Verifying cyberphysical systems	Fall 2019
Sayan Mitra	*Extra problems for Chapter 2: Modeling computation*
	Last updated September 2, 2019

Problem. Bidirectional array. Model the following token-based mutual exclusion algorithm that works on a *bidirectional array*. There are N processes $\{0, \ldots, N-1\}$ in an array. Each process i, has a single variable $s[i]$ that takes values in the set $\{0, 1, 2, 3\}$ independent of the size of the array. The two processes 0 and $N-1$ behave differently from the rest, they can take two values each $s[0]$ can take values $\{1, 3\}$ and $s[N-1]$ can take values $\{0, 2\}$. Let $Nbrs(i)$ be the set of neighboring processes for process i.

1. Program for processes, i, $i = 0$ or $i = N-1$:
   if $\exists\, j \in Nbrs(i)$:$s[j] = s[i] + 1 \mod 4$ then $s[i] = s[i] + 2 \mod 4$
2. Program for processes, i, $0 < i, N-1$:
   if $\exists\, j \in Nbrs(i)$:$s[j] = s[i] + 1 \mod 4$ then $s[i] = s[i] + 1 \mod 4$

In this protocol, process i *has token* iff

$$\exists j \in Nbrs(i) : s[j] = s[i] + 1 \mod 4.$$

(a) Write the model of the bidirectional array system with N processes using the language we saw in class.

(b) Write an execution of the algorithm that starts from a state with a single token. Mark the process with the token.

(c) Write an execution (of length at least 6) that starts from a state with multiple tokens.

(d) Prove the invariant "system has a single token" using the inductive invariance theorem.

Problem. LCR Leader election. In this problem, you will create a model of a leader election algorithm in a unidirectional ring [2]. Here is the informal description of the protocol:

Each process sends its identifier to its successor around the ring. When a process receives an incoming identifier, it compares that identifier to its own. If the incoming identifier is greater than its own, it keeps passing the identifier; if it is less than its own, it discards the incoming identifier; if it is equal to its own the process declares itself as the leader.

(a) Write the model of the system with n processes in the ring using the language we saw in class. To get you started, the set of variables is:

- $send$: The identifier to send or *null*,
- $status$: Takes values in $\{unknown, leader\}$ to indicate that the leader has been elected or not.

(b) Write an execution of the system in which status of at least one process is eventually set to *leader*.

(c) Write two candidate invariants.

Problem 3. Impossibility of election. A *symmetric function* $f : S^k \to S$ has the property that for all $s_1, s_2 \in S^k$, if s_1 is a permutation of s_2 then $f(s_1) = f(s_2)$. That is, for $k = 3$, $f(\langle 1,2,3\rangle) = f(\langle 2,3,1\rangle) = f(\langle 3,1,2\rangle)$, etc.

Consider a synchronous algorithm SymGk running on a graph G with *indegree* k. That is, each process P_j reads the states of exactly k other processes. In this algorithm. SymGk, every node updates its state x_i according to some symmetric function $f : S^k \to S$, that is,

$$x[i] := f(\langle x[j] \mid (j,i) \in G)$$

Show that it is impossible for SymGk to elect a leader if initially every process has the same value of $x[i]$. *[Hint: State an invariant and prove it by induction on rounds.]*

Coding problem. Invariance with Z3. Use the Z3 SMT solver to encode and check that for the Dijkstra's token ring mutual exclusion algorithm (DijkstraTR of Chapter 2) the predicate ϕ_{legal} is an inductive invariant.

First install z3 in your computer. This is a very quick process. On the MacOS `pip install z3-solver` does it in less than a minute. For Windows and Linux you can get it from: https://github.com/Z3Prover/z3. There is a lot of help available online for installation related issues.

Next, download the file `DijkstraTRind.py` given for this homework and read it carefully. There are several function given and you have to complete several other functions. The documentation given with the program, the lecture slides, and the discussions in the book chapter 2 and 7 should be adequate for you to complete this problem. Here are some additional notes.

- `has_token(x_list, j)`: Generates a Z3 Boolean expression (predicate) that represents whether process P_j holds the token. Here `x_list` is a list of Z3 variables, for example, `[x[0], x[1], x[2], x[3]]`, and `j` is the index of process P_j.
- `legal_config(x_list)`: Generates a Z3 Boolean expression that represents whether the system is in a legal configuration. This is the implementation of ϕ_{legal}. This function is given to and **you do not have to change it**.
- `transition_relation(old_x_list, new_x_list)`: Generates a Z3 Boolean expression representing transition from `old_x_list` to `new_x_list`.
- `prove(conjecture)`: Checks whether the Boolean predicate `conjecture` is valid using the Z3 solver to check the *unsatisfiability* of the negation of `conjecture`. **You do not have to change this.**

Run your program with `python DijkstraTRind.py`. If the functions are written correctly, then validity of the base case and the induction case imply (by the induction invariance Theorem) that `legal_config` is indeed an invariant.

Problem. Traffic light. Model a traffic light which follows these rules. It can display three colors: green, amber, and red. It cycles through these colors in this order, pausing some number of ticks in each color. There is a button that pedestrians can press to cross the road. If the color is green when the button is pressed then within 5 ticks it becomes amber. Otherwise, pressing the button does not change the behavior of the light.

Coding problem. Invariant verification. Verify inductive invariants of other distributed algorithms, such as the bidirectional array in Problem 1, following the pattern of Problem 4. Several interesting examples appear in Chapter 17 of [1].

References

[1] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*. Chapman & Hall/CRC, 2nd edition, 2014.

[2] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.