Richard Zimmerman [U16632404] | Alec Braynen [U41628244]

Two Robots Sensing One Space Ahead

Specifications

The Robot Does Not Hit Any Obstacles.

A[] (grid[robotOne.myPosX][robotOne.myPosY] != 1 or grid[robotTwo.myPosX][robotTwo.myPosY])

The array represented by the variable grid is a 2-d boolean array with the dimension of p and q where p represents the number of rows in the grid and q represents the number of columns in the grid. A status of false indicates the grid space represent by *s* and *t* in grid[*s*][*t*] indicates the grid square (*s,t*) is not blocked by a static obstacle. When grid[*s*][*t*] is true, this indicates the grid square (*s,t*) is blocked.

The statement above confirms that there is no such trace where the current position of either robot (in effect, any position the robots have traveled) is blocked by a static obstacle.

A[] (robotOne.myPosX != robotTwo.myPosX and robotOne.myPosY != robotTwo.myPosY)

The statement above ensures that the two robots never exist in the same place at once by confirming that there are no traces such that the robots x and y coordinates match.

Currently, our model does not satisfy this requirement and allows the robots to be in the same place at the same time. There are certain cases where one robot could enter a space that blocks the other robot from being able to reach the destination. To work around this issue, the robots could use a shared memory space to communicate where each robot currently is and, importantly, organize a “shuffle” operation. This “shuffle” operation could be implemented by allowing the blocking robot to move back a space at a time until the blocked robot was able to move a space that was not immediately blocked by the other robot.

The Robot Does Not Cross any Boundaries.

A[] robotOne.myPosX <= 0 and robotOne.myPosX < dimX

A[] robotTwo.myPosX <= 0 and robotTwo.myPosX < dimX

A[] robotOne.myPosY <= 0 and robotOne.myPosY < dimX

A[] robotTwo.myPosY <= 0 and robotTwo.myPosY < dimX

In the above statement, the robot’s local variables, myPosX and myPosY, that represent the current X and Y coordinates of the robot on the grid (respectively) are validated so that the integer value of X or Y is never greater than

In the above statement, robotOne (the only robot) has local variables myPosX and myPosY that represent the current X and Y coordinates of the robot on the grid, respectively.

The Robot Will Eventually Reach the Destination.

A<> (robotOne.Complete and robotTwo.Complete)

The above statement confirms that for every trace, robotOne will reach the Complete state which is only entered if the robot current position (myPosX and myPosY) matches the coordinates of the destination point.

Model Summary

Grid Controller

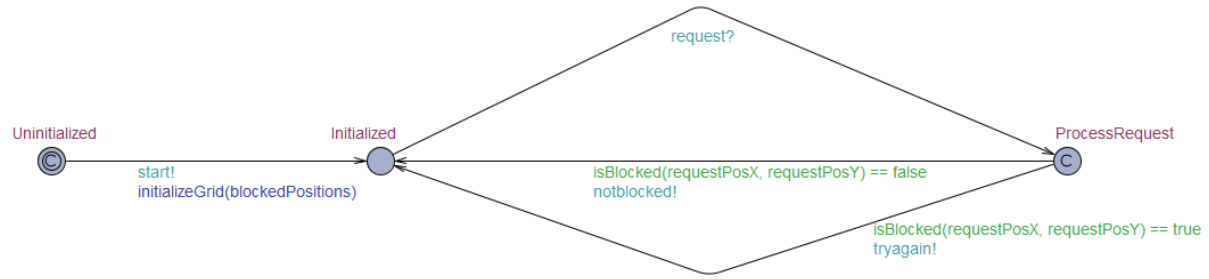

Figure 1 Model of Grid Controller in UPPAAL

The grid controller starts in a committed state named Uninitialized which, upon system start, is immediately exited and the transition from state Uninitialized to state Initialized is taken. This transition raises a signal on a broadcast channel named *start* and then calls a function that initializes the grid array (the 2d boolean array named *grid* mentioned in *Specifications -> The robot does not hit any obstacles*) which makes sure that the grid's static obstacles are enabled/instantiated.

The grid controller then waits in the Initialized state until it receives a signal on the *request* channel. Upon receiving the request signal, the grid controller enters the committed state ProcessReqeust and in the next "step" of the system, raises a signal on either the *notblocked* or *tryagain* channel which represents whether the grid targeted by the requester is not blocked or blocked, respectively.

Because the Processing state is committed, the grid controller must respond to an active request before any robot in the system is allowed to raise a second request. Effectively, this prevents the robots from moving at the same time to the same space.

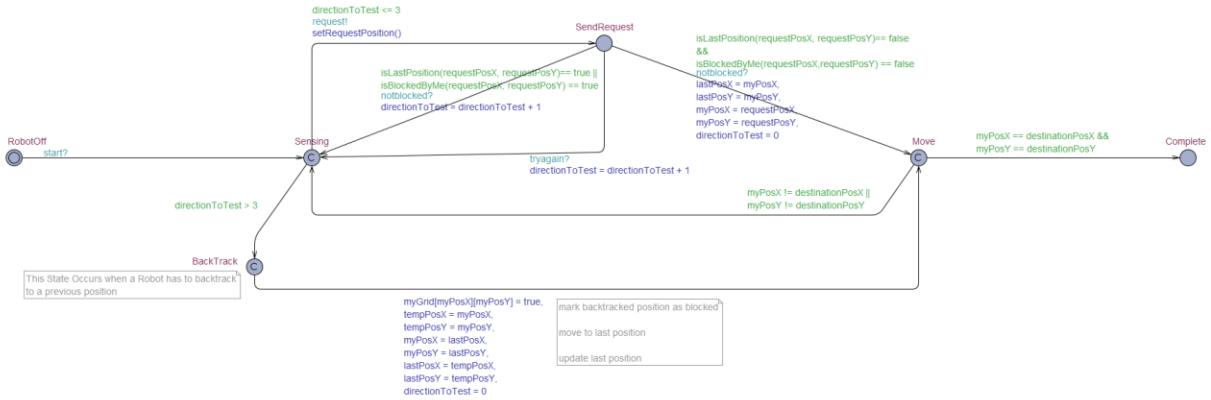

Figure 2 Model of RobotOne

The robot "starts" in the RobotOff state but enters the Sensing state when the *start* signal is received from the controller.

Unless the robot has tested every possible move, the robot transitions from Sensing to SendRequest. This transition raises the request signal to the grid controller and sets the global variables that the grid controller will refence representing the target grid-square the controller is testing for blocked-status.

If the robot receives the *notblocked* signal from the grid controller for a target position and the target position is also not the last position the robot was in, the robot transitions from SendRequest to Move. From the Move state, the robot moves to Complete if new position is the destination or moves back to Sensing if the robot is not current at the robot's destination. The committed nature of the Move state ensures the robot dos not enter a non-deterministic state.

If the robot receives the *tryagain* signal from the grid controller, the robot increments its internal compass which is used to determine which direction from the robot's current position to test for valid moves next.

In the case that the robot receives a *notblocked* signal for a target grid-square, but the target square was the previous position of the robot, the robot will keep testing other possible moves.

After the robot has exhausted all available moves, represented by the directiosnToTest variable being greater than 3 (internal, 4-direction compass), the robot will enter the Backtrack state, transition to Move, then back to Sensing. As the robot's state crosses these 3 edges, the robot moves backwards and marks the space that the robot is currently leaving as blocked so it will not return to this dead-end.