Specification and Analysis of Systems with TLA+ An Brief Introduction

Ramón Béjar Torres

Departament d'Informàtica i Diseny Digital
Universitat de Lleida

TLA+ - Introduction

A system is specified as:

1. A set of initial states
2. A transition function $T(State, State')$ that represents possible transitions.

We also include a set of properties that we indicate that any behavior (valid execution sequence) of the system should satisfy, using invariants and other temporal logic assertions.

TLA+ - Introduction

Two ways to specify the system:

1. The mathematical language of TLA+ (based on mathematical expressions that use basically set theory to express states and the relation between states)
2. A more high-level language: PlusCal (but it does not allow to specify as much systems as the TLA+ language) The TLA+ tools make the automated translation of a PlusCal algorithm to an equivalent TLA+ version.

When low-level detail of the system is needed (e.g. for very concrete hardware systems), TLA+ language may be the only choice

TLA+ - Example 1 with PlusCal

MODULE *Add2FG*

```
********   --algorithm Add2FG{
  variable x = 0;
  process ( A = 1 )
   variable ta = -1; {
    a1: ta := x;
    a2: x := ta + 1;  }
  process ( B = 2 )
   variable tb = -1; {
    b1: tb := x;
    b2: x := tb + 1;  }
********
```

TLA+ - Example 1 with PlusCal

BEGIN TRANSLATION $(chksum(pcal) = \text{“}1b626abd\text{”} \wedge chksum(tla) = \text{“}5ff5e3e0$

VARIABLES x, pc, ta, tb

$$vars \triangleq \langle x, pc, ta, tb \rangle$$

$$ProcSet \triangleq \{1\} \cup \{2\}$$

$Init \triangleq$ Global variables

$$\wedge\ x = 0$$

Process A

$$\wedge\ ta = -1$$

Process B

$$\wedge\ tb = -1$$

$$\wedge\ pc = [self \in ProcSet \mapsto \text{CASE } self = 1 \rightarrow \text{“a1”} \;\Box\; self = 2 \rightarrow \text{“b1”}]$$

TLA+ - Example 1 with PlusCal

$$
\begin{aligned}
a1 \triangleq\ & \land\ pc[1] = \text{“a1”} \\
& \land\ ta' = x \\
& \land\ pc' = [pc \ \text{EXCEPT}\ ![1] = \text{“a2”}] \\
& \land\ \text{UNCHANGED}\ \langle x, tb \rangle \\
a2 \triangleq\ & \land\ pc[1] = \text{“a2”} \\
& \land\ x' = ta + 1 \\
& \land\ pc' = [pc \ \text{EXCEPT}\ ![1] = \text{“Done”}] \\
& \land\ \text{UNCHANGED}\ \langle ta, tb \rangle \\
A \triangleq\ & a1 \lor a2
\end{aligned}
$$

Two only possible steps for process A: a1 and a2

TLA+ - Example 1 with PlusCal

$$
\begin{aligned}
b1 \;\triangleq\; & \land\, pc[2] = \text{“b1”} \\
& \land\, tb' = x \\
& \land\, pc' = [pc \ \text{EXCEPT} \ ![2] = \text{“b2”}] \\
& \land\, \text{UNCHANGED} \ \langle x, ta \rangle \\
b2 \;\triangleq\; & \land\, pc[2] = \text{“b2”} \\
& \land\, x' = tb + 1 \\
& \land\, pc' = [pc \ \text{EXCEPT} \ ![2] = \text{“Done”}] \\
& \land\, \text{UNCHANGED} \ \langle ta, tb \rangle \\
B \;\triangleq\; & b1 \lor b2
\end{aligned}
$$

Two only possible steps for process B: b1 and b2

TLA+ - Example 1 with PlusCal

Allow infinite stuttering to prevent deadlock on termination.

$$Terminating \triangleq \land \forall self \in ProcSet : pc[self] = \text{“Done”}$$
$$\land \text{ UNCHANGED } vars$$

$$Next \triangleq A \lor B$$
$$\lor Terminating$$

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

$$Termination \triangleq \Diamond(\forall self \in ProcSet : pc[self] = \text{“Done”})$$

END TRANSLATION

TLA+ - Example 1 with PlusCal

The TLA+ equivalent translation of our algorithm describes the sets of possibles behaviors (valid executions) with the specification:

$$Init \wedge \Box[Next]_{vars}$$

with two things:

- An initial condition that specifies the possible starting states (predicate Init).
- A next-state relation that specifies the possible steps (predicate Next that specifies the relation between pairs of successive states). The symbol $\Box$ is a **temporal logic operator** that means for every state of the behavior

TLA+ - Example 1 with PlusCal

In the transition relation:

$$Next \triangleq A \lor B \lor Terminating$$

where:

$$Terminating \triangleq \begin{array}{l} \land \forall self \in ProcSet : pc[self] = Done \\ \land \text{UNCHANGED } vars \end{array}$$

we indicate that the concurrent system is in either a A (a1 or a2) or B (b1 or b2) transition or both processes are in the Done state so the variables at the next state have the same value that in the previous state

TLA+ - Example 1 with PlusCal

Then, if we want to check whether our system satifies the following temporal property (invariant for any state of any valid exercution):

$$Spec \Rightarrow \Box((pc[1] = "Done") \wedge (pc[2] = "Done") => (x = 2))$$

we can use the Model Checker TLC provided with the TLA+ tools.

