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

## 11 Luglio 2013 – Tempo a disposizione: 2 h – Risultato: 32/32 punti

### Esercizio 1 (7 punti)

Si considerino le seguenti frasi:

1. *John possiede un cane*
2. *Chiunque possiede dei cani è un amante degli animali*
3. *Chiunque è amante degli animali non uccide animali*
4. *O John o la curiosità hanno ucciso il gatto Felix* (or non esclusivo)
5. *Ogni cane è un animale*
6. *Ogni gatto è un animale*

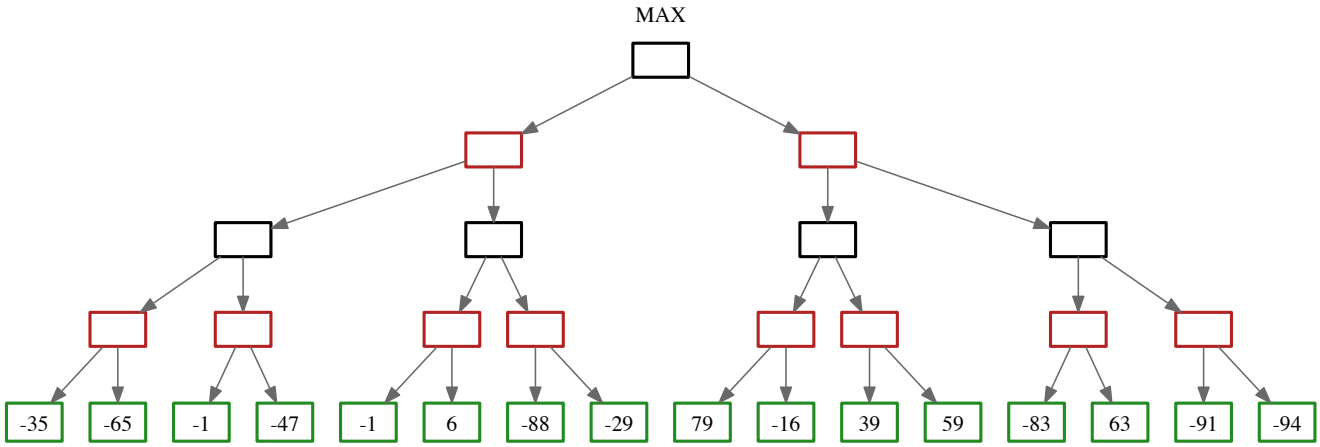
Si formalizzino in *logica dei predicati del I ordine*, utilizzando i seguenti predicati:

- `dog(X)` X è un cane
- `cat(X)` X è un gatto
- `animal(X)` X è un animale
- `animal_lover(X)` X è un amante degli animali
- `owns(X,Y)` X possiede Y
- `kills(X,Y)` X uccide/ha ucciso Y

Infine si trasformino in *clausole* e si dimostri tramite *risoluzione* che *la curiosità ha ucciso Felix*.

### 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)

Dato il seguente programma *Prolog*:

```
foo([H|T],A,R) :- foo(T,[H|A],R).
foo([],A,A).
bar(L,R) :- foo(L,[],R).
```

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

```
?- bar([1,2,3,4,5],X).
```

## Esercizio 4 (5 punti)

Si scriva un predicato `duplicate(L1,L2)`, che, data una lista `L1`, produce la lista `L2` che presenta ogni elemento di `L1` ed un suo duplicato, rispettando l'ordine in cui appaiono in `L1`.

Esempio:

```
?-duplicate([a,b,c],L).
Yes  L=[a,a,b,b,c,c]
?-duplicate([1,2],[1,1,2,2]).
Yes
```

## Esercizio 5 (6 punti)

Si consideri il seguente gioco: si hanno 12 monete ripartite in tre pile distinte. Inizialmente, la prima pila è formata da 4 monete, la seconda da 7 e la terza da 1.

Una certa quantità di monete può essere spostata da una pila all'altra a patto che si raddoppi il numero di monete nella pila di destinazione.

L'obiettivo è raggiungere la configurazione in cui tutte e tre le pile sono composte da 4 monete.

