# Apalache: symbolic model checker for TLA+

TLA+ tutorial at DISC 2021

Igor Konnov <igor@informal.systems>

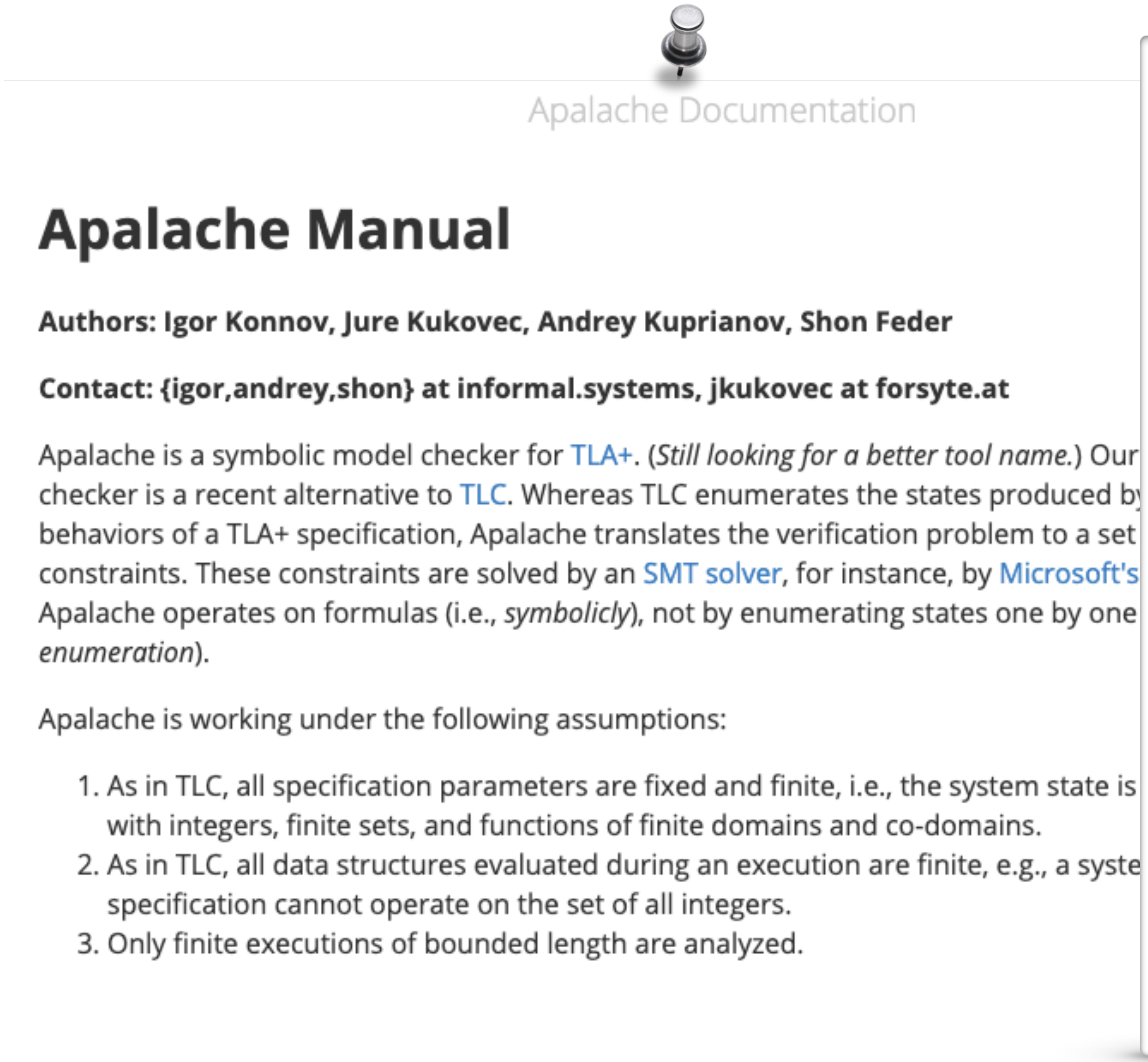
## Material for this talk

[ github.com/informalsystems/tla-apalache-workshop ]

specs,
commands,
longer talk

informal SYSTEMS

# apalache.informal.systems ⏎



Apalache Documentation

## Apalache Manual

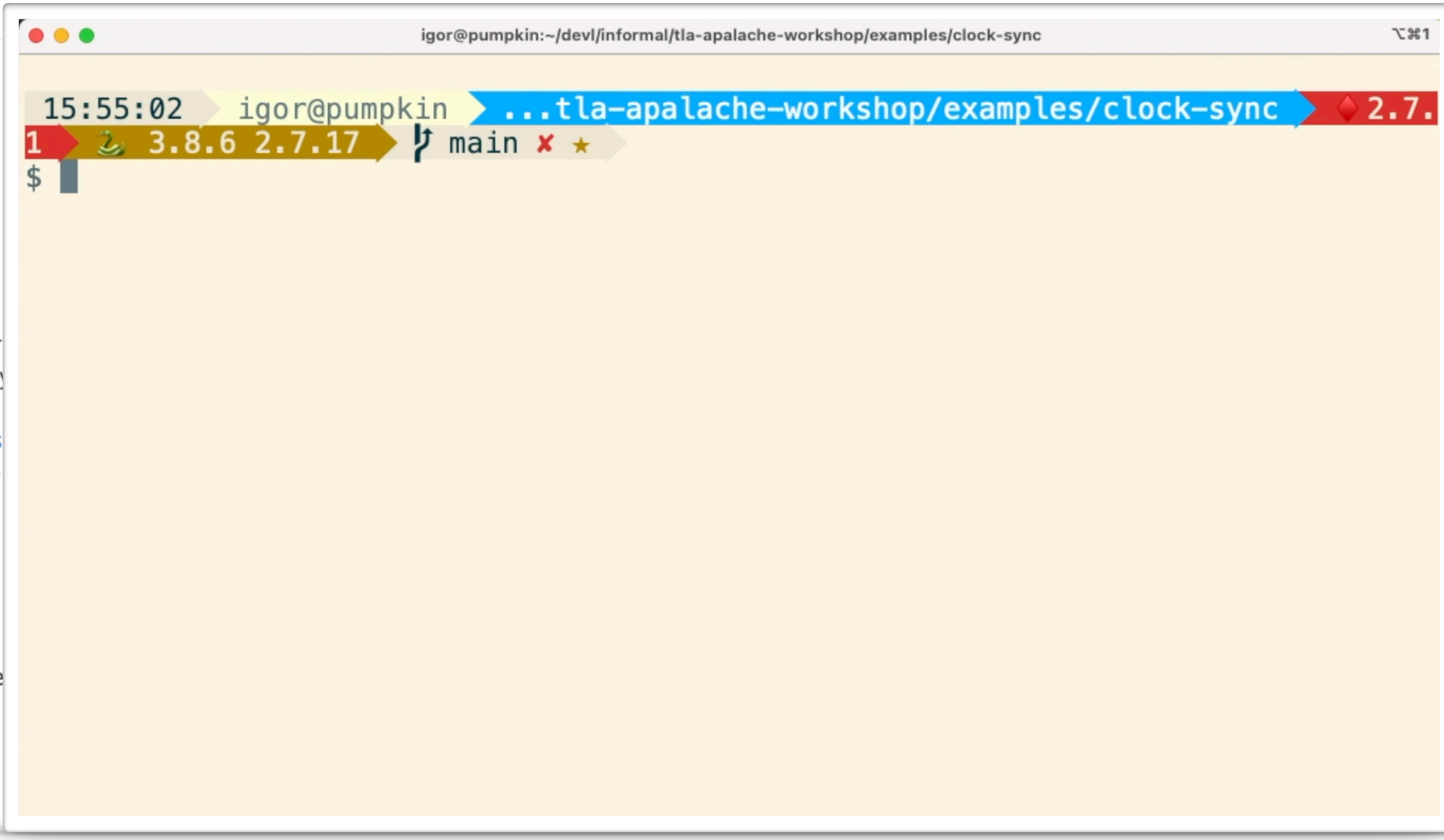
**Authors: Igor Konnov, Jure Kukovec, Andrey Kuprianov, Shon Feder**

**Contact: {igor,andrey,shon} at informal.systems, jkukovec at forsyte.at**

Apalache is a symbolic model checker for TLA+. (*Still looking for a better tool name.*) Our checker is a recent alternative to TLC. Whereas TLC enumerates the states produced by behaviors of a TLA+ specification, Apalache translates the verification problem to a set constraints. These constraints are solved by an SMT solver, for instance, by Microsoft's Apalache operates on formulas (i.e., *symbolicly*), not by enumerating states one by one *enumeration*).

Apalache is working under the following assumptions:

1. As in TLC, all specification parameters are fixed and finite, i.e., the system state is with integers, finite sets, and functions of finite domains and co-domains.
2. As in TLC, all data structures evaluated during an execution are finite, e.g., a syste specification cannot operate on the set of all integers.
3. Only finite executions of bounded length are analyzed.

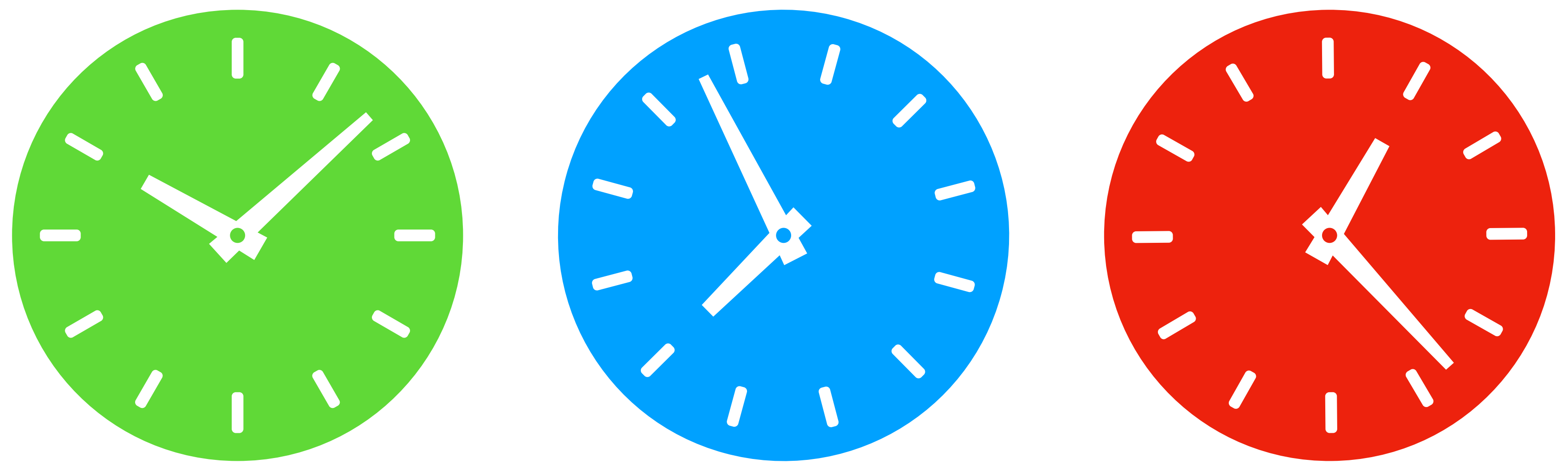


```
igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync        ⌥⌘1

 15:55:02  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync  2.7.
1  3.8.6 2.7.17  main ✘ ★
$ 
```

