1° Compitino di Linguaggi di Programmazione
28 Novembre 2012
Compito A

1) Data la seguente descrizione di uno stato delle cose:

Se l'olio è nel motore, oppure non è nel motore, allora la macchina non parte. O la macchina parte oppure non arriva benzina. Se non arriva benzina si va dal meccanico.

Dimostrare se si va o no dal meccanico indicando la sequenza di formule per arrivare alla formula finale. Usate le regole di inferenza usate a lezione.

2) Dato il seguente programma Prolog:

(C1) antenato(X,Y) :- padre(X,Y).
(C2) antenato(X,Y) :- antenato(X,Z), padre(Z,Y).
(C3) padre(a, b).

si definisca l'albero di derivazione relativo al goal antenato(X, Y), utilizzando la regola di selezione right-most.

3) Definire le regole e i fatti affinché il predicato

compress(List, NewList).

sia vero se NewList è la List nella quale gli elementi ripetuti siano sostituiti da un singolo rappresentante, preservando l'ordine originale.

Es. ? :- compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e], X).
X = [a,b,c,a,d,e]

4) Date un esempio di due clausole: una clausola non Horn con letterali positivi e negativi e una clausola di Horn. Usate come letterali i simboli L_1, L_2, ... , L_n.

5) Dato il seguente programma Prolog:

```
accept(Xs) :- initial(Q), accept(Xs,Q).
accept([],Q) :- finale(Q).
accept([X|Xs],Q) :- delta(Q,X,NewQ), accept(Xs,NewQ).
initial(q0).
final(q2).
delta(q0,a,q1),
delta(q0,b,q0).
delta(q1,a,q1).
delta(q1,b,q2).
delta(q2,a,q2).
delta(q2,b,q2).
```

disegnare l'automa riconosciuto dal predicato accept/2. Definire inoltre le risposte alle seguenti query:

```
?- accept([b]).
?- accept([a,b,a,b,b]).
?- accept([a,b,a,b,a,b,b]).
?- accept([a,b,a,b,a,b]).
```

6) Dare una definizione di regola di inferenza, e fare un esempio di almeno due regole di inferenza viste a lezione.

PARZIALE 28 NOVEMBRE 2012

ES 2

(C1) antenato(X, Y) :- padre(X, Y).

(C2) antenato(X, Y) :- antenato(X, Z), padre(Z, Y).

(C3) padre(a, b)

Albero di derivazione right-most.

ES 1

P1) (OM v ¬OM) → ¬MP

P2) MP v ¬B

(3) ¬B → M P3)

~~P3) MP, MODUS PONENS ¬MP~~

P4) ~~TERZO ESCLUSO~~ TERZO ESCUSO OM v ¬OM vero → ¬MP
P1,

P5) P4, MODUS PONENS ¬MP

P6) P5, P2, ~~EQUIVALENZA~~ ¬MP v B

P7) P6, EQUIVALENZA MP → B

~~P8)~~ P6) P5, P2, MODUS TOLLENS $\dfrac{B \rightarrow MP,\ \neg MP}{\neg B}$

P7) P6, INTRODUZIONE OR ~~¬B v M~~

P8) P7, EQUIVALENZA LOGICA B → M

P9) P6, ELIMINAZIONE ¬ B

P10) P7, P9, MODUS PONENS M

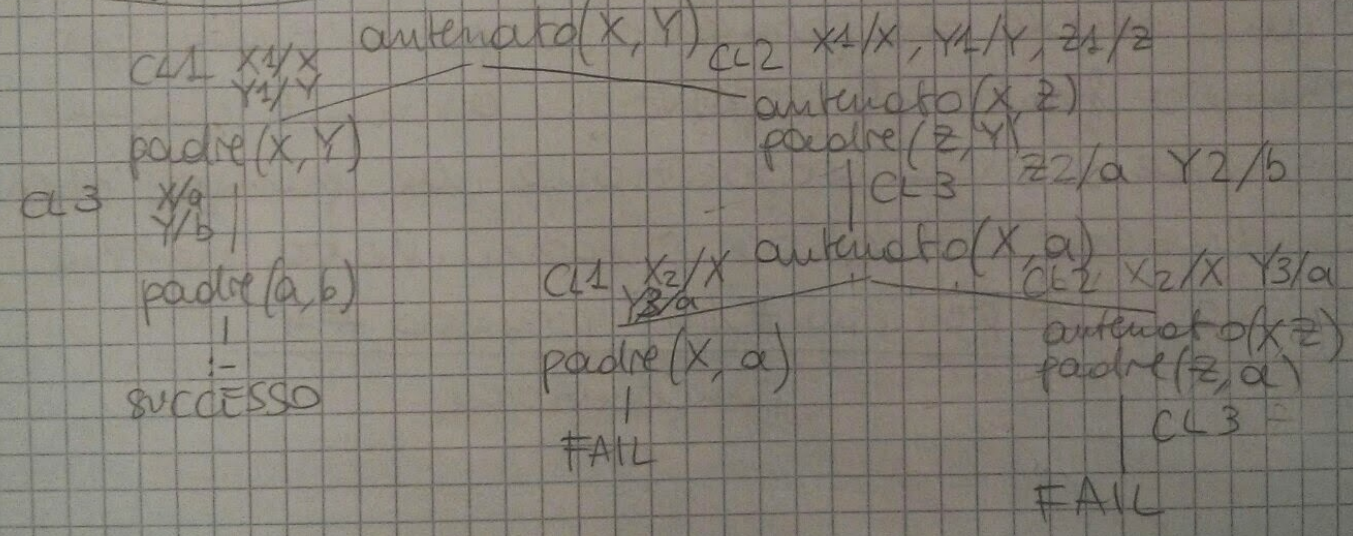
P7) P3, P6, MODUS PONENS M

ES 3

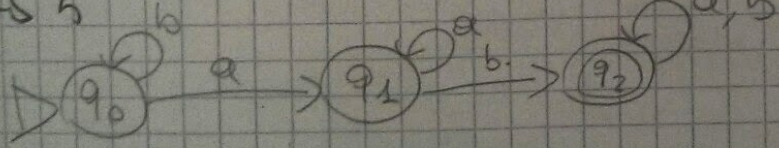
```
compress(List, Newlist).
compress([],[]) :- !.
compress([X],[X]) :- !.
compress([X,X|Xs],Zs) :- compress([X|Xs],Zs),!.
compress([X,Y|Xs],[X|Zs]) :- X\=Y, compress([Y|Ys],Zs).
```

ES 2

(C1) antenato(X,Y) :- padre(X,Y).
(C2) antenato(X,Y) :- antenato(X,Z), padre(Z,Y).
(C3) padre(a,b).
goal antenato(X,Y)
SLD right-most

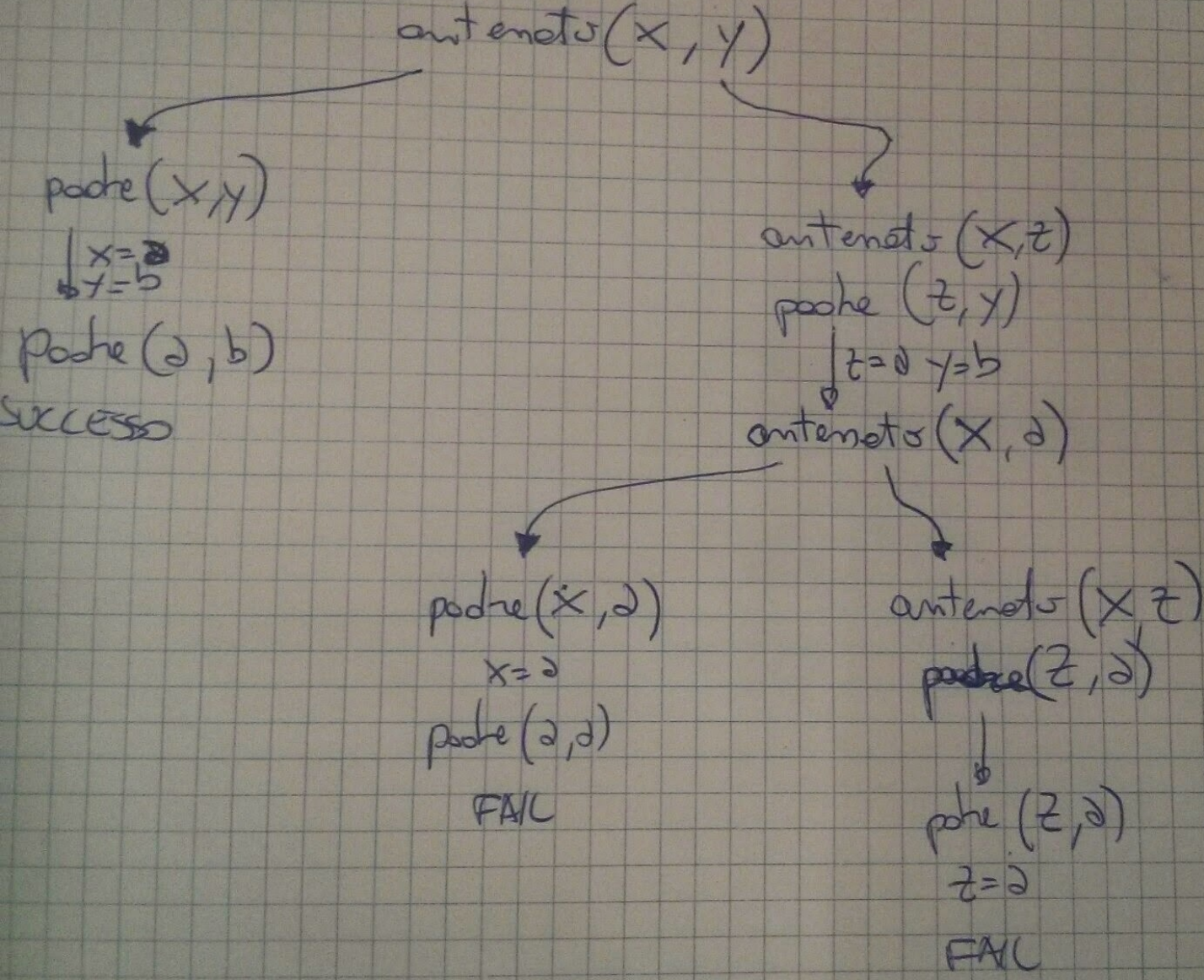


ES 5



accept([b]) false
accept([a,b,a,b,b]) true
accept([a,b,a,b,a,b,b]) true
accept([a,b,a,b,a,b]) true

2) antenato(X,Y) :- padre(X,Y)
antenato(X,Y) :- antenato(X,Z), padre(Z,Y)
padre(a,b)



ES 4

$\neg A \vee \neg B \vee C$ => clausola di Horn

↓

$\neg(A \wedge B) \vee C$

$A \wedge B \rightarrow C$

$\neg A \vee B \vee C$ => non clausola di Horn

ES 6

Inferenza: processo con cui dal una proposizione accolta come vera si passa ad una seconda proposizione la cui verità è derivata dal contenuto della prima.

Una regola d'inferenza è uno schema formale che si applica nell'esprimere un'inferenza. Di fatto è una regola che permette di passare da un numero finito di proposizioni assunte come premesse a una proposizione che funge da conclusione. Di solito è formulata nella forma: premessa #1, ..., premessa #n / conclusione