{\mathtt{T}})$$

$$\mathtt{T} ::= \mathtt{X} \mid \mathtt{g}(\overline{\mathtt{T}}) \mid \mathtt{R} \mid \ldots$$

legenda

$\mathtt{q}, \mathtt{u} \in \Pi;\ \mathtt{g} \in \Sigma;\ \mathtt{not} \equiv \backslash+;\ \mathtt{cut} \equiv !;$

$\mathtt{Q} =$ predicati primtitivi

$\mathtt{R} =$ costruttori primtitivi

Example

```
append([],Z,Z).
append([X|Xs],Y,[X|Zs]) :-append(Xs,Y,Zs).
```

PROLOG: Semantica

Table18 – PROLOG Goal
Struttura dei Goals
$\texttt{G} ::= \overline{\texttt{L}}. \mid \texttt{G} \ ; \ \texttt{G}$
$\texttt{L} ::= \texttt{A} \mid \texttt{not L} \mid \texttt{cut} \mid \texttt{L} \texttt{->} \texttt{L}; \texttt{L} \mid \texttt{Q} \mid \ldots$
$\texttt{A} ::= \texttt{T} = \texttt{T} \mid \texttt{u}(\overline{\texttt{T}})$
$\texttt{T} ::= \texttt{X} \mid \texttt{g}(\overline{\texttt{T}}) \mid \texttt{R} \mid \ldots$
Semantica
$\texttt{SLD resolution} = \texttt{LUSH} + \texttt{LeftMost} + \texttt{TopMost}$
legenda :
$\texttt{LeftMost} =$ primo letterale a sinistra
$\texttt{TopMost} =$ prima clausola dall'alto

Example

$\texttt{append}([\texttt{X}|\texttt{Xs}], \texttt{Y}, [\texttt{X}|\texttt{Zs}]) :- \texttt{append}(\texttt{Xs}, \texttt{Y}, \texttt{Zs}).$

$\texttt{append}([], \texttt{Z}, \texttt{Z}).$

cosa calcola il goal? : $\texttt{append}([\texttt{a}, \texttt{b}, \texttt{c}], [\texttt{d}, \texttt{e}], \texttt{Z}).$

e il goal? : $\texttt{append}(\texttt{X}, [\texttt{d}, \texttt{e}], \texttt{Z}).$

PROLOG: Uso dei Simboli

Example

```
append([X|Xs], Y, [X|Zs]) :-append(Xs, Y, Zs).
append([], Z, Z).
```

- Simboli di Predicato. Introducono le funzioni (con cui vogliamo calcolare) sotto forma di relazione.
  - `append(I1,I2,O)` è una relazione per $\mathtt{f}_{\mathtt{append}}$ che unisce liste. Ed è tale che: per ogni tripla di liste $\mathtt{I1}, \mathtt{I2}, \mathtt{O} \in \mathtt{HU}_{\mathtt{Sigma}}$, $\mathtt{f}_{\mathtt{append}}(\mathtt{I1}, \mathtt{I2}) = \mathtt{O}$ sse `append(I1,I2,O)`.
- Simboli di funzioni. Introducono costruttori per valori tipati per i nostri calcoli.
  - $\mathtt{int} =_{\mathtt{min}} \{\mathtt{0}, \mathtt{s(t)} | \mathtt{t} \in \mathtt{HU}_{\{\mathtt{0_0}, \mathtt{s_1}\}}\}$
  - $\mathtt{list}(\Sigma) =_{\mathtt{min}} \{[], [\mathtt{t}|\mathtt{L}] | \mathtt{t} \in \mathtt{HU}_{\Sigma}, \mathtt{L} \in \mathtt{list}(\Sigma)\}$
  - Quelli sono tipi: Ci accontentiamo di valori tipati

Programmazione Logica: Spazio di Ricerca/1

Example

un programma in L.P.

X, Y, Z, W: int s: int $\rightarrow$ int

plus: int $\times$ int $\times$ int times: int $\times$ int $\times$ int

fact: int $\times$ int

PROLOG *non ha tipi*.

