Checking Liveness

Vesal Vojdani
System Modeling 2018

```
NextColor(c) == CASE c = "red" -> "green"
                  [] c = "green" -> "red"

(*--algorithm traffic
variables
  at_light = TRUE,
  light = "red";

process light = "light"
begin Cycle:
  while at_light do
    light := NextColor(light);
  end while;
end process;

process car = "car"
begin Drive:
  when light = "green";
  at_light := FALSE;
end process;

end algorithm;*)
```

Liveness properties

Do It!

- Create a model and add/check "Termination" under *Model Overview* > *What to Check?* > *Properties*.

- Make both processes fair. (Run Again! Note the difference!)

- Make the car process strongly fair.

Fairness

- Stuttering: we can always do nothing! Not an issue for safety properties (model checker must find faults).

- A weakly fair action will, if it stays enabled, eventually happen.

- A strongly fair action, if it’s repeatedly enabled, will eventually happen.

Mutual Exclusion

```
EXTENDS TLC, Integers
CONSTANT Threads
(*--algorithm dekker
variables flag = [t \in Threads |-> FALSE]

process thread \in Threads
begin
  P1: flag[self] := TRUE;
  P2: await \A t \in Threads \ {self}: flag[t] = FALSE;
  CS: skip;
  P3: flag[self] := FALSE;
end process;

end algorithm; *)
```

Mutual Exclusion

(First attempt)

Safety condition

- Only one thread should enter the critical section at a time.
- Write this invariant yourself and run the model checker with and without the await-condition.
- It should fail without the check.
- And will deadlock with the check!

```
process thread \in Threads
begin
  P1: flag[self] := TRUE;
  P2: await \A t \in Threads \ {self}: flag[t] = FALSE;
  CS: skip;
  P3: flag[self] := FALSE;
end process;
```

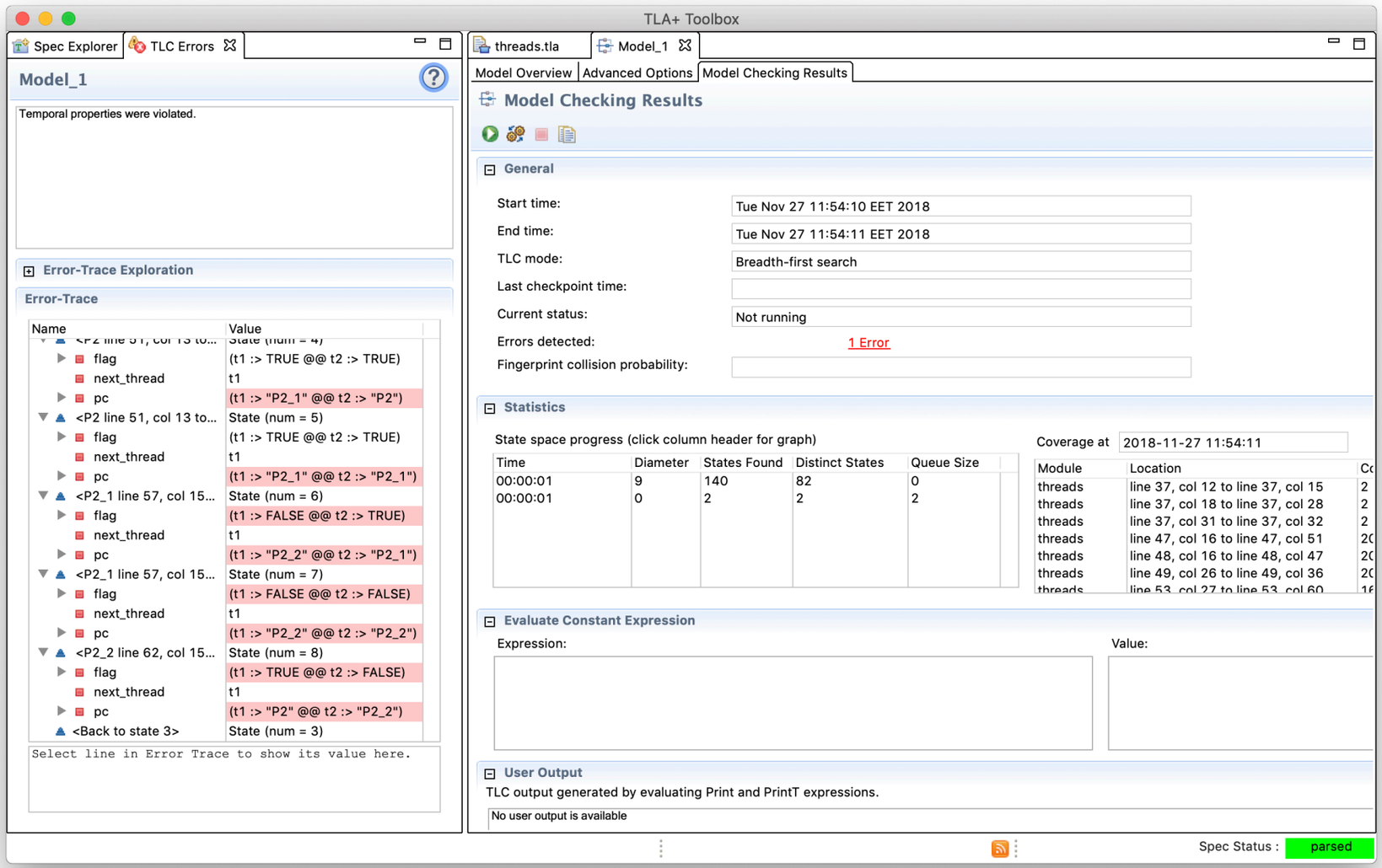
```
process thread \in Threads
begin
  P1: flag[self] := TRUE;
  P2: while \E t \in Threads \ {self}: flag[t] do
        P2_1: flag[self] := FALSE;
        P2_2: flag[self] := TRUE;
      end while;
  CS: skip;
  P3: flag[self] := FALSE;
end process;
```

Simple Fix!

Busy wait and signal our interest briefly.
Does this work?

TLC says yes!

- Okay, but today's topic is liveness.
- Let's ask another question: will both threads eventually make it to the critical section?
- Write this down as a TLA formula and run this "property"!
- Make sure your process is "fair"!
- Remember: [] means always and <> eventually.



Livelock!

(note the “back to state 3”)

```
variables
  flag = [t \in Threads |-> FALSE],
  next_thread \in Threads;

fair process thread \in Threads
begin
  P1: flag[self] := TRUE;
  P2: while \E t \in Threads \ {self}: flag[t] do
      P2_1:
        if next_thread /= self then
          P2_1_1: flag[self] := FALSE;
          P2_1_2: await next_thread = self;
          P2_1_3: flag[self] := TRUE;
        end if;
    end while;
  CS: skip;
  P3: with t \in Threads \ {self} do
        next_thread := t;
      end with;
  P4: flag[self] := FALSE;
end process;
```

Dekker's algorithm

Semi-realistic Use

Medium post by Hilllel Wayne

Using TLA+ at eSpark Learning

- Go to the Medium post and read the motivation & background.

- App scopes define configuration of apps to be installed.

- A device can enter a scope and will then receive necessary installs upon sync, but this can take time.

- Device synchronization happens at different rates and whenever devices are ready.

The state

- `AppScope` initially states whether app is already in scope. It will then count how many times entered.

- `Installs` allows devices to signal readiness for the update.

- `batch_pool` is the pool of devices that should enter the AppScope next.

```
variables
  AppScope \in [Devices -> {0, 1}];
  Installs \in [Devices -> BOOLEAN];
  batch_pool = {};
```

```
procedure ChangeAppScope()
begin
  Add:
    AppScope := [d \in Devices |->
        IF d \in batch_pool THEN AppScope[d] + 1
        ELSE AppScope[d]
     ];
  Clean:
    batch_pool := {};
    return;
end procedure;
```

Entire map is updated at once

ChangeAppScope

All devices in the batch_pool increased by one and then the pool is cleaned.

```
procedure ChangeAppScope()
begin
  Add:
    AppScope := [d \in Devices |->
        IF d \in batch_pool THEN AppScope[d] + 1
        ELSE AppScope[d]
     ];
  Clean:
    batch_pool := {};
  return;
end procedure;
```

```
fair process SyncDevice \in Devices
begin
  Sync:
    if Installs[self] then
        batch_pool := batch_pool \union {self};
    end if;
end process;
```

```
fair process TimeLoop = 0
begin
  Start:
    while TRUE do
      await batch_pool /= {};
      call ChangeAppScope();
    end while;
end process;
```

Processes

Check Safety

- Now we can check that devices do not enter the app scope more than once.
- We run this with a single device {d1}. Add your safety condition and disable deadlocks.
- Add the filter command from the article!
- Run again; then try with more devices.
- Continue reading on your own…