Verification CA#2

Student Name:
Pouya Haji Mohammadi Gohari

SID:810102113

Date of deadline
Thursday 9th February, 2023

Dept. of Computer Engineering

University of Tehran

Contents

1 Introduction

In this CA we are going to implement the previous project in rebeca where we have already done it in promela. The actuators responsibility:

- Sensor: This actuator must collect data from it's enviroment and send one byte at a time for corresponded CPU.However in Rebeca env we will abstract the data of sensor and assume it will send the number 1 each time.
- Computing Unit: It has two main responsibility where the first one is to pack the data from corresponding sensors into 4 bytes, then sending pack data for Radio Communication as it's second duty.
- Radio Communication: This actuator will recive the packed data from it's corresponded Computing Unit and then send it to Collector.
- Collector: Each radio communication units will send the data for this collector with this constraints that if a particular RCD[1] which already sent it's own data, must wait untill remaining RCD's(where they didn't send their data yet) send their data then it has permission to send again.

The procedure of the report is:
Firstly we are going to review on my Core Rebeca model and discuss why I decided to add some functions and methods.At next step we going to explain what my code does and analyse each reactive class. Eventually we are going to define a LTL property to make sure that each sensor eventually will send it's data through actuators to collector. We will do the same procedure to the timed-rebeca model as well.

[1]Radio Communication

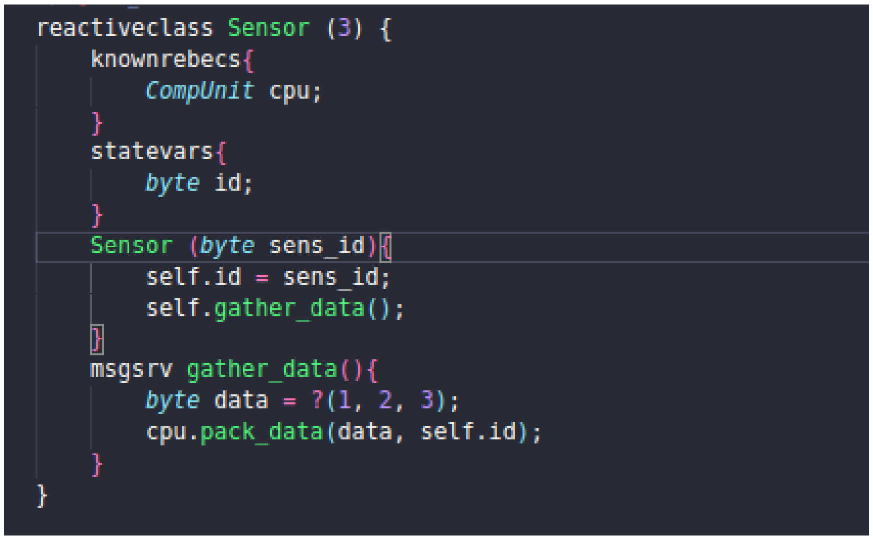
2 Core Rebeca Model

2.1 Review

In this section we are going to define each reactiveclass alongside with methods and brifely explain each.

2.1.1 Sensor

Every sensor is designed to collect 1 byte of data and transmit it to the associated CPU, which is identified through predefined connections. Each sensor is also assigned a unique identifier, which is provided during its creation via the constructor. Upon the instantiation of a sensor, it automatically initiates its message service (msgsrv) named "gather data" to start the data collection process.



```
reactiveclass Sensor (3) {
    knownrebecs{
        CompUnit cpu;
    }
    statevars{
        byte id;
    }
    Sensor (byte sens_id){
        self.id = sens_id;
        self.gather_data();
    }
    msgsrv gather_data(){
        byte data = ?(1, 2, 3);
        cpu.pack_data(data, self.id);
    }
}
```