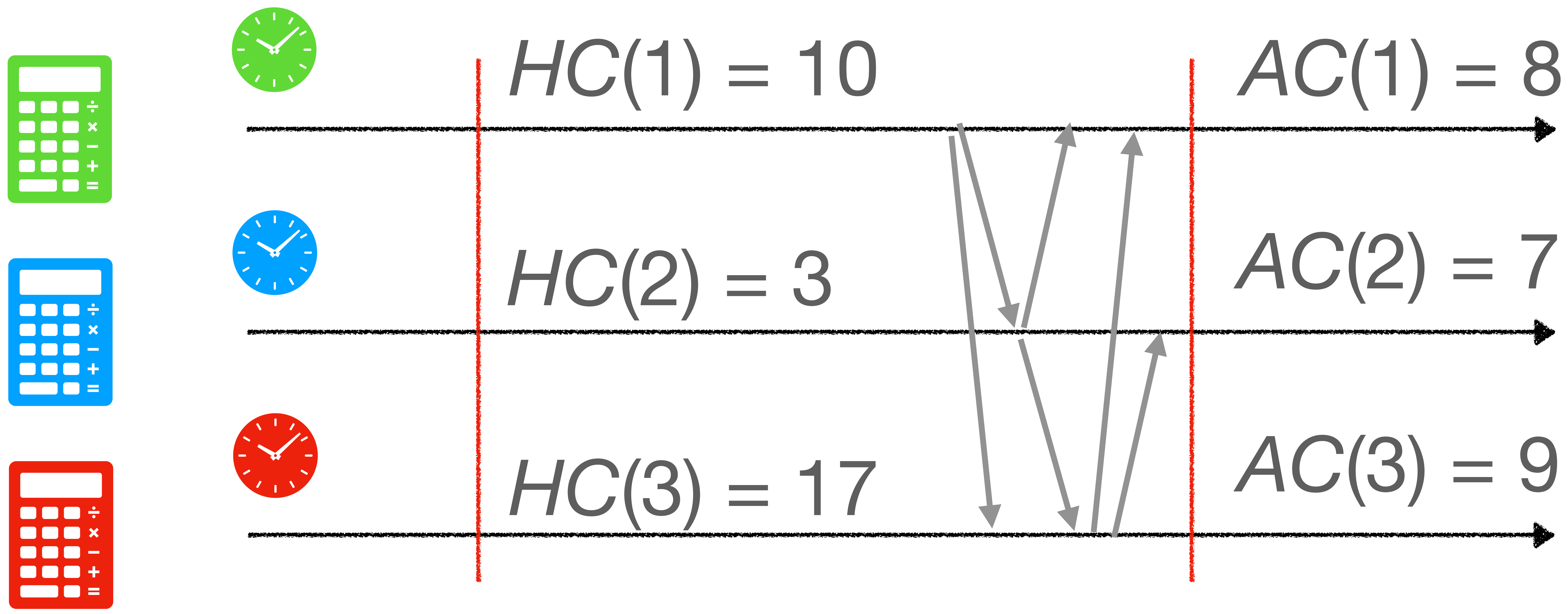
apalache.informal.systems/docs/

## Example:

## clock synchronization

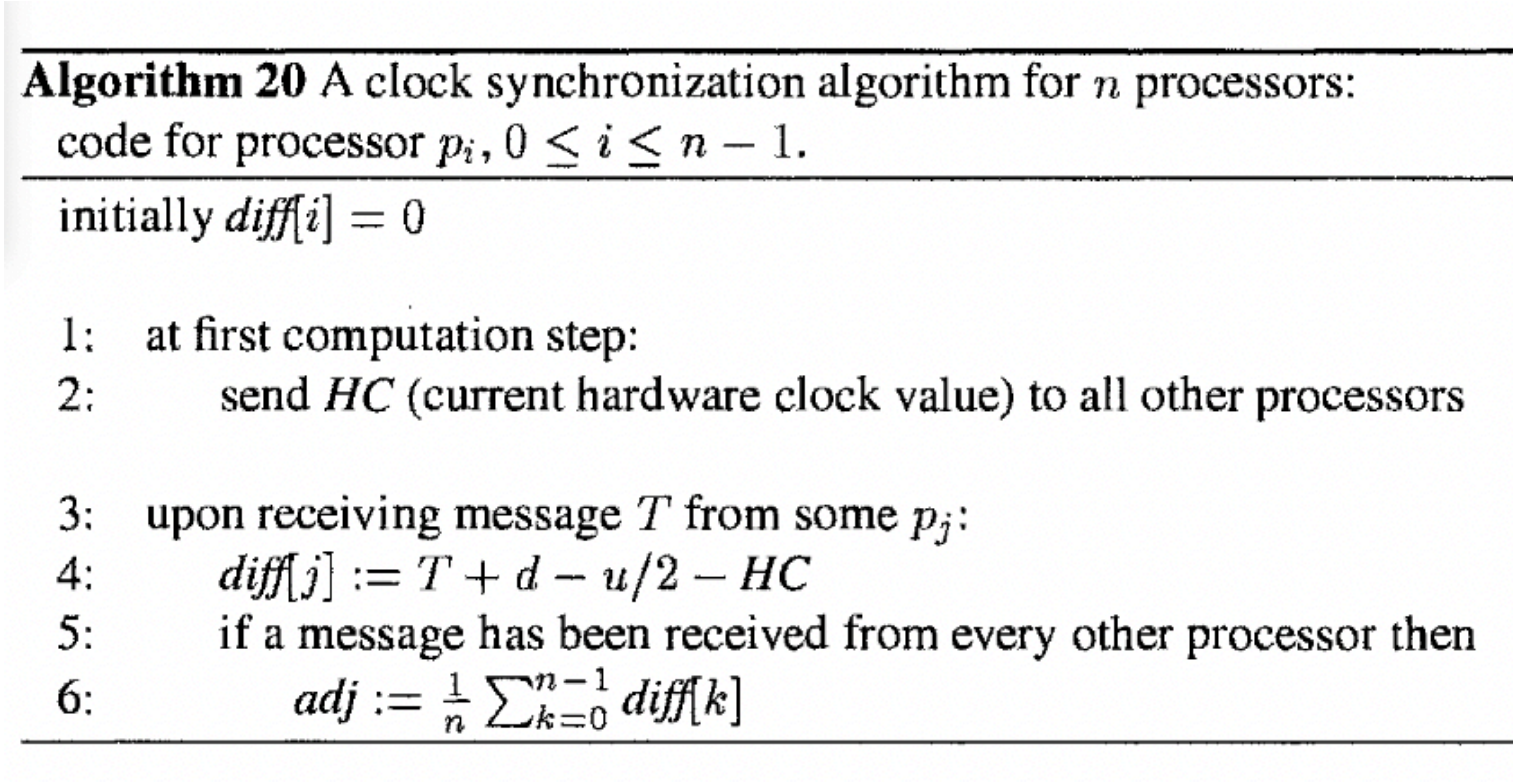


# Clock sync. problem



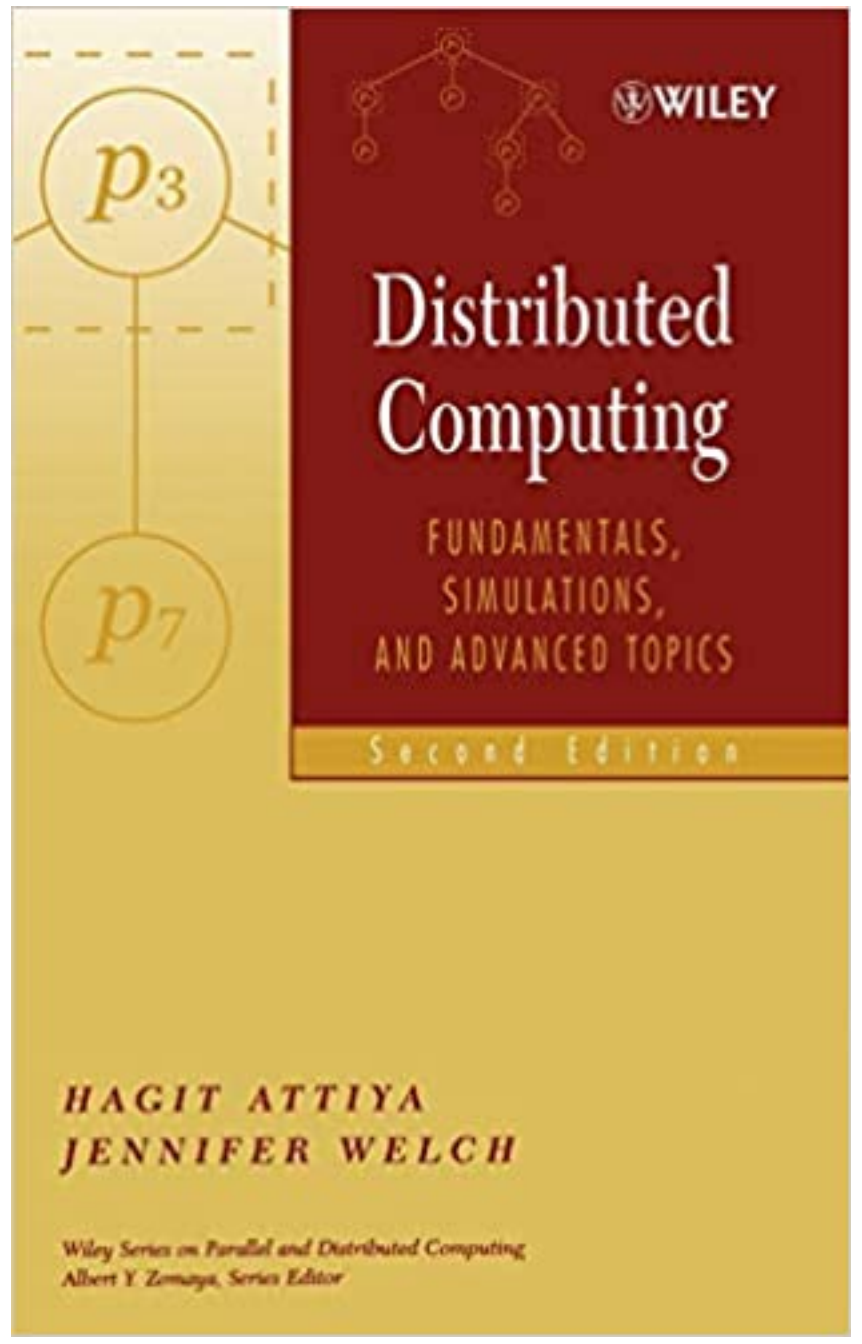
## Property: bounded clock skew

$$|AC(p) - AC(q)| \leq \epsilon \qquad \text{for } p, q \in Proc$$



**Algorithm 20** A clock synchronization algorithm for $n$ processors: code for processor $p_i$, $0 \le i \le n-1$.

```
initially $diff[i] = 0$

1:  at first computation step:
2:      send $HC$ (current hardware clock value) to all other processors

3:  upon receiving message $T$ from some $p_j$:
4:      $diff[j] := T + d - u/2 - HC$
5:      if a message has been received from every other processor then
6:          $adj := \frac{1}{n} \sum_{k=0}^{n-1} diff[k]$
```



WILEY

Distributed Computing

FUNDAMENTALS, SIMULATIONS, AND ADVANCED TOPICS

Second Edition

HAGIT ATTIYA

JENNIFER WELCH

Wiley Series on Parallel and Distributed Computing

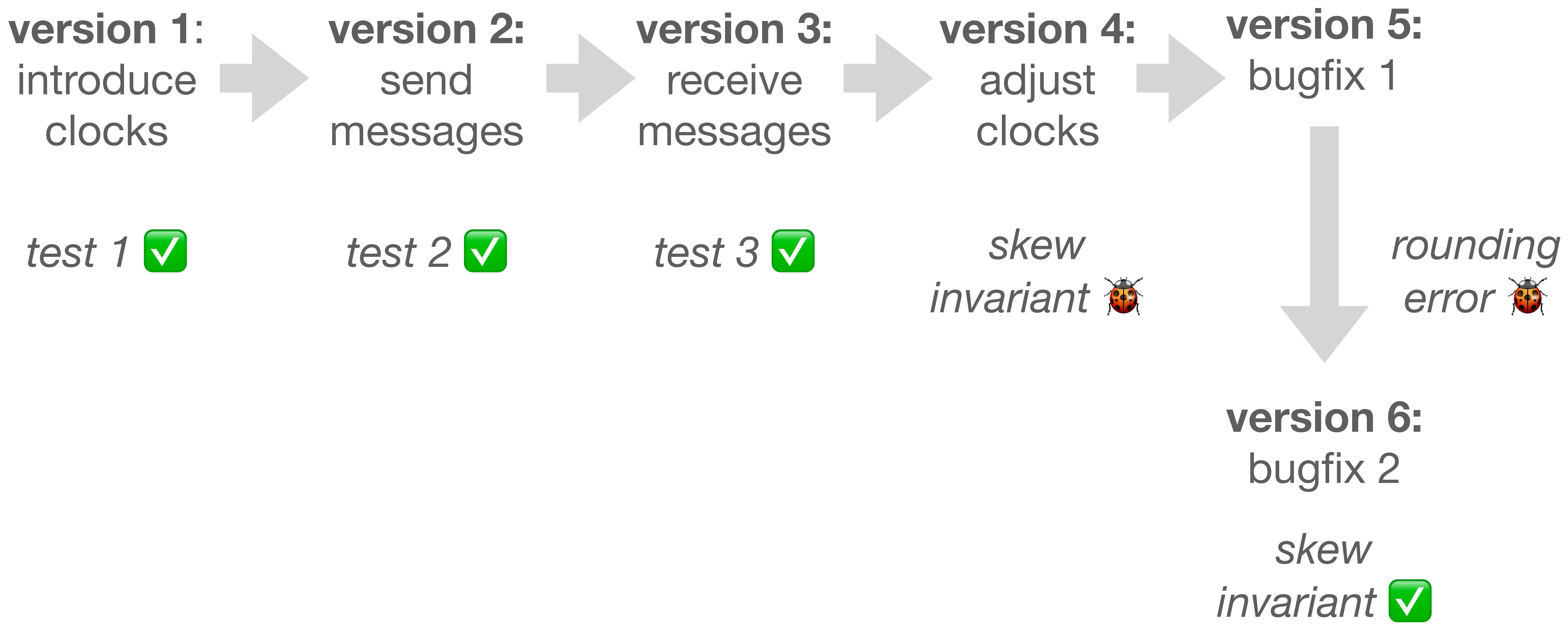
Albert Y. Zomaya, Series Editor

## How to turn 6 lines of pseudo-code

+ 9 pages of math text

## into 170 lines of TLA+

## Incremental spec writing

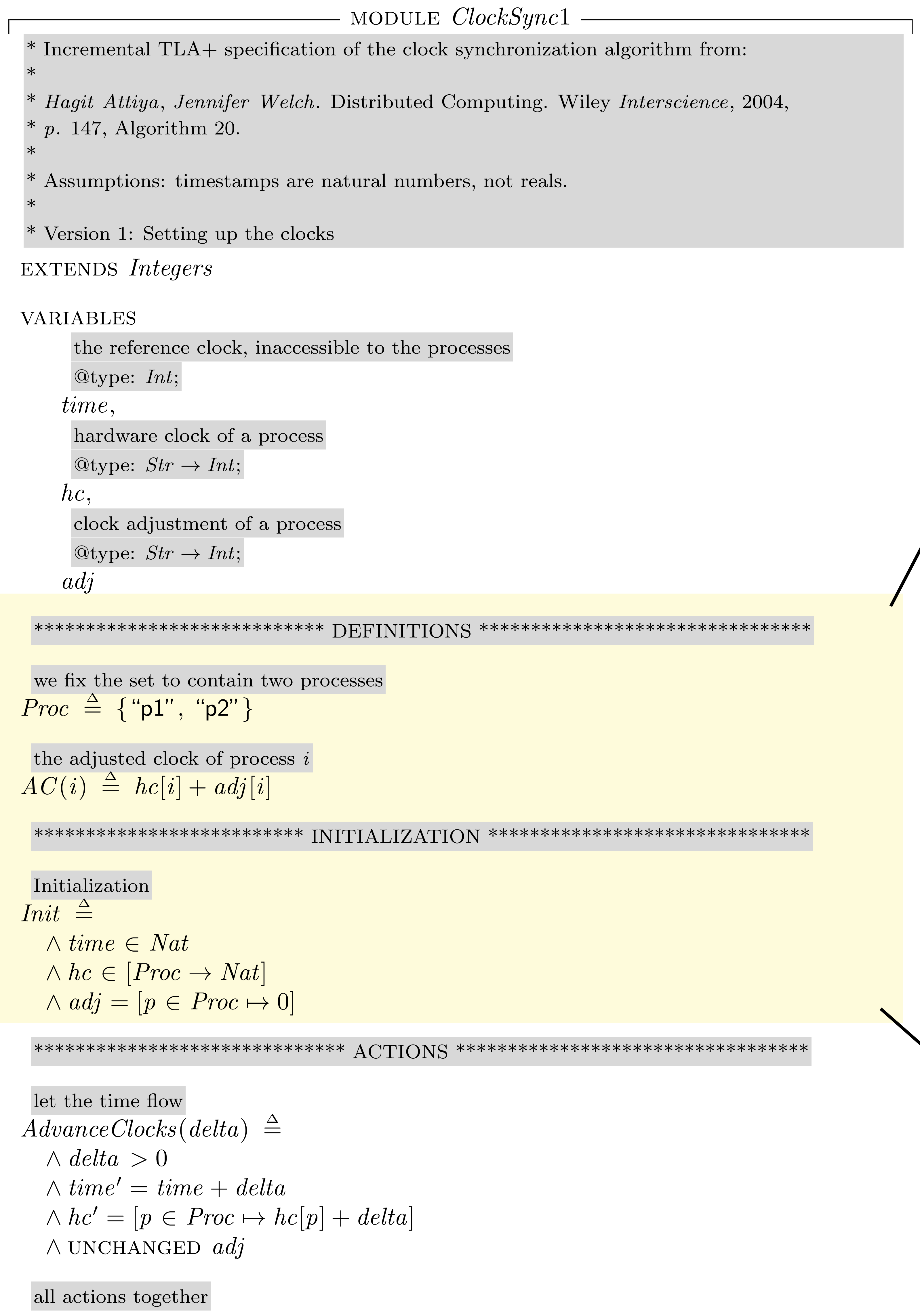


# Version 1: introduce clocks (1)

```
MODULE ClockSync1
* Incremental TLA+ specification of the clock synchronization algorithm from:
*
* Hagit Attiya, Jennifer Welch. Distributed Computing. Wiley Interscience, 2004,
* p. 147, Algorithm 20.
*
* Assumptions: timestamps are natural numbers, not reals.
*
* Version 1: Setting up the clocks
EXTENDS Integers

VARIABLES
      the reference clock, inaccessible to the processes
      @type: Int;
    time,
      hardware clock of a process
      @type: Str → Int;
    hc,
      clock adjustment of a process
      @type: Str → Int;
    adj
 **************************** DEFINITIONS ********************************

 we fix the set to contain two processes
Proc ≜ {"p1", "p2"}

 the adjusted clock of process i
AC(i) ≜ hc[i] + adj[i]

 ************************** INITIALIZATION ******************************

 Initialization
Init ≜
  ∧ time ∈ Nat
  ∧ hc ∈ [Proc → Nat]
  ∧ adj = [p ∈ Proc ↦ 0]

 ****************************** ACTIONS *********************************

 let the time flow
AdvanceClocks(delta) ≜
  ∧ delta > 0
  ∧ time′ = time + delta
  ∧ hc′ = [p ∈ Proc ↦ hc[p] + delta]
  ∧ UNCHANGED adj

 all actions together
```

```
EXTENDS Integers

VARIABLES
      the reference clock, inaccessible to the processes
      @type: Int;
    time,
      hardware clock of a process
      @type: Str → Int;
    hc,
      clock adjustment of a process
      @type: Str → Int;
    adj
```

# Version 1: introduce clocks (2)



MODULE *ClockSync1*

* Incremental TLA+ specification of the clock synchronization algorithm from:
*
* *Hagit Attiya*, *Jennifer Welch*. Distributed Computing. Wiley *Interscience*, 2004,
* *p*. 147, Algorithm 20.
*
* Assumptions: timestamps are natural numbers, not reals.
*
* Version 1: Setting up the clocks

EXTENDS *Integers*

VARIABLES

the reference clock, inaccessible to the processes
@type: *Int*;
*time*,
hardware clock of a process
@type: $Str \rightarrow Int$;
*hc*,
clock adjustment of a process
@type: $Str \rightarrow Int$;
*adj*

*************************** DEFINITIONS *******************************

we fix the set to contain two processes
$Proc \triangleq \{\text{“p1”}, \text{“p2”}\}$

the adjusted clock of process $i$
$AC(i) \triangleq hc[i] + adj[i]$

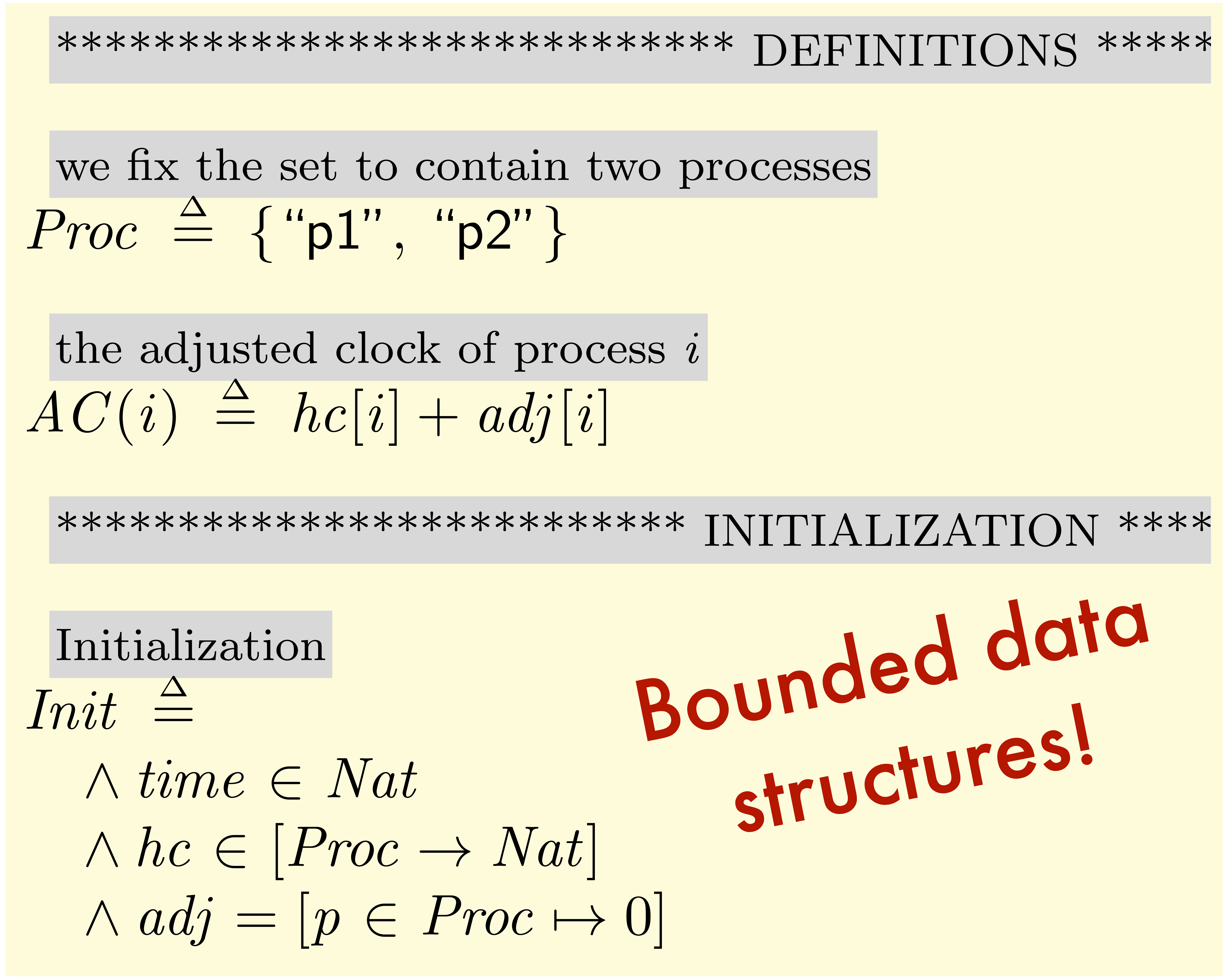
************************* INITIALIZATION *****************************

