Delft University of Technology

Quantitative Evaluation of Embedded Systems
IN4390

Tool Survey: TLA+ & TLA+ Toolbox

Authors: Jiaxuan Zhang(5258162)
Yiting Li(5281873)
Group ID: (26)

January 22, 2021

1 Report Structure

This report focuses on TLA+ and TLA+ Toolbox. Section 1 introduces the TLA+ and TLA+ toolbox's basic information and their usage and industrial example in evaluating a system. Section 2 presents some essential conceptions and main ideas about how TLA+ works from a high-level perspective. Section 3 and Section 4 summarized the necessary grammar and Toolbox Operation to model a system and check its properties. In section 5, we point out the strengths and weaknesses of TLA+ and compare the different tools we have met for system modeling and property checking.

2 Introduction and Basic Idea

2.1 TLA+

TLA+ is a formal specification language developed by Leslie Lamport, a very famous computer scientist who has obtained Turing Award. TLA+ is used for designing, modeling, recording and verifying software systems, hardware systems and real-life systems, especially distributed systems and concurrent systems. TLA+ always focuses on top-layer abstraction of systems, which means one who uses TLA+ should always try to abstract software above the code and abstract hardware above the circuit level.

2.2 TLA+ Toolbox

TLA+ Toolbox is an integrated development environment(IDE) for the TLA+ tools. It provides an integrated environment to create specifications, check language syntax, and generate a printed version of our module. Moreover, it also provides a TLC model checker GUI to check the property and add extra constraints to our module.

2.3 Uses

TLA+ can be used to model systems and check the functionality and non-functionality property of a realistic system. The word "realistic" here means a system with a finite bound. Details will be illustrated in subsection 3.3. Lots of computer scientists also use TLA+ to check the correctness of distributed algorithms. Properties of large systems and algorithms are always too complicated and too hard to be proved artificially or mathematically. TLA+ provides an autonomous way of testing these properties.

TLA+ has been widely used in several large, complex, and essential projects in several large companies, such as Amazon Web Services(AWS) of Amazon, Xbox 360 of Microsoft, etc.

3 Key conception and Main idea

This part will briefly introduce some key ideas and pre-request knowledge to help readers learn TLA+ from a high-level view.

3.1 State-Based & State-Machine

TLA+ is state-based. When describing a system, it models an execution of a system as a sequence of states. A TLA+ program tries to describe a system as a state machine. Formally, a state machine contains set of states S, set of initial state L and Next-State relations N on S, in which $\mathcal{L} \subseteq \mathcal{S}, \mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$.

So when writing a TLA+ program model, two-part should be specified:

1. Initial condition: Initial condition specifies the initial state of a system or an algorithm

2. Next-State relation: Next-State relation specifies possible action from the current state and describes the state change after the action is taken

3.2 Ordering Pairs & Binary Relation

Ordering Pairs and Binary Relation are two critical conceptions that are important for understanding TLA+ language systems.

Ordering Pairs: Given sets X and Y, the Cartesian product $x \times y$ is defined as $\{(x, y)|x \in X \wedge y \in Y\}$, and its elements are called ordered pairs and is written as $<< x_i, y_i >>$. Based on Ordering Pairs, the Binary Relation is defined. If a set is not an empty set with all elements are order pairs, or it is an empty set, then it is called a **Binary Relation**. Therefore a **Function** can be regarded as a binary relation in which the first component of different ordered pairs is different. And that is exactly the declaration method of a Function in the TLA+ Language system.

According to the conception of ordering pairs and binary relations, we can describe a state machine as following.

$$\{<< S_0, S_1 >>, ... << S_i, S_{i+1} >>, ...\} \tag{1}$$

In which S_i is the previous state, and S_{i+1} is the next state after an available action from state S_i. So we can regard the declaration of Next-State Relation in TLA+ as a way to describe the state machine as a binary relations.

3.3 Property Check

After obtaining the system model in TLA+ Language System, we need to translate the functionality and non-functionality properties into the TLA+ language system. There are two main types of description of properties in the TLA+ system: Invariant and temporal formula.

1. **Invariant**: An invariant is a property that remains unchanged after operations or transformation. In TLA+, it is always used to describe some property if the property is supposed to be held in every state.

