

# FONDAMENTI DI INTELLIGENZA ARTIFICIALE (8 CFU)

15 Giugno 2015 – Tempo a disposizione: 2 h – Risultato: 32/32 punti

## Esercizio 1 (punti 5)

Si considerino le seguenti affermazioni.

(1) P e Q sono due punti del piano.

(2) R1 e R2 sono due regioni del piano.

(3) P appartiene a R1 e Q appartiene a R2.

(4) Se due regioni coincidono, allora un punto appartenente alla prima regione appartiene anche alla seconda regione.

(5) R1 e R2 coincidono.

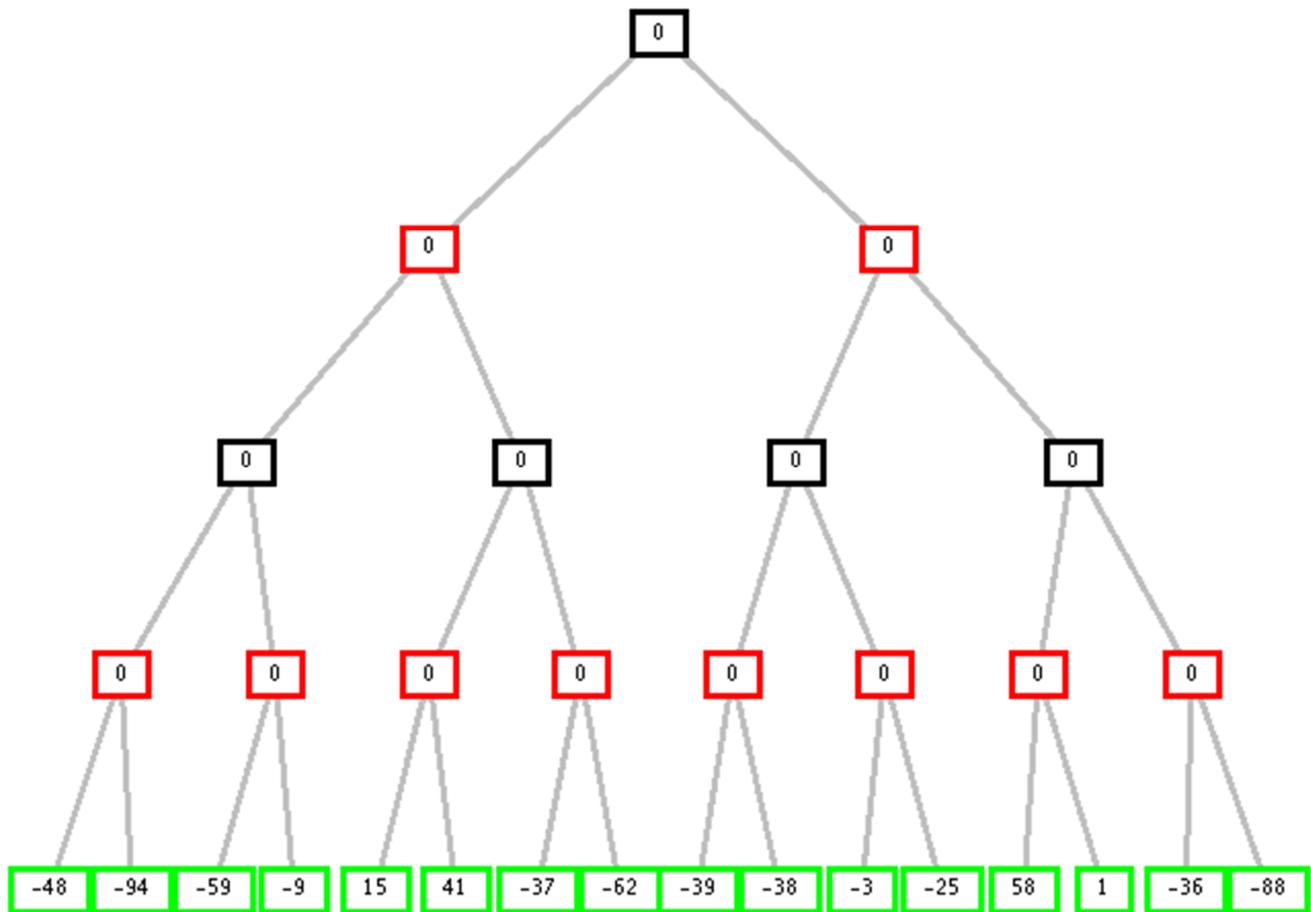
Si rappresentino le affermazioni (1)-(5) come formule della logica dei predicati del primo ordine.

Si applichi la risoluzione alla teoria formata dalle formule ottenute sopra per verificare se vale l'affermazione seguente:

(a) P appartiene a R2.

## Esercizio 2 (punti 5)

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 (punti 5)

Si consideri il seguente problema di soddisfacimento di vincoli.

Variabili: A, B, C, D, E.

Domini:  $D_A = \{1,2,3,4\}$ ,  $D_B = \{1,2,3,4\}$ ,  $D_C = \{1,2,3,4,5\}$ ,  $D_D = \{1,2,3,4\}$ ,  $D_E = \{1\}$ .

Vincoli:  $A > B$ ,  $B > C$ ,  $C \geq D$ ,  $D > E$ .

Si applichi la consistenza d'arco partendo dai domini iniziali.

Dal risultato, cosa possiamo concludere rispetto alle soluzioni del problema?

### Esercizio 4 ( punti 5)

Per decomporre un atomo in una lista di caratteri (o meglio nella lista dei loro codici ASCII), o per comporre un atomo da una lista di codici di caratteri, si può usare il predicato

**name (Atomo, Lista)**

Una possibile applicazione è quella di sapere se un atomo ha un certo prefisso. Ad esempio gli studenti di un corso di Diploma hanno un login name tipo diaaa241 mentre quelli di un corso di Laurea hanno un login name tipo lie241. Possiamo distinguerli usando in predicato:

**prefisso (Prefix, Atom) .**

che dovrà rispondere affermativamente ai due goal:

?- **prefisso (diaaa, diaaa241) .**

?- **prefisso (lie, lie241) .**

Scrivere il programma Prolog che realizza tale comportamento, accettando prefissi di qualunque dimensione.

### Esercizio 5 ( punti 6)

Si consideri il seguente programma Prolog:

```
nodupl ([], []).  
nodupl ([X|Xs], Ys) :-  
    member (X, Xs),  
    nodupl (Xs, Ys).  
nodupl ([X|Xs], [X|Ys]) :-  
    \+(member (X, Xs)),  
    nodupl (Xs, Ys).
```

Tenendo presente che l'operatore \+ indica la negazione per fallimento in Prolog, si mostri l'albero SLD-NF che si ottiene per il goal Prolog:

?-**nodupl ([1, 1], Y) .**

### Esercizio 6 ( punti 2)

Si spieghi brevemente in cosa consiste la ricerca locale, vantaggi e svantaggi e si specifichi l'algoritmo base dell'Hill-climbing.

### Esercizio 7 ( punti 4)

Si spieghi brevemente in cosa consiste la meta-interpretazione in Prolog specificandone vantaggi e svantaggi. Si mostri il meta-interprete per Prolog puro in Prolog e come modificarlo per tenere in considerazione clausole che possono contenere anche sottogoals in or (simbolo ";"). Ad esempio: (a(X);b(Y)) indica l'or dei due sottogoals a(X) e b(Y).

# FONDAMENTI DI INTELLIGENZA ARTIFICIALE

Giugno 2015 – Soluzioni

## Esercizio 1

Nota: nella soluzione diversamente da Prolog le variabili cominciano con una lettera minuscola mentre predicati e costanti con una lettera maiuscola

Le formule che rappresentano le affermazioni sono:

1.  $\text{Point}(P) \wedge \text{Point}(Q)$
2.  $\text{Region}(R1) \wedge \text{Region}(R2)$
3.  $\text{Belongs}(P, R1) \wedge \text{Belongs}(Q, R2)$
4.  $\forall x, y \text{ Region}(x) \wedge \text{Region}(y) \wedge \text{Same}(x, y) \rightarrow (\forall z \text{ Point}(z) \wedge \text{Belongs}(z, x) \rightarrow \text{Belongs}(z, y))$
5.  $\text{Same}(R1, R2)$

La corrispondente KB (con le formule in forma normale congiuntiva e la tesi negata) è:

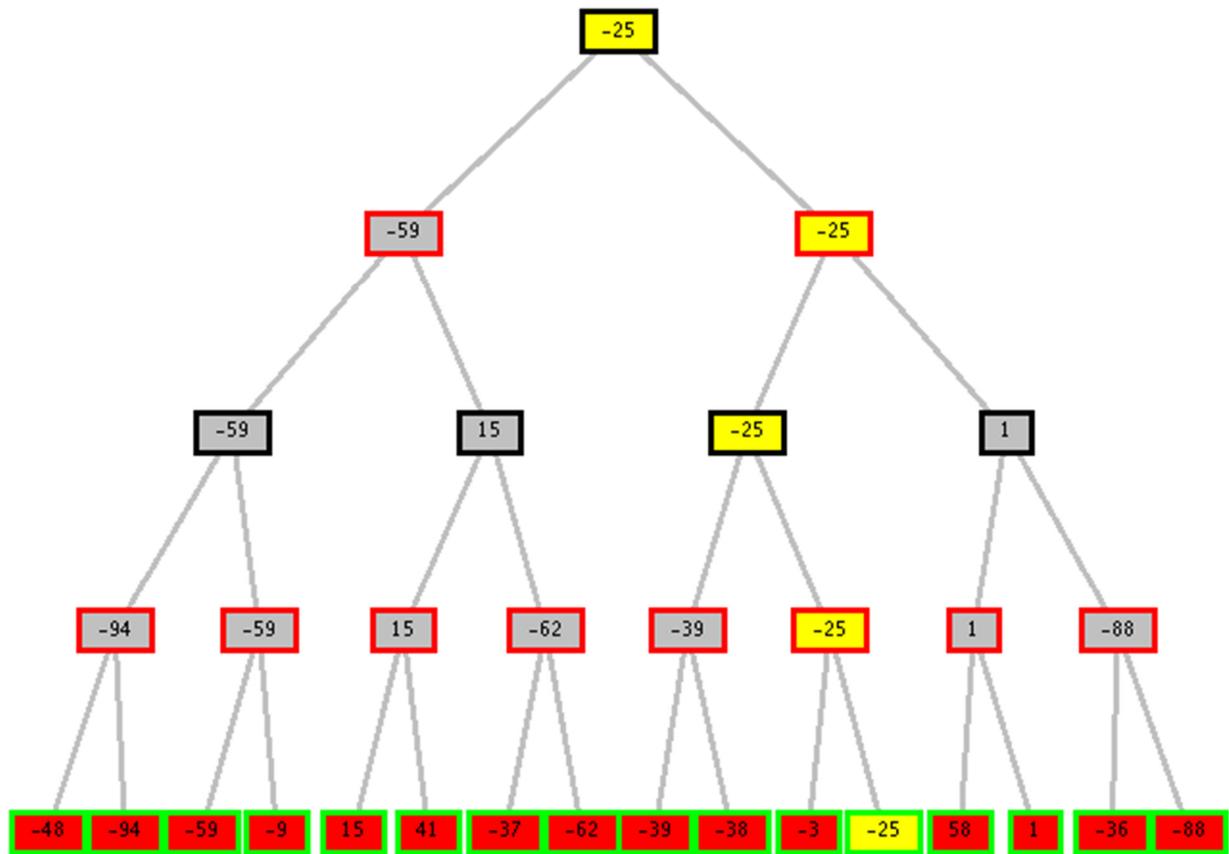
1.  $\text{Point}(P)$
2.  $\text{Point}(Q)$
3.  $\text{Region}(R1)$
4.  $\text{Region}(R2)$
5.  $\text{Belongs}(P, R1)$
6.  $\text{Belongs}(Q, R2)$
7.  $\neg \text{Region}(x) \vee \neg \text{Region}(y) \vee \neg \text{Same}(x, y) \vee \neg \text{Point}(z) \vee \neg \text{Belongs}(z, x) \vee \text{Belongs}(z, y)$
8.  $\text{Same}(R1, R2)$
9.  $\neg \text{Belongs}(P, R2)$

Da cui, applicando la risoluzione:

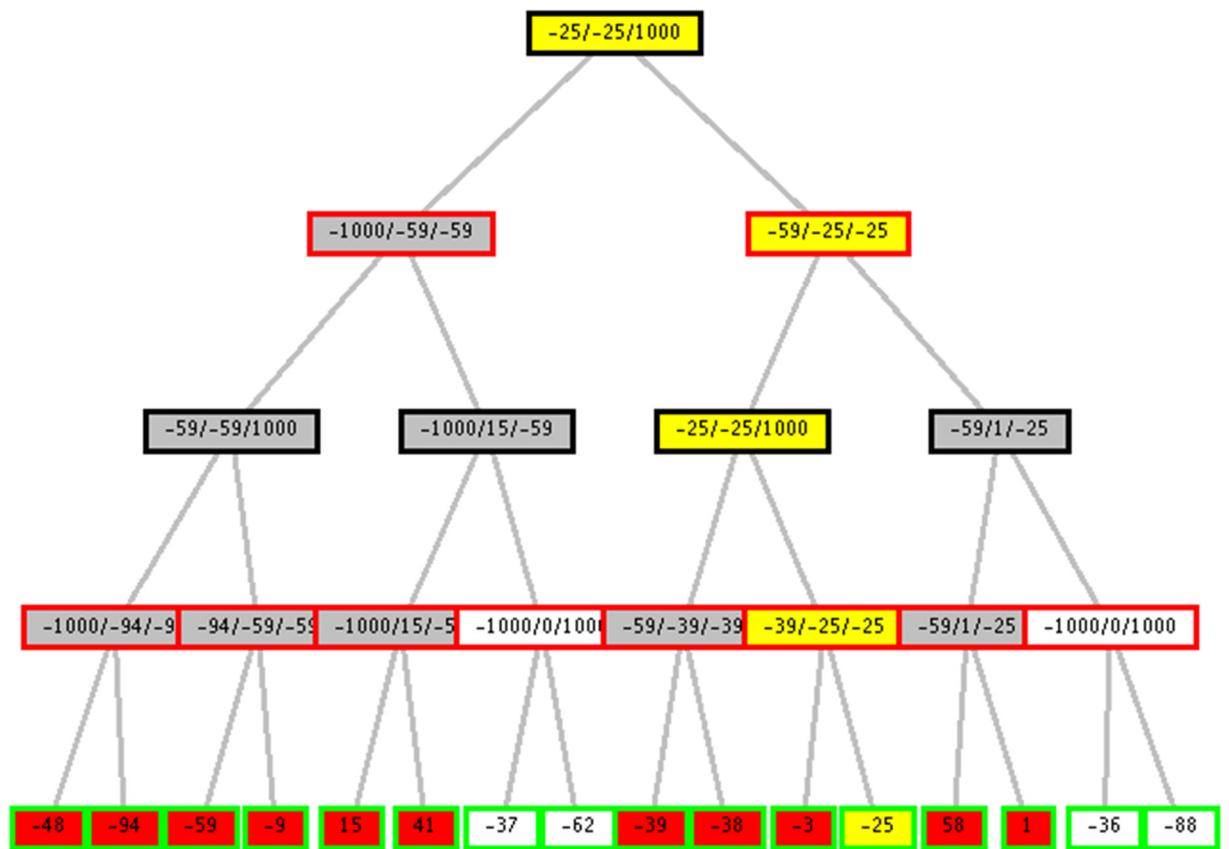
9.  $\neg \text{Region}(R1) \vee \neg \text{Region}(R2) \vee \neg \text{Point}(z) \vee \neg \text{Belongs}(z, R1) \vee \text{Belongs}(z, R2) \quad R(7, 8, \{x/R1, y/R2\})$
10.  $\neg \text{Region}(R2) \vee \neg \text{Point}(z) \vee \neg \text{Belongs}(z, R1) \vee \text{Belongs}(z, R2) \quad R(3, 9, \{\})$
11.  $\neg \text{Point}(z) \vee \neg \text{Belongs}(z, R1) \vee \text{Belongs}(z, R2) \quad R(4, 10, \{\})$
12.  $\neg \text{Belongs}(P, R1) \vee \text{Belongs}(P, R2) \quad R(1, 11, \{z/P\})$
13.  $\text{Belongs}(P, R2) \quad R(5, 12, \{\})$
14.  $\perp \quad R(\neg a, 13, \{\})$

## Esercizio 2

Min-Max:



Alfa-Beta:



I nodi che portano alla soluzione sono in giallo, quelli tagliati in bianco.

### Esercizio 3 ( punti)

Considero gli archi dei vincoli in sequenza:

A → B:  $D_A = \{2, 3, 4\}$

B → A:  $D_B = \{1, 2, 3\}$

B → C:  $D_B = \{2, 3\}$

C → B:  $D_C = \{1, 2\}$

C → D:  $D_C$  immutato

D → C:  $D_D = \{1, 2\}$

D → E:  $D_D = \{2\}$

E → D:  $D_E$  immutato

Considero di nuovo gli archi che sono stati rimessi in coda:

A → B:  $D_A = \{3, 4\}$

B → A:  $D_B$  immutato

B → C:  $D_B$  immutato

C → B:  $D_C$  immutato

C → D:  $D_C = \{2\}$

D → C:  $D_D$  immutato

E → D:  $D_E$  immutato

Considero di nuovo gli archi che sono stati rimessi in coda:

B → A:  $D_B$  immutato

B → C:  $D_B = \{3\}$

D → C:  $D_D$  immutato

Considero di nuovo gli archi che sono stati rimessi in coda:

A → B:  $D_A = \{4\}$

C → B:  $D_C$  immutato

Considero di nuovo gli archi che sono stati rimessi in coda:

B → A:  $D_B$  immutato

Tutti gli archi sono stati rimossi dalla coda e l'algoritmo termina.

Visto che tutti i domini contengono un solo valore, si può concludere che il CSP considerato ha una sola soluzione: A=4, B=3, C=2, D=2, E=1.

### Esercizio 4 ( punti)

Possiamo scrivere il predicato che ci interessa in questo modo:

**prefisso (Prefix, Atom) :-**

```
    name (Prefix, Plist), name (Atom, Alist), conc (Plist, _, Alist).
```

prima scomponiamo con **name/2 Prefix** in una lista di caratteri **Plist**, e analogamente facciamo per **Atom** ottenendo **Alist**. Poi con il predicato **conc/2** controlliamo che i caratteri di **Plist** siano quelli iniziali di **Alist**.

**prefisso (Prefix, Atom) :-**

```
    name (Prefix, Plist), % scompone Prefix
```

```
    name (Atom, Alist), % scompone Atom
```

```
    conc (Plist, _, Alist). % Plist e' la parte iniziale di Alist
```

```
% concatena due liste
```

```
conc ([], L, L).
```

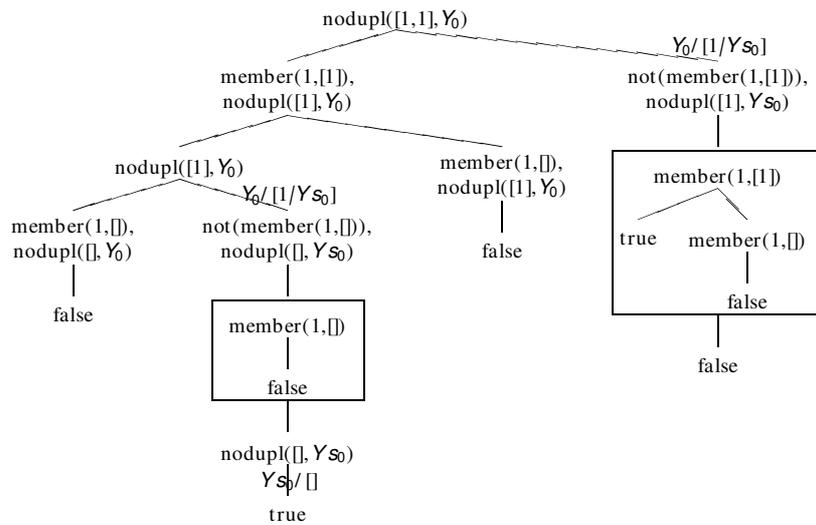
```
conc ([X|L1], L2, [X|L3]) :- conc (L1, L2, L3).
```

```
%Oppure, usando un altro predicato check/2 al posto di conc/3:
check([], _).
check([H | T1], [H | T2]) :- check(T1, T2).
```

### Esercizio 5

Supponendo che la member sia così definita:

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



### Esercizio 6

Vedi slides del corso.

### Esercizio 7

Vedi slides del corso. Metainterprete con OR (;) in Prolog.

```
solve(true).
solve((A,B)) :- solve(A), solve(B).
solve((A;B)) :- solve(A).
solve((A;B)) :- solve(B).
solve(X) :- clause(X,B), solve(B).
```