FONDAMENTI DI INTELLIGENZA ARTIFICIALE – 1° parte (6 CFU)

14 Giugno 2013 – Tempo a disposizione: 2 h – Risultato: 32/32 punti

Esercizio 1 (7 punti)

Si considerino le seguenti frasi:

1. *Tutti gli insegnanti sono persone*
2. *Giovanni è il preside*
3. *I presidi sono insegnanti*
4. *Tutti gli insegnanti considerano il preside un amico o non lo conoscono*
5. *Ognuno è amico di qualcuno*
6. *La gente (le persone) critica soltanto le persone che non sono loro amiche*
7. *Marco è un insegnante*
8. *Marco ha criticato Giovanni*

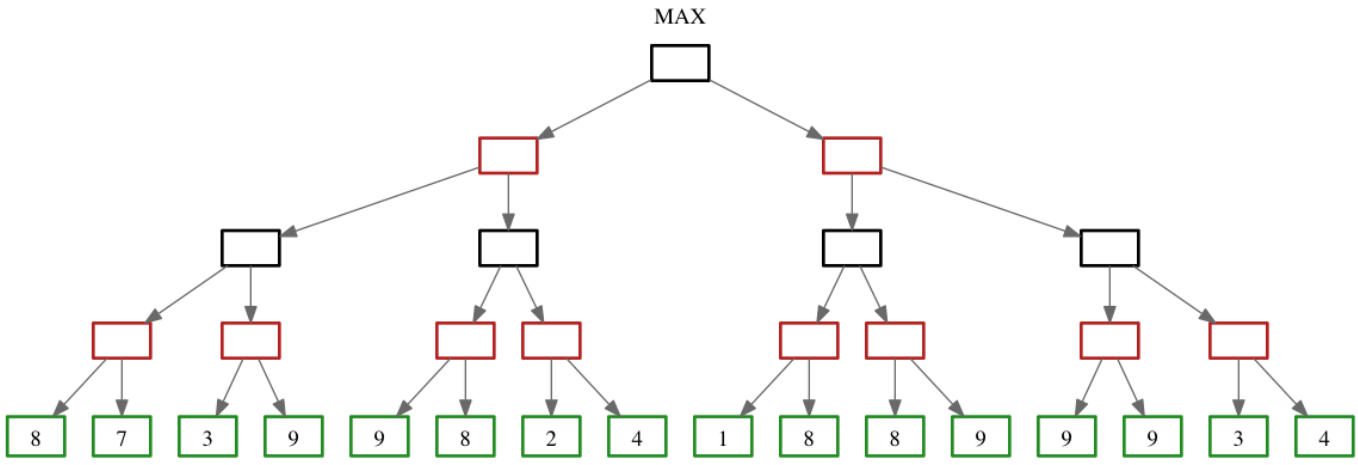
Si formalizzino in *logica dei predicati del I ordine*, utilizzando i seguenti predicati:

- `criticise(X,Y)` `X` critica `Y`
- `knows(X,Y)` `X` conosce `Y`
- `isFriendOf(X,Y)` `X` è amico di `Y`
- `isPerson(X)` `X` è una persona
- `isPrincipal(X)` `X` è il preside
- `isProfessor(X)` `X` è un professore

Infine si trasformino in *clausole* e si dimostri tramite *risoluzione* che *Giovanni non è amico di Marco*.

Esercizio 2 (5 punti)

Si consideri il seguente albero di gioco in cui la valutazione dei nodi terminali è dal punto di vista del primo giocatore (*MAX*). Si mostri come gli algoritmi *min-max* e *alfa-beta* risolvono il problema.



Esercizio 3 (6 punti)

Il seguente programma *Prolog* restituisce il numero `N` di vocali contenute nella lista `P` passata come parametro.

```
contavocali([], 0).
contavocali([H|T], N) :- vocale(H), contavocali(T, NT), N is NT + 1, !.
contavocali([H|T], N) :- contavocali(T, N).
vocale(a). vocale(e). vocale(i). vocale(o). vocale(u).
```

Si mostri l'*albero SLD* relativo al goal:

```
?- contavocali([o,k], X).
```

Esercizio 4 (5 punti)

Si scriva un predicato `remove_at(X,L,K,LR)`, che, data una lista `L` e un intero positivo `K`, rimuove il `K esimo` elemento di `L` restituendo la lista residua `LR` e il `K`-esimo elemento rimosso `X`. Il primo elemento delle lista ha `K=1`.