Si disegni l'albero di ricerca nel caso in cui sia applicata una *ricerca a profondità limitata*, con *limite di profondità pari a 3* (il nodo iniziale è etichettato come 0) e si fermi la ricerca dopo aver trovato la prima soluzione. A parità di profondità, si prediligano gli stati che sono stati generati da mosse per le quali la differenza tra il numero di monete nella pila di partenza e quello nella pila di destinazione è maggiore. A parità di questo valore, si preferiscano gli stati che sono stati generati da mosse il cui indice della pila di destinazione è maggiore.

## Esercizio 6 (3 punti)

Si spieghi cosa si intende per *funzione di valutazione ammissibile* e per *funzione di valutazione consistente*.

*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)

## 11 Luglio 2013 – Tempo a disposizione: 45 min – Risultato: 32/32 punti

### Esercizio 7 (10 punti)

Si introducano le idee base a cui si riferiscono le *tecnologie semantiche* e si illustri con un esempio in un'area applicativa il vantaggio della loro adozione.

### Esercizio 8 (10 punti)

Si scriva un meta-interprete `solve(Goal,C)` per clausole scritte nelle forma `clause(Head, Body, Cost)` che, in caso di soluzione con successo per il `Goal`, restituisce in `C` il costo relativo.

Tale costo relativo va inteso come somma dei costi delle clausole utilizzate per la soluzione. Si noti che, per definizione, `true` ha costo `0`.

Si definisca poi il predicato minimale `solutions(G,N,L)` che, utilizzando `solve`, restituisca in uscita il numero `N` di soluzioni e la lista `L` dei costi delle diverse soluzioni.

### Esercizio 9 (12 punti)

Una fabbrica dispone di due macchinari per la trasformazione di materie prime in beni di consumo.

Ad ogni attivazione, la macchina A produce un oggetto α consumando 2 unità di materie prime. La macchina B, invece, produce un oggetto β consumando 6 unità di materie prime oppure 2 oggetti di tipo α e 1 unità di materie prime a seconda della modalità su cui è impostata. Per passare da una modalità all'altra, la macchina B deve subire un processo di riconfigurazione che consuma 1 unità di materie prime. Ovviamente, se il numero di materie prime e oggetti α disponibili è inferiore alla quantità richiesta per una certa operazione, non è possibile eseguire tale operazione.

Si supponga di voler produrre 3 oggetti di tipo α e 2 oggetti di tipo β, sapendo che la fabbrica dispone di 15 unità di materie prime, che nessun oggetto α e β è inizialmente disponibile e che la macchina B è inizialmente impostata sulla prima delle modalità descritte sopra.

Si formalizzi il problema nei termini previsti dall'*algoritmo STRIPS*, avendo cura di specificare lo *stato iniziale*, lo *stato obiettivo* e l'insieme degli *operatori* (ognuno corredato dai propri insiemi di *requisiti* ed *effetti*): `MAKE_ALPHA`, `MAKE_BETA` e `CHANGE_MODE`. A tal proposito, si utilizzino i seguenti predicati:

- `MATERIALS(x)` x unità di materie prime disponibili
- `ALPHA(x)` x unità di oggetti α disponibili
- `BETA(x)` x unità di oggetti β disponibili
- `MODE(0)` la macchina B crea 1 oggetto β con 6 unità di materie prime
- `MODE(1)` la macchina B crea 1 oggetto β con 2 oggetti α e 1 unità di materie prime

Inoltre si introduca brevemente il concetto di pianificatore gerarchico con riferimento all'esempio del problema appena esposto.

