Formal Modeling (SS 2019)
Assignment 2 (June 5)

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 2a (25 Points): Word Comparison

We consider the problem of the lexicographic comparison of words[1]. The attached RISCAL file `Lexicographic.txt` gives an algorithm `compare` (together with accompanying 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., $N = 4$ and $C = 2$). Press in the "Operation" panel the button "Show/Hide Tasks" to open the "Tasks" menu.
3. Validate your specification 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 running (with option "Nondeterminism" switched *off* and option "Silent" switched *on*) the tasks "Is precondition satisfiable?", "Is precondition not trivial?", "Is postcondition always satisfiable?", "Is postcondition always not trivial?", "Is postcondition sometimes not trivial?", "Is result uniquely determined?". Are the results as you have expected?
5. Run task "Execute Operation" to check whether the algorithm indeed satisfies your specification (respectively, whether your specification matches the algorithm; the algorithm most likely is correct).

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

Assignment 2b (35 Points): Ultimate Tic-Tac-Toe

We consider the game "Ultimate-Tic-Tac Toe"[2]. The RISCAL file `UltimateTicTacToe.txt` contains a procedure `play` (together with accompanying auxiliary definitions) that plays (depending on the execution option "Non-determinism") some/all possible instances of this game.

1. Complete the definition of the predicate `wins` that determines whether a particular player has won the game.
2. Complete the definition of the predicate `legal` that determines whether a particular move (a choice of a local board and a position in that board) is legal.

[1] https://en.wikipedia.org/wiki/Lexicographical_order
[2] https://en.wikipedia.org/wiki/Ultimate_tic-tac-toe

Validate your specification by running (with option "Nondeterminism" switched *on* and option "Silent" switched *on*) all possible games; the procedure prints out all games which was won by some players (the games resulting in a draw are not printed).

The procedure may have to execute some 10^5 non-deterministic execution branches (which may take a minute or so) to find a game that was won by some player. Once some such games have been found, you may interrupt the execution (press the "Stop Execution" button) and investigate some game(s) to determine whether it was indeed correctly played and the winner was correctly determined.

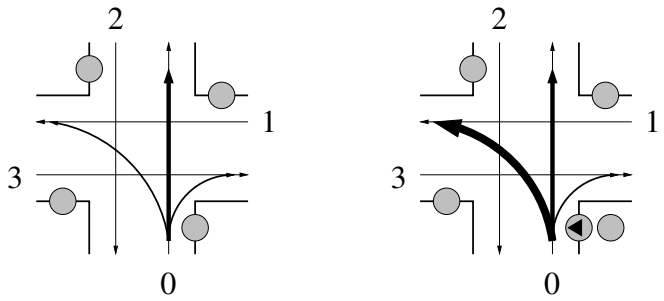
Explain in your submission one played game in detail and why it was correctly played.

Assignment 2c (40 Points): Traffic Light Controller

Consider a traffic crossing which consists of P paths and L traffic lights which control the paths by showing colors red, green, and amber. In detail:

- When it is green, a traffic light allows cars to proceed along some (generally multiple) paths.
- Each path may conflict with some (generally multiple) paths (thus the drivers of cars running on conflicting paths must generally take care not to collide with each other by obeying the usual priority rules).
- However, a traffic light may mark some of its paths as *priority*; cars running on such paths must not encounter cars on conflicting priority paths (while cars on such paths may still encounter cars from conflicting paths, they always have priority over these).
- Moreover, a traffic light may grant (by direction arrows) access to all of its paths *exclusively* (cars running on such paths must not encounter cars on any conflicting paths).

For instance, the figures below shows two typical crossings with $P = 4$ paths where each path conflicts with two other paths (the ones running perpendicular to it). The left crossing has $L = 4$ traffic lights without arrows where each light gives access to three paths (straight ahead, left, right). In each direction the straight ahead path is the only priority path; since this path crosses two perpendicular priority paths, when its light is green, the lights on the conflicting paths must be red. However, the light on the path from the opposite direction (whose priority path is not in conflict) may be green; cars that come from this direction and turn left have lower priority when crossing the path.



The right figure depicts a similar crossing with $L = 5$ lights where one of the entry points has an additional traffic light with a left arrow; if this light is switched on, cars may turn left without being hindered by the traffic from the opposite direction (whose light must be therefore red).

Each traffic light runs ("US/Japanese style") in the cycle red→green→amber→ red. The goal is to model a traffic light controller that prevents collisions by implementing the protocol described above. Furthermore, to give drivers some time to recognize changes in the traffic situation, the controller must ensure the following secondary safety properties:

- A traffic light may switch from amber to red only after it has been amber for at least *TA* time units.
- A traffic light may switch from red to green only after it has been red for at least *TR* time units. Furthermore, all traffic lights that must be red on conflicting paths (according to above protocol), must have also been red for *TR* units before the switch.

There is no minimal time for switching from green to amber, because emergency cars may override the traffic control at any time to shorten green phases.

The RISCAL file `TrafficLight.txt` contains the skeleton for the simulation of such a traffic controller. Here the state of each traffic light is modeled by its current color and by the information how long the duration is since the traffic light has been changed for the last time (thus only relative, not absolute, times are stored to minimize the state space). Complete this skeleton by giving a definition for the predicate `valid` that describes whether the traffic light controller is allowed to perform a particular action; such an action consists of a choice of a traffic light to be changed and the length of the time period since the last action (i.e., the predicate constrains *what* light may be changed *when*).

Please note that a crossing is generally described by a minimal set of conflicting pairs of paths: if $\langle p_1, p_2 \rangle$ is in this set, then $\langle p_2, p_1 \rangle$ is not; this it becomes handy to define above predicates with the help of an auxiliary predicate *conflicts* that denotes the symmetric (and reflexive?) closure of the given conflict relation.

Validate your model by checking (with option "Nondeterminism" switched *on* and option "Silent" switched *off*) the function `control1()` that computes for above left crossing (set $P = L = 4$) all possible action sequences of length N such that no time variable overflows the maximum time span T units (if an action sequence would lead to the overflow of some time variable, this sequence will not be returned). Choose appropriate values for N and T and demonstrate how some of the computed action sequences satisfies the requirements stated above.

Repeat the validation by defining and executing a similar function `control2()` that adds to the crossing the fifth light shown above. Demonstrate the validity of some execution that switches this additional light at least once to green.