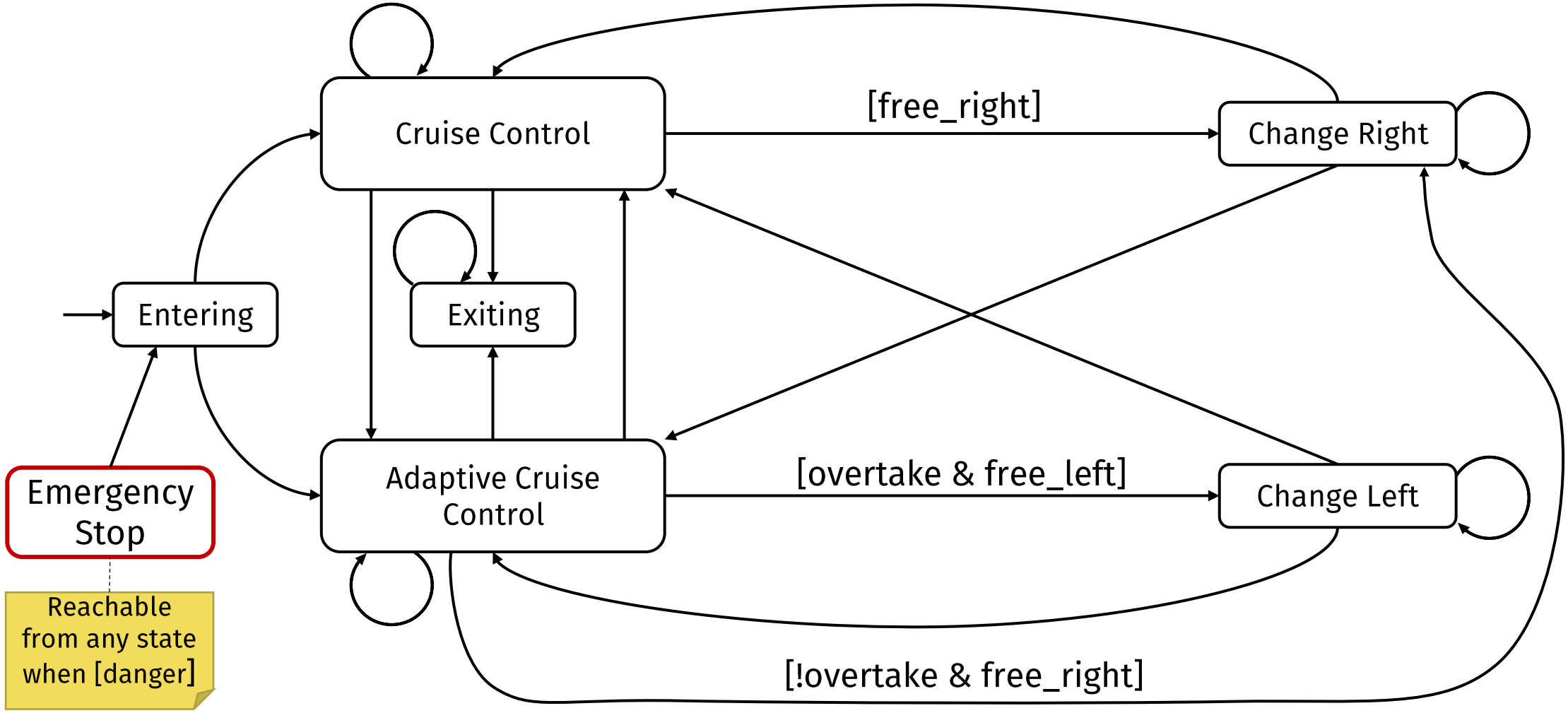
# Model Checking for Autonomous Vehicles

## Model checking a highway-lane controller with nuSMV

Luigi Libero Lucio STARACE
luigiliberolucio.starace@unina.it

May 28, 2020
University of Naples, Federico II

# The Highway Lane Controller Model



# Involved variables

- `danger (bool)`
  Models the occurrence of dangers requiring emergency stops.
- `free_left/right (bool)`
  Model whether the left/right lane are free.
- `overtake (bool)`
  Models the fact that the vehicle is willing to perform an overtake
- `current_lane (int)`
  Integer modelling the current lane occupied by the vehicle.

```
DEFINE
  NUMBER_OF_LANES := 10;

VAR
  danger       : boolean;
  overtake     : boolean;
  free_left    : boolean;
  free_right   : boolean;
  current_lane : 1..NUMBER_OF_LANES;
  state        : {
      entering, cruise_control,
      adaptive_cc, change_left,
      change_right, emergency_stop,
      exiting
  };
```

# Variable initialization

- `danger (bool)`
  Can be either TRUE or FALSE;
- `free_left/right (bool)`
- `overtake (bool)`
  We initialize the other boolean variables to false;
- `current_lane (int)`
  The vehicle starts at lane 1.

```
ASSIGN
  init(state)        := entering;
  init(danger)       := {TRUE,FALSE};
  init(overtake)     := FALSE;
  init(free_left)    := FALSE;
  init(free_right)   := FALSE;
  init(current_lane) := 1;
```

# Variable evolution: danger

- Danger can always happen!
- At each next step, it can either be TRUE or FALSE;

```
next(danger) := {TRUE, FALSE};
```

# Variable evolution: `overtake`

The need to overtake a vehicle can only happen (but does not need to!) when there is a preceeding vehicle.

```
next(overtake) := case
  next(state) = adaptive_cc : {TRUE, FALSE};
  TRUE: FALSE;
esac;
```

# Variable evolution: `free_left/right`

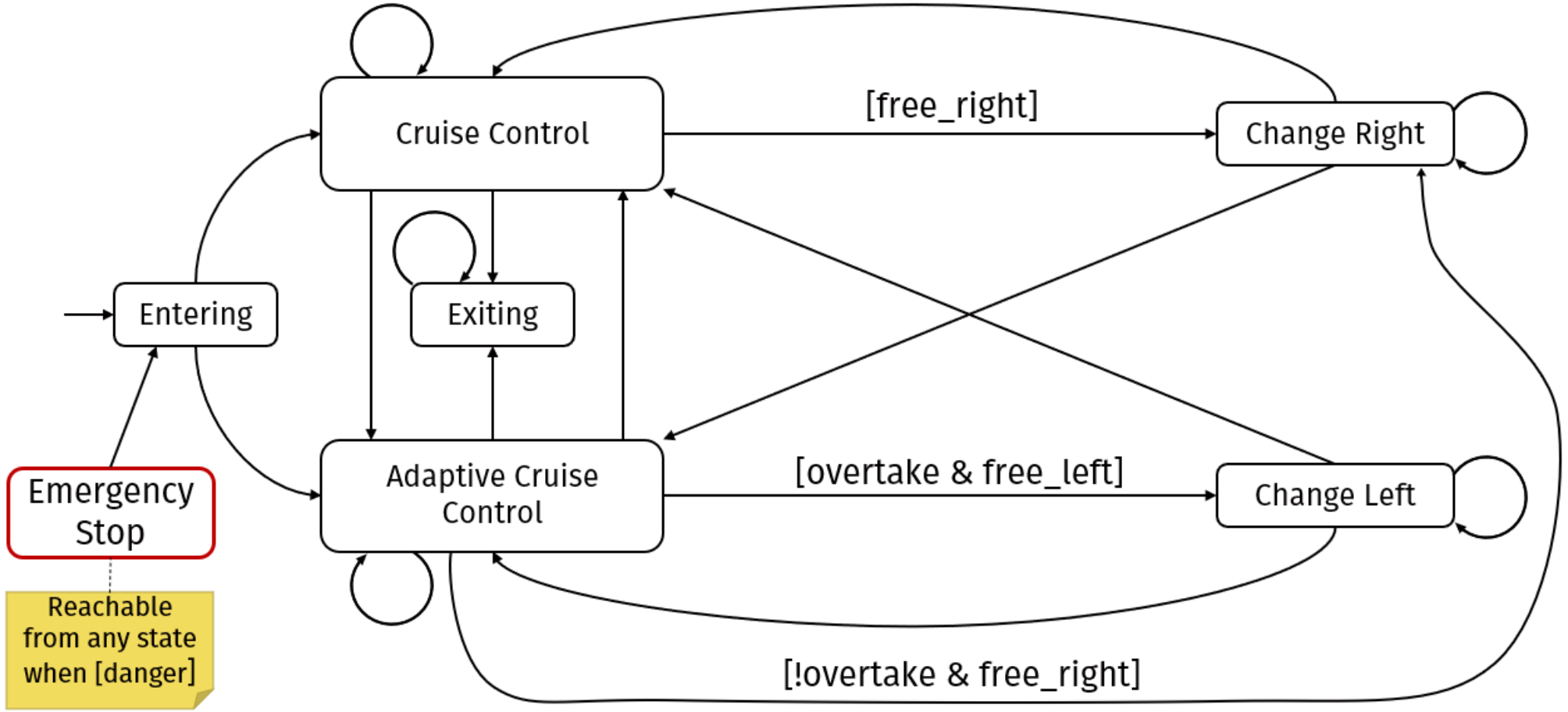
The lane to the right (left) cannot be free if we're in the first (last) lane! Otherwise, it can either be free or not.

```
next(free_right)   := case
  next(current_lane) = 1: FALSE;
  TRUE: {TRUE, FALSE};
esac;

next(free_left)    := case
  next(current_lane) < NUMBER_OF_LANES: {TRUE, FALSE};
  TRUE: FALSE;
esac;
```

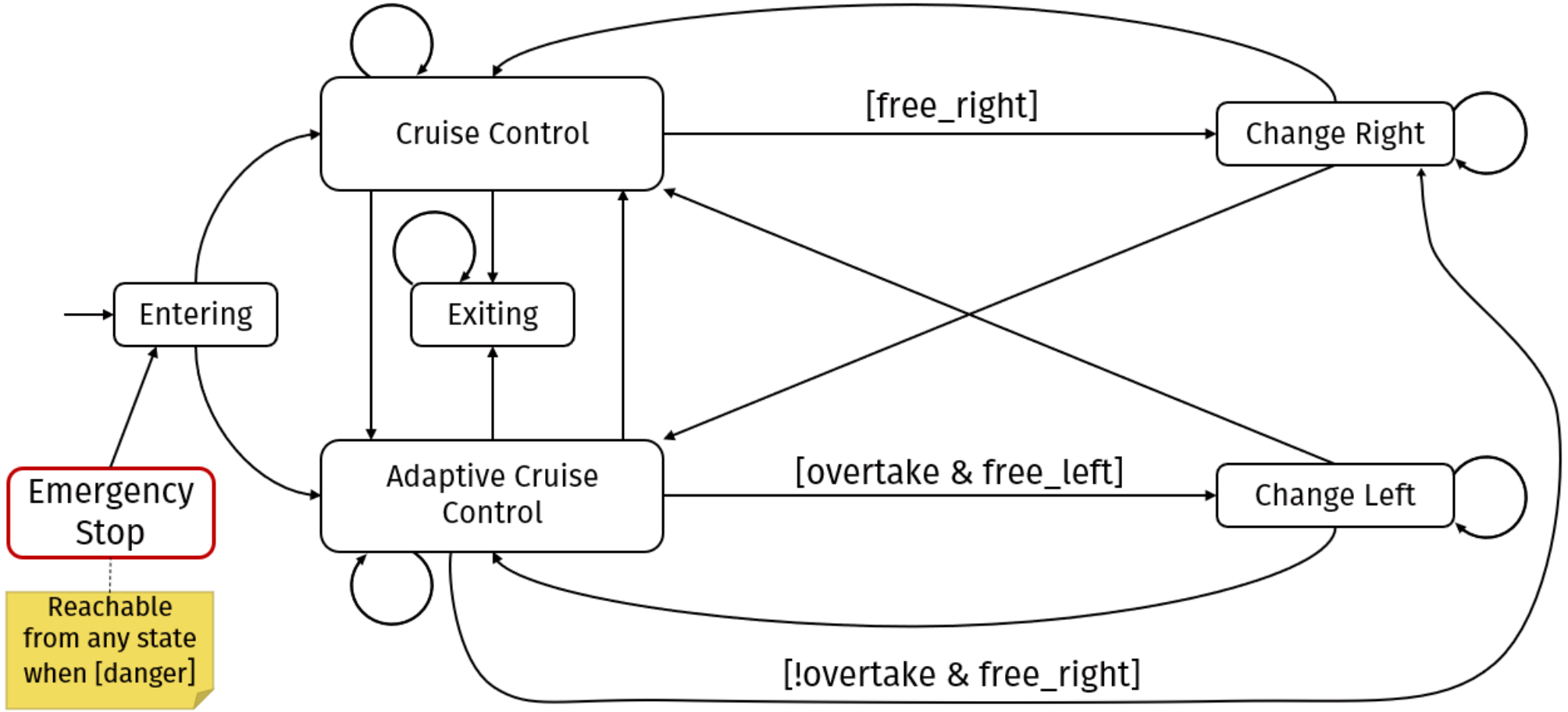
# Variable evolution: `current_lane`

If we're currently in the `change_left/right` state, but we won't be in the next state, then in the next state we will have completed the lane change, and `current_lane` can be updated accordingly. Otherwise `current_lane` remains unchanged.

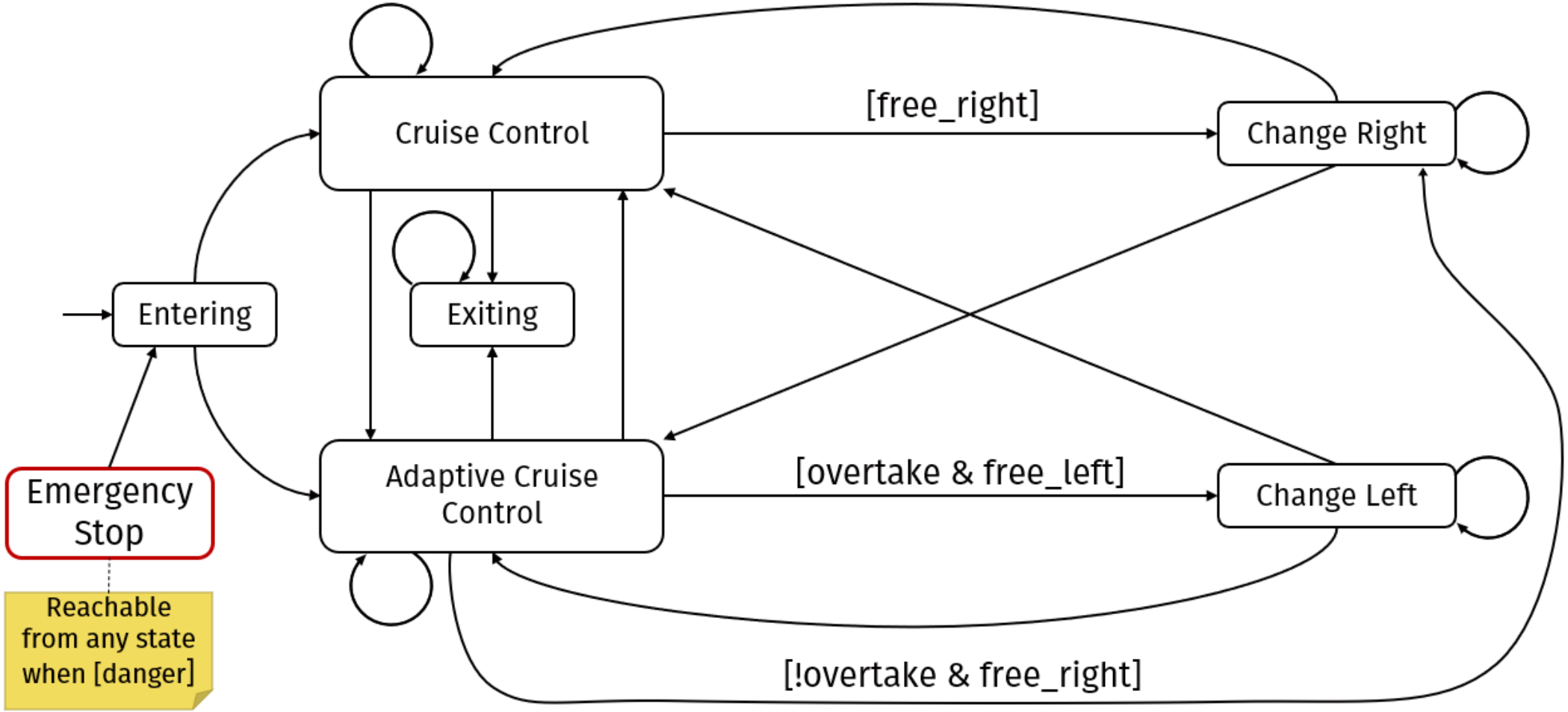
```
next(current_lane) := case
  state = change_left  & next(state) != change_left : current_lane + 1;
  state = change_right & next(state) != change_right: current_lane - 1;
  TRUE: current_lane;
esac;
```



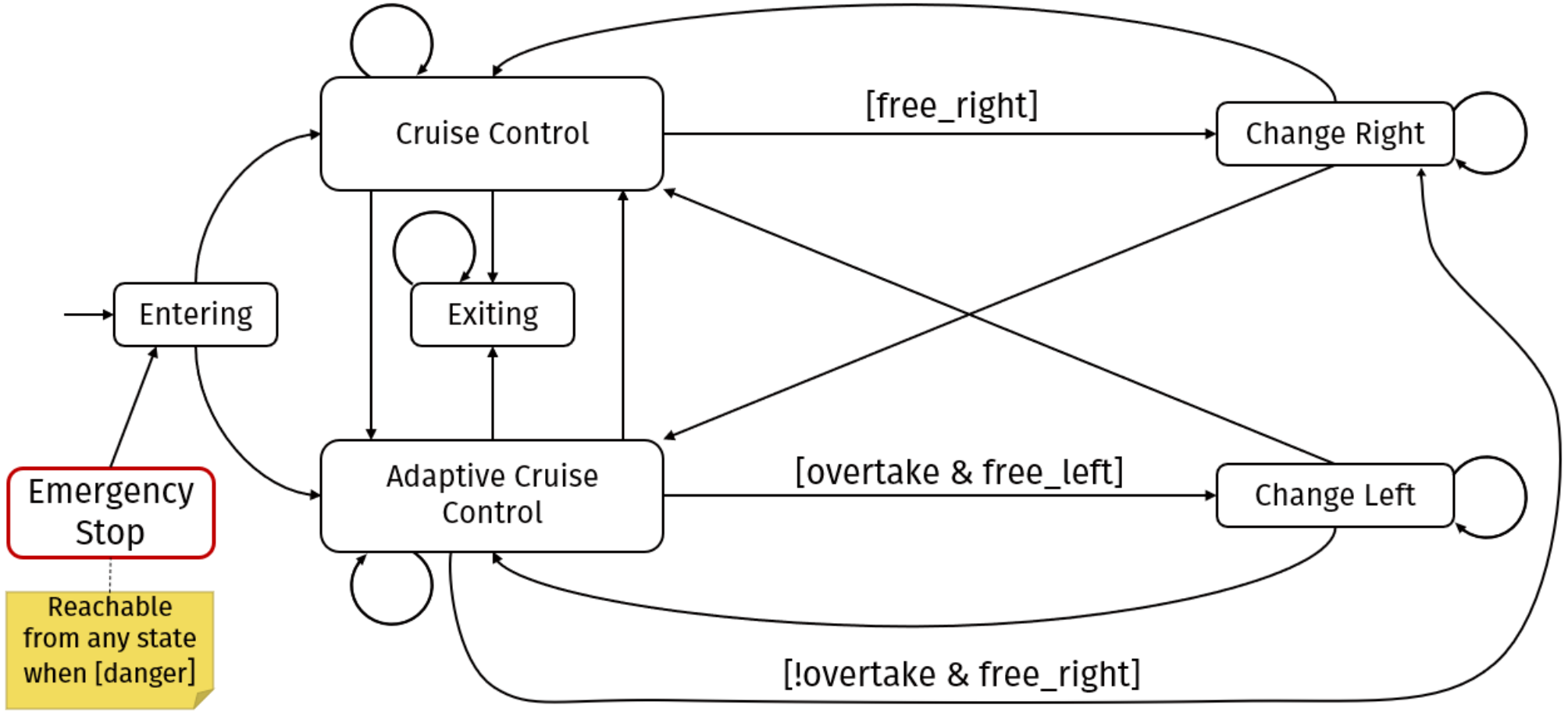

```
next(state) := case

  danger : emergency_stop;

  state = entering: {cruise_control, adaptive_cc};
```




```
state = cruise_control:  case
  free_right: change_right;
  TRUE: {cruise_control, exiting, adaptive_cc};
esac;
```




```
state = adaptive_cc:  case
  overtake  & free_left  : change_left;
  !overtake & free_right : change_right;
  TRUE: {cruise_control, exiting, adaptive_cc};
esac;
```




```
  state = change_right   : {cruise_control, adaptive_cc, change_right};
  state = change_left    : {cruise_control, adaptive_cc, change_left};
  state = emergency_stop : {emergency_stop, entering};
  state = exiting        : exiting;
esac;
```

# Running NuSMV

1. Open a command prompt, and run the NuSMV binary with the `–int` parameter to execute it in interactive mode.
2. Load a model specification file with `read_model –i <file>`
3. Prepare the model for verification with the `go` command.

```
D:\NuSMV-2.5.4-i386-pc-mingw32>bin\NuSMV.exe -int
*** This is NuSMV 2.5.4 (compiled on Fri Oct 28 14:06:40 UTC 2011)
*** Copyright (c) 2010, Fondazione Bruno Kessler
*** (omissis)

NuSMV > read_model -i specification.smv
NuSMV > go
NuSMV >
```

# Running NuSMV – Statistics

We can check how many states our finite state model has with the command `print_reachable_states`.

Why 1120? We have 7 states, 10 possible lane values, and four Boolean variables. $7 \cdot 10 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 1120$ !

```
NuSMV > print_reachable_states
############################################################
system diameter: 21
reachable states: 568 (2^9.14975) out of 1120 (2^10.1293)
############################################################
NuSMV >
```

# Running NuSMV – Verifying properties

We can verify LTL and CTL properties with the `check_ltlspec` and `check_ctlspec` commands.

```
NuSMV > check_ltlspec -p "G(current_lane<11)"
-- specification  G current_lane < 11  is true

NuSMV > check_ctlspec -p "AG(state!=entering)"
-- specification AG state != entering  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  state = entering
  NUMBER_OF_LANES = 10
NuSMV >
```

# Some interesting properties

- It never happens that the need to perform an overtake arises when in the Cruise Control state (and there's no vehicle ahead!)

```
G!( state=cruise_control & overtake )
```

- It never happens that the vehicle is the first (last) lane and free_right (free_left) are true.

```
G!( current_lane = 1 & free_right )
```

# Some interesting properties

- If a danger occurs, the vehicle must perform emergency stop operations.

```
G( danger -> X(state=emergency_stop) )

G!( danger & X(state!=emergency_stop) )

AG( danger -> AX(state=emergency_stop) )

AG!( danger & AX(state!=emergency_stop) )
```

# Some interesting properties

- A vehicle cannot go from `change_left` to `change_right` and vice-versa.

```
G!( state=change_left & X(state=change_right))
G!( state=change_right & X(state=change_left))

G( state=change_right -> X(state!=change_left) )
G( state=change_left -> X(state!=change_right) )
```

# Some interesting properties

- Whatever happens, a vehicle can always potentially reach the `exiting` state.

```
AG(EF(state=exiting))
```

# Some interesting properties

- In any possible behaviour, a vehicle cannot visit the `change_right` state before it visited the `change_left` one. In other words, if it visits the `change_right` state, it must have visited the `change_left` state before.

```
state=change_left V state!=change_right
```

The release temporal operator R is written as V in NuSMV!

# Some interesting properties

- If a vehicle start a maneuver to change lane, that maneuver must end in a finite number of steps and the vehicle should transit either in one of the Cruise Control states, or in `emergency_stop`.

```
G( state=change_left -> F(state!=change_left) )
```

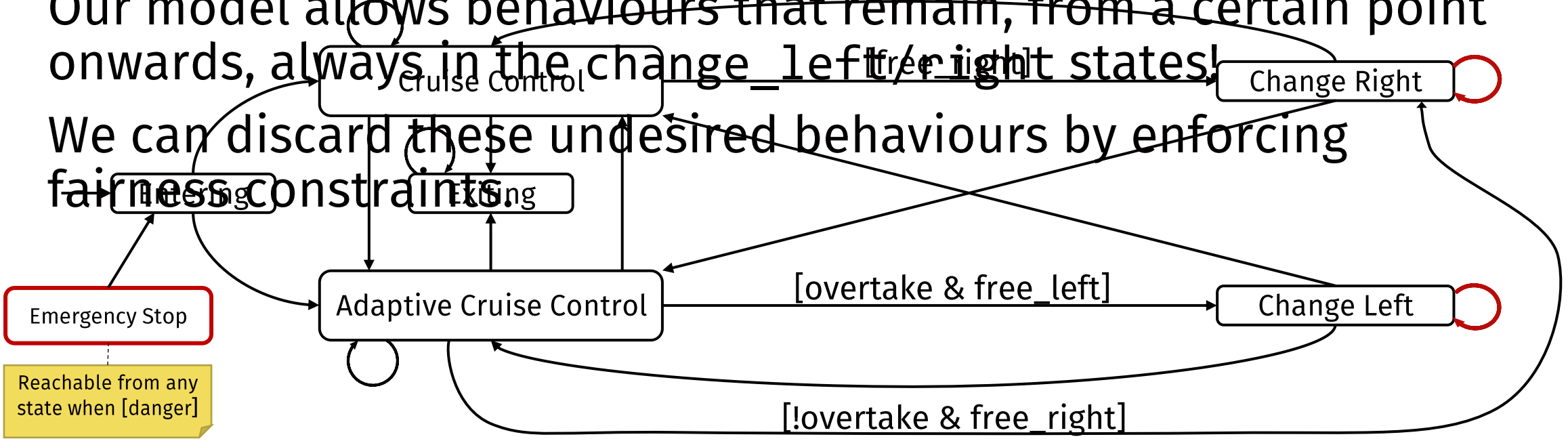
# Some interesting properties

```
G( state=change_left -> F(state!=change_left) )
```

The above specification is correct, but it's false on our model!

Our model allows behaviours that remain, from a certain point onwards, always in the change_left/right states!

We can discard these undesired behaviours by enforcing fairness constraints.



# Enforcing fairness

In NuSMV, fairness constraints can be expressed by defining formulae that must hold infinitely often in the acceptable runs.

All the runs in which these formulae do not hold infinitely often are discarded.

In our case, we can discard the runs that get stuck in the change lane states by adding to our model

```
FAIRNESS state!=change_left ;
FAIRNESS state!=change_right ;
```

# That’s all! Any questions?

# References

- NuSMV 2.5 User Manual
  http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf
- NuSMV 2.5 Tutorial (for more examples)
  http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf