Functional and Logic Programming Fall 2017

S07 : Prolog (Introduction)

Professor : Le Peutrec Stephane
Assistant : Lauper Jonathan

Exercise 1

(`=`)

The predicat `=/2`, is the unification operator. It unify the two terms given in argument. Note : the standard unification algorithm does not check occurences, use `unify_with_occurs_check/2` for that.

(`\=`)

Succeed if and only if both argument of `\=/2` can't be unified. For example, `a(B) \= a(C).` is false because we could found that $A = C$ or $C = A$.

(`==`)

Succeed if and only if both argument of == are the same atom or the same variable. For example, `X == Y` is false and `X == X` is true.

(`\==`)

Succeed if and only if both argument of `\==/2` are not the same atom or not the same variable. For example, `X \== Y` is true and `X \== X` is false.

(`=\=`)

Succeed if and only if both argument are instancied variable, number and both number are different. For example, `1 =\= 2` is true.

(`=:=`)

Succeed if and only if both argument are instancied variable, number and both number are the same. For example, `2 =:= 2` is true.

(`is`)

Unify the first argument with the second argument which must be an expression. For example, `1 is 3 - 2` is true and `A is 5 - 3` is true and A is unified with 2. Note that in the left expression, all variable has to be instancied, so we can't write something like `2 is 3 - X.`, this cause a failure.

```prolog
% Ex1.a
% unification of X and Y
predA(X,Y) :- X = Y.

% Ex1.b
% succeed if X and Y can't be unified
predB(X,Y) :- X \= Y.

% Ex1.c
% succed if X and Y are the same atom or exactly the same variable
predC(X,Y) :- X == Y.

% Ex1.d
% succeed if X and Y are not the same atom or not exactly the same variable
predD(X,Y) :- X \== Y.

% Ex1.e
% succeed if the evaluation of X and Y are number and if X is not equal to Y
predE(X,Y) :- X =\= Y.

% Ex1.f
% succeed if the evaluation of X and Y are number and if X is equal to Y
predF(X,Y) :- X =:= Y.

% Ex1.g
% succeed if X is a variable and if the evaluation of Y is a number
predG(X,Y) :- X is 3 + Y.
```

Exercise 2

child/2

```prolog
% Just to simplify launching...
:- initialization go.
```

father/2

```prolog
    findall(ancestor(X,Y),ancestor(X,Y),Bag),
    length(Bag,L),
    writef("%q, length : %q",[Bag,L]).
```

mother/2

```prolog
child(X,Y) :- parent(Y,X).

% father(?X,?Y), succeed if X is the father of Y
father(X,Y) :-
```

grandParent/2

```prolog
    male(X).

% mother(?X,?Y), succeed if X is the mother of Y
mother(X,Y) :-
```

grandFather/2

```
    female(X).

% grandParent(?X,?Y), succeed if X is the grandParent of Y
grandParent(X,Y) :-
```

uncle/2

```
    parent(T,Y).

% grandFather(?X,?Y), succeed if X is the grandFather of Y
grandFather(X,Y) :-
    father(X,T),
    parent(T,Y).
```

ancestor/2

```
1 % ancestor(?X,?Y), succeed if X is the ancestor of Y
2 ancestor(X,Y) :- parent(X,Y).
3 ancestor(X,Y) :-
4     parent(X,SomeBody),
5     ancestor(SomeBody,Y).
```

Note : inverting the order of the two predicates ancestor/2 does not affect its behaviour, because we call parent anyway and parent/2 always unify both variable. On the other hand, if we invert the $4th$ and the $5th$ line, we would have an infinite recursion and then a stack overflow, because ancestor would call itself with variables again and again.

Exercise 3

```
% hanoi(+Depth:int,+Start:atom,+Middle:atom,+End:atom)
% hanoi tower problem resolution
hanoi(0,_,_,_).
hanoi(Height,Start,Middle,End) :-
    NextHeight is Height - 1,
    hanoi(NextHeight,Start,Middle,End),
    writef('dep from %q to %q\n',[Start,End]),
    hanoi(NextHeight,Middle,End,Start).
```