Liveness and Fairness in TLA+

William Schultz

January 17, 2020

Temporal Formulas

Let σ be a behavior and F be a temporal formula F. We say that σ satisfies F, written $\sigma \vDash F$, if F is true on every suffix of σ. Note that when F is a state predicate, it is true if and only if it is true in the first state of a behavior. So $\Box F$ means that F is true in every state of the behavior, since it means that F is true in the first state of every suffix, which covers all states in the behavior. This point is key for deconstructing more involved temporal formulas like $F \rightsquigarrow G$ ("F leads to G"). The definition of $F \rightsquigarrow G$ is defined in terms of the box operator ("always") and the diamond operator ("eventually"):

$$F \rightsquigarrow G \equiv \Box(F \Rightarrow \Diamond G)$$

and the diamond operator is itself defined in terms of the more box operator:

$$\Diamond F \equiv \neg(\Box \neg F)$$

This is just saying that "eventually F is true" is equivalent to saying "it is not true that F is always false". So we can understand the "leads to" operator in its most primitive form:

$$\begin{aligned} F \rightsquigarrow G &\equiv \Box(F \Rightarrow \Diamond G) \\ &\equiv \Box(F \Rightarrow (\neg(\Box \neg G))) \end{aligned}$$

which is a bit hard to parse all at once but we can break it down. The outer box operator means that the inner expression

$$(F \Rightarrow (\neg(\Box \neg G)))$$

holds true on all suffixes of the behavior. So, for all suffixes of the behavior, if F is true on that suffix, then it must be true that G is not always false on that suffix. In other words, G must eventually be true on that suffix. So, the overall expression i.e. "leads to" just says that if F holds true at any point in the behavior, then eventually G must hold true after that point.

Stuttering and Weak Fairness

The naive definition of a TLA+ spec by default allows for *stuttering* steps i.e. steps that don't change any variables. Of course, a spec that takes an infinite amount of stuttering steps (never changes any variables), will always trivially satisfy any safety properties. It just never makes any meaningful progress. Consider the simple toy spec:

$$\begin{aligned} Init \triangleq & \wedge x = 0 \\ & \wedge y = 0 \end{aligned}$$

$$\begin{aligned} Next \triangleq & \wedge x' = (x+1)\%3 \wedge y' = y \\ & \wedge y' = (y+1)\%3 \wedge x' = x \end{aligned}$$

$$Spec \triangleq Init \wedge \Box[Next]_{\langle x,y \rangle}$$

As written, *Spec* allows for stuttering steps. So, a valid behavior (remember that all behaviors are infinite) would be

$$\begin{bmatrix} x = 0 \\ y = 0 \end{bmatrix} \rightarrow \begin{bmatrix} x = 1 \\ y = 0 \end{bmatrix} \rightarrow \begin{bmatrix} x = 1 \\ y = 0 \end{bmatrix} \rightarrow ...$$

stuttering indefinitely. This is probably not what we intuitively want the spec to describe. We want to allow for stuttering steps, but also want to make sure that *eventually* the spec makes progress. Whatever "eventually" means precisely is not that important. For example, if the spec allows 500 stuttering steps before eventually taking a non stuttering step or 5 million stuttering steps before a non-suttering step it doesn't make much difference, since a sequence of stuttering steps, however long, can always be appropriately "de-stuttered". What we really care about is that there is never an *infinite* sequence of stuttering steps, as long as there is an enabled, non-stuttering action. This is where the concept of *weak fairness* comes into play.

For a behavior σ, we say that $\mathrm{WF}_v(A)$ asserts that σ does not contain suffix in which an $\langle A_v \rangle$ step is always enabled but never occurs, where $\langle A_v \rangle$ is defined as $A \wedge (vars' \neq vars)$, if $vars$ is the tuple of all variables in a system specification. That is, $\langle A_v \rangle$ is a *non-stuttering* A step. So, if we changed the above spec to instead be

$$Spec \triangleq Init \wedge \Box[Next]_{\langle x,y \rangle} \wedge WF_{\langle x,y \rangle}(Next)$$

then we don't allow for any behavior with an infinite stuttering suffix. When checking liveness properties, this is important. For example, given the spec above, consider the

following liveness property:

$$EventuallyTwo \triangleq \diamond(x = 2 \vee y = 2)$$

which simply states that eventually $x = 2$ or $y = 2$. If we did not specify weak fairness of our spec, this liveness property would not hold, since we could always allow a behavior to stutter indefinitely and never reach a state where $x = 2 \vee y = 2$. Once we specify weak fairness, however, this liveness property holds, since we force either x or y to increment as long as it is possible for a non-stuttering step to be taken.

Weak fairness $WF_v(A)$ is formally defined as

$$WF_v(A) \equiv \Box(\Box\text{ENABLED}\langle A\rangle_v \Rightarrow \Box\langle A\rangle_v)$$

This formula says that, if a non-stuttering A step ever becomes forever enabled, then eventually a non-stuttering A step must be taken. The weak fairness formula can be written equivalently as

$$WF_v(A) \equiv \Diamond\Box(\text{ENABLED}\langle A\rangle_v) \Rightarrow \Box\Diamond\langle A\rangle_v$$

Strong Fairness

Weak fairness says that an action A must be eventually be taken if it is enabled *continuously* i.e. forever, after some point. Strong fairness, however, only asserts that an action is eventually taken if A is enabled *infinitely often*. This does not mean it must be enabled continuously, though. It is formally defined as

$$SF_v(A) \equiv \Box\Diamond\text{ENABLED}\langle A\rangle_v \Rightarrow \Box\Diamond\langle A\rangle_v$$

which is stating that if action A occurs infinitely often, then eventually A will be taken. we can compare the strong and weak fairness definitions

$$WF_v(A) \equiv \Diamond\Box(\text{ENABLED}\langle A\rangle_v) \Rightarrow \Box\Diamond\langle A\rangle_v$$
$$SF_v(A) \equiv \Box\Diamond(\text{ENABLED}\langle A\rangle_v) \Rightarrow \Box\Diamond\langle A\rangle_v$$

and see that they are equivalent except for their left hand side. Strong fairness requires a step to be taken only if it is always possible for an A step to be *eventually* taken, whereas weak fairness requires a step to be taken only if A is enabled continuously. Note that Lamport argues that a strong fairness condition should only be used in a specification if it is needed.

One concrete example illustrating the difference between weak and strong fairness is that of messages being sent between some sender and receiver. Let's say that the sender

sends a message by placing it in the inbox of the receiver. If there is currently a message in the inbox, then the "receive" action is enabled. If the channel is reliable, then once a message is sent, we can be sure that it will stay in the inbox until the receiver processes it. In other words, the receipt action will be continuously enabled. So, in this case, weak fairness on the actions of the receiver would be enough to ensure progress. The channel between sender and receiver, however, may be lossy. In this case, every time a message gets placed into the inbox it may subsequently get lost. So, it means that after sending, a message may not exist continuously in the inbox of the receiver. If the sender continuously resends a failed message, though, then the inbox of the receiver should be filled infinitely often, even though it is not filled continuously. In this case, we would need strong fairness on the actions of the receiver to guarantee progress.

Machine Closure

The canonical way to specify a system in TLA+ is to state is as a conjunction of an initial state predicate and next state relation i.e.

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

This only specifies what the system *may* do, but not what it *must* do. In order to specify the latter, we also need to specify a liveness condition:

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

Remember that in TLA+, though, there is no fundamental difference between a specification and a property. They are both defined as temporal logic formulas. In the case of a specification, we pick a "standard" model for describing the system that is convenient and, presumably, implementable to some extent. We could, however, simply describe our system in terms of the abstract properties it satisfies, without giving any kind of realistic, concrete specification. For example, we could say that our system consists of the set of all behaviors that satisfy some set of invariants. This can act as a specification, it is just a very abstract one. That is why we have to be careful with how we specify the *Liveness* condition. If we allowed *Liveness* to be an arbitrary temporal logic formula, the liveness condition would have too much potential specification power. For example, it could itself be a separate next-state relation that imposes extra conditions on how variables may transition. So, we have to impose some constraints on the types of formulas that can be used as a liveness condition. A spec of the form

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

is called *machine closed* if the conjunct *Liveness* constrains neither the initial state nor what steps may occur.