Initialization
$Init \triangleq$
$\land\ time \in Nat$
$\land\ hc \in [Proc \rightarrow Nat]$
$\land\ adj = [p \in Proc \mapsto 0]$

****************************** ACTIONS *********************************

let the time flow
$AdvanceClocks(delta) \triangleq$
$\land\ delta > 0$
$\land\ time' = time + delta$
$\land\ hc' = [p \in Proc \mapsto hc[p] + delta]$
$\land$ UNCHANGED *adj*

all actions together



******************************* DEFINITIONS *****

we fix the set to contain two processes
$Proc \triangleq \{\text{“p1”}, \text{“p2”}\}$

the adjusted clock of process $i$
$AC(i) \triangleq hc[i] + adj[i]$

**************************** INITIALIZATION ****

Initialization
$Init \triangleq$
$\land\ time \in Nat$
$\land\ hc \in [Proc \rightarrow Nat]$
$\land\ adj = [p \in Proc \mapsto 0]$

**Bounded data structures!**

# Version 1: introduce clocks (3)

MODULE *ClockSync1*

* Incremental TLA+ specification of the clock synchronization algorithm from:
*
* *Hagit Attiya*, *Jennifer Welch*. Distributed Computing. Wiley *Interscience*, 2004,
* *p*. 147, Algorithm 20.
*
* Assumptions: timestamps are natural numbers, not reals.
*
* Version 1: Setting up the clocks

EXTENDS *Integers*

VARIABLES

the reference clock, inaccessible to the processes
@type: *Int*;
$time$,
hardware clock of a process
@type: $Str \rightarrow Int$;
$hc$,
clock adjustment of a process
@type: $Str \rightarrow Int$;
$adj$

****************************** DEFINITIONS **********************************

we fix the set to contain two processes
$Proc \triangleq \{\text{"p1"}, \text{"p2"}\}$

the adjusted clock of process $i$
$AC(i) \triangleq hc[i] + adj[i]$

**************************** INITIALIZATION ********************************

Initialization
$Init \triangleq$
$\land\ time \in Nat$
$\land\ hc \in [Proc \rightarrow Nat]$
$\land\ adj = [p \in Proc \mapsto 0]$

******************************** ACTIONS ************************************

let the time flow
$AdvanceClocks(delta) \triangleq$
$\land\ delta > 0$
$\land\ time' = time + delta$
$\land\ hc' = [p \in Proc \mapsto hc[p] + delta]$
$\land$ UNCHANGED $adj$

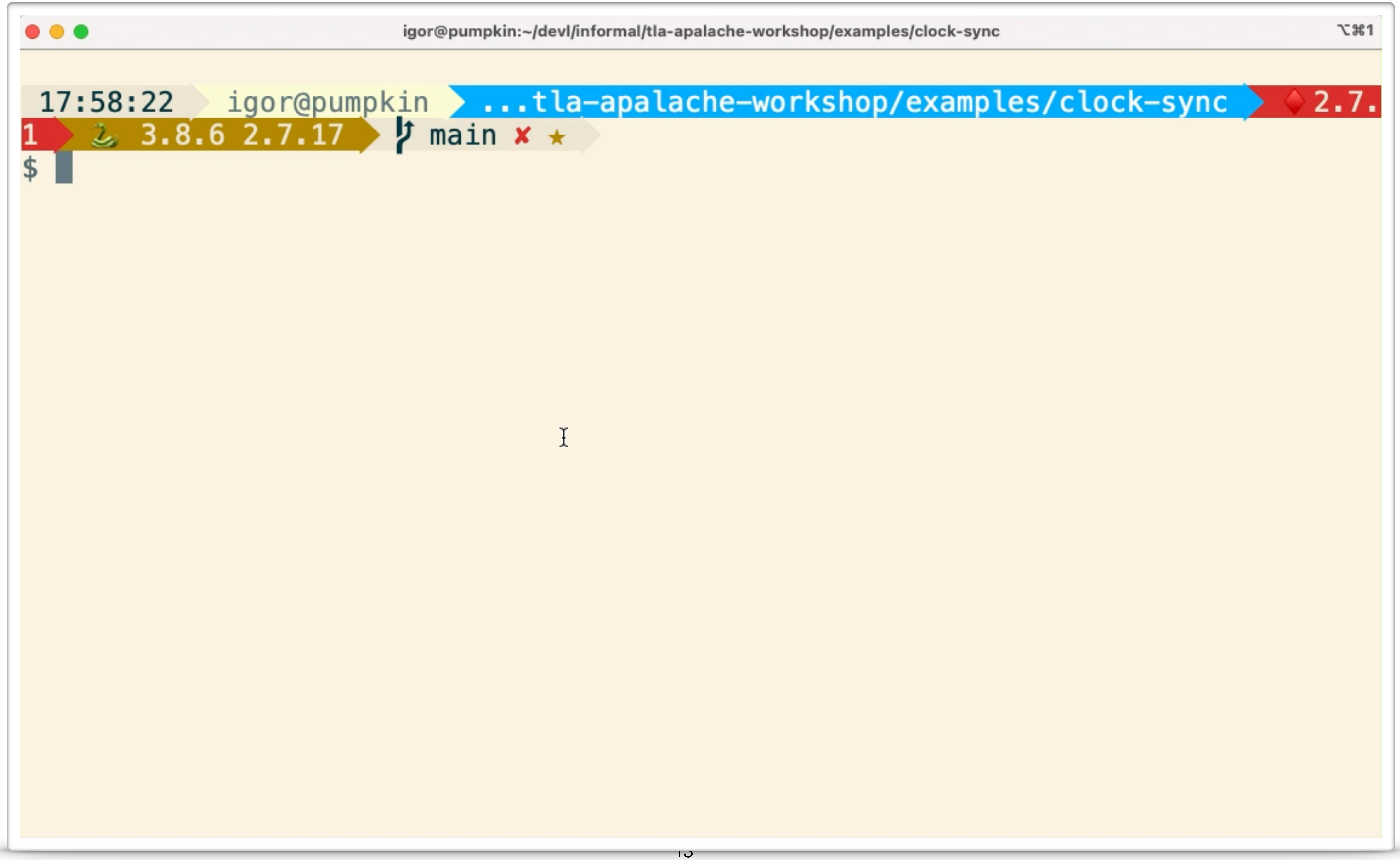
all actions together

******************************** ACTIONS *******

let the time flow
$AdvanceClocks(delta) \triangleq$
$\land\ delta > 0$
$\land\ time' = time + delta$
$\land\ hc' = [p \in Proc \mapsto hc[p] + delta]$
$\land$ UNCHANGED $adj$

all actions together
$Next \triangleq$
$\exists\, delta \in Int :$
$AdvanceClocks(delta)$

## Run apalache



```
igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync
17:58:22  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync  2.7.
1  3.8.6 2.7.17  main ✘ ★
$
```

## bounded model checking explained

## Symbolic execution

| Frame 0 | Frame 1 | Frame 2 | ... | Frame 10 |
|---|---|---|---|---|
| $time_0 = 0$ | $time_1 = time_0 + \delta_1$ | $time_2 = time_1 + \delta_2$ | | $time_{10} = time_9 + \delta_{10}$ |
| $hc[1]_0 = c_1$ <br> $hc[2]_0 = c_2$ | $hc[1]_1 = hc[1]_0 + \delta_1$ <br> $hc[2]_1 = hc[2]_0 + \delta_1$ | $hc[1]_2 = hc[1]_1 + \delta_2$ <br> $hc[2]_2 = hc[2]_1 + \delta_2$ | | $hc[1]_{10} = hc[1]_9 + \delta_{10}$ <br> $hc[2]_{10} = hc[2]_9 + \delta_{10}$ |
| $adj[1]_0 = 0$ <br> $adj[2]_0 = 0$ | $adj_1 = adj_0$ | $adj_2 = adj_1$ | | $adj_{10} = adj_9$ |

A *frame* represents multiple concrete states (symbolically)

## Bounded model checking (0 steps)

$$time_0 = 0$$
$$\wedge$$
$$hc[1]_0 = c_1$$
$$\wedge$$
$$hc[2]_0 = c_2$$
$$\wedge$$
$$adj[1]_0 = 0$$
$$\wedge$$
$$adj[2]_0 = 0$$

**Apalache: satisfiable?**

**Z3: Yes, here is a model**

**Apalache: one more step**

## Bounded model checking (1 step)

$$time_0 = 0 \land hc[1]_0 = c_1 \land hc[2]_0 = c_2 \land adj[1]_0 = 0 \land adj[2]_0 = 0$$

$$\land$$

$$time_1 = time_0 + \delta_1 \land hc[1]_1 = hc[1]_0 + \delta_1 \land hc[2]_1 = hc[2]_0 + \delta_1 \land adj_1 = adj_0$$

**Apalache: satisfiable?**

**Z3: Yes, here is a model**

**Apalache: one more step**

informal SYSTEMS

# Bounded model checking (2 steps)

$$\begin{array}{ccccc} time_0 = 0 & & time_1 = time_0 + \delta_1 & & time_2 = time_1 + \delta_2 \\ \wedge & & \wedge & & \wedge \\ hc[1]_0 = c_1 & & hc[1]_1 = hc[1]_0 + \delta_1 & & hc[1]_2 = hc[1]_1 + \delta_2 \\ \wedge & \wedge & \wedge & \wedge & \wedge \\ hc[2]_0 = c_2 & & hc[2]_1 = hc[2]_0 + \delta_1 & & hc[2]_2 = hc[2]_1 + \delta_2 \\ \wedge & & \wedge & & \wedge \\ adj[1]_0 = 0 & & adj_1 = adj_0 & & adj_2 = adj_1 \\ \wedge & & & & \\ adj[2]_0 = 0 & & & & \end{array}$$

**A: SAT?**

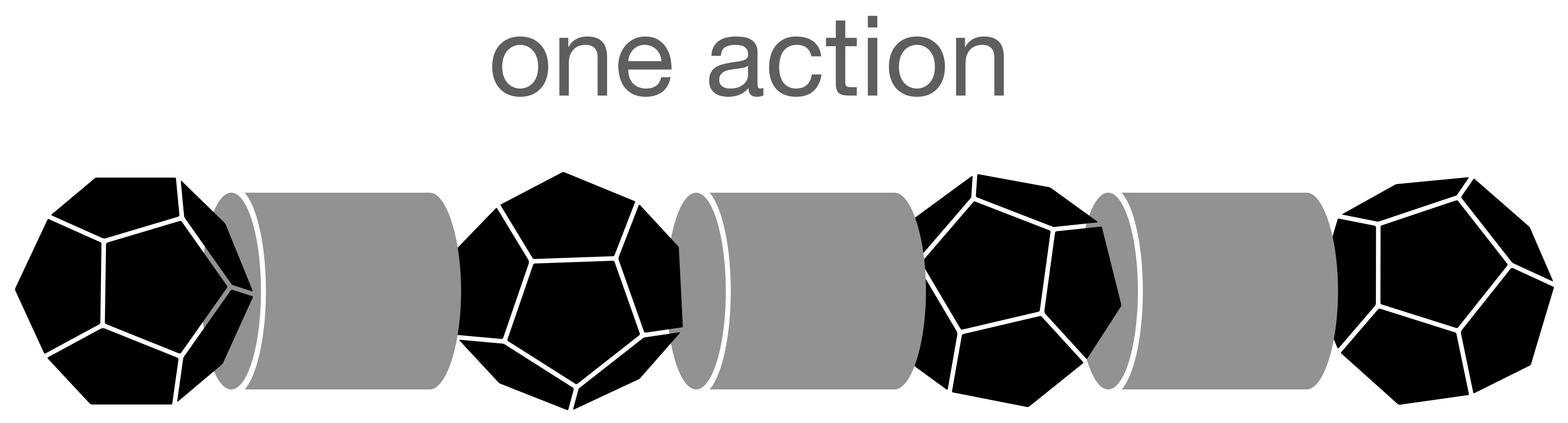
**Z3: yes**

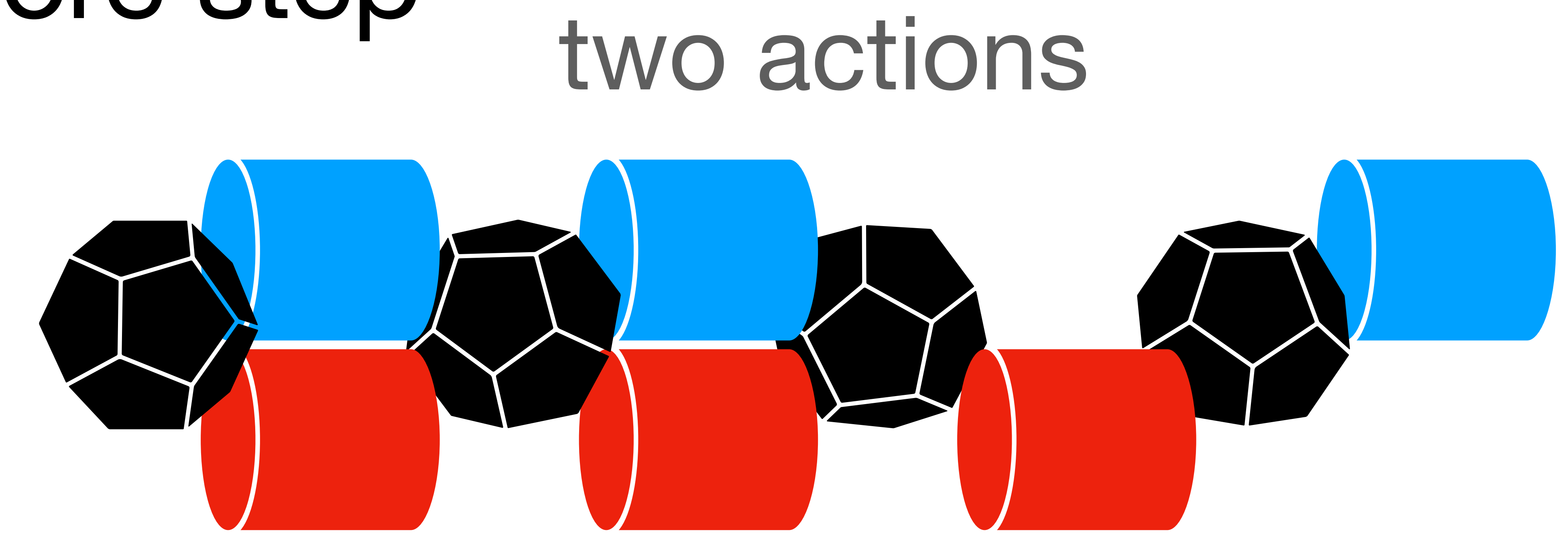
**A: go on**

## Symbolic exploration

By default, Apalache:

- finds enabled actions, e.g., *AdvanceClocks*
- adds non-deterministic choice of one enabled action
- extends the symbolic execution by one more step
- until the bound is reached, e.g., 10 steps





## Symbolic vs. concrete executions

$time_0 = 0$ $\qquad time_1 = time_0 + \delta_1 \qquad time_2 = time_1 + \delta_2$

$\wedge \qquad \wedge \qquad \wedge$

$hc[1]_0 = c_1$

$\wedge$

$hc[2]_0 = c_2$

$\wedge$

$adj[1]_0 = 0$

$\wedge$

$adj[2]_0 = 0$

$adj_1 = adj_0 \qquad adj_2 = adj_1$

infinitely many solutions:
infinite number of states and
executions!

## Checking an invariant (candidate)

the adjusted clock of process $i$

$$AC(i) \triangleq hc[i] + adj[i]$$

$$NaiveSkewInv \triangleq \forall p, q \in Proc : AC(p) = AC(q)$$

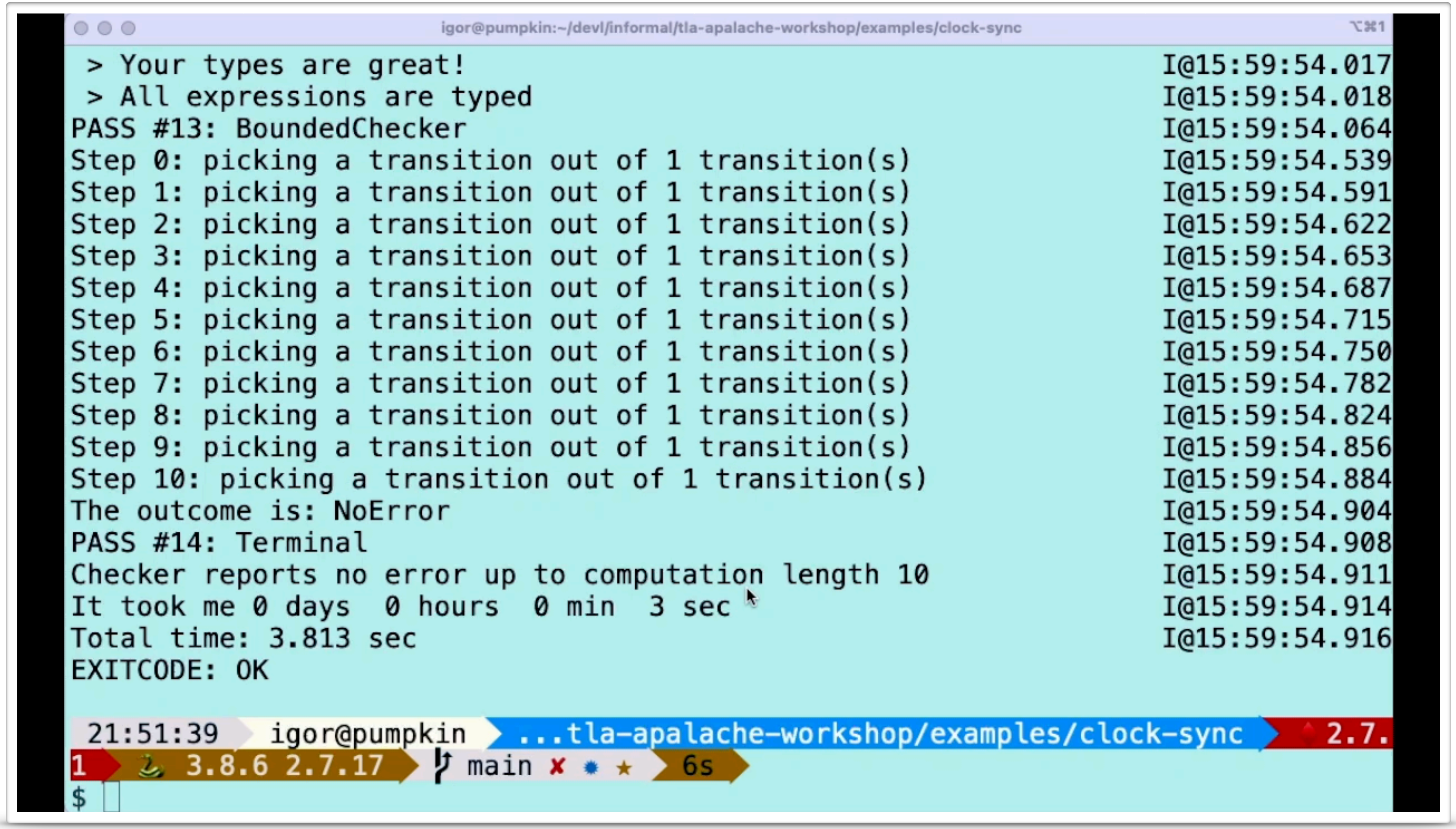
$$time_0 = 0 \land hc[1]_0 = c_1 \land hc[2]_0 = c_2 \land adj[1]_0 = 0 \land adj[2]_0 = 0$$

$$\land \; hc[p]_0 + adj[p]_0 \neq hc[q]_0 + adj[q]_0$$

**Apalache: SAT?**

**Z3: yes**

**A: error!** 🐞



```
igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync                    ⌥⌘1
 > Your types are great!                                          I@15:59:54.017
 > All expressions are typed                                      I@15:59:54.018
PASS #13: BoundedChecker                                          I@15:59:54.064
Step 0: picking a transition out of 1 transition(s)               I@15:59:54.539
Step 1: picking a transition out of 1 transition(s)               I@15:59:54.591
Step 2: picking a transition out of 1 transition(s)               I@15:59:54.622
Step 3: picking a transition out of 1 transition(s)               I@15:59:54.653
Step 4: picking a transition out of 1 transition(s)               I@15:59:54.687
Step 5: picking a transition out of 1 transition(s)               I@15:59:54.715
Step 6: picking a transition out of 1 transition(s)               I@15:59:54.750
Step 7: picking a transition out of 1 transition(s)               I@15:59:54.782
Step 8: picking a transition out of 1 transition(s)               I@15:59:54.824
Step 9: picking a transition out of 1 transition(s)               I@15:59:54.856
Step 10: picking a transition out of 1 transition(s)              I@15:59:54.884
The outcome is: NoError                                           I@15:59:54.904
PASS #14: Terminal                                                I@15:59:54.908
Checker reports no error up to computation length 10              I@15:59:54.911
It took me 0 days  0 hours  0 min  3 sec                          I@15:59:54.914
Total time: 3.813 sec                                             I@15:59:54.916
EXITCODE: OK

21:51:39  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync  2.7.
1  3.8.6 2.7.17  main ✘ ● ★  6s
$
```

# Writing basic tests

MODULE $MC\_ClockSync1$

VARIABLES

the reference clock, inaccessible to the processes

@type: $Int$;

$time$,

hardware clock of a process

@type: $Str \rightarrow Int$;

$hc$,

clock adjustment of a process

@type: $Str \rightarrow Int$;

$adj$

INSTANCE $ClockSync1$

test that the clocks are non-decreasing

$Test1\_Init \triangleq$

$\land\ time \in Nat$

$\land\ hc \in [Proc \rightarrow Nat]$

$\land\ adj \in [Proc \rightarrow Int]$

$Test1\_Next \triangleq$

$\exists\, delta \in Int :$

$AdvanceClocks(delta)$

$Test1\_Inv \triangleq$

$\land\ time' \geq time$

$\land\ \forall\, p \in Proc : hc'[p] \geq hc[p]$

INSTANCE $ClockSync1$

test that the clocks are non-decreasing

$Test1\_Init \triangleq$

$\land\ time \in Nat$

$\land\ hc \in [Proc \rightarrow Nat]$

$\land\ adj \in [Proc \rightarrow Int]$

$Test1\_Next \triangleq$

$\exists\, delta \in Int :$

$AdvanceClocks(delta)$

$Test1\_Inv \triangleq$

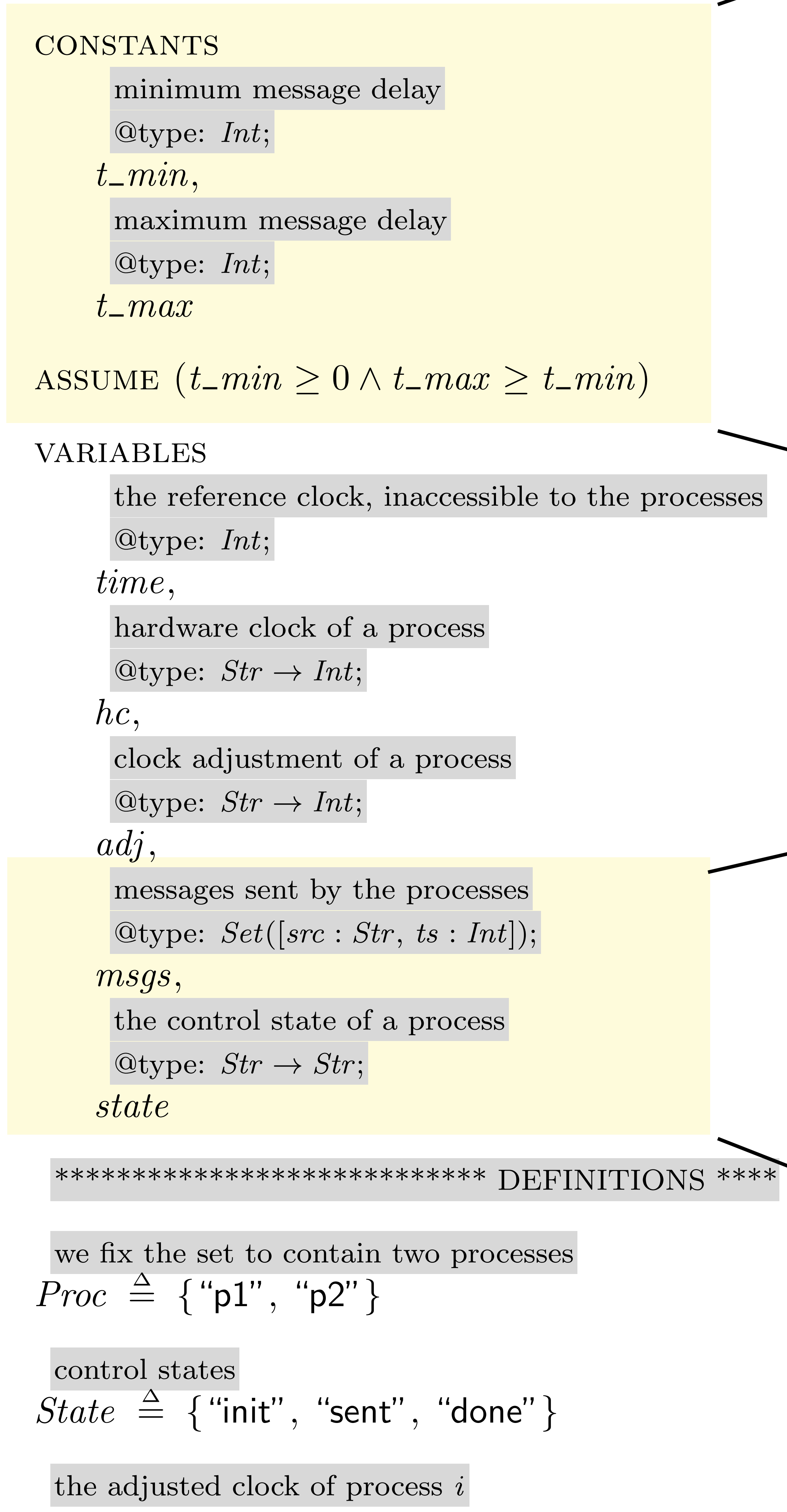
$\land\ time' \geq time$

$\land\ \forall\, p \in Proc : hc'[p] \geq hc[p]$

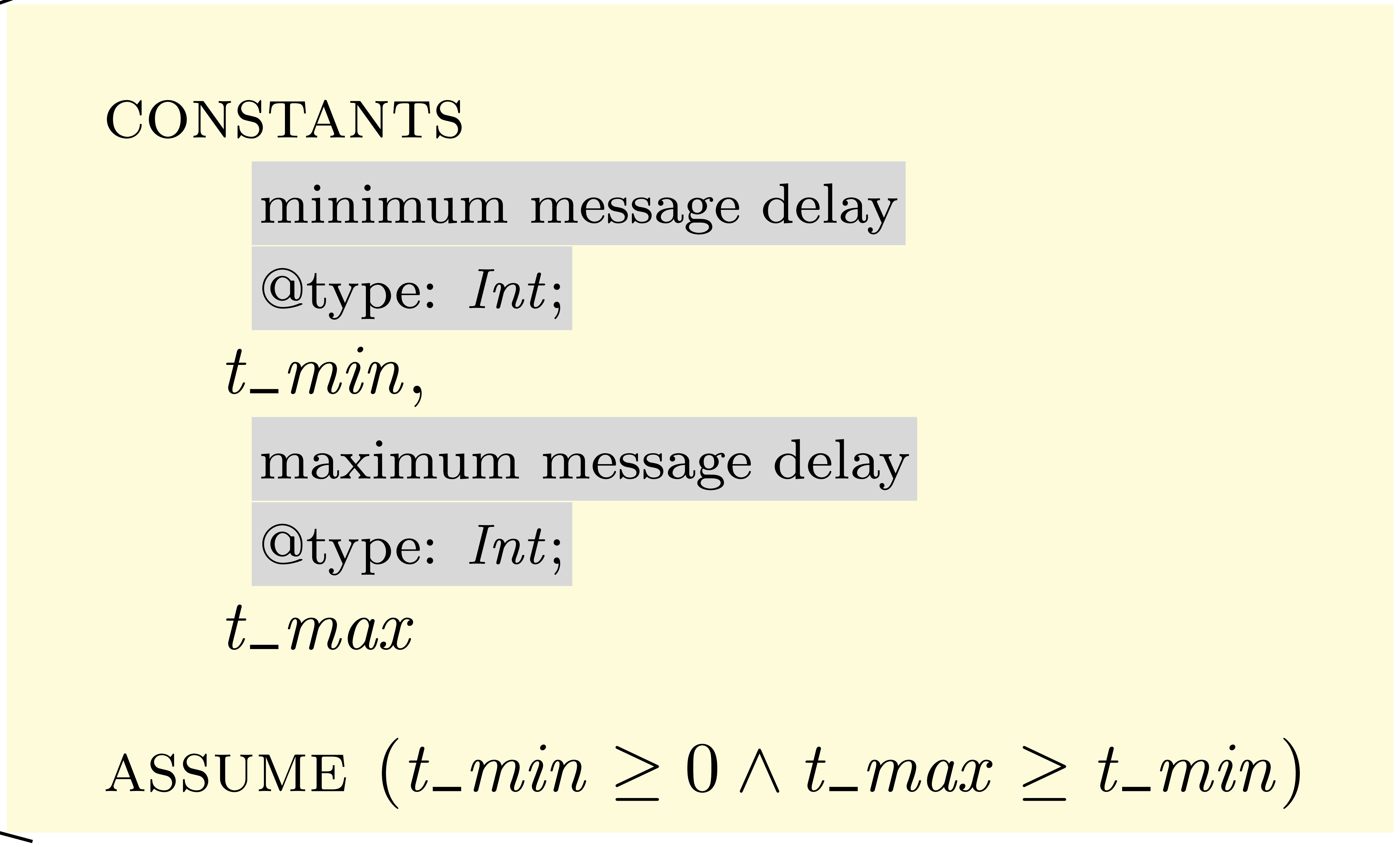
informal SYSTEMS

## version 2: sending messages

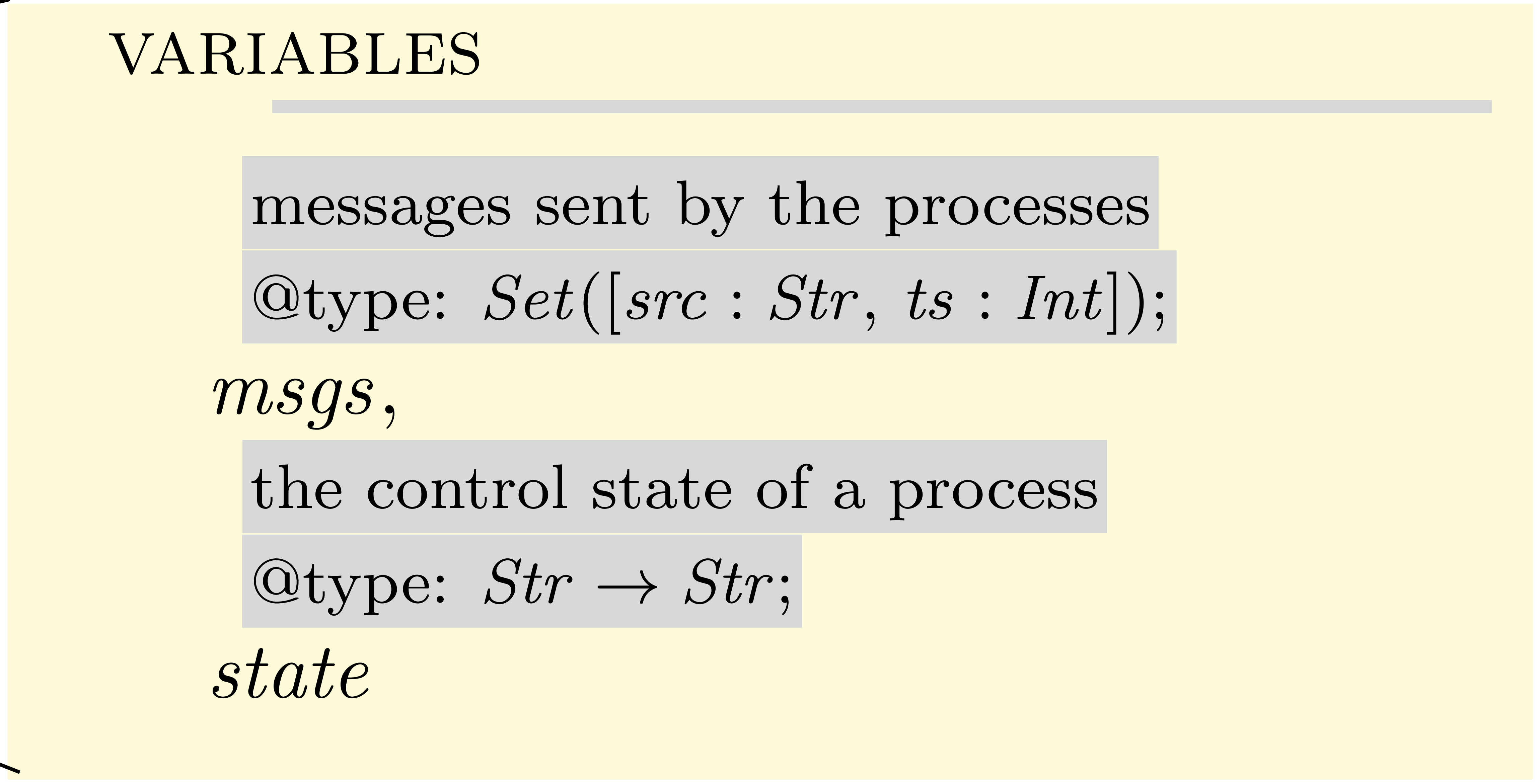
# States



```
CONSTANTS
    minimum message delay
    @type: Int;
  t_min,
    maximum message delay
    @type: Int;
  t_max

ASSUME (t_min ≥ 0 ∧ t_max ≥ t_min)

VARIABLES
    the reference clock, inaccessible to the processes
    @type: Int;
  time,
    hardware clock of a process
    @type: Str → Int;
  hc,
    clock adjustment of a process
    @type: Str → Int;
  adj,
    messages sent by the processes
    @type: Set([src : Str, ts : Int]);
  msgs,
    the control state of a process
    @type: Str → Str;
  state

 *************************** DEFINITIONS ****

 we fix the set to contain two processes
Proc ≜ {"p1", "p2"}

 control states
State ≜ {"init", "sent", "done"}

 the adjusted clock of process i
```

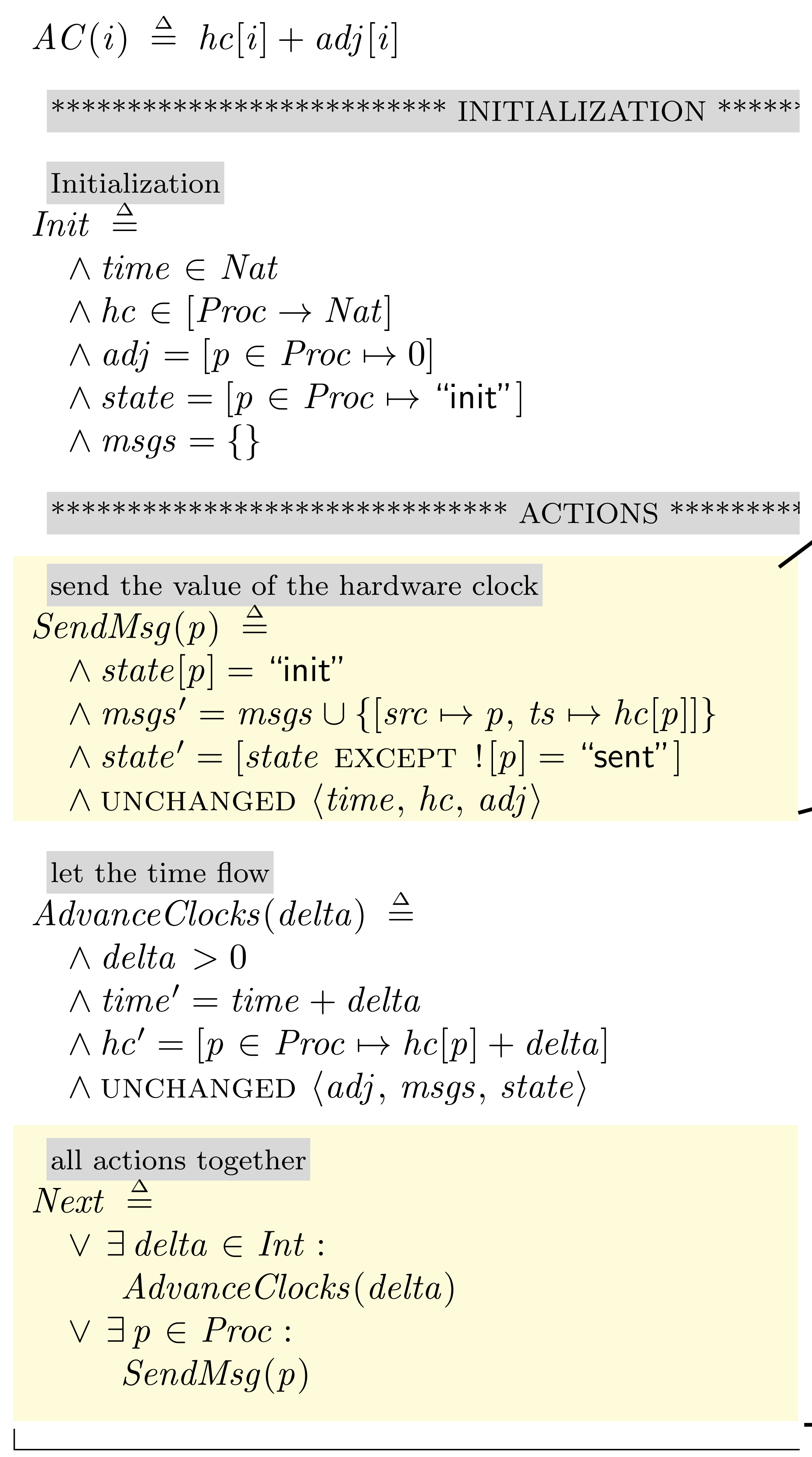


```
CONSTANTS
    minimum message delay
    @type: Int;
  t_min,
    maximum message delay
    @type: Int;
  t_max

ASSUME (t_min ≥ 0 ∧ t_max ≥ t_min)
```



```
VARIABLES

    messages sent by the processes
    @type: Set([src : Str, ts : Int]);
  msgs,
    the control state of a process
    @type: Str → Str;
  state
```

# Action SendMsg



$AC(i) \triangleq hc[i] + adj[i]$

************************* INITIALIZATION ******

Initialization

$Init \triangleq$

$\land\ time \in Nat$

$\land\ hc \in [Proc \rightarrow Nat]$

$\land\ adj = [p \in Proc \mapsto 0]$

$\land\ state = [p \in Proc \mapsto \text{"init"}]$

$\land\ msgs = \{\}$

***************************** ACTIONS *********

send the value of the hardware clock

$SendMsg(p) \triangleq$

$\land\ state[p] = \text{"init"}$

$\land\ msgs' = msgs \cup \{[src \mapsto p,\ ts \mapsto hc[p]]\}$

$\land\ state' = [state \text{ EXCEPT } ![p] = \text{"sent"}]$

$\land\ \text{UNCHANGED } \langle time,\ hc,\ adj \rangle$

let the time flow

$AdvanceClocks(delta) \triangleq$

$\land\ delta > 0$

$\land\ time' = time + delta$

$\land\ hc' = [p \in Proc \mapsto hc[p] + delta]$

$\land\ \text{UNCHANGED } \langle adj,\ msgs,\ state \rangle$

all actions together

$Next \triangleq$

$\lor\ \exists\, delta \in Int :$

$\quad AdvanceClocks(delta)$

$\lor\ \exists\, p \in Proc :$

$\quad SendMsg(p)$

send the value of the hardware clock

$SendMsg(p) \triangleq$

$\land\ state[p] = \text{"init"}$

$\land\ msgs' = msgs \cup \{[src \mapsto p,\ ts \mapsto hc[p]]\}$

$\land\ state' = [state \text{ EXCEPT } ![p] = \text{"sent"}]$

$\land\ \text{UNCHANGED } \langle time,\ hc,\ adj \rangle$

all actions together

$Next \triangleq$

$\lor\ \exists\, delta \in Int :$

$\quad AdvanceClocks(delta)$

$\lor\ \exists\, p \in Proc :$

$\quad SendMsg(p)$

# Testing 2.1

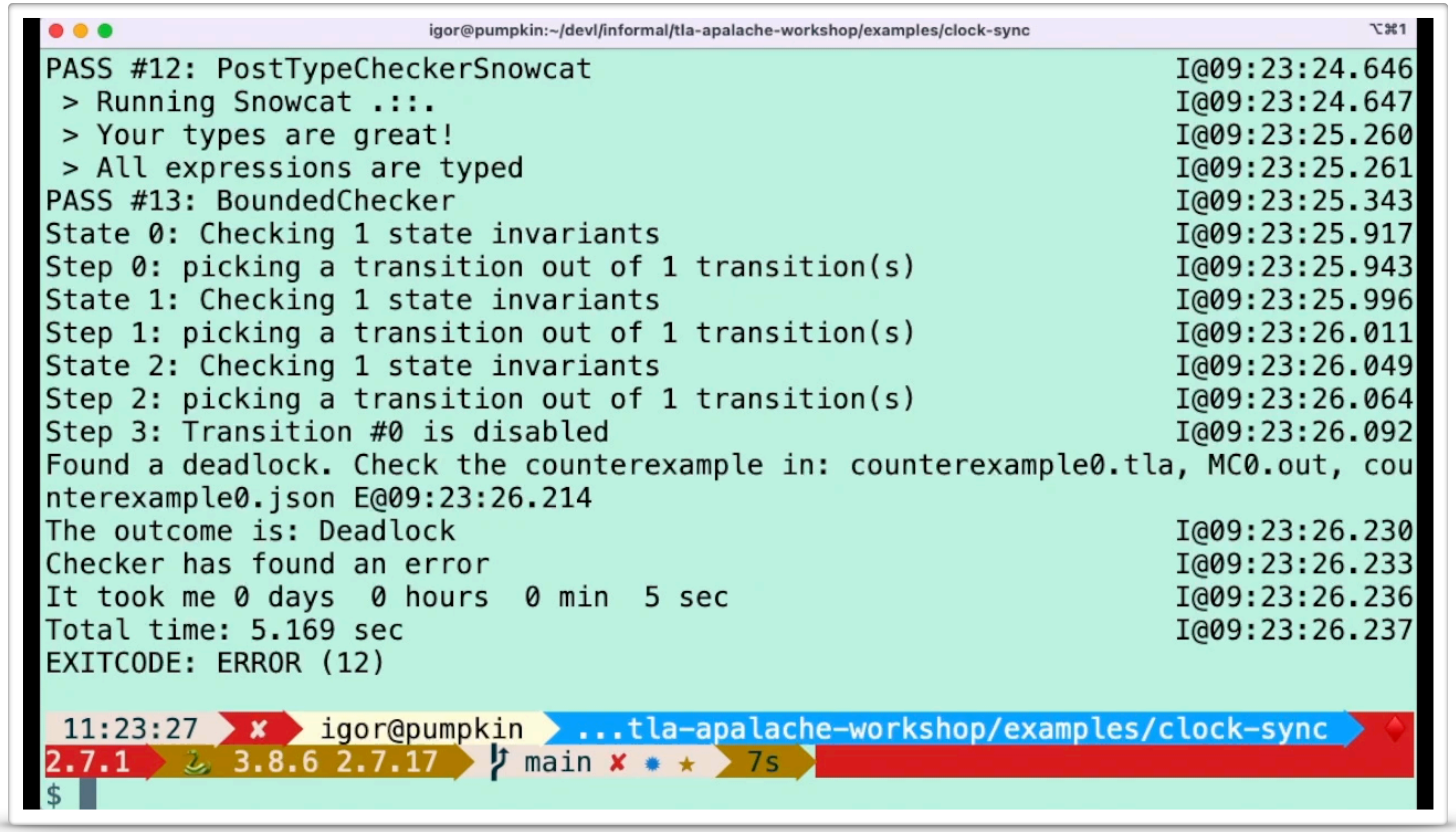
```
─────────────── MODULE MC_ClockSync2 ──
t_min ≜ 17
t_max ≜ 91

VARIABLES
    the reference clock, inaccessible to the processes
    @type: Int;
  time,
    hardware clock of a process
    @type: Str → Int;
  hc,
    clock adjustment of a process
    @type: Str → Int;
  adj,
    messages sent by the processes
    @type: Set([src : Str, ts : Int]);
  msgs,
    the control state of a process
    @type: Str → Str;
  state

INSTANCE ClockSync2

 like TypeOK, but used only in initialization
TypeInit ≜
  ∧ time ∈ Nat
  ∧ hc ∈ [Proc → Nat]
  ∧ adj ∈ [Proc → Int]
  ∧ state ∈ [Proc → State]
  ∧ ∃ t   ∈ [Proc → Int] :
     msgs ∈ SUBSET {[src ↦ p, ts ↦ t[p]] : p ∈ Proc}

 test that the clocks are non-decreasing
Test1_Init ≜
  TypeInit

Test1_Next ≜
 ∃ delta ∈ Int :
   AdvanceClocks(delta)

Test1_Inv ≜
  ∧ time′ ≥ time
  ∧ ∀ p ∈ Proc : hc′[p] ≥ hc[p]
```

```
─────────────── MODULE MC_ClockSync2 ──
t_min ≜ 17
t_max ≜ 91

VARIABLES
```

```
INSTANCE ClockSync2

 like TypeOK, but used only in initialization
TypeInit ≜
  ∧ time ∈ Nat
  ∧ hc ∈ [Proc → Nat]
  ∧ adj ∈ [Proc → Int]
  ∧ state ∈ [Proc → State]
  ∧ ∃ t   ∈ [Proc → Int] :
     msgs ∈ SUBSET {[src ↦ p, ts ↦ t[p]] : p ∈ Proc}
```

recall, bounded data structures!

## Testing 2.2


igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync

```
PASS #12: PostTypeCheckerSnowcat                                     I@09:23:24.646
 > Running Snowcat .::.                                              I@09:23:24.647
 > Your types are great!                                             I@09:23:25.260
 > All expressions are typed                                         I@09:23:25.261
PASS #13: BoundedChecker                                             I@09:23:25.343
State 0: Checking 1 state invariants                                 I@09:23:25.917
Step 0: picking a transition out of 1 transition(s)                  I@09:23:25.943
State 1: Checking 1 state invariants                                 I@09:23:25.996
Step 1: picking a transition out of 1 transition(s)                  I@09:23:26.011
State 2: Checking 1 state invariants                                 I@09:23:26.049
Step 2: picking a transition out of 1 transition(s)                  I@09:23:26.064
Step 3: Transition #0 is disabled                                    I@09:23:26.092
Found a deadlock. Check the counterexample in: counterexample0.tla, MC0.out, cou
nterexample0.json E@09:23:26.214
The outcome is: Deadlock                                             I@09:23:26.230
Checker has found an error                                           I@09:23:26.233
It took me 0 days  0 hours  0 min  5 sec                             I@09:23:26.236
Total time: 5.169 sec                                                I@09:23:26.237
EXITCODE: ERROR (12)

 11:23:27  ✘  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync
2.7.1  3.8.6 2.7.17  main ✘ * ★  7s
$
```

test that messages are sent

$$
\begin{aligned}
&Test2\_Inv \triangleq \\
&\quad \forall p \in Proc : \\
&\qquad state[p] = \text{“sent”} \equiv \\
&\qquad\quad \exists m \in msgs : \\
&\qquad\qquad m.src = p
\end{aligned}
$$

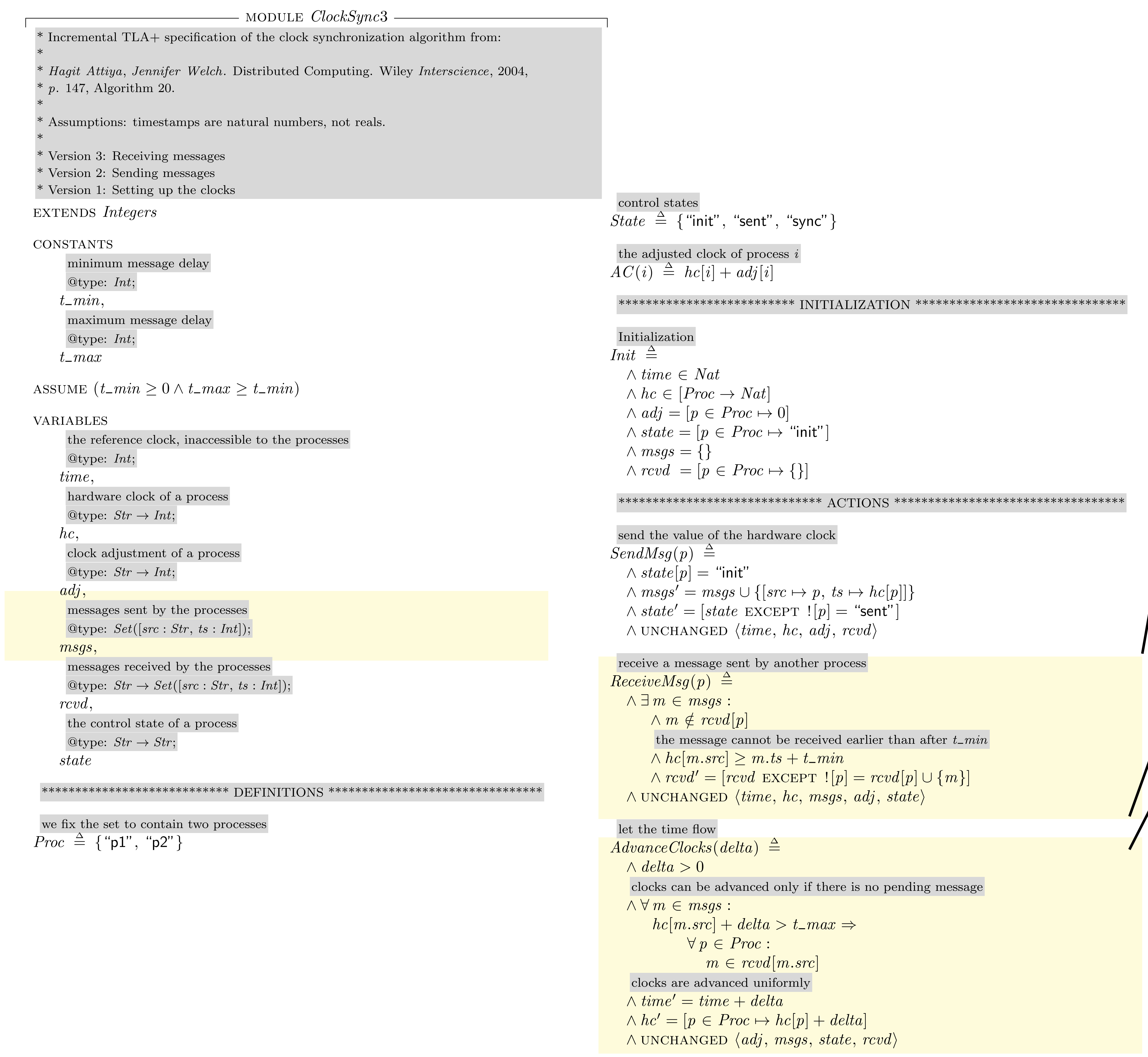
$$
\begin{aligned}
&Test2\_Init \triangleq \\
&\quad \land TypeInit \\
&\quad \land Test2\_Inv
\end{aligned}
$$

$$
\begin{aligned}
&Test2\_Next \triangleq \\
&\quad \exists p \in Proc : \\
&\qquad SendMsg(p)
\end{aligned}
$$

informal SYSTEMS

## version 3: receiving messages

# Receive messages



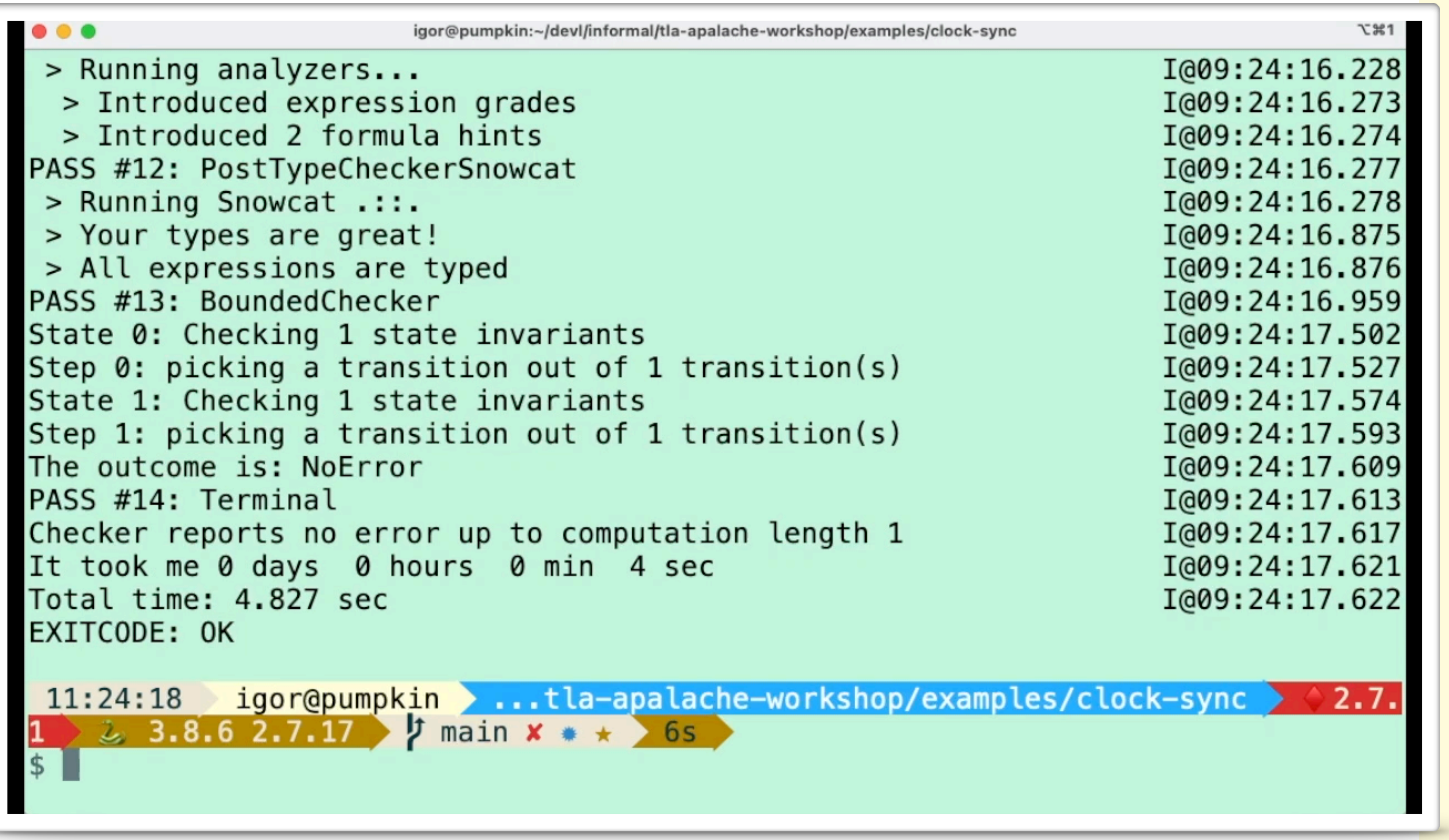
```
MODULE ClockSync3
* Incremental TLA+ specification of the clock synchronization algorithm from:
*
* Hagit Attiya, Jennifer Welch. Distributed Computing. Wiley Interscience, 2004,
* p. 147, Algorithm 20.
*
* Assumptions: timestamps are natural numbers, not reals.
*
* Version 3: Receiving messages
* Version 2: Sending messages
* Version 1: Setting up the clocks
EXTENDS Integers

CONSTANTS
    minimum message delay
    @type: Int;
  t_min,
    maximum message delay
    @type: Int;
  t_max

ASSUME (t_min ≥ 0 ∧ t_max ≥ t_min)

VARIABLES
    the reference clock, inaccessible to the processes
    @type: Int;
  time,
    hardware clock of a process
    @type: Str → Int;
  hc,
    clock adjustment of a process
    @type: Str → Int;
  adj,
    messages sent by the processes
    @type: Set([src : Str, ts : Int]);
  msgs,
    messages received by the processes
    @type: Str → Set([src : Str, ts : Int]);
  rcvd,
    the control state of a process
    @type: Str → Str;
  state

**************************** DEFINITIONS ********************************

we fix the set to contain two processes
Proc ≜ {"p1", "p2"}

control states
State ≜ {"init", "sent", "sync"}

the adjusted clock of process i
AC(i) ≜ hc[i] + adj[i]

************************** INITIALIZATION ********************************

Initialization
Init ≜
  ∧ time ∈ Nat
  ∧ hc ∈ [Proc → Nat]
  ∧ adj = [p ∈ Proc ↦ 0]
  ∧ state = [p ∈ Proc ↦ "init"]
  ∧ msgs = {}
  ∧ rcvd  = [p ∈ Proc ↦ {}]

****************************** ACTIONS ***********************************

send the value of the hardware clock
SendMsg(p) ≜
  ∧ state[p] = "init"
  ∧ msgs′ = msgs ∪ {[src ↦ p, ts ↦ hc[p]]}
  ∧ state′ = [state EXCEPT ![p] = "sent"]
  ∧ UNCHANGED ⟨time, hc, adj, rcvd⟩

receive a message sent by another process
ReceiveMsg(p) ≜
  ∧ ∃ m ∈ msgs :
      ∧ m ∉ rcvd[p]
        the message cannot be received earlier than after t_min
      ∧ hc[m.src] ≥ m.ts + t_min
      ∧ rcvd′ = [rcvd EXCEPT ![p] = rcvd[p] ∪ {m}]
  ∧ UNCHANGED ⟨time, hc, msgs, adj, state⟩

let the time flow
AdvanceClocks(delta) ≜
  ∧ delta > 0
    clocks can be advanced only if there is no pending message
  ∧ ∀ m ∈ msgs :
      hc[m.src] + delta > t_max ⇒
           ∀ p ∈ Proc :
              m ∈ rcvd[m.src]
    clocks are advanced uniformly
  ∧ time′ = time + delta
  ∧ hc′ = [p ∈ Proc ↦ hc[p] + delta]
  ∧ UNCHANGED ⟨adj, msgs, state, rcvd⟩
```

messages received by the processes
@type: $Str \rightarrow Set([src : Str,\ ts : Int])$;
$rcvd$,

```
ReceiveMsg(p) ≜
  ∧ ∃ m ∈ msgs :
      ∧ m ∉ rcvd[p]
        the message cannot be received earlier than after t_min
      ∧ hc[m.src] ≥ m.ts + t_min
      ∧ rcvd′ = [rcvd EXCEPT ![p] = rcvd[p] ∪ {m}]
  ∧ UNCHANGED ⟨time, hc, msgs, adj, state⟩
```

```
AdvanceClocks(delta) ≜
  ∧ delta > 0
    clocks can be advanced only if there is no pending message
  ∧ ∀ m ∈ msgs :
      hc[m.src] + delta > t_max ⇒
           ∀ p ∈ Proc :
              m ∈ rcvd[m.src]
    clocks are advanced uniformly
  ∧ time′ = time + delta
  ∧ hc′ = [p ∈ Proc ↦ hc[p] + delta]
  ∧ UNCHANGED ⟨adj, msgs, state, rcvd⟩
```

# Testing 3.1

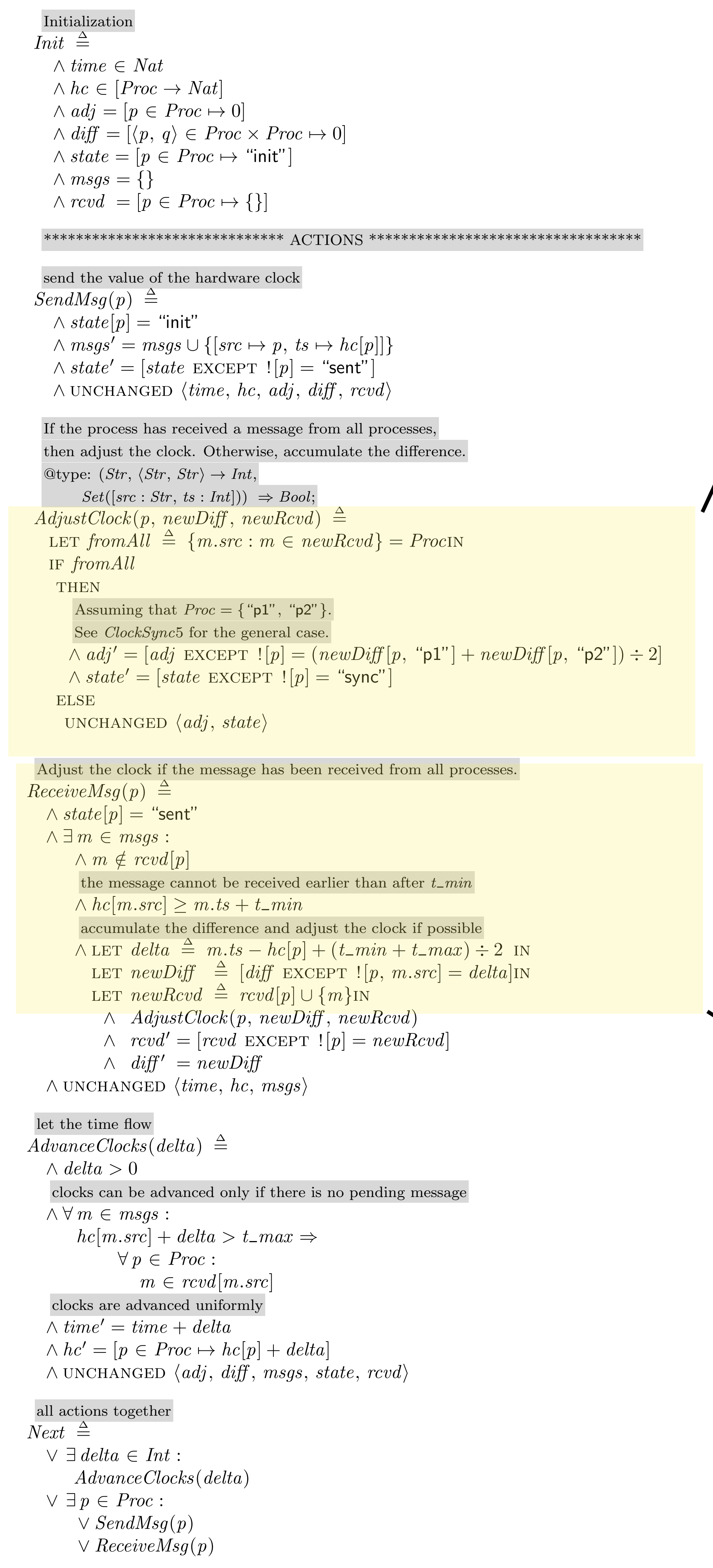


```
igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync                    ⌥⌘1
 > Running analyzers...                                          I@09:24:16.228
  > Introduced expression grades                                 I@09:24:16.273
  > Introduced 2 formula hints                                   I@09:24:16.274
PASS #12: PostTypeCheckerSnowcat                                 I@09:24:16.277
 > Running Snowcat .::.                                          I@09:24:16.278
 > Your types are great!                                         I@09:24:16.875
 > All expressions are typed                                     I@09:24:16.876
PASS #13: BoundedChecker                                         I@09:24:16.959
State 0: Checking 1 state invariants                             I@09:24:17.502
Step 0: picking a transition out of 1 transition(s)              I@09:24:17.527
State 1: Checking 1 state invariants                             I@09:24:17.574
Step 1: picking a transition out of 1 transition(s)              I@09:24:17.593
The outcome is: NoError                                          I@09:24:17.609
PASS #14: Terminal                                               I@09:24:17.613
Checker reports no error up to computation length 1              I@09:24:17.617
It took me 0 days  0 hours  0 min  4 sec                         I@09:24:17.621
Total time: 4.827 sec                                            I@09:24:17.622
EXITCODE: OK

 11:24:18  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync  2.7.
1  3.8.6 2.7.17  main ✘ ✱ ★  6s
$
```

```
test that messages are received within [t_min, t_max]
$Test3\_Inv \triangleq$
  $\land\ \forall m \in msgs :$
        no messages from the future
      $m.ts \leq hc[m.src]$
  $\land\ \forall p \in Proc :$
      $\forall m \in rcvd[p] :$
          the message is received no earlier than after $t\_min$
        $hc[m.src] \geq m.ts + t\_min$
  $\land\ \forall m \in msgs :$
        the message is received no later than before $t\_max$
      $m.ts \geq hc[m.src] + t\_max \Rightarrow$
         $\forall p \in Proc :$
           $m \in rcvd[p]$

$Test3\_Init \triangleq$
  $\land\ TypeInit$
  $\land\ Test3\_Inv$

$Test3\_Next \triangleq$
  $\lor\ \exists delta \in Int :$
      $AdvanceClocks(delta)$
  $\lor\ \exists p \in Proc :$
      $ReceiveMsg(p)$
```

## version 4: adjusting clocks

# Adjust clocks



```
Initialization
Init ≜
  ∧ time ∈ Nat
  ∧ hc ∈ [Proc → Nat]
  ∧ adj = [p ∈ Proc ↦ 0]
  ∧ diff = [⟨p, q⟩ ∈ Proc × Proc ↦ 0]
  ∧ state = [p ∈ Proc ↦ "init"]
  ∧ msgs = {}
  ∧ rcvd  = [p ∈ Proc ↦ {}]

***************************** ACTIONS **********************************

send the value of the hardware clock
SendMsg(p) ≜
  ∧ state[p] = "init"
  ∧ msgs' = msgs ∪ {[src ↦ p, ts ↦ hc[p]]}
  ∧ state' = [state EXCEPT ![p] = "sent"]
  ∧ UNCHANGED ⟨time, hc, adj, diff, rcvd⟩

If the process has received a message from all processes,
then adjust the clock. Otherwise, accumulate the difference.
@type: (Str, ⟨Str, Str⟩ → Int,
       Set([src : Str, ts : Int])) ⇒ Bool;
AdjustClock(p, newDiff, newRcvd) ≜
  LET fromAll ≜ {m.src : m ∈ newRcvd} = ProcIN
  IF fromAll
   THEN
     Assuming that Proc = {"p1", "p2"}.
     See ClockSync5 for the general case.
     ∧ adj' = [adj EXCEPT ![p] = (newDiff[p, "p1"] + newDiff[p, "p2"]) ÷ 2]
     ∧ state' = [state EXCEPT ![p] = "sync"]
   ELSE
    UNCHANGED ⟨adj, state⟩

Adjust the clock if the message has been received from all processes.
ReceiveMsg(p) ≜
  ∧ state[p] = "sent"
  ∧ ∃ m ∈ msgs :
      ∧ m ∉ rcvd[p]
      the message cannot be received earlier than after t_min
      ∧ hc[m.src] ≥ m.ts + t_min
      accumulate the difference and adjust the clock if possible
      ∧ LET delta ≜ m.ts − hc[p] + (t_min + t_max) ÷ 2 IN
        LET newDiff ≜ [diff EXCEPT ![p, m.src] = delta]IN
        LET newRcvd ≜ rcvd[p] ∪ {m}IN
          ∧  AdjustClock(p, newDiff, newRcvd)
          ∧  rcvd' = [rcvd EXCEPT ![p] = newRcvd]
          ∧  diff' = newDiff
  ∧ UNCHANGED ⟨time, hc, msgs⟩

let the time flow
AdvanceClocks(delta) ≜
  ∧ delta > 0
   clocks can be advanced only if there is no pending message
  ∧ ∀ m ∈ msgs :
      hc[m.src] + delta > t_max ⇒
            ∀ p ∈ Proc :
                m ∈ rcvd[m.src]
   clocks are advanced uniformly
  ∧ time' = time + delta
  ∧ hc' = [p ∈ Proc ↦ hc[p] + delta]
  ∧ UNCHANGED ⟨adj, diff, msgs, state, rcvd⟩

all actions together
Next ≜
  ∨ ∃ delta ∈ Int :
      AdvanceClocks(delta)
  ∨ ∃ p ∈ Proc :
      ∨ SendMsg(p)
      ∨ ReceiveMsg(p)
```

```
If the process has received a message from all processes,
then adjust the clock. Otherwise, accumulate the difference.
@type: (Str, ⟨Str, Str⟩ → Int,
       Set([src : Str, ts : Int])) ⇒ Bool;
AdjustClock(p, newDiff, newRcvd) ≜
  LET fromAll ≜ {m.src : m ∈ newRcvd} = ProcIN
  IF fromAll
   THEN
     Assuming that Proc = {"p1", "p2"}.
     See ClockSync5 for the general case.
     ∧ adj' = [adj EXCEPT ![p] = (newDiff[p, "p1"] + newDiff[p, "p2"]) ÷ 2]
     ∧ state' = [state EXCEPT ![p] = "sync"]
   ELSE
    UNCHANGED ⟨adj, state⟩
```

```
Adjust the clock if the message has been received from all processes.
ReceiveMsg(p) ≜
  ∧ state[p] = "sent"
  ∧ ∃ m ∈ msgs :
      ∧ m ∉ rcvd[p]
      the message cannot be received earlier than after t_min
      ∧ hc[m.src] ≥ m.ts + t_min
      accumulate the difference and adjust the clock if possible
      ∧ LET delta ≜ m.ts − hc[p] + (t_min + t_max) ÷ 2 IN
        LET newDiff ≜ [diff EXCEPT ![p, m.src] = delta]IN
        LET newRcvd ≜ rcvd[p] ∪ {m}IN
          ∧  AdjustClock(p, newDiff, newRcvd)
          ∧  rcvd' = [rcvd EXCEPT ![p] = newRcvd]
          ∧  diff' = newDiff
  ∧ UNCHANGED ⟨time, hc, msgs⟩
```

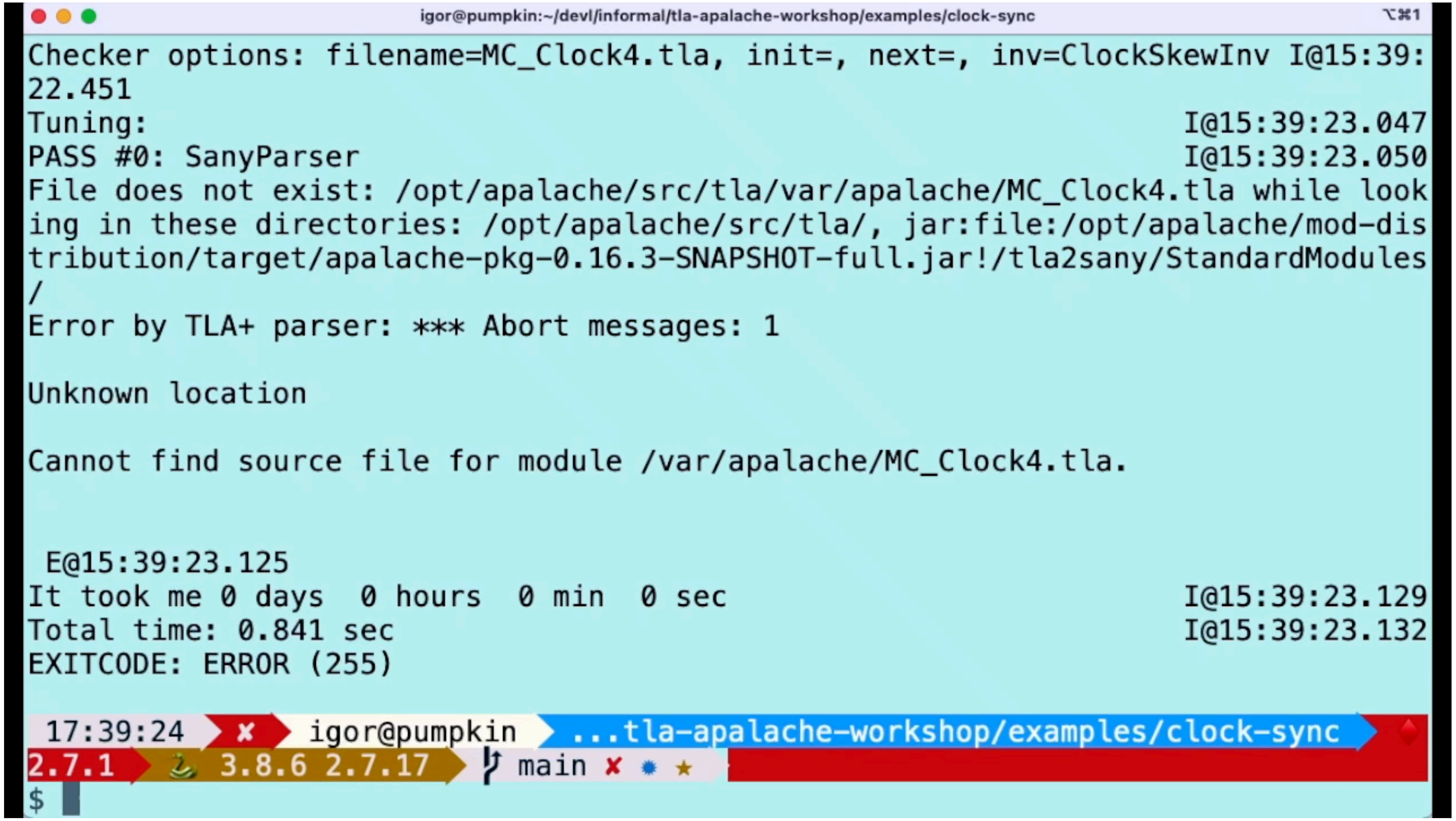
# Specifying bounded clock skew

Theorem 6.15 from $AW04$:

Algorithm achieves $u *$ $(1 - 1/n)$-synchronization for n processors.

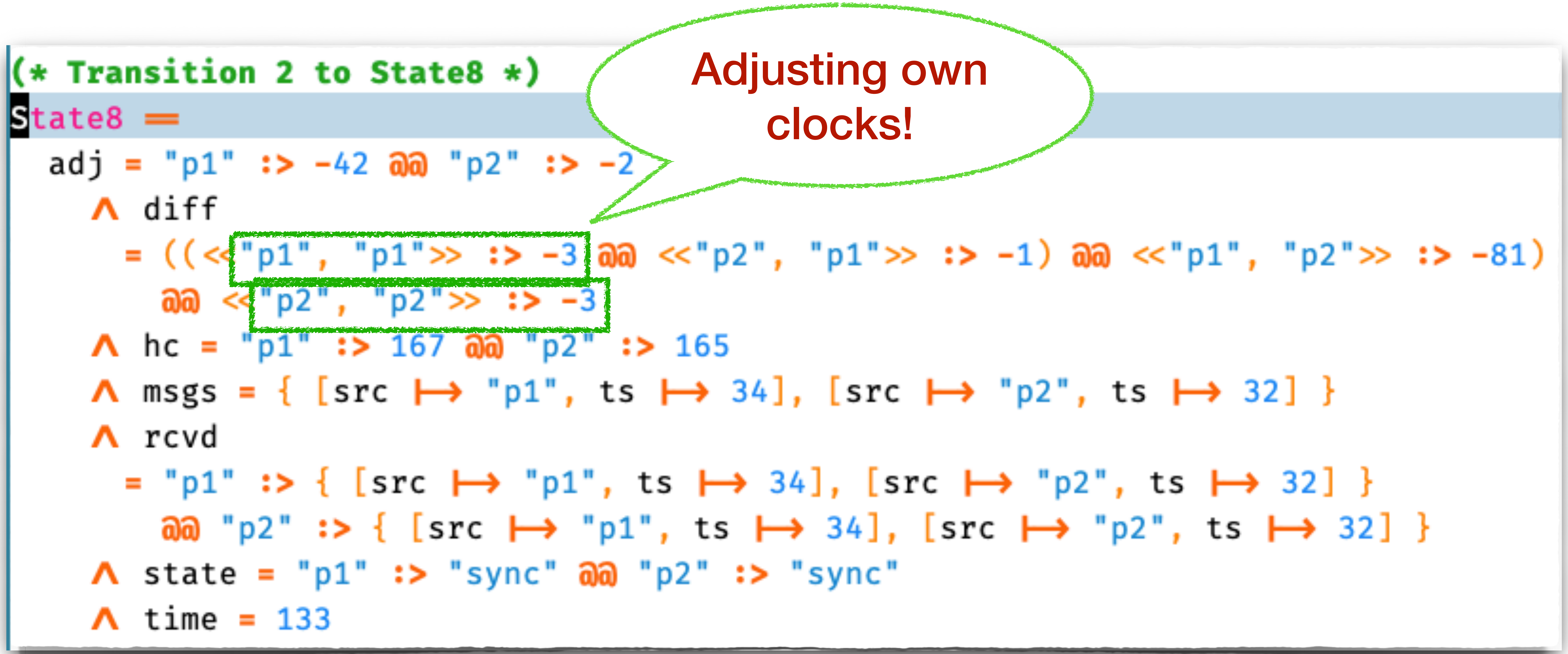
```
$SkewInv \triangleq$
    LET $allSync \triangleq$
        $\forall p \in Proc : state[p] = $ "sync"
    IN
    LET $boundedSkew \triangleq$
        LET $bound \triangleq (t\_max - t\_min) * (NProc - 1)$
        IN
        $\forall p, q \in Proc :$
            LET $df \triangleq AC(p) - AC(q)$
            IN
             $- bound \leq df * NProc \land df * NProc \leq bound$
    IN
    $allSync \Rightarrow boundedSkew$
```

## Check SkewInv 🐞


igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync

```
Checker options: filename=MC_Clock4.tla, init=, next=, inv=ClockSkewInv I@15:39:
22.451
Tuning:                                                          I@15:39:23.047
PASS #0: SanyParser                                              I@15:39:23.050
File does not exist: /opt/apalache/src/tla/var/apalache/MC_Clock4.tla while look
ing in these directories: /opt/apalache/src/tla/, jar:file:/opt/apalache/mod-dis
tribution/target/apalache-pkg-0.16.3-SNAPSHOT-full.jar!/tla2sany/StandardModules
/
Error by TLA+ parser: *** Abort messages: 1

Unknown location

Cannot find source file for module /var/apalache/MC_Clock4.tla.


 E@15:39:23.125
It took me 0 days  0 hours  0 min  0 sec                         I@15:39:23.129
Total time: 0.841 sec                                            I@15:39:23.132
EXITCODE: ERROR (255)

 17:39:24  ✘  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync
2.7.1  3.8.6 2.7.17  main ✘ ● ★
$
```

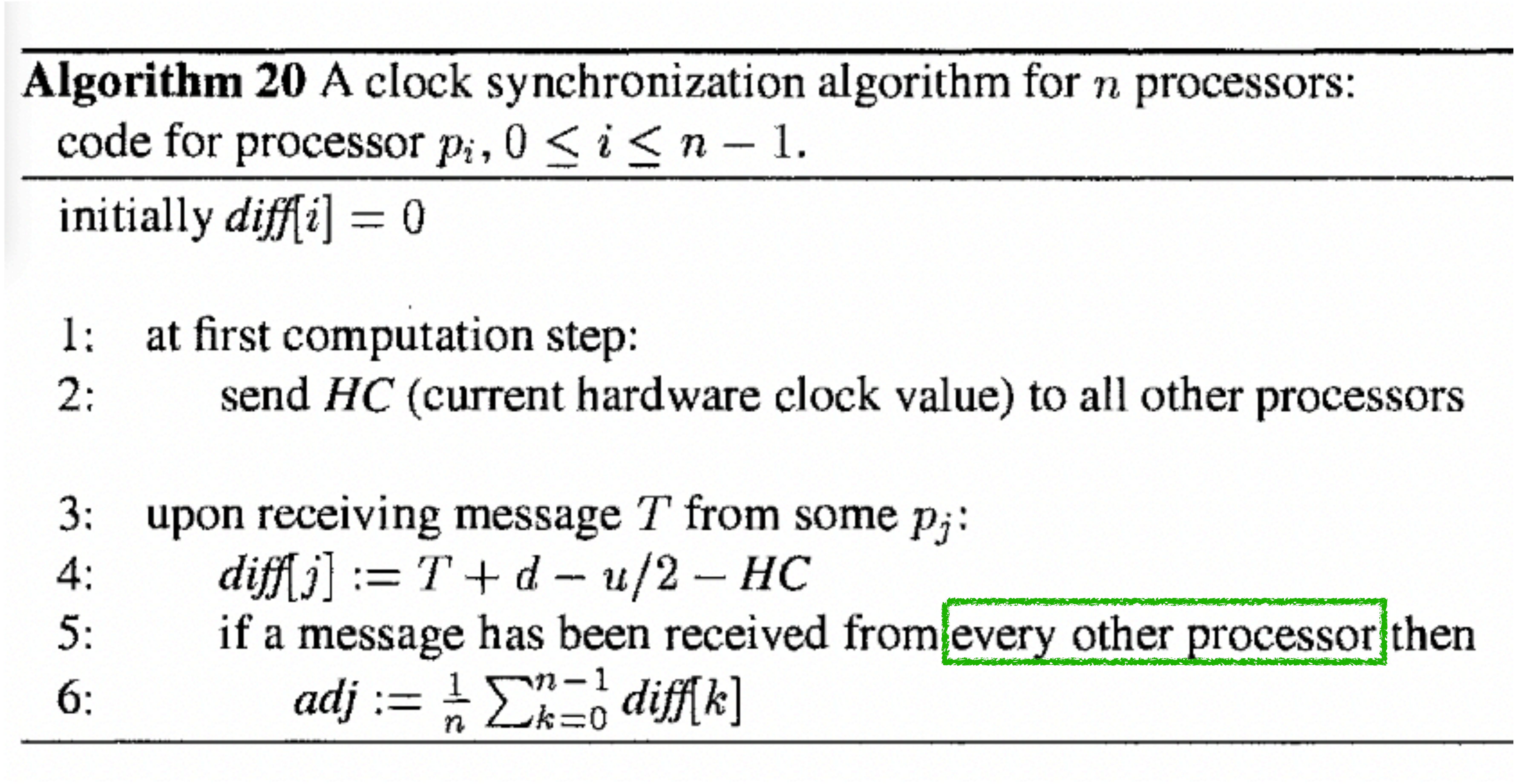
## Analyzing the counterexample



```
(* Transition 2 to State8 *)
State8 ==
  adj = "p1" :> -42 @@ "p2" :> -2
    ∧ diff
      = ((<<"p1", "p1">> :> -3 @@ <<"p2", "p1">> :> -1) @@ <<"p1", "p2">> :> -81)
        @@ <<"p2", "p2">> :> -3
    ∧ hc = "p1" :> 167 @@ "p2" :> 165
    ∧ msgs = { [src ↦ "p1", ts ↦ 34], [src ↦ "p2", ts ↦ 32] }
    ∧ rcvd
      = "p1" :> { [src ↦ "p1", ts ↦ 34], [src ↦ "p2", ts ↦ 32] }
        @@ "p2" :> { [src ↦ "p1", ts ↦ 34], [src ↦ "p2", ts ↦ 32] }
    ∧ state = "p1" :> "sync" @@ "p2" :> "sync"
    ∧ time = 133
```

informal SYSTEMS

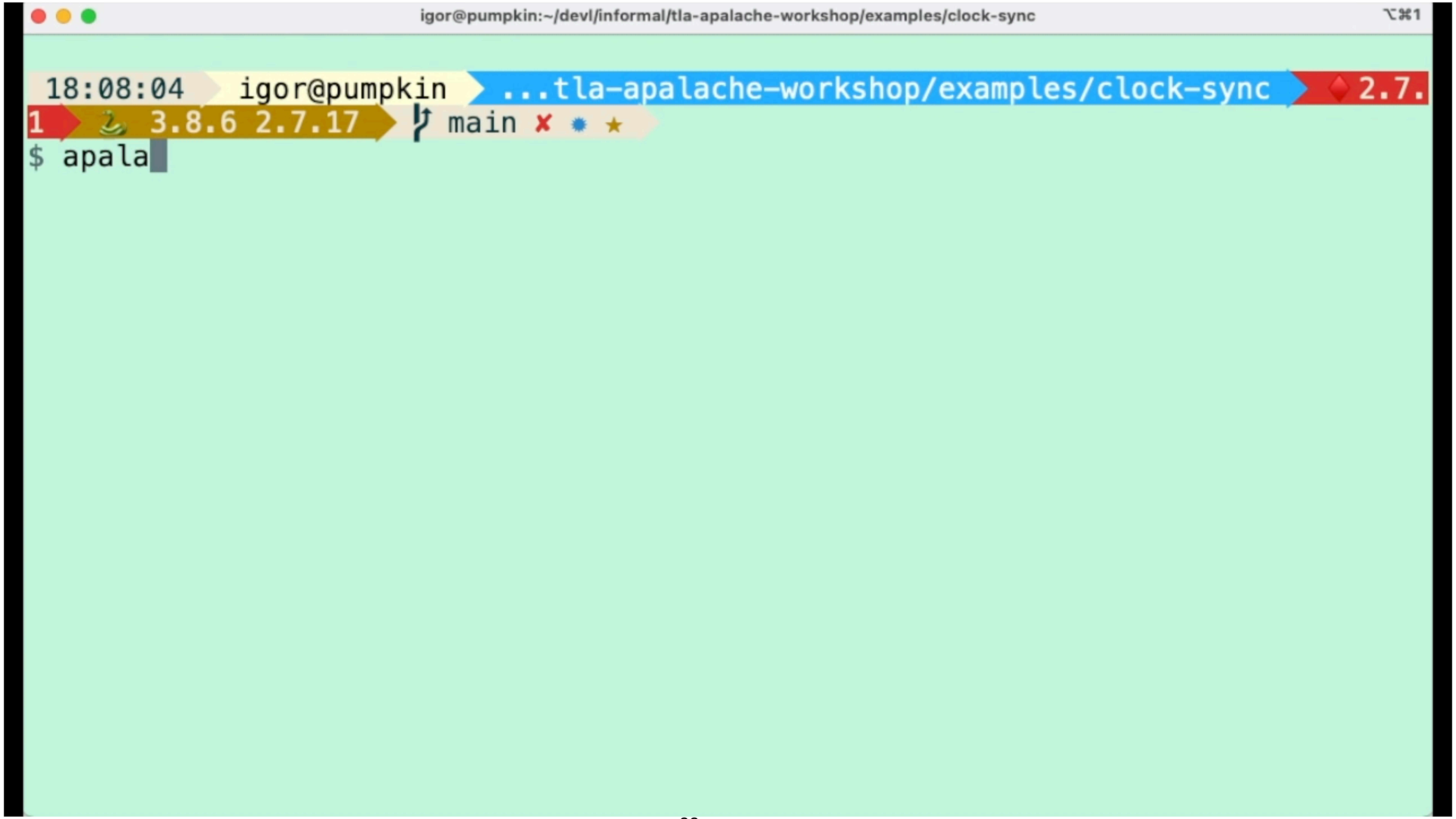
# Check the pseudo-code



**Algorithm 20** A clock synchronization algorithm for $n$ processors: code for processor $p_i$, $0 \le i \le n-1$.

```
initially $diff[i] = 0$

1:  at first computation step:
2:      send $HC$ (current hardware clock value) to all other processors

3:  upon receiving message $T$ from some $p_j$:
4:      $diff[j] := T + d - u/2 - HC$
5:      if a message has been received from every other processor then
6:          $adj := \frac{1}{n} \sum_{k=0}^{n-1} diff[k]$
```

## Fix in ClockSync5 ✅



```
igor@pumpkin:~/devl/informal/tla-apalache-workshop/examples/clock-sync                ⌥⌘1

 18:08:04  igor@pumpkin  ...tla-apalache-workshop/examples/clock-sync  2.7.
1  3.8.6 2.7.17  main ✘ ✹ ★
$ apala
```

## what about $t_{min}$ and $t_{max}$?

# Parameterized time bounds

ASSUME$(t_{min} < t_{max})$

use $-cinit = ConstInit$ to check for all $t\_min$ and $t\_max$

$$ConstInit \triangleq \\ \land\ t\_min \in Nat \\ \land\ t\_max \in Nat$$

```
apalache check -cinit=ConstInit\
 --inv=SkewInv MC_ClockSync6.tla
```

🐞 error due to integer rounding!

Fix by increasing the bounds: ✅

$$\text{LET } bound \triangleq (t\_max - t\_min) * (NProc - 1) + NProc * NProc$$

informal SYSTEMS

## Next steps

## Does it work?

- Parameterize by the set of processes: ClockSync6p ✅
- Check 4 unit tests ✅
- Check for 2 and 3 processes ✅
- Check for arbitrary $t_{min}$ and $t_{max}$ (with `ConstInit`) ✅

🐇 ⇝ 🐢

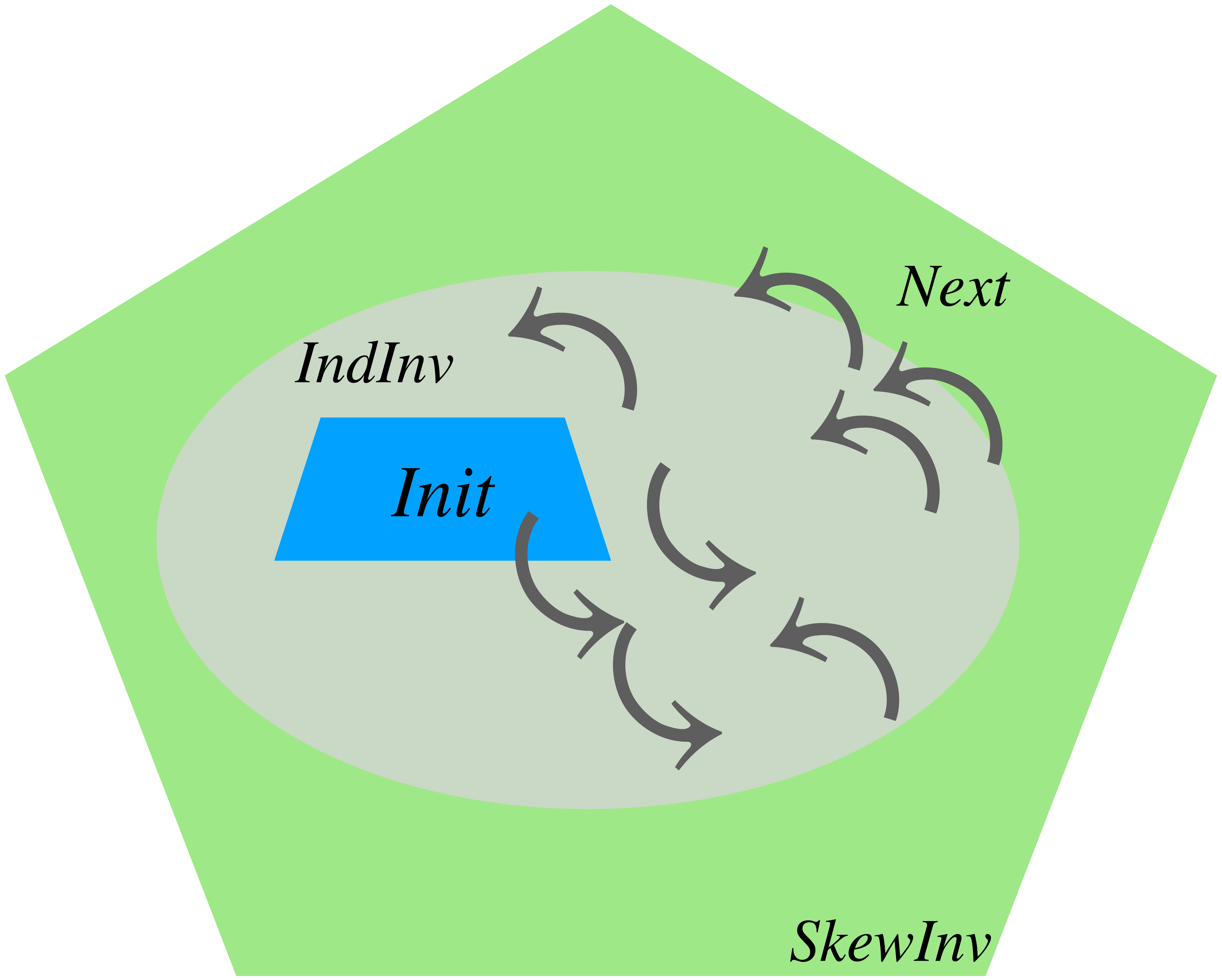
- Check an inductive invariant ❓

informal SYSTEMS

# Inductive invariants

Find a predicate $IndInv$ over states:

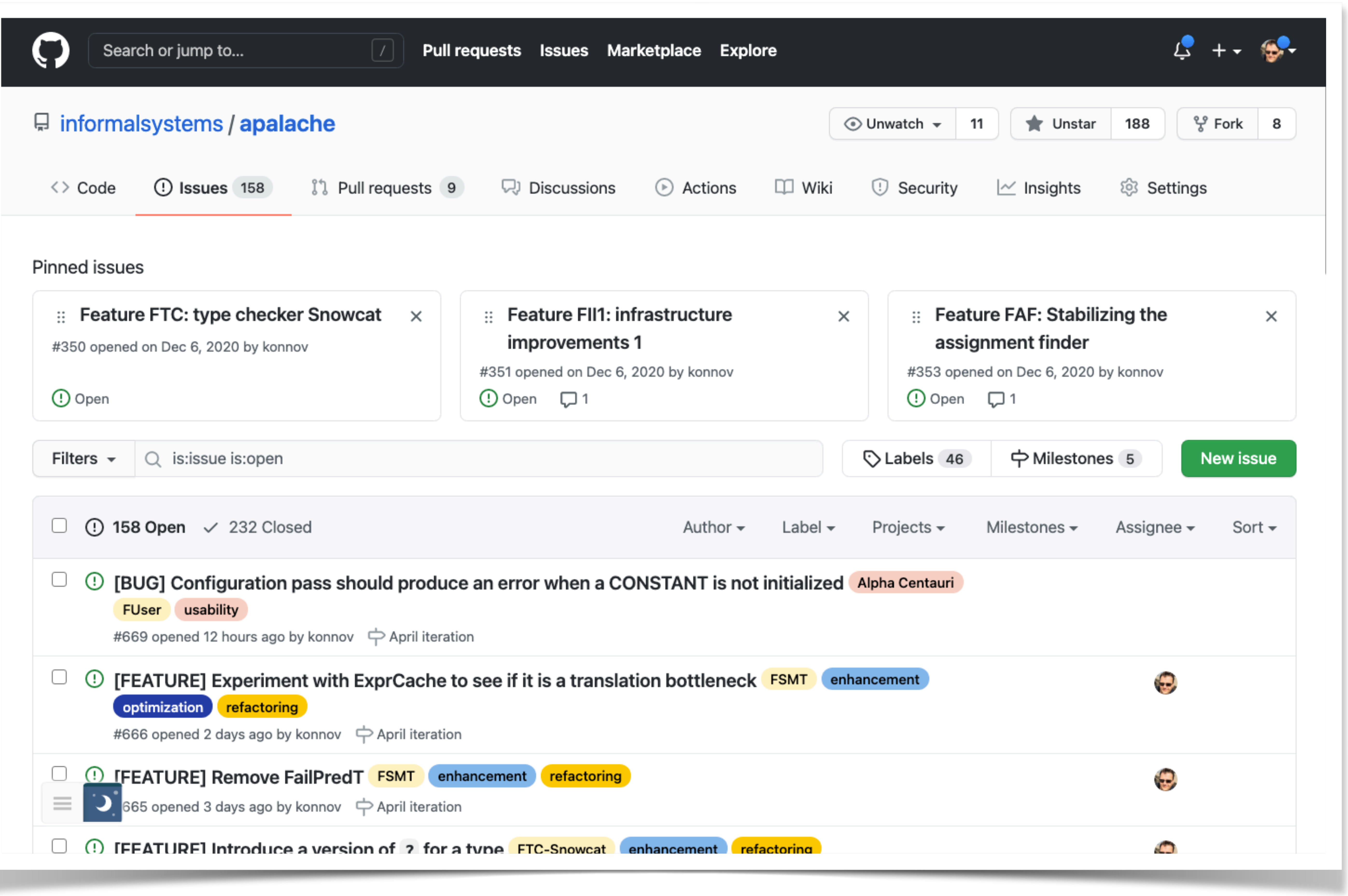
1. $Init \Rightarrow IndInv$
2. $IndInv \land Next \Rightarrow IndInv'$
3. $IndInv \Rightarrow SkewInv$



Shallow queries of length 0 and 1 in Apalache!

[need one more session]

# github.com/informalsystems/apalache ⏎



Search or jump to... Pull requests Issues Marketplace Explore

informalsystems / apalache — Unwatch 11 — Unstar 188 — Fork 8

Code | Issues 158 | Pull requests 9 | Discussions | Actions | Wiki | Security | Insights | Settings

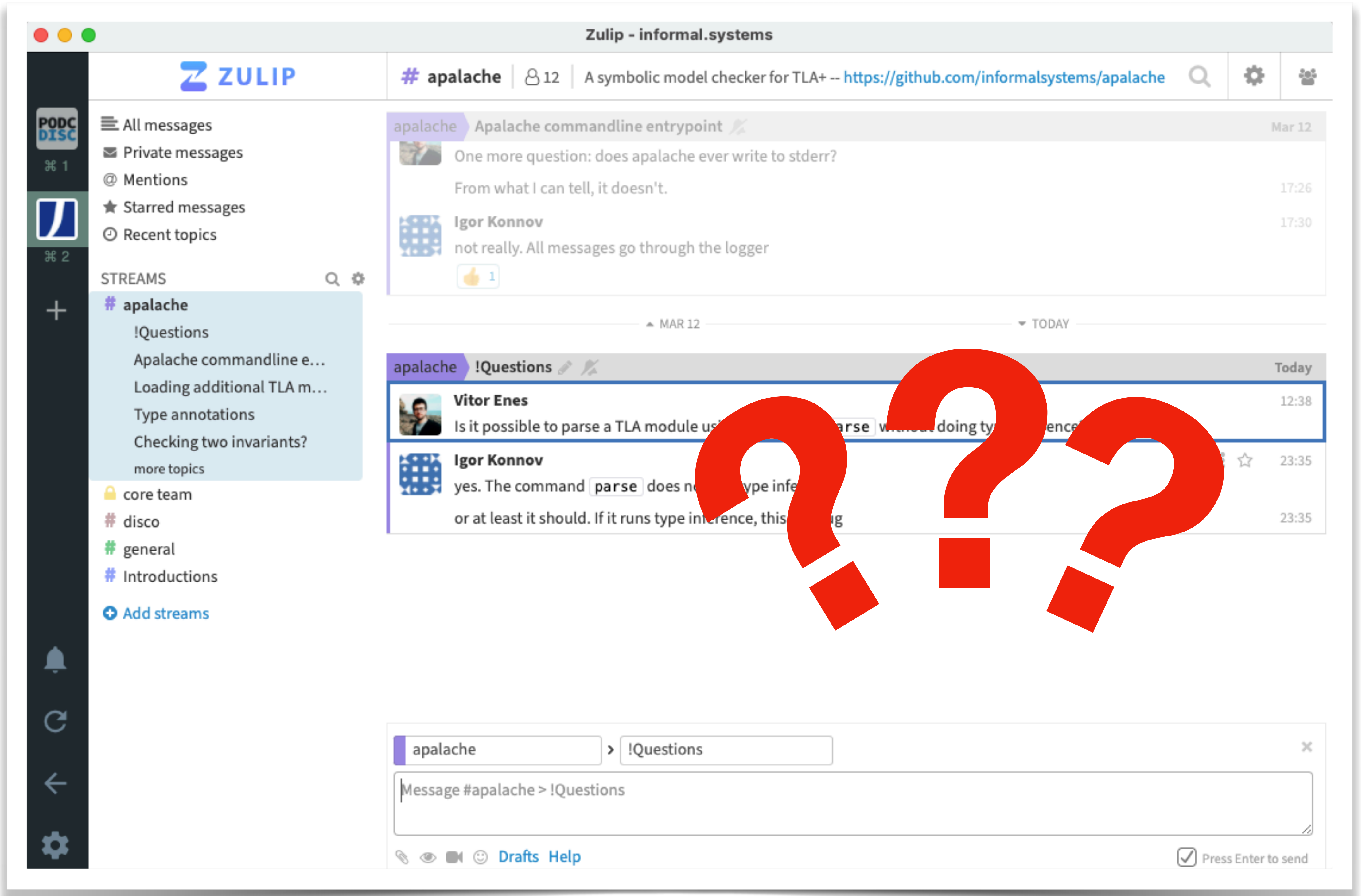
Pinned issues

- Feature FTC: type checker Snowcat — #350 opened on Dec 6, 2020 by konnov — Open
- Feature FII1: infrastructure improvements 1 — #351 opened on Dec 6, 2020 by konnov — Open — 1
- Feature FAF: Stabilizing the assignment finder — #353 opened on Dec 6, 2020 by konnov — Open — 1

Filters — is:issue is:open — Labels 46 — Milestones 5 — New issue

158 Open ✓ 232 Closed — Author — Label — Projects — Milestones — Assignee — Sort

- [BUG] Configuration pass should produce an error when a CONSTANT is not initialized — Alpha Centauri — FUser — usability — #669 opened 12 hours ago by konnov — April iteration
- [FEATURE] Experiment with ExprCache to see if it is a translation bottleneck — FSMT — enhancement — optimization — refactoring — #666 opened 2 days ago by konnov — April iteration
- [FEATURE] Remove FailPredT — FSMT — enhancement — refactoring — #665 opened 3 days ago by konnov — April iteration
- [FEATURE] Introduce a version of ? for a type — FTC-Snowcat — enhancement — refactoring



Zulip - informal.systems

ZULIP — # apalache — 12 — A symbolic model checker for TLA+ -- https://github.com/informalsystems/apalache

All messages
Private messages
Mentions
Starred messages
Recent topics

STREAMS
- apalache
  - !Questions
  - Apalache commandline e...
  - Loading additional TLA m...
  - Type annotations
  - Checking two invariants?
  - more topics
- core team
- disco
- general
- Introductions

Add streams

apalache Apalache commandline entrypoint Mar 12

One more question: does apalache ever write to stderr?

From what I can tell, it doesn't. 17:26

Igor Konnov 17:30
not really. All messages go through the logger 👍 1

MAR 12 — TODAY

apalache !Questions Today

Vitor Enes 12:38
Is it possible to parse a TLA module u[illegible] `arse` [illegible] doing ty[illegible]ence

Igor Konnov 23:35
yes. The command `parse` does n[illegible]ype infe[illegible]

or at least it should. If it runs type in[illegible]rence, this [illegible]g 23:35

apalache > !Questions

Message #apalache > !Questions

Drafts Help — Press Enter to send

chat in zulip

## Our mission is to bring verifiability to distributed systems and organizations.

**Our vision** is an open-source ecosystem of cooperatively owned and governed distributed organizations running on reliable distributed systems.

## Formal Verification Tools

We build formal verification tools that we leverage in our protocol design, engineering, and security audits

### Apalache

Symbolic model checker for TLA+ – formally verify TLA+ specifications for real-world distributed systems protocols

### Model Based Testing

A methodology and tool used to auto-generate tests for real implementations from an underlying TLA+ model.

## Blockchain Infrastructure

We are core developers of the Tendermint and IBC projects, with a focus on software implementations in Rust.

### tendermint-rs

Tendermint is a Byzantine Fault Tolerant state machine replication engine for applications written in Rust.

### ibc-rs

Inter-Blockchain Communication (IBC) is a protocol for secure, packet-based communication between distinct blockchains.

### Hermes

Hermes is an open-source Rust implementation of a relayer for IBC, released under the ibc-relayer-cli crate.