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

12 Settembre 2013 – Tempo a disposizione: 2 h – Risultato: 32/32 punti

Esercizio 1 (6 punti)

Si considerino le seguenti frasi:

1. *Le zebre sono mammiferi di grandezza media con il manto a strisce*
2. *Il manto a strisce è non uniforme*
3. *I mammiferi sono animali a sangue caldo*
4. *Nina è una zebra*

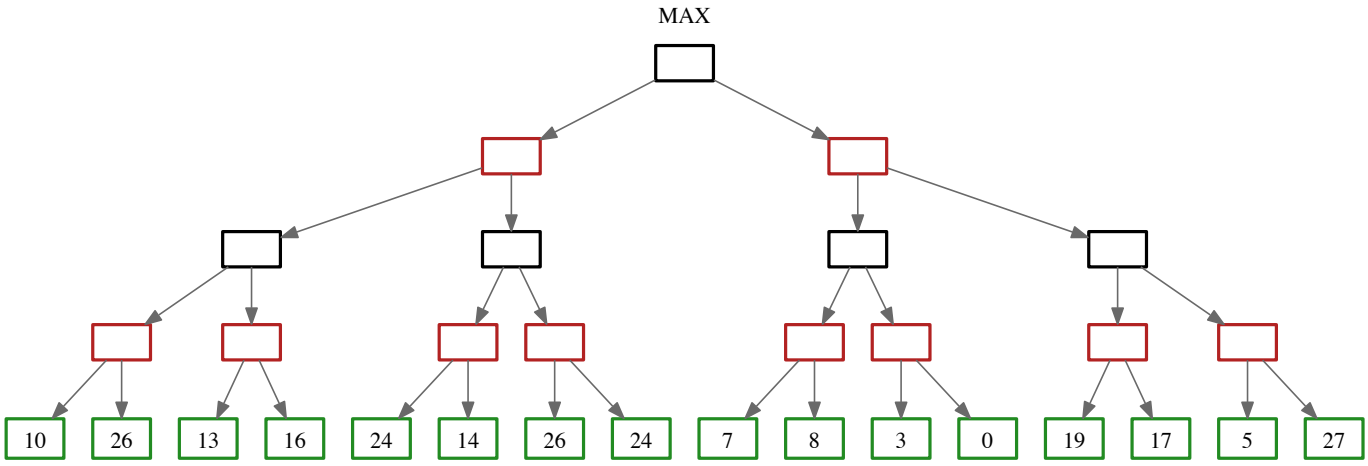
Si formalizzino in *logica dei predicati del I ordine*, utilizzando i seguenti predicati:

- `zebra(X)` X è una zebra
- `mammal(X)` X è un mammifero
- `small (X)` X è di grandezza piccola
- `medium(X)` X è di grandezza media
- `large(X)` X di grandezza grande
- `striped(X)` X ha il manto a strisce
- `uneven(X)` X ha il manto non uniforme
- `warmblood(X)` X ha il sangue caldo

Infine si trasformino in *clausole* e si dimostri tramite *risoluzione* che *Nina ha manto non uniforme*.

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 (7 punti)

Il seguente programma *Prolog* restituisce nel secondo argomento il numero `N` di liste contenute nella lista passata come primo argomento.

```
sublist([],0).
sublist([H|T],N) :- isList(H), !, sublist(T,NT), N is NT+1.
sublist([H|T],N) :- sublist(T,N).
islist([]).
islist([_|_]).
```

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

```
?- sublist([[a,b],g,[f,h]],X).
```

Esercizio 4 (5 punti)

Si scriva un programma *Prolog* `listint(M,N,L)` che generi in uscita una lista `L` contenente i numeri interi fra i valori `M` ed `N` indicati (estremi compresi).
Esempi:

```
?- listint(3,5,L).
Yes  L=[3,4,5]
?- listint(5,3,L).
No
```

Esercizio 5 (7 punti)

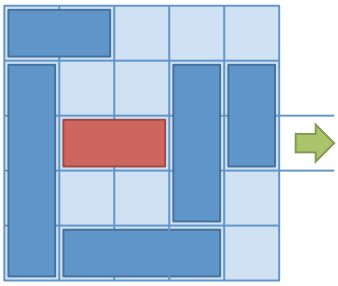


Figura 1

Nel rompicapo *Rush Hours* si ha un tavolo di gioco che rappresenta un parcheggio, in cui sono parcheggiati vari veicoli di diversa lunghezza. Nel parcheggio c'è una sola uscita, rappresentata dalla freccia sulla destra. Si veda, a titolo di esempio, la configurazione in Figura 1.

