COM3028 Systems Verification

River Phillips

May 10, 2020

1. $[|AFAG(p \vee q)|]|_{M1}$

   $SAT_{AF}(AG(p \vee q))$

   $SAT_{AF}(SAT_{AG}(p \vee q))$

   $SAT_{AF}(SAT_{AG}(SAT(p) \cup SAT(q)))$

   $SAT_{AF}(SAT_{AG}(\{S_0, S_2\} \cup \{S_3\}))$

   $SAT_{AF}(SAT_{AG}(\{S_0, S_2, S_3\}))$

   $SAT_{AF}(\neg SAT_{EF}(\neg\{S_0, S_2, S_3\}))$

   $SAT_{AF}(\neg SAT_{EF}(\{S_1\}))$

   $SAT_{AF}(\neg SAT_{E}(\top \cup \{S_1\}))$

   $SAT_{AF}(\neg\{S_0, S_1, S_2, S_3\})$

   $SAT_{AF}(\emptyset)$

   $\bot$

   This model does not satisfy $AFAG(p \vee q)$

2. For the NuSMV solution with 5 robots and 1 food source the robots have 4 possible states resting, random walk, moving towards food and scanning area. The transitions between some of the states are non deterministic, for example resting always leads to random walk, but random walk may transition to resting or moving towards food, when the state is updated so is the previous state variable.

   The variable food found is set to true when the robot transition from moving towards food, into moving towards food. When food found is true in any of the robots the food module marks itself as found in it's own internal state.

The second NuSMV solution with 3 robots and 2 food sources required the addition of a controller for the food sources, this controller randomly selected the next food source that would be found, the same criteria were required for it to be found, a robot has to be in the moving towards food state and transition into that state again.

In both these solutions when one food source was found all of the other robots were aware that the food source has been found.

NuSMV made this implementation difficult to adapt to the two situations as it is not possible to pass in a array of Modules as arguments to other Modules, this means each situation needs a different constructor, with different parameters for each module, despite having very similar logic, in addition to this a controller was required to coordinate the next food source to be found.

3. (a) This Dafny code iterates through a list and ensures that each item is not equal to the item in position 0

   The loop invariant ensures that all items in a position greater than 0 and less than the current index do not equal the item in position 0.

   (b) This Dafny code iterates through a list and ensures that the current item does not appear in the subsequent indexes in the list.

   The outer loop invariant ensures that all items less than the current index do not have a duplicate anywhere in the list.

   The inner loop invariant ensures that there are no duplicates of the current item in a position greater than the outer loop index and less than the inner loop index.

   (c) This method is very similar to the answer to (b) it iterates though the column in position 0 of the array in the same way, ensuring that each item does not have a duplicate in a position greater that it's own in the list.

   The outer loop invariant ensures that all items less than the current index do not have a duplicate anywhere else in the list.

   The inner loop invariant ensures there are no duplicates of the current item in a position greater than the current item and less than the inner loop index.

   (d) These methods contains 3 nested loops, the first iterates through either the row or column, for each column or row it checks that each item is not repeated in a higher position in the row or column.

The inner two loops use the same loop invariants as described in (c), the outermost loops ensures that there are no duplicated values in the rows or columns that have already been checked.

(e) This method performs a logical and with the result of CheckCols and CheckRows, this ensures a value is not duplicated in a given row or column for the entire matrix.