Traffic Light Controller Examples in SMV

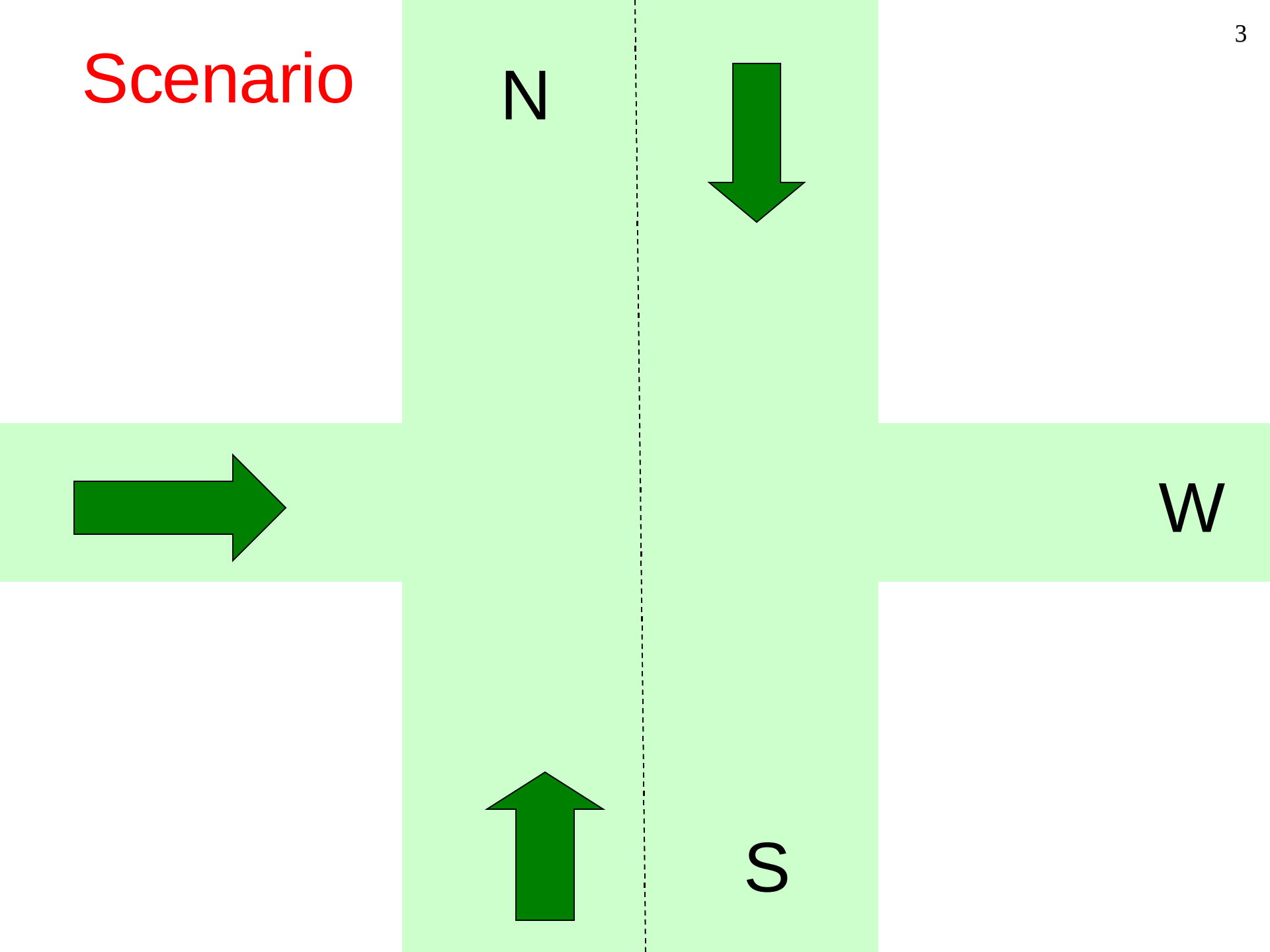
Himanshu Jain

Bug catching (Fall 2007)

