

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.

QUESTION 2

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

QUESTION 3

Write a predicate `times(W,X,Y,Z)` that holds when $W \cdot X + Y = Z$ and either $0 \leq Y < W$ or $W < Y \leq 0$. This should work when any 3 arguments are bound to integers, or when Z and one of W and X are bound to integers. In these modes, this predicate should be deterministic.

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

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.

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

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.