I veicoli possono essere orientati in verticale o in orizzontale; i veicoli verticali possono essere mossi solo in su o in giù, mentre quelli orizzontali possono essere mossi solo verso destra o verso sinistra. Lo scopo del gioco è fare uscire l'automobile rossa, spostando il numero minimo di veicoli.

Si supponga che ogni veicolo possa essere mosso di quante caselle si desidera, ma soltanto una volta. Chiaramente, se si ha un veicolo che è già stato mosso, che è frapposto fra l'auto rossa e l'uscita, si è in una situazione di fallimento.

Si consideri il problema come un *problema di ricerca in uno spazio degli stati*, in cui uno stato è finale se non ci sono veicoli frapposti fra l'auto rossa e l'uscita. Si consideri l'*algoritmo A**, utilizzando la seguente euristica:

- L'euristica conta con peso 2 le auto che sono
  - frapposte fra l'auto rossa e l'uscita
  - e non possono essere mosse nella prossima mossa (perché hanno un'auto davanti o dietro)
- considera con peso 1 le auto che sono
  - frapposte fra l'auto rossa e l'uscita
  - e che possono essere mosse nella prossima mossa

Questa euristica è ammissibile?

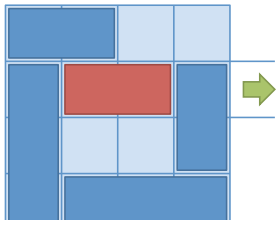


Figura 2

Si mostri come l'*algoritmo A** risolve il problema **partendo dalla configurazione descritta dalla figura a destra (Figura 2)**. Si supponga di provare prima i movimenti in orizzontale, poi quelli in verticale; a parità di questa condizione, si provino prima i blocchi più in alto, poi quelli in basso; a parità di tali condizioni, si muovano prima quelli a sinistra, poi quelli a destra.

Esercizio 6 (2 punti)

Si spieghi con un esempio la differenza fra *backward* e *forward chaining* e si discuta quando è preferibile l'applicazione di uno e dell'altro con adeguate motivazioni.

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)

12 Settembre 2013 – Tempo a disposizione: 45 min – Risultato: 32/32 punti

Esercizio 7 (8 punti)

Si introducano i concetti di *Open* e *Close World Assumption* con particolare riferimento alle logiche descrittive. A corredo si fornisca anche un breve esempio di come l'una e l'altra assunzione influiscano sui possibili risultati inferibili.

Esercizio 8 (12 punti)

Dato il programma proposizionale *Prolog*:

```
p:-q.
q:-p.
p:-r.
r.
```

la chiamata `?- p.` non termina, se non per *stack overflow*, poiché le prime due clausole del programma danno origine a in un ciclo infinito (*loop*). Tuttavia, `p`, `q` e `r` sono tutte e tre conseguenze logiche di questo programma. Prolog non è in grado di derivare `p` per la sua incompletezza dovuta alla ricerca *depth-first*. L'incompletezza sarebbe evitabile integrando nell'interprete un controllo sui cicli (*loop detection*).

Si supponga di avere solo *clausole ground* e si scriva un meta-interprete `clause_tree(Goal,ListaChiusi)` per un linguaggio Prolog solo proposizionale che sia in grado di fare *loop detection* (identificare i *loop*), controllando, per ogni goal atomico incontrato, se esso è già stato espanso, ovvero se esso appartiene alla `ListaChiusi` .

Se si incontra un goal atomico già appartenente alla lista dei nodi chiusi, si evita di espanderlo, e si fallisce. Il fallimento porta a considerare eventuali clausole alternative a quella applicata in precedenza. Se invece si deve risolvere un goal atomico non appartenente a tale lista, se esiste una clausola la cui testa unifica con il goal atomico, si inserisce il goal nella lista dei nodi chiusi, e si procede a risolvere il corpo della clausola applicata.