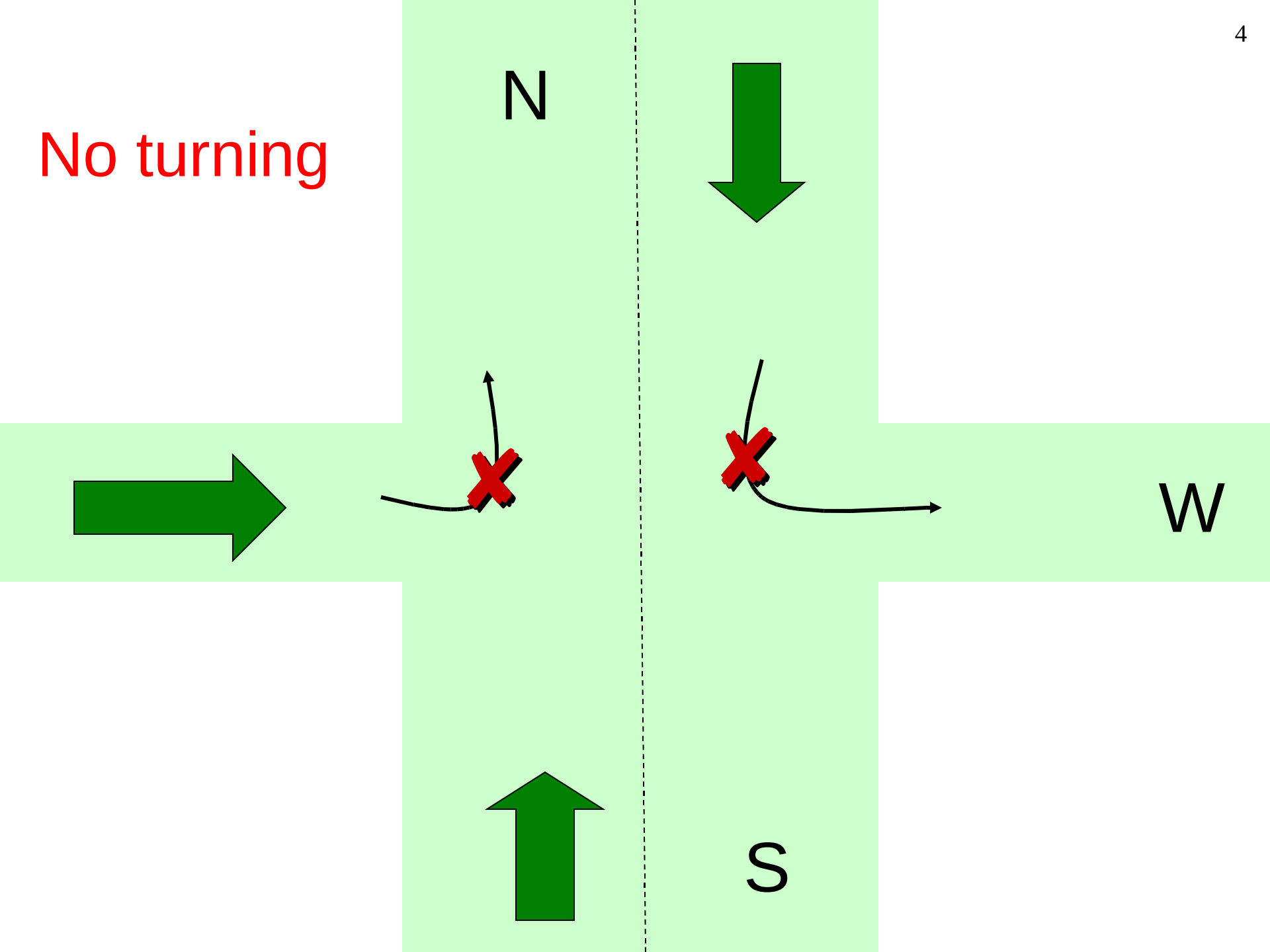
Plan for today

- Modeling Traffic Light Controller in SMV
- Properties to Check
- Four different SMV models for traffic light controller

Scenario

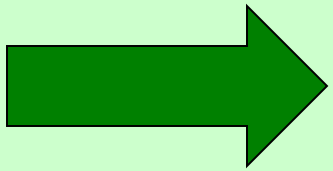


No turning



Binary traffic lights

N



W

S

Safety Property

N

W

This should not happen

S

Safety Property

N

W

S

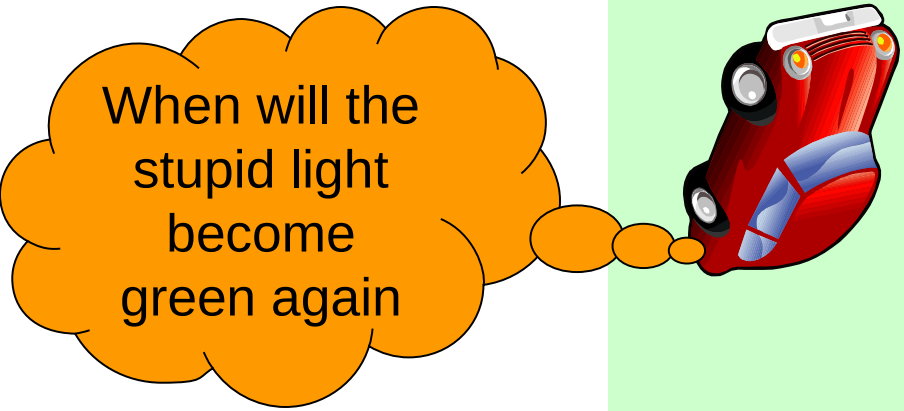
This should not happen

Liveness Property

N

W

S



Liveness Property

N

W

Thank God!

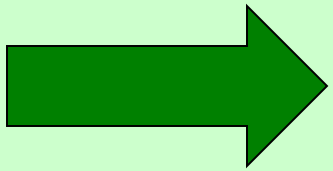
S

Traffic in each direction must be served

Let’s see how to model all this in SMV

SMV variables

Three Boolean variables track the status of lights

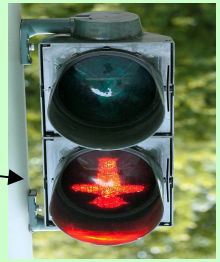


N

N-go=0

W

W-go=1



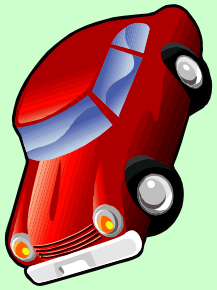
S-go=0

S

SMV variables

Three Boolean variables sense the traffic in each direction

N



S-sense =1

W-sense =0

W

N-sense =1

S

These variables are called N, Sy, W in the code I will show you

Properties we would like to check

- Mutual exclusion
  - **SPEC AG !(W-Go & (N-Go | S-Go))**
- Liveness in North direction
  - **SPEC AG(N-sense & !N-Go -> AF N-Go)**
- Similar liveness properties for south and west

Properties we would like to check

- No strict sequencing
  - We don’t want the traffic lights to give turns to each other (if there is no need for it)
  - For example, if there is no traffic on west lane, we do not want W-go becoming 1 periodically
- We can specify such properties atleast partially
  - **AG(W-Go -> A[W-Go U (!W-Go & A[!W-Go U (N-Go | S-Go)])])**
  - See code other such properties
  - We want these properties to FAIL

SMV modules

N

North module will control

West module will control

W

South module will control

S

Main module will
-Initialize variables
-Start north, south, west modules

What if north light is always green and there is always traffic in north direction

N

W

S

Fairness Constraints

- What if north light is always green and there is always traffic in north direction
- We will avoid such scenarios by means of fairness constraints
- FAIRNESS running & !(N-Go & N-sense)
- *On an infinite execution, there are infinite number of states where either north light is not green or there is no traffic in north direction*
- Similar, fairness constraints for south and west directions

Now we look at some concrete implementations

Some more variables

- To ensure mutual exclusion
  - We will have two Boolean variables
  - **NS-Lock**: denotes locking of north/south lane
  - **EW-Lock**: denotes locking of west lane
- To remember that there is traffic on a lane
  - Boolean variables: N-Req, S-Req, W-Req
  - If N-sense becomes 1, then N-Req is set to true
  - Similarly, for others….

Traffic1.smv

```
MODULE main
VAR
        N : boolean;       --senses traffic going along north
        Sy : boolean;      --senses traffic going along south
        W : boolean;       --senses traffic going westward
        N-Req : boolean;   --rememebers that there is traffic along north that needs to go
        S-Req : boolean;   --rememebers that there is traffic along south that needs to go
        W-Req : boolean;   --rememebers that there is traffic along west  that needs to go
        N-Go : boolean;    --north direction green light on
        S-Go : boolean;    --south direction green light on
        W-Go : boolean;    --west direction green light on
        NS-Lock : boolean; --north/south lane locked
        EW-Lock : boolean; --east/west lane locked

        north : process north1(NS-Lock, EW-Lock, N-Req, N-Go,N,S-Go);
        south : process south1(NS-Lock,EW-Lock,S-Req,S-Go,Sy,N-Go);
        west : process west1(NS-Lock,EW-Lock,W-Req,W-Go,W);

ASSIGN
        init(NS-Lock) := 0; init(Sy) := 0;
        init(W) := 0;
        init(W-Req) := 0; ……..OTHER INITIALIZATIONS
```

```
MODULE  north(NS-Lock, EW-Lock, N-Req, N-Go,N,S-Go)
VAR
      state : {idle, entering , critical , exiting};
ASSIGN
      init(state) := idle;
      next(state) :=
            case
                  state = idle : case
                                    N-Req = 1 : entering;
                                    1 : state;
                                   esac;
                  state = entering & !EW-Lock : critical;
                  state = critical & !N : exiting;
                  state = exiting : idle;
                  1 : state;
            esac;

      next(NS-Lock) :=
            case
                  state = entering & !EW-Lock : 1 ;
                  state = exiting & !S-Go : 0;
                  1 : NS-Lock;
            esac;
```

```
next(N-Req) :=
            case
                  !N-Req & N : 1;
                  state = exiting : 0;
                  1 : N-Req;
            esac;

      next(N-Go) :=
            case
                  state = critical : 1;
                  state = exiting : 0;
                  1 : N-Go;
            esac;

-- non-deterministically chose N
      next(N) := {0,1};

FAIRNESS
      running & !(N-Go & N)
```

Module south is similar

Module west1 is a little different

Everything seems ok!

Let us run a model checker

Mutual exclusion fails (Counterexample)

1. All variables zero
2. N-sense=1 (North module executed)
3. S-sense=1 (South module executed)
4. S-Req=1
5. south.state=entering
6. S-sense=0, **NS-Lock=1**, **south.state=critical**
7. S-sense=1,**S-go=1**,south.state=exiting
8. N-Req=1
9. north.state=entering
10. **north.state=critical**
11. S-Req=0, **S-Go=0**, **NS-Lock=0**, south.state=idle
12. W=1
13. W-Req=1
14. west.state=entering
15. EW-lock=1, **west.state=critical**
16. **W-Go=1**
17. **N-Go=1**

One module is executing at each step

Mutual exclusion fails (Counterexample)

1. All variables zero
2. N-sense=1 (North module executed)
3. S-sense=1 (South module executed)
4. S-Req=1
5. south.state=entering
6. S-sense=0, **NS-Lock=1**, **south.state=critical**
7. S-sense=1,**S-go=1**,south.state=exiting
8. N-Req=1
9. north.state=entering
10. **north.state=critical**
11. S-Req=0, **S-Go=0**, **NS-Lock=0**, south.state=idle
12. W=1
13. W-Req=1
14. west.state=entering
15. EW-lock=1, **west.state=critical**
16. **W-Go=1**
17. **N-Go=1**

One module is executing at each step

Even though north.state is critical the NS-lock is released

Mutual exclusion fails (Counterexample)

1. All variables zero
2. N-sense=1 (North module executed)
3. S-sense=1 (South module executed)
4. S-Req=1
5. south.state=entering
6. S-sense=0, **NS-Lock=1**, **south.state=critical**
7. S-sense=1,**S-go=1**,south.state=exiting
8. N-Req=1
9. north.state=entering
10. **north.state=critical**
11. S-Req=0, **S-Go=0**, **NS-Lock=0**, south.state=idle
12. W=1
13. W-Req=1
14. west.state=entering
15. EW-lock=1, **west.state=critical**
16. **W-Go=1**
17. **N-Go=1**

One module is executing at each step

One problem is the one-step difference Between North.state=critical and N-Go=1

```
MODULE  north(NS-Lock, EW-Lock, N-Req, N-Go,N,S-Go)
VAR
      state : {idle, entering , critical , exiting};
ASSIGN
      init(state) := idle;
      next(state) :=
            case
                  state = idle : case
                                    N-Req = 1 : entering;
                                    1 : state;
                                 esac;
                  state = entering & !EW-Lock : critical;
                  state = critical & !N : exiting;
                  state = exiting : idle;
                  1 : state;
            esac;

      next(NS-Lock) :=
            case
                  state = entering & !EW-Lock : 1 ;
                  state = exiting & !S-Go : 0;
                  1 : NS-Lock;
            esac;
```

```
next(N-Req) :=
            case
                  !N-Req & N : 1;
                  state = exiting : 0;
                  1 : N-Req;
            esac;

      next(N-Go) :=
            case
                  state = critical : 1;
                  state = exiting : 0;
                  1 : N-Go;
            esac;

-- non-deterministically chose N
      next(N) := {0,1};

FAIRNESS
      running & !(N-Go & N)
```

This problem is fixed in traffic2.smv

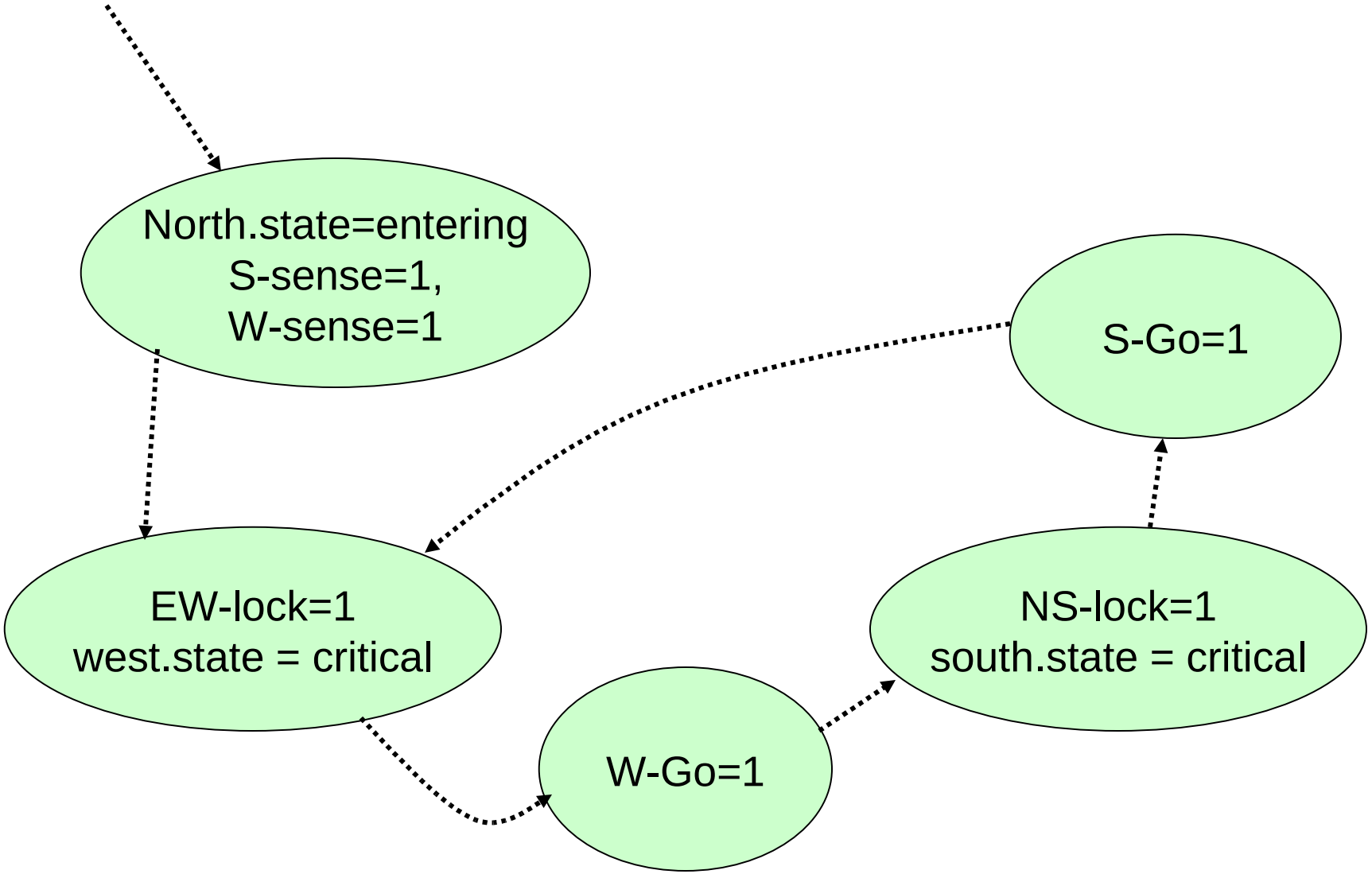
```
next(state) :=
        case
            state = idle : case
                            N-Req = 1 : entering;
                            1 : state;
                           esac;
            state = entering & !EW-Lock : critical;
            state = critical & !N : exiting;
            state = exiting : idle;
            1 : state;
        esac;

next(N-Go) :=
        case
            state = entering & !EW-Lock : 1;    --change here
            state = exiting : 0;
            1 : N-Go;
        esac;
```

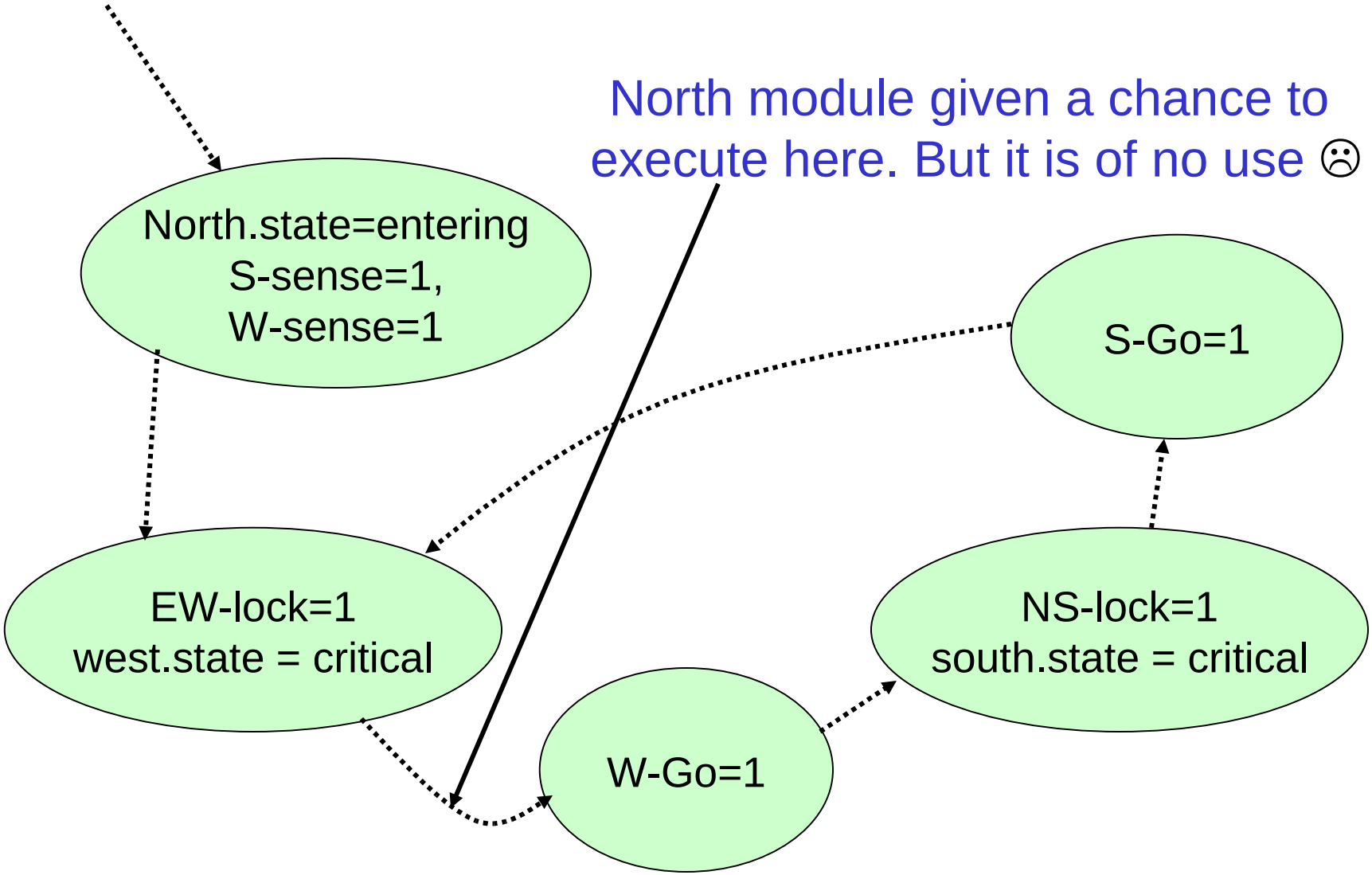
Model checking traffic2.smv

- Mutual exclusion property is satisfied
- Liveness property for North direction fails
  - AG ((N & !N-Go) -> AF N-Go) is false

Counterexample for liveness property contains a loop



Counterexample for liveness property contains a loop



Ensuring liveness requires more work

- This is in traffic3.smv

- Introduce a Boolean variable called turn
  - Give turn to others (if I have just exited the critical section)
  - turn = {nst, ewt}

```
MODULE  north1(NS-Lock, EW-Lock, N-Req, N-Go,N,S-Go,S-Req,E-Req,turn)
VAR
        state : {idle, entering , critical , exiting};

ASSIGN
        init(state) := idle;
        next(state) :=
                case
                        state = idle & N-Req = 1 : entering;
                        state = entering & !EW-Lock & (!E-Req | turn=nst): critical;
                        state = critical & !N : exiting;
                        state = exiting : idle;
                        1 : state;
                esac;

next(turn) :=
                case
                        state=exiting & turn=nst  & !S-Req : ewt;
                        1 : turn;
                esac;
```

Similar code in south and west modules

Model check again

- Mutual exclusion holds
- What about liveness properties
  - In north direction?
  - In south direction?
  - In west direction?

Model check again

- Mutual exclusion holds

- What about liveness properties
  - In north direction? HOLDS
  - In south direction? HOLDS
  - In west direction? FAILS ☹

Traffic4.smv ☺

- Two more variables to distinguish between north and south completion
  - ndone and sdone
- When north module exits critical section ndone is set to 1
  - Similarly for south module and sdone
- When west module exits both sdone and ndone are set to 0

```
MODULE  north1(NS-Lock, EW-Lock, N-Req, N-Go,N,S-Go,S-Req,E-
Req,turn,ndone,sdone)
VAR
        state : {idle, entering , critical , exiting};
ASSIGN
next(state) :=
                case
                        state = idle & N-Req = 1 : entering;
                        state = entering & !EW-Lock & (!E-Req | turn=nst): critical;
                        state = critical & !N : exiting;
                        state = exiting : idle;
                        1 : state;
                esac;
 next(turn) :=
                case
                        state=exiting & turn=nst  & (!S-Req | (sdone & E-Req)): ewt;
                        1 : turn;
                esac;
next(ndone) :=
                case
                        state=exiting : 1;
                        1 : ndone;
                esac;
```

Hurray!

- Mutual exclusion holds
- Liveness for all three directions holds
- Strict sequencing does not hold
  - That is what we want

Think about

- How to allow north, south, east, west traffic
- How to model turns
- Instead of writing code for four modules have a generic module
  - Instantitate it with four times. Once for each direction
- Ensure properties without changing fairness constraints

We will make the SMV code and slides available

QUESTIONS