Homework Assignment 1

CS 599: Autonomous Cyber-Physical Systems

Instructor: Jyotirmoy V. Deshmukh,
Grader: Nicole Fronda

Due Date: Feburary 4, 2019, Time: 2pm PT

Problem 1. (**Synchronous Reactive Components**) [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 rules: every time the event *cruise* occurs, the variable *on* is toggled (i.e. if the state variable *on* is 1, it becomes 0 and if it is 0, it becomes 1.). The component's behavior is different accordingly to the state of the *on* variable:
(1) $on = 1$: If $on = 1$, then the component outputs the cruising speed using the *cruiseSpeed* output variable. You can imagine this as the command being passed to a downstream module that is responsible for regulating the vehicle speed to the input from the `SetSpeed` module. 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`). 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`). Finally, the component outputs a 1 on the `status` output whenver $on = 1$.
(2) $on = 0$: If $on = 0$, then 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. You can depict the component using the notation shown in the slides, or you can depict it as a state machine.

(b) Now consider the design of the `SetSpeed` component with two additional input signals: *brake* and *resume*. Whenever the `SetSpeed` component has $on = 1$, if it receives the *brake* command, it stops producing the output (i.e. can be considered to be temporarily suspended). In this new state, the *cruiseSpeed* output is absent, the `status` output is still equal to 1, the events *inc* and *dec* are ignored, and receiving the *cruise* event causes *on* to toggle to 0 (resetting all state variables to their initial states). When on =1, receiving the *resume* event causes `SetSpeed` to restore the *cruiseSpeed* output to the last known cruise speed setting.

If $on = 0$, 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. (**Asynchronous Reactive Components**) [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_1 for agent 1 shown in Fig. 1:

Process P_1 has three state variables, both of type `real`. The variable x_1 stores a desired location for the rendezvous for process P_1. The variables sum_1 and $nRcv_1$ are used as temporary variables during computation of the average of the information states across all processes. It has three inputs:

- `bool` $sendReq_2$,$sendReq_3$: These inputs receive requests from Processes P_2 and P_3 respectively.
- `real` in_1 : This input receives some real value from either P_2 or P_3.

It has three outputs:

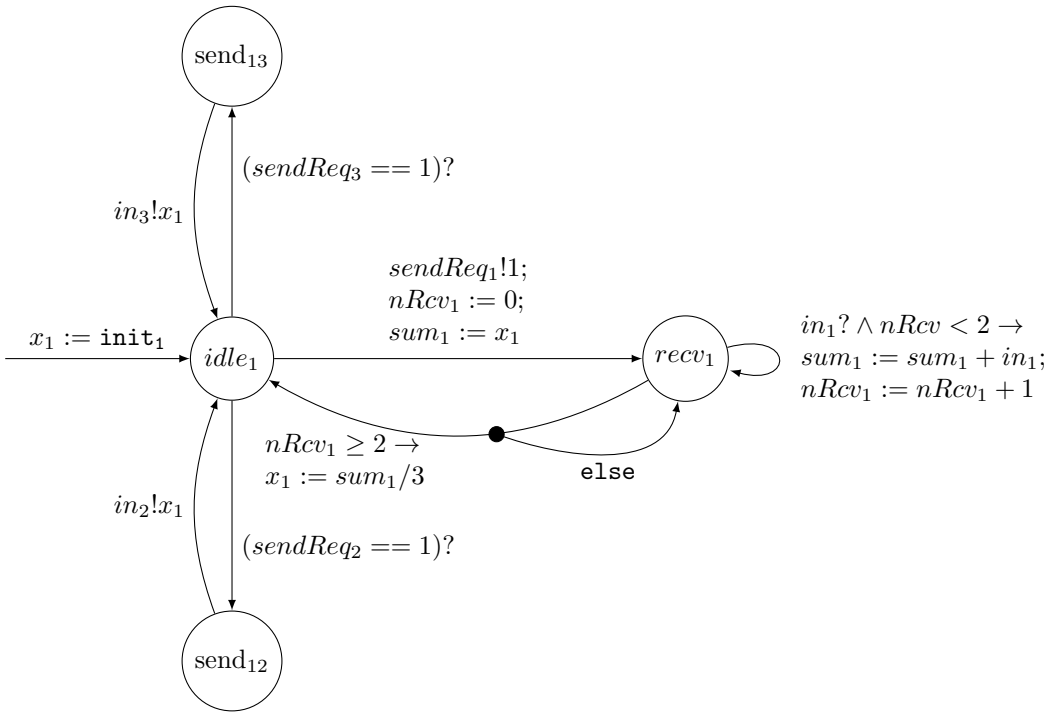

Figure 1: Process P_i

- `bool` $sendReq_1$: This output channel is used to send a request to processes P_2 and P_3.
- `real` in_2, in_3: These output channel is used to send the values of the state variable x_1 to processes P_2 and P_3 respectively.

The extended state machines for Process P_2 and P_3 are symmetric versions of Fig. 1, i.e. for process P_2, swap indices 2 and 1, and for process P_3, swap 1 and 3.

(a) Consider an execution where the three processes are initialized such that $\texttt{init}_1, \texttt{init}_2, \texttt{init}_3$ 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_1 sends a request first, and when P_1 transitions back to the state $idle_1$, P_2 sends a request, and when P_2 transitions back to the state $idle_2$, P_3 sends a request. Recall that these are three asynchronous processes communicating! This means that the sends and receives have to execute in a synchronized fashion, but no other transitions are synchronized.

(b) What would change if you had one more autonomous agent P_4? Sketch the modified asynchronous process P_1.

Problem 3. (**Timed Components**) [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:

1. It awaits some symbol (say a) on in and once it receives this input, it outputs a on $\texttt{out}_1$ at least 6 seconds later and at most 10 seconds later.
2. After receiving the first a, it continues waiting for a second a. If it does not receive a second a within 5 seconds, it goes into a special error state.
3. If it does receive a second a within 5 seconds of receiving the first a, it outputs the second a on the output channel $\texttt{out}_2$, at least 5 seconds after and at most 8 seconds after it outputs on channel $\texttt{out}_1$.
4. Once it receives the two a's as specified above, any further a's are ignored till both outputs are produced. Once both outputs are produced, it goes back to waiting for the first input.
5. If it is in the error state, it stays in the error state for at most 60 seconds, and then goes back to waiting for the first input.

(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.
(c) Can you make do with lesser number of clocks?

Problem 4. (**Dynamical Systems**) [25 points] The state-space model for controlling a DC motor are given below:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & \frac{K}{J} \\ -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} v \tag{1}$$

Here, the state x_1 represents the angular velocity of the motor shaft, x_2 represents the armature current, and v represents the input voltage. The meaning of various symbols appearing in the above equations and their typical values are provided below:

- J = 0.01 kg m^2 : Moment of inertia of the rotor
- b = 0.1 N.m.s : Motor viscous friction constant
- K = 0.1 (value representing either electromotive force constant/ motor torque constant)
- R = 1 ohm (electrical resistance)

- L = 0.5 H (electrical inductance)

(a) If the input voltage v is 0V, is the resulting system stable in the sense of Lyapunov to the equilibrium point $(0, 0)$? What does this mean in the physical sense?
(b) Suppose at time 0, the values of x_1 and x_2 are respectively 0.2 and 1, if the input voltage is 0V, what will the values of x_1 and x_2 be as time goes to ∞?
(c) Suppose at time 0, the values of x_1 and x_2 are respectively 0.2 and 1, if the input voltage is 1V, how will you compute the values of x_1 and x_2 be as time goes to ∞? **Hint**: You do not have to compute a matrix exponential!

(d) Suppose we have to use a *full state-feedback* controller $v = -C \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, where $C = [10 - 1]$, is the resulting system stable?

(e) Suppose, there was a bug in coding the controller, and instead of using the value of C as indicated above, we used the value $C = [10 - 2]$, what happens to the system stability?
Tip: Feel free to use Matlab® or Python to calculate eigenvalues.