*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
## 11 Luglio 2013 – Soluzioni

### Esercizio 1

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

a. *John possiede un cane*
   ∃X: dog(X) ∧ owns(john,X)
   {dog(d1), owns(john,d1)}

b. *Chiunque possiede dei cani è un amante degli animali*
   ∀X ∀Y: (dog(Y) ∧ owns(X,Y) ⇒ animal_lover(X))
   {¬dog(Y) ∨ ¬owns(X,Y) ∨ animal_lover(X)}

c. *Chiunque è amante degli animali non uccide animali*
   ∀X ∀Y: (animal_lover(X) ∧ animal(Y) ⇒ ¬kills(X,Y))
   {¬animal_lover(X) ∨ ¬animal(Y) ∨ ¬kills(X,Y)}

d. *O John o la curiosità hanno ucciso il gatto Felix*
   (kills(john,felix) ∨ kills(curiosity,felix)) ∧ cat(felix)
   {kills(john,felix) ∨ kills(curiosity,felix), cat(felix)}

e. *Ogni cane è un animale*
   ∀X: dog(X) ⇒ animal(X)
   {¬dog(X) ∨ animal(X)}

f. *Ogni gatto è un animale*
   ∀X: cat(X) ⇒ animal(X)
   {¬cat(X) ∨ animal(X)}

g. *Goal: la curiosità ha ucciso Felix*
   ¬kills(curiosity,felix)
   {¬kills(curiosity,felix)}

*Teoria a clausole*:

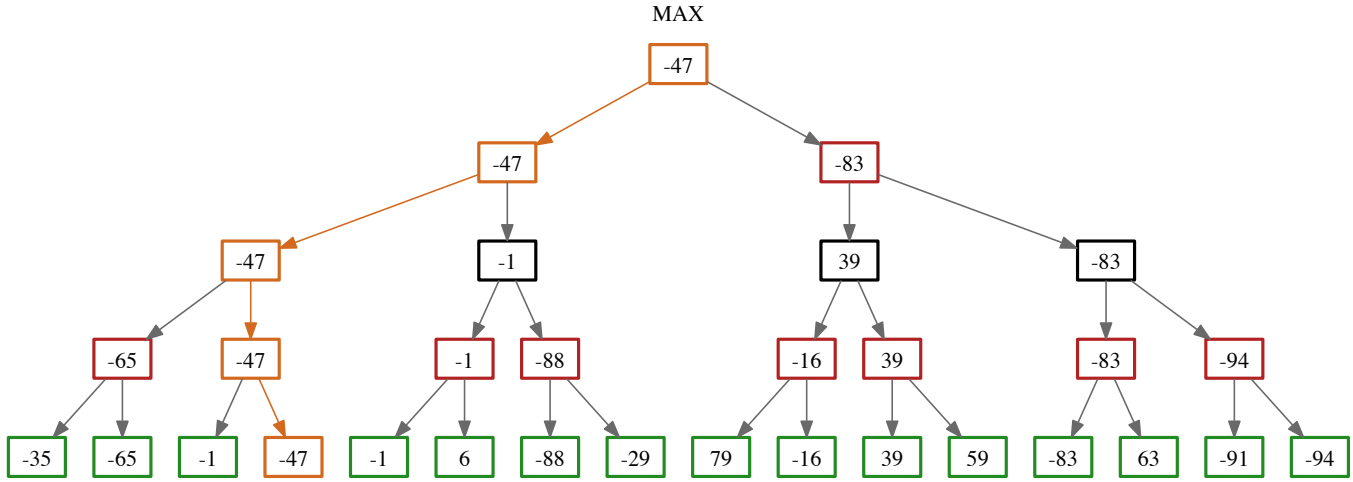
1. dog(d1)
2. owns(john,d1)
3. ¬dog(Y) ∨ ¬owns(X,Y) ∨ animal_lover(X)
4. ¬animal_lover(X) ∨ ¬animal(Y) ∨ ¬kills(X,Y)
5. kills(john,felix) ∨ kills(curiosity,felix)
6. cat(felix)
7. ¬dog(X) ∨ animal(X)
8. ¬cat(X) ∨ animal(X)
9. ¬kills(curiosity,felix)

*Risoluzione*:

