Homework Assignment 1

CS 599: Autonomous Cyber-Physical Systems

Instructor: Jyotirmoy V. Deshmukh

Due Date: Feburary 1, 2018, Time: AoE (UTC -12)

Problem 1. [25 points]
(a) In this problem, we will design a synchronous reactive component to set the cruising speed of a vehicle. This component corresponds to the `SetSpeed` component in the cruise-control system `CruiseController`. The `SetSpeed` component takes the following inputs:

1. `event(bool)` *cruise*: This event models the user turning the cruise control on or off.
2. `nat` *speed*: This input models the speed input from the vehicle (corresponds to the current speed of the vehicle).
3. `event` *inc*: This input models the user requesting an increase in the cruising speed.
4. `event` *dec*: This input models the user requesting a decrease in the cruising speed.

It has two outputs:

1. `event(nat)` *cruiseSpeed*: This output, if present, contains the current cruising speed.
2. `bool status`: This output is 1 if the cruise control is on, and is 0 if it is off.

There are two constants called `minSpeed` and `maxSpeed` that we will use within the component. The operation of the component is as follows. You can assume that the component uses one or more state variables to keep track of its state. Let one of the state variables be a Boolean variable called *on*. The component updates the state variable *on* according to the following rule: every time the event *cruise* occurs, the variable *on* is toggled. Whenever the *on* variable is set to 1, then the component outputs the cruising speed using the *cruiseSpeed* output variable. If the current speed *speed* is within the legal range `minSpeed` and `maxSpeed`, then the *cruiseSpeed* variable is set to *speed*; if not, it is set to the closest legal value (i.e. `minSpeed` or `maxSpeed`). Also, when *on* is 1, if a *dec* event is received, then the cruising speed is decremented (restricting the lowest speed to `minSpeed`), and when the *inc* event is received, the cruising speed is incremented (restricting the highest speed to `maxSpeed`). If the *on* variable is set to 0, all input events except *cruise* are ignored, and there is no output

produced on *cruiseSpeed*, and we output a 0 on `status`. Show the synchronous reactive component that implements this functionality.

(b) Now consider the design of the `SetSpeed` component with two additional input signals: *brake* and *resume*. Whenever the `SetSpeed` component has the *on* state set to 1, if it receives the *brake* command, the `SetSpeed` component stops producing the output (i.e. can be considered to be temporarily suspended). In this state, the *cruiseSpeed* output should be absent, the `status` output should still indicate that the cruise control is on, and events *inc* and *dec* should be ignored. Upon receiving the *resume* event, the cruise controller should restore the desired speed to the last known cruise speed setting. In this state, receiving a *cruise* event should turn the `SetSpeed` component off, and reset all state variables to the initial states. In the off state, receiving *resume* or *brake* events will have no effect. Redesign `SetSpeed` with the new functionality. Comment on whether you require additional state variables. You can choose to show your design either as a synchronous reactive component block diagram, or as an extended state machine.

Problem 2. [25 points] Design a synchronous reactive component with the following functionality. It has three inputs A, B and reset, and one output O. All inputs and outputs are Boolean. It starts in an idle state, and once it receives an input on both A and B (in the same round or in different rounds), it emits output on O. If in any round it receives the input reset, it goes back to the idle state without emitting output. Show the extended state machine representing this component. Explain if your component is finite-state, if it is deterministic, and if it is input-enabled in one sentence each.

** If you are interested, read the wikipedia article on the synchronous programming language Esterel. The article shows the Esterel code for this example!

Problem 3. [25 points] When you have a team of autonomous agents trying to achieve consensus over some action, it is common to employ a distributed consensus seeking algorithm. There are numerous distributed consensus algorithms in the literature, and in this example, we will look at a very simple form of consensus seeking. Consensus algorithms are usually backed with a proof that as the number of communication rounds between agents tends to infinity, the "information" states of the agents converge to each other asymptotically.

In this problem, we will consider three UAVs that are trying to decide on a location to rendezvous. Each UAV uses an asynchronous process to achieve consensus. Consider the asynchronous process P_i for agent i shown in Fig. 1:

Process P_i has three state variables, both of type `real`. The variable x_i stores a desired location for the rendezvous for process P_i. The variables sum_i and $nRcv_i$ are used as temporary variables during computation of the average of the information states across all processes. It has three inputs:

- `bool` $sendReq_j$,$sendReq_k$: These inputs receive requests from Processes P_j and P_k respectively.

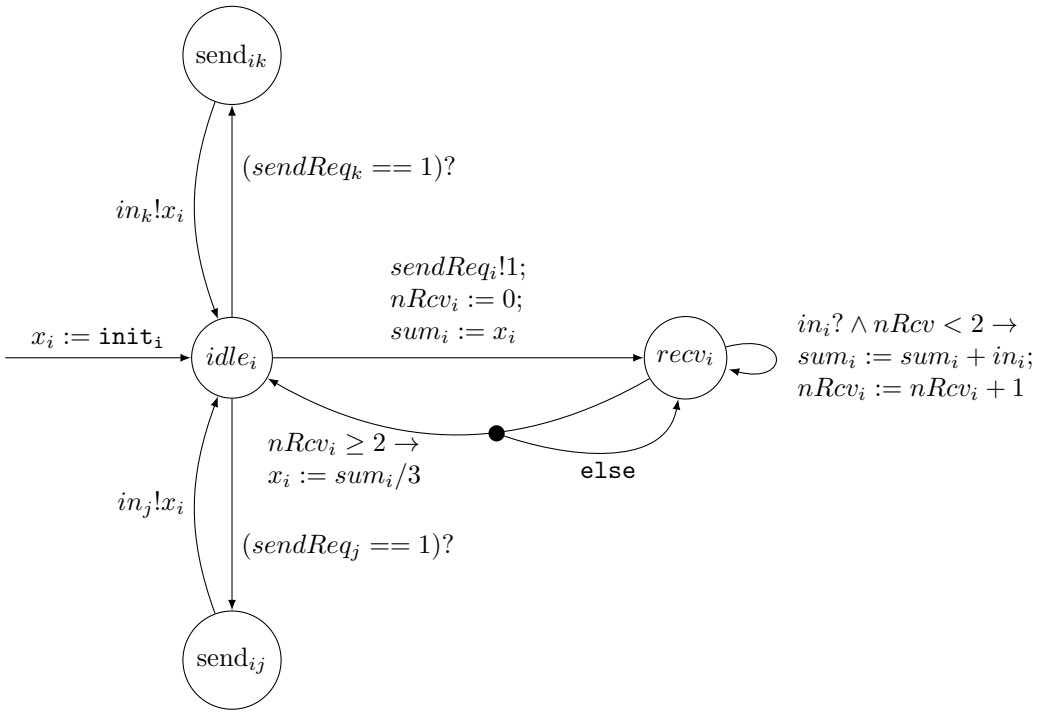

Figure 1: Process P_i

- `real` in_i : This input receives some real value from either P_j or P_k.

It has three outputs:
- `bool` $sendReq_i$: This output channel is used to send a request to processes P_j and P_k.
- `real` in_j, in_k: These output channel is used to send the values of the state variable x_i to processes P_j and P_k respectively.

The extended state machines for Process P_j and P_k are symmetric versions of Fig. 1, i.e. for process P_j, all indices i and j are swapped, and for process P_k, $k \leftrightarrow i$.

(a) Consider an execution where the three processes are initialized such that $\texttt{init}_\texttt{i}, \texttt{init}_\texttt{j}, \texttt{init}_\texttt{k}$ are respectively 10, 15, and 17.Show one possible execution containing at least 10 actions of the composition of the three processes. Assume that the scheduler chooses the process executions such that P_i sends a request first, and when P_i transitions back to the state $idle_i$, P_j sends a request, and when P_j transitions back to the state $idle_j$, P_k sends a request.

(b) What would change if you had one more autonomous agent? Sketch the modified asynchronous process P_i.

Problem 4. [25 points] Consider the following timed process:

- Set of input channels I = {`in`},
- Set of output channels O = {$\texttt{out}_1, \texttt{out}_2$},
- Set of clock variables C (to be determined)
- Set of discrete modes (to be determined)

The behavior of the timed process can be described in following steps:
Step 1: It awaits an input on `in` and once it receives this input, it produces an output on $\texttt{out}_1$ at least 3 seconds later, and at most 5 seconds later. It ignores any input received during this time (i.e. after receiving input on `in` and before producing output on out_1).
Step 2: After producing the output on $\texttt{out}_1$, it again awaits an input on `in`. Once it receives this input, it then produces an output on $\texttt{out}_2$ at most 4 seconds later. Again, it ignores any inputs received in the interim. After producing output on $\texttt{out}_2$, it goes back to Step 1.

(a) Draw a state machine representation for the timed process.
(b) Make a note of how many clock variables you had to use and the number of discrete modes.