On TLA as a Logic

Martín Abadi[1] and Stephan Merz[2]

[1] Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, U.S.A.
[2] Institut für Informatik, TU München, Arcisstr. 21, 80290 München, Germany

Summary. We describe the Temporal Logic of Actions (TLA) from a logical perspective. After giving the syntax and semantics of TLA, we discuss some methods for representing reactive systems in TLA and study verification rules.

1 The L in TLA

The Temporal Logic of Actions (TLA) is a variant of temporal logic, designed for the specification and verification of reactive systems in terms of their actions. In this paper we describe TLA from a logical perspective; our description of TLA has three aspects:

1. As a logic, TLA has a precise syntax and semantics. We define these in the next section. Our intent is not to develop a new TLA, but rather to explain and to refine Lamport's definition of TLA [19].
2. Like HOL [13] and other logics, TLA can serve for representing reactive systems in several styles. In particular, a specification may describe concurrent steps as interleaved or simultaneous; communication between components may be synchronous or asynchronous. We discuss a few styles in section 3.
3. Proofs in TLA rely on basic rules of temporal logic, rules for refinement, and rules for composition. We state the principal rules in sections 4 and 5. Following [7, 8], we show that some of them arise from general logical (or algebraic) considerations, largely independent of the details of TLA

This paper is a self-contained presentation of TLA. It is however not a survey, in that it includes technical novelties and in that it is far from comprehensive.

Lamport's original work on TLA [19] provides much additional, useful material, and in particular some motivation for the TLA approach and a proof system for TLA. Other papers discuss mechanical verification in TLA [11, 16], refinement and composition [6, 4], real-time systems and hybrid systems [5, 18, 12], and medium-size examples [20]. There are also works on PTLA [1, 29], a propositional logic based on a preliminary version of TLA. Finally, the logic TLR has many similarities with TLA [28].

2 A Definition of TLA

In this section we define a syntax and a semantics for TLA. The definition is rather precise; it is intended to answer questions of detail, such as:

– what are the rules for substitution of terms for variables?
– what is stuttering equivalence?

With this goal in mind, we may err on the side of dotting too many i's.

Lamport has described TLA in fairly informal terms, leaving open some questions of this sort. These questions are often boring, but often necessary. They have come up frequently: they have been asked by confused TLA beginners, by meticulous referees, and by experienced colleagues who wished to mechanize TLA.

Lamport has also introduced a specification language based on TLA, named TLA+ [18, 20]. The definition of TLA+ is precise, and in fact includes a concrete syntax. Some advantages of TLA+ over TLA are that TLA+ provides a module system, abundant syntactic sugar, and a built-in set theory. The principal disadvantage of TLA+ may be its complexity: it is a complete language rather than a core logic.

2.1 Syntax and Informal Semantics

TLA has four tiers:

1. In one tier, we find formulas whose meaning is state-independent. They are called *constant*. *Rigid variables*, whose value is state-independent, may occur in constant formulas.
2. The second tier is concerned with reasoning about particular states. The formulas of this tier are called *state formulas*; they comprise *state functions* and *state predicates* (or *actions*). Both rigid variables and *flexible variables*, whose value is state-dependent, may occur in state formulas.
3. The third tier is concerned with reasoning about pairs of states. The formulas of this tier are called *transition formulas*; they comprise *transition functions* and *transition predicates*. Flexible variables may occur primed in transition formulas; the primed occurrences are evaluated at a different state than the others.
4. The fourth tier is concerned with reasoning about behaviors, which are infinite sequences of states. The formulas of this tier are called *temporal formulas* (or *behavior predicates*). They are built from formulas of the other tiers using temporal operators.

Next we cover some syntactic preliminaries, and then define the syntax of the third and fourth tiers in turn. We obtain the first and second tiers as special cases of the third. We also sketch an informal semantics; a possible-world semantics appears in section 2.2.

2.1.1 Basics TLA formulas are built from predicate and function symbols, variable symbols, and the special symbols $\neg$, $\wedge$, $\Box$, $\exists$, $\boldsymbol{\exists}$, $'$, and $=$. (In addition, TLA formulas include parentheses, which we use rather loosely.) We assume given:

- An infinite set of *variables* $\mathcal{V}$. These are partitioned into an infinite set $\mathcal{V}_R$ of *rigid* variables and an infinite set $\mathcal{V}_F$ of *flexible* variables.
- A sequence of symbols $\mathcal{L}$, partitioned into a sequence $\mathcal{L}_P$ of *predicate* symbols and a sequence $\mathcal{L}_F$ of *function* symbols. To each of the symbols in $\mathcal{L}$ is assigned a natural number, its *arity*.

These sets of symbols should be disjoint from each other and from the set of special symbols $\neg, \wedge, \ldots$. Moreover, no symbol in $\mathcal{V}$ should be of the form x'.

When writing TLA specifications, one usually does not present these sets of symbols explicitly. For example, it is common to assume without mention that x is a variable, that $\emptyset$ is a function symbol of arity 0, and that $+$ is a function symbol of arity 2. We make such assumptions below, in our examples. On the other hand, we cannot afford such informality in the definition of TLA.

2.1.2 Transition Formulas Let $\mathcal{V}_{F'} \triangleq \{x' \mid x \in \mathcal{V}_F\}$ be the set of primed flexible variables, and $\mathcal{V}_E \triangleq \mathcal{V}_R \cup \mathcal{V}_F \cup \mathcal{V}_{F'}$ be the set of rigid variables, flexible variables, and primed flexible variables.

A *transition function* is a first-order expression over the predicate and function symbols of $\mathcal{L}$ and over the variables of $\mathcal{V}_E$. A *transition predicate* is a first-order predicate over the predicate and function symbols of $\mathcal{L}$ and over the variables of $\mathcal{V}_E$. Transition predicates are commonly called *actions*. For example, if $f \in \mathcal{L}_F$ is a function symbol of arity 3, $p \in \mathcal{L}_P$ is a predicate symbol of arity 1, $x \in \mathcal{V}_R$, and $y \in \mathcal{V}_F$, then $f(x, y, y')$ is a transition function and $\exists y'.(p(f(x, y, y')) \wedge \neg(x = y'))$ is a transition predicate.

The following inductive definitions are more explicit (but equivalent). The set of transition functions is the smallest set such that:

- If $x \in \mathcal{V}_E$ then x is a transition function.
- If $f \in \mathcal{L}_F$ is an n-ary function symbol and $v_1, \ldots, v_n$ are transition functions then $f(v_1, \ldots, v_n)$ is a transition function.

The set of transition predicates is the smallest set such that:

- If v_1 and v_2 are transition functions then $v_1 = v_2$ is a transition predicate.
- If $p \in \mathcal{L}_P$ is an n-ary predicate symbol and $v_1, \ldots, v_n$ are transition functions then $p(v_1, \ldots, v_n)$ is a transition predicate.
- If A is a transition predicate then so is $\neg A$.
- If A and B are transition predicates then so is $(A \wedge B)$.
- If $x \in \mathcal{V}_E$ and A is a transition predicate then so is $\exists x.A$.

The definitions of *free variables* and *substitution* are the usual ones for first-order logic (over the set of variables $\mathcal{V}_E$). We write $\mathrm{FV}_{\mathrm{trans}}(v)$ and $\mathrm{FV}_{\mathrm{trans}}(A)$ for the sets of free variables of v and A, respectively; these are subsets of $\mathcal{V}_E$. Given a variable $x \in \mathcal{V}_E$, we write $v\{a/x\}$ and $A\{a/x\}$ for the results of substituting the transition function a for the free occurrences of x in v and A, respectively. For example, $\mathrm{FV}_{\mathrm{trans}}(x = x')$ is $\{x, x'\}$ and the substitution $(x = x')\{y/x'\}$ yields $x = y$.

A *state function* is a transition function with no free primed variables (that is, the transition function v is a state function iff $\mathrm{FV}_{\mathrm{trans}}(v) \cap \mathcal{V}_{F'} = \emptyset$). Analogously, a *state predicate* is a transition predicate with no free primed variables. If v is a state function then v' is an abbreviation for the result of priming the free flexible variables of v: if $\mathrm{FV}_{\mathrm{trans}}(v) \cap \mathcal{V}_F = \{x_1, \ldots, x_n\}$ then v' is $v\{x'_1/x_1\}\{\ldots/\ldots\}\{x'_n/x_n\}$. If P is a state predicate then P' is defined similarly.

Further, a *constant function* is a state function with no free flexible variables, and a *constant predicate* is a state predicate with no free flexible variables. If v is a constant function then v' equals v.

2.1.3 Temporal Formulas Behavior predicates are the only temporal formulas. At present, TLA has no corresponding notion of behavior function.

The set of *behavior predicates* is the smallest set such that:

- If P is a state predicate then P is a behavior predicate.
- If A is a transition predicate and v is a state function then $\Box[A]_v$ is a behavior predicate.
- If F is a behavior predicate then so is $\neg F$.
- If F and G are behavior predicates then so is $(F \wedge G)$.
- If F is a behavior predicate then so is $\Box F$.
- If $x \in \mathcal{V}_R$ and F is a behavior predicate then so is $\exists x.F$.
- If $x \in \mathcal{V}_F$ and F is a behavior predicate then so is $\boldsymbol{\exists} x.F$.

Thus, if A is a transition predicate, then $\Box A$ is a behavior predicate only if A is in fact a state predicate. This restriction is designed to make possible the proof of Proposition 2.1, given below.

We write $\mathrm{FV}_{\mathrm{temp}}(F)$ for the set of *free variables* of the behavior predicate F; this is a subset of $\mathcal{V}$. We define:

$$\begin{array}{rcl}
\mathrm{FV}_{\mathrm{temp}}(P) & = & \mathrm{FV}_{\mathrm{trans}}(P) \\
\mathrm{FV}_{\mathrm{temp}}(\Box[A]_v) & = & \{x \in \mathcal{V} \mid x \in \mathrm{FV}_{\mathrm{trans}}(A) \text{ or } x' \in \mathrm{FV}_{\mathrm{trans}}(A)\} \\
 & & \cup\, \mathrm{FV}_{\mathrm{trans}}(v) \\
\mathrm{FV}_{\mathrm{temp}}(\neg F) & = & \mathrm{FV}_{\mathrm{temp}}(F) \\
\mathrm{FV}_{\mathrm{temp}}(F \wedge G) & = & \mathrm{FV}_{\mathrm{temp}}(F) \cup \mathrm{FV}_{\mathrm{temp}}(G) \\
\mathrm{FV}_{\mathrm{temp}}(\Box F) & = & \mathrm{FV}_{\mathrm{temp}}(F) \\
\mathrm{FV}_{\mathrm{temp}}(\exists x.F) & = & \mathrm{FV}_{\mathrm{temp}}(F) - \{x\} \\
\mathrm{FV}_{\mathrm{temp}}(\boldsymbol{\exists} x.F) & = & \mathrm{FV}_{\mathrm{temp}}(F) - \{x\}
\end{array}$$

This definition is somewhat ambiguous (and so are several others below). In particular, if P is a state predicate, then $\mathrm{FV}_{\mathrm{temp}}(\neg P)$ is defined in two ways, as $\mathrm{FV}_{\mathrm{temp}}(P)$ and as $\mathrm{FV}_{\mathrm{trans}}(P)$. Fortunately, the two definitions coincide. This slight problem can be avoided altogether by making explicit the coercion from the set of state formulas to the set of behavior predicates.

We also define a form of *substitution*. Given a rigid or flexible variable $y \in \mathcal{V}$, a state function b, and a behavior predicate F, we write $F[b/y]$ for the result of substituting b for the free occurrences of y in F. When y is rigid ($y \in \mathcal{V}_R$), the substitution $F[b/y]$ is useful to us only when b is in fact a constant function. However, the definitions of $F[b/y]$ for $y \in \mathcal{V}_R$ and $y \in \mathcal{V}_F$ are mostly identical, so we treat them together:

$$\begin{array}{rcl}
P[b/y] & = & P\{b/y\} \\
(\Box[A]_u)[b/y] & = & \Box[A\{b/y\}]_{u\{b/y\}} \quad \text{if } y \in \mathcal{V}_R \\
(\Box[A]_u)[b/y] & = & \Box[A\{b/y\}\{b'/y'\}]_{u\{b/y\}} \quad \text{if } y \in \mathcal{V}_F \\
(\neg F)[b/y] & = & \neg(F[b/y]) \\
(F \wedge G)[b/y] & = & F[b/y] \wedge G[b/y] \\
(\Box F)[b/y] & = & \Box(F[b/y]) \\
(\exists x.F)[b/y] & = & \exists x.(F[b/y]) \quad \text{if } x \notin \mathrm{FV}_{\mathrm{trans}}(b) \cup \{y\} \\
(\exists y.F)[b/y] & = & \exists y.F \\
(\boldsymbol{\exists} x.F)[b/y] & = & \boldsymbol{\exists} x.(F[b/y]) \quad \text{if } x \notin \mathrm{FV}_{\mathrm{trans}}(b) \cup \{y\} \\
(\boldsymbol{\exists} y.F)[b/y] & = & \boldsymbol{\exists} y.F
\end{array}$$

The definition is partial because the two clauses for existential quantification require $x \notin \mathrm{FV}_{\mathrm{trans}}(b)$. In other words, $F[b/y]$ is defined only when b is free for y in F (and some renaming is needed otherwise). This simplifies the definition while preventing the capture of variables.

These definitions should be contrasted with the corresponding definitions for transition formulas. According to these definitions, if $x \in \mathcal{V}_F$ is a flexible variable, then the x in x' is treated as an occurrence of x; in the definitions for transition formulas, x' is viewed as a separate variable, unrelated to x. For example, $\mathrm{FV}_{\mathrm{temp}}(\Box[x = x']_x)$ is $\{x\}$, while $\mathrm{FV}_{\mathrm{trans}}(x = x')$ is $\{x, x'\}$. Similarly, $(\Box[x = x']_x)[b/x]$ yields $(\Box[b = b']_b)$, while $(x = x')\{b/x\}$ yields $b = x'$.

2.1.4 Informal semantics The meaning of a constant formula is its usual first-order meaning. A constant formula is like a constant expression in a programming language, hence the name (borrowed from TLA+).

A state function is like an expression in a programming language. Semantically, a state is a mapping from the set of flexible variables to a set of values; the value of rigid variables is state-independent. A state function has a value at each state. Similarly, a state predicate is either true or false at each state.

Unlike a state formula, a transition formula is not evaluated at a state, but at a pair of states. Given a pair of states, the primed variables refer to the second state and the unprimed variables to the first. For example, $p(x, y')$

is true at a pair of states (s,t) iff p holds for the value of x in s and the value of y in t. An A *step* is a pair of states satisfying A.

A behavior is an infinite sequence of states. A behavior predicate is true or false of a behavior:

- A state predicate is true of a behavior iff it is true of its first state.
- The Boolean connectives $\neg$ and $\wedge$ are standard. So is quantification over rigid variables: $\exists x.F$ means that there is some way of choosing a value for x such that F holds.
- The $\Box$ operator is the "always" operator of temporal logic. As usual, $\Box F$ means that F is true always in the future.
- Similarly, $\Box[A]_v$ means that, in the future, every pair of consecutive states satisfies A or $v' = v$.
- The formula $\boldsymbol{\exists} x.F$ roughly means that there is some way of choosing a sequence of values for x such that F holds. We call x an internal or a hidden variable of $\boldsymbol{\exists} x.F$.

2.1.5 Abbreviations Many abbreviations are commonly used in TLA. We introduce some of the essential ones.

The Boolean abbreviations true, false, $\vee$, $\Rightarrow$, and $\equiv$ are the usual ones, and so is the definition of $\forall x.F$ as $\neg\exists x.\neg F$. Analogously, $\boldsymbol{\forall} x.F$ stands for $\neg\boldsymbol{\exists} x.\neg F$.

By far the most delicate abbreviation is the one for writing "enabled" predicates:

- If A is a transition predicate then *Enabled* A is an abbreviation for the state predicate obtained by existentially quantifying the free primed variables of A (in any order): if $\mathrm{FV}_{\mathrm{trans}}(A) \cap \mathcal{V}_{F'} = \{x'_1, \ldots, x'_n\}$ then *Enabled* A is $\exists x'_1. \ldots \exists x'_n.A$.
  We say that A is enabled in state s iff *Enabled* A is true at s. Thus, A is enabled in s iff there exists a state t such that (s,t) is an A step.

In defining the remaining abbreviations, we assume that v, v_1, ..., v_n are state functions, that A is a transition predicate, and that F is a behavior predicate. Under these assumptions:

- $\Box[A]_{v_1,\ldots,v_n}$ stands for the behavior predicate $\Box[A]_{v_1} \wedge \ldots \wedge \Box[A]_{v_n}$. This predicate is a generalization of $\Box[A]_v$; it means that, in the future, every pair of consecutive states satisfies A or $v'_1 = v_1 \wedge \ldots \wedge v'_n = v_n$.
- $\Diamond\langle A\rangle_{v_1,\ldots,v_n}$ stands for $\neg\Box[\neg A]_{v_1,\ldots,v_n}$, and means that some future pair of consecutive states satisfies A and $v'_1 \neq v_1 \vee \ldots \vee v'_n \neq v_n$.
- $\Diamond F$ stands for the behavior predicate $\neg\Box\neg F$. The $\Diamond$ operator is the "sometime" operator of temporal logic. As usual, $\Diamond F$ means that F is true sometime in the future.
- $\mathrm{WF}_{v_1,\ldots,v_n}(A)$ stands for the behavior predicate

$$\Diamond\Box(\mathit{Enabled}\ (A \wedge (v'_1 \neq v_1 \vee \ldots \vee v'_n \neq v_n)) \Rightarrow \Box\Diamond\langle A\rangle_{v_1,\ldots,v_n}$$

This is a *weak fairness* formula; it holds for a behavior if either there are infinitely many A steps where at least one of $v_1, \ldots, v_n$ changes, or there are infinitely many states from which such a step is impossible.

– Similarly, $\mathrm{SF}_{v_1,\ldots,v_n}(A)$ stands for the behavior predicate

$$\Box \Diamond (\mathit{Enabled}\ (A \wedge (v_1' \neq v_1 \vee \ldots \vee v_n' \neq v_n)) \Rightarrow \Box \Diamond \langle A \rangle_{v_1,\ldots,v_n}$$

This is a *strong fairness* formula; it holds for a behavior if either there are infinitely many A steps where at least one of $v_1, \ldots, v_n$ changes, or there are only finitely many states from which such a step is possible.

2.2 Possible-World Semantics

We formalize the semantics of TLA in terms of possible worlds, much as usual for modal logics. The sequence of definitions for the semantics mostly parallels that for the syntax.

2.2.1 Basics Just as the sequence of symbols $\mathcal{L}$ is usually not given explicitly, the first-order structure that underlies the meaning of a TLA formula is usually taken for granted. A *structure* $\mathcal{M}$ is a non-empty set $\mathcal{U}$ together with a sequence of *relations* on $\mathcal{U}$ and a sequence of *functions* on $\mathcal{U}$. Each relation and each function has a natural number as *arity*.

The structure $\mathcal{M}$ is a structure for $\mathcal{L}$ if the number of relations and the number of functions in $\mathcal{M}$ are equal to the number of predicate symbols and the number of function symbols in $\mathcal{L}$, and if the sequences of arities for $\mathcal{M}$ and $\mathcal{L}$ are identical. When $\mathcal{M}$ is a structure for $\mathcal{L}$, there is an evident bijection C from the symbols in $\mathcal{L}$ to relations and functions. In the following definitions, $\mathcal{M}$ and C are fixed.

An *interpretation* over a set of variables $\mathcal{W}$ is a mapping from $\mathcal{W}$ to $\mathcal{U}$. A *state* is simply an interpretation over $\mathcal{V}_F$. If α is an interpretation over $\mathcal{W}$, $x \in \mathcal{W}$, and $e \in \mathcal{U}$, then $\alpha\{x \leftarrow e\}$ equals α except that it maps x to e. If α and β are both interpretations over $\mathcal{W}$, then α and β are *similar up to* x, written $\alpha \simeq_x \beta$, iff $\beta = \alpha\{x \leftarrow e\}$ for some e.

Next we define the semantics of TLA expressions. When t is a classical, first-order expression, we write $[\![t]\!]_\alpha$ for the meaning of t under interpretation α. We use the same notation independently of the sort of t; the meaning of t may be either a truth value or an element of the universe $\mathcal{U}$. In general, when t is an arbitrary TLA expression, we write $[\![t]\!]_{\alpha,\theta}$ for the meaning of t with interpretation α, and state, pair of states, or behavior θ. We rely on the type of θ to resolve ambiguities.

2.2.2 Transition Formulas The semantics of first-order expressions is the usual one:

– If $x \in \mathcal{V}_E$ then $[\![x]\!]_\alpha$ is $\alpha(x)$.
– $[\![f(v_1, \ldots, v_n)]\!]_\alpha$ is $\mathrm{C}(f)([\![v_1]\!]_\alpha, \ldots, [\![v_n]\!]_\alpha)$.

and

- $[\![v_1 = v_2]\!]_\alpha$ is true iff $[\![v_1]\!]_\alpha$ and $[\![v_2]\!]_\alpha$ are equal.
- $[\![p(v_1, \ldots, v_n)]\!]_\alpha$ is true iff $\mathrm{C}(p)([\![v_1]\!]_\alpha, \ldots, [\![v_n]\!]_\alpha)$ is true.
- $[\![\neg A]\!]_\alpha$ is true iff $[\![A]\!]_\alpha$ is not.
- $[\![A \wedge B]\!]_\alpha$ is true iff both $[\![A]\!]_\alpha$ and $[\![B]\!]_\alpha$ are.
- $[\![\exists x.A]\!]_\alpha$ is true iff $[\![A]\!]_\beta$ is true for some $\beta \simeq_x \alpha$.

Given an interpretation α over $\mathcal{V}_R$ and a pair of states (s, t), we define an interpretation $E(\alpha, s, t)$ over $\mathcal{V}_E$ by:

$$\begin{array}{rcll} E(\alpha,s,t)(x) & = & \alpha(x) & \text{if } x \in \mathcal{V}_R \\ E(\alpha,s,t)(x) & = & s(x) & \text{if } x \in \mathcal{V}_F \\ E(\alpha,s,t)(x') & = & t(x) & \text{if } x \in \mathcal{V}_F \end{array}$$

The semantics $[\![\ldots]\!]_{\alpha,(s,t)}$ of a transition formula under α at a pair of states (s,t) is its first-order semantics $[\![\ldots]\!]_{E(\alpha,s,t)}$ under $E(\alpha, s, t)$:

$$\begin{array}{rcl} [\![v]\!]_{\alpha,(s,t)} & = & [\![v]\!]_{E(\alpha,s,t)} \\ [\![A]\!]_{\alpha,(s,t)} & = & [\![A]\!]_{E(\alpha,s,t)} \end{array}$$

The semantics of a state formula at a pair of states does not depend on the second state of the pair. Hence, when P and b are state formulas, we may shorten $[\![P]\!]_{\beta,(s,t)}$ and $[\![b]\!]_{\beta,(s,t)}$ to $[\![P]\!]_{\beta,s}$ and $[\![b]\!]_{\beta,s}$. Similarly, the semantics of a constant formula at a pair of states does not depend on the states at all.

2.2.3 Temporal Formulas A *behavior* is an infinite sequence of states. If σ is $s_0, s_1, \ldots$, then $\sigma|_n$ is its suffix $s_n, s_{n+1}, \ldots$. The result of prefixing the finite sequence ρ to σ is $\rho \circ \sigma$. The finite sequence consisting of $t_0, t_1, \ldots, t_m$ is $\langle t_0, t_1, \ldots, t_m \rangle$.

Stuttering equivalence is the finest equivalence relation on behaviors such that any two behaviors $\rho \circ \langle t, t \rangle \circ \sigma$ and $\rho \circ \langle t \rangle \circ \sigma$ are stuttering equivalent. Two behaviors $\sigma = s_0, s_1, \ldots$ and $\tau = t_0, t_1, \ldots$ are *equal up to x* iff $s_i \simeq_x t_i$ for all i. They are *similar up to x*, written $\sigma \simeq_x \tau$, iff there exists σ' and τ' such that:

- σ' and τ' are equal up to x;
- σ and σ' are stuttering equivalent;
- τ and τ' are stuttering equivalent.

Given an interpretation α over $\mathcal{V}_R$ and a behavior $\sigma = s_0, s_1, \ldots$, we extend the semantics to behavior predicates:

- $[\![P]\!]_{\alpha,\sigma}$ is $[\![P]\!]_{\alpha,s_0}$.
- $[\![\Box[A]_v]\!]_{\alpha,\sigma}$ is true iff $[\![A \vee (v' = v)]\!]_{\alpha,(s_n,s_{n+1})}$ is true for all $n \geq 0$.
- $[\![\neg F]\!]_{\alpha,\sigma}$ is true iff $[\![F]\!]_{\alpha,\sigma}$ is not.
- $[\![F \wedge G]\!]_{\alpha,\sigma}$ is true iff both $[\![F]\!]_{\alpha,\sigma}$ and $[\![G]\!]_{\alpha,\sigma}$ are.
- $[\![\Box F]\!]_{\alpha,\sigma}$ is true iff $[\![F]\!]_{\alpha,\sigma|_n}$ is true for all $n \geq 0$.
- If $x \in \mathcal{V}_R$ then $[\![\exists x.F]\!]_{\alpha,\sigma}$ is true iff $[\![F]\!]_{\beta,\sigma}$ is true for some $\beta \simeq_x \alpha$.
- If $x \in \mathcal{V}_F$ then $[\![\boldsymbol{\exists} x.F]\!]_{\alpha,\sigma}$ is true iff $[\![F]\!]_{\alpha,\tau}$ is true for some $\tau \simeq_x \sigma$.

A logic is invariant under stuttering when none of its formulas can distinguish between two stuttering-equivalent behaviors. Invariance under stuttering is important in connection with refinement [17]. A fundamental result about TLA is that it is invariant under stuttering, namely:

Proposition 2.1 (Invariance under stuttering). *If F is a behavior predicate, α is an interpretation over $\mathcal{V}_R$, and σ and τ are two stuttering-equivalent behaviors, then $[\![F]\!]_{\alpha,\sigma} = [\![F]\!]_{\alpha,\tau}$.*

PROOF: We use the following simple facts:

1. If σ and τ are stuttering equivalent, then for every $n \geq 0$ there exists $m \geq 0$ such that $\sigma|_n$ and $\tau|_m$ are stuttering equivalent.
2. If $\sigma = s_0, s_1, \ldots$ and $\tau = t_0, t_1, \ldots$ are stuttering equivalent then
   a. $t_0 = s_0$, and
   b. $t_1 = s_1$ or $t_1 = s_0$.
3. For any $x \in \mathcal{V}_F$, similarity up to x ($\simeq_x$) is an equivalence relation on the set of behaviors.

The proof is by induction on the structure of F:

1. CASE: F is a state predicate.
   PROOF: Since σ and τ are stuttering equivalent, Fact 2a implies $s_0 = t_0$. Therefore, $[\![F]\!]_{\alpha,\sigma} = [\![F]\!]_{\alpha,s_0} = [\![F]\!]_{\alpha,t_0} = [\![F]\!]_{\alpha,\tau}$.
2. CASE: F is $\Box[A]_v$ where A is a transition predicate and v is a state function.
   PROOF: Since stuttering equivalence is an equivalence relation, by the definition of $[\![\Box[A]_v]\!]_{\alpha,\sigma}$ it suffices to:
   ASSUME: 1. $[\![\Box[A]_v]\!]_{\alpha,\tau}$ is true.
   2. $n \geq 0$
   PROVE: $[\![A \vee (v' = v)]\!]_{\alpha,(s_n,s_{n+1})}$ is true.
   2.1. Choose $m \geq 0$ such that $\sigma|_n$ and $\tau|_m$ are stuttering equivalent.
   PROOF: Such m exists by the assumption that σ and τ are stuttering equivalent, and Fact 1.
   2.2. $s_n = t_m$
   PROOF: By the choice of m in Step 2.1, and Fact 2a.
   2.3. CASE: $s_{n+1} = t_{m+1}$
   PROOF: Step 2.2 and the case assumption imply $(s_n, s_{n+1}) = (t_m, t_{m+1})$, and hence $[\![A \vee (v' = v)]\!]_{\alpha,(s_n,s_{n+1})} = [\![A \vee (v' = v)]\!]_{\alpha,(t_m,t_{m+1})}$; since $[\![\Box[A]_v]\!]_{\alpha,\tau}$ is true by assumption, $[\![A \vee (v' = v)]\!]_{\alpha,(t_m,t_{m+1})}$ is true.
   2.4. CASE: $s_{n+1} = t_m$
   PROOF: Step 2.2 and the case assumption imply $s_n = s_{n+1}$. Therefore, $[\![v' = v]\!]_{\alpha,(s_n,s_{n+1})}$ is true, and hence $[\![A \vee (v' = v)]\!]_{\alpha,(s_n,s_{n+1})}$ is true.
   2.5. Q.E.D.
   PROOF: The choice of m in Step 2.1 and Fact 2b ensure that $s_{n+1} = t_{m+1}$ or $s_{n+1} = t_m$. The assertion follows by Steps 2.3 and 2.4.

3. Case: F is $\neg G$ for some behavior predicate G.
   Proof: By the induction hypothesis, $[\![G]\!]_{\alpha,\sigma} = [\![G]\!]_{\alpha,\tau}$, and this immediately implies the assertion.
4. Case: F is $G \wedge H$ for some behavior predicates G, H.
   Proof: $[\![F]\!]_{\alpha,\sigma}$ is true
   iff $[\![G]\!]_{\alpha,\sigma}$ and $[\![H]\!]_{\alpha,\sigma}$ are true
   iff $[\![G]\!]_{\alpha,\tau}$ and $[\![H]\!]_{\alpha,\tau}$ are true (by induction hypothesis)
   iff $[\![F]\!]_{\alpha,\tau}$ is true
5. Case: F is $\Box G$ for some behavior predicate G.
   Proof: Since stuttering equivalence is an equivalence relation, by the definition of $[\![\Box G]\!]_{\alpha,\sigma}$ it suffices to:
   Assume: 1. $[\![\Box G]\!]_{\alpha,\tau}$ is true.
   2. $n \geq 0$
   Prove: $[\![G]\!]_{\alpha,\sigma|_n}$ is true.
   5.1. Choose $m \geq 0$ such that $\sigma|_n$ and $\tau|_m$ are stuttering equivalent.
   Proof: Such m exists by the assumption that σ and τ are stuttering equivalent, and Fact 1.
   5.2. Q.E.D.
   Proof: The assumption that $[\![\Box G]\!]_{\alpha,\tau}$ is true implies that $[\![G]\!]_{\alpha,\tau|_m}$ is true for all $m \geq 0$. Step 5.1 and the induction hypothesis imply that $[\![G]\!]_{\alpha,\sigma|_n}$ is true.
6. Case: F is $\exists x.G$ for some behavior predicate G and $x \in \mathcal{V}_R$.
   Proof: $[\![F]\!]_{\alpha,\sigma}$ is true
   iff $[\![G]\!]_{\beta,\sigma}$ is true for some $\beta \simeq_x \alpha$
   iff $[\![G]\!]_{\beta,\tau}$ is true for some $\beta \simeq_x \alpha$ (by induction hypothesis)
   iff $[\![F]\!]_{\alpha,\tau}$ is true
7. Case: F is $\boldsymbol{\exists} x.G$ for some behavior predicate G and $x \in \mathcal{V}_F$.
   Proof: $[\![F]\!]_{\alpha,\sigma}$ is true
   iff $[\![G]\!]_{\alpha,\rho}$ is true for some $\rho \simeq_x \sigma$
   iff $[\![G]\!]_{\alpha,\rho}$ is true for some $\rho \simeq_x \tau$, by the assumption that σ and τ are stuttering equivalent (and therefore $\sigma \simeq_x \tau$), since $\simeq_x$ is transitive (Fact 3)
   iff $[\![F]\!]_{\alpha,\tau}$ is true
8. Q.E.D.
   Proof: From Steps 1–7 by induction on the definition of behavior predicates. □

2.2.4 Validity Given a structure $\mathcal{M}$ for $\mathcal{L}$, the state predicate P is $\mathcal{M}$-valid iff $[\![P]\!]_{\alpha,s}$ is true for all interpretations α and states s. Similarly, a transition predicate A is $\mathcal{M}$-valid iff $[\![A]\!]_{\alpha,(s,t)}$ is true for all interpretations α and states s and t. Finally, a behavior predicate F is $\mathcal{M}$-valid iff $[\![F]\!]_{\alpha,\sigma}$ is true for all interpretations α and behaviors σ.

More generally, given a class of structures $\mathcal{S}$ (for example, the models of a first-order theory), a formula is $\mathcal{S}$-valid iff it is $\mathcal{M}$-valid for all $\mathcal{M}$ in $\mathcal{S}$.

Often, the class of structures of interest is clear from context, and then we say that the formula is valid.

There can be a slight ambiguity, since "P is valid" may mean "P is a valid state predicate" or "P is a valid temporal formula"; fortunately, these two readings are equivalent.

2.3 Notes

Other presentations of TLA are possible. The following are some notes on the choices that we made consciously in our presentation, and on alternatives.

2.3.1 TLA and "Ordinary Logic" Throughout, we have chosen to exploit the syntax and semantics of ordinary, first-order logic in defining those of TLA. In particular, our transition formulas are simply first-order formulas over a large set of variables ($\mathcal{V}_E$). To obtain this, it is important that $'$ be applied only to variables.

In an alternative definition, we could have taken $'$ to be an operator that can be applied to any state function. With that definition, $'$ would be treated much like the "next" operator ($\circ$) of temporal logic.

2.3.2 Subscripts Invariance under stuttering is part of the essence of TLA. Syntactically, this means that subscripts are part of the essence of TLA. The use of subscripts has given rise to many abbreviations, and to many different conventions.

Traditionally $\Box[A]_{v_1,\ldots,v_n}$ is not an abbreviation for $\Box[A]_{v_1} \wedge \ldots \wedge \Box[A]_{v_n}$. Instead, one writes $\Box[A]_{(v_1,\ldots,v_n)}$ where $(v_1, \ldots, v_n)$ is the tuple of the state functions $v_1, \ldots, v_n$. If the underlying first-order language is sufficiently rich, tupling is definable, so $(v_1, \ldots, v_n)$ is in fact a state function. We have taken $\Box[A]_{v_1,\ldots,v_n}$ as an abbreviation in order to avoid assumptions on the first-order language. In particular, we can reason about finite domains, where tupling is not available.

Another common convention is to take $[A]_v$ as an abbreviation for $A \vee (v' = v)$. This convention can be useful, but we avoid it in order to simplify the parsing of formulas.

2.3.3 TLA+ From a logical point of view, TLA+ is essentially a special case of TLA, with many added definable constructs.

The main differences between TLA as we have described it and its formalization in TLA+ are syntactic. In TLA+, $\mathcal{L}$ consists of a single predicate symbol $\in$ with arity 2. In addition, TLA+ includes Hilbert's choice operator ε. More importantly, TLA+ provides some syntactic sugar and a module system, both useful for writing specifications in practice.

In TLA+, the structures of interest are models of a set theory; they are equipped with one binary membership relation and no functions.

3 Representing a Component

Like other logics, TLA supports several different styles for modelling a component. We now discuss, by means of an example, some of these styles. Which style is most appropriate depends on the problem at hand.

All the styles we describe have some common logical aspects:

- Temporal formulas are used both for describing components and for specifying their properties.
- Variable hiding is represented by existential quantification.
- The composition of two components is represented by the conjunction of their specifications: composition means logical conjunction.
- A component implements a property if the formula for the component implies the formula for the property: implementation means logical implication.

For simplicity, we explain how to describe a component with one input ("environment") variable e, one output ("module") variable m, and one internal variable x. Correspondingly, we use three flexible variables, e, m, and x. Other situations are discussed in [4], and in particular the important special case where the component is a complete system, with no input from the outside.

3.1 The Standard Interleaving Style

An interleaving representation of a component is one that disallows simultaneous steps by the component and its environment. Interleaving representations are studied in some detail in [4]. We call standard the style of specification developed there.

3.1.1 The Form of a Specification In the standard interleaving style, a specification has the form

$$\exists x.(\mathit{Init} \wedge \Box[N]_{m,x} \wedge L)$$

where:

Init is a state predicate describing the initial values of m and x.

N is a transition predicate describing the component steps. Since the component does not change its input variable, N should imply $e' = e$.

m, x appears as the subscript for N because $\Box[N]_{m,x}$ allows any state change that leaves m and x unchanged. Such a state change can affect only e and variables not mentioned in the specification, so we think of it as an environment step. Thus, according to $\Box[N]_{m,x}$, the environment may do anything but change m and x.

L is the conjunction of fairness conditions, each of the form $\mathrm{WF}_{m,x}(A)$ or $\mathrm{SF}_{m,x}(A)$. It is common that N is a disjunction, and that each A is one of N's disjuncts.

The specification disallows simultaneous changes of the input variable and the output variable. In fact, the specification without the quantifier (that is, $Init \wedge \Box[N]_{m,x} \wedge L$) also disallows simultaneous changes of the input variable and the internal variable. More precisely, if

$$N \;\Rightarrow\; e' = e$$

is a valid transition formula then

$$\exists x.(Init \wedge \Box[N]_{m,x} \wedge L) \;\Rightarrow\; \Box[e' = e \vee m' = m]_{e,m}$$

and

$$(Init \wedge \Box[N]_{m,x} \wedge L) \;\Rightarrow\; \Box[e' = e \vee (m' = m \wedge x' = x)]_{e,m,x}$$

are valid temporal formulas.

3.1.2 A Lossy Queue As an illustration of the standard interleaving style, we specify a simple, lossy queue. Figure 1 shows a picture of this lossy queue. The lossy queue's interface consists of two "wires", i for input and o for output. Because there is no acknowledgment protocol, inputs may be lost. Similarly, an input may be added to the lossy queue several times.

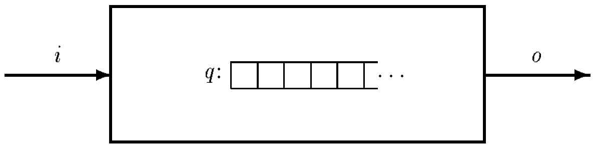

Fig. 1. A simple queue.

The specification of the lossy queue is shown in Figure 2. In the specification, a list of formulas, each prefaced by $\wedge$, denotes the conjunction of the formulas, and indentation is used to eliminate parentheses.

The specification is a temporal formula that mentions the flexible variables i and o, as well as the flexible variable q, which equals the sequence of messages received but not yet output; q is hidden by quantification. The specification uses standard predicate and function symbols for sequences; in particular, $Head(q)$ denotes the first element of q, and $Tail(q)$ denotes the sequence obtained by removing the first element of q.

The state predicate $Init_Q$ describes the initial state. It asserts that the values of i and o are equal, and the value of q is the empty sequence.

The action Enq represents the receipt of a message by the lossy queue. This action is always enabled. The action Deq represents the operation of removing a message from the head of q and sending it on the output wire. This action is enabled iff the value of q is not the empty sequence. The action N_Q is the specification's complete next-state relation.

$$
\begin{array}{lcl}
\mathit{Init}_Q & \triangleq & \land\ o = i \\
 & & \land\ q = \langle\rangle \\
\mathit{Enq} & \triangleq & \land\ q' = q \circ \langle i \rangle \\
 & & \land\ i' = i \\
 & & \land\ o' = o \\
\mathit{Deq} & \triangleq & \land\ q \neq \langle\rangle \\
 & & \land\ o' = \mathit{Head}(q) \\
 & & \land\ q' = \mathit{Tail}(q) \\
 & & \land\ i' = i \\
N_Q & \triangleq & \mathit{Enq} \lor \mathit{Deq} \\
L_Q & \triangleq & \mathrm{WF}_{o,q}(\mathit{Enq}) \land \mathrm{WF}_{o,q}(\mathit{Deq}) \\
\Pi_Q & \triangleq & \mathit{Init}_Q \land \Box[N_Q]_{o,q} \land L_Q \\
\Phi_Q & \triangleq & \boldsymbol{\exists} q.\Pi_Q
\end{array}
$$

Fig. 2. A specification of a lossy queue.

The formula L_Q is the fairness specification of the lossy queue. It consists of fairness conditions for Enq and Deq.

The formula Π_Q is the complete specification of the lossy queue before q is quantified. The first conjunct, Init_Q, describes the initial state. The second conjunct, $\Box[N_Q]_{o,q}$, asserts that every step is either an N_Q step or leaves o and q unchanged. The third conjunct is L_Q.

The formula Φ_Q is the actual specification. The free flexible variables of Φ_Q are only i and o, while q is existentially quantified.

Writing this TLA specification of the lossy queue is not much different from writing a corresponding piece of code, except that a TLA formula has a precise meaning and can be manipulated with logical rules, while a piece of code can be executed.

The same TLA is used for expressing the properties expected of the lossy queue. For example, the temporal formula

$$\forall u.(\Diamond \Box (i = u) \Rightarrow \Diamond \Box (o = u))$$

expresses that if eventually i settles to a value u then eventually o settles to the same value u. The lossy queue implements this property. More precisely, for structures where the queue operations have the expected meaning, the temporal formula $\Phi_Q \Rightarrow \forall u.(\Diamond \Box (i = u) \Rightarrow \Diamond \Box (o = u))$ is valid.

3.1.3 The Lossy Queue, da Capo The specification of the lossy queue is simple, but peculiar. (In the terminology of [2], the specification is neither fin nor internally continuous.) For example, consider a behavior where i remains constant. In this case, an implementation of the lossy queue could simply do nothing, leaving o equal to i. Curiously, however, the weak fairness condition on the Enq action implies that Enq has to take place infinitely often.

$$
\begin{array}{lcl}
Init_Q & \triangleq & \land\ o = i \\
 & & \land\ q = \langle\rangle \\
Enq^{\dagger} & \triangleq & \land\ i \neq Last(\langle o\rangle \circ q) \\
 & & \land\ q' = q \circ \langle i\rangle \\
 & & \land\ i' = i \\
 & & \land\ o' = o \\
Deq & \triangleq & \land\ q \neq \langle\rangle \\
 & & \land\ o' = Head(q) \\
 & & \land\ q' = Tail(q) \\
 & & \land\ i' = i \\
N_Q^{\dagger} & \triangleq & Enq^{\dagger} \lor Deq \\
L_Q^{\dagger} & \triangleq & \mathrm{WF}_{o,q}(Enq^{\dagger}) \land \mathrm{WF}_{o,q}(Deq) \\
\Pi_Q^{\dagger} & \triangleq & Init_Q \land \Box[N_Q^{\dagger}]_{o,q} \land L_Q^{\dagger} \\
\Phi_Q^{\dagger} & \triangleq & \boldsymbol{\exists} q. \Pi_Q^{\dagger}
\end{array}
$$

Fig. 3. An alternative specification of a lossy queue.

We reformulate the specification of the lossy queue in Figure 3. The enqueuing action is new; it is enabled iff $i \neq Last(\langle o\rangle \circ q)$ where $Last(\langle o\rangle \circ q)$ denotes the last element of $\langle o\rangle \circ q$. Hence, the queue cannot enqueue the same input twice in a row. Furthermore, when the lossy queue is empty and the input is equal to the output, an enqueue cannot take place.

Despite these changes, the resulting specification is logically equivalent to the original one, so the two specifications implement one another. We discuss how to prove this equivalence in section 4.

3.2 An Interleaving Style with Synchronous Communication

Specifications in the standard interleaving style describe components that communicate asynchronously; all synchronization relies on handshakes. In this section we describe another style for writing interleaving specifications which models a form of synchronous communication.

In this style, a specification still does not allow behaviors where an input variable and an output variable of a component change value simultaneously; the visible steps of component and environment are interleaved. However, a specification may allow behaviors where an input variable and an internal variable change value simultaneously.

This possibility is useful in modelling a sort of synchronous communication, where the component can record in its internal variable every step of the environment, as it happens. This sort of communication is less laborious (but perhaps less realistic) than an explicit handshake between component and environment.

3.2.1 The Form of a Specification In this style, a specification has the form

$$\exists x.(Init \wedge \Box[N]_{e,m,x} \wedge L)$$

where:

$Init$ is the usual state predicate.

N is no longer expected to imply $e' = e$. Instead, N should be expressible as the disjunction of an action N_{mod} that implies $e' = e$ and an action N_{env} that implies $m' = m$. Intuitively, the former describes the component steps, while the latter describes the environment steps and the resulting changes to x.

e, m, x appears as the subscript in $\Box[N]_{e,m,x}$ in order to guarantee that N accounts for every state change that modifies any of the variables of interest.

L is the conjunction of fairness conditions, each of the form $\mathrm{WF}_{e,m,x}(A)$ or $\mathrm{SF}_{e,m,x}(A)$. Now N_{mod} may be a disjunction, and each A may be one of the disjuncts of N_{mod}.

The specification disallows simultaneous changes of e and m. More precisely, if the transition formulas

$$N \Rightarrow (N_{mod} \vee N_{env}) \qquad N_{mod} \Rightarrow (e' = e) \qquad N_{env} \Rightarrow (m' = m)$$

are valid then the temporal formula

$$\exists x.(Init \wedge \Box[N]_{e,m,x} \wedge L) \Rightarrow \Box[e' = e \vee m' = m]_{e,m}$$

is valid. However, $Init \wedge \Box[N]_{e,m,x} \wedge L$ may allow simultaneous changes of e and x.

3.2.2 A Queue with Synchronous Input We can take advantage of the synchronous style for writing a simple specification of a (non-lossy) queue. This queue never misses an input because each change on its input wire i is simultaneously reflected in its contents q. The specification $\Phi_Q^\star$ of the queue is given in Figure 4.

In this specification, there is no enqueue action. Instead, we find an input action, Inp. This action asserts that i changes, that the new value of i is added to the queue, and that o does not change. Conversely, Deq implies that i does not change. The specification includes a fairness condition on Deq. It does not include a fairness condition on Inp, because we do not require that the environment produce infinitely many inputs. The queue implements the lossy queue, as the implication $\Phi_Q^\star \Rightarrow \Phi_Q$ is valid.

While the Inp action is part of the queue specification, it does not restrict how the input variable i may change. The specification allows arbitrary environment steps. In section 5 we discuss how to express assumptions about the environment.

$$
\begin{array}{lcl}
\mathit{Init}_Q & \triangleq & \land\ o = i \\
 & & \land\ q = \langle\rangle \\
\mathit{Inp} & \triangleq & \land\ i' \neq i \\
 & & \land\ q' = q \circ \langle i' \rangle \\
 & & \land\ o' = o \\
\mathit{Deq} & \triangleq & \land\ q \neq \langle\rangle \\
 & & \land\ o' = \mathit{Head}(q) \\
 & & \land\ q' = \mathit{Tail}(q) \\
 & & \land\ i' = i \\
N_Q^\star & \triangleq & \mathit{Inp} \lor \mathit{Deq} \\
L_Q^\star & \triangleq & \mathrm{WF}_{i,o,q}(\mathit{Deq}) \\
\Pi_Q^\star & \triangleq & \mathit{Init}_Q \land \Box[N_Q^\star]_{i,o,q} \land L_Q^\star \\
\Phi_Q^\star & \triangleq & \exists\!\!\exists q . \Pi_Q^\star
\end{array}
$$

Fig. 4. A synchronous specification of a queue.

It would be easy to restrict the *Inp* action. For example, we could add that the length of q is at most 8 as a conjunct to *Inp*, with the effect that an input step would be disallowed when the queue contains 8 elements. We would obtain a component specification that also imposes conditions on the component's environment. Such a specification is interestingly reminiscent of those that can be written in process calculi like CCS [24]; our experience with this kind of specification is limited.

3.3 A Noninterleaving Style

All of the specifications given so far are interleaving specifications, where component and environment do not take steps simultaneously. A noninterleaving specification, in contrast, allows such simultaneous steps.

In our experience, interleaving specifications are often easier to reason about than noninterleaving specifications. On the other hand, noninterleaving specifications lead to a direct treatment of composition; we give an example below.

3.3.1 The Form of a Specification One appealing way of writing a noninterleaving specification is "variable by variable", as:

$$\exists\!\!\exists x . (\mathit{Init} \land \Box[N_{inp}]_e \land \Box[N_{out}]_m \land \Box[N_{int}]_x \land L)$$

The actions N_{inp}, N_{out}, and N_{int} are intended to describe changes to one variable each (to e, m, and x, respectively). The choice of subscripts implies that a step that changes e, m, or x satisfies N_{inp}, N_{out}, or N_{int}, respectively.

The actions N_{inp}, N_{out}, and N_{int} may be mutually consistent. Therefore, a behavior may satisfy $\mathit{Init} \land \Box[N_{inp}]_e \land \Box[N_{out}]_m \land \Box[N_{int}]_x \land L$ even if it contains simultaneous changes to e, m, and x. Therefore, a behavior may satisfy $\exists x.(Init \wedge \Box[N_{inp}]_e \wedge \Box[N_{out}]_m \wedge \Box[N_{int}]_x \wedge L)$ even if it contains simultaneous changes to e and m.

Another style for noninterleaving specifications appears in [4].

3.3.2 A Noninterleaving Queue A noninterleaving specification of the queue appears in Figure 5. Its actions are defined in terms of auxiliary transition functions di and do that represent the values enqueued or dequeued in a transition, if any.

$$
\begin{array}{lcl}
Init_Q & \triangleq & \wedge\ o = i \\
 & & \wedge\ q = \langle\rangle \\
di & \triangleq & \textbf{if } i' = i \textbf{ then } \langle\rangle \textbf{ else } \langle i' \rangle \\
do & \triangleq & \textbf{if } o' = o \textbf{ then } \langle\rangle \textbf{ else } \langle o' \rangle \\
Inp^\bullet & \triangleq & \wedge\ i' \neq i \\
 & & \wedge\ q \circ di = do \circ q' \\
Deq^\bullet & \triangleq & \wedge\ q \neq \langle\rangle \\
 & & \wedge\ o' = Head(q) \\
 & & \wedge\ q \circ di = do \circ q' \\
L_Q^\bullet & \triangleq & \mathrm{WF}_o(Deq^\bullet) \\
\Pi_Q^\bullet & \triangleq & Init_Q \wedge \Box[Inp^\bullet]_i \wedge \Box[Deq^\bullet]_o \wedge \Box[Inp^\bullet \vee Deq^\bullet]_q \wedge L_Q^\bullet \\
\Phi_Q^\bullet & \triangleq & \exists q.\Pi_Q^\bullet
\end{array}
$$

Fig. 5. A noninterleaving specification of a queue.

The specification allows simultaneous input and output steps when the queue is not empty. The conjunct $q \circ di = do \circ q'$ appears in the input action to relate the old value of the queue with the new one, whether there has been an output or not. The conjunct $q \circ di = do \circ q'$ appears in the output action with an analogous purpose. This conjunct implies that any change to i is reflected in the queue, so that the queue never misses an input, and that any change to o comes from the queue.

The specification allows but does not require simultaneous input and output steps. Therefore, a behavior where input and output never coincide may satisfy the specification. In particular, the interleaving queue implements the noninterleaving queue: the implication $\Phi_Q^\star \Rightarrow \Phi_Q^\bullet$ is valid.

3.3.3 Composing Queues We may want to prove that two queues Q_1 and Q_2 in series implement a single queue Q. This proof may not be possible if Q has an interleaving specification, and we represent the composition of Q_1 and Q_2 by the conjunction of their specifications. In particular, Q_1 may receive an input at the same time as Q_2 produces an output, so the specification of the composite queue would allow simultaneous input and output steps, while an interleaving specification of Q would not. We may avoid this problem by simply assuming that input and output do not happen simultaneously (as in [4]). In contrast, this problem does not even arise for noninterleaving specifications, as we show next.

Consider two queues with noninterleaving specifications $\Phi^{\bullet}_{Q_1}$ and $\Phi^{\bullet}_{Q_2}$. Suppose that the first queue has input variable i, the second queue has output variable o, and the two queues are connected by the variable c. Thus, $\Phi^{\bullet}_{Q_1}$ is $\Phi^{\bullet}_{Q}[c/o]$ and $\Phi^{\bullet}_{Q_2}$ is $\Phi^{\bullet}_{Q}[c/i]$. The composition of the queues with c exposed is represented by the conjunction $\Phi^{\bullet}_{Q_1} \wedge \Phi^{\bullet}_{Q_2}$. It implements a single queue, as $(\Phi^{\bullet}_{Q_1} \wedge \Phi^{\bullet}_{Q_2}) \Rightarrow \Phi^{\bullet}_{Q}$ is valid. The composition of the queues with c hidden is represented by $\exists c.(\Phi^{\bullet}_{Q_1} \wedge \Phi^{\bullet}_{Q_2})$. Since c does not occur free in $\Phi^{\bullet}_{Q}$, it follows that $\exists c.(\Phi^{\bullet}_{Q_1} \wedge \Phi^{\bullet}_{Q_2}) \Rightarrow \Phi^{\bullet}_{Q}$ is valid as well.

4 Proof Rules

So far we have explained TLA formulas in terms of their semantics. This semantics is sufficient for writing specifications, and sometimes for informal reasoning, but the proof of TLA formulas requires precise rules of inference. In this section, we give some rules for TLA and discuss their use in verification.

4.1 Logical Rules

We describe some basic rules of temporal logic and rules for existential quantification. We adapt the rules from Lamport's work, with some attention to logical details. The rules that we present are quite incomplete; our purpose is only to illustrate an approach.

To understand the rules, it is useful to keep in mind an overall proof strategy that underlies the TLA approach (but is certainly not unique to it). The meaning of a temporal formula is given in terms of behaviors, and in principle one has to reason about sets of behaviors to prove a temporal formula. Unfortunately, reasoning about sets of behaviors can be hard. The rules of TLA have as objective reducing that hard, temporal reasoning to easier reasoning about actions. We do not give rules for reasoning about actions, since no new rules are needed—actions are first-order formulas.

Other proof systems for temporal logics (for example, that of Manna and Pnueli [22]) include rules for relating programs and logical formulas. TLA does not include such rules: in the TLA approach, formulas represent programs, and the rules used for verification are purely logical.

4.1.1 Invariants for $\Box$ Most verifications require proving an invariant, as a starting point. A basic proof rule for proving an invariant is:

$$\frac{P \wedge (N \vee v' = v) \;\Rightarrow\; P'}{P \wedge \Box[N]_v \;\Rightarrow\; \Box P} \qquad \text{(Inv-proof1)}$$

where P is a state predicate, N an action, and v a state function. This rule is sound, in the following sense:

– take any sets of symbols ($\mathcal{L}$, $\mathcal{V}_R$, $\mathcal{V}_F$), and take any class of structures for $\mathcal{L}$ (possibly a single structure);
– take any state predicate P, action N, and state function v;
– assume that $P \wedge (N \vee v' = v) \Rightarrow P'$ is valid, as a transition formula;
– then $P \wedge \Box[N]_v \Rightarrow \Box P$ is valid, as a temporal formula.

The proof of soundness is by a straightforward induction on behaviors.

Whenever we state a rule, from now on, we implicitly adopt the same reading. We assume given a set of symbols and a class of structures. A rule is sound when, if its antecedents are valid for the given class of structures, then so is its conclusion. Similarly, an axiom is sound if it is valid for the given class of structures. Clearly, not every useful axiom and rule is sound for every class of structures (consider $x+0=x$); however, all the axioms and rules listed here are valid for every class of structures.

Once an invariant is proved, it can be used. The following rule allows that:

$$\frac{P \wedge P' \wedge (N \vee v' = v) \;\Rightarrow\; (M \vee u' = u)}{\Box\, P \wedge \Box[N]_v \;\Rightarrow\; \Box[M]_u} \qquad \text{(Inv-use1)}$$

In particular, this rule yields:

$$\Box P \wedge \Box[N]_v \;\Rightarrow\; \Box[N \wedge P \wedge P']_v$$

Note that the converse of this implication does not always hold. The formula on the right-hand side is true of any behavior where v never changes, independently of the value of P.

The rules for invariants have straightforward generalizations that handle several transition formulas at once:

$$\frac{P \wedge (N_1 \vee v_1' = v_1) \wedge \ldots \wedge (N_k \vee v_k' = v_k) \;\Rightarrow\; P'}{P \wedge \Box[N_1]_{v_1} \wedge \ldots \wedge \Box[N_k]_{v_k} \;\Rightarrow\; \Box P} \qquad \text{(Inv-proof2)}$$

$$\frac{P \wedge P' \wedge (N_1 \vee v_1' = v_1) \wedge \ldots \wedge (N_k \vee v_k' = v_k) \Rightarrow (M \vee u' = u)}{\Box\, P \wedge \Box[N_1]_{v_1} \wedge \ldots \wedge \Box[N_k]_{v_k} \;\Rightarrow\; \Box[M]_u} \qquad \text{(Inv-use2)}$$

4.1.2 Lattices for $\Diamond$ The TLA method for proving fairness properties is based on a common lattice rule [26, 21]. The lattice rule relies on a well-founded relation $\succ$ (a binary relation such that there is no infinite chain $x_0 \succ x_1 \succ x_2 \succ \ldots$). In our formalization, we assume that $x \succ y$ is a constant predicate with free variables x and y, and $a \succ b$ stands for $(x \succ y)\{a/x\}\{b/y\}$. For now (following Lamport) we express that $\succ$ is well-founded by "$\succ$ is well-founded", as if this were a logical formula.

In logical form, the lattice rule reads:

$$\frac{\succ \text{ is well-founded}}{H \wedge \Box\forall x.(H \Rightarrow \Diamond(G \vee \exists y.(x \succ y) \wedge H[y/x])) \;\Rightarrow\; \Diamond G} \qquad \begin{array}{l}\text{(Lattice)}\\ x, y \in \mathcal{V}_R\\ x \notin \mathrm{FV}_{\mathrm{temp}}(G)\\ y \notin \mathrm{FV}_{\mathrm{temp}}(H)\end{array}$$

where x and y are rigid variables such that $x \notin \mathrm{FV}_{\mathrm{temp}}(G)$ and $y \notin \mathrm{FV}_{\mathrm{temp}}(H)$. For emphasis, let us write $H(x)$ for H, and $H(y)$ for $H[y/x]$. This rule gives a way of proving that G is eventually true, from the assumptions:

- $H(x)$ is true initially, for some x, and
- always, for every x, $H(x)$ implies that eventually G is true or $H(y)$ is true for some y such that $x \succ y$.

The assumption that $\succ$ is well-founded can be formalized in several ways. It can be written as a nontemporal formula, such as:

$$\forall f.\exists i.(i \in \mathsf{Nat} \wedge \neg(f(i) \succ f(i+1))$$

provided the underlying language is rich enough. Interestingly, it can also be written as a temporal formula:

$$\forall x.\neg(x \succ x) \wedge \boldsymbol{\forall} u.(\Box[u \succ u']_u \Rightarrow \Diamond \Box [\mathsf{false}]_u)$$

The first conjunct of this formula expresses antireflexivity. For each particular behavior, the second conjunct says only that if always $u \succ u'$ or $u = u'$ then after some point $u = u'$. If this temporal formula is valid for a structure, then $\succ$ is well-founded for that structure. This temporal formula may be a fine way of formalizing the assumption that a relation $\succ$ is well-founded. On the other hand, it may be hard to prove this formula from more elementary ones, should that be desirable.

In addition to the lattice rule, TLA includes specialized rules for reasoning about fairness; according to Lamport [19], those are not necessary for completeness, but important in practice.

4.1.3 Quantification The rules for $\exists$ are familiar from first-order logic:

$$\frac{G \Rightarrow F}{(\exists x.G) \Rightarrow F} \qquad \begin{array}{l} (\exists\text{-left}) \\ x \in \mathcal{V}_R \\ x \notin \mathrm{FV}_{\mathrm{temp}}(F) \end{array}$$

$$\frac{G \Rightarrow F[b/x]}{G \Rightarrow (\exists x.F)} \qquad \begin{array}{l} (\exists\text{-right}) \\ x \in \mathcal{V}_R \\ b \text{ a constant function} \end{array}$$

The rules for $\boldsymbol{\exists}$ are similar:

$$\frac{G \Rightarrow F}{(\boldsymbol{\exists} x.G) \Rightarrow F} \qquad \begin{array}{l} (\boldsymbol{\exists}\text{-left}) \\ x \in \mathcal{V}_F \\ x \notin \mathrm{FV}_{\mathrm{temp}}(F) \end{array}$$

$$\frac{G \Rightarrow F[b/x]}{G \Rightarrow (\boldsymbol{\exists} x.F)} \qquad \begin{array}{l} (\boldsymbol{\exists}\text{-right}) \\ x \in \mathcal{V}_F \\ b \text{ a state function} \end{array}$$

Since existential quantification over flexible variables corresponds to hiding, these rules play a central role in proofs of refinement for reactive systems. However, the rules are the standard, logical ones.

4.1.4 Auxiliary Variables Unfortunately, it is not always possible to prove a formula of the form $\exists x.F$ using ($\exists$-right). Additional principles are needed for completeness. Consider for example the trivial formula:

$$\exists x.\mathrm{WF}_x(\mathsf{true})$$

This formula states that if there exist at least two different values there is a sequence of values for x such that eventually $x' \neq x$. Although this formula is valid, we cannot exhibit any state function b such that $\mathrm{WF}_b(\mathsf{true})$ is valid. For any state function b, there are behaviors where all the flexible variables that occur in b remain constant, and hence b remains constant too. Therefore, $\exists x.\mathrm{WF}_x(\mathsf{true})$ cannot be derived directly using ($\exists$-right).

One approach to solving this problem consists in adding *auxiliary variables*. History variables are the most common auxiliary variables, but there are other sorts too [2]. In this approach, we reduce a proof that $G \Rightarrow (\exists x.F)$ to a proof that $G_{aux} \Rightarrow (\exists x.F)$, where G_{aux} is G plus an auxiliary variable. The proof that $G_{aux} \Rightarrow (\exists x.F)$ is performed with ($\exists$-right). The proof that G_{aux} is equivalent to G relies on special rules for auxiliary variables. For simple auxiliary variables, like history variables, this latter proof is often routine.

We discuss auxiliary variables again in section 4.2.3, in the context of an example. That example is both larger and conceptually clearer than $\exists x.\mathrm{WF}_x(\mathsf{true})$.

An alternative approach has been explored for other formalisms, but not for TLA (e.g., [23]). In that approach, the notion of state function is extended to that of state relation. The use of appropriate state relations removes the need for auxiliary variables. On the other hand, the rules for quantification become more complicated (and perhaps less logical).

4.2 Verification

The rules of TLA are designed to be used in proving that one reactive system implements another. Next we explain the overall strategy for their use, and then consider examples.

4.2.1 The Strategy As we have seen in section 3, there are several reasonable ways to represent systems by formulas; however, all the formulas that we have examined have the form:

$$\exists x_1 \ldots \exists x_n.(\mathit{Init} \wedge \Box[N_1]_{v_1} \wedge \ldots \wedge \Box[N_k]_{v_k} \wedge L)$$

Simplifying matters, first we consider only formulas of the modest form $\mathit{Init} \wedge \Box[N]_v$. If we represent two systems S_F and S_G by two such formulas,

$$\begin{array}{rcl} F & \triangleq & I \wedge \Box[M]_u \\ G & \triangleq & J \wedge \Box[N]_v \end{array}$$

then the validity of $G \Rightarrow F$ means that S_G implements S_F. The rules given above, together with propositional logic, yield the following method for proving $G \Rightarrow F$:

1. prove $J \Rightarrow I$;
2. pick a state predicate P, and prove $J \Rightarrow P$ and $P \wedge (N \vee v' = v) \Rightarrow P'$;
3. using (Inv-proof1) and (2), derive $G \Rightarrow \Box P$;
4. prove $P \wedge P' \wedge (N \vee v' = v) \Rightarrow (M \vee u' = u)$;
5. using (Inv-use1), (1), and (4), derive $\Box P \wedge G \Rightarrow F$;
6. obtain $G \Rightarrow F$ from (3) and (5).

To extend this method to deal with multiple transition formulas, it suffices to replace (Inv-proof1) with (Inv-proof2) and (Inv-use1) with (Inv-use2). Handling fairness properties requires use of (Lattice) and other rules; see [19] for details and examples.

The quantifier $\boldsymbol{\exists}$ can be treated with ($\boldsymbol{\exists}$-left) and ($\boldsymbol{\exists}$-right). When we have two specifications $F \triangleq \boldsymbol{\exists} x.F^I$ and $G \triangleq \boldsymbol{\exists} y.G^I$, with $y \notin \mathrm{FV}_{\mathrm{temp}}(F)$, we can prove that $G \Rightarrow F$ by finding a state function b such that $G^I \Rightarrow F^I[b/x]$, then applying ($\boldsymbol{\exists}$-right) and ($\boldsymbol{\exists}$-left) in sequence. The state function b is called an *abstraction function*, or a *refinement mapping* (e.g, [14, 2] and the references therein). A refinement mapping gives a value (an instantiation) for the variable x in terms of the other variables, including y.

Refinement mappings are an important tool for proving that one specification implements another one. However, they do not always suffice; in practice, verifications sometimes require auxiliary variables. Section 4.2.3, below, illustrates this.

4.2.2 A Simple Example As a simple example of a verification, we argue that the specification Φ_Q of the lossy queue of section 3.1.2 implements the alternative specification $\Phi_Q^\dagger$ of section 3.1.3. That is, we prove that $\Phi_Q \Rightarrow \Phi_Q^\dagger$ is valid.

The formulas Φ_Q and $\Phi_Q^\dagger$ are $\boldsymbol{\exists} q.\Pi_Q$ and $\boldsymbol{\exists} q.\Pi_Q^\dagger$, respectively, so they both have q as an internal variable. Informally, we call q the low-level queue or the high-level queue, depending on whether we refer to Φ_Q or $\Phi_Q^\dagger$. According to the specifications given above, the low-level queue may contain an element several times in sequence, while the high-level queue does not have any such repetitions.

Finding a refinement mapping means finding an expression for the high-level queue in terms of i, o, and the low-level queue. According to our rules, it suffices to check that

$$\Pi_Q \Rightarrow \Pi_Q^\dagger[\bar{q}/q]$$

for some state function $\bar{q}$. We choose:

$$\bar{q} \triangleq \mathit{Tail}(\natural(\langle o \rangle \circ q))$$

where $\natural(\langle o \rangle \circ q)$ is the sequence obtained by removing all repetitions from $\langle o \rangle \circ q$; in general, $\natural\rho$ is defined inductively by:

$$\begin{array}{rcll}
\natural(\langle\rangle) & = & \langle\rangle & \\
\natural(\langle a\rangle) & = & \langle a\rangle & \\
\natural(\langle a,b\rangle\circ\rho) & = & a\circ\natural(\langle b\rangle\circ\rho) & \text{if } a\neq b \\
\natural(\langle a,a\rangle\circ\rho) & = & a\circ\natural(\rho) &
\end{array}$$

Once this refinement mapping is given, it remains to prove:

$$(\mathit{Init}_Q \wedge \Box[N_Q]_{o,q} \wedge L_Q) \Rightarrow (\mathit{Init}_Q \wedge \Box[N_Q^\dagger]_{o,q} \wedge L_Q^\dagger)[\bar{q}/q]$$

The first step is checking an invariant: simply the type invariant that q is always a finite queue, which we write $\mathit{Queue}(q)$. By propositional logic, it suffices to prove:

$$\begin{array}{l}
\mathit{Queue}(q) \wedge \mathit{Enq} \Rightarrow \mathit{Queue}(q') \\
\mathit{Queue}(q) \wedge \mathit{Deq} \Rightarrow \mathit{Queue}(q') \\
\mathit{Queue}(q) \wedge o' = o \wedge q' = q \Rightarrow \mathit{Queue}(q')
\end{array}$$

and then (Inv-proof2) yields $\mathit{Queue}(q) \wedge \Pi_Q \Rightarrow \Box\mathit{Queue}(q)$; since $\mathit{Init}_Q \Rightarrow \mathit{Queue}(q)$, it also follows that $\Pi_Q \Rightarrow \Box\mathit{Queue}(q)$.

Now it remains to prove:

$$\begin{array}{l}
\mathit{Queue}(q) \wedge \mathit{Init}_Q \Rightarrow (\mathit{Init}_Q)[\bar{q}/q] \\
\Box\mathit{Queue}(q) \wedge \Box[N_Q]_{o,q} \Rightarrow (\Box[N_Q^\dagger]_{o,q})[\bar{q}/q] \\
\Box\mathit{Queue}(q) \wedge \Pi_Q \Rightarrow (L_Q^\dagger)[\bar{q}/q]
\end{array}$$

The first of these formulas is easy to prove directly in first-order logic. The second one can be proved by (Inv-use2). The third one is more difficult—we would not expect to handle it with (Lattice), but to use specialized rules for reasoning about fairness.

4.2.3 A Harder Example The proof of the converse implication, $\Phi_Q^\dagger \Rightarrow \Phi_Q$, is more difficult. The natural refinement mapping (defining $\bar{q}$ as q) does not work. To use this mapping, we would have to prove in particular

$$\Pi_Q^\dagger \Rightarrow \mathrm{WF}_{i,o,q}(\mathit{Enq})$$

This formula is not valid, because $\Pi_Q^\dagger$ is true for a behavior where i and o never change and the high-level queue remains empty, while $\mathrm{WF}_{i,o,q}(\mathit{Enq})$ is false for such a behavior.

In fact, the proof that $\Phi_Q^\dagger \Rightarrow \Phi_Q$ requires more than a refinement mapping. If i and o remain constant and q remains empty throughout a behavior, any state function $\bar{q}$ defined from i, o, and q remains constant too; $\Pi_Q^\dagger$ is true for this behavior but $\Pi_Q[\bar{q}/q]$ cannot be.

The proof can be performed by adding a dummy, auxiliary variable s to $\Phi_Q^\dagger$. This variable may be set to 1 when q is empty and i equals o. Then s is decremented to 0. The other variables (i, o, q) remain unchanged whenever s is set; to the observer of these variables, these transitions look like stutters.

A fairness condition guarantees that if i and o remain constant and equal, and q remains empty, then s is set infinitely often.

The specification with the auxiliary variable is $\Phi^{\dagger}_{QS}$, given in Figure 6. It is obtained by conjoining a formula Φ_S with the internal variable s to the formula $\Pi^{\dagger}_{Q}$ that describes q, and then hiding q as usual. In Φ_S, the formula $\Box[Pass]_{i,o,q}$ means that, whenever i, o, or q change, s equals 0 and s does not change; the formula $\Box[Set]_s$ means that, whenever s changes, q equals $\langle\rangle$, i equals o, and s' is **if** $(s = 0)$ **then** 1 **else** 0.

