Formal Modeling (SS 2024)
Extra Assignment 1 (September 30, 2024)

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
Wolfgang.Schreiner@risc.jku.at

The result is to be submitted by the deadline stated above ia the Moodle interface of the course as a single archive file in `.zip` or `.tgz` format which contains the following files:

1. A single PDF file with the following contents:
   - a cover page identifying the course, the assignment, and the submitter;
   - a section for each part of the assignment, which contains
   - the deliverables requested for this section with a snapshot of the listing of the corresponding RISCAL file that contains all additions/changes to the skeleton file handed out (nicely formatted and typeset in a fixed-width font of readable size with no line overflows), and
   - optionally any explanations or comments you would like to make.
2. All RISCAL files developed in the assignment.

All assignments only ask to complete the definitions of predicates by formulas in first order logic (operations $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists$). You may also use if-then-else and let-in expressions to make the formulas more readable.

Hint: you may annotate arbitrary formulas and terms by the `print` expression (see the RISCAL manual section B.5.15) to understand the derived results. For instance:

```
// result is p(e), prints first e and then p(e) in separate lines
print p(print e)

// result is f(x,y), prints x and y in one line, then f(x,y) in another
print "x:{1}, y:{2}", x, y in print f(x,y)
```

Assignment 1A (40P): Word Comparison

We consider the problem of the lexicographic comparison of words[1]. The attached RISCAL file `Lexicographic.txt` gives an algorithm `compare` (accompanied by auxiliary definitions) to decide whether word a is equal to word b, comes before b, or comes after b in the lexicographic order.

1. Complete the definition of the predicate `compares` that defines the postcondition of the algorithm.
2. Define in the pop-up window "Other Values" suitable values for the model parameters N and C (e.g., $N = 5$, $C = 4$). Press in the "Operation" panel the button "Show/Hide Tasks" to open the "Tasks" menu.
3. Validate the specification of the procedure by running (with option "Nondeterminism" switched *on* and option "Silent" switched *off*) the task "Execute specification". Analyze the printed output to investigate which input/output pairs are allowed by your definition. Are those (and only those) pairs printed that you expect?
4. Further validate your specification by checking (with the option "Apply SMT Solver to all Theorems" in the corresponding popup menu) the various conditions listed under "Validate Specification". Are the results as you have expected? If a condition is false, use the option "Show Counterexample" in the corresponding popup menu to derive counterexample values for the formula (whose definition is given by the option "Print Definition"). Explain the results.
5. Run task "Execute Operation" to check whether the procedure indeed satisfies your specification (you can assume that the procedure is correct; any errors indicate deficiencies in the specification).

Demonstrate by (a reasonable selection of) the RISCAL output that you indeed have performed above tasks. Interpret the results and judge whether your specifications are adequate.

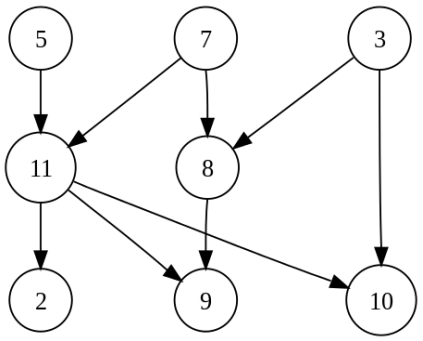
[1] https://en.wikipedia.org/wiki/Lexicographical_order

You only have to perform *one* of the alternatives of assignment 1B.

Assignment 1B (60P, Alternative 1): Topological Sorting

We consider the problem of "topological sorting"[2] a directed graph whose nodes are numbered $0 \ldots N$. The RISCAL file `topsort.txt` contains an implementation of a simple variant of Kahn's algorithm as the following procedure:

```
proc topologicalSort(G:Graph):VertexSeq
  requires acyclic(G);
  ensures covers(result,|G.V|,G.V);
  ensures sorted(result,|G.V|,G.E);
{
  var seq:VertexSeq = Array[N+1,Vertex](0);
  var i:Number := 0;
  var V0:Set[Vertex] = G.V;
  var E0:Set[Edge] = G.E;
  while V0 ≠ ∅[Vertex] do
    invariant ... ; decreases ... ;
  {
    choose v ∈ V0 with ¬∃e ∈ E0. (e.to = v);
    seq[i] := v;
    V0 := V0\{ v };
    E0 := E0\{ e | e ∈ E0 with e.from = v };
    i := i+1;
  }
  return seq;
}
```



A topological sorting: $[3, 7, 8, 5, 11, 10, 2, 9]$

While the algorithm is of its own interest, its details are actually not relevant for this assignment. The main point is that above procedure is annotated with the problem's precondition and postcondition based on predicates `acyclic(`G`)` (graph G is acyclic), `covers(`s`,`n`,`V`)` (all elements of V occur among the first n vertices of path s) and `sorted(`s`,`n`,`E`)` (the first n vertices of s are topologically sorted with respect to edge set E). Your task is as follows:

1. Complete the definition of these predicates. The predicate `acyclic(`G`)` should be based on a predicate `path(`s`,`n`,`G`)` that states that s is a path of length n in graph G.

2. Validate the adequacy of your definition for small values of the model parameters (e.g. $N = 2$ or $N = 3$) in the same way as for assignment 1A (execute the specification itself, check the various validation conditions, and execute the procedure with your pre- and postcondition.

3. Additionally check by application of the menu option "Apply SMT Solver to All Theorems" the other conditions displayed in the task menu; they represent the verification conditions generated from the procedure specification and the loop annotations (all these conditions must be valid for a correct definition of the predicates).

Again, demonstrate by (a reasonable selection of) the RISCAL output that you indeed have performed above tasks. Interpret the results and judge whether your conditions are adequate.

[2] https://en.wikipedia.org/wiki/Topological_sorting

Assignment 1B (60P, Alternative 2): Elevator Control

Consider a building with with floors numbered $0 \ldots F$ that are served by elevators numbered $0 \ldots E$. On each floor, there are two call buttons "up" and "down" that the user may press to announce the intention of going up or down. Inside each elevator, there is a button for each floor that the user may press to request to go to that floor (this request needs not match the intention originally announced by the call). If a button is pressed, its light turns on, until the elevator controller switches it off. For serving floors, elevators apply the "elevator algorithm"[3] which is also utilized in disk scheduling[4]: the elevator continues to travel in its current direction while there are remaining requests in that same direction; if there are no further requests in that direction, then it stops and becomes idle, or changes direction if there are requests in the opposite direction.

The RISCAL file `Elevator.txt` contains the skeleton for the controller of an elevator system whose state consists of the following components:

- $call[f]$ denotes the set of call buttons (*up*/*down*) on floor f whose lights are "on".
- $button[e]$ denotes the set of floor buttons in elevator e whose lights are "on".
- $direction[e]$ denotes the direction d in which elevator e is moving ($d = none$ if e is stopped).
- $motor[e]$ indicates whether the motor of elevator e is switched on (true) or off (false).
- $door[e]$ indicates whether the door of elevator e is open (true) or not (false).
- $floor[e]$ denotes the floor in which elevator e currently is.

The system has various actions; your task is to formulate the guard conditions of these actions (indicated by the placeholder `true`) as logical formulas such that the system behaves in a "reasonable" way. As a minmum requirement, the system must not violate the safety property (stated as a corresponding invariant) that no elevator has its motor on when its door is open.

However, to really validate that the elevator behaves as "expected", uncomment the other invariant that causes the printing of some/every system run that leads from the initial state where all elevators are at the bottom floor and have their doors closed to a state where all elevators are at the top floor and have their doors open. Investigate these runs manually and check whether they correspond to your intuitive understanding of how the elevators "should" behave. For example, if an elevator is moving upwards and passes a floor where the call button to go upwards is lit, the elevator should turn off the motor, open the door, close the door, turn on the motor, and continue on its way. This stop should switch off the light of the floor button that caused the stop; if in the elevator the light of the button for the corresponding floor was switched on, it should be switched off, too.

To investigate such runs, use small model values (e.g., $E = 0, 1$ and $F = 2$) and use the execution parameter "Depth" to limit the lengths of the investigated runs (depths less than 50 should suffice).

Since there are many variations/interpretations of the requirements of an elevator controller, there is no really "correct" or "wrong" solution to the problem; it is the overall adequacy of the presented model as illustrated by the printed system runs that will be judged. Please give at least two significantly different runs and give informal explanations/justifications of their adequacy.

[3] https://www.popularmechanics.com/technology/infrastructure/a20986/the-hidden-science-of-elevators/
[4] https://en.wikipedia.org/wiki/Elevator_algorithm