Figure 1: Sensor actuator

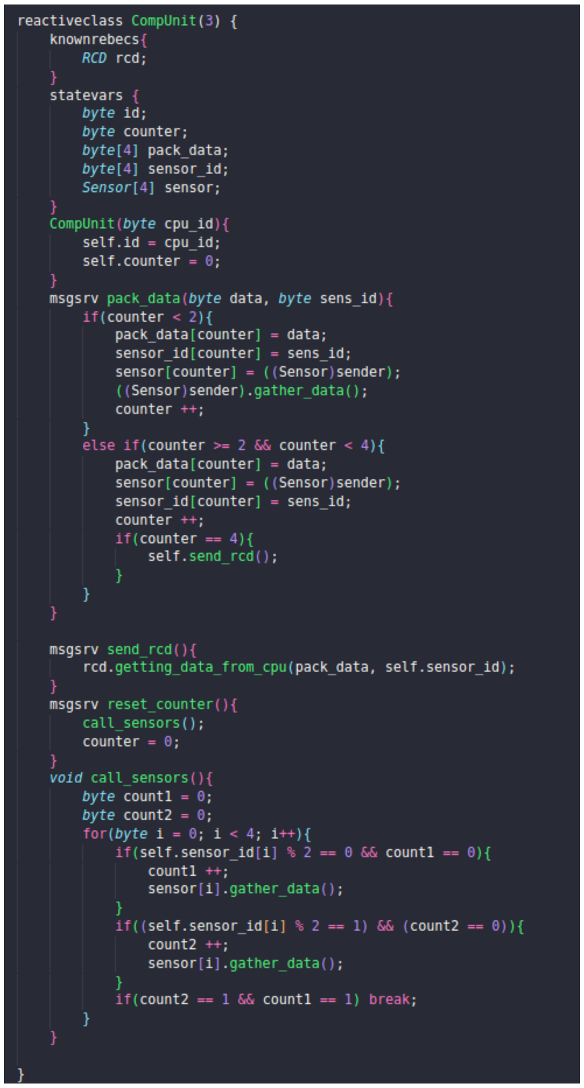
As illustrated in Figure 1, each sensor is configured to recognize its corresponding CPU actuator and is assigned a unique identifier, which is stored in its state variables. To simplify the scenario and avoid overcomplicating our model (specifically, to prevent an expansion of the state space within the transition system), we assume that the sensor transmits a data value of 1(In figure 1 it is set to $?(1, 2, 3)$). Within its message service function named "gather data," the sensor is responsible for forwarding this data to the CPU.

2.1.2 CPU

The actuator is programmed to identify the RCD to which it must transmit the data. Three distinct message services have been developed for this actuator to facilitate its operations:

- Pack data: This massage server will get the data 1 byte at a time from it's sensors and pack them into 4 bytes.
- Send the data to RCD: After packing data was completed, it should send it to the RCD.
- Reset Counter: This massage server designed to help us to reset the counter and get data from sensors again.

We have defined multiple state variables in CPU's actuator as illustrated in figure 2:



```
reactiveclass CompUnit(3) {
    knownrebecs{
        RCD rcd;
    }
    statevars {
        byte id;
        byte counter;
        byte[4] pack_data;
        byte[4] sensor_id;
        Sensor[4] sensor;
    }
    CompUnit(byte cpu_id){
        self.id = cpu_id;
        self.counter = 0;
    }
    msgsrv pack_data(byte data, byte sens_id){
        if(counter < 2){
            pack_data[counter] = data;
            sensor_id[counter] = sens_id;
            sensor[counter] = ((Sensor)sender);
            ((Sensor)sender).gather_data();
            counter ++;
        }
        else if(counter >= 2 && counter < 4){
            pack_data[counter] = data;
            sensor[counter] = ((Sensor)sender);
            sensor_id[counter] = sens_id;
            counter ++;
            if(counter == 4){
                self.send_rcd();
            }
        }
    }

    msgsrv send_rcd(){
        rcd.getting_data_from_cpu(pack_data, self.sensor_id);
    }
    msgsrv reset_counter(){
        call_sensors();
        counter = 0;
    }
    void call_sensors(){
        byte count1 = 0;
        byte count2 = 0;
        for(byte i = 0; i < 4; i++){
            if(self.sensor_id[i] % 2 == 0 && count1 == 0){
                count1 ++;
                sensor[i].gather_data();
            }
            if((self.sensor_id[i] % 2 == 1) && (count2 == 0)){
                count2 ++;
                sensor[i].gather_data();
            }
            if(count2 == 1 && count1 == 1) break;
        }
    }

}
```

Figure 2: Computing Unit actuator

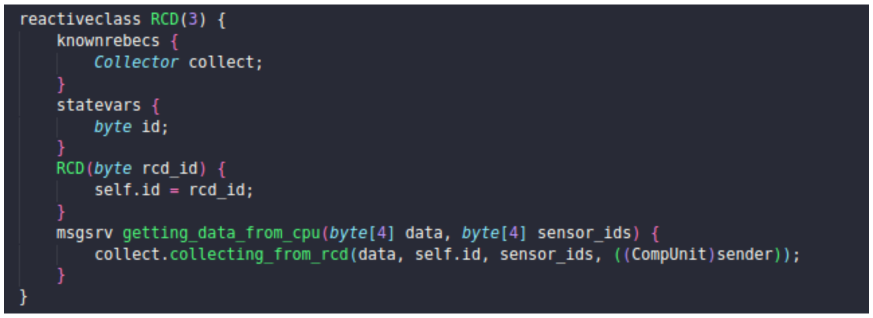
In order to packing data from sensors we have defined counter and pack data.We know that if we get an instance from sensor reactiveclass it will send one data to cpu so while counter is less than 2 we call senders to send again. But after counter reach to the number two, we won't call sensor in order to avoid queue overflow for this actuator. (Cuase CPU will already see it's 4 data to pack, however even if we collect sensors so much, firstly it would cuase cpu to queue overflow and secondly it can be considered to be missed data since rcd will send data to get his aknowledge) Whenver the counter is equal to 4, the data is ready to be transmitted to the RCD using the "send to rcd" message service. Assuming the data has been successfully sent from the CPU to the RCD, the sensors that have already transmitted their data can be called once more with Sensor's arrays.

Once the CPU has finished packaging the data, it will also pass the IDs of the sensors to RCD from which the data was received. An array has been set up to store these sensor IDs(In order to use it LTL model checking) This array is designed to make sure that each sensor's data will be accounted.

In the "reset counter" message service, the system will initiate a new round of data transmissions by prompting the sensors to resend their data.

2.1.3 RCD

The known rebecs that the RCD is aware of include the Collector unit (actuator), and each RCD will have a unique identifier (ID). These actuators are responsible for handling the data received from the CPU. They will receive packed data from the CPU and transmit it, along with the sensor IDs, to the collector.The code has been illustrated in figure 3:



```
reactiveclass RCD(3) {
    knownrebecs {
        Collector collect;
    }
    statevars {
        byte id;
    }
    RCD(byte rcd_id) {
        self.id = rcd_id;
    }
    msgsrv getting_data_from_cpu(byte[4] data, byte[4] sensor_ids) {
        collect.collecting_from_rcd(data, self.id, sensor_ids, ((CompUnit)sender));
    }
}
```

Figure 3: Radio center device actuator

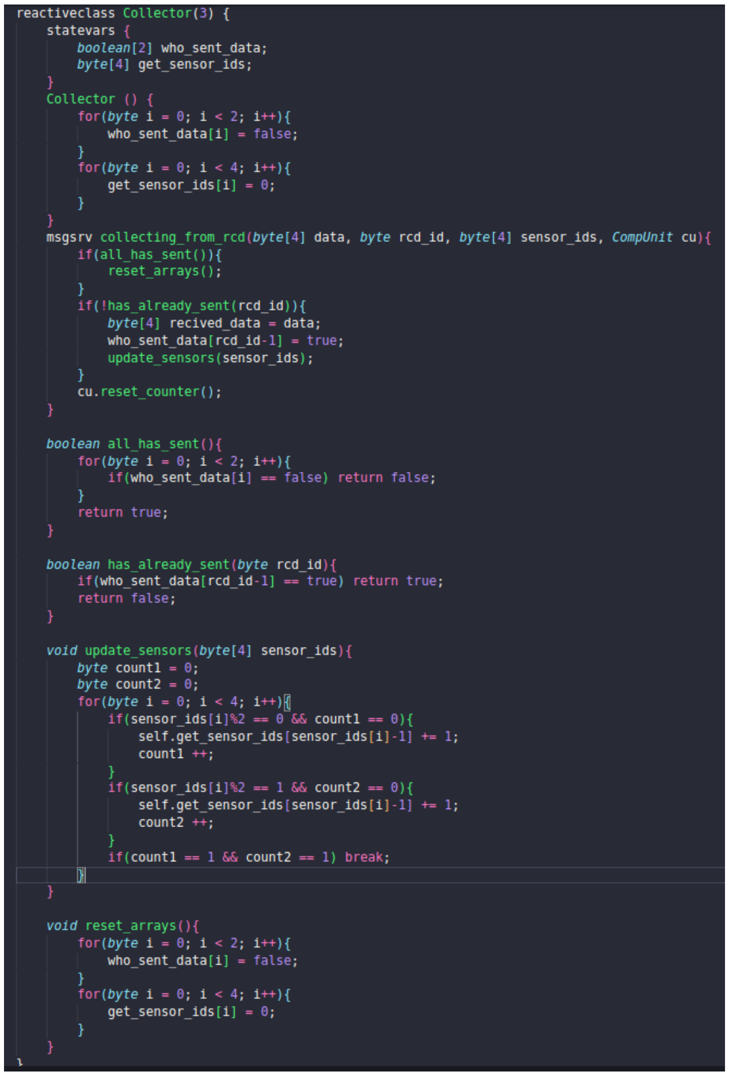
2.1.4 Collector

In this actuator we have created two array where the purpoose of each are:

- Who Sent data: The size of boolean array is same for the number of nodes we will create where the purpose is we want to control whether a RCD has already transmitted data or not.If a single RCD has already transmitted data thus it must wait to other RCDs transmit their data.
- Get Sensor Id's:The size is same number of noder multiply by two(since for each node we have two sensors).The purpose is to see if all data of each sensor will eventually reach to Collector on some points or not.

Whenever we get an instance of this actuator, the ”boolean” and ”sensors id's” arrays will initialize with false and zero respectively. The ”collecting from rcd” massage server has been designed in order to get the data from RCD.As explained before If all RCD's has transmitted data, we will reset every array in it's state variables.
The implemented code is illustrated in figure 4:

Figure 4: Collector actuator



```
reactiveclass Collector(3) {
    statevars {
        boolean[2] who_sent_data;
        byte[4] get_sensor_ids;
    }
    Collector () {
        for(byte i = 0; i < 2; i++){
            who_sent_data[i] = false;
        }
        for(byte i = 0; i < 4; i++){
            get_sensor_ids[i] = 0;
        }
    }
    msgsrv collecting_from_rcd(byte[4] data, byte rcd_id, byte[4] sensor_ids, CompUnit cu){
        if(all_has_sent()){
            reset_arrays();
        }
        if(!has_already_sent(rcd_id)){
            byte[4] recived_data = data;
            who_sent_data[rcd_id-1] = true;
            update_sensors(sensor_ids);
        }
        cu.reset_counter();
    }

    boolean all_has_sent(){
        for(byte i = 0; i < 2; i++){
            if(who_sent_data[i] == false) return false;
        }
        return true;
    }

    boolean has_already_sent(byte rcd_id){
        if(who_sent_data[rcd_id-1] == true) return true;
        return false;
    }

    void update_sensors(byte[4] sensor_ids){
        byte count1 = 0;
        byte count2 = 0;
        for(byte i = 0; i < 4; i++){
            if(sensor_ids[i]%2 == 0 && count1 == 0){
                self.get_sensor_ids[sensor_ids[i]-1] += 1;
                count1 ++;
            }
            if(sensor_ids[i]%2 == 1 && count2 == 0){
                self.get_sensor_ids[sensor_ids[i]-1] += 1;
                count2 ++;
            }
            if(count1 == 1 && count2 == 1) break;
        }
    }

    void reset_arrays(){
        for(byte i = 0; i < 2; i++){
            who_sent_data[i] = false;
        }
        for(byte i = 0; i < 4; i++){
            get_sensor_ids[i] = 0;
        }
    }
}
```

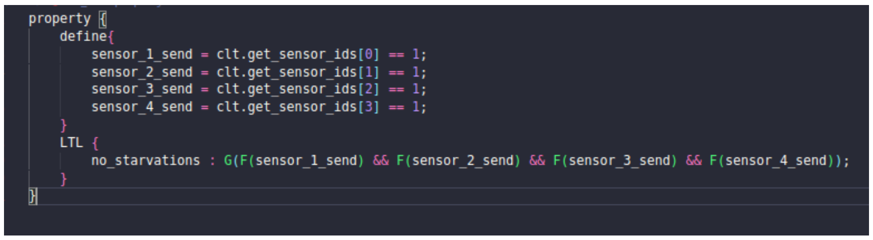
Attention: If we want to add nodes we must do it mannually since rebeca language will not support getting instantiation dynamicaly in main!So if we want add nodes we must careful to change the size of arrays in Collector's variables and consider changing foor loops in it's methods, increament the queue size of collector corresponding to number of nodes.
There are two same functions in Collector and CPU actuators where in Collector we are updating sensors to see if a partiucal sensor send the data through CPU,RCD to Collector or not.
However in CPU whenever Collector reset counters we will call gather data for sensors.

2.2 LTL property

In rebeca we have created a property which the name is no starvation. The LTL propery is:

$$\text{No starvation} = \Box\,(\Diamond(\text{Sensor 1 sent}) \wedge \Diamond(\text{Sensor 2 sent}) \wedge \cdots \wedge \Diamond(\text{Sensor 2n sent})) \tag{1}$$

Where n is the number of nodes.The syntax for $\Box$ and $\Diamond$ are G(Globally) and F(Finally) respectively. The defined property in rebeca language has been illustrated in figure 5:



```
property {
    define{
        sensor_1_send = clt.get_sensor_ids[0] == 1;
        sensor_2_send = clt.get_sensor_ids[1] == 1;
        sensor_3_send = clt.get_sensor_ids[2] == 1;
        sensor_4_send = clt.get_sensor_ids[3] == 1;
    }
    LTL {
        no_starvations : G(F(sensor_1_send) && F(sensor_2_send) && F(sensor_3_send) && F(sensor_4_send));
    }
}
```

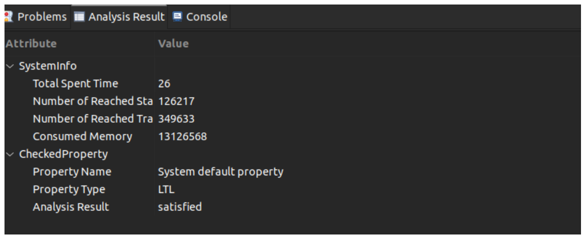
Figure 5: No starvation property

2.3 Analysis Results

In sensors we set the data nondeterministicly to choose between 1 and 2 and 3:

$$data =?(1, 2, 3) \tag{2}$$

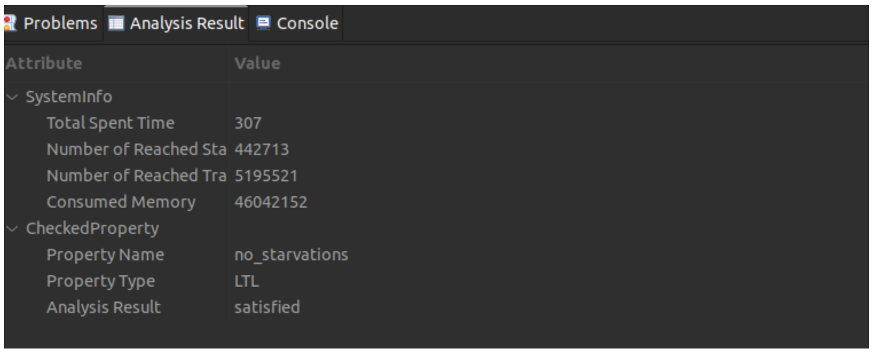
For simplicity we can set it to 1 since nondeterminism will explode our transition system thus we will create 2 node at first to see if our model will correctly run and satisfy the default property defined in rebeca(deadlock, queue overflow and assertions) or not. The result of two nodes has been illustrated in figure 6:



```
Attribute                 Value
⌄ SystemInfo
    Total Spent Time          26
    Number of Reached Sta     126217
    Number of Reached Tra     349633
    Consumed Memory           13126568
⌄ CheckedProperty
    Property Name             System default property
    Property Type             LTL
    Analysis Result           satisfied
```

Figure 6: Result for two nodes

As it's been illustrated in figure 6 we can see that for two nodes, the system default property has been satisfied and approximately created 126k states with 350k transitions.
Checking if no starvation will be satisfied or not:



Problems | Analysis Result | Console

Attribute	Value
SystemInfo	
Total Spent Time	307
Number of Reached Sta	442713
Number of Reached Tra	5195521
Consumed Memory	46042152
CheckedProperty	
Property Name	no_starvations
Property Type	LTL
Analysis Result	satisfied

Figure 7: Satification of No starvation

As the figure 7 has been illustrated that we have no starvation thus every sensor's data will eventually reach to the Collector actuator.
Question: Why reachable states gets higer than default properties?
Becuase when we define LTL the rebeca will automatically generate NBA[2] and multiply it to our transition system therefor the reachable states increased.

[2]Nondeterministic Buchi Automata

3 Timed Rebeca Model

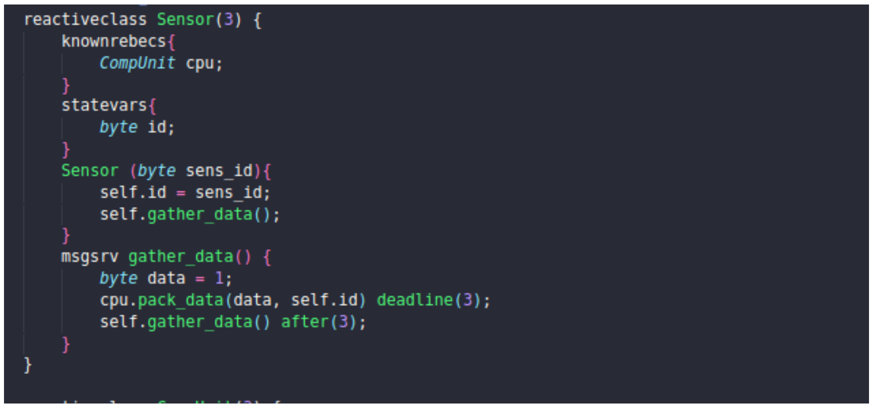
3.1 Review

In time rebeca we have 3 different time fucntions where:

- Deadline: What is the maximum time that this message is valid.
- Delay: Computation time.
- After: How long does it take to deliver message.(Used for both message delivery time and periodic behavior)

3.1.1 Sensor

Like before Sensors has a identifier variablity (ID).When get an instance of this object it will call " gather data" massage server in order to transmit data.
When this actuator wants to transmit data for CPU it will set a deadline of 3 time units where if CPU response after 3 time units it will cause deadline to be missed.
In "gather data" message server Sensor will periodicly(3 time units) call itself to gather data from it's enviroment. The figure 8 will illustrate this concept more:

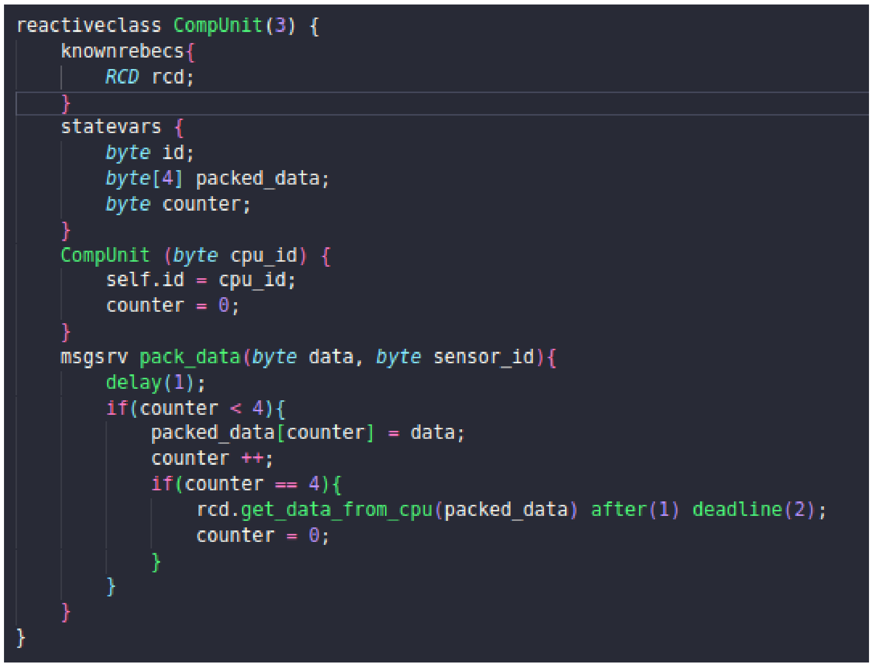


```
reactiveclass Sensor(3) {
    knownrebecs{
        CompUnit cpu;
    }
    statevars{
        byte id;
    }
    Sensor (byte sens_id){
        self.id = sens_id;
        self.gather_data();
    }
    msgsrv gather_data() {
        byte data = 1;
        cpu.pack_data(data, self.id) deadline(3);
        self.gather_data() after(3);
    }
}
```

Figure 8: Sensor of timed rebeca model

3.1.2 CPU

The CPU are same as before but it has 1 time unit delay represent as computational delay. Furthermore it will pack transmitted data from sensors into 4 data units and transmit them to rcd after 1 time units and 2 time units as deadline.If RCD response to CPU after any time of 3 it cause us to have a deadline missing. The code implented is in figure 9:

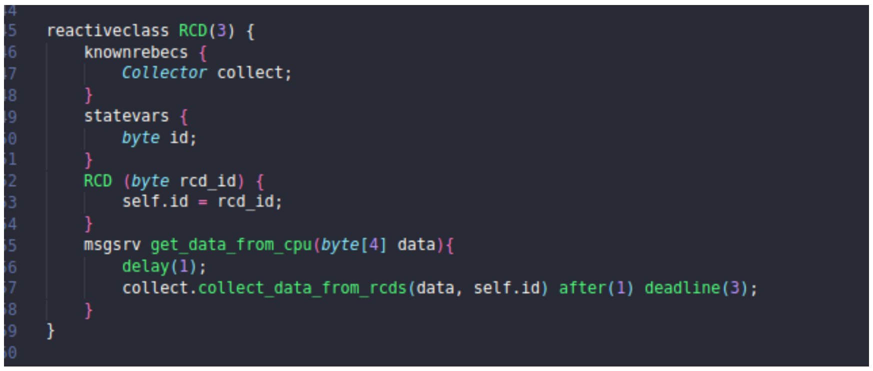


```
reactiveclass CompUnit(3) {
    knownrebecs{
        RCD rcd;
    }
    statevars {
        byte id;
        byte[4] packed_data;
        byte counter;
    }
    CompUnit (byte cpu_id) {
        self.id = cpu_id;
        counter = 0;
    }
    msgsrv pack_data(byte data, byte sensor_id){
        delay(1);
        if(counter < 4){
            packed_data[counter] = data;
            counter ++;
            if(counter == 4){
                rcd.get_data_from_cpu(packed_data) after(1) deadline(2);
                counter = 0;
            }
        }
    }
}
```

Figure 9: CPU of timed rebeca model

3.1.3 RCD

This actutator will process it's queue with one time unit delay.Additionally, we have set a deadline for RCD as well, the deadline time is 3(This means if Collector does not response any time sooner than 3, the deadline will be consider missed).

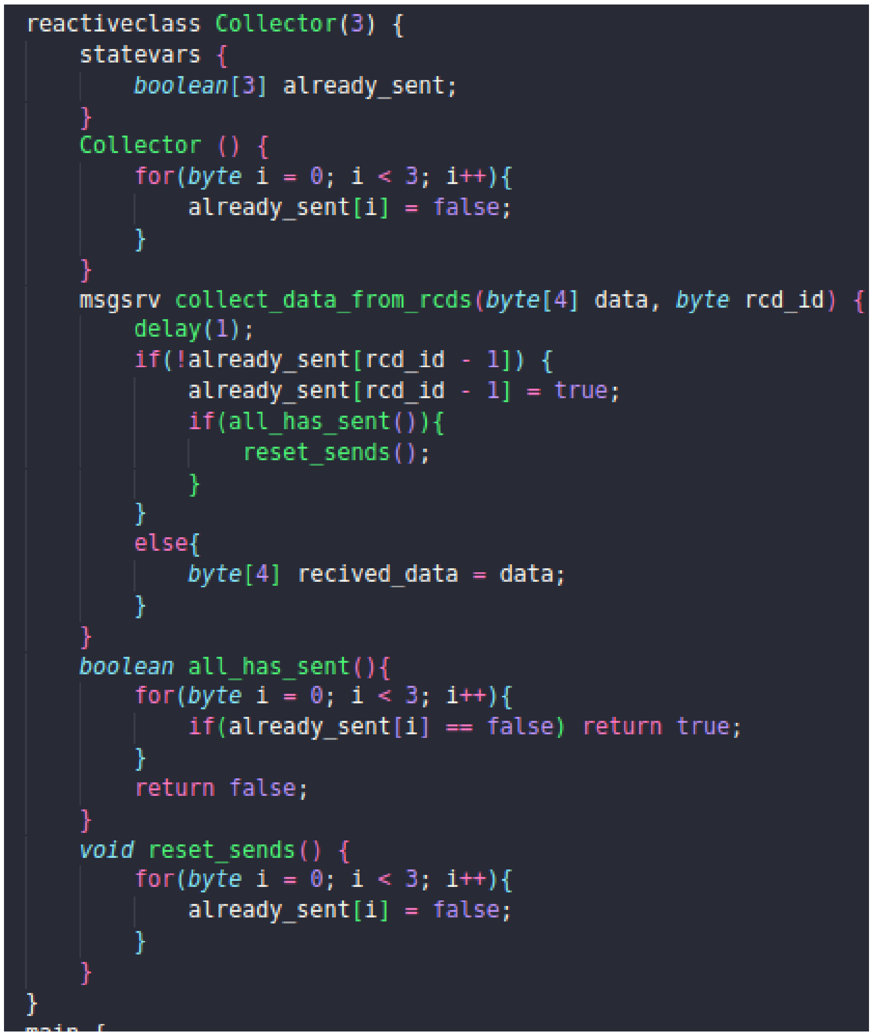


```
reactiveclass RCD(3) {
    knownrebecs {
        Collector collect;
    }
    statevars {
        byte id;
    }
    RCD (byte rcd_id) {
        self.id = rcd_id;
    }
    msgsrv get_data_from_cpu(byte[4] data){
        delay(1);
        collect.collect_data_from_rcds(data, self.id) after(1) deadline(3);
    }
}
```

Figure 10: RCD of timed rebeca model

3.1.4 Collector

This actuator decide to which RCD's data will be taken.Since this actuator will not transmit to any particular device, thus we didn't established any after or deadline times.It just has a delay(1 time unit) for computational works.



```
reactiveclass Collector(3) {
    statevars {
        boolean[3] already_sent;
    }
    Collector () {
        for(byte i = 0; i < 3; i++){
            already_sent[i] = false;
        }
    }
    msgsrv collect_data_from_rcds(byte[4] data, byte rcd_id) {
        delay(1);
        if(!already_sent[rcd_id - 1]) {
            already_sent[rcd_id - 1] = true;
            if(all_has_sent()){
                reset_sends();
            }
        }
        else{
            byte[4] recived_data = data;
        }
    }
    boolean all_has_sent(){
        for(byte i = 0; i < 3; i++){
            if(already_sent[i] == false) return true;
        }
        return false;
    }
    void reset_sends() {
        for(byte i = 0; i < 3; i++){
            already_sent[i] = false;
        }
    }
}
```

Figure 11: Collector of timed rebeca model

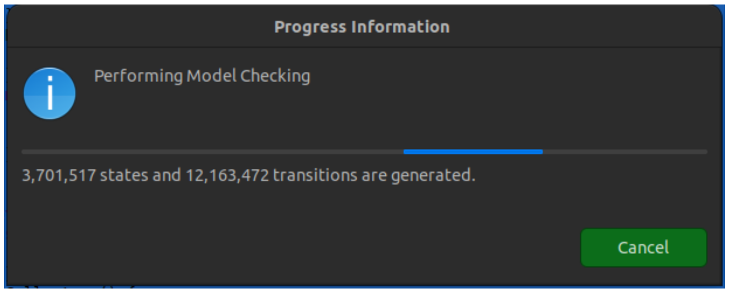
Note that if we want to increase the node for our system we should consider changing the Collector's queue and the boolean array for this(also for mehtods working with that array).
Timing would a crucial problem too, since our model might functional with 1 or 2 nodes, we might consider to change time units for "delays" and "afters" and "deadlines" as well.One strategy which will work is assuming whenever number of nodes increases Sensor's periodic behaviour(as depicted in figure 8) would increase as well.
Furthermore assume that we want to check our model's functionality with five nodes, therefore we must increase "after"(period) for sensors to 5 time units.(This strategy also reflects approaches used in real-world scenarios as well)

3.2 Analysis Results

Consider two nodes where we have nondeterminism as follow:

$$data = ?(1, 2) \tag{3}$$

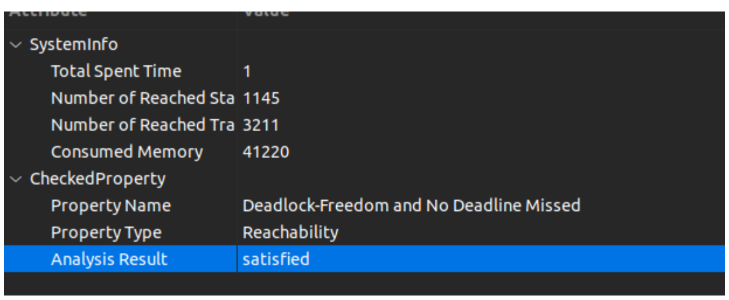
Since states will so much as you can see in fiugre 12, we decide to delete nondeterminism and set data equal to one.



3,701,517 states and 12,163,472 transitions are generated.

Figure 12: Too much states while having nondeterminism

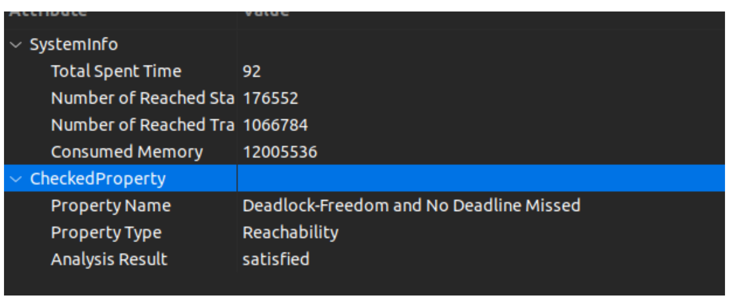
The reuslt for two nodes without nondeterminism is as depicted in figure 13: As you can see the default



```
SystemInfo
    Total Spent Time         1
    Number of Reached Sta    1145
    Number of Reached Tra    3211
    Consumed Memory          41220
CheckedProperty
    Property Name            Deadlock-Freedom and No Deadline Missed
    Property Type            Reachability
    Analysis Result          satisfied
```

Figure 13: Two nodes satisfiction

property are satisfied with 1200 states and 3210 transitions.
The result for four nodes without nondeterminism:



```
SystemInfo
    Total Spent Time         92
    Number of Reached Sta    176552
    Number of Reached Tra    1066784
    Consumed Memory          12005536
CheckedProperty
    Property Name            Deadlock-Freedom and No Deadline Missed
    Property Type            Reachability
    Analysis Result          satisfied
```

Figure 14: Model Checking four nodes

As it has been illustrated in figure 14 the reachable states are 177k and transitions are 1m.
Some scenarios cause missing datas:

- Sensor missed: Assume that massage has been transmitted to the CPU with deadline 3 therefore CPU has loaded one size of it's queue by sensor.when a counter example has been given, every queue will be followed by two main parts which are time of arrval and deadline.Any time after arriaval $+ 3$, deadline would be considered to missed.
- CPU missed: Now assume CPU has packed data and wants to transmit it to RCD.The amount of time that will accepted where deadline will not be missed is:

  $$(\text{arrival time}, \text{arrival time} + 1) \tag{4}$$

  Since the data will transmit after 1 time units and deadline is 2 from current time, it means that RCD must response to the CPU within 1 time units overall.
- RCD missed:The overall time that Collector needs to process it's queue when a RCD transmit data is:

  $$(\text{arrival time}, \text{arrival time} + 2) \tag{5}$$

  Since we send data from RCD after 1 time units and the deadline is 3 from current time, overall time for Collector needed to process it's corresponding queue is 2 time units.

Since Collector does not have any deadline or after (cause it won't send any data), we wont discusse the deadline misses for this particularl actuator.