The specification Φ_S is written in a regular form, as an instance of a general template for auxiliary variables. Using general results, it is easy to derive that Φ_S is valid. Hence, $\Pi^{\dagger}_{Q}$ is equivalent to $\Pi^{\dagger}_{QS}$, and $\Phi^{\dagger}_{Q}$ to $\Phi^{\dagger}_{QS}$.

$$
\begin{array}{lcl}
Init_S & \triangleq & s = 0 \\
Pass & \triangleq & (s = 0) \land (s' = s) \\
Set & \triangleq & \land\ (q = \langle\rangle) \land (i = o) \\
 & & \land\ s' = \textbf{if } (s = 0) \textbf{ then } 1 \textbf{ else } 0 \\
\Pi_S & \triangleq & Init_S \land \Box[Pass]_{i,o,q} \land \Box[Set]_s \land \mathrm{WF}_s(Set) \\
\Phi_S & \triangleq & \exists s.\Pi_S \\
\Pi^{\dagger}_{QS} & \triangleq & \Pi^{\dagger}_{Q} \land \Phi_S \\
\Phi^{\dagger}_{QS} & \triangleq & \exists q.\Pi^{\dagger}_{QS}
\end{array}
$$

Fig. 6. A lossy queue with an auxiliary variable.

A refinement mapping can be given in terms of q and s:

$$\bar{q} \triangleq \textbf{if } s \neq 1 \textbf{ then } q \textbf{ else } \langle i \rangle$$

The auxiliary variable s has enabled us to "fake" an Enq immediately followed by a Deq, in a circumstance when q is empty and i and o are equal. These actions have no externally observable effect because i and o stay unchanged.

Using this refinement mapping, we can prove that

$$\Pi^{\dagger}_{Q} \land \Pi_S \;\Rightarrow\; \Pi_Q[\bar{q}/q]$$

From this it follows that

$$\Phi^{\dagger}_{QS} \;\Rightarrow\; \Phi_Q$$

and finally that

$$\Phi^{\dagger}_{Q} \;\Rightarrow\; \Phi_Q$$

as desired.

5 Assumption/Guarantee Specifications

Few components can be expected to work as intended in arbitrary environments. Usually, the specification of a component must describe the environment in which the component is supposed to operate. There are at least two different methods for this:

- A *closed system specification* constrains both the environment and the component. It may be written in the form $E \wedge M$, where E describes the environment and M the component. A behavior satisfies this specification iff it satisfies both E and M.
- An *open system specification* does not constrain the environment. Instead, it states an *assumption* about the environment and a *guarantee* by the component (e.g., [15, 25, 27, 3, 10]). It may be written in the form $E \Rightarrow M$, where E is the assumption and M the guarantee. A behavior that does not satisfy E does satisfy $E \Rightarrow M$. (We refine this form in section 5.2.)

The proof rules of section 4 are adequate for reasoning about closed system specifications. On the other hand, reasoning about assumptions and guarantees requires new rules. We give an account of compositional reasoning about assumption/guarantee specifications in a rather general algebraic setting. Returning to TLA, we obtain a proof rule that reduces reasoning about open system specifications to reasoning about closed system specifications.

5.1 Safety Properties

As a preliminary, we extend the semantics of temporal formulas to finite sequences of states:

- $[\![F]\!]_{\alpha,\langle t_0,\ldots,t_n\rangle}$ is true iff either $\langle t_0,\ldots,t_n\rangle$ is empty or there exists some (infinite) behavior θ that extends $\langle t_0,\ldots,t_n\rangle$ such that $[\![F]\!]_{\alpha,\theta}$ is true.

This definition implies that if F is true for a behavior then it is true for every prefix of the behavior.

It is customary to classify properties as *safety* or *liveness* properties [9]. A safety property is true for an infinite behavior σ iff it is true for all finite prefixes of σ. In particular, if P is a state predicate, A a transition formula, and v a state function, then $\Box P$ and $P \wedge \Box[A]_v$ are safety properties. A liveness property is true for every finite behavior. For example, $\Diamond P$, $\Diamond \Box P$, and $\Box \Diamond P$ are all liveness properties if P is a satisfiable state predicate. Manna and Pnueli give a more detailed classification of properties in their temporal logic [22].

Since TLA is invariant under stuttering (by Proposition 2.1), if a formula F is true of $\langle t_0,\ldots,t_n\rangle$ then it is also true of $\langle t_0,\ldots,t_n,t_n\rangle$. Hence, any safety property F is true of the finite state sequence $\langle t_0,\ldots,t_n\rangle$ iff it is true of the behavior $\langle t_0,\ldots,t_n,t_n,t_n,\ldots\rangle$. In turn, this implies that a conjunction of safety properties is again a safety property, and is true of $\langle t_0, \ldots, t_n \rangle$ iff every conjunct is true of $\langle t_0, \ldots, t_n \rangle$. Similarly, a finite disjunction of safety properties is again a safety property, and is true of $\langle t_0, \ldots, t_n \rangle$ iff some disjunct is true of $\langle t_0, \ldots, t_n \rangle$.

For any property G, there is a strongest safety property $\mathcal{C}(G)$, called the *closure* of G, such that $G \Rightarrow \mathcal{C}(G)$ is valid. The property $\mathcal{C}(G)$ has a syntactic definition in TLA, in terms of the TLA primitives (see section 5.3). However, in practice, we do not use that definition, but instead rely on the following semantics. Let σ be a behavior $s_0, s_1, \ldots$; then:

- $[\![\mathcal{C}(F)]\!]_{\alpha,\sigma}$ is true iff $[\![F]\!]_{\alpha,\langle s_0,\ldots,s_n\rangle}$ is true for every $n \geq 0$.

5.2 Connectives for Assumption/Guarantee Specifications

The evident way to write an open system specification with assumption E and guarantee M is as an implication, $E \Rightarrow M$, with both E and M in one of the forms discussed in section 3. Following [4], we prefer to use a stronger formula, $E \stackrel{+}{\rightarrow\!\!\triangleright} M$. Informally, $E \stackrel{+}{\rightarrow\!\!\triangleright} M$ is true for a behavior iff:

(a) if E holds, then M holds;
(b) if both E and M are violated, then M is violated later than E.

An intermediate formula, $E \rightarrow\!\!\triangleright M$, is sometimes useful too; $E \rightarrow\!\!\triangleright M$ is true for a behavior iff:

(a) if E holds, then M holds;
(b) if both E and M are violated, then M is violated no sooner than E.

If no step can violate both E and M, then $E \rightarrow\!\!\triangleright M$ and $E \stackrel{+}{\rightarrow\!\!\triangleright} M$ are equivalent. In all cases, $E \stackrel{+}{\rightarrow\!\!\triangleright} M$ implies $E \rightarrow\!\!\triangleright M$, and $E \rightarrow\!\!\triangleright M$ implies $E \Rightarrow M$. Although $E \rightarrow\!\!\triangleright M$ is strictly stronger than $E \Rightarrow M$, they are equivalid:

$$\frac{E \Rightarrow M}{E \rightarrow\!\!\triangleright M} \qquad \text{(while)}$$

An analogous rule is not sound for $\stackrel{+}{\rightarrow\!\!\triangleright}$: for example, $\mathsf{false} \Rightarrow \mathsf{false}$ is valid, but $\mathsf{false} \stackrel{+}{\rightarrow\!\!\triangleright} \mathsf{false}$ is not.

Like $\mathcal{C}$, both $\rightarrow\!\!\triangleright$ and $\stackrel{+}{\rightarrow\!\!\triangleright}$ have complicated syntactic definitions. The following semantic definitions are more transparent. Let σ be a behavior $s_0, s_1, \ldots$; then:

- $[\![F \rightarrow\!\!\triangleright G]\!]_{\alpha,\sigma}$ is true iff
  1. for all $n \geq 0$, if $[\![F]\!]_{\alpha,\langle s_0,\ldots,s_n\rangle}$ is true, then $[\![G]\!]_{\alpha,\langle s_0,\ldots,s_n\rangle}$ is true, and
  2. if $[\![F]\!]_{\alpha,\sigma}$ is true, then $[\![G]\!]_{\alpha,\sigma}$ is true.
- $[\![F \stackrel{+}{\rightarrow\!\!\triangleright} G]\!]_{\alpha,\sigma}$ is true iff
  1. for all $n \geq 0$, if $[\![F]\!]_{\alpha,\langle s_0,\ldots,s_{n-1}\rangle}$ is true then $[\![G]\!]_{\alpha,\langle s_0,\ldots,s_n\rangle}$ is true, and
  2. if $[\![F]\!]_{\alpha,\sigma}$ is true, then $[\![G]\!]_{\alpha,\sigma}$ is true.

In particular, these definitions imply that $F \rightarrow\!\!\triangleright G$ and $F \stackrel{+}{\rightarrow\!\!\triangleright} G$ are safety properties whenever G is a safety property.

5.3 Syntactic Definitions

We give the definition of $\mathcal{C}(G)$ in order to illustrate the expressiveness of TLA, and in order to clarify that TLA is not an open-ended logic with an ever growing set of operators. Let $x, y, z \in \mathcal{V}_F$ and $u \in \mathcal{V}_R$ be different variables, and suppose for simplicity that $\mathrm{FV}_{\mathrm{temp}}(G) \cap \mathcal{V}_F \subseteq \{x\}$. (The definition of $\mathcal{C}(G)$ when G has several free flexible variables is a straightforward generalization.) Then:

$$\mathcal{C}(G) \;\equiv\; \boldsymbol{\forall} z.\forall u.\left(\begin{array}{c} \Box[z' \neq u]_z \wedge \mathrm{WF}_z(z' \neq u) \\ \Rightarrow \\ \boldsymbol{\exists} y.\, \Box\,(u = z \Rightarrow x = y) \wedge G[y/x] \end{array}\right)$$

Roughly, the antecedent of the implication, $\Box[z' \neq u]_z \wedge \mathrm{WF}_z(z' \neq u)$, says that z may equal u for some time and then becomes different from u, say at time n. The consequent, $\boldsymbol{\exists} y.\, \Box\,(u = z \Rightarrow x = y) \wedge G[y/x]$, says that the sequence of values for x up to time $n-1$ can be extended to an infinite sequence of values for y that satisfies G. The quantifiers $\boldsymbol{\forall} z$ and $\forall u$ are in effect quantifying over all n.

The technique used in the definition of $\mathcal{C}$ yields a definition of $\mathrel{-\!\!\triangleright}$ for safety properties. Let $x, y, z \in \mathcal{V}_F$ and $u \in \mathcal{V}_R$ be different variables, and suppose for simplicity that $\mathrm{FV}_{\mathrm{temp}}(F) \cap \mathcal{V}_F \subseteq \{x\}$ and that $\mathrm{FV}_{\mathrm{temp}}(G) \cap \mathcal{V}_F \subseteq \{x\}$. Then:

$$\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G) \;\equiv\; \boldsymbol{\forall} z.\forall u.\left(\begin{array}{c} \Box[z' \neq u]_z \wedge \mathrm{WF}_z(z' \neq u) \\ \Rightarrow \\ \left(\begin{array}{c} \boldsymbol{\exists} y.\, \Box\,(u = z \Rightarrow x = y) \wedge F[y/x] \\ \Rightarrow \\ \boldsymbol{\exists} y.\, \Box\,(u = z \Rightarrow x = y) \wedge G[y/x] \end{array}\right) \end{array}\right)$$

We obtain a definition of $F \mathrel{-\!\!\triangleright} G$ for arbitrary F and G from the definition of $\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)$, through the equivalence:

$$F \mathrel{-\!\!\triangleright} G \;\equiv\; (\mathcal{C}(F) \mathrel{-\!\!\triangleright} \mathcal{C}(G)) \;\wedge\; (F \Rightarrow G)$$

Finally, we obtain a definition of $\overset{+}{\mathrel{-\!\!\triangleright}}$, through:

$$\mathcal{C}(F) \overset{+}{\mathrel{-\!\!\triangleright}} \mathcal{C}(G) \;\equiv\; ((\mathcal{C}(G) \mathrel{-\!\!\triangleright} \mathcal{C}(F)) \mathrel{-\!\!\triangleright} \mathcal{C}(G))$$

and

$$F \overset{+}{\mathrel{-\!\!\triangleright}} G \;\equiv\; (\mathcal{C}(F) \overset{+}{\mathrel{-\!\!\triangleright}} \mathcal{C}(G)) \;\wedge\; (F \Rightarrow G)$$

This definition of $\overset{+}{\mathrel{-\!\!\triangleright}}$ in terms of $\mathrel{-\!\!\triangleright}$ has an important counterpart in section 5.5 below.

5.4 Composing Assumption/Guarantee Specifications

If we have two systems with specifications $E_1 \stackrel{+}{\triangleright} M_1$ and $E_2 \stackrel{+}{\triangleright} M_2$, their composition is described by $(E_1 \stackrel{+}{\triangleright} M_1) \wedge (E_2 \stackrel{+}{\triangleright} M_2)$. To show that the composite system implements a specification $E \stackrel{+}{\triangleright} M$, we would have to prove:

$$(E_1 \stackrel{+}{\triangleright} M_1) \wedge (E_2 \stackrel{+}{\triangleright} M_2) \Rightarrow (E \stackrel{+}{\triangleright} M)$$

A natural but unsound rule for proving this formula is:

$$\frac{\begin{array}{l} E \wedge M_1 \wedge M_2 \Rightarrow E_1 \wedge E_2 \\ E \wedge M_1 \wedge M_2 \Rightarrow M \end{array}}{(E_1 \stackrel{+}{\triangleright} M_1) \wedge (E_2 \stackrel{+}{\triangleright} M_2) \Rightarrow (E \stackrel{+}{\triangleright} M)}$$

The first hypothesis serves to discharge the environment assumptions of the components. For example, when E is true, it expresses that the component guarantees M_1 and M_2 imply the component assumptions E_1 and E_2. The second hypothesis expresses that the final guarantee follows from the environment assumption E and the component guarantees M_1 and M_2.

The literature contains a number of sound approximations and variants of the unsound rule. We give one below. In the meantime, we illustrate the use of the unsound rule, because reasoning with the unsound rule is similar to reasoning with its sound variants, and in order to justify later restrictions to the unsound rule. We consider two components X and Y that communicate through the variables x and y as shown in Figure 7.

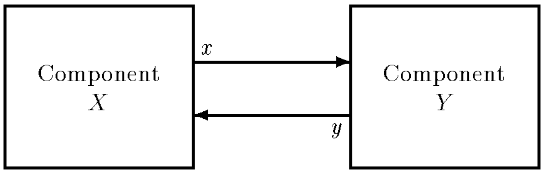

Fig. 7. A simple example of composition.

As a first example, we let:

$$M_X \triangleq \Box[x' > x]_x \quad \text{and} \quad M_Y \triangleq \Box[y' > y]_y$$

and specify X by $M_Y \stackrel{+}{\triangleright} M_X$, and Y by $M_X \stackrel{+}{\triangleright} M_Y$. That is, each component guarantees that the value of its output variable does not decrease, provided the value of its input variable does not decrease. According to the rule, we can derive $M_X \wedge M_Y$ from the conjunction of $M_X \stackrel{+}{\triangleright} M_Y$ and $M_Y \stackrel{+}{\triangleright} M_X$. That is, the composition of X and Y guarantees that the values of x and y do not decrease. This conclusion is correct.

As a second example, we let:

$$M_X \triangleq \Diamond \Box\, odd(x) \quad \text{and} \quad M_Y \triangleq \Diamond \Box\, odd(y)$$

and specify X by $M_Y \overset{+}{\rhd} M_X$, and Y by $M_X \overset{+}{\rhd} M_Y$. That is, each component guarantees that its output variable eventually becomes odd, provided its input variable eventually becomes odd. Again, the rule would allow us to deduce $M_X \wedge M_Y$. This conclusion is incorrect. To see this, consider the specifications Φ_X and Φ_Y that imply $M_Y \overset{+}{\rhd} M_X$ and $M_X \overset{+}{\rhd} M_Y$, respectively:

$$\begin{aligned} \Phi_X &\triangleq (x = 0) \wedge \Box[x' = y + 2]_x \wedge \mathrm{WF}_x(x' = y + 2) \\ \Phi_Y &\triangleq (y = 0) \wedge \Box[y' = x + 2]_y \wedge \mathrm{WF}_y(y' = x + 2) \end{aligned}$$

Both x and y remain even throughout any behavior that satisfies $\Phi_X \wedge \Phi_Y$, so $\Phi_X \wedge \Phi_Y$ does not imply $M_X \wedge M_Y$. As this example illustrates, reasoning about assumption/guarantee specifications is more problematic for liveness properties than for safety properties.

5.5 A Logic of Inductive Orderings

We now study the problem of composition formally, first in an algebraic framework, and later in the context of TLA. We take the algebraic detour because of its generality, and because it makes proofs more transparent. Continuing some of the work of [8], we obtain an explanation of composition in a general intuitionistic logic of specifications.

We assume given a set Σ, together with a well-founded pre-order $\sqsubseteq$ on Σ. We write $\sigma \sqsubset \tau$ if $\sigma \sqsubseteq \tau$ and $\sigma \neq \tau$. A set $S \subseteq \Sigma$ is *downward closed* if $\tau \in S$ whenever $\sigma \in S$ and $\tau \sqsubseteq \sigma$.

For $P, Q \subseteq \Sigma$, we define the sets $P \rightarrow Q \subseteq \Sigma$ and $P \overset{+}{\rightarrow} Q \subseteq \Sigma$ by:

$$\begin{aligned} &\sigma \in P \rightarrow Q \quad \text{iff} \quad \text{for all } \tau \sqsubseteq \sigma : \tau \in P \text{ implies } \tau \in Q \\ &\sigma \in P \overset{+}{\rightarrow} Q \quad \text{iff} \quad \text{for all } \tau \sqsubseteq \sigma : (\rho \in P \text{ for all } \rho \sqsubset \tau) \text{ implies } \tau \in Q \end{aligned}$$

By definition, $P \rightarrow Q$ and $P \overset{+}{\rightarrow} Q$ are downward closed.

Let $\mathcal{S}$ be a set of downward closed subsets of Σ such that $\mathcal{S}$ is closed under arbitrary intersections and unions, and under $\rightarrow$. For example, we may take $\mathcal{S}$ to be the set of all downward closed subsets of Σ.

Any such set $\mathcal{S}$ supports a natural style of reasoning where $\rightarrow$ acts as implication, and intersection and union as conjunction and disjunction. This follows from Proposition 5.1 below, which shows that $\mathcal{S}$ forms a complete Heyting algebra. A complete Heyting algebra $(A, \leq, \bigwedge, \bigvee, \rightarrow)$ is a complete distributive lattice A, with a partial order $\leq$, with a meet operation $\bigwedge$ and a join operation $\bigvee$, and an implication operation $\rightarrow$ such that $(P \wedge Q) \leq R$ iff $P \leq (Q \rightarrow R)$ [31]. In fact, $\rightarrow$ can be defined from $\bigwedge$ and $\bigvee$, by $Q \rightarrow R = \bigvee\{P \mid P \wedge Q \leq R\}$. The meet and join operations apply to arbitrary (possibly infinite) sets; we write $P \wedge Q$ for $\bigwedge\{P, Q\}$. Any complete Heyting algebra is a model of propositional intuitionistic logic [30, pp. 702–706].

Proposition 5.1. $(\mathcal{S}, \subseteq, \bigcap, \bigcup, \rightarrow)$ *is a complete Heyting algebra.*

The set $\mathcal{S}$ does not yield a Boolean algebra, because the complement of a downward closed set need not be downward closed. In other words, it is essential to use intuitionistic logic rather than classical logic to reason in $\mathcal{S}$.

Somewhat surprisingly, $\stackrel{+}{\rightarrow}$ can be defined from $\rightarrow$. The proof of the following proposition proceeds by induction, and thus relies on the assumption that $\sqsubseteq$ is well-founded.

Proposition 5.2. *For any* $P, Q \in \mathcal{S}$, $(P \stackrel{+}{\rightarrow} Q) = ((Q \rightarrow P) \rightarrow Q)$.

ASSUME: $P, Q \in \mathcal{S}$
PROVE: $P \stackrel{+}{\rightarrow} Q = (Q \rightarrow P) \rightarrow Q$
1. $P \stackrel{+}{\rightarrow} Q \subseteq (Q \rightarrow P) \rightarrow Q$
   PROOF: Using induction on $\sqsubseteq$, it suffices to:
   ASSUME: 1. $\sigma \in P \stackrel{+}{\rightarrow} Q$
   2. For all $\rho \sqsubset \sigma$, $\rho \in P \stackrel{+}{\rightarrow} Q$ implies $\rho \in (Q \rightarrow P) \rightarrow Q$.
   3. $\tau \sqsubseteq \sigma$
   4. $\tau \in Q \rightarrow P$
   PROVE: $\tau \in Q$
   1.1. For all $\rho \sqsubset \tau$, $\rho \in P$.
      1.1.1. For all $\rho \sqsubset \tau$, $\rho \in P \stackrel{+}{\rightarrow} Q$.
         PROOF: This follows from the assumptions that $\sigma \in P \stackrel{+}{\rightarrow} Q$ and $\tau \sqsubseteq \sigma$, since $P \stackrel{+}{\rightarrow} Q$ is downward closed.
      1.1.2. For all $\rho \sqsubset \tau$, $\rho \in (Q \rightarrow P) \rightarrow Q$.
         PROOF: From Step 1.1.1 by the induction hypothesis (Assumption 2), the assumption that $\tau \sqsubseteq \sigma$, and the transitivity of $\sqsubseteq$.
      1.1.3. For all $\rho \sqsubset \tau$, $\rho \in Q \rightarrow P$.
         PROOF: From the assumption that $\tau \in Q \rightarrow P$, since $Q \rightarrow P$ is downward closed.
      1.1.4. For all $\rho \sqsubset \tau$, $\rho \in Q$.
         PROOF: From Steps 1.1.2 and 1.1.3, by the definition of $\rightarrow$.
      1.1.5. Q.E.D.
         PROOF: Steps 1.1.3 and 1.1.4, by the definition of $\rightarrow$.
   1.2. $\tau \in P \stackrel{+}{\rightarrow} Q$
      PROOF: From the assumptions that $\sigma \in P \stackrel{+}{\rightarrow} Q$ and $\tau \sqsubseteq \sigma$, since $P \stackrel{+}{\rightarrow} Q$ is downward closed.
   1.3. Q.E.D.
      PROOF: From Steps 1.1 and 1.2, by the definition of $\stackrel{+}{\rightarrow}$.
2. $(Q \rightarrow P) \rightarrow Q \subseteq P \stackrel{+}{\rightarrow} Q$
   PROOF: By the definition of $\stackrel{+}{\rightarrow}$, it suffices to:
   ASSUME: 1. $\sigma \in (Q \rightarrow P) \rightarrow Q$
   2. $\tau \sqsubseteq \sigma$
   3. For all $\rho \sqsubset \tau$, $\rho \in P$.
   PROVE: $\tau \in Q$

2.1. If $\tau \notin Q$, then $\tau \in Q \rightarrow P$.
PROOF: By the definition of $\rightarrow$, it suffices to:
ASSUME: 1. $\tau \notin Q$
2. $\rho \sqsubseteq \tau$
3. $\rho \in Q$
PROVE: $\rho \in P$
2.1.1. $\rho \sqsubset \tau$
PROOF: The assumptions that $\rho \in Q$ and $\tau \notin Q$ imply $\rho \neq \tau$. The assertion follows from the assumption that $\rho \sqsubseteq \tau$.
2.1.2. Q.E.D.
PROOF: Step 2.1.1, and the assumption that $\rho \in P$ for all $\rho \sqsubset \tau$.
2.2. $\tau \in (Q \rightarrow P) \rightarrow Q$
PROOF: From the assumptions that $\sigma \in (Q \rightarrow P) \rightarrow Q$ and $\tau \sqsubseteq \sigma$, since $(Q \rightarrow P) \rightarrow Q$ is downward closed.
2.3. If $\tau \in Q \rightarrow P$, then $\tau \in Q$.
PROOF: From Step 2.2 by the definition of $\rightarrow$, since $\sqsubseteq$ is reflexive.
2.4. Q.E.D.
PROOF: By Steps 2.1, 2.3, and propositional reasoning.
3. Q.E.D.
PROOF: Steps 1 and 2