EECS4315 Mission-Critical Systems

Tutorials on TLA+

Instructor: Jackie Wang

Created: Winter 2023

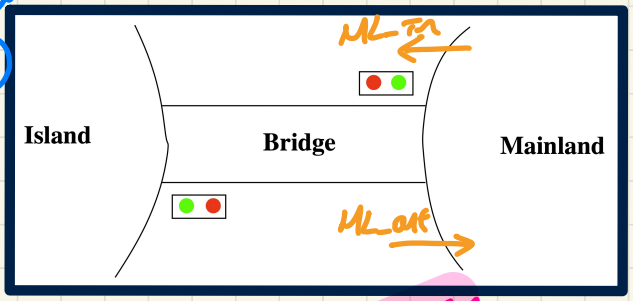
Bridge Controller:

Requirements Document

E-descriptions (document the working environment constituents)

ENV1	The system is equipped with two traffic lights with two colors: green and red.
ENV2	The traffic lights control the entrance to the bridge at both ends of it.
ENV3	Cars are not supposed to pass on a red traffic light, only on a green one.
ENV4	The system is equipped with four sensors with two states: on or off.
ENV5	The sensors are used to detect the presence of a car entering or leaving the bridge: "on" means that a car is willing to enter the bridge or to leave it.
REQ1	The system is controlling cars on a bridge connecting the mainland to an island.
REQ2	The number of cars on bridge and island is limited.
REQ3	The bridge is one-way or the other, not both at the same time.

R-descriptions (document the intended functionalities)



Correct by Construction

↳ - do not encode all details in a single model
- distribute variables & properties in gradual refinement models

Bridge Controller: State Transitions of the Initial Model

REQ2	The number of cars on bridge and island is limited.

Initial abstraction: no notion of bridge!

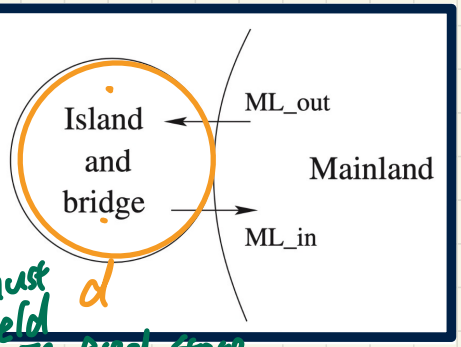
constants: d — max # cars in IB compound

axioms:
axm0_1 : $d \in \mathbb{N}$ — 0..∞

variables: n

invariants:
inv0_1 : $n \in \mathbb{N}$
inv0_2 : $n \leq d$

safety invariant that must be held in every state.



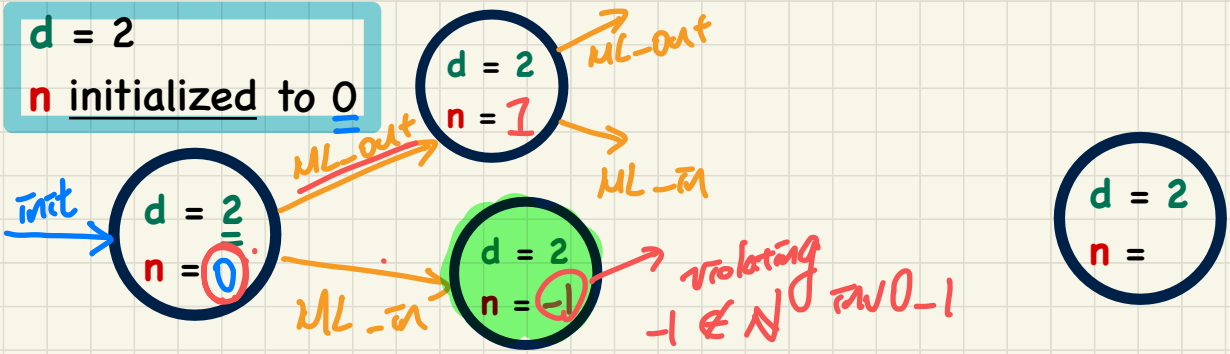
- ML_out
  begin
  $n := n + 1$
  end

implicitly: guard is T. — action

- ML_in
  begin
  $n := n - 1$
  end

State Transition Diagram on an Example Configuration

d = 2
n initialized to 0



P, Q → R

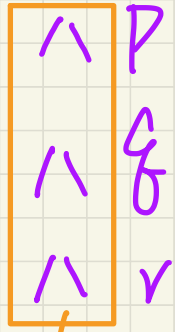
operands

Logic

P ∧ Q ∧ R

operators = # operands - 1

TLA+



operators = # operands

Event Execution via a Central Controller

To model check, system under verification should be kept finite.

```
\* main program
{
  \* # iterations: bound
  while (i < bound) {
    \* Use the choice operator to simulate the selection of event execution by some central controller.
    (*
      When multiple conditions are satisfied, it is non-deterministic as to which branch would be executed.
    *)
    either {
      if(TRUE) {
        call ML_out();
      };
    }
    or {
      if(TRUE) {
        call ML_in();
      };
    };
    i := i + 1
  }
}
```

a constant which will be instantiated differently when creating "models"

design-level construct

3 iterations

i=0

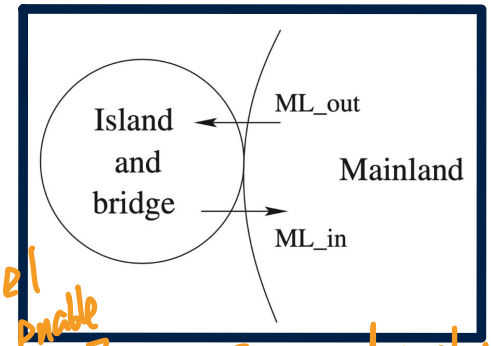
i=1

i=2

non-deterministic choice

if (..)

else if (..)



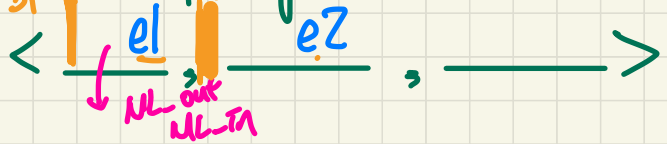
for this part of interleaving to be valid, the post-state of e1 must enable e2 (i.e., e2's guard evaluates to T)

if T at the same time, the choice of execution is deterministic

System Trace: sequence of event occurrences

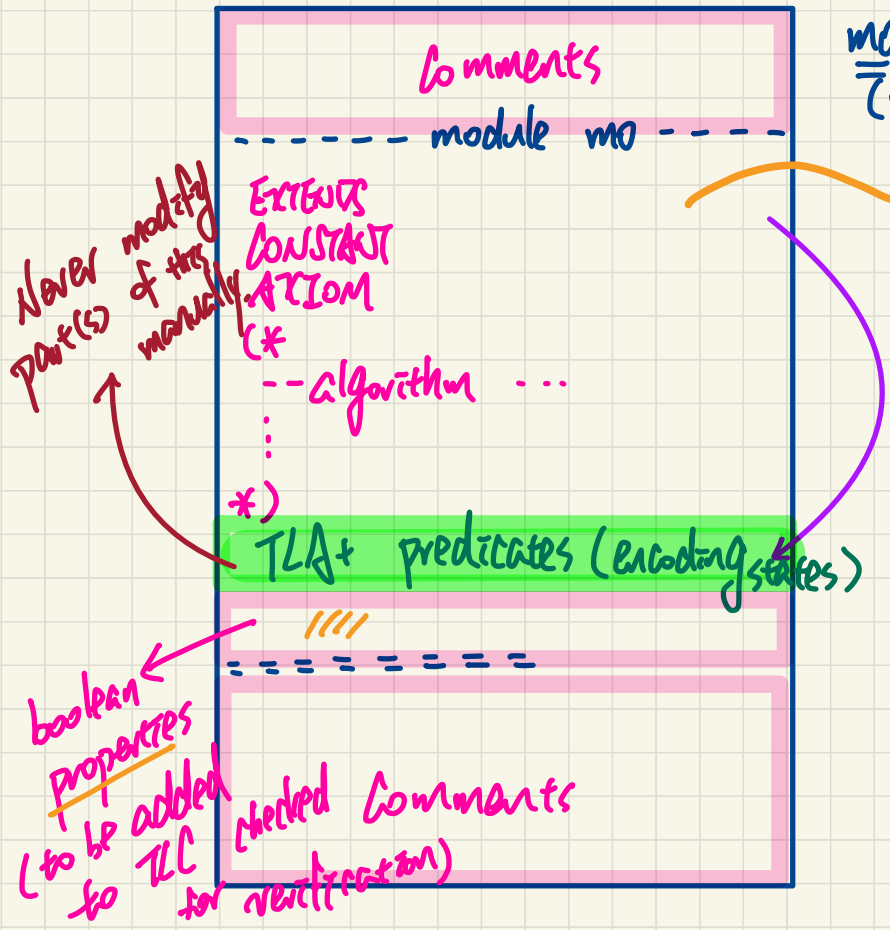
e.g. bound = 3 : interleaving of sys. events of length 3

pre-state of e1



2^3 possible interleavings

Using TLA+: Manual vs. Automated



m0.tla = (module)

Re-translation unnecessary if (1) the properties part modified (2) algo. part unchanged.

(1) Model checking cannot be performed directly upon PlusCal

(2) Translate the PlusCal algorithm into (TLA+ predicates) whenever before-after predicate an intermediate version of algo. is done (so as to find errors) syntax, types.

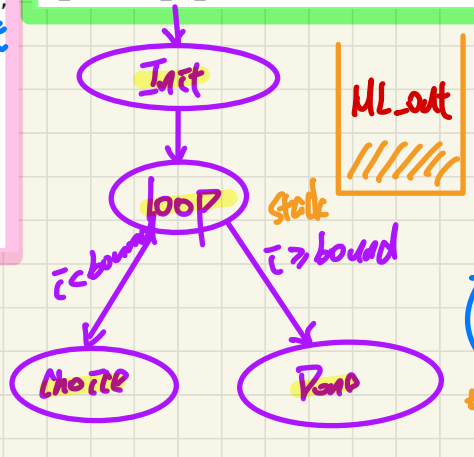
PlusCal vs. TLA+

```
--algorithm bridgeController_m0 {
  variable n = 0, i = 0;

  procedure ML_out() {
    ML_out_action: n := n + 1;
    return;
  }

  procedure ML_in() {
    ML_in_action: n := n - 1;
    return;
  }

  \* main program
  {
    loop: while (i < bound) {
        choice: either {
            ML_out_guard: if(TRUE) {
              * ML_out_occurs: call ML_out();
            };
        }
        or {
            ML_in_guard: if(TRUE) {
                ML_in_occurs: call ML_in();
            };
        };
        progress: i := i + 1
    }
  }
```

PlusCal

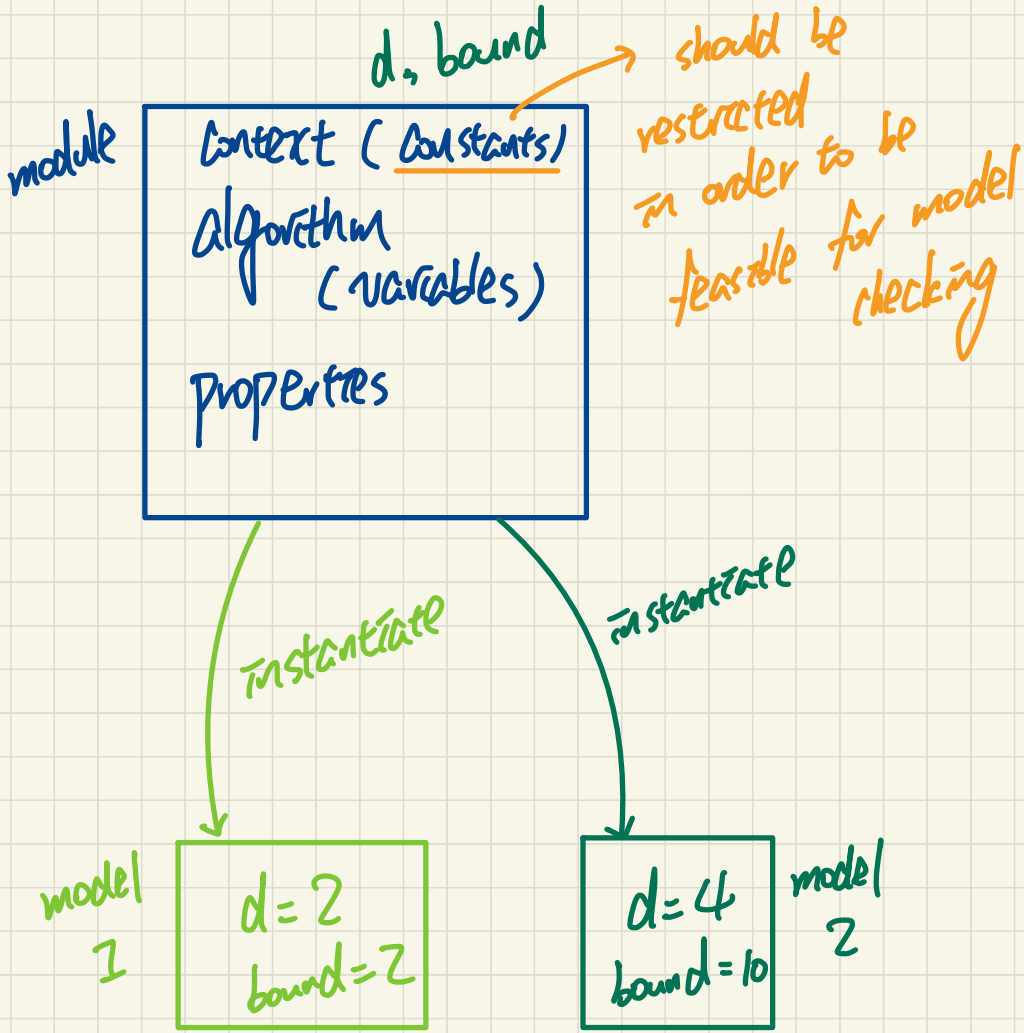
Exercise: Complete the Control-Flow Diagram

```
VARIABLES n, i, pc, stack

vars == << n, i, pc, stack >>

Init == (* Global variables *)
        /\ n = 0
        /\ i = 0
        /\ stack = << >>
        /\ pc = "loop"

ML_out_action == /\ pc = "ML_out_action"
                 /\ n' = n + 1
                 /\ pc' = Head(stack).pc
                 /\ stack' = Tail(stack)
                 /\ i' = i

ML_out == ML_out_action

ML_in_action == /\ pc = "ML_in_action"
                /\ n' = n - 1
                /\ pc' = Head(stack).pc
                /\ stack' = Tail(stack)
                /\ i' = i

ML_in == ML_in_action
```



TLA+ Translation (control flow)

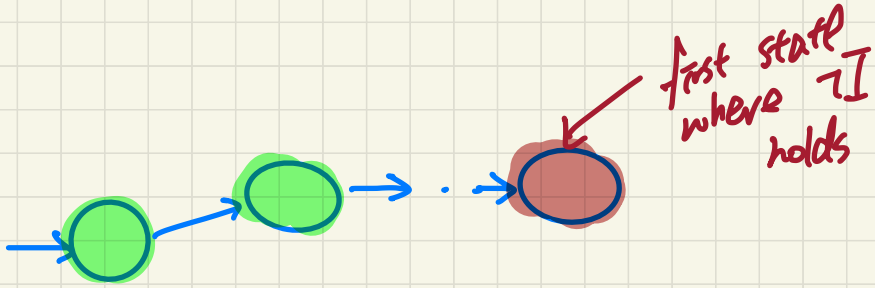
```
loop == /\ pc = "loop"
        /\ IF i < bound
              THEN /\ pc' = "choice"
              ELSE /\ pc' = "Done"
        /\ UNCHANGED << n, i, stack >>

choice == /\ pc = "choice"
          /\ \/ /\ pc' = "ML_out_guard"
             \/ /\ pc' = "ML_in_guard"
          /\ UNCHANGED << n, i, stack >>

ML_out_guard == /\ pc = "ML_out_guard"
                /\ IF TRUE
                      THEN /\ pc' = "ML_out_occurs"
                      ELSE /\ pc' = "progress"
                /\ UNCHANGED << n, i, stack >>

ML_out_occurs == /\ pc = "ML_out_occurs"
                 /\ stack' = << [ procedure |-> "ML_out",
                                  pc        |-> "progress" ] >>
                              \o stack
                 /\ pc' = "ML_out_action"
                 /\ UNCHANGED << n, i >>

ML_in_guard == /\ pc = "ML_in_guard"
               /\ IF TRUE
                     THEN /\ pc' = "ML_in_occurs"
                     ELSE /\ pc' = "progress"
               /\ UNCHANGED << n, i, stack >>

ML_in_occurs == /\ pc = "ML_in_occurs"
                /\ stack' = << [ procedure |-> "ML_in",
                                 pc        |-> "progress" ] >>
                             \o stack
                /\ pc' = "ML_in_action"
                /\ UNCHANGED << n, i >>

progress == /\ pc = "progress"
            /\ i' = i + 1
            /\ pc' = "loop"
            /\ UNCHANGED << n, stack >>

(* Allow infinite stuttering to prevent deadlock on termination. *)
Terminating == pc = "Done" /\ UNCHANGED vars

Next == ML_out \/ ML_in \/ loop \/ choice \/ ML_out_guard \/ ML_out_occurs
        \/ ML_in_guard \/ ML_in_occurs \/ progress
        \/ Terminating

Spec == Init /\ [][Next]_vars
```


d, bound
should be restricted in order to be feasible for model checking
module
context (constants)
algorithm
(variables)
properties
instantiate
instantiate
model 1
d = 2
bound = 2
d = 4
bound = 10
model 2

Remotelabs File Transfer

Windows: Ctrl + shift + Alt

Mac: Ctrl + shift + option



Error Trace

In checking that invariant property I holds on each state, if it fails, it implies that there's one system state s.t. ¬I holds. An error trace shows how to get to this state from init.

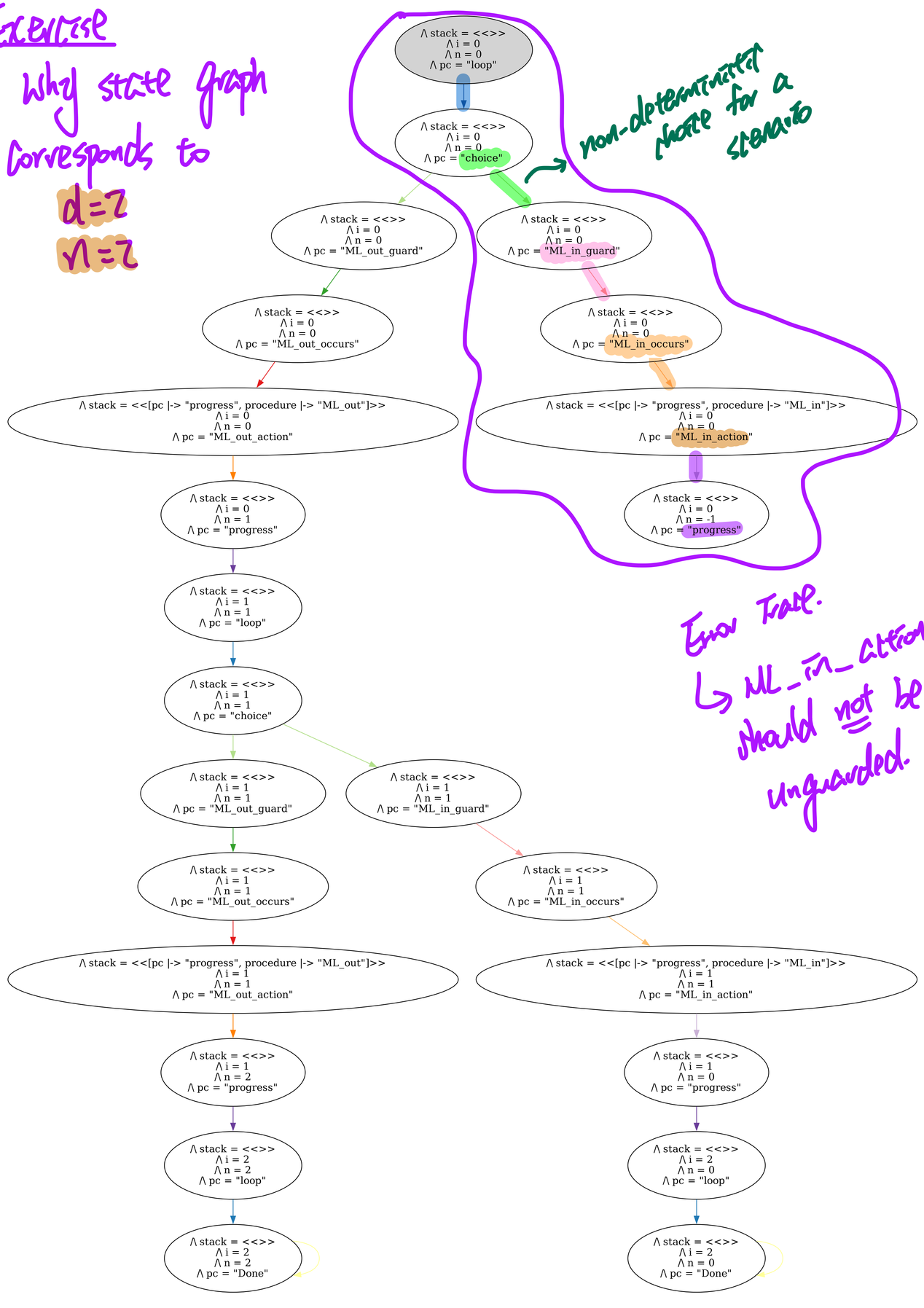
Using an error trace and the state graph, debug by finding the problematic part of the model/module.

Exercise

Why state graph corresponds to

d=2

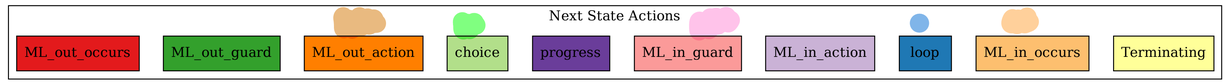
n=2



non-determinism! More for a scenario

Error Trace.

↳ ML_in_action should not be unguarded.




```
<<
[
 i |-> 0,
 n |-> 0,
 pc |-> "loop",
 stack |-> <<>>
],
[
 i |-> 0,
 n |-> 0,
 pc |-> "choice",
 stack |-> <<>>
],
[
 i |-> 0,
 n |-> 0,
 pc |-> "ML_in_guard",
 stack |-> <<>>
],
[
 i |-> 0,
 n |-> 0,
 pc |-> "ML_in_occurs",
 stack |-> <<>>
],
[
 i |-> 0,
 n |-> 0,
 pc |-> "ML_in_action",
 stack |-> <<[pc |-> "progress", procedure |-> "ML_in"]>>
],
[
 i |-> 0,
 n |-> -1,
 pc |-> "progress",
 stack |-> <<>>
]
>>
```

pre-state value of n before ML_in_action takes effect

post-state value

-1 ∈ ℕ ≡ False

↳ where the n value violates TNO_1 (n ∈ ℕ)

```
--algorithm bridgeController_m0 {
 variable n = 0, i = 0;

 procedure ML_out() {
   ML_out_action: n := n + 1;
   return;
 }

 procedure ML_in() {
   ML_in_action: n := n - 1;
   return;
 }

 \* main program
 {
   loop: while (i < bound) {
       choice: either {
           ML_out_guard: if(TRUE) {
               ML_out_occurs: call ML_out();
           };
       }
       or {
           ML_in_guard: if(TRUE) {
               ML_in_occurs: call ML_in();
           };
       };
       progress: i := i + 1
   }
 }
```

more restrictive!

1. Add inv0_2 to TLC checker for invariant checking.

2. If it fails, study the error trace & state graph
   - ↳ fix the model accordingly
   - ↳ Re-translate PlusCal
   - ↳ Re-run TLC checker

practice more by injecting different errors to algorithm, and see how error traces can indicate fixes

3. Update the ML_out_guard if necessary
   - ↳ add deadlock_free to TLC checker for invariant checking.