Cyber-Physical Systems (CSC.T431), 2020 Solutions for Assignment (1)

1. The extended-state machine in Figure 1 implements the desired component. The initial mode is 0. When the input x is 1, the component switches to the mode 1, and subsequently when the input y is 1, it switches to the mode 2. Symmetrically, in the initial mode, when the input y is 1, the component switches to the mode 3, and subsequently when the input x is 1, it switches to the mode 2. Note that in the initial mode, if both input variables x and y equal to 1, the component directly switches to the mode 2. The transitions to the mode 2 set the output z to 1, and all other transitions set the output z to 0. In mode 2, when the condition ($reset = 1$) holds, the component returns to the initial mode.

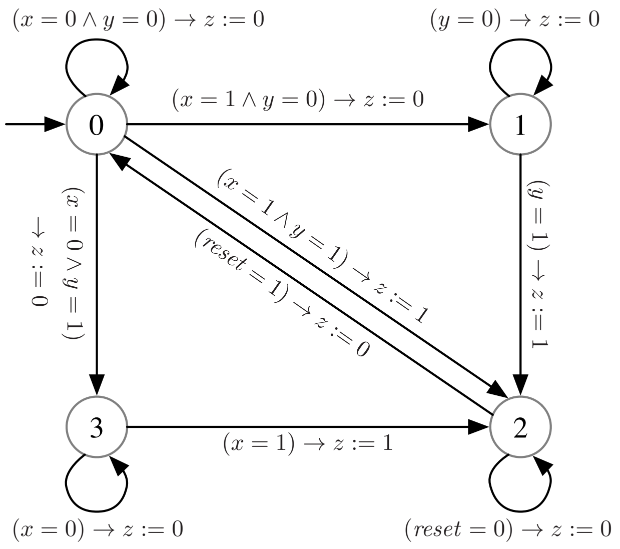
$(x = 0 \wedge y = 0) \to z := 0$
$(y = 0) \to z := 0$
$(x = 1 \wedge y = 0) \to z := 0$
0
1
$(x = 0 \wedge y = 1) \to z := 0$
$(x = 1 \wedge y = 1) \to z := 1$
$(reset = 1) \to z := 0$
$(y = 1) \to z := 1$
3
2
$(x = 1) \to z := 1$
$(x = 0) \to z := 0$
$(reset = 0) \to z := 0$

Figure 1: Extended-State Machine for Question 1

2. The reaction of the component are listed below (the output lists the values of y and z in that order):

$$0 \xrightarrow{0/00} 0;\ 0 \xrightarrow{1/00} 1;\ 0 \xrightarrow{1/01} 1;\ 1 \xrightarrow{0/10} 0;\ 1 \xrightarrow{0/11} 0;\ 1 \xrightarrow{1/11} 1;$$

The output y does not await the input x. The output z awaits the input x.

3. The component DoubleSplitDelay has input variable *in*, output variable *out*, sate variables x_1 and x_2, and local variable *temp*, all of type bool. Its reaction description consists of 4 tasks as described in Figure 2. The output variable *out* does not await the input variable *in*.

4. Each state is denoted by listing the values of the variables *west*, *east*, $mode_W$ and $mode_E$, in that order. We use a, w, b, g, and r as abbreviations for the values away, wait, bridge, green, and read, respectively. Then, the initial state is *ggba*, and has transitions to itself and to states *rgaw*, *grwa*, and *rgww*. To compute the set of reachable states, we

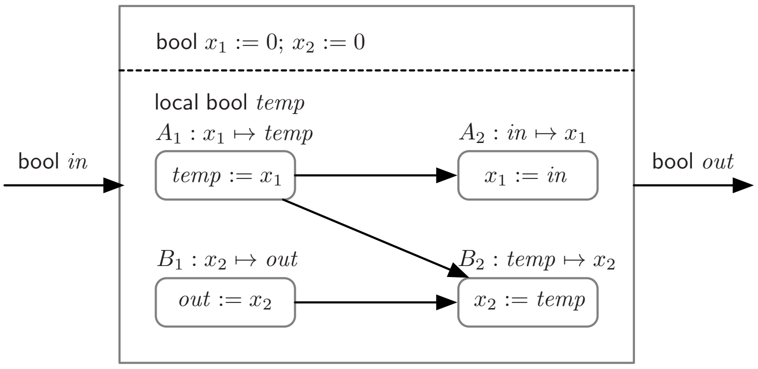

Figure 2: DoubleSplitDelay

need to explore transitions from these three newly discovered states, and keep repeating till no new states are found to be reachable. It turns out that the following 13 sates are reachable:

$$\begin{array}{ccccccc} ggaa & rgaw & grwa & rgww & rgab & rrwb & grba \\ rrbw & rgwb & ggwa & ggba & rgbw & rgbb & \end{array}$$

5. The transition system Mult$(m,\ n)$ has 3 state variables: *mode* of the enumerated type {loop, stop}, x of type nat, and y of type nat. The sole initial state is (loop, m, 0). The set of transitions is defined as follows: for every positive natural number a and for every natural number b, there is a transition from the state (loop, a, b) to the state (loop, $a-1$, $b+n$); and for every natural number b, there is a transition from the state (loop, 0, b) to the state (stop, 0, b). The transition system is deterministic, and has the following execution:

$$\begin{array}{l} (\text{loop},\ m,\ 0) \ \rightarrow\ (\text{loop},\ m-1,\ n)\ \rightarrow \cdots \rightarrow\ (\text{loop},\ m-j,\ nj)\ \rightarrow \cdots \rightarrow \\ (\text{loop},\ 0,\ mn)\ \rightarrow\ (\text{stop},\ 0,\ mn). \end{array}$$

6. The process AsyncAnd maintains the following state variables:

bool $x_1 := 0$; $x_2 := 0$; $x := 0$; {stable, unstable, hazard} *mode* := stable

The variables x_1 and x_2 are used to remember the most recent values of the input variables in_1 and in_2, respectively; the variable x corresponds to the value of the output; and the mode variable indicates whether the gate is stable, or unstable, or has encountered a hazard.

Since the state variable x is intended to correspond to the current output, the output task is simply given by $out\,!\,x$.

The input task responsible for processing the input in_1 is specified by the update code:

```
$x_1 := in_1$;
if ($mode$ = stable) $\wedge$ ($x \neq x_1 \wedge x_2$) then $mode$ := unstable
else if ($mode$ = unstable) $\wedge$ ($x = x_1 \wedge x_2$) then $mode$ := hazard.
```

The logic is analogous to that of the asynchronous process AsyncNot. In the mode stable, if the newly received input warrants a change in the output, that is, when the value of x differs from the desired conjunction $x_1 \wedge x_2$, the mode switches to unstable indicating a pending change in the output. In the mode unstable, if the newly received input warrants another change in the output, the mode switches to hazard indicating unpredictable output. The input task responsible for processing the input in_2 is symmetric.

The internal task is responsible for the changes in the value of the state variable x. In the mode unstable, a change in the output value is pending, and the internal task can flip the value of x switching the mode to stable, while in the mode hazard, the output can change to an arbitrary value. Thus, the internal task is specified by the update code:

```
if ($mode$ = unstable) then {$x$ := $\neg x$; $mode$ := stable}
else if ($mode$ = hazard) then $x$ := choose{0, 1}.
```