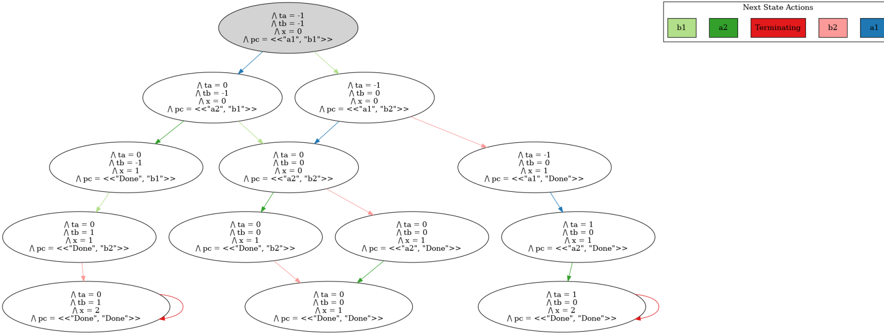
The TLC Model Checker for TLA+ (Simplified)

Initially: $seen = toexpand = [s \mid s \text{ satisfies the Init predicate}]$

while there are new states in toexpand:

1. Remove state s from the front of toexpand
2. For any state t that satisfies $Next(s, t)$ and not on seen queue :
   1. Add t to the queue seen with a pointer to s
   2. If t satisfies all the invariant properties (if any) then add t to the end of the queue toexpand.
   3. Else, report ”found behaviour ending in t that violates some invariant”

TLA+ - Example 1 with PlusCal



Valid sequence of actions giving the resulting sequence of states (behaviour) not satisfying the invariant:

$$[b1, a1, b2, a2]$$

TLA+ - Example 2 directly with TLA+

MODULE *concurrent_buffer*

****** Example from: https://*levelup.gitconnected.com*/debugging-concurrent-systems-with-a-model-checker-c7eee210d86f
******

EXTENDS *Naturals*, *Sequences*

CONSTANTS *Producers*,
Consumers,
BufCapacity,
Data

$$\text{ASSUME } \land Producers \neq \{\} \\ \land Consumers \neq \{\} \\ \land Producers \cap Consumers = \{\} \\ \land BufCapacity > 0 \\ \land Data \neq \{\}$$

TLA+ - Example 2 directly with TLA+

$$\text{VARIABLES } \mathit{buffer}, \mathit{waitSet}$$

$$\mathit{Participants} \triangleq \mathit{Producers} \cup \mathit{Consumers}$$

$$\mathit{RunningThreads} \triangleq \mathit{Participants} \setminus \mathit{waitSet}$$

$$\begin{aligned} \mathit{TypeInv} \triangleq\ & \land\ \mathit{buffer} \in \mathit{Seq}(\mathit{Data}) \\ & \land\ \mathit{Len}(\mathit{buffer}) \in 0 \,..\, \mathit{BufCapacity} \\ & \land\ \mathit{waitSet} \subseteq \mathit{Participants} \end{aligned}$$

$$\begin{aligned} \mathit{Notify} \triangleq\ & \text{IF } \mathit{waitSet} \neq \{\} \\ & \text{THEN } \exists x \in \mathit{waitSet} : \mathit{waitSet}' = \mathit{waitSet} \setminus \{x\} \\ & \text{ELSE UNCHANGED } \mathit{waitSet} \end{aligned}$$

$$\mathit{NotifyAll} \triangleq \mathit{waitSet}' = \{\}$$

$$\mathit{Wait}(t) \triangleq \mathit{waitSet}' = \mathit{waitSet} \cup \{t\}$$

TLA+ - Example 2 directly with TLA+

$$Init \triangleq buffer = \langle\rangle \land waitSet = \{\}$$

$$\begin{array}{l}Put(t, m) \triangleq \text{IF } Len(buffer) < BufCapacity \\ \qquad \text{THEN } \land buffer' = Append(buffer, m) \\ \qquad\qquad\quad \land Notify \\ \qquad \text{ELSE } \land Wait(t) \\ \qquad\qquad\quad \land \text{UNCHANGED } buffer\end{array}$$

$$\begin{array}{l}Get(t) \triangleq \text{IF } Len(buffer) > 0 \\ \qquad \text{THEN } \land buffer' = Tail(buffer) \\ \qquad\qquad\quad \land Notify \\ \qquad \text{ELSE } \land Wait(t) \\ \qquad\qquad\quad \land \text{UNCHANGED } buffer\end{array}$$

$$\begin{array}{l}Next \triangleq \exists t \in RunningThreads : \lor t \in Producers \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \land \exists m \in Data : Put(t, m) \\ \qquad\qquad\qquad\qquad\qquad\quad \lor t \in Consumers \land Get(t)\end{array}$$

TLA+ - Example 2 directly with TLA+

$$Prog \triangleq Init \land \Box[Next]_{\langle buffer, waitSet \rangle}$$

$$NoDeadlock \triangleq \Box(RunningThreads \neq \{\})$$

$$\text{THEOREM } Prog \Rightarrow \Box TypeInv \land NoDeadlock$$

TLA+ - Example 2 directly with TLA+

If we run TLC with:

- Producers = $\{"p1", "p2", "p3"\}$
- Consumers = $\{"c1", "c2"\}$
- BufCapacity = 2
- Data = $\{"m1"\}$

and specifying the predicate NoDeadlock as an invariant we obtain an error trace with **24 states** where the final state violates the invariant:

- buffer = $< "m1", "m1" >$
- waitSet = $\{"p1", "p2", "p3", "c1", "c2"\}$

This happens only due to a very specific ordering of the processes and too many producers !