Specifying Systems Notes

by Arnav Kumar

Web: https://arnavcs.github.io

Specifying Systems is a publication written by Leslie Lamport on the TLA$^+$ language. I choose purposely to omit leaving details in these notes about the grammar of the language, since this can be easily found in the author's summary of the text on the TLA website. Additionally, this is not a summary or recreation of the next in any manner. As such, please read the text to gain a better understading of the contents.

1 System Specifications

Definition	Note

System Specification	A system specification is a description of what a system should do or is intended to do. The behavioural properties of a system are also called the functional or logical properties of a system and is our focus. We do not consider performance properties.
State of a System and a Step	A state is an assignment of values to variables. A pair of successive states is called a step. We can mathematically write a pair as shown below. $$\begin{bmatrix} a = 1 \\ b = 0 \end{bmatrix} \rightarrow \begin{bmatrix} a = 1 \\ b = 1 \end{bmatrix}$$
Behaviour	Formally, a behaviour is a sequence of states.
Temporal Formula	A temporal logic formula is a formula that describes a system's behaviour by relating the next state of a system with the current state.
TLA$^+$	TLA$^+$ stands for the Temporal Logic of Actions and supports both assertional resoning and temporal logic. This system is quite good with describing asynchronous systems, but can be used for nearly any purpose: APIs and distributed systems included.
Propositional Logic	The two basic boolean values, TRUE and FALSE can be used in propositional logic with the operators $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\equiv$ (from highest to lowest precendence).
Tautology	A tautology is a proposition that is true for all possible truth values of its identifiers. For example, the following logic-proposition is a tautology: $$F \Rightarrow F \vee G$$
Sets	A set is a collection of elements that is determined by its elements. We denote sets with curly brackets, so the set of the first three natural numbers is $\{1, 2, 3\}$. The empty set will be denoted as $\{\}$, and operations on sets are $\cap$, $\cup$, $\setminus$, and $\subseteq$ (highest to lowest precedence). Membership is denoted with $\in$.

<table>
<tr><td>Predicate Logic</td><td>The two quantifiers, $\forall$ and $\exists$, are followed with a colon and the variable in the quantifier is called "bound" as opposed to a "free" variable. See the example below where x and y are both bound.
$$\forall x \in S : (\forall y \in T : F)$$</td></tr>
<tr><td>Formulas vs. Statements</td><td>Note that by default, something like $2 * x > x$ is a noun; it is true or false depending on the value of x. On the other hand, if we would like to assert if the formula is true, then we should instead write the statement $2 * x > x$ is true.</td></tr>
<tr><td>Action</td><td>An action is true or false of a step, meaning that it it contains primed variables (from the second state) and unprimed variables (from the first state).</td></tr>
<tr><td>Anatomy of a TLA$^+$ Specification</td><td>A TLA$^+$ specification usually consists of an initial predicate and a next-state relation.
<table>
<tr><td>Initial Predicate</td><td>Specifies all the possible initial values of the initial state. This is a predicate that is true if the variables are possible initial values and false otherwise.</td></tr>
<tr><td>Next-State Relation</td><td>This is an action that specifies how the state can change in any step. The relation is true if the step is valid, and false otherwise.</td></tr>
</table>
We can use the temporal-logic unary operator ($\Box F$) to ensure that F is always true. Thus, given that I and N are an initial predicate and a next-state relation respectively, we see the specification of our system is described as $I \land \Box N$.</td></tr>
<tr><td>Theorem</td><td>A theorem is a temporal formula that is satisfied by every behaviour.</td></tr>
<tr><td>Expressions and Operators</td><td>The $\triangleq$ symbol is used to define both expressions and operators, but what is the difference between the two? Firstly, the definition operator assigns a corresponding expression or operator to the symbol on the left hand side. For expressions, this will look like $S_{\text{id}} \triangleq E_{\text{exp}}$, and for operators it looks like $S_{\text{id}}(p_1, p_2, \dots p_n) \triangleq E_{\text{exp}}$. Secondly, using the defined symbol in an expresion is different. An expression is simply replaced, but for an operator, brackets must be used to specify arguments.</td></tr>
<tr><td>Uniqueness of Specifications</td><td>Since there are multiple ways to model the same thing, two specifications of the same thing are not neccessarly unique. The only thing that matters is that if F_1 and F_2 are formulas for the same behaviour, then $F_1 \equiv F_2$ is a theorem.</td></tr>
<tr><td>What to include in a Specification?</td><td>It is important to only include certain aspects of a specification and not include everything. For example, a step might consist of more than one atomic operation in order to keep the specification simple. This specification will still prove correctness for a system using the intended interface. For hardware specifications, the implementer of a system might not know, for example, that the `val` line should stabilize before the `rdy` line is changed, even though both of these actions happen in the same step.

This is perhaps the hardest part of making a specification—the task of choosing the correct abstraction.</td></tr>
</table>

Constants and Variables	In a module, a constant is a parameter of the specification that doesn't change, whereas a variable is something that can vary depending on the state.
State Function and State Predicate	This is an ordinary expression (without any $'$ or $\Box$) that can contain variables and constants. When it is boolean-valued, it is called a state predicate.
Invariant	If I is an invariant of a specification S, then $S \Rightarrow \Box I$ is a theorem.
Type	A variable v has type T in a specification S if and only if $v \in T$ is an invariant of S, or in order words, $S \Rightarrow \Box(v \in T)$ is a theorem. Types for records can be specified with square brackets, like below. `[val : Data, rdy : {0,1}, ack : {0,1}]`
Type Invairant	To specify the types of variables, a definition can be used to check that the all variable are of the correct type.
Enabled vs. Disabled Actions	An action is enabled in a state when it is possible to take a step with the action in question, otherwise it is disabled. The definition of any action usually begins with its enabling condition.
Avoiding Variable Proliferation	To avoid the issue of having too many variables in a specifcation and its interface, replace individual varibles with records or ordered tuples. This allows for syntax that might be easier to read, the following example ensures that the only changed record fields of chan are changing `.rdy` to `1 - chan.rdy` and setting `.val` to `d`. `chan' = [chan EXCEPT !.val = d, !.rdy = 1 - @]`
Symbol Scope	For constants, variables, and definitions, the scope is the part of the module that follows after it. For operator definitions, the scope of the arguments is local, and for predicate logic expression, the same is true. TLA$^+$ has no variable overshadowing. A symbol cannot be defined if one with the same name already exists.