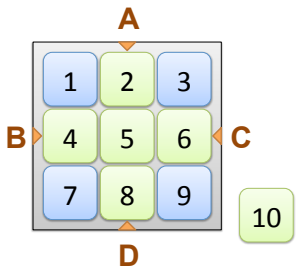
Con riferimento al precedente programma proposizionale, la chiamata al meta-interprete:

```
:- clause_tree(p,[]).
Yes
```

ha successo, poiché il meta-interprete, la seconda volta che si trova a risolvere il goal `p` lo trova già nella lista dei nodi chiusi, ed evita di espanderlo nuovamente, fallendo. Il fallimento porta a considerare la clausola `p :- r.` alternativa a quella applicata in precedenza, `p :- q.` , poi tramite il fatto `r.` si arriva al successo.

Esercizio 9 (12 punti)

Il *Labirinto Magico* è un gioco con un tabellone su cui sono posate delle tessere. Alcune sono fissate al tabellone a intervalli regolari e formano percorsi retti in cui le altre possono scorrere. Riempiti i percorsi, rimane una tessera con cui si può compiere a ogni turno una mossa. Le mosse modificano la configurazione del tabellone di gioco e consistono nello spingere la tessera libera in un percorso a scelta, liberando quella all'altra estremità del percorso.



Si consideri un tabellone di gioco semplificato 3x3 con 10 tessere numerate, come in figura. Le tessere blu `1`, `3`, `7` e `9` agli angoli del tabellone sono fisse, mentre quelle verdi `2`, `4`, `5`, `6` e `8` sono possono scorrere lungo i percorsi `A`, `B`, `C` e `D`. La tessera inizialmente libera per le mosse è la numero `10`.

Considerando come *stato iniziale* la configurazione in figura e come *stato obiettivo* una qualsiasi configurazione in cui *la tessera 10 si trovi al centro del tabellone*, si formalizzi il problema nei termini previsti dall'*algoritmo STRIPS*. Si esprima lo *stato iniziale*, lo *stato obiettivo* e le *operazioni* `PUSH_A()`, `PUSH_B()`, `PUSH_C()` e `PUSH_D()` (ognuna corredata dai propri *insieme dei requisiti* e *insieme degli effetti*) mediante i seguenti predicati:

- `TOP(x)` la tessera mobile x occupa la posizione in alto
- `BOTTOM(x)` la tessera mobile x occupa la posizione in basso
- `LEFT(x)` la tessera mobile x occupa la posizione a sinistra
- `RIGHT(x)` la tessera mobile x occupa la posizione a destra
- `CENTER(x)` la tessera mobile x occupa la posizione al centro
- `FREE(x)` la tessera x è libera

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
12 Settembre 2013 – Soluzioni

Esercizio 1

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

a. *Le zebre sono mammiferi di grandezza media con il manto a strisce*
   $\forall X: zebra(X) \Rightarrow [mammal(X) \wedge medium(X) \wedge striped(X)]$
   $\{\neg zebra(X) \vee mammal(X), \neg zebra(X) \vee medium(X), \neg zebra(X) \vee striped(X)\}$

b. *Il manto a strisce è non uniforme*
   $\forall X: striped(X) \Rightarrow uneven(X)$
   $\{\neg striped(X) \vee uneven(X)\}$

c. *I mammiferi sono animali a sangue caldo*
   $\forall X: mammal(X) \Rightarrow warmblood(X)$
   $\{\neg mammal(X) \vee warmblood(Y)\}$

d. *Nina è una zebra*
   $zebra(nina)$
   $\{zebra(nina)\}$

e. *Goal: Nina ha il manto non uniforme*
   $\neg uneven(nina)$
   $\{\neg uneven(nina)\}$

Teoria a clausole:

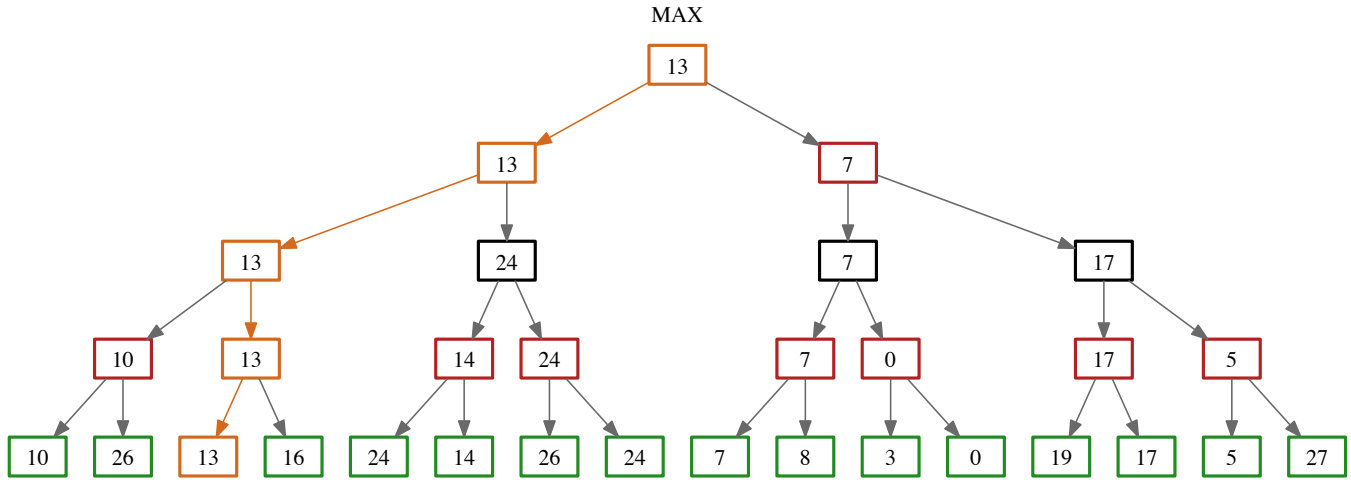
1. $\neg zebra(X) \vee mammal(X)$
2. $\neg zebra(X) \vee medium(X)$
3. $\neg zebra(X) \vee striped(X)$
4. $\neg striped(X) \vee uneven(X)$
5. $\neg mammal(X) \vee warmblood(Y)$
6. $zebra(nina)$
7. $\neg uneven(nina)$

Risoluzione:

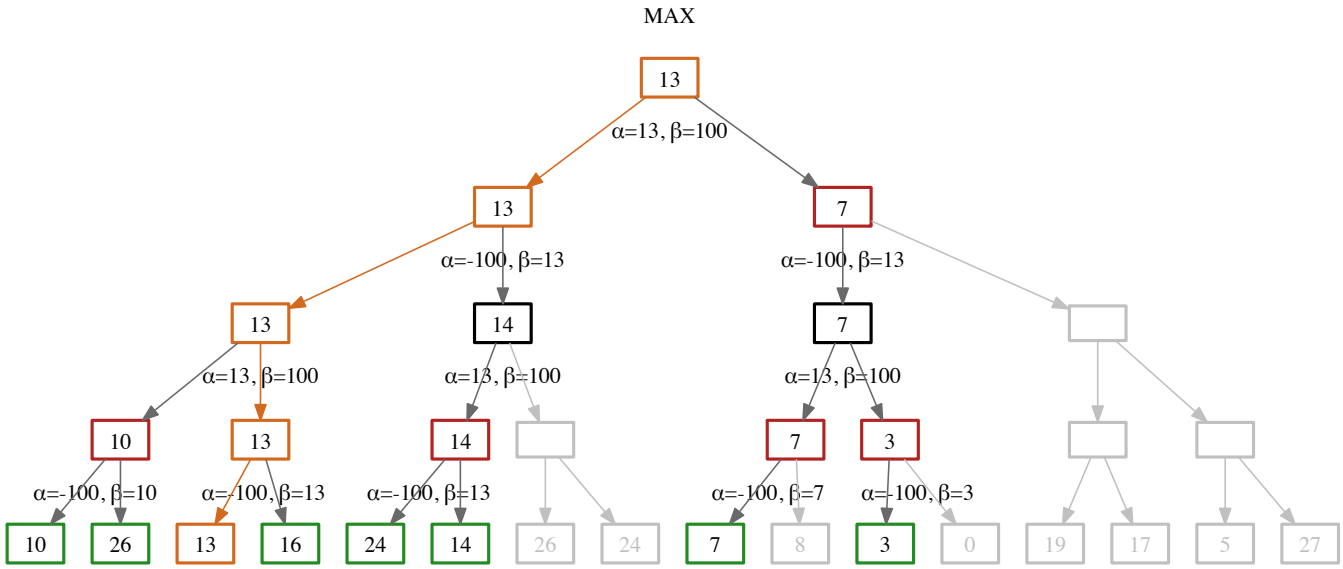
8. = 7. $\cup$ 4. $\neg striped(nina)$ $\{X/nina\}$
9. = 8. $\cup$ 3. $\neg zebra(nina)$ $\{X/nina\}$
10. = 9. $\cup$ 6. $\square$

Esercizio 2

min-max:



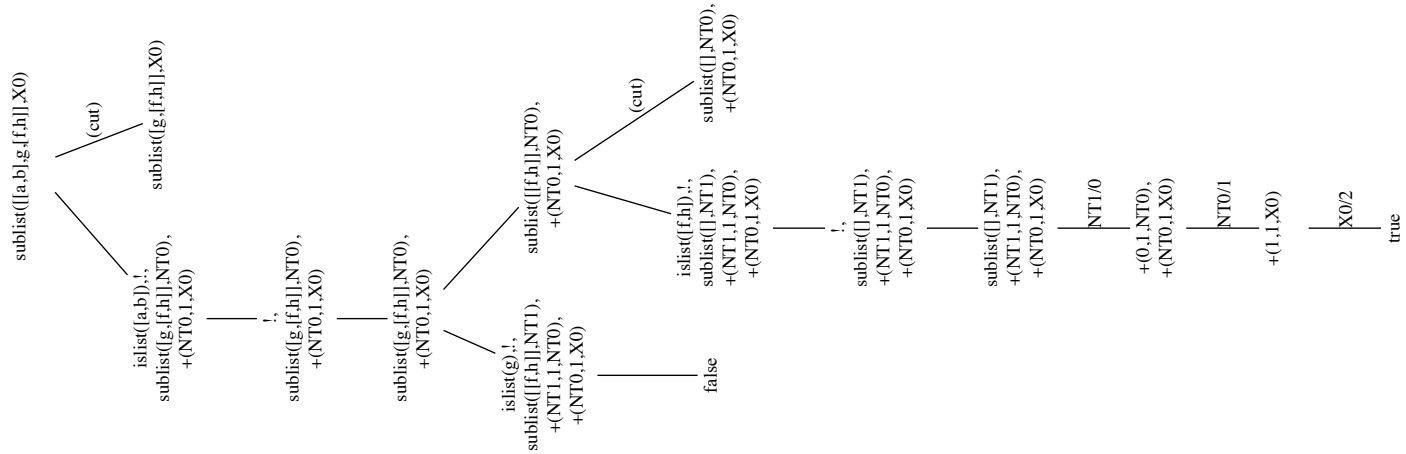
alfa-beta:



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

Esercizio 3

Albero SLD:

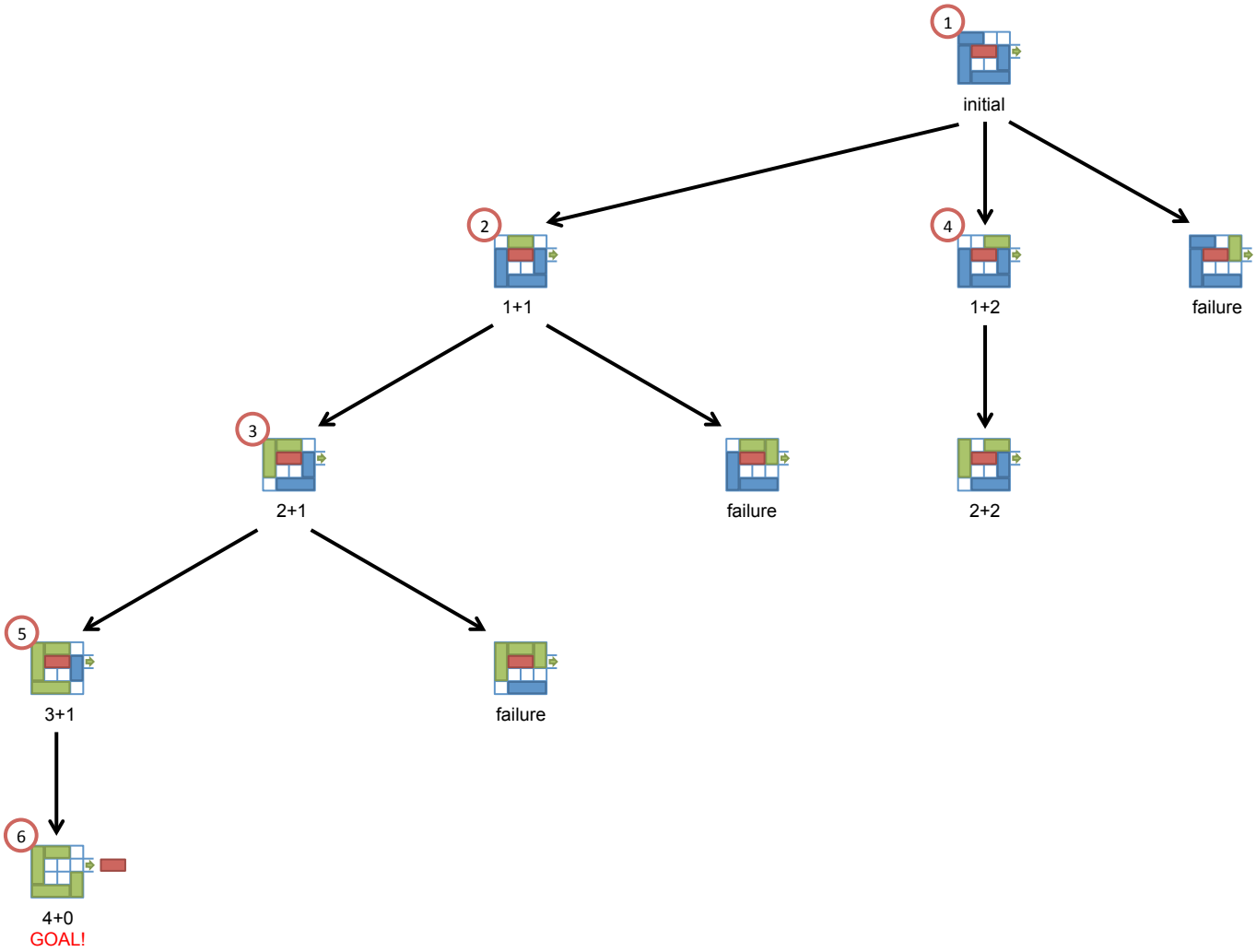


Esercizio 4

```
% listint(M,N,L)
% --------------
% Generates a list L containing the integer values between M and N (included).
listint(X,X,[X]) :- number(X), !.
listint(M,N,[M|L]) :- M<N, M1 is M+1, listint(M1,N,L).
```

Esercizio 5

Nel seguente albero, sono mostrati i rami generati; i numeri nei cerchi rappresentano l'ordine di esplorazione da parte dell'*algoritmo A**. I veicoli in verde sono quelli che sono già stati mossi e non possono essere mossi nuovamente.



Esercizio 6

Vedi slides del corso.

FONDAMENTI DI INTELLIGENZA ARTIFICIALE – 2° parte
12 Settembre 2013 – Soluzioni

Esercizio 7

Vedi slides del corso.

Esercizio 8

```
clause_tree(true,_) :- !.
clause_tree((G,R),Trail) :-
   !,
   clause_tree(G,Trail),
   clause_tree(R,Trail).
clause_tree(G,Trail) :-
   loop_detect(G,Trail),
   !,
   fail.
clause_tree(G,Trail) :-
   clause(G,Body),
   clause_tree(Body,[G|Trail]).
loop_detect(G,[G1,_]) :- G == G1.
loop_detect(G,[_,R])  :- loop_detect(G,R).
```

Esercizio 9

Stato iniziale:

```
FREE(10), TOP(2), BOTTOM(8), LEFT(4), RIGHT(6), CENTER(5)
```

Stato obiettivo:

```
CENTER(10)
```

Operazioni:

```
PUSH_A()
      Pre:   FREE(w),  TOP(x),  CENTER(y),  BOTTOM(z)
      Post: ¬FREE(w), ¬TOP(x), ¬CENTER(y), ¬BOTTOM(z),
             FREE(z),  TOP(w),  CENTER(x),  BOTTOM(y)

PUSH_B()
      Pre:   FREE(w),  LEFT(x),  CENTER(y),  RIGHT(z)
      Post: ¬FREE(w), ¬LEFT(x), ¬CENTER(y), ¬RIGHT(z),
             FREE(z),  LEFT(w),  CENTER(x),  RIGHT(y)
```

PUSH_C()

Pre: FREE(w), RIGHT(x), CENTER(y), LEFT(z)

Post: ¬FREE(w), ¬RIGHT(x), ¬CENTER(y), ¬LEFT(z),
FREE(z), RIGHT(w), CENTER(x), LEFT(y)

PUSH_D()

Pre: FREE(w), BOTTOM(x), CENTER(y), TOP(z)

Post: ¬FREE(w), ¬BOTTOM(x), ¬CENTER(y), ¬TOP(z),
FREE(z), BOTTOM(w), CENTER(x), TOP(y)