Programming Paradigms Exercise (2)
Model Answers

Marco Montali Thomas Tschager

2nd Semester 2017/18

1. (a)
```
parent(john,sarah).
parent(john,jim).
parent(mary,sarah).
parent(mary,jim).
parent(sarah,betty).
parent(dave,betty).
parent(jim,jill).
parent(jim,susan).
parent(jim,bob).
parent(kate,jill).
parent(kate,susan).
parent(kate,bob).
female(sarah).
female(mary).
female(betty).
female(jill).
female(kate).
female(susan).
male(john).
male(jim).
male(dave).
male(bob).
```

   (b)
```
% assuming half sisters are sisters
sister(X,Y) :- female(X), parent(P,X), parent(P,Y), \+(X=Y).
```
   or
```
% assuming half sisters are not sisters
sister(X,Y) :-
   female(X),
   parent(Mother,X), parent(Mother,Y), female(Mother),
   parent(Father,X), parent(Father,Y), male(Father),
   \+(X=Y).
```

   (c) i. `?- sister(susan,bob).`

ii. `?- sister(X,susan).`

iii. `?- sister(susan,Y).`

Alternatively, one could first define the notions of father and mother, and then use these notions to in turn define the notion of (full) sister:

```
father(X,Y) :- male(X), parent(X,Y).
mother(X,Y) :- female(X), parent(X,Y).


% assuming half sisters are not sisters
sister(X,Y) :-
   female(X),
   mother(M,X), mother(M,Y),
   father(F,X), father(F,Y),
   \+(X=Y).
```

(d) The order of appearance is:

```
X = sarah,
Y = jim  ;
X = jill,
Y = susan ;
X = jill,
Y = bob ;
X = susan,
Y = jill ;
X = susan,
Y = bob ;
false.
```

The reason for this is that the subgoals of sister (we are looking at the latter definition here) are satisfied from left to right. Sarah is the first female in the list, so `X` gets instantiated with `sarah`. Then `parent(Z,sarah)` has to be satisfied. Again the clauses are scanned from top to bottom, first one that matches is `parent(john,sarah)` (which gets thrown out later, because `female(john)` cannot be satisfied. Compare to the lecture slides explaining backtracking.

(e) `grandfather(X,Y) :- male(X),parent(X,Z),parent(Z,Y).`

Alternatively, one could use the notion of father and mother, and argue that X is grandfather of Y either because he is father of the father of Y, or because he is father of the mother of Y. This is useful to illustrate how alternative definitions for the same predicate can easily be realized in Prolog:

```
grandfather(X,Y) :- father(X,Z), father(Z,Y).
grandfather(X,Y) :- father(X,Z), mother(Z,Y).
```

(f) `aunt(X,Y) :- female(X),sister(X,Z),parent(Z,Y).`

2. (a)
```
subdir(private, documents).
subdir(images, private).
subdir(videos, private).
subdir(work, documents).
subdir(research, work).
subdir(teaching, work).
```

   (b)
```
descendant(X,Y) :- subdir(X,Y).
descendant(X,Z) :- subdir(Y,Z), descendant(X,Y).
```

   Important: from the purely declarative point of view, the program above is equivalent to the program below:

```
descendant(X,Z) :- descendant(X,Y), subdir(Y,Z).
descendant(X,Y) :- subdir(X,Y).
```

   However, if we query the program below with Prolog, then we get that the Prolog interpreter does not terminate. This is because of the left-most depth-first strategy implemented in Prolog to expand the logical definitions. In the first program, Prolog always tries first the base case of recursion, and only enters into the recursive case if the base case fails. In the second program, instead, Prolog loops by continuously expanding the descendant predicate using the same predicate over and over again, and never reaches the alternative computation where, sooner or later, the base case actually applies.

3.
```
fib(1,1).
fib(2,1).
fib(N,F) :-
   N > 2, N2 is N-2, N1 is N-1,
   fib(N2,F2),
   fib(N1,F1),
   F is F2+F1.
```

   To compute the 10th Fibonacci number one can query:

```
?- fib(10,X).
```

4. (a)
```
follows(anne,fred).
follows(fred,julie).
follows(fred,susan).
follows(john,fred).
follows(julie,fred).
follows(susan,john).
follows(susan,julie).

tweets(anne,tweet1).
tweets(anne,tweet5).
tweets(fred,tweet2).
tweets(fred,tweet7).
tweets(fred,tweet8).
tweets(john,tweet3).
tweets(john,tweet4).
tweets(julie,tweet6).
tweets(susan,tweet9).
tweets(susan,tweet10).
```

(b) i.
```
fred_sees(X) :- follows(fred,Y),tweets(Y,X).
?- fred_sees(X).
```

ii.
```
friends(X,Y) :- follows(X,Y),follows(Y,X).
?- friends(X,Y).
```

iii.
```
sees(X,Z) :- follows(X,Y),tweets(Y,Z).
?- sees(X,Z).
```

iv.
```
fred_sees(X) :- follows(fred,Y),tweets(Y,X).
fred_sees(X) :- follows(fred,Y),follows(Y,Z),tweets(Z,X).
?- fred_sees(X).
```