Esempio:

```
?-remove(X,[a,b,c,d,e],3,L).
Yes X=c L=[a,b,d,e]
```

Esercizio 5 (6 punti)

Si consideri il problema di ordinare in ordine crescente gli elementi di un vettore di interi `V = (4, 3, 2, 1)`. Ad ogni passo si può eseguire un'operazione `swap(x,y)`, mediante la quale è possibile scambiare di posto gli interi in posizione `x` ed `y`. Si risolva il problema utilizzando l'*algoritmo A* con eliminazione degli stati ripetuti* e avendo cura di *indicare l'ordine di espansione dei nodi*, tenendo presente che:

- ogni operazione di `swap(x,y)` ha costo unitario;
- ad ogni passo, gli stati futuri si ottengono applicando ogni possibile operazioni di `swap(x,y)` nell'ordine: `swap(1,2)`, `swap(1,3)`, `swap(1,4)`, `swap(2,3)`, `swap(2,4)`, `swap(3,4)`;
- l'euristica di ogni stato è data da ⌈n/2⌉, dove `n` è il numero di elementi fuori posto;
- a parità di costo totale si preferisca il nodo a profondità minore ancora da esplorare;
- a parità di profondità si preferisca il nodo da esplorare che è stato generato prima.

Esercizio 6 (3 punti)

Si introduca il concetto di *arc-consistenza*, si mostri un esempio di *rete arc-consistente*

e si delinei in pseudocodice l'*algoritmo AC3*.

VOTO:

- *Esame da 6 CFU, il voto è determinato da questa I parte*
- *Esame da 9 CFU, è la media pesata della I parte (che vale 2/3) e della II (che vale 1/3) ovvero il voto finale è dato da:* $\frac{2\times voto\ I\ parte + voto\ II\ parte}{3}$ *e varia quindi da 0 ad un massimo di 32 (equivalente alla lode).*

FONDAMENTI DI INTELLIGENZA ARTIFICIALE – 2° parte (3 CFU)

14 Giugno 2013 – Tempo a disposizione: 30+15 min – Risultato: 16/16 punti

Esercizio 7 (5 punti)

Si introducano le *logiche descrittive* sottolineandone le peculiarità rispetto alla logica classica.

Esercizio 8 (8 punti)

Definire un predicato Prolog `count_unifying_terms(T, L, Res)` che, dati una lista `L` non necessariamente ground ed un termine `T` non necessariamente ground, restituisca il numero di elementi della lista `L` che unificano con `T`.

Esercizio 9 (3 punti) – ulteriori 15 min

Nota: lo deve svolgere solamente chi non ha partecipato alle lezioni/esercitazioni su Prolog e grammatiche.

Data la seguente grammatica, che definisce cosa è un nome (un nome inizia con una lettera, 'a' o 'b' per limitarci a sole due lettere, seguita da cifre – 0 o 1 - o lettere comunque alternate, senza altri simboli; ad esempio, *a*, *ax*, *a1b, b1* sono nomi, *1a* non lo è):

```
G = (Vn, Vt, P, S)
Vn = {nome, lettera, cifra, lettCifra}
Vt = {a,b,0,1}
P = { nome ::= lettera | lettera lettCifra
      lettera ::= a | b
      cifra ::= 0 | 1
      lettCifra ::= lettera | lettera lettCifra !
                    cifra  | cifra lettCifra    }
S = nome
```

Si scrivano le corrispondenti clausole in versione DCG estesa che verificano la correttezza di una frase (ad esempio il nome `a123`).

