COMPITO DI INTELLIGENZA ARTIFICIALE (v.o.) – PARTE I
FONDAMENTI DI INTELLIGENZA ARTIFICIALE

13 Dicembre 2007 (Tempo a disposizione 2h; su 32 punti)

Esercizio 1 (punti 6)
Si rappresentino in logica dei predicati del I ordine le seguenti frasi:

- Tutti gli avvocati sono prolissi;
- Giorgio ama la montagna;
- Tutte le persone che amano la montagna sono prolisse.

Si rappresenti tale teoria in forma a clausole e si verifichi se tramite il principio di risoluzione con strategia *linear-input* è possibile derivare una o più delle seguenti frasi (le si rappresenti in logica e poi a clausole):

- Giorgio è un avvocato
- Tutte le persone prolisse sono avvocati
- Tutti gli avvocati amano la montagna
- Giorgio è prolisso.

Esercizio 2 (punti 8)
Si consideri il seguente programma Prolog:

```
depth(X,Y,0):- !.
depth([],[],N):-!.
depth([H|T],[A|L],N):-!, M is N-1,
    depth(H,A,M), depth(T,L,N).
depth(X,X,N).
```

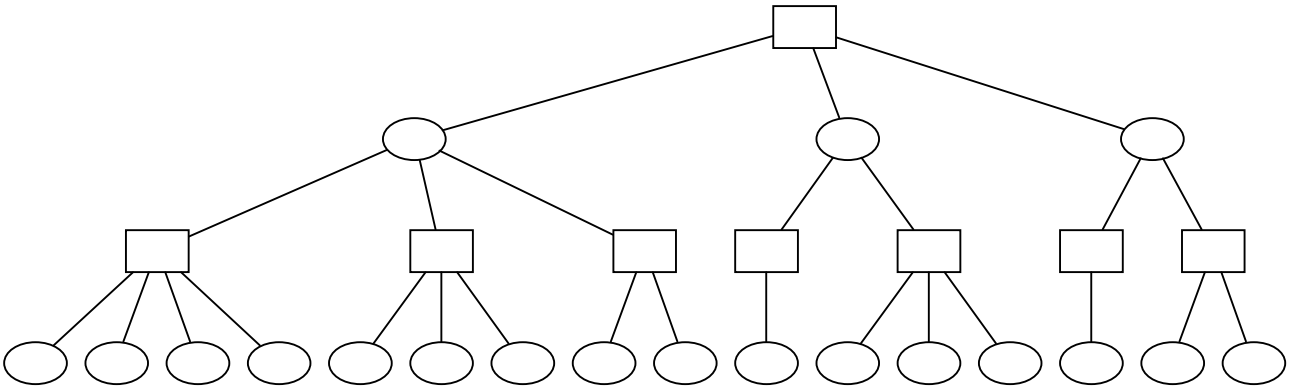
Si mostri l'albero SLD corrispondente al goal: `depth([1,[2]],L,2)`.

Esercizio 3 (punti 5)
Si scriva un programma Prolog che date due liste, `List1` composta a sua volta da liste di interi e `List2` composta da numeri interi, restituisca in uscita una lista `List3` che contiene tutti gli elementi di `List2` che appartengono ad almeno una delle liste `List1`.
Esempio:
`?- listint([[4,5],[3], [1,9]], [10, 4, 1, 7], L) yes L= [4,1].`

Esercizio 4 (punti 5)
Il proprietario di una grande azienda vinicola ha nella sua cantina 4 botti (*A, B, C, D*) di vino pregiato. Un giorno viene a sapere che solo una di esse, prima di essere riempita, era stata trattata con un potente veleno. Non sapendo di quale botte si tratti, si fa aiutare da un chimico che è però in grado di effettuare due sole analisi. Si procede come segue: si saggiano un campione costituito da vino prelevato dalle botti *A*+*B*, e un campione di vino prelevato dalle botti *B*+*C*. L'analisi del campione *A*+*B* indica "veleno", quella del campione *B*+*C* "non veleno".
Si modelli il problema di identificare la botte avvelenata come CSP nelle variabili *A, B, C* e *D* e lo si risolva applicando lo standard backtracking. Si considerino le variabili secondo l'ordinamento alfabetico, ed i valori di dominio (opportunamente scelto) in modo da considerare prima il valore corrispondente a "non veleno" e dopo quello corrispondente a "veleno".

Esercizio 5 (punti 5)

Si consideri il seguente albero di gioco, dove i punteggi sono tutti dal punto di vista del primo giocatore (Max):



Si mostri come l'algoritmo min-max risolve il problema e quale mossa viene scelta. Si mostrino poi i tagli alfa-beta.

