# FONDAMENTI DI INTELLIGENZA ARTIFICIALE (8 CFU)

## 11 Settembre 2014 – Tempo a disposizione: 2 h – Risultato: 32/32 punti

### Esercizio 1 (6 punti)

Si esprimano in logica dei predicati del I ordine le seguenti frasi:

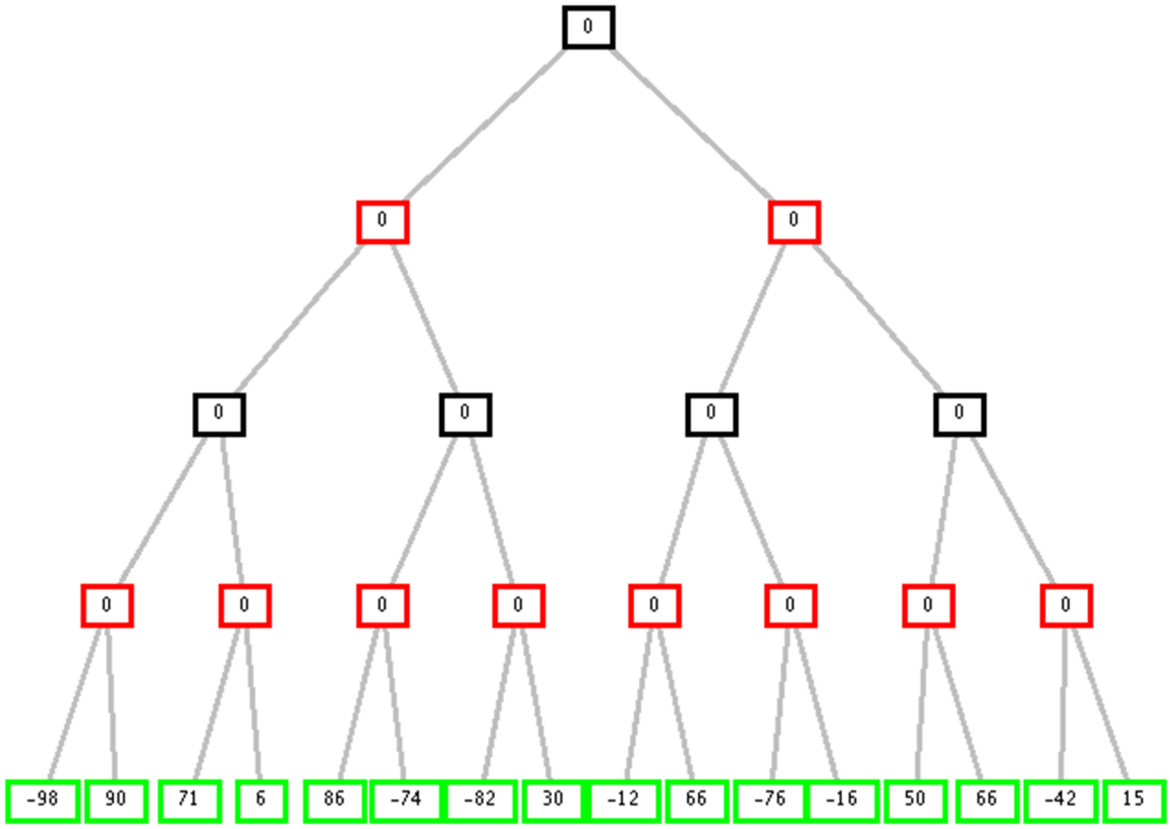
- Se si fa parte del personale dell'università allora si è "docente" o "tecnico".
- Se uno è un docente allora esiste almeno un corso universitario tenuto da questi.
- Carlo lavora all'università.
- Carlo non tiene alcun corso.

Si usi poi il principio di risoluzione per dimostrare che Carlo è un "tecnico".

Si usi il seguente vocabolario: predicati: doc(X), tec(X), univ(X), corso(Y), tiene(X,Y) con significato "X tiene Y"; costanti: carlo.

### Esercizio 2 (4 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.




Si mostri come l'algoritmo min-max risolve il problema, indicando con una freccia la strada scelta dall'algoritmo. Si mostrino poi i tagli alfa-beta.

### Esercizio 3 (4 punti)

Definire nel linguaggio *Prolog* il predicato `sottoin (L1,L2)` , che risulta vero quando, date due liste (ground) di interi, `L2` rappresenta un sottoinsieme degli elementi della lista `L1`. Esempio:

```
?- sottoin([-1,2,5,4], [-1,4]).
yes

?- sottoin([-1,2,4,5], [4,7,9]).
no
```

## Esercizio 4 (5 punti)

Si definisca un meta interprete Prolog `solve(Goal)` che, qualora fallisca nella dimostrazione di un goal, provveda a chiedere all'utente se considerare comunque come vero o no tale goal. Nel primo caso il meta interprete deve continuare nel cercare di dimostrare il goal iniziale, nel secondo caso ovviamente deve terminare la dimostrazione. A tal scopo, si supponga di avere a disposizione il predicato `ask_user(Sub_goal)` che provvede a chiedere all'utente se considerare o meno di successo il sotto-goal `Sub_goal`. Ad esempio, dato il programma:

```
ask_user(S) :-
      write("Considero vero il predicato: "),
      write(S),
      write(" ??? "),
      read(stdin, Answer),
      Answer=y.
p(X) :- q(X), r(X), z(X).
q(1).
z(1).

?- solve(p(X)).
Considero vero il predicato: r(1) ??? y
yes, X/1.
```

## Esercizio 5 (6 punti)

Si modelli come CSP il seguente problema di cripto-aritmetica:

```
  DUE +
  DUE =
 ----
  SEI
```

nel quale ciascuna lettera può assumere un valore tra 1 e 6, le lettere corrispondono tutte a cifre diverse, i riporti siano pari a zero (quindi non è necessario modellarli).
Si risolva poi il problema applicando Forward Checking. Si selezionino le variabili secondo MRV, e in caso di parità, si selezionino le variabili secondo l'ordine: E, I, U, D, S (in pratica prima le vocali e poi le consonanti); si assegnino i valori del dominio secondo l'ordine sugli interi (in senso crescente).

## Esercizio 6 (4 punti)

Si confrontino in maniera concisa le proprietà (complessità spaziale, temporale, ottimalità e completezza) delle strategie di ricerca non informate.

## Esercizio 7 (3 punti)

Si dia la definizione di correttezza e di completezza per la logica deduttiva.
Il risolutore del linguaggio Prolog è completo?

# FONDAMENTI DI INTELLIGENZA ARTIFICIALE
## 11 Settembre 2014 – Soluzioni

### Esercizio 1

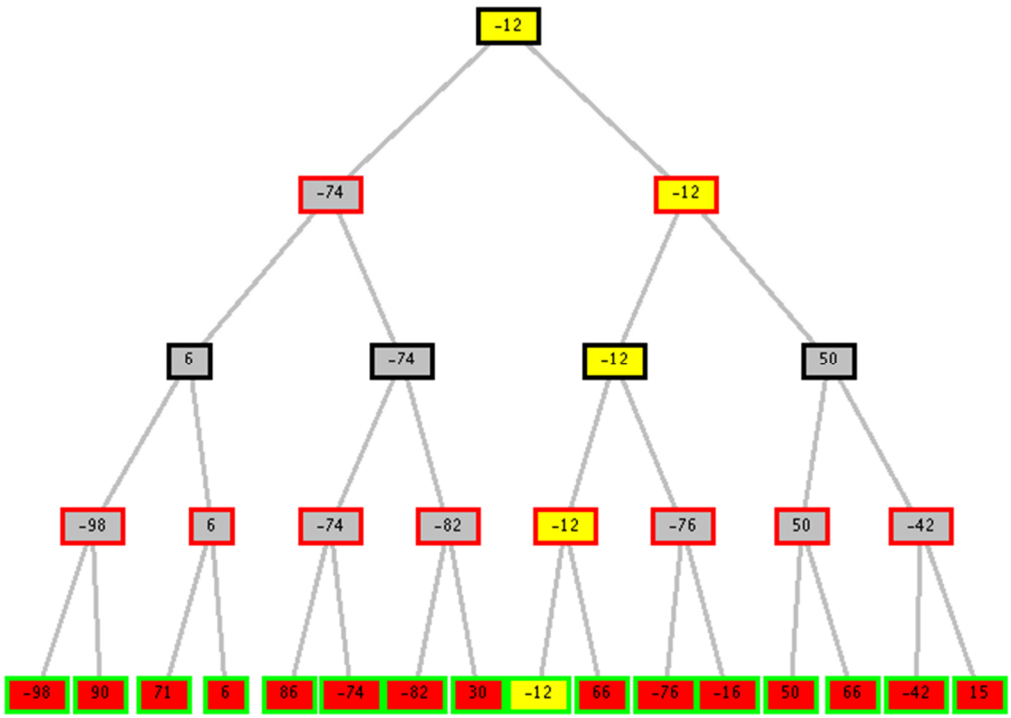
$\forall$ X  univ(X) $\rightarrow$ doc(X) ex-or tec(X)
$\forall$ X  doc(X) $\rightarrow \exists$ Y corso(Y) and tiene(X,Y)
univ(carlo)
not $\exists$ Y ( corso(Y) and tiene(carlo,Y) )
Formula da dimostrare: tec(carlo)

*Trasformazione in clausole:*
C1: not univ(X) or doc(X) or tec(X)
C2: not univ(X) or not doc(X) or not tec(X)
C3: not doc(X) or corso(f(X))
C4: not doc(X) or tiene(X,f(X))
C5: univ(carlo)
C6: not tiene(carlo,Y) or not corso(Y)
Goal negato: not tec(carlo)

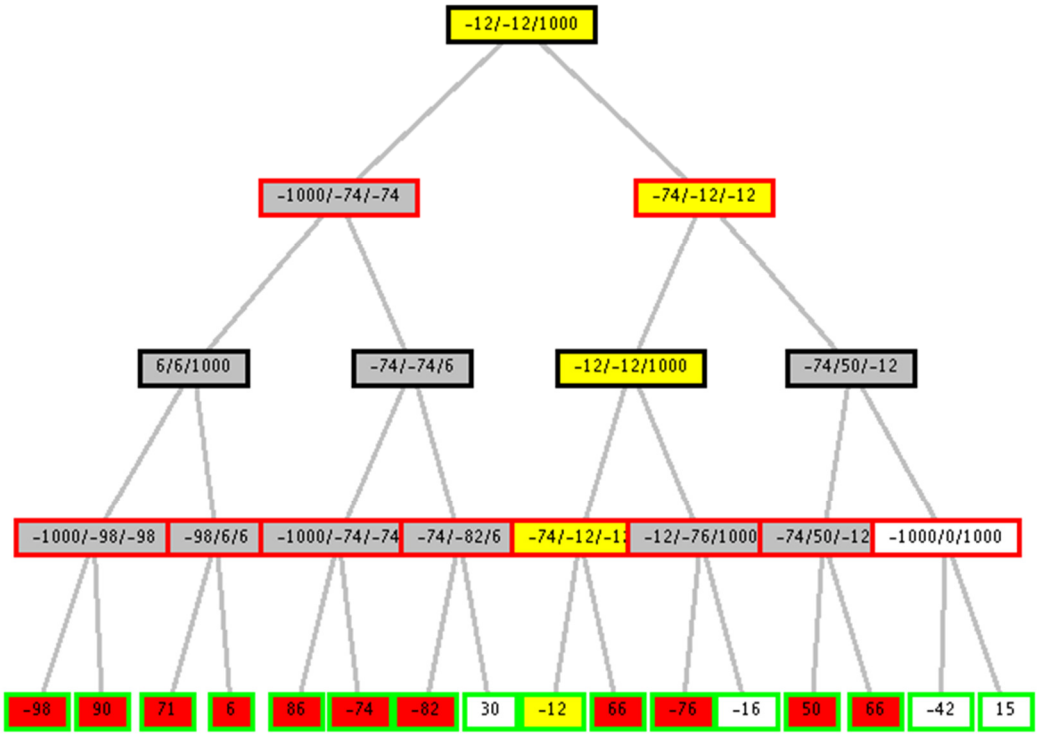
*Applicando il Principio di Risoluzione:*
C7 (da C1 e GN): not univ(carlo) or doc(carlo)
C8 (da C7 e C5): doc(carlo)
C9 (da C8 e C3): corso(f(carlo))
C10 (da C8 e C4): tiene(carlo,f(carlo))
C11 (da C10 e C6): not corso(f(carlo))
C12 (da C11 e C9): clausola vuota

### Esercizio 2

Min-Max:



Alfa-Beta:




I nodi in bianco sono quelli tagliati.

## Esercizio 3

```
subset(_,[]).
subset(L, [X|L1]) :- member(X,L), subset(L,L1).

member(X, [X|_]) :- !.
member(X, [Y|L]) :- member (X,L).
```

Nota: questa soluzione ammette che sia L1 che L2 abbiano elementi ripetuti, ma non c'è controllo del numero di ripetizioni…

## Esercizio 4

```
solve(true).
solve((X,Y)) :- !, solve(X), solve(Y).
solve(X) :- clause(X, Body), solve(Body).
solve(X) :- ask_user(X).
```

## Esercizio 5

Variabili:
E :: [1,2,3,4,5,6]
I :: [1,2,3,4,5,6]
U :: [1,2,3,4,5,6]
D :: [1,2,3,4,5,6]
S :: [1,2,3,4,5,6]

Vincoli:
C1: all_different(E,I,U,D,S)
C2: E+E = I
C3: U + U = E
C4: D + D = S

Ulteriori vincoli derivanti dal fatto che i riporti sono pari a zero:
C5: E+E<10 (cioè E<5)
C6: U+U<10 (cioè U<5)
C7: D+D<10 (cioè D<5)
NOTA: I vincoli C5, C6 e C7 possono essere applicati immediatamente per ridurre i corrispondenti domini. Tale passo è equivalente a garantire la "node consistency". La soluzione di seguito riportata fa riferimento all'applicazione della tecnica di "Forward Checking", senza aver applicato la node consistency in precedenza.

Primo passo: E=1, propagando C1, C2 e C3:
E = 1
I :: [2]
U :: []
D :: [2,3,4,5,6]
S :: [2,3,4,5,6]

La variabile U ha dominio vuoto, quindi si esegue un passo di backtracking e si assegna: E=2, propagando C1, C2 e C3:
E = 2
I :: [4]
U :: [1]
D :: [1,3,4,5,6]
S :: [1,3,4,5,6]

Secondo passo: I=4, propagando C1:
E = 2
I = 4
U :: [1]
D :: [1,3,5,6]
S :: [1,3,5,6]

Terzo passo: U=1, propagando C1:
E = 2
I = 4
U = 1
D :: [3,5,6]
S :: [3,5,6]

Quarto passo: D=3, propagando C1 e C4:
E = 2
I = 4
U = 1
D = 3
S :: [6]

Ultimo passo: S=6.

## Esercizio 6

Vedi slide del corso.

## Esercizio 7

Vedi slide del corso.