Programmazione logica

e

Prolog

Clausole

Implicazione : $\neg B \vee A$ corrisponde a $B \rightarrow A$

oppure A implicato da B (A se B) : $A \leftarrow B$

Per una clausola :

$$A_1 \vee A_2 \vee \ldots \vee A_n \vee \neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_m$$

Usando ul teorema di De Morgan :

$$(A_1 \vee A_2 \vee \ldots \vee A_n) \vee \neg(B_1 \wedge B_2 \wedge \ldots \wedge B_m)$$

$$A_1 \vee A_2 \vee \ldots \vee A_n \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$$

Una clausola di Horn ha un solo letterale positivo :

$A \vee \neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_m$

$A \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$

In particolare :

regola	$A \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$
fatto	$A \leftarrow$
goal	$\leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_m$
contraddizione	$\leftarrow$

Notazione PROLOG

regola	$A :- B_1, B_2, \ldots, B_m.$
fatto	A.
goal	$:- B_1, B_2, \ldots, B_m.$
contraddizione	*false*.

Un **programma logico**:

sum(0,0,X).

sum(s(X),Y,s(Z)) :- sum(X,Y,Z).

Possiamo interpretare s(N) come il successore del numero N

allora 0, s(0), s(s(0)), s(s(s(0)))… rappresentano 0,1,2,3…

Questo programma definisce la somma fra due numeri naturali.

Domande :

$\exists W$ sum(s(0),0 , W) $\quad W = 1 + 0$

$\exists W$ sum(s(s(0)), s(0), W) $\quad W = 2 + 1$

La negazioni sono :

:- sum(s(0),0 , W) . $\quad W = 1 + 0$

:- sum(s(s(0)), s(0), W). $\quad W = 2 + 1$

ricordare che $\neg \forall x A(x) \leftrightarrow \exists x \neg A(x)$

Dato un insieme di clausole di Horn è possibile derivare la clausola vuota solo se c`è una sola clausola senza testa e tutte le altre clausole hanno la testa.

Quindi in un programma logico P tutte le clausole debbono avre la testa mentre la clausola goal G_0. non avrà testa.

Si deve dimostrare che da $P \cup \{G_0\}$ è possibile derivare la clausola vuota.

Se si tentassero ad ogni passo tutte le risoluzioni possibili e si aggiungessero le clausole inferite all' insieme di partenza si avrebbe una esplosione combinatoria.

Si deve adottare una strategia di soluzione opportuna.

Risoluzione ad input lineare

La risoluzione avviene sempre fra l'ultimo goal derivato in ciascun passo e una clausola di programma, mai fra due clausole di programma o fra una clausola di programma ed un goal derivato in precedenza.

Risoluzione ad input lineare

Sia dato un programma logico P e un goal G_0. Si deve dimostrare che da

$$P \cup \{G_0\}$$

è possibile derivare la clausola vuota.

dal goal G_i $\quad :- A_1, A_2, \ldots, A_m.$

e dalla regola regola $\quad A :- B_1, B_2, \ldots, B_m.$

se esiste un unificatore ϑ tale che $[A]\vartheta = [A_1]\vartheta$

si ottiene in nuovo goal

$G_{i+1} \quad :- B_1, B_2, \ldots, B_m, A_2, \ldots, A_m.$

La risoluzione avviene sempre fra l'ultimo goal e una clausola di programma.

Si puo' avere: successo (viene generata la clausola vuota)

insuccesso finito

insuccesso infinito

La sostituzione di risposta è la sequenza di unificatori usati. Applicati alle variabili del goal iniziale danno la risposta.

Strategia di risoluzione in profondità con backtracking

Possono esserci più clausole di programma utilizzabili per applicare la risoluzione con il goal corrente.

Si possono adottare diverse strategie di ricerca:

in profondità : si sceglie una clausola e si mantiene fissa questa scelta, finchè non si arriva alla clausola vuota o alla impossibilità di fare nuove risoluzioni. In questo ultimo caso si riconsiderano le scelte fatte precedentemente.

in ampiezza: si considerano in parallelo tutte le possibili alternative.

Il prolog adotta una strategia di risoluzione in profondità con backtracking.

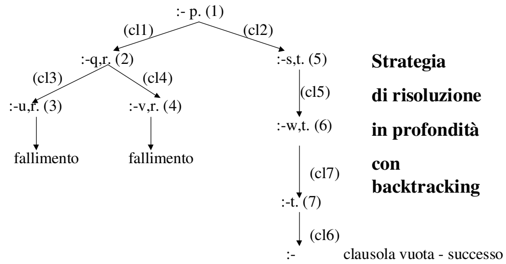
•Permette di risparmiare memoria.

•Non è completa per le clausole di Horn.

(cl1) p:-q,r.
(cl2) p:-s,t.
(cl3) q:-u.
(cl4) q:-v.
(cl5) s:-w.
(cl6) t.
(cl7) w.