VOTO:

- *Esame da 3 CFU, il voto è determinato da questa 2° parte*
- *Esame da 9 CFU, è la media pesata della 1° parte (che vale 2/3) e della 2° (che vale 1/3) ovvero il voto finale è dato da:* $\frac{2 \times voto\ I\ parte + voto\ II\ parte}{3}$ *e varia quindi da 0 ad un massimo di 32 (equivalente alla lode).*

FONDAMENTI DI INTELLIGENZA ARTIFICIALE – 1° parte
14 Giugno 2013 – Soluzioni

Esercizio 1

Frasi in *logica dei predicati del I ordine* e loro traduzione in *clausole*:

1. *Tutti gli insegnanti sono persone*
   ∀X: isProfessor(X) ⇒ isPerson(X)
   {¬isProfessor(X) ∨ isPerson(X)}
2. *Giovanni è il preside*
   isPrincipal(gio)
   {isPrincipal(gio)}
3. *I presidi sono insegnanti*
   ∀X: isPrincipal(X) ⇒ isProfessor(X)
   {¬isPrincipal(X) ∨ isProfessor(X)}
4. *Tutti gli insegnanti considerano il preside un amico o non lo conoscono*
   ∀X,Y: isProfessor(X) ∧ isPrincipal(Y) ⇒ isFriendOf(X,Y) ∨ ¬knows(X,Y)
   {¬isProfessor(X) ∨ ¬isPrincipal(Y) ∨ isFriendOf(X,Y) ∨ ¬knows(X,Y)}
5. *Ognuno è amico di qualcuno*
   ∀X,∃Y: isFriendOf(X,Y)
   {isFriendOf(X, g(X))} (g(X) è la funzione di Skolem)
6. *La gente (le persone) critica soltanto le persone che non sono loro amiche*
   ∀X,Y: isPerson(X) ∧ isPerson(Y) ∧ criticise(X,Y) ⇒ ¬isFriendOf(Y,X)
   {¬isPerson(X) ∨ ¬isPerson(Y) ∨ ¬criticise(X,Y) ∨ ¬isFriendOf(Y,X)}
7. *Marco è un insegnante*
   isProfessor(mar)
   {isProfessor(mar)}
8. *Marco ha criticato Giovanni*
   criticise(mar,gio)
   {criticise(mar,gio)}
9. *Goal: Giovanni non è amico di Marco*
   ¬isFriendOf(gio,mar)
   {isFriendOf(gio,mar)}

Teoria a clausole:

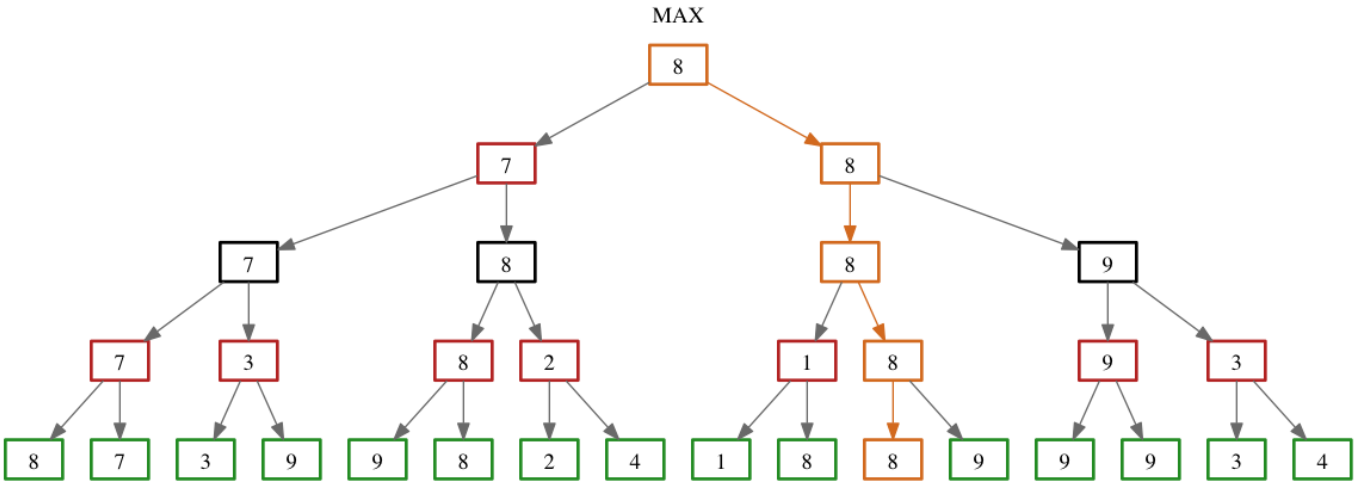
1. ¬isProfessor(X) ∨ isPerson(X)
2. isPrincipal(gio)
3. ¬isPrincipal(X) ∨ isProfessor(X)
4. ¬isProfessor(X) ∨ ¬isPrincipal(Y) ∨ isFriendOf(X,Y) ∨ ¬knows(X,Y)
5. isFriendOf(X, g(X))
6. ¬isPerson(X) ∨ ¬isPerson(Y) ∨ ¬criticise(X,Y) ∨ ¬isFriendOf(Y,X)
7. isProfessor(mar)
8. criticise(mar,gio)
9. isFriendOf(gio,mar)

Risoluzione:

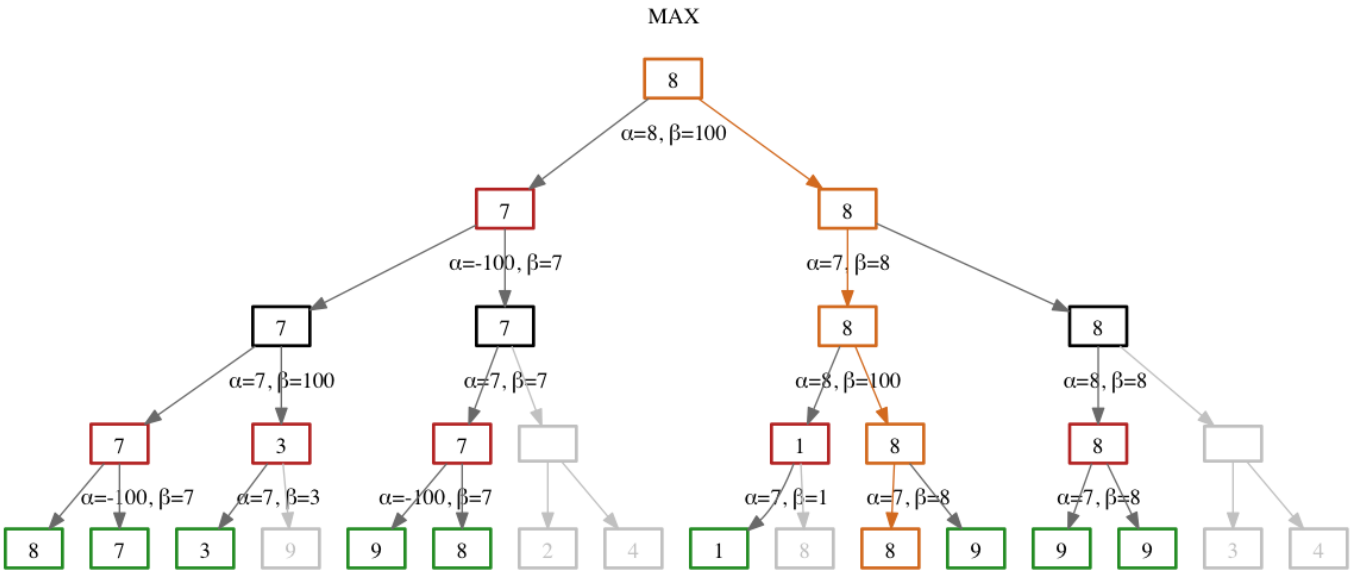
10. = 9. ∪ 6. ¬isPerson(mar) ∨ ¬isPerson(gio) ∨ ¬criticise(mar,gio) {X/mar,Y/gio}
11. = 10. ∪ 8. ¬isPerson(mar) ∨ ¬isPerson(gio) {}
12. = 11. ∪ 1. ¬isProfessor(mar) ∨ ¬isPerson(gio) {X/mar}
13. = 12. ∪ 7. ¬isPerson(gio) {}
14. = 13. ∪ 1. ¬isProfessor(gio) {X/gio}
15. = 14. ∪ 3. ¬isPrincipal(gio) {X/gio}
16. = 15. ∪ 2. □

Esercizio 2

min-max:



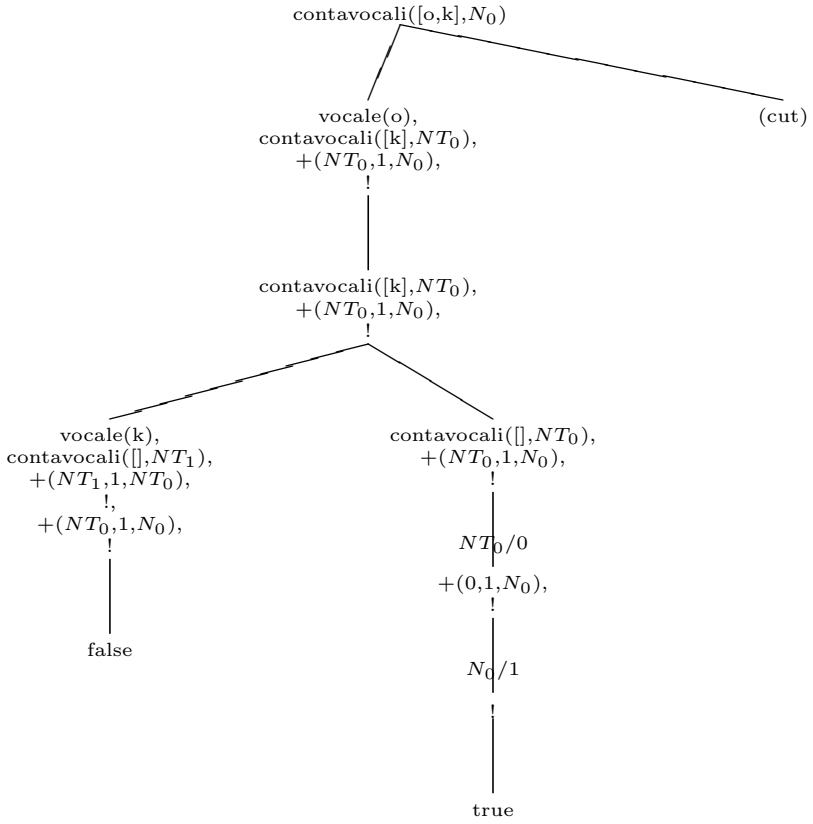
alfa-beta:



I nodi colorati in grigio sono quelli che vengono tagliati dall'algoritmo alfa-beta.

Esercizio 3

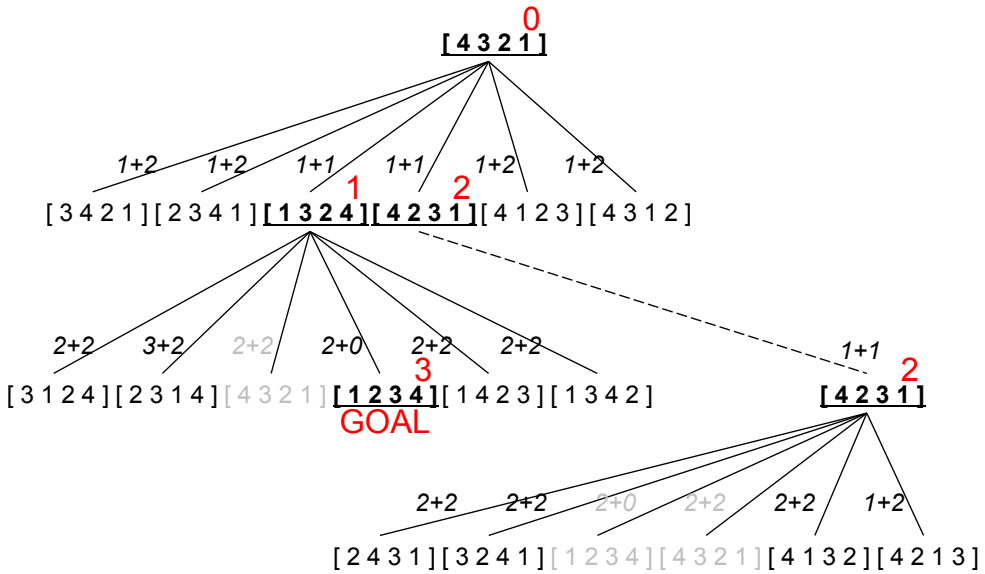
Albero SLD:



Esercizio 4

```
% Remove the K'th element from a list.
% The first element in the list is number 1.
% remove_at(X,L,K,R) :- X is the K'th element of the list L; R is the
%    list that remains when the K'th element is removed from L.
remove_at(X, [X|Xs], 1, Xs).
remove_at(X, [Y|Xs], K, [Y|Ys]) :- K > 1,  K1 is K - 1,  remove_at(X,Xs,K1,Ys).
```

Esercizio 5



Esercizio 6

Vedi slides del corso.

FONDAMENTI DI INTELLIGENZA ARTIFICIALE – 2° parte
14 Giugno 2013 – Soluzioni

Esercizio 7

Vedi slides del corso.

Esercizio 8

Una prima soluzione:

```
count_unifying_terms1(Term, List, NUMBER) :-
    aux_count_unifying_terms1(Term, List, NUMBER,0).
aux_count_unifying_terms1(Term, [], NUMBER, NUMBER).
aux_count_unifying_terms1(Term, [Term|Tail], NUMBER, AUX_NUMBER) :-
    !, NEW_AUX_NUMBER is AUX_NUMBER + 1,
    aux_count_unifying_terms1(Term, Tail, NUMBER, NEW_AUX_NUMBER).
aux_count_unifying_terms1(Term, [Head|Tail], NUMBER, AUX_NUMBER) :-
    aux_count_unifying_terms1(Term, Tail, NUMBER, AUX_NUMBER).
```

Tale soluzione non è corretta in alcuni casi, poiché nel caso in cui Term contenga variabili e i termini nella lista contengano costanti o altri termini ground, nell'unificazione la sostituzione delle variabili verrebbe propagata lungo la chiamata ricorsiva. Ad esempio, la soluzione di cui sopra, se invocata con:

```
:- count_unifying_terms1(p(1), [p(X), p(Y)], NUMBER).
```

Restituisce true, X/1, Y/1, NUMBER/2
Se invece invocata con il goal:

```
:- count_unifying_terms1(p(X), [p(1), p(2)], NUMBER).
```

Il predicato restituisce true con X/1, NUMBER/1 (invece il risultato atteso è NUMBER/2)

Soluzioni possibili in tal senso possono essere diverse:

```
%%% CONTROLLA LE NON UNIFICAZIONI, IMPONENDO UN FAIL QUANDO UNIFICANO
count_unifying_terms(Term, List, NUMBER) :-
     aux_count_unifying_terms(Term, List, NUMBER,0).
aux_count_unifying_terms(Term, [], NUMBER, NUMBER).
aux_count_unifying_terms(Term, [Head|Tail], NUMBER, AUX_NUMBER):-
     not_unifies(Term, Head),
     !,
     aux_count_unifying_terms(Term, Tail, NUMBER, AUX_NUMBER).
aux_count_unifying_terms(Term, [Head|Tail], NUMBER, AUX_NUMBER):-
     NEW_AUX_NUMBER is AUX_NUMBER + 1,
     aux_count_unifying_terms(Term, Tail, NUMBER, NEW_AUX_NUMBER).

not_unifies(Term, Term) :- !, fail.
not_unifies( _ , _ ).
```

Oppure:

```
%%% ASSERISCE TERM COME “TEMPLATE” DI CLAUSOLA
count_unifying_terms1(Term, List, NUMBER) :-
     assert(term(Term)),
     aux_count_unifying_terms1(List, NUMBER,0).

aux_count_unifying_terms1([], NUMBER, NUMBER).
aux_count_unifying_terms1([X|Tail], NUMBER, AUX_NUMBER):-
     term(X),
     !,
     NEW_AUX_NUMBER is AUX_NUMBER + 1,
     aux_count_unifying_terms1(Tail, NUMBER, NEW_AUX_NUMBER).
aux_count_unifying_terms1([_ | Tail], NUMBER, AUX_NUMBER):-
     aux_count_unifying_terms1(Tail, NUMBER, AUX_NUMBER).
```

Oppure:

```
%%% ASSERT di tutti gli elementi della lista, e poi uso di findall
count_unifying_terms1(Term, List, NUMBER) :-
     assert_all(List),
     findall(Term, term(Term), L),
     length(L, NUMBER).

assert_all([]).
assert_all([H | T]) :-
     assert(term(H)),
     assert_all(T).
```

Esercizio 9

```
nome --> lettera.

nome --> lettera, lettCifra.

lettera --> [a].

lettera --> [b].

cifra --> [0].

cifra --> [1].

lettCifra --> lettera.

lettCifra --> cifra.

lettCifra --> lettera, lettCifra.

lettCifra --> cifra, lettCifra.
```