COMP90048 Declarative Programming
Semester 1, 2018
Peter J. Stuckey

Declarative Programming

Answers to workshop exercises set 09.

QUESTION 1

Write a Prolog predicate same_elements(L1, L2) that holds when all elements of list L1 are in L2 and vice versa, though they may be in different orders and have different numbers of occurrences of the elements. This need only work in modes where both arguments are ground.

ANSWER

```
same_elements(L1, L2) :-
    all_in(L1, L2),
    all_in(L2, L1).

all_in([], _).
all_in([E|Es], L) :-
    member(E, L),
    all_in(Es, L).
```

QUESTION 2

Rewrite your same_elements predicate to work in n log(n) time, where n is the length of the longer list.

ANSWER

```
same_elements1(L1, L2) :-
    sort(L1, Sorted),
    sort(L2, Sorted).
```

QUESTION 3

Write a predicate times(W,X,Y,Z) that holds when all arguments are integers
and W*X + Y = Z and 0 <= Y < |W| (where |W| denotes the absolute value

of
W). This implies that W \= 0, and that |W| <= |Z|. This predicate should work when W and at least one of X and Z are bound to integers, and should
be deterministic as long as at least three of W, X, Y and Z are bound.

This predicate is similar to the built-in predicate plus/2, which can do addition and subtraction, depending on which arguments are bound when you call it. But times/4 handles multiplication and addition or division and modulus, depending on how you call it.

Hint: use the predicate integer/1 to check if an argument is bound to an integer.

Hint: the builtin between(X,Y,Z) holds when X <= Z <= Y, and work when X and Y are bound to integers. If Z is unbound, it will nondeterministically generate Z between X and Y inclusive.

Hint: use the function abs/1 (on the right side of is) to compute absolute value.

Hint: the expressions X div Y rounds down while X // Y rounds toward zero.
Also, the expression X mod Y produces a negative result when Y is negative, while X rem Y produces a negative result when X is negative. So you want to use div together with mod and // together with rem.

Hint: the goal
    throw(error(instantiation_error, context(times/4,_)))
will throw an exception reporting that a call to times/4 had insufficiently instantiated arguments. Similarly, the goal
    throw(error(type_error(integer X), context(times/4,_)))
will throw an exception reporting that X was expected to be an integer and was not in a call to times/4.

ANSWER

```
times(W,X,Y,Z) :-
        ( nonvar(Y), \+ integer(Y) ->
            throw(error(type_error(integer, Y), context(times/4,_)))
        ; integer(W) ->
            Ybound is abs(W) - 1,
            Ybound >= 0,
            ( integer(X) ->
```

```
                WX is W * X,
                ( integer(Z) ->
                    Y is Z - WX,
                    between(0, Ybound, Y)
                ; nonvar(Z) ->
                    throw(error(type_error(integer,                 Z),
context(times/4,_)))
                ;   between(0, Ybound, Y),
                    Z is WX + Y
                )
            ; nonvar(X) ->
                throw(error(type_error(integer,                     X),
context(times/4,_)))
            ; integer(Z) ->
                ( Z < 0 ->
                    X is (Z - Ybound) // W,
                    Y is Z - W * X
                ;   X is Z // W,
                    Y is Z rem W
                )
            ; nonvar(Z) ->
                throw(error(type_error(integer,                     Z),
context(times/4,_)))
            ;   throw(error(instantiation_error, context(times/4,_)))
            )
        ; nonvar(W) ->
            throw(error(type_error(integer, W), context(times/4,_)))
        ;   throw(error(instantiation_error, context(times/4,_)))
        ).
```

QUESTION 4

Write a program to solve the water containers problem.

You have two containers, one able to hold 5 litres and the other able to hold 3 litres. Your goal is to get exactly 4 litres of water into the 5 litre container.

You have a well with an unlimited supply of water.
For each ``turn,'' you are permitted to do one of the following:

completely empty one container, putting the water back in the well

completely fill one container from the well

pour water from one container to the other just until the source container is empty or the receiving container is full.

In the last case, the original container is left with what it originally had less whatever unfilled space the receiving container originally had.

All containers begin empty.

Write a Prolog predicate containers(Moves) such that Moves is a list of actions to take in order to obtain the desired state. Each action on the list is of one of the following forms:

fill(To), where To is the capacity of the container to fill from the well

empty(From), where From is the capacity of the container to empty

pour(From,To) where From is capacity of the container to pour from andTo is the capacity of the container to pour into

Hint: write a predicate that computes the effect of each of the possible moves. Then write a predicate that nondeterministically explores all the possible sequences of moves, computing their effect.

Hint: you will need to stop Prolog from repeatedly returning to the same state, otherwise Prolog will search longer and longer sequences of moves just pouring water back and forth between the two containers. You can prevent this by keeping the list of states seen so far in the search, and reject any more that returns to a state you have seen before.

ANSWER

```
containers(Steps) :-
        containers(3, 5, 0, 0, _, 4, [0-0], Steps).

containers(_, _, V1, V2, V1, V2, _, []).
containers(C1, C2, V1, V2, T1, T2, Hist, [Move|Steps]) :-
        move(C1, C2, V1, V2, N1, N2, Move),
        State = N1-N2,
        \+ member(State, Hist),
        containers(C1, C2, N1, N2, T1, T2, [State|Hist], Steps).
```

```
move(C1, _, _, V2, 0, V2, empty(C1)).
move(_, C2, V1, _, V1, 0, empty(C2)).
move(C1, _, _, V2, C1, V2, fill(C1)).
move(_, C2, V1, _, V1, C2, fill(C2)).
move(C1, C2, V1, V2, N1, N2, pour(C1,C2)) :-
        pour(C2, V1, V2, N1, N2).
move(C1, C2, V1, V2, N1, N2, pour(C2,C1)) :-
        pour(C1, V2, V1, N2, N1).

pour(C2, V1, V2, N1, N2) :-
        (   V1 + V2 > C2 ->
                N1 is V1 - (C2 - V2),
                N2 is C2
        ;   N1 = 0,
            N2 is V1 + V2
        ).
```