Albero di risoluzione

per il goal **:-p.**

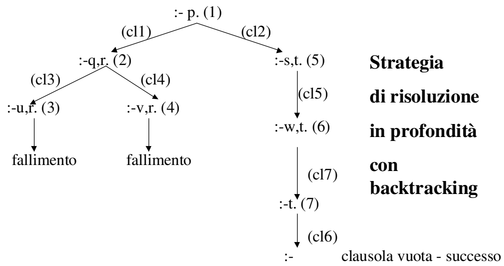


Strategia di risoluzione in profondità con backtracking

(cl1) p:-q,r.
(cl2) p:-s,t.
(cl3) q:-u.
(cl4) q:-v.
(cl5) s:-w.
(cl6) t.
(cl7) w.

Albero di risoluzione

per il goal **:-p.**



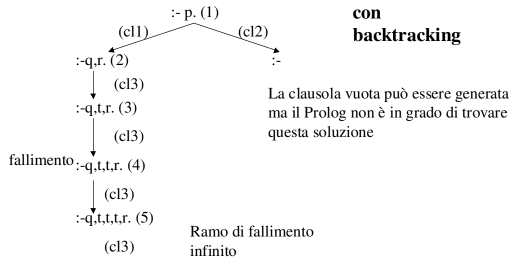
Strategia di risoluzione in profondità con backtracking

(cl1) p:-q,r.
(cl2) p.
(cl3) q:-q,t.

Albero di risoluzione

per il goal **:-p.**

Strategia di risoluzione in profondità con backtracking



Dal **programma logico**:

sum(0,X,X). (c1)

sum(s(X),Y,s(Z)) :- sum(X,Y,Z). (c2)

E dal **goal** sum(s(s(0)),s(0),W) (g0)

risolvendo il goal g0 con c2 con {X/s(0),Y/s(0),W/s(Z1)}

:- sum(s(0),s(0),Z1) (g1)

risolvendo il goal g1con c2 con {X/0,Y/s(0),Z1/s(Z2)}

:- sum(0,s(0),Z2) (g2)

risolvendo il goal g1con c1 con {X/0,Y/s(0),Z2/s(0)}

:- clausola vuota

La risposta si ottiene dalla sequenza di sostituzioni

W=s(Z1) Z1=s(Z2) Z2=s(Z1) quindi W=s(s(s(0))) cioè 2+1=3

PARENTELE

$\forall(m,c)\mathrm{Madre}(m,c) \Leftrightarrow \mathrm{Femmina}(m) \wedge \mathrm{Genitore}(m,c)$

$\forall(p,c)\mathrm{Genitore}(p,c) \Leftrightarrow \mathrm{Figlio}(c, p)$

$\forall(g,c)(\mathrm{Nonno}(g,c) \Leftrightarrow (\exists p)\mathrm{Genitore}(g,p) \wedge \mathrm{Genitore}(p,c))$

$\forall(x, y)(\mathrm{Fratello}(x, y) \Leftrightarrow x \neq y \wedge (\exists p)\mathrm{Genitore}(p, x) \wedge \mathrm{Genitore}(p, y))$

da :

$\mathrm{Genitore}(\mathrm{Giovanni}, \mathrm{Mario}) \wedge \mathrm{Genitore}(\mathrm{Giovanni}, \mathrm{Luca})$

si inferisce :

$\mathrm{Fratello}(\mathrm{Mario}, \mathrm{Luca})$

Programmazione logica e Prolog

PARENTELE

madre(m,c) :– femmina(m), genitore(m,c).

genitore(p,c) :– figlio(c, p).

nonno(g,c) :– genitore(g,p), genitore(p,c).

fratello(p,c) :– diverso(p,c), genitore(g,p), genitore(g,c))

genitore(luisa, luca).

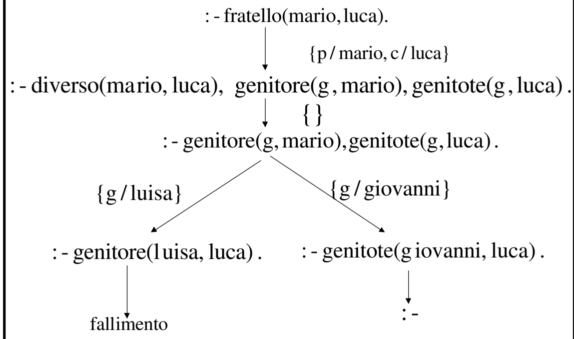
genitore(mario, marco).

genitore(giovanni, mario).

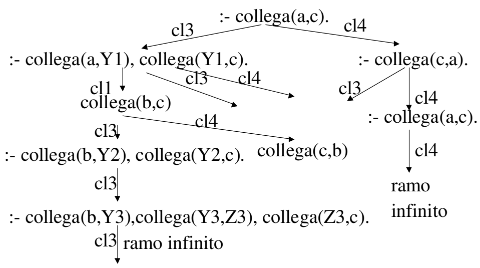
denitore(giovanni, luca). diverso(marco, mario). diverso(mario, luca).

:- Fratello(mario, luca).

Programmazione logica e Prolog



Programmazione logica e Prolog



(cl1) collega(a,b).

(cl2) collega(c,b).

(cl3) collega(X,Z):- collega(X,Y),collega(Y,Z).

(cl4) collega(X,Y):-collega(Y,X).

Programmazione logica e Prolog

P :- Q,R.

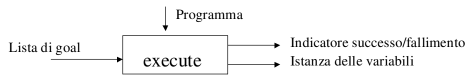
Interpretazione dichiarativa:

P è vero se sono veri Q e R.

Interpretazione procedurale:

Il problema P può essere scomposto nei sottoproblemi P e Q.

Modello di esecuzione:



Un goal può essere visto come una chiamata ad una procedura.

Una regola può essere vista come la definizione di una procedura in cui la testa è l'intestazione mentre la parte destra è il corpo.

Programmazione logica e Prolog

Linguaggi imperativi (C , Pascal etc.)

Si specifica come risolvere un problema

Linguaggi dichiarativi

Funzionali (Lisp puro)

Usa il concetto di funzione e di composizione di funzioni

Logici (Prolog)

Si descrivono le relazioni che intercorrono fra i dati. Non ci sono assegnamenti. Non determinismo.

Programmazione logica e Prolog

Per rendere il Prolog un linguaggio effettivamente utilizzabile vengono aggiunti:

- Notazione per le liste.
- Possibilità di manipolare strutture dati.
- Operazioni aritmetiche.
- Meccanismi di controllo del backtracking.
- Trattamento della negazione.
- Predicati di input/output.
- Meccanismi per modificare/accedere alla base di conoscenza.
- Meccanismi per il caricamento del codice prolog.

Esempi

Liste:

member(X,[X|Resto]).

member(X,[Testa|Coda]) :- membro(X,Coda).

Aritmetica:

area_rettangolo(Base,Altezza,Area) :- Area **is** Base*Altezza.

minimo(X,Y,X) :- X <= Y.

minimo(X,Y,Y) :- X > Y.

CUT e controllo del backtracking

si consideri la clausola:

p:- q1, q2,.. ,qi, !, qi+1,.. ,qn.

Tutte le scente fatte dalla scelta della clausola p e di q1,..,qn vengono rese definitive.

Corrisponde ad un pruning dell'albero di refutazione.

Esempio:

a(X,Y) :- b(X), ! ,c(Y).
a(0,0) .
b(1).
b(2).
c(1).
c(2).

Modifica il significato dichiarativo del programma.

Per il goal :-a(X,Y). Si hanno le risposte:
yes X=1, Y=1;
X=1, Y=2;

no

Negazione per fallimento

Nelle clausole di Horn non è possibile rappresentare direttamente informazione negativa del tipo ¬ P

Ipotesi di Mondo Chiuso (O di negazione per fallimento)

¬ P è vero se è impossibile dimostrare che è vero P.

Se si dispone di una descrizione completa del mondo la negazione per fallimento coincide con la negazione logica.

Negazione per fallimento finito

Il prolog usa la negazione per fallimento finito.

not(P) ha successo su un insieme di clausole se e solo se la dimostrazione di P su tale insieme fallisce in modo finito.

Ad esempio da:
citta(X) :- capitale(X).
citta(X) :- capoluogo(X).
capoluogo(bologna).
capitale(roma).
Si ottiene
:- not(citta(parma)).

(fallimento finito)

mentre da:
citta(X) :- citta(X).
citta(X) :- capoluogo(X).
capoluogo(bologna).
Non si può ottenere
:- not(citta(parma)).

(fallimento infinito)

Negazione per fallimento finito

In prolog la definizione di negazione viene data come:

not(P) :- call(P), !, fail.

not(P).

Dato il programma
disoccupato(X):-
adulto(X), not occupato(X).
occupato(giovanni).
adulto(mario).
Il goal:
:- disoccupato(Y).
Da yes, Y=mario.

Scambiando l' ordine dei goal nella regola:
disoccupato(X):-
not occupato(X), adulto(X).
:- disoccupato(Y). fallisce.
:- disoccupato(mario).
ha successo

Uso del cut e della negazione

Consideriamo il programma che realizza l'intersezione di due insiemi rappresentati come liste.

intersezione([],S,[]).
intersezione([X|Resto],S,[X|Resto1]) :- membro(X,S), intersezione(Resto,S,Resto1).
intersezione([X|Resto],S,Resto1) :- intersezione(Resto,S,Resto1).
:- intersezione([1,2,3],[2,3,4],L). Da come risultato L=[2,3], L=[2], L=[3], L=[].

In quanto manca la mutua esclusione fra le clausole 2 e 3

Si ottiene il programma corretto usando un cut:

intersezione([],S,[]).
intersezione([X|Resto],S,[X|Resto1]) :- membro(X,S), intersezione(Resto,S,Resto1).
intersezione([X|Resto],S,Resto1) :- intersezione(Resto,S,Resto1).

O in maniera dichiarativamente chara con il not:

intersezione([],S,[]).
intersezione([X|Resto],S,[X|Resto1]) :- membro(X,S), intersezione(Resto,S,Resto1).
intersezione([X|Resto],S,Resto1) :- not membro(X,S), intersezione(Resto,S,Resto1).