COL750 : Foundations of Automatic Verification - Assignment

Aneeket Yadav

March 2025

1 Finding a Safe Path - SAT encoding in CNF

Variable Representation

Each position (i, j) at step t is represented by a Boolean variable $X_{t,i,j}$. If $X_{t,i,j} = 1$, the agent is at position (i, j) at step t.

Start and Goal Constraints

- The agent must start at $(1, 1)$ at step 0:
$$X_{0,1,1}$$
- The agent must reach (N, N) at the last step:
$$X_{\text{path\_length},N,N}$$

Unique Position per Step

At each step t, the agent can be in at most one position. This is encoded as:

$$\neg X_{t,i_1,j_1} \vee \neg X_{t,i_2,j_2}, \quad \forall (i_1, j_1) \neq (i_2, j_2)$$

Movement Constraints

The agent can move only to the right ($\rightarrow$) or down ($\downarrow$). If the agent is at (i, j) at step t, it must move to $(i+1, j)$ or $(i, j+1)$ at step $t+1$:

$$X_{t,i,j} \Rightarrow X_{t+1,i,j+1} \vee X_{t+1,i+1,j}$$

which in CNF form is:

$$\neg X_{t,i,j} \vee X_{t+1,i,j+1} \vee X_{t+1,i+1,j}$$

If no valid moves exist, the position must not be occupied:

$$\neg X_{t,i,j}$$

Obstacle Constraints

For every obstacle position (i, j), the agent can never occupy that position at any step:

$$\neg X_{t,i,j}, \quad \forall t$$

2 SMT Constraints

Variable Representation

We define integer variables for the agent's position at each step t:

$$x_t, y_t \quad \forall t \in \{0, 1, \ldots, \text{path\_length}\}$$

where x_t and y_t represent the row and column positions at step t.

Start and Goal Constraints

- The agent starts at $(1, 1)$:

$$x_0 = 1, \quad y_0 = 1$$

- The agent must reach (N, N) at the last step:

$$x_{\text{path\_length}} = N, \quad y_{\text{path\_length}} = N$$

Movement Constraints

The agent can move only right ($\rightarrow$) or down ($\downarrow$), which is encoded as:

$$(x_t = x_{t-1} \land y_t = y_{t-1} + 1) \quad \text{(Move Right)}$$
$$(x_t = x_{t-1} + 1 \land y_t = y_{t-1}) \quad \text{(Move Down)}$$

Thus, at each step t:

$$(x_t = x_{t-1} \land y_t = y_{t-1} + 1) \lor (x_t = x_{t-1} + 1 \land y_t = y_{t-1})$$

Step Progression Constraint

To ensure that the steps progress correctly and follow a valid path, we enforce:

$$x_t + y_t = t + 2, \quad \forall t$$

This guarantees that the path follows a strictly increasing order in terms of row and column sum.

Obstacle Constraints

For each obstacle position (i, j), the agent cannot be there at any step:

$$\neg(x_t = i \land y_t = j), \quad \forall t$$

Grid Bounds Constraints

The agent must always stay within the grid:

$$1 \leq x_t \leq N, \quad 1 \leq y_t \leq N, \quad \forall t$$

3 Description of NuSMV file:

The 4-way traffic intersection system has been modelled as follows-

- To model the state of the traffic lights. It takes 4 values. When `phase = NS_green` or `phase = NS_yellow`, it is implicit that `WE_red` would also hold and vice versa.

```
phase : {NS_GREEN, NS_YELLOW, WE_GREEN, WE_YELLOW};
```

- The phase changes in a cyclic order according to a fixed timer.

```
  -- Traffic light phase transitions
  next(phase) := case
    phase = NS_GREEN & timer = 0 : NS_YELLOW;
    phase = NS_YELLOW & timer = 0 : WE_GREEN;
    phase = WE_GREEN & timer = 0 : WE_YELLOW;
    phase = WE_YELLOW & timer = 0 : NS_GREEN;
    TRUE : phase;
  esac;

-- Timer countdown logic
next(timer) := case
(phase = NS_GREEN | phase = WE_GREEN) :
```

```
  (timer > 0) ? (timer - 1) : 1;
(phase = NS_YELLOW | phase = WE_YELLOW) :
  (timer > 0) ? (timer - 1) : 3;
TRUE : timer;
esac;
```

- For each of the 4 roads, traffic has been modelled as a queue of capacity almost `N`, which is a user input. Any element of the queue i.e the car in that spot can have one of three states - `waiting, passing or cleared`.

```
north_queue : array 0..N-1 of {waiting, passing, cleared};
south_queue : array 0..N-1 of {waiting, passing, cleared};
west_queue : array 0..N-1 of {waiting, passing, cleared};
east_queue : array 0..N-1 of {waiting, passing, cleared};
```

- If the car at the head of the queue is waiting, and its signal is green, it passes. If it was passing in the earlier cycle, it is now cleared. If a car is cleared and the car behind it is not, then that car moves one step ahead in the queue. If a car is waiting. If the tail is empty, a new car can non-deterministically be added.

```
        # First position
  next({direction}_queue[0]) := case
    {direction}_queue[0] = waiting & {green_cond} : passing;
    {direction}_queue[0] = passing : cleared;
    {direction}_queue[0] = cleared & {direction}_queue[1] != cleared : {direction
        }_queue[1];
    TRUE : {direction}_queue[0];
  esac;

        # Middle positions
  next({direction}_queue[{i}]) := case
    {direction}_queue[{i}] = cleared & {direction}_queue[{i+1}] != cleared : {
        direction}_queue[{i+1}];
    TRUE : {direction}_queue[{i}];
  esac;

        # Last position (if N > 1)
  next({direction}_queue[{N-1}]) := case
    {direction}_queue[{N-1}] = cleared : {{cleared, waiting}};
    TRUE : {direction}_queue[{N-1}];
  esac;
```

- Properties checked : Self-explanatory-

```

-- Liveness: Every car in every queue eventually clears
-- 1. Basic Liveness: Every car eventually clears
CTLSPEC AG (AF (north_queue[0] = cleared));
CTLSPEC AG (AF (south_queue[0] = cleared));
CTLSPEC AG (AF (west_queue[0] = cleared));
CTLSPEC AG (AF (east_queue[0] = cleared));

-- 2. Queue Progression: If a car is waiting behind cleared spaces, it moves
    forward
--    (For position 1: if position 0 is cleared, car should eventually move to
    position 0)
CTLSPEC AG (north_queue[1] = waiting & north_queue[0] = cleared -> AF north_queue
    [0] = passing);
CTLSPEC AG (south_queue[1] = waiting & south_queue[0] = cleared -> AF south_queue
    [0] = passing);
CTLSPEC AG (west_queue[1] = waiting & west_queue[0] = cleared -> AF west_queue[0]
     = passing);
```

```
CTLSPEC AG (east_queue[1] = waiting & east_queue[0] = cleared -> AF east_queue[0]
    = passing);

-- 3. FIFO Order Preservation: A car cannot pass while cars ahead are waiting
CTLSPEC AG !(north_queue[1] = passing & north_queue[0] = waiting);
CTLSPEC AG !(south_queue[1] = passing & south_queue[0] = waiting);
CTLSPEC AG !(west_queue[1] = passing & west_queue[0] = waiting);
CTLSPEC AG !(east_queue[1] = passing & east_queue[0] = waiting);

-- 4. No Blocking: New cars can always enter the queue if there is space
CTLSPEC AG (north_queue[{N-1}] = cleared -> EX north_queue[{N-1}] = waiting);
CTLSPEC AG (south_queue[{N-1}] = cleared -> EX south_queue[{N-1}] = waiting);
CTLSPEC AG (west_queue[{N-1}] = cleared -> EX west_queue[{N-1}] = waiting);
CTLSPEC AG (east_queue[{N-1}] = cleared -> EX east_queue[{N-1}] = waiting);

-- 5. Fairness: Traffic lights alternate infinitely
CTLSPEC AG (AF phase = NS_GREEN) & AG (AF phase = WE_GREEN);

    # Fairness and safety
    passing_north = " | ".join([f"north_queue[{i}] = passing | south_queue[{i}] =
        passing" for i in range(N)])
    passing_west = " | ".join([f"west_queue[{i}] = passing | east_queue[{i}] =
       passing" for i in range(N)])

-- 6. Safety: No simultaneous passing from perpendicular directions
DEFINE
  passing_north := ({passing_north});
  passing_west := ({passing_west});
CTLSPEC AG !(passing_north & passing_west);
```

References

[1] PySAT Documentation. https://pysat.readthedocs.io/en/latest/tutorial.html

[2] NuSMV Tutorial. https://nusmv.fbk.eu/tutorial/v26/tutorial.pdf

[3] Z3Py Guide. https://ericpony.github.io/z3py-tutorial/guide-examples.htm