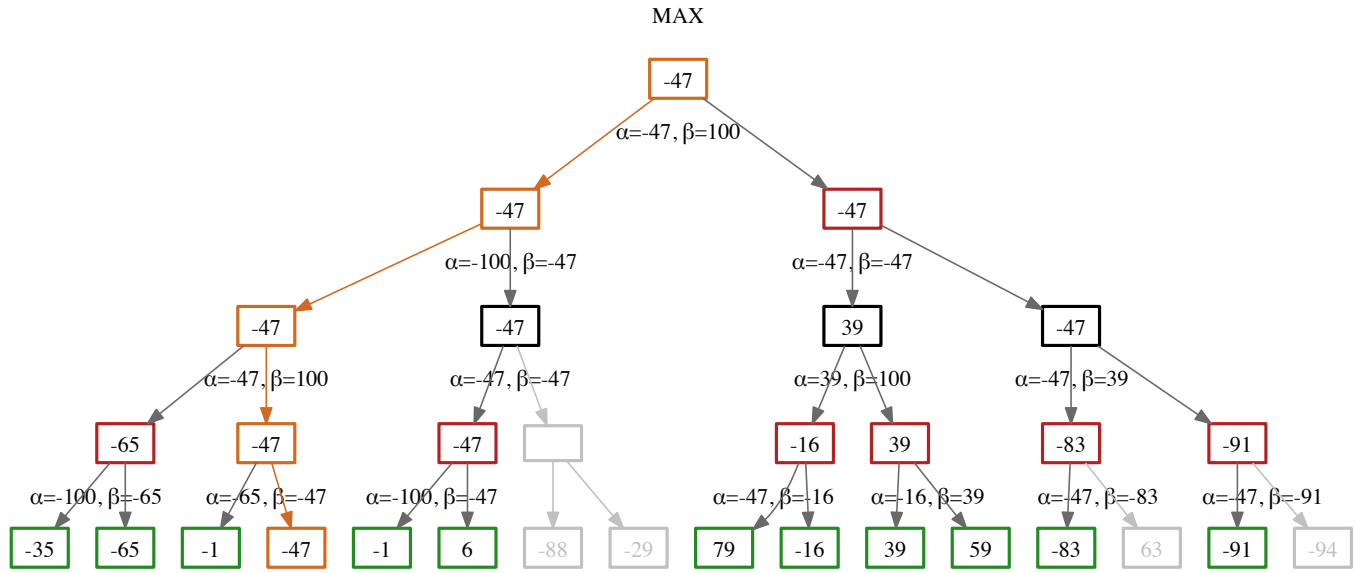
| | | | |
|---|---|---|---|
| 10. = 9. ∪ 5. | kills(john,felix) | {} |
| 11. = 10. ∪ 4. | ¬animal_lover(john) ∨ ¬animal(felix) | {X/john,Y/felix} |
| 12. = 11. ∪ 8. | ¬animal_lover(john) ∨ ¬cat(felix) | {X/felix} |
| 13. = 12. ∪ 6. | ¬animal_lover(john) | {} |
| 14. = 13. ∪ 3. | ¬dog(Y) ∨ ¬owns(john,Y) | {X/john} |
| 15. = 14. ∪ 2. | ¬dog(d1) | {Y/d1} |
| 16. = 15. ∪ 1. | □ | |

## Esercizio 2

*min-max:*



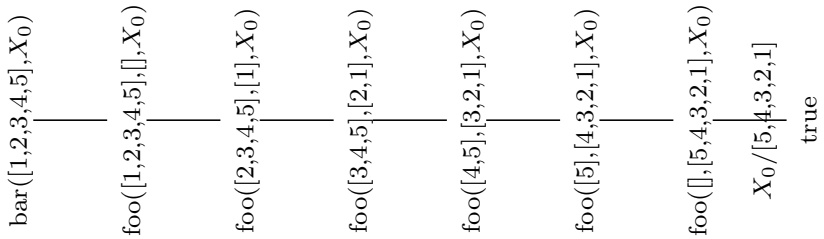

*alfa-beta:*




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

## Esercizio 3

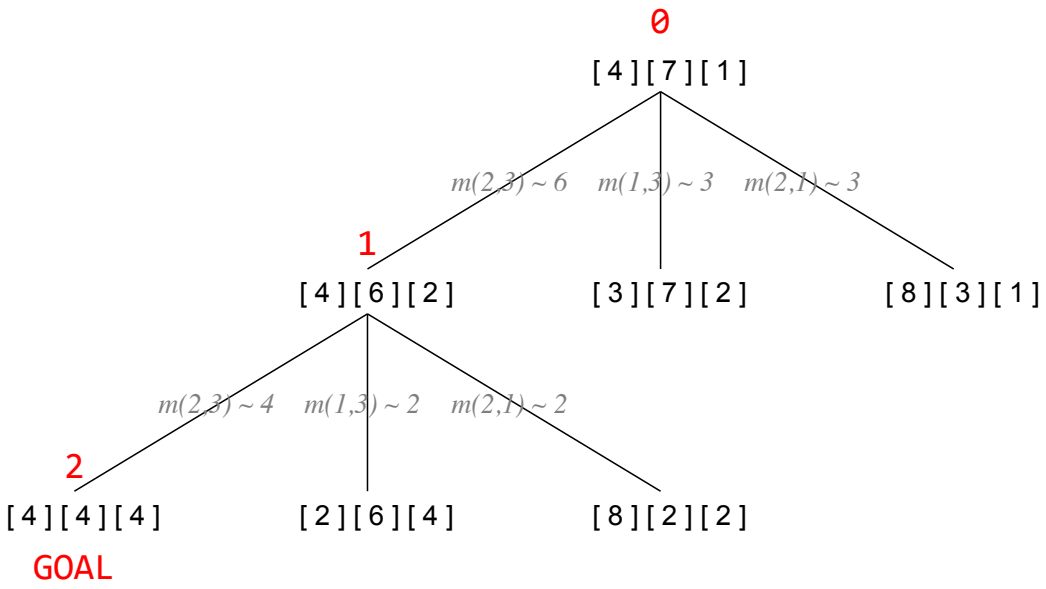
*Albero SLD:*




## Esercizio 4

```
% duplicate(L1,L2)
% ----------------
% Duplicates the elements of a list L1 in the list L2.
% L2 is a list containing all the elements of list L1 twice, in the same order.
duplicate([],[]).
duplicate([X|Xs],[X,X|Ys]) :- duplicate(Xs,Ys).
```

## Esercizio 5




## Esercizio 6

Vedi slides del corso.

# FONDAMENTI DI INTELLIGENZA ARTIFICIALE – 2° parte
# 11 Luglio 2013 – Soluzioni

## Esercizio 7

Vedi slides del corso.

## Esercizio 8

```
solve(true,0).
solve(A,B,C) :- solve(A,C1), solve(B,C2), C is C1+C2.
solve(A,C) :- clause(A,B,C1), solve(B,C2), C is C1+C2.

solutions(G,N,L) :- findall(C,solve(G,C),L),length(L,N).
length([],0).
length([X|Y],N) :- length(Y,N1), N is N1+1.
```

## Esercizio 9

Stato iniziale:
`MATERIALS(15), ALPHA(0), BETA(0), MODE(0)`

Stato obiettivo:
`ALPHA(3), BETA(2)`

Operazioni:

`MAKE_ALPHA()`
- Pre: `MATERIALS(x), x≥2`
- Post: `¬MATERIALS(x), MATERIALS(x-2), ¬ALPHA(y), ALPHA(y+1)`

`MAKE_BETA()`
- Pre: `MODE(x), MATERIALS(y), ALPHA(z), x=0 ∨ x=1, y≥6-5x, z≥2x`
- Post: `¬MATERIALS(y), MATERIALS(y-6+5x), ¬ALPHA(z), ALPHA(z-2x), ¬BETA(k), BETA(k+1)`

N.B.: siccome x può assumere solo il valore `0` o `1`, la funzione `6-5x` restituisce rispettivamente `6` o `1`. Dunque se `x=0`, l'espressione `y≥6-5x` verifica che ci siano almeno `6` unità di materie prime disponibili, mentre se `x=1` verifica che ce ne sia almeno `1`. Analogamente accade con `z≥2x` per gli oggetti α.

`CHANGE_MODE()`
- Pre: `MODE(x), MATERIALS(y), x=0 ∨ x=1, y≥1`
- Post: `¬MATERIALS(y), MATERIALS(y-1), ¬MODE(z), MODE((z+1) mod 2)`