Esercizio 6 (punti 3)

Si definiscano formalmente le proprietà di correttezza e completezza di un sistema logico.
Si discuta poi di come la programmazione logica e Prolog si possono considerare in base a tali proprietà (si ponga attenzione a non confondere la programmazione logica con il linguaggio Prolog).

SOLUZIONE

Esercizio 1

Logica dei predicati:

- *Tutti gli avvocati sono prolissi;*
- *Giorgio ama la montagna;*
- *Tutte le persone che amano la montagna sono prolisse.*

$\forall X$ avv(X) $\Rightarrow$ prolisso(X)
ama(giorgio,montagna)
$\forall X$ ama(X, montagna) $\Rightarrow$ prolisso(X)
Query:

- *Giorgio è un avvocato*
- *Tutte le persone prolisse sono avvocati*
- *Tutti gli avvocati amano la montagna*
- *Giorgio è prolisso.*

avv(giorgio)
$\forall X$ prolisso(X) $\Rightarrow$ avv(X)
$\forall X$ avv(X) $\Rightarrow$ ama(X,montagna)
prolisso(giorgio)

Clausole (definite):

C1) not avv(X) or prolisso(X)
C2) ama(giorgio,montagna)
C3) not ama(X, montagna) or prolisso(X)

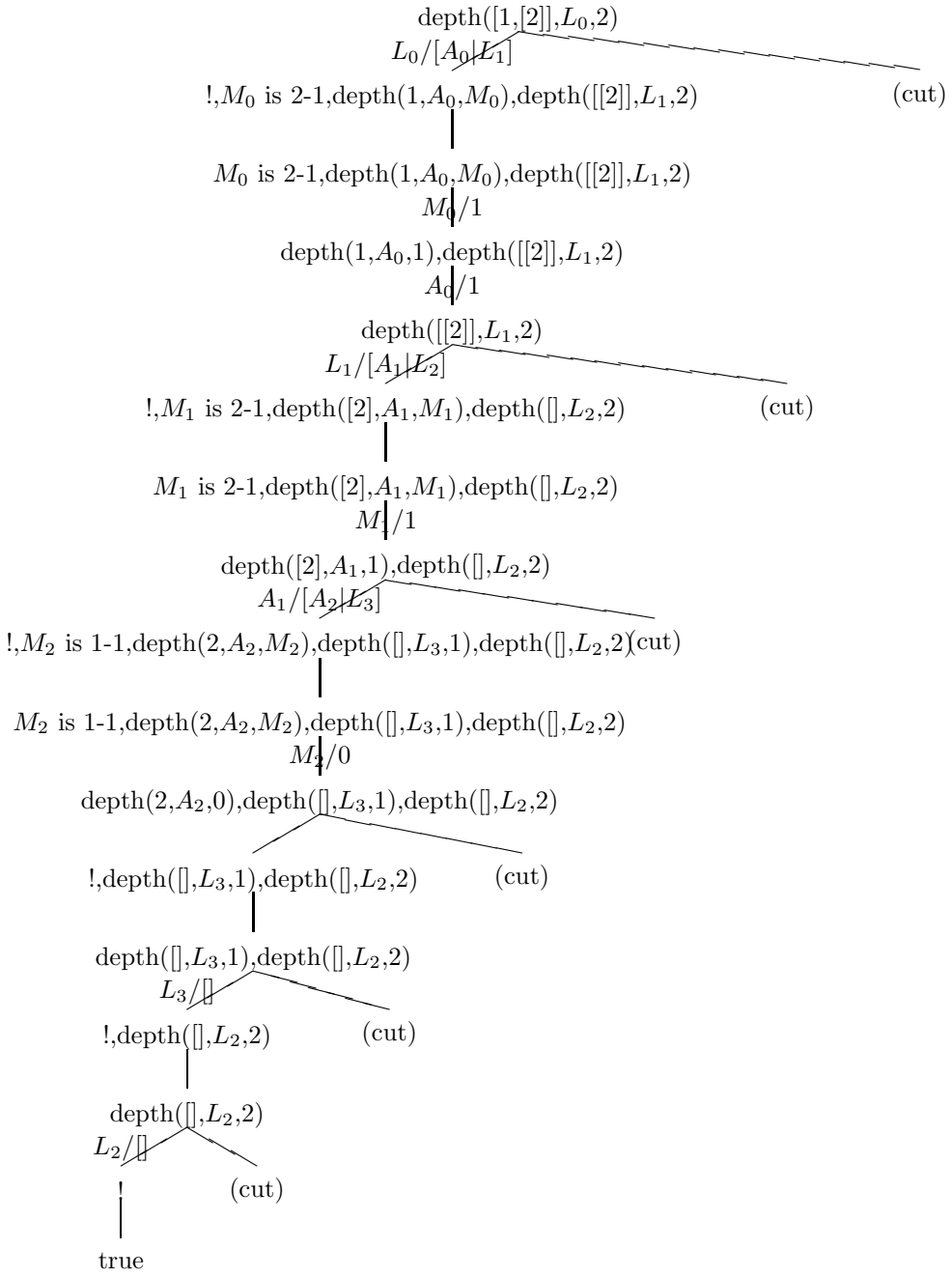
Clausole goal (Horn):

