

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;
```

```
end algorithm;*)
```

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

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 l-> 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 $\langle \rangle$ eventually.

Model_1

Temporal properties were violated.

Error-Trace Exploration

Error-Trace

Name	Value
<P2 line 31, col 13 to...	State (num = 4)
flag	(t1 := TRUE @@ t2 := TRUE)
next_thread	t1
pc	(t1 := "P2_1" @@ t2 := "P2")
<P2 line 51, col 13 to...	State (num = 5)
flag	(t1 := TRUE @@ t2 := TRUE)
next_thread	t1
pc	(t1 := "P2_1" @@ t2 := "P2_1")
<P2_1 line 57, col 15...	State (num = 6)
flag	(t1 := FALSE @@ t2 := TRUE)
next_thread	t1
pc	(t1 := "P2_2" @@ t2 := "P2_1")
<P2_1 line 57, col 15...	State (num = 7)
flag	(t1 := FALSE @@ t2 := FALSE)
next_thread	t1
pc	(t1 := "P2_2" @@ t2 := "P2_2")
<P2_2 line 62, col 15...	State (num = 8)
flag	(t1 := TRUE @@ t2 := FALSE)
next_thread	t1
pc	(t1 := "P2" @@ t2 := "P2_2")
<Back to state 3>	State (num = 3)

Select line in Error Trace to show its value here.

Model Checking Results

General

Start time: Tue Nov 27 11:54:10 EET 2018
 End time: Tue Nov 27 11:54:11 EET 2018
 TLC mode: Breadth-first search
 Last checkpoint time:
 Current status: Not running
 Errors detected: **1 Error**
 Fingerprint collision probability:

Statistics

State space progress (click column header for graph)

Time	Diameter	States Found	Distinct States	Queue Size
00:00:01	9	140	82	0
00:00:01	0	2	2	2

Coverage at 2018-11-27 11:54:11

Module	Location	Cc
threads	line 37, col 12 to line 37, col 15	2
threads	line 37, col 18 to line 37, col 28	2
threads	line 37, col 31 to line 37, col 32	2
threads	line 47, col 16 to line 47, col 51	20
threads	line 48, col 16 to line 48, col 47	20
threads	line 49, col 26 to line 49, col 36	20
threads	line 53, col 27 to line 53, col 60	16

Evaluate Constant Expression

Expression: Value:

User Output

TLC output generated by evaluating Print and PrintT expressions.

No user output is available

Spec Status: **parsed**

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 Hillel 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 l->
      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...