University of Glasgow

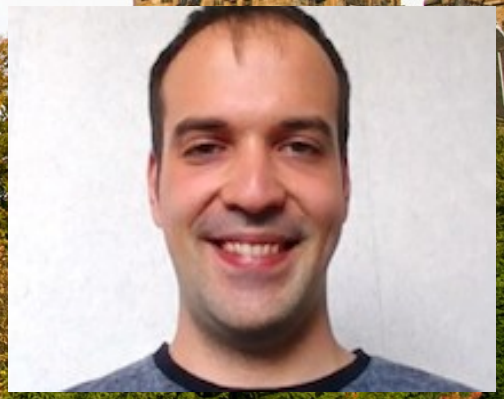
Simulation and Model Checking for Close to Real-time Overtaking Planning

Daumantas Pagojus[1], Alice Miller[1], Bernd Porr[2] and Ivaylo Vaklov[1]

[1]School of Computing Science

[2]School of Biomedical Engineering



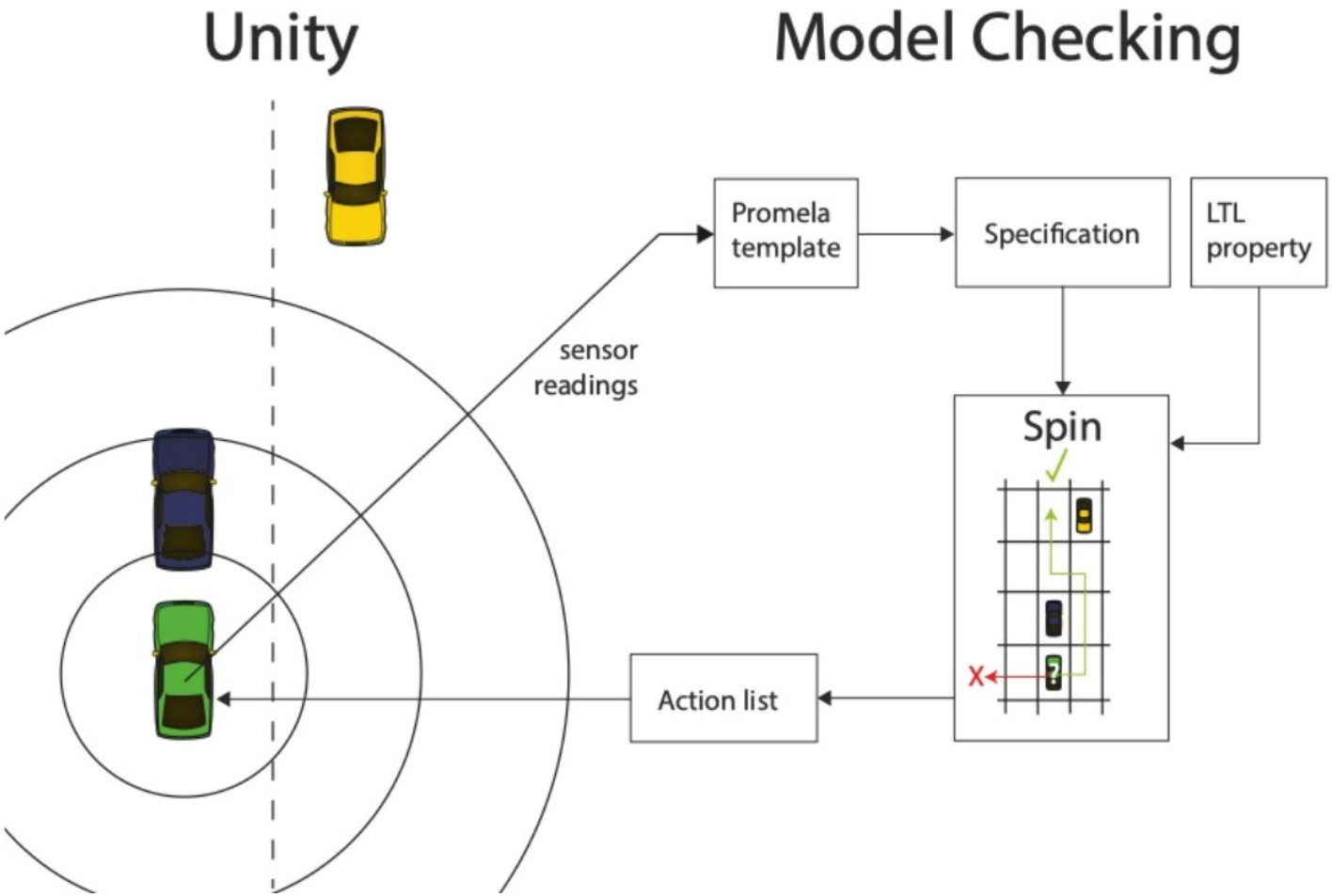
Outline

- Overview of approach
- Video
- Promela and Spin
- System and Promela model
- Unity
- Combining Unity simulation and Spin
- Results + discussion
- Conclusion

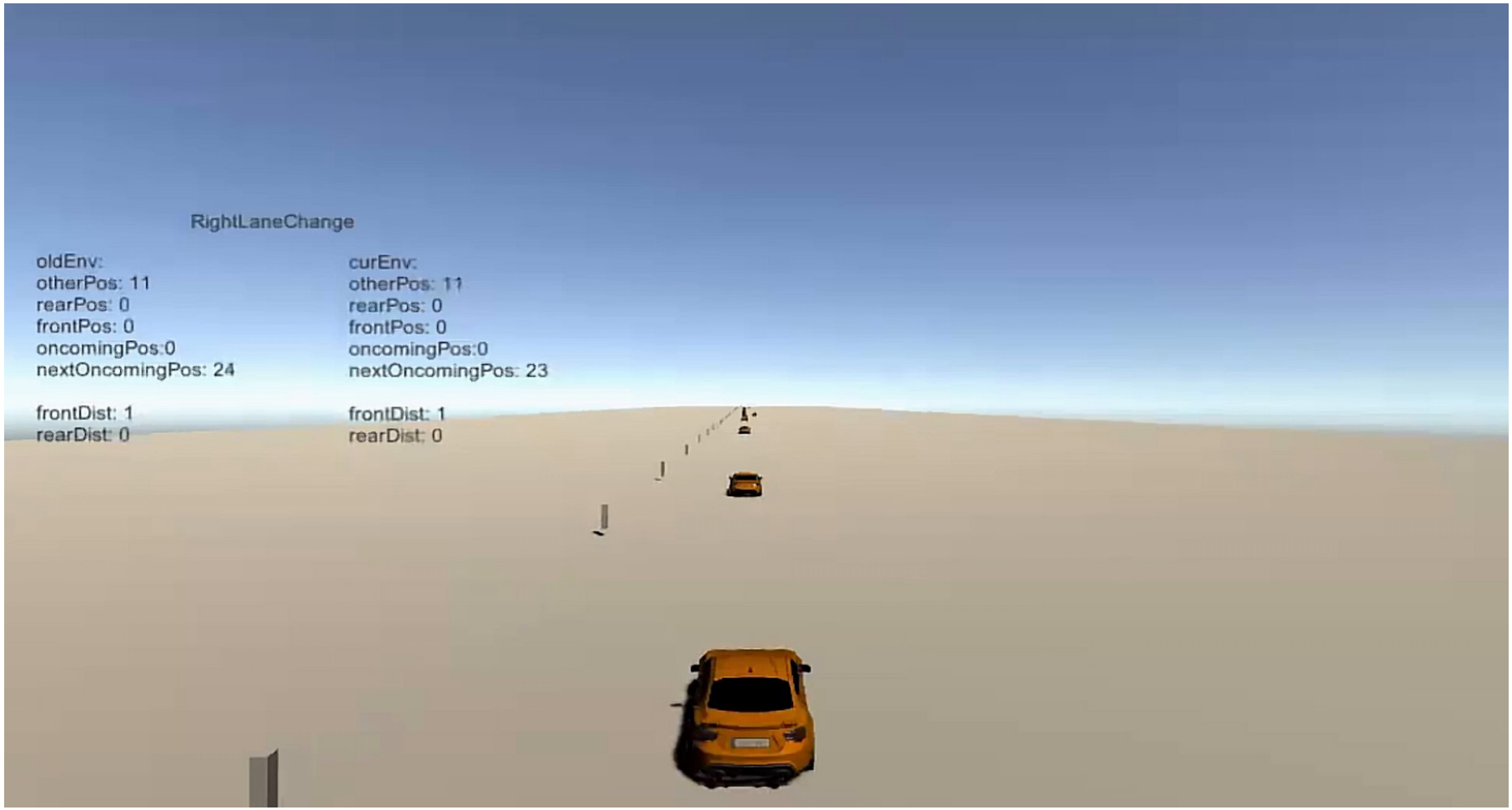
Thanks to: UKRI Strategic Priorities Fund to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020-2024).

EPSRC Science of Sensor Systems Software grant (EP/N007565/1, 2016-2022

Overview


Unity
Model Checking
Promela template
Specification
LTL property
sensor readings
Spin
Action list
X

Video


RightLaneChange
oldEnv:
otherPos: 11
rearPos: 0
frontPos: 0
oncomingPos:0
nextOncomingPos: 24
frontDist: 1
rearDist: 0
curEnv:
otherPos: 11
rearPos: 0
frontPos: 0
oncomingPos:0
nextOncomingPos: 23
frontDist: 1
rearDist: 0

Promela and Spin

- Promela - *pro*cess *me*ta *la*nguage

- Spin – tool for analysing Promela programs by
  - Simulation (random, interactive, or guided)
  - verification of state space
    - check for deadlock/unexecutable code/non-progress
    - check for truth (or otherwise) of Linear time temporal logic (LTL) properties

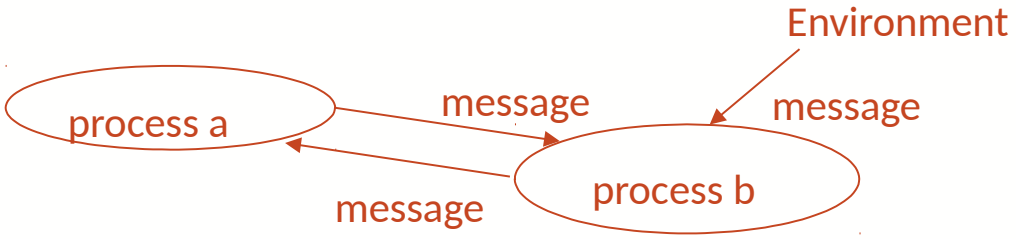
Promela Programs

- Consist of
  - message channels
  - processes (global) which specify *behaviour*
    - variables
  - channels and variables are global *or* local and only updated by processes.

- Our models are simple though – only consist of single process+ a process representing a linear time temporal logic property

- Main feature we exploit is non-determinism (via if…fi and do…od loops)

  - Allows our autonomous vehicle to make its own decisions

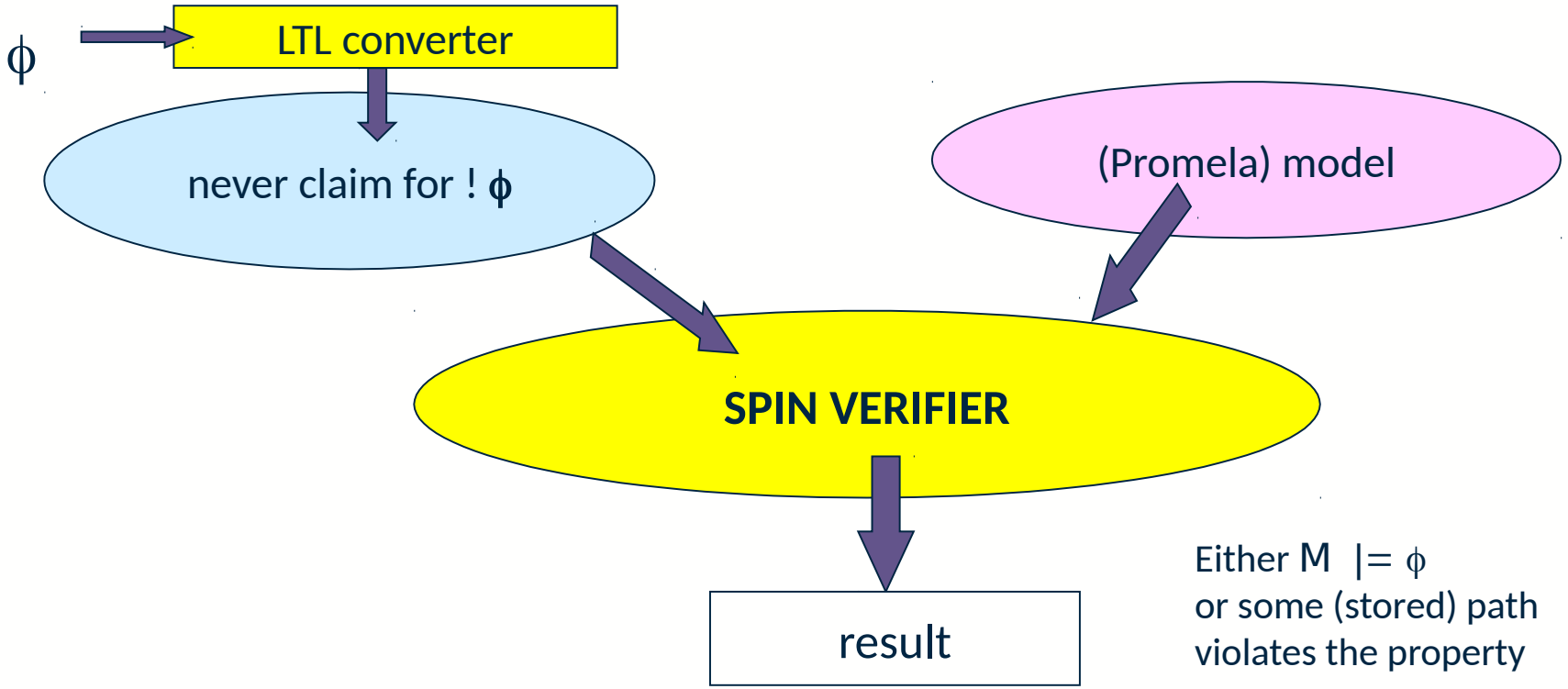


Proving properties with SPIN

Suppose we have a model M and an LTL property ϕ and we want to show that M satisfies ϕ (ie ϕ is true on all paths of M) We want to show that

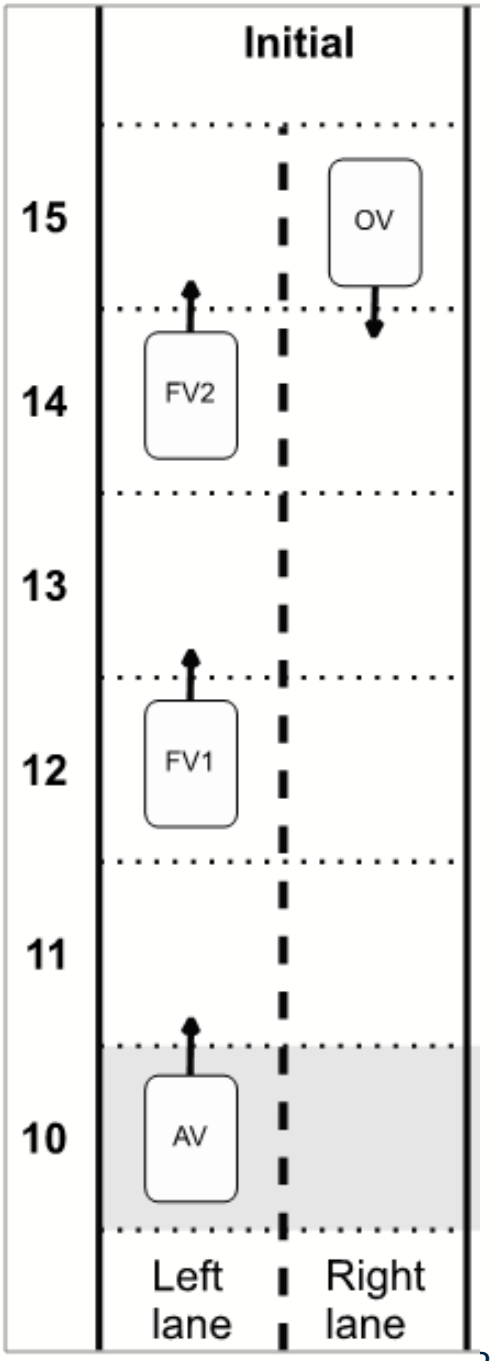
M $\models \phi$ (M models ϕ – i.e. ϕ holds for all paths of M)

We provide SPIN with a (Promela specification of) model M and ϕ and SPIN does the rest

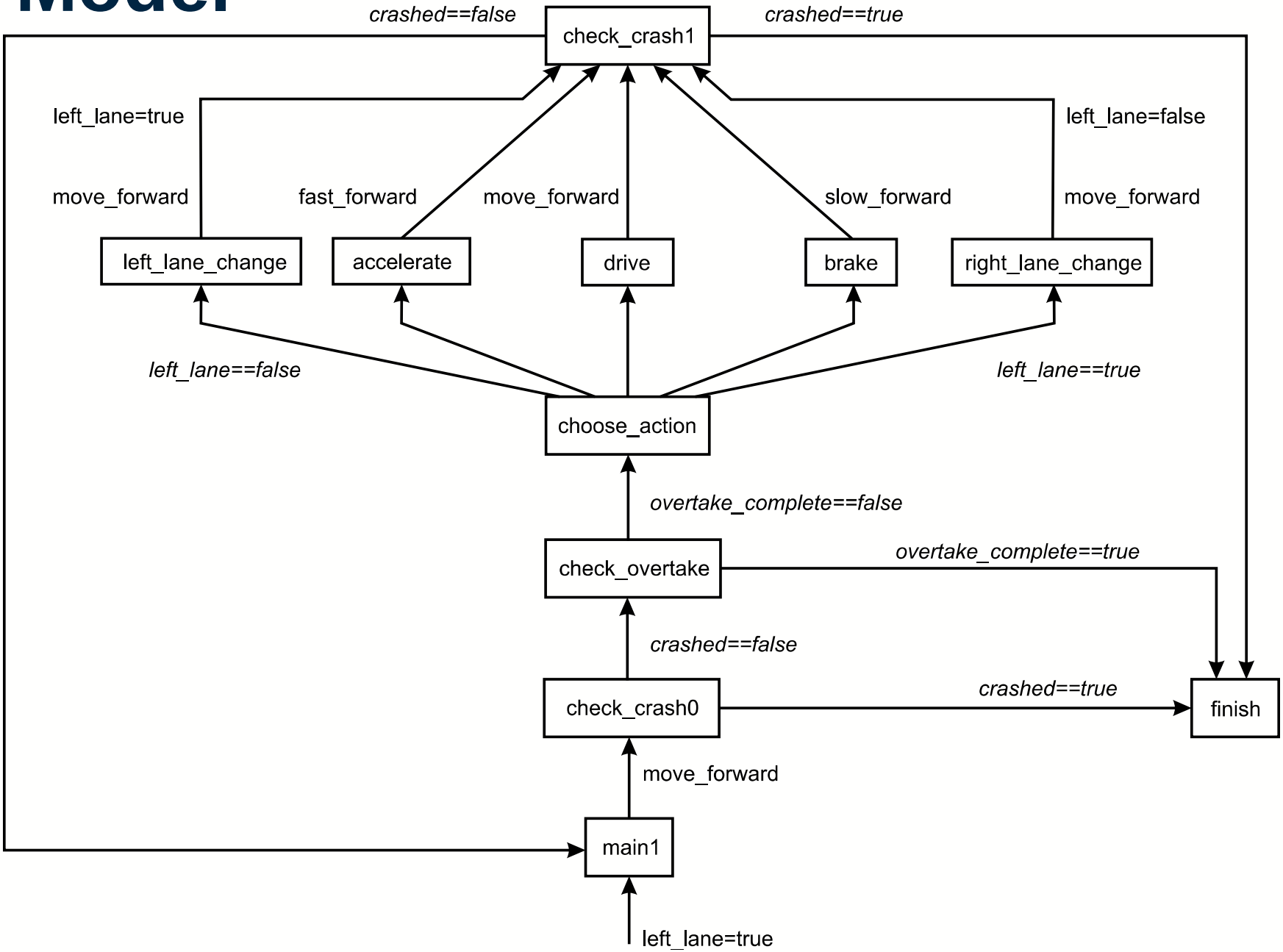


Our system

- Consists of autonomous vehicle (AV), other vehicles moving in same direction (FV_1, FV_2, etc.) and an oncoming vehicle OV
- All vehicles move with respect to AV (see later slide)
- Our single process is AV, which knows where other vehicles are relative to itself. AV based on transition system on next slide…
- Other assumptions are made – introduced later.



Model


crashed==false
check_crash1
crashed==true
left_lane=true
left_lane=false
move_forward
fast_forward
move_forward
slow_forward
move_forward
left_lane_change
accelerate
drive
brake
right_lane_change
left_lane==false
left_lane==true
choose_action
overtake_complete==false
check_overtake
overtake_complete==true
crashed==false
check_crash0
crashed==true
finish
move_forward
main1
left_lane=true

Actions + conditions

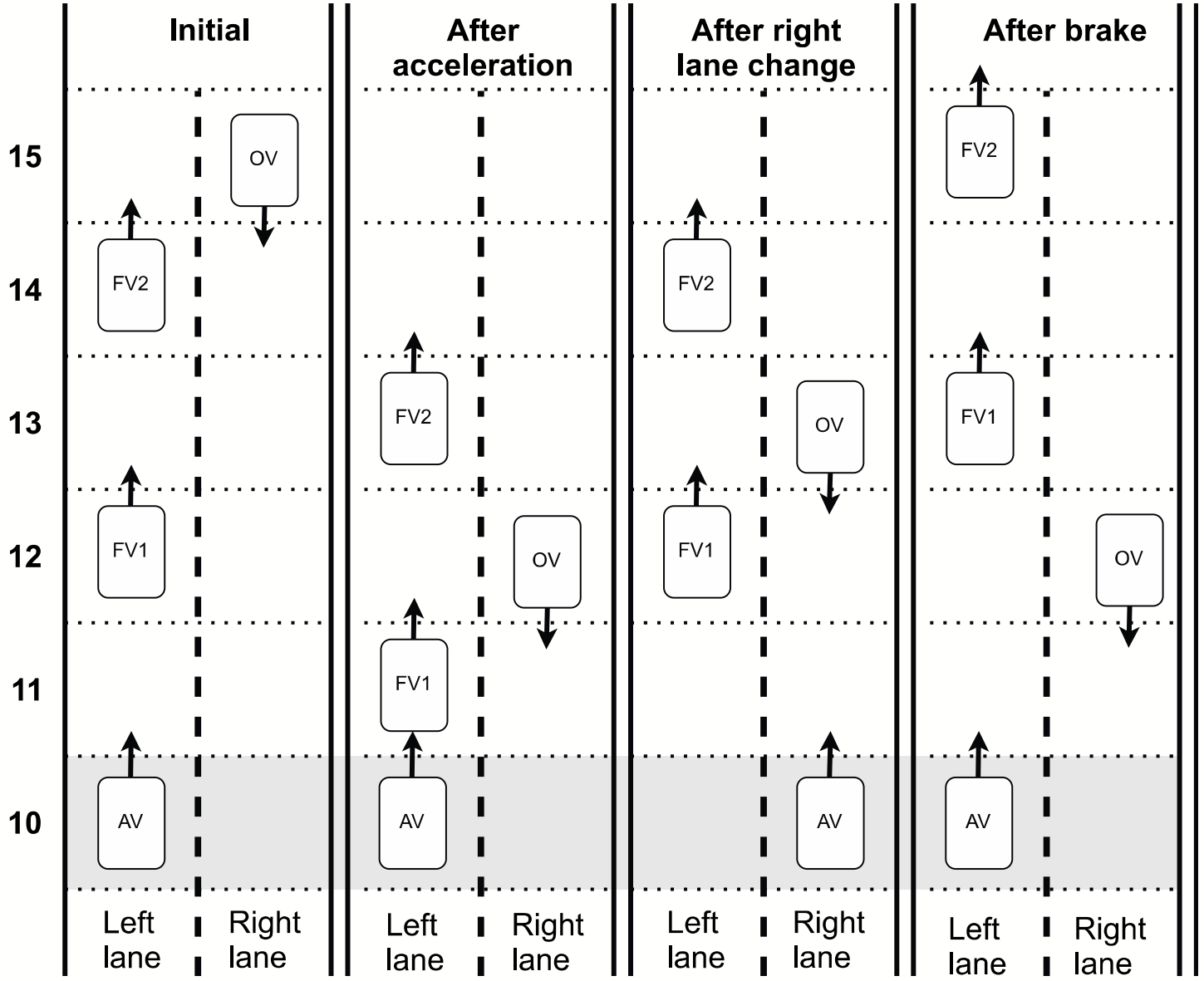
actions

Left-lane-change	accelerate	drive	brake	Right-lane-change
Move into left lane and move forward 1 position/ timestep	Move forward 2 positions/ timestep	Move forward 1 position/ timestep	Move forward at 1 position/2 timestep2	Move into right lane and move forward 1 position/time step

conditions

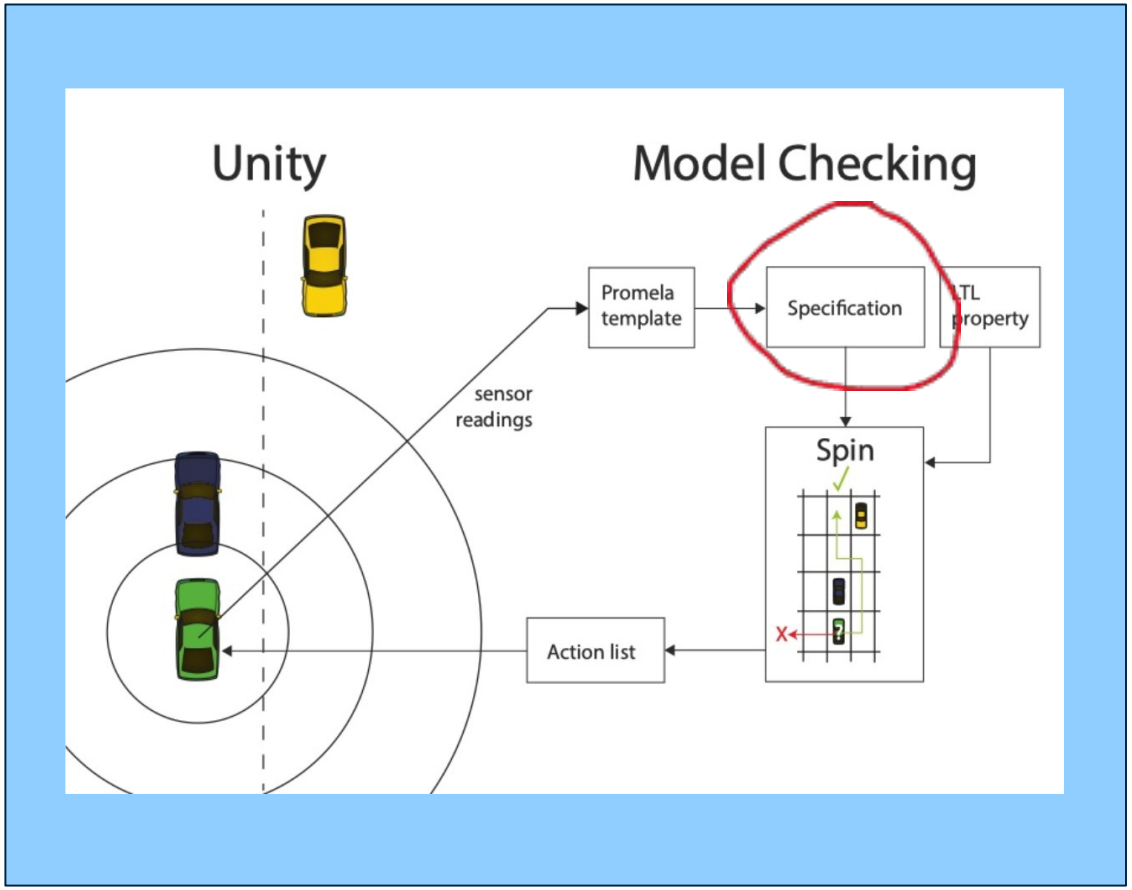
left_lane==true	*crashed==true*	*overtake_complete==true*
AV is in the left hand lane	AV has crashed into OV or FV	AV has overtaken FV without crashing

Lanes


Initial
After acceleration
After right lane change
After brake
15
14
13
12
11
10
OV
FV2
FV1
AV
Left lane
Right lane

Promela model

specification created from template and initial conditions (provided by simulator)



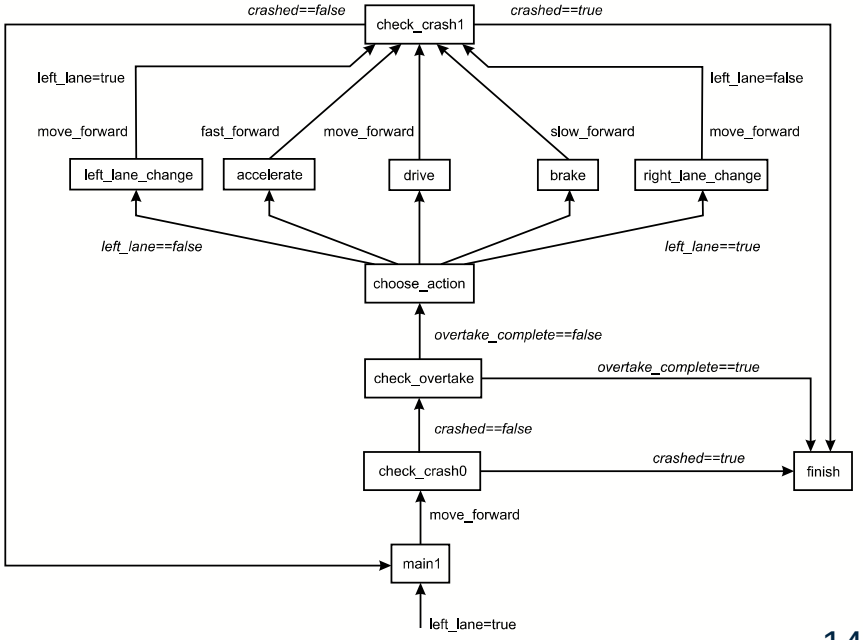
Promela extract 1 - preamble

```
// variable declarations and initialisations (omitted)
#define q (overtaken == true) //define proposition
#include “nc_overtake.pml” //include never claim

//inline functions, e.g.
inline choose_action()
{
 abs(pos_other, pos_autonomous, distance_other);
 do
 :: (lane != left && numLaneChanges < numAllowedLaneChanges) ->
     goto left_lane_change;
 :: goto accelerate;
 :: (lane != right && numLaneChanges < numAllowedLaneChanges) ->
    goto right_lane_change;
 :: goto drive;
 :: goto brake;
 od;
}
```

Promela extract 2 – AV process

```
Proctype mainProc()
{main1: //label – yes we use gotos!
 //call some inlines and goto other labels

 accelerate: //ditto


 left_lane_change: //ditto


 (etc.)



}
```

Labels correspond to state names in the transition system



Our LTL property

- Want to find a path in which a successful overtake occurs

- All LTL properties implicitly hold “for all paths”

  so:

  [ ] p means: “for all paths, p is always true”

  <> p means: “for all paths, p is eventually true”

- Can’t explicitly say “for some path r is true” – but we can catch such a path as a violation of the property

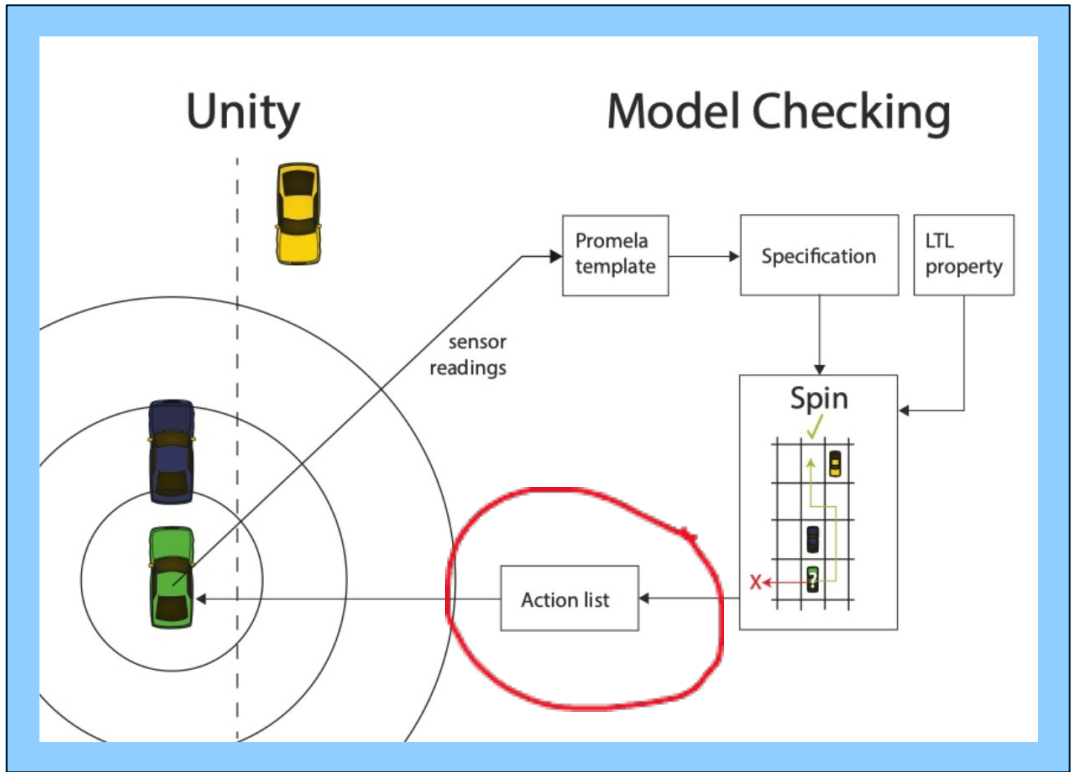
  [ ] ! r

- So our property is

  [ ] ! (successful overtake) - error path is what we want!

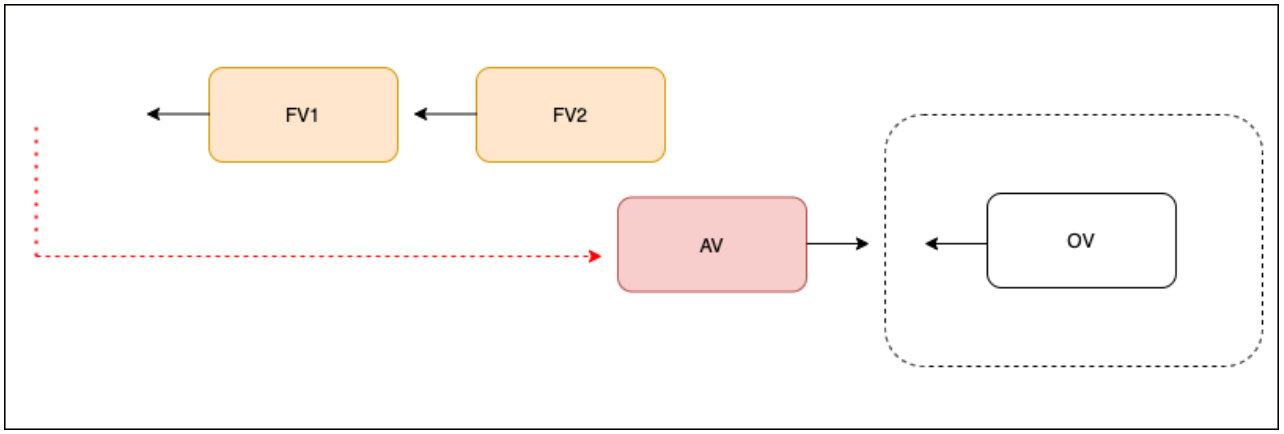
Guided simulation to instruction set

- When a path violating our property is found, Spin stores it.
- Can run a “guided simulation” to reproduce path
- By annotating our Promela code with print statements, can embed the actions taken into the guided simulation
- Extract list of actions to provide instructions for a successful overtake.



Two models, and why

- Coarse discrete representation of space can lead to unsafe behaviour
- For AV to judge if it is safe distance from OV need to err on side of caution
- Add “danger zone” area around OV – `final_model`
- However – can lead to AV unable to get itself out of trouble (two identified scenarios)
  - revert to one of two versions of a different model `preparations_model`, to recover safe position.



Scenario 1: AV thinks it can’t get back into own lane (it can)

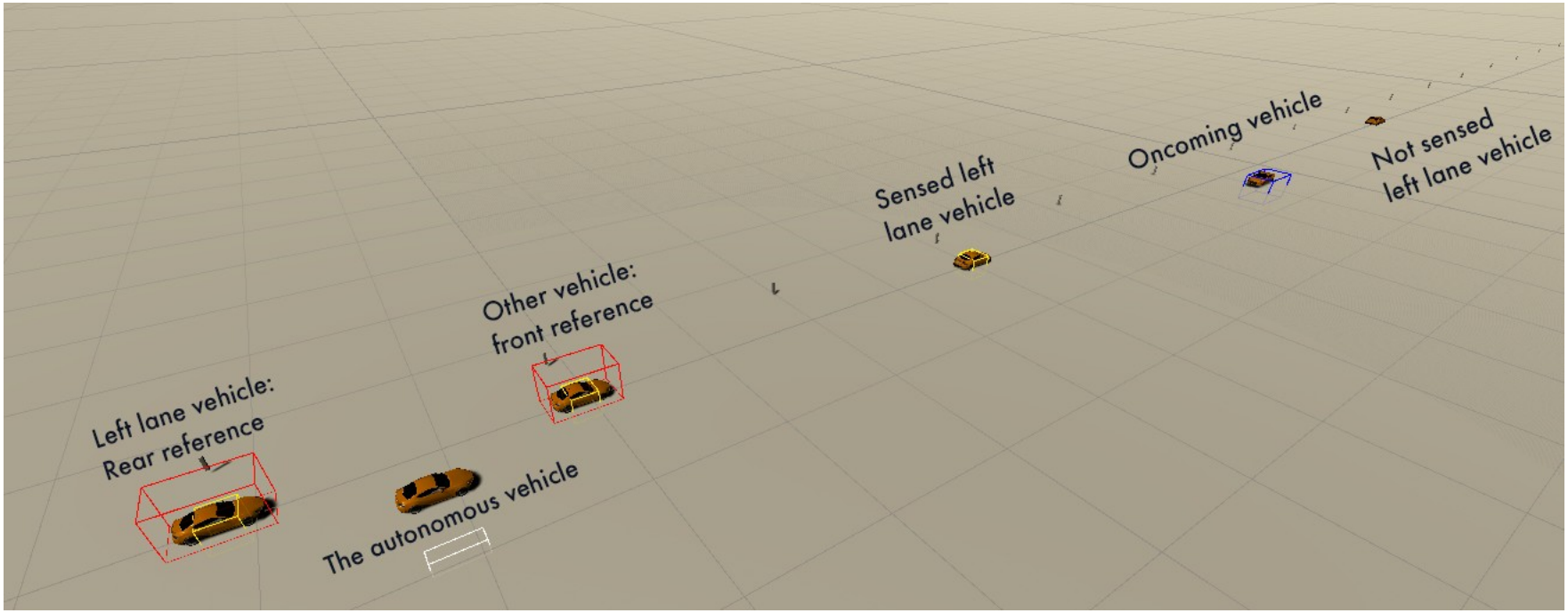
Unity 1

- Vehicle model from Unity Asset store
- Affected by unity physics engine
- Can be controlled by adjusting throttle, brake and steering input
  - we adjust those inputs
- Max speed of AV is
  - 24.2 km/h in `Drive` state
  - up to 50.4 km/h in `Accelerate` state and
  - up to 7km/h in `Brake` state

- Road divided into 21m segments
- Manoeuvres take up to 3 secs in simulator, driving at normal speed will cover 21m (one second) in 3 secs
  - Makes it easier to match simulator model to Promela models

Unity 2

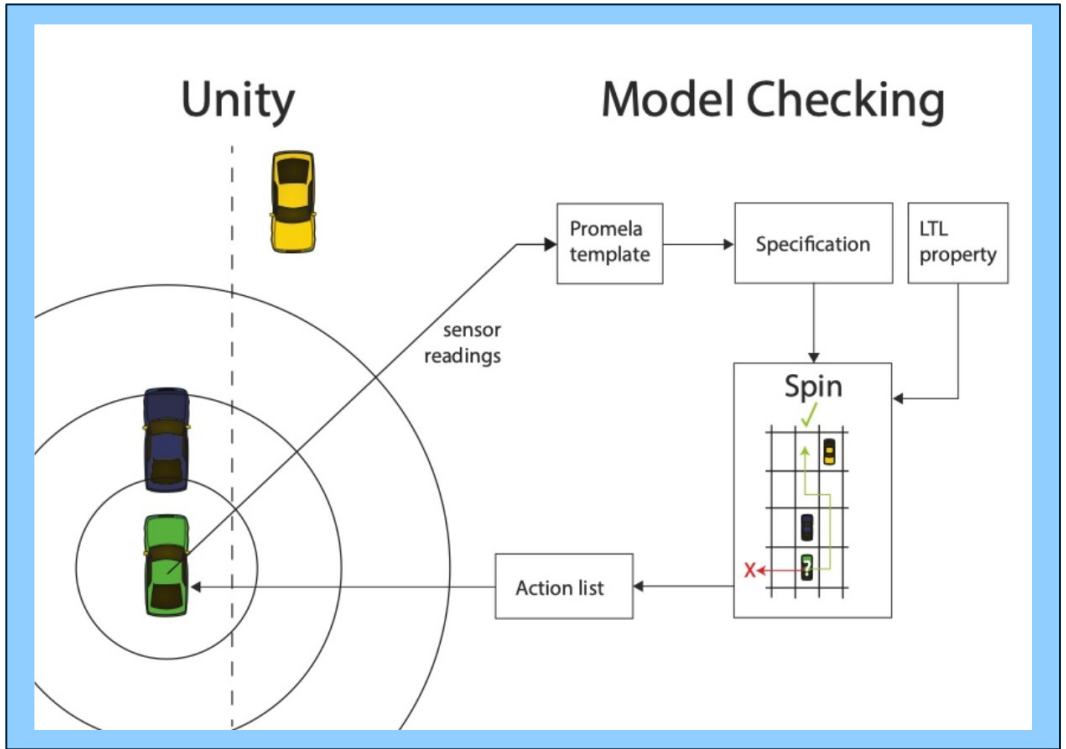
- Final sensor system chosen:
  - 360 degree sensor with 100m range for sensing left lane vehicles
  - front right Lane sensor with 357m range for detecting OV
  - right side sensor for sensing obstacles on right side not detected by front right lane sensor



AV overtaking two consecutive vehicles. Yellow boxes - sensed by the 360-degrees sensor. Red boxes - used as reference points for the acceleration and braking manoeuvres. OV (blue box), sensed using the long range sensor

Combining Unity + Spin

- Verification takes ~ 20ms
  - but compilation of Promela to C takes 3 secs (too long)
  - have to pause the simulation to generate solution path
  - pauses were removed from our video!
- only recompute the trajectory when new surrounding vehicles sensed or no remaining generated actions



If no actions returned – no successful overtake possible. (Did not happen in our experiments).

Results

Simulation ID	*Runtime*	*Vehicles over-taken*	*Distance Travelled (km)*	*Failure cause*
1	5h 30m	480	92.8	Sensor failure (two vehicles at position 9)
2	2h 20m	230	43.0	Sensor failure (fake gap)
3	2h 00m	200	38.2	Sensor failure (2 vehicles at position 11)
4	5h 20m	515	102.3	Sensor failure (fake gap)
5	1h 00m	100	19.6	Sensor failure (fake gap)
6	2h 30m	260	52.5	Steering failure
7	1h 30m	149	28.7	Sensor failure (rear sensor sees an extra obstacle)
8	3h 10m	296	60.5	Sensor failure (fake gap)
9	0h 40m	64	12.0	Sensor failure (fake gap)
10	2h 30m	105	21.9	Sensor failure (fake gap)
11	4h 30m	229	43.8	Sensor failure (rear sensor sees an extra obstacle)
12	0h 30m	44	10.0	Sensor failure (rear sensor sees an extra obstacle)

Table 1: Simulation results.

Discussion (results)

- 12 runs before failures. Total time 31.5 hours
- 2672 successful overtakes executed, distance 525 km
  - average 5 overtakes/km
  - median distance before failure 40.6 km
  - median number of overtakes before crashing 215.5
- Time of pausing not included (simulator clock stopped)

- All errors due to sensor or steering failures

- To speed up experiments AV forced to overtake as much as possible

Conclusion

- Successful proof of concept (getting all the bits to work together) – works well

- Limitations:
  - Road segment size (21m) too coarse for congested locations (finer resolution will lead to bigger state space)
  - Predictability of other road users unrealistic (e.g. all travel at constant speed) – maybe try Prism next time?
  - Pausing of simulation – biggest problem, to be addressed in future work (e.g. use portion of model checking algorithm)
  - Unity sensors are unreliable – but this reflects the real world!