(1) fact(0, s(0)).
(2) fact(s(X), Y) :−fact(X, Z), times(s(X), Z, Y).
(3) times(0, Y, 0).
(4) times(s(X), Y, W) :−times(X, Y, Z), plus(Y, Z, W).
(5) plus(0, Y, Y).
(6) plus(s(X), Y, s(W)) :−plus(X, Y, W).

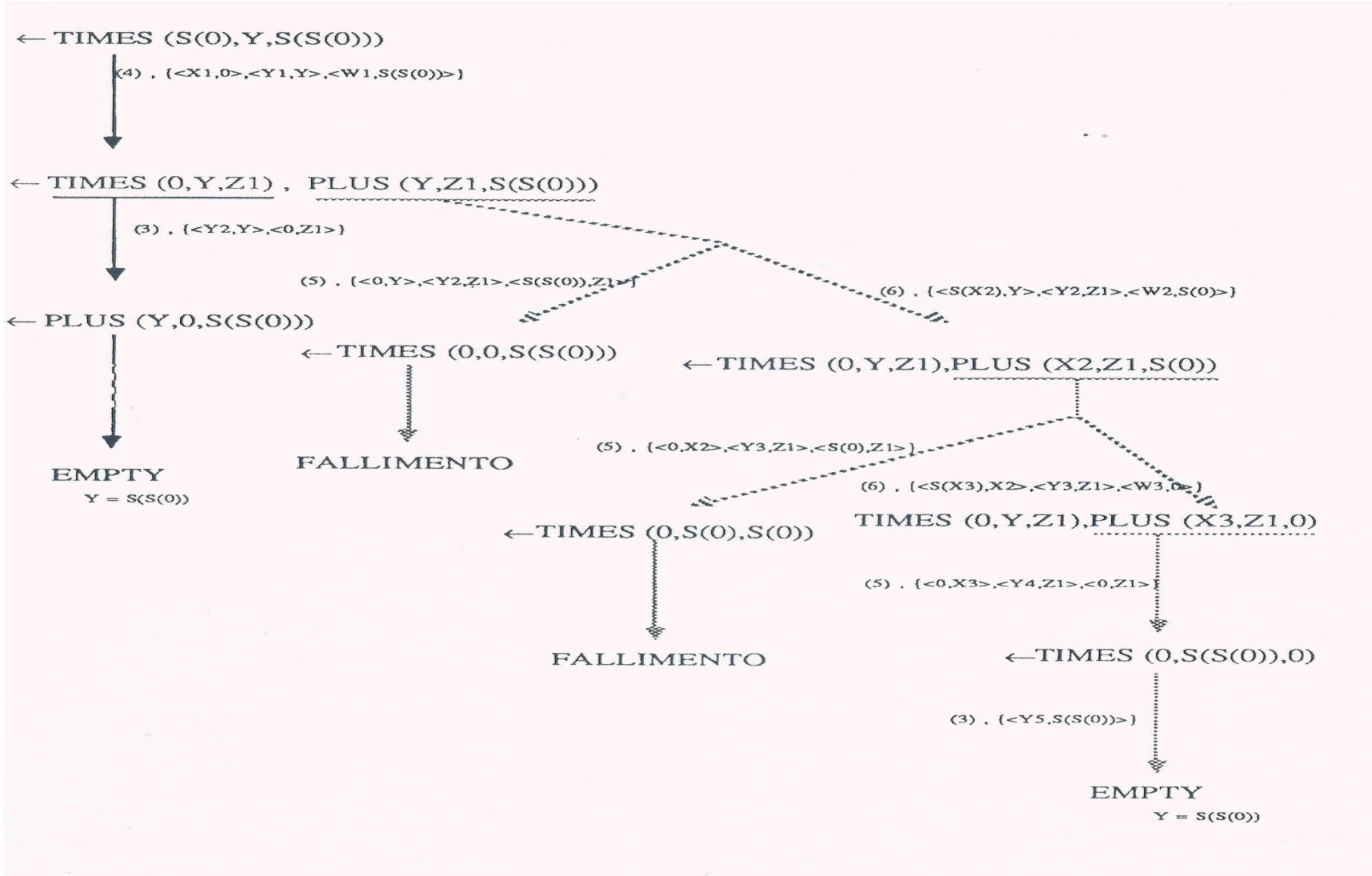
Example

due Goals per il programma

fact(X, Y), plus(0, s(0), X).
times(s(0), Y, s(s(0))).

Programmazione Logica: Spazio di Ricerca (S.R.)/2

un albero di risolventi con radice il Goal



Implementazione dello S. R.: **Backtrack** (di risolventi)

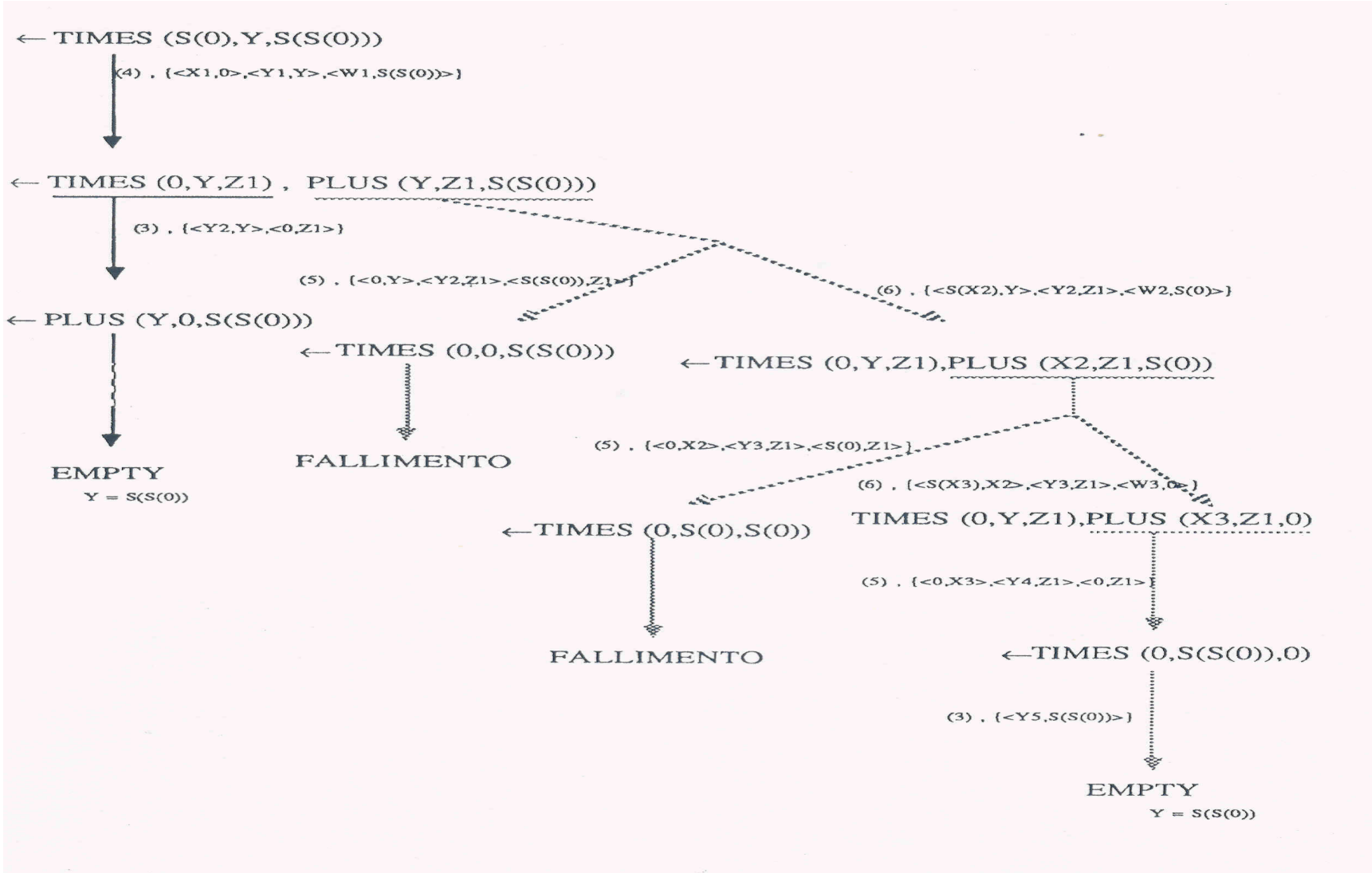
- (1) Sia $\mathtt{G}$ il goal corrente e $\mathtt{R}(\mathtt{Ci}_1, \mathtt{Lj}_1(\mathtt{G})), ..., \mathtt{R}(\mathtt{Ci}_k, \mathtt{Lj}_k(\mathtt{G}))$[1] la sequenza dei suoi risolventi.
- (2) $\mathtt{R}(\mathtt{Ci}_1, \mathtt{Lj}_1(\mathtt{G}))$ è selezionato e utilizzato per il nuovo corrente goal $\mathtt{G}_{\mathtt{R}(\mathtt{Ci}_1, \mathtt{Lj}_1(\mathtt{G}))}$ gli altri sono posti, in ordine inverso, sul top dello stack dei risolventi. E si ripete (1)-(2)
- (3) **Fallimento**Se $\mathtt{G}$ non ha risolventi e non è $\overline{\mathtt{E}}$[2]. Allora:
  - (a)Se lo stack è non vuoto si rimuove dal top dello stack il risolvente $\mathtt{R}(\mathtt{Ci}_k, \mathtt{Lj}_k(\mathtt{G}_m))$ e si va la passo (2)
  - (b)**Finito** Se lo stack è vuoto, si fallisce in modo finito.
  - (4) **Successo.** Se $\mathtt{G} = \overline{\mathtt{E}}$. Allora:
  - (1) Si fornisce la soluzione, ovvero la sostituzione $\overline{\mathtt{E}}$.
  - (2a) **Stop** se non sono richieste altre soluzioni oppure lo stack è vuoto
  - (2b) si rimuove dal top dello stack il risolvente $\mathtt{R}(\mathtt{Ci}_k, \mathtt{Lj}_k(\mathtt{G}_m))$ e si va la passo (2)

[1] Con $\mathtt{j}_n \leq \mathtt{j}_{n+1}$ e se $\mathtt{j}_n = \mathtt{j}_{n+1}$ allora $\mathtt{i}_n < \mathtt{i}_{n+1}$. In Prolog, si usa SLD resolution, quindi: $\mathtt{j}_1 = ... = \mathtt{j}_k$

[2] in forma risolta

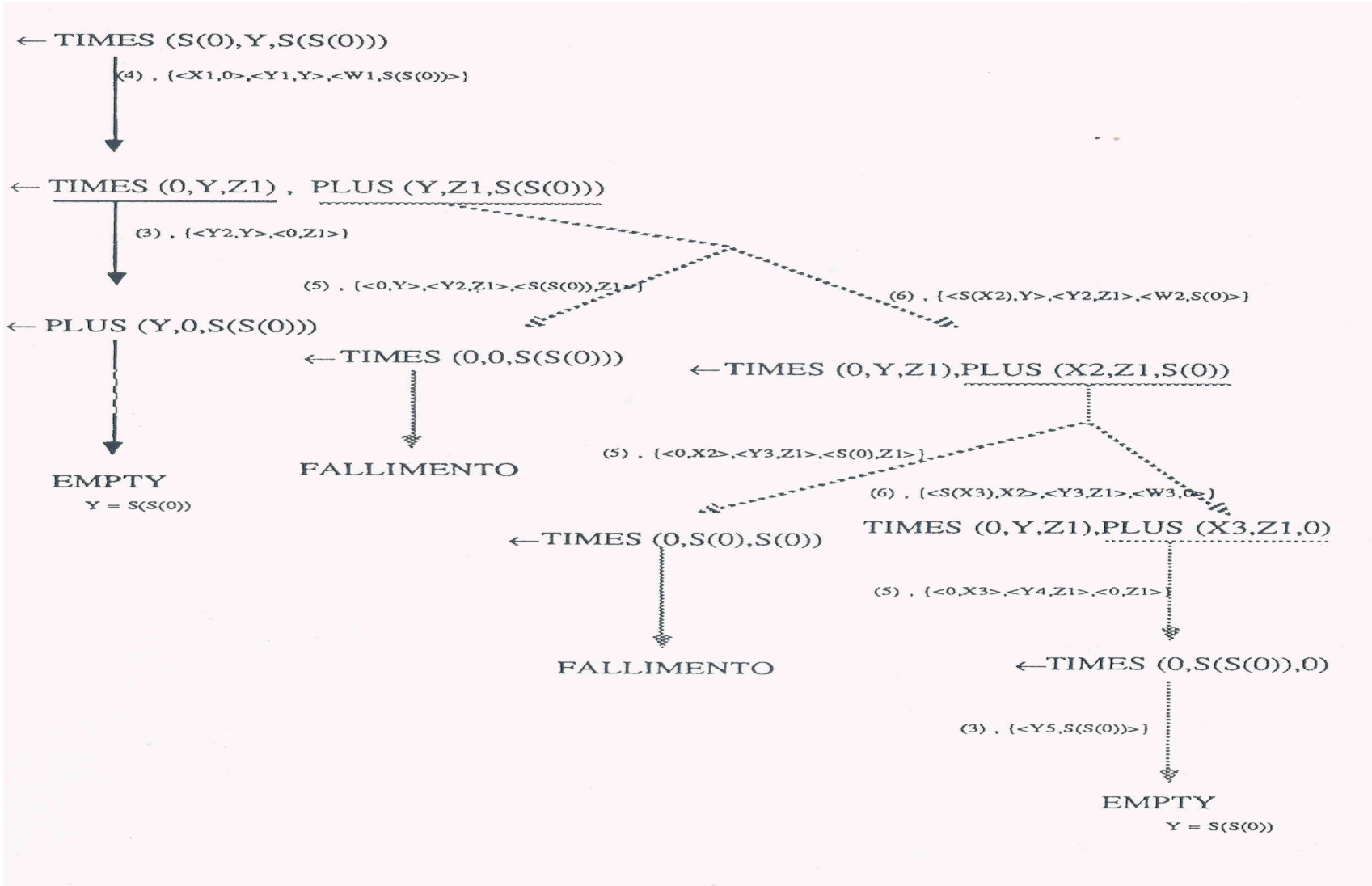
Programmazione Logica: Spazio di Ricerca/2

un albero di risolventi con radice il Goal



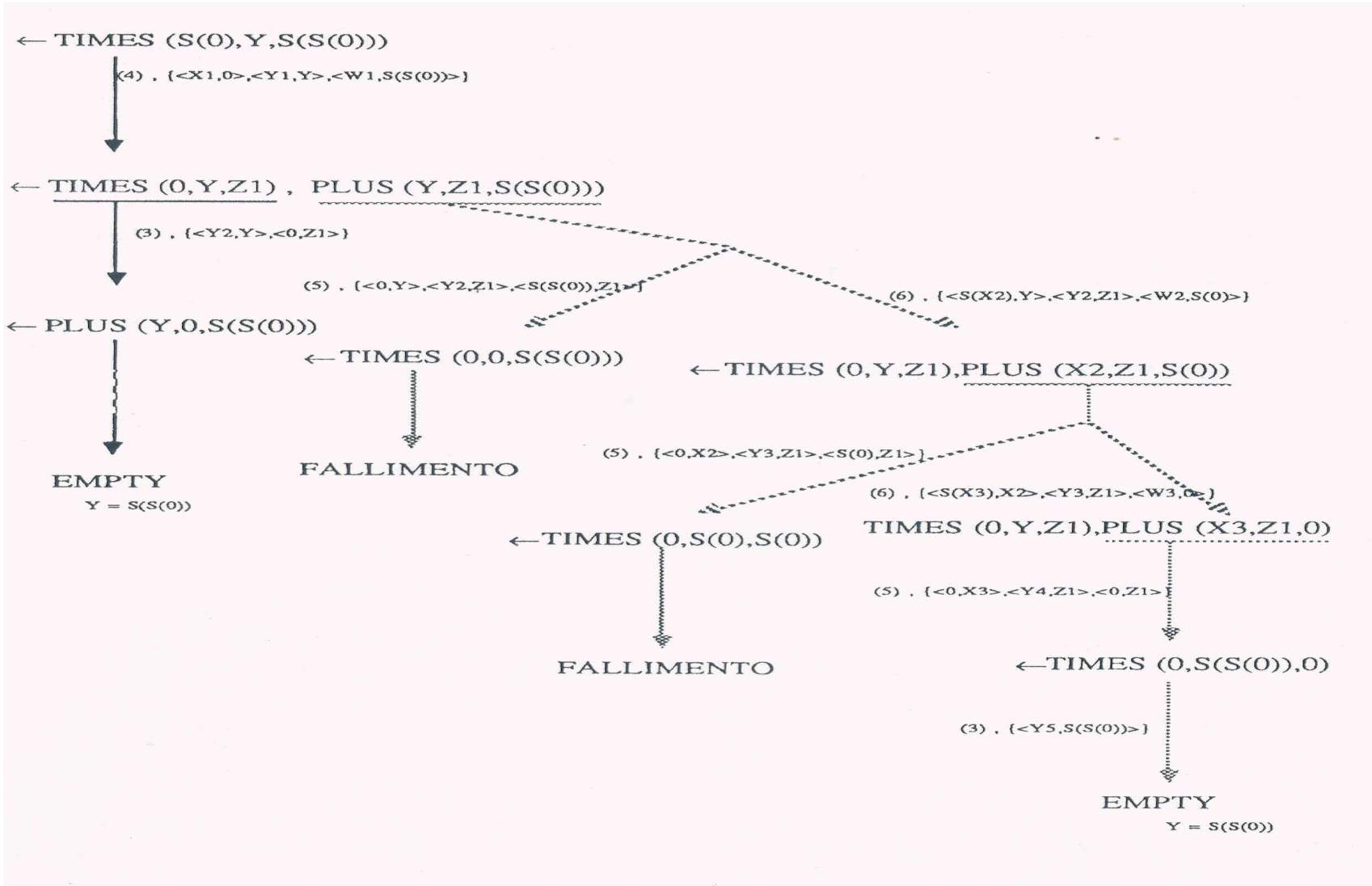
Spazio di Ricerca: Non determinismo

`plus(Y,Z1,s(s(0)))` conduce a differenti risolventi e possibili valori



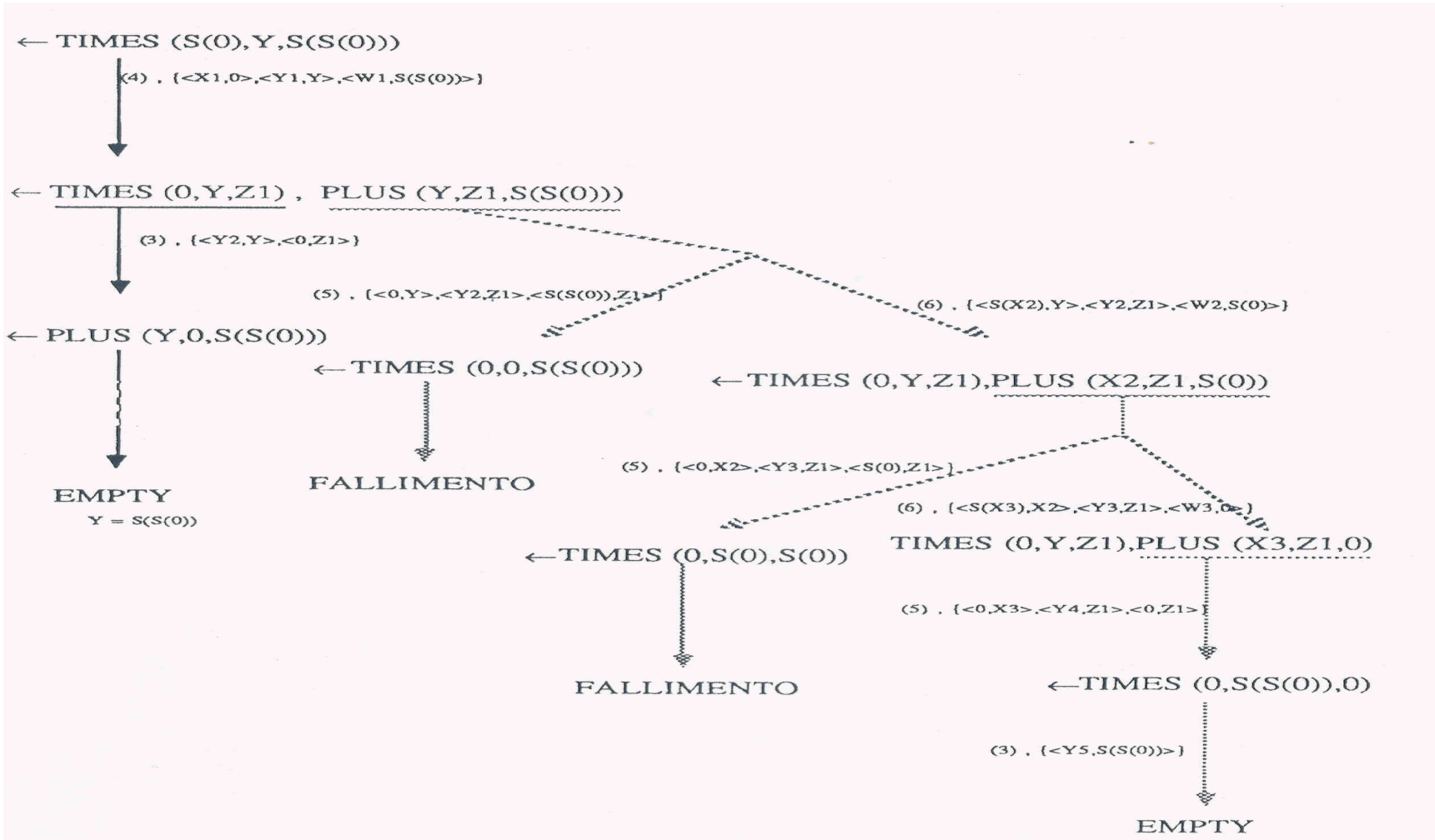
Spazio di Ricerca: Variabili Logiche e Invertibilità

`plus(Y,Z1,s(s(0)))` `Y` e `Z1` usati per trovare tutte le coppie la cui somma ...



Spazio di Ricerca: dichiaratività

risolvendo con il risolvente in linea continua si utilizza un algoritmo di calcolo, risolvendo con quello in linea tratteggiata si utilizza un differente algoritmo di calcolo



Una/Tutte le soluzioni

```
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2011 Daniel Diaz
| ?- [fact].
compiling /Users/marcobellia/Desktop/Paradigmi2012/LINGUAGGI/PROLOG/Programmi/fact.pl for byte code...
/Users/marcobellia/Desktop/Paradigmi2012/LINGUAGGI/PROLOG/Programmi/fact.pl:1: warning: singleton variables [Y] for times/3
/Users/marcobellia/Desktop/Paradigmi2012/LINGUAGGI/PROLOG/Programmi/fact.pl compiled, 0 lines read - 1553 bytes written, 14
ms

(1 ms) yes
| ?- fact(X,Y).

X = 0
Y = s(0) ? ;

X = s(0)
Y = s(0) ? ;

X = s(s(0))
Y = s(s(0)) ? ;

X = s(s(s(0)))
Y = s(s(s(s(s(s(0)))))) ? ;

X = s(s(s(s(0))))
Y = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))))))))))))))) ? ;

X = s(s(s(s(s(0)))))
Y = s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(
s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0)))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))) ?
```

Metodologie: Programmazione Deduttiva

Vogliamo usare linguaggi come PROLOG per scrivere programmi in grado di dare soluzione a problemi come *la torre di Hanoi* o come questi sotto.

Example

Si dispongano `N` regine su una scacchiera $\texttt{N} \times \texttt{N}$ senza che due regine si coprano reciprocamente. Si scriva un programma che mostra se e quante configurazioni diverse diverse possono essere fornite per dato `N`.

Example

Si consideri un piano suddiviso in un arbitrario naturale `n`, $\texttt{n} < \texttt{N} \in \texttt{Nat}$ di regioni chiuse. Si abbiano `k` diversi colori a disposizione con cui colorare regioni adiacenti in modo tale che regione adiacenti abbiano colori distinti. Si scriva un programma che mostra se e quante colorazioni diverse del piano possono essere ottenute per dato `(n,k)`.