- G0: not avv(giorgio)
- G1: not ($\forall X$ prolisso(X) $\Rightarrow$ avv(X))
  $\exists X$ not (not prolisso(X) or avv(X))
  $\exists X$ prolisso(X) and not avv(X) Skolemizzazione:
  prolisso(c) and not avv(c)
  - G1') prolisso(c)
  - G1'') not avv(c)
- G2: not ($\forall X$ avv(X) $\Rightarrow$ ama(X,montagna))
  $\exists X$ not (not avv(X) or ama(X,montagna))
  $\exists X$ avv(X) and not ama(X,montagna) Skolemizzazione:
  avv(d) and not ama(d,montagna)
  - G3') avv(d)
  - G3'') not ama(d,montagna)
- G3: not prolisso(giorgio)

Risoluzione:

- da G0 nessun passo
- da G1' nessun passo
- da G1'' nessun passo
- da G2'+C1): prolisso(d) da qui nessun passo di risoluzione
- da G2'' nessun passo di risoluzione
- da G3+C3): not ama(giorgio, montagna da cui +C2) si deriva la clausola vuota

Esercizio 2



Esercizio 3:

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

listint(L,[],[]).
listint(L, [A|B], [A|C]):-memberL(A,L),!,listint(L,B,C).
listint(L, [A|B], L2):- listint(L, B, L2).
memberL(X,[L|R]):- member(X,L),!.
memberL(X,[Y|R]) :- memberL(X,R).
```

Esercizio 4

A,B,C,D::[0,1]
A+B+C+D=1
A+B=1
B+C=0

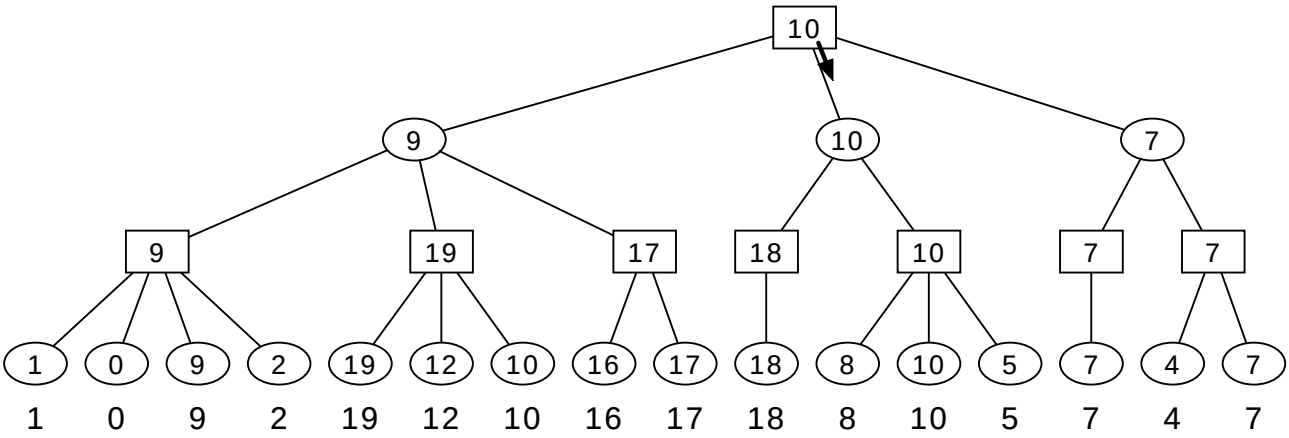
Standard backtracking:

A=0	B=0 fail		
“”	B=1	C=0 fail	
“”	“”	C=1 fail	
A=1	B=0	C=0	D=0 soluzione

(volendo si puo’ dare A+B=0 e B+C=0 che porta a D=1 ed ha una ricerca piu’ lunga, ma non aggiunge granche’)

Esercizio 5

Min-Max:



AlfaBeta: