Railway Crossing Gate Controller

Design and Model Verification

GROUP 5:

SAHIB KHAN (160101057)
SAURABH BAZARI (160101061)

PROBLEM DESCRIPTION:

Consider a railway track crossing, that is a road crosses a railway line. We have to design a railway crossing gate which is built on the road to prevent vehicles from crossing while a train is passing.

APPROACH:

We consider the following assumptions:

1. The train arrives non deterministically on the track.
2. Before the train arrives it sounds a Horn. Assume that the horn is sounded when the train is sufficiently far for the gate to close.
3. Only one train arrives at a time.
4. We can't control the train if it arrives it will not stop.
5. The vehicle will stop when the horn is sounded.

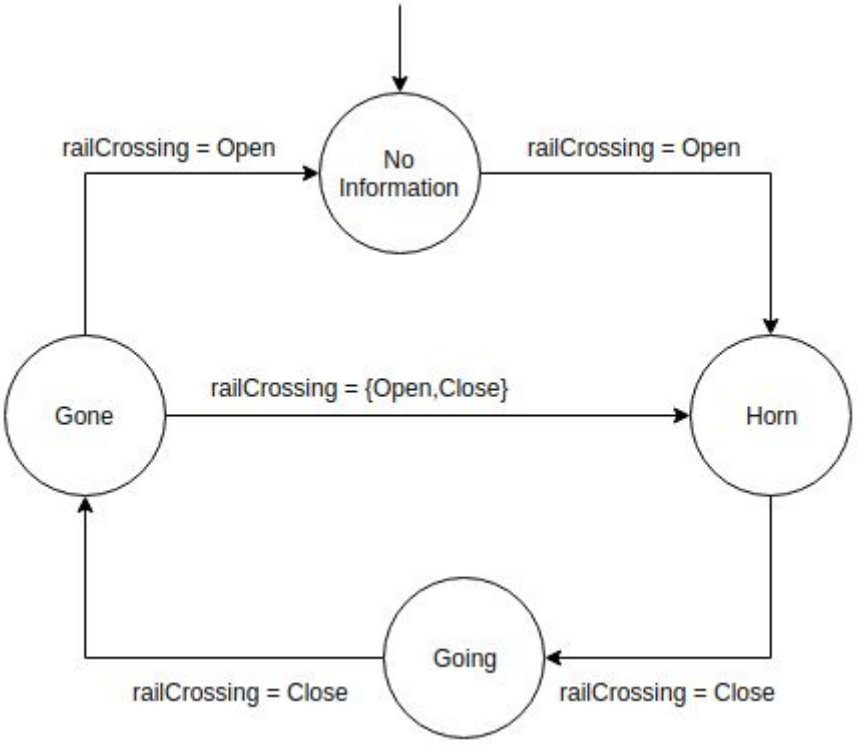
We have designed 4 modules. A module for the train, for the gate, for vehicles and main module.

Modules

1. TRAIN MODULE:

This module stores the status of the train. A variable trainStatus is maintained which can take one of the 4 values: NoInfo, Horn, going, gone.
NoInfo: This is the default status and it means a train can or cannot arrive now or later
Horn: If the horn has been sounded then this status is set. It means the train is arriving any time now.
Going: It means that the train is passing the crossing. (Note at this time the crossing gate should be closed otherwise accident may occur).
Gone: When a train has crossed the crossing, this status is set.

State Diagram



CODE FOR TRAIN MODULE

```
MODULE Train(railCrossingStatus)
VAR
    trainStatus : {noInfo , horn , going, gone};

ASSIGN
    init(trainStatus) := noInfo;

    next(trainStatus) := case
                                trainStatus = horn & railCrossingStatus = close : going;
                                trainStatus = going & railCrossingStatus = close : gone;
                                trainStatus = gone & railCrossingStatus = open : {noInfo,horn};
                                trainStatus = noInfo & railCrossingStatus = open : horn;
                                trainStatus = gone & railCrossingStatus = close : {gone,horn};
                                TRUE : trainStatus;
                            esac;
```

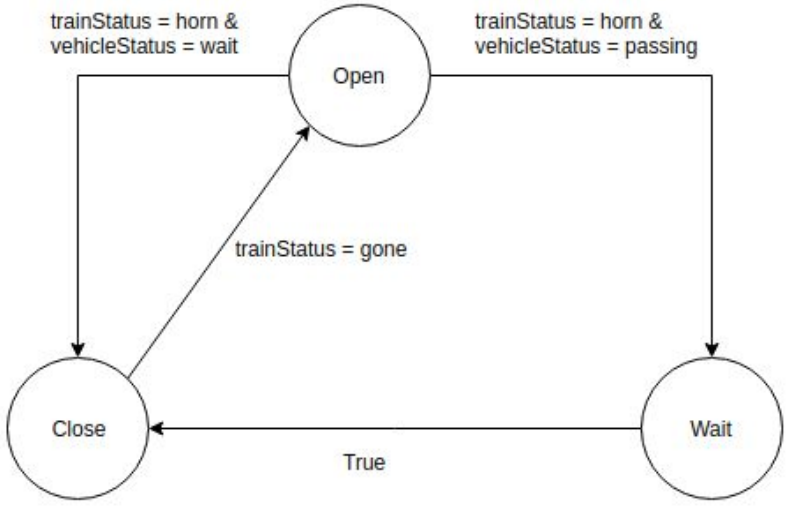
2. RAILWAY CROSSING GATE MODULE

This module stores the state of the crossing gate. The gate state can be closed, open, wait.
Closed: This state means that the gate is closed. No vehicle can pass the crossing
Open: This means that any vehicle can pass the crossing.
Wait: If a vehicle is crossing the track and the gate needs to close, then the gate must wait for the vehicle to pass before closing. This state fulfills that condition.

This module is set its state based on trainStatus and vehicleStatus

1) If the train sounded the horn and some vehicle is not passing the track then the gate is set to close.
2) If the train has sounded the horn and a vehicle is crossing the track then the gate goes to the waiting state. Essentially it is waiting for the vehicle to pass before closing.
3) If the train has gone then it can remain open or closed.
4) Finally, if none of the above conditions hold true and the status is waiting then the gate must be closed.

State Diagram



CODE FOR THE RAILWAY CROSSING GATE MODULE

```
MODULE RailwayCrossing(trainStatus,vehicleStatus)
VAR
    railCrossingStatus : {open,close,wait};

ASSIGN

    init(railCrossingStatus) := open;
    next(railCrossingStatus) := case
                                railCrossingStatus = open & trainStatus = horn & vehicleStatus != passing : close;
                                railCrossingStatus = open & trainStatus = horn & vehicleStatus = passing : wait;
                                railCrossingStatus = close & trainStatus = gone : {open,close};
                                railCrossingStatus = wait : close;
                                TRUE : railCrossingStatus;
                            esac;
```

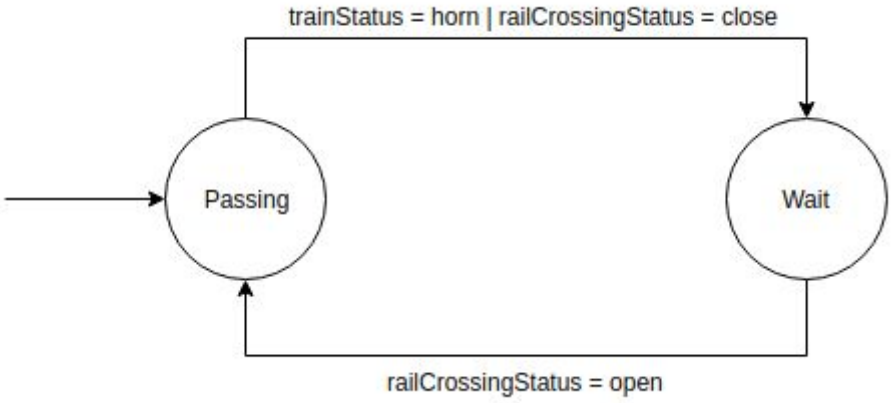
3. Vehicle Module

This module is meant for the vehicles which need to pass the crossing.
It stores the vehicleStatus that is passing or waiting.
Passing: When a vehicle is passing the crossing (Going over the railway tracks)
Waiting: The Vehicle is waiting because the gate is closed. It must wait for the gate to be open

The train status set according to the following condition:

1. If the horn is sounded or the gate is closed then the vehicle will wait.
2. If the gate is open then the vehicle will pass.

STATE DIAGRAM



CODE FOR THE VEHICLE MODULE

```
MODULE Vehicle(railCrossingStatus,trainStatus)
VAR
    vehicleStatus : {wait,passing};

ASSIGN
    init(vehicleStatus) := passing;
    next(vehicleStatus) := case
                            trainStatus = horn | railCrossingStatus = close : wait;
                            railCrossingStatus = open : passing;
                            TRUE : vehicleStatus;
                        esac;
```

4. Main Module

This module is responsible for making instances of the above modules and passing the parameters required by each other.

CODE FOR THE MAIN MODULE

```
MODULE main
VAR
    train : Train(railwayCrossing.railCrossingStatus);
    railwayCrossing : RailwayCrossing(train.trainStatus,vehicle.vehicleStatus);
    vehicle : Vehicle(railwayCrossing.railCrossingStatus,train.trainStatus);
```

Requirements

1. **Safety**: This is the most important requirement among others. Because we must ensure that the gate is closed while the train is passing because otherwise, accidents may happen. In the below spec we ensure that while a train is passing the gate should be closed to prevent cars from getting near the train.

```
SPEC AG ( train.trainStatus=going -> railwayCrossing.railCrossingStatus=close )
```

2. This spec is again ensuring **safety**. We are ensuring that a vehicle can't be passing and the train can't be going simultaneously. If this condition is violated then the code is not safe.

```
SPEC AG !( train.trainStatus=going & vehicle.vehicleStatus = passing )
```

3. Also, we must ensure that while the vehicle is passing the gate must be open. Otherwise, the passengers in the vehicle may get injured. The gate must wait for the vehicle to pass and the close.

```
SPEC AG !( vehicle.vehicleStatus = passing & railwayCrossing.railCrossingStatus=close )
```

4. **Starvation**: We must ensure that if a vehicle is waiting at the crossing it will eventually pass. It must not wait indefinitely.

```
SPEC AG ( vehicle.vehicleStatus = wait -> EF vehicle.vehicleStatus = passing  )
```

5. This is for the **safety** of the vehicle when the railway crossing gate is going to close. So, whenever train sounds horn and vehicle are passing then in all next step railway crossing gate must wait for passing current vehicles.

```
SPEC AG ( (train.trainStatus = horn & vehicle.vehicleStatus = passing) ->
AX (railwayCrossing.railCrossingStatus = wait)   )
```

6. Similarly, the gate must also open sometime in the future if it was closed.

```
SPEC AG ( railwayCrossing.railCrossingStatus=close -> EF(
railwayCrossing.railCrossingStatus = open ) )
```

7. Whenever train sounds horn and railway crossing gate is closed then the gate will remain closed until the train leaves.

```
SPEC AG ((train.trainStatus=horn & railwayCrossing.railCrossingStatus = close) -> ( A[ (
railwayCrossing.railCrossingStatus=close) U (train.trainStatus = gone)]))
```

8. This is to ensure that there is no fake horn. The train will be on track after the horn.

```
SPEC AG ( (train.trainStatus=horn) -> AX( train.trainStatus=horn | train.trainStatus=going ))
```

9. Below 2 rules are false because if the train comes continuously and the railway crossing gate will not open.

```
--  SPEC AG ( vehicle.vehicleStatus = wait -> AF vehicle.vehicleStatus = passing )

-- SPEC AG ( railwayCrossing.railCrossingStatus=close -> EF( railwayCrossing.railCrossingStatus = open ) )
```

6. Conclusion

We have designed a railway crossing gate controller. We ensure the safety of passengers at all times. Here one assumption is made that the train can't be controlled in any way. Specifications are used to verify the model, considering the safety issues. This can lead to starvation in a very special case(If the train keeps on coming). Although this doesn't happen in real life as two trains never run on the same track so close to each other. We could improve this implementation by adding light for the train (a traffic light). This way the controller will be two ways, for vehicles as well as for the train. Also doing such a thing will allow us to solve the starvation problem.