Safety, Liveness, and Fairness

Leslie Lamport

26 May 2019

The terms *safety*, *liveness*, and *fairness* are used informally throughout the TLA$^+$ documentation—including in the TLA$^+$ Video Course. I assume you can understand a simple TLA$^+$ specification, so you probably have some idea what these terms mean. They are defined precisely here and their significance is discussed. Although they are not written formally in TLA$^+$, the definitions are mathematical and so is the discussion. If you're not used to reading mathematics, this may be tough going. It's worth the effort if you want a more complete understanding of TLA$^+$.

Text colored like this in the table of contents or like this elsewhere is a clickable link.

Contents

1 A Partial Semantics of TLA$^+$ **1**

2 Safety and Liveness **2**

3 Fairness **3**

4 The `WF` and `SF` Operators **5**

References **8**

1 A Partial Semantics of TLA$^+$

Values

A *value* is what is called a set in formal mathematics. It includes all objects that can be written as constant expressions of TLA$^+$. For example, $\sqrt{3}$ is a value, as is the set of all pairs $\langle x, y \rangle$ of real numbers such that $x^2 + y^2$ equals 1. These two values can be written like this in TLA$^+$:

```
CHOOSE v \in Real : (v > 0) /\ (v^2 = 3)
{z \in Real \X Real : z[1]^2 + z[2]^2 = 1}
```

States

A *state* is any assignment of values to variables. There are infinitely many variables (assuming no bound on the length of TLA$^+$ identifiers), but any formula can contain only a finite number of them. A *step* is a pair of states, which I will write $u \to v$ instead of u, v. It's important to remember that a state can assign any value to any variable; variables do not have types.

Behaviors

A *behavior* is any infinite sequence of states. I will write the infinite sequence $s_1, s_2, s_3, \ldots$ of states as $s_1 \to s_2 \to s_3 \to \cdots$. A step of a behavior is any pair of consecutive states in the behavior. For example, $s_{42} \to s_{43}$ is a step of the behavior $s_1 \to s_2 \to s_3 \to \cdots$. It's important to remember that a behavior is *any* sequence of states, not one satisfying some specification.

Properties

A *property* is a predicate on behaviors—that is, it assigns a boolean value (TRUE or FALSE) to every behavior. I will let $b \models P$ be the boolean value that property P assigns to the behavior b. If $b \models P$ equals TRUE, we say that b *satisfies* P.

In TLA$^+$, properties are written as temporal formulas. I will conflate a TLA$^+$ formula with the property it describes, letting $b \models F$ for a formula F mean $b \models P$ for the property P described by F. For example, if b is the behavior $s_1 \to s_2 \to s_3 \to \cdots$ and F is the formula `(x=1) /\ [][x'=x+1]_x`, then $b \models F$ equals TRUE iff (if and only if) state s_1 assigns the value 1 to `x` and, for every step $s_i \to s_{i+1}$ of b, the value assigned to `x` in state s_{i+1} is equal to or 1 greater than the value assigned to `x` in state s_i.

Every property that can be written in TLA$^+$ is *stuttering insensitive*. This means that for any temporal TLA$^+$ formula F and any behavior b, if the behavior $\widehat{b}$ is obtained from b by adding and/or deleting stuttering steps, then $b \models F$ equals $\widehat{b} \models F$. For example, if b is the behavior

$$s_1 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow s_3 \rightarrow s_3 \rightarrow s_4 \rightarrow s_4 \rightarrow s_5 \rightarrow s_5 \rightarrow s_6 \rightarrow s_6 \rightarrow \cdots$$

and $\widehat{b}$ is the behavior

$$s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow s_3 \rightarrow s_3 \rightarrow s_3 \rightarrow s_4 \rightarrow s_4 \rightarrow s_4 \rightarrow s_4 \rightarrow s_5 \rightarrow \cdots$$

then $b \models F$ equals $\widehat{b} \models F$. The reason why TLA$^+$ only allows formulas that describe stuttering-insensitive properties is explained on the TLA$^+$ Advanced Topics web page. Because we are interested only in such properties, henceforth *property* will mean *stuttering-insensitive property*.

2 Safety and Liveness

Define a *prefix* of a behavior b to be a sequence consisting of the first n states of b, for some $n > 0$. For example, s_1 and $s_1 \rightarrow \cdots \rightarrow s_{42}$ are prefixes of the behavior $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots$.

For any finite sequence s of states, let $s^\sharp$ be the behavior obtained by appending infinitely many copies of the last state. For example, if s is the sequence $s_1 \rightarrow \cdots \rightarrow s_{42}$ of states, then $s^\sharp$ is the behavior

$$s_1 \rightarrow \cdots \rightarrow s_{42} \rightarrow s_{42} \rightarrow s_{42} \rightarrow s_{42} \rightarrow \cdots$$

The behavior $s^\sharp$ represents an execution of a system that halts after reaching the last state of s. (Obviously, s is a prefix of $s^\sharp$.)

Safety

For any finite sequence s of states and any property P, we define $s \models P$ to equal $s^\sharp \models P$. A property P is defined to be a *safety* property iff it satisfies the following condition, for all behaviors b:

> $b \models P$ equals TRUE iff $s \models P$ equals TRUE for all prefixes s of b.

Here are some equivalent ways of expressing this condition:

> b satisfies P iff every prefix of b satisfies P.

$b \models P$ equals FALSE iff $s \models P$ equals FALSE for some prefix s of b.

b doesn't satisfy P iff there is a prefix of b that doesn't satisfy P.

If a behavior b doesn't satisfy a safety property P, then there is some shortest prefix s^{min} of b that doesn't satisfy P. If s^{min} contains two or more states, we can think of the last step of s^{min} as the step of b that violates P. If s^{min} consists of a single state, we can think of the initial state of b as violating P. One way to think of a safety property is as a property that, if it's violated, we can point to a place in the behavior where it's violated. Any TLA$^+$ formula of the form `Init /\ [][Next]_vars` is a safety property.

The formula `<>(x = 3)` is not a safety property. That formula is satisfied by a behavior b iff `x` has the value 3 in some state of b. We can't tell that b does not satisfy the formula by looking at any prefix of b. We need to view the entire infinite behavior to know that `x` does not equal 3 in any state.

Liveness

A behavior b is said to be an *extension* of a finite sequence s iff s is a prefix of b. A property P is defined to be a *liveness* property iff every finite sequence of states can be extended to a behavior that satisfies P. The formula `<>(x = 3)` is a liveness property; any finite sequence of states can be extended to a behavior satisfying it by simply appending a state in which `x` equals 3 and then appending any infinite sequence of states.

Safety and Liveness

Any property can be written as the conjunction of a safety property and a liveness property. This result, and the precise definition of liveness, are due to Alpern and Schneider [2]. Because they didn't assume all properties to be stuttering insensitive, they gave a different definition of safety than the one above. For a stuttering-insensitive property, the two definitions are equivalent.

3 Fairness

If S is a safety property and L a liveness property, then the pair S,L is said to be *machine closed* iff the following condition is satisfied [1]:

Every finite sequence of states that satisfies S can be extended to a behavior that satisfies $S \wedge L$.

Since L is a liveness property, every finite sequence s of states can be extended to a behavior satisfying L. Machine closure of S,L means that if s also satisfies S, then the extension can be chosen to satisfy both S and L.

In the context of specification, the term *fairness* has been used in quite a few different ways. It generally refers to a condition on how the specification is written. The most common use of the term is in the idea of fair scheduling of process execution; but in TLA$^+$, the concept of a process is an artifact of how the specification is written. Two equivalent TLA$^+$ formulas can look like specifications of systems with different numbers of processes [4]. The only definition of fairness I can think of that encompasses all the uses of the term I have seen is the following: L is a fairness property for S iff S,L is machine closed. We can then call L a fairness property if the safety property S is understood from the context.

Suppose S equals `Init /\ [][Next]_vars`, as it does for essentially all TLA$^+$ system specifications. In this case, fairness of L asserts that $S \wedge L$ does not disallow any initial state satisfying `Init` or any step satisfying `Next`. For example, let S and L be these formulas:

S = `(x=1) /\ [][\E i \in 2..7 : x' = i*x]_x`

L = `[]<>(x % 2 = 1)`

Formula S is satisfied by behaviors in which `x` equals 1 in the first state and each step that changes `x` multiplies it by some integer in `2..7`. Formula L is satisfied by behaviors in which the value of `x` is odd in infinitely many of its states. Formula L is a liveness property, since any finite sequence of states can be extended to a behavior satisfying it by appending infinitely many states with `x` equal to an odd number. However, it's not a fairness property for S because a finite sequence s of states satisfying S cannot be extended to a behavior satisfying $S \wedge L$ if s contains a step that multiplies `x` by an even number. Formula L disallows steps allowed by the next-state action of S — namely, steps that multiply `x` by 2, 4, or 6.

Specifications that aren't machine closed are hard to understand because you can't tell if a step is allowed just by looking at the `Next` action. The system specifications you write should be machine closed. However, it's sometimes necessary to use a formula that isn't machine closed when verifying that one specification implements another [3].

4 The WF and SF Operators

The standard way to write fairness conditions in TLA$^+$ is with the `WF` and `SF` operators. Before defining these operators, we need to define some notation. An action formula `A` is a predicate on steps. We say that a step is an `A` step iff `A` equals TRUE on that step. The formula `ENABLED A` is defined to equal TRUE on a state u iff there exists a state w such that $u \to w$ is an `A` step. For an action `A`, the action `<<A>>_v` is defined to equal `A /\ (v'/= v)`. Finally, we define a *suffix* of a behavior $s_1 \to s_2 \to s_3 \to \cdots$ to be a behavior $s_i \to s_{i+1} \to s_{i+2} \to \cdots$, where i is a positive integer.

We now define `WF_v(A)` and `SF_v(A)`. The state formula `v` is needed to ensure that these formulas are stuttering insensitive. We define a behavior b to satisfy `WF_v(A)` iff any of the following equivalent conditions are satisfied. These conditions can be hard to understand if you're not used to thinking about infinite sets and sequences, so it may not be obvious that they're all equivalent. You will understand the meaning of `WF_v(A)` when you find their equivalence to be clear.

- Any suffix of b whose states all satisfy `ENABLED <<A>>_v` contains an `<<A>>_v` step.

- b does not contain a suffix with no `<<A>>_v` step whose states all satisfy `ENABLED <<A>>_v`.

- Any suffix of b must contain an `<<A>>_v` step or a state on which `ENABLED <<A>>_v` equals FALSE. (As in all such mathematical statements written in English, "or" means "and/or".)

- If b contains a suffix all of whose states satisfy `ENABLED <<A>>_v`, then b contains infinitely many `<<A>>_v` steps.

- b must contain infinitely many `<<A>>_v` steps or infinitely many states that do not satisfy `ENABLED <<A>>_v`.

We define `SF` by defining a behavior b to satisfy `SF_v(A)` iff the following equivalent conditions are satisfied.

- Any suffix of b containing infinitely many states that satisfy `ENABLED <<A>>_v` contains an `<<A>>_v` step.

- b does not have a suffix that has infinitely many states satisfying `ENABLED <<A>>_v` and has no `<<A>>_v` step.

- Any suffix of b must contain an `<<A>>_v` step or have a suffix in which `ENABLED <<A>>_v` equals FALSE in all its states.

- If b contains infinitely many states satisfying `ENABLED <<A>>_v`, then it contains infinitely many `<<A>>_v` steps.

Any behavior satisfying `SF_v(A)` also satisfies `WF_v(A)`. (Make sure you understand why.) If `<<A>>_v` is not equivalent to `FALSE`, then `WF_v(A)` and `SF_v(A)` are liveness properties. This is because `<<A>>_v` not equivalent to `FALSE` implies that there is some step $u \rightarrow w$ that satisfies `<<A>>_v`. Any finite sequence s of states can be extended to one satisfying `SF_v(A)`, and hence `WF_v(A)`, by appending this infinite sequence of steps:

$$u \rightarrow w \rightarrow u \rightarrow w \rightarrow u \rightarrow w \rightarrow u \rightarrow \cdots$$

However, `WF_v(A)` and `SF_v(A)` need not be fairness properties. For example, let S equal `(x=0) /\ [][x'=x+1]_x`, let `A` equal `x'=x+2` and let `v` equal `x`. An `<<x'=x+2>>_x` step is possible in any reachable state of S, but no such step satisfies `x'=x+1`. (A reachable state of S is one that occurs in some behavior that satisfies S.) Hence, no finite sequence of states satisfying S can be extended to satisfy S as well as `WF_v(A)` or `SF_v(A)`.

We now restrict ourselves to the standard situation, in which S equals `Init /\ [][Next]_vars` for some state formula `Init`, action `Next`, and state expression `vars`. For this safety property, the `v` in `WF_v(A)` and `SF_v(A)` is usually equal to `vars`, but it needn't be. Formulas `WF_v(A)` and `SF_v(A)` are fairness properties for S if `<<A>>_v` is a subaction of `Next`, where an action `B` is defined to be a subaction of `Next` iff the following condition is satisfied:

> If u is a reachable state of S and $u \rightarrow v$ is a step satisfying `B`, then $u \rightarrow v$ satisfies `Next`.

Note that `B` is a subaction of `B \/ C` for any action `C`.

Understanding why `WF_v(A)` and `SF_v(A)` are fairness properties for S if `<<A>>_v` is a subaction of `Next` will help you understand `WF` and `SF`. Here is a proof of their fairness.

For any finite sequence s of states satisfying S, we must show how to extend s to a behavior satisfying `WF_v(A)` and to a behavior satisfying `SF_v(A)`. Since any behavior satisfying `SF_v(A)` satisfies `WF_v(A)`, it suffices to extend s to a behavior satisfying `SF_v(A)`. We do this with the following algorithm for repeatedly appending states to a sequence. It begins with the sequence equal to s, and we let u be the last state of the sequence obtained so far.

if `ENABLED <<A>>_v` equals TRUE in state u

  then append a state w such that $u \to w$ is an `<<A>>_v` step
  (the truth of `ENABLED <<A>>_v` implies the existence of w)

  else **if** `ENABLED Next` equals TRUE in state u

    then append a state w such that $u \to w$ is a `Next` step
    (the truth of `ENABLED Next` implies the existence of w)

    else append u

If the final **else** clause is ever executed, then it will be executed again the next time, so the sequence will be extended forever with stuttering steps. The assumption that `<<A>>_v` is a subaction of `Next` implies that the step $u \to w$ added in either **then** clause is a `Next` step, so the behavior b constructed by the algorithm satisfies S. If `ENABLED <<A>>_v` is true for infinitely many of the added states, then the algorithm adds infinitely many `<<A>>_v` steps, so b satisfies `SF_v(A)`.

In general, the conjunction of any finite number of formulas, each of the form `WF_v(A)` or `SF_v(A)` for subactions `A` of `Next`, is a fairness property for S. Proving this requires generalizing the algorithm for extending the sequence s used in the proof above to work for a finite set of subactions `A`. Finding the appropriate generalization is left as a nice little programming problem.

This result holds also for the "conjunction" of a countably infinite number of `WF` and `SF` formulas. More precisely, suppose that L equals `\A i \in Nat : F(i)`, where for all `i` in `Nat`, each `F(i)` equals `WF_v(A(i))` or `SF_v(A(i))` and `<<A(i)>>_v` is a subaction of `Next`. Then L is a fairness condition for S. The basic idea of the proof is to modify the algorithm for extending the state sequence s to a behavior that satisfies a finite conjunction to obtain an algorithm that produces a behavior satisfying L. The new algorithm works essentially as follows. It finds the first new state by using the algorithm for satisfying `SF_v(A(0))`; it finds the second new state using the algorithm for satisfying `SF_v(A(0)) /\ SF_v(A(1))`; it finds the third new state using the algorithm for satisfying `SF_v(A(0)) /\ SF_v(A(1)) /\ SF_v(A(2))`; and so on.

References

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[2] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[3] Leslie Lamport. Auxiliary variables in TLA+. Web page.

[4] Leslie Lamport. Processes are in the eye of the beholder. *Theoretical Computer Science*, 179:333–351, 1997.