2. **Temporal Formula**: Temporal Formula describes temporal logic. Temporal Logic is predicates and propositions, whose truth value may change in terms of time. In TLA+, it is always used to describe some property supposed to be held in certain states but not all states.

Although the TLA+ system is mathematically based, it cannot always check a property "theoretically" by generating a uniform formula for all states. When using TLA+ to inspect the property, the TLC model checker always tries to search all reachable states in the state space and check whether they satisfy the given property. That means TLA+ may not assert a property is held for a system with infinite queue but will keep searching for the next state available in the state-space. By default, the searching method is Breadth-First Search (BFS). Starting at a point in the state space, BFS always tries to firstly search all neighbors at the same depth rather than search deeper.

4 Grammar

TLA+ language is relevant to mathematics, therefore grammar should be focus before coding.

4.1 Operators

An operator is a thing that does a thing. Some of operators in TLA+ can be intuitive, cause they are already exist in math or other language. It can be a symbol indicating a mathematical operation, like $==$ in $Five == 5 + b$. It can be a logical operator or logical connective in mathematical logic, like $\backslash/$

or /\ in $Next == \/(/\ a\ /\ f)\ \/ c$. Some of operators in TLA+ are not so intuitive, like $\backslash subseteq$ or $\backslash A$. Those operators are created to integrate math to TLA+ better, for example, $\backslash subseteq$ is a subset operator, $\backslash A$ means for all. Those not intuitive operators will be covered by later section.

4.2 Set, Tuple and Logic

Set is a important component of TLA+, it appears a lot in actual model, such as $\{1, 2\}$. Creating a set without declare its member is allowed in TLA+, often use capital *constant*. Using $\{, ..., \}$ can define a set with member declaration, using $x\ \backslash in\ 1...N$ can define a arithmetic sequence. A tuple is a finite ordered list (sequence) of elements and they are 1-indexed. Define it like << 1, 2 >>. Logic means propositional logic, such as for all, exist.
TLA+ also provides operations for them, examples are shown in the table1, example for exist means there is at least one x in the set such that P(x) is true.

Meaning	Operator	Example
in set	\in	1 \in {1, 2}
not in set	\notin	1 \notin {}
is subset	\subseteq	{1, 2} \subseteq {1, 2, 3}
for all	\A	\A x \in S : P(x)
exist	\E	\E x \in S : P(x)
first element	Head	Head(<<1, 2>>) == 1
add an element to end	Append	Append(<<1>>, 2) = <<1, 2>>

Table 1: Set Operation Example

4.3 Expression

An expression is a finite combination of symbols allowed according to applicable TLA+ rule. In other word, an expression is anything that follows an operator. The number of expressions is so large that it is difficult to categorize. Therefore, expressions that appear a lot in pratical coding will be listed in the table2. Constant's, variables' and initial state's definitions are intuitive. Next state relation is also called action, The second row can be regarded as a condition, meaning prepare action can happen if *rm* component of *rmState* is string working. The third row is a state transformation, the prime here is indicating the next state of *rmState*. This raw means in the next state, only *rm* component of *rmstate* will convert to string prepared.

Name	Keyword	Example
constant	CONSTANT	CONSTANT rm
variable	VARIABLES	VARIABLES state
initial state	==, /\, \/	Init == /\state=0
next state relation	EXCEPT, !	transitionA == /\state[2]>=1 /\state'=[state EXCEPT ![3]=state[3]+1]

Table 2: Set Operation Example

4.4 Function

Function in TLA+ slightly differs from function in other programming language, in TLA+ they are closer in nature to hashes or dictionaries. The formal definition is that a function is a binary relation between two sets that associates every element of the first set to exactly one element of the second set.

Using equation2 to define a function in TLA+.

$$\begin{aligned} Function &== [s \in S |-> foo] \\ Function[s \in S] &== foo \end{aligned} \tag{2}$$

4.5 Formula

Formula is introduced to describe properties of a TLA+ model. Those formulas look the same with expressions like equation3, sometimes with a few new operators which are listed in table3.

$$\begin{aligned} rm1, rm2 \in RM : \sim/\backslash rmState[rm1] = "aborted" \\ /\backslash rmState[rm2] = "committed" \end{aligned} \tag{3}$$

Equation3 means string *aborted* and string *committed* cannot appear in the *rmState* simultaneously.

Operator	Logic
[]P	P holds true for every states
<>P	for every possible behavior, at least one state has P as true
P ~>Q	if P ever becomes true, Q must be true eventually
<>[]P	at some point P becomes true, then stays true

Table 3: Set Operation Example

5 Toolbox Usage

2In this section, the basic usage of TLA+ toolbox will be introduced.

5.1 Preparation

Find the assets in this link https://lamport.azurewebsites.net/tla/toolbox.html, download and install it. Following these two steps: first modeling then verifying, while operating TLA+ toolbox. Detailed instructions will be included by step-by-step video guide.

5.2 Modeling

You may code in a new spec in toolbox. Open the *toolbox.exe*, in the top bar, follow these steps: click $file \Rightarrow$ click *Open spec* $\Rightarrow$ click *Add a new spec*. After that, you can reach a window where specification name is needed, browse one file folder then enter your spec name, then you can code. If no grammar is detected, a parsed symbol will appear in the right bottom of screen.

5.3 Verifying

After formula and spec are built, verifying is the next step. In the top bar, follow these steps: click *TLC Model Checker* $\Rightarrow$ click *New model* $\Rightarrow$ enter your model name. Several things should be paid attention to three items: 1.The *behavior* declaration: you may check whether the *init*(*initial state*) and *next*(*next state relation*) are consistent with spec. 2.Constants specification: a clear constants definition should be presented. 3.Property checking: illustrate which property should be checked, such as deadlock free. After configuration is set, click run and toolbox will search states space until arrive at state that violate predefined property(if so, a error and trace to error can be seen).

6 Evaluation

In this section, we will analyze the strengths and weaknesses of TLA+ systems. And make a comparison among TLA+, mCRL22, and Petri-Net.

6.1 Advantages

1. Based on mathematics, TLA+ provides a high-level way to describe a system without paying attention to less important details. During the process, a designer may explore a more concise and efficient way to organize the system.

2. TLA+ provides convenient way to model concurrent systems based on conception partial ordering. And the TLC checker in the TLA+ toolbox can check property automatically, which is really useful for a real-life, extensive, complex system.

3. The basic grammar of TLA+ is based on mathematics, especially discrete mathematics. So it is easy for a user to learn how to read a TLA+ model. This is very useful in the industrial design process.

6.2 Disadvantages

1. TLA+ is not able to check a property "theoretically." When trying to check a property, TLA+ will use Breadth-First search to search all states in the state space. Therefore, if the state space is large, it may take a very long time to check the property. Sometimes if the state space is infinitely large and the property is actually held, the search will never stop.

2. To overcome disadvantage 1, we always need to set an upper bound on the system variables. It is rational because the real-life system always has an upper-bound. However, sometimes when modeling a system, we do not know exactly the real upper-bound. So how to set an appropriate upper-bound is a really important and difficult problem. Some properties are hard to model in TLA+. Based on set theory, TLA+ is very expressive, but some properties may need a very complex structure or very careful conversion and translation before they can be correctly processed by the TLC checker. Therefore, the TLC checker system of TLA+ is hard to master.

6.3 Comparison

6.3.1 Compared to mCRL2

mCRL2 is another formal specification language with an associated tool-set. Unlike TLA+, mCRL2's language systems are less abstract and have a more concise way to model an action sequence. Besides, the property check system of mCRL2 is based on μ-calculus, which packed more useful usage and is more expressive than the basic mathematical logic expression used in TLA+. However, closer to mathematical means TLA+ is more useful and more convenient to directly model and verify (mathematical) objects like distributed algorithms.

6.3.2 Compared to Petri Net

TLA+ and Petri Nets are similar to each other, both of them can be regarded as a state machine, which uses state to describe the system behavior. Main difference between them is that, TLA+ focuses more on describing overall logic of the system from the top-level, which makes TLA+ more abstract. It also means we always ignore some low-level details, for example time delay when using TLA+. Therefore, the verification result of TLA+ may still has a lot distance to the real performance, because TLA+ is too ideal. Petri Nets is less logical and its graphical language is easier for programmers to understand systems first. It also provides some convenient ways to contains lower-level details in the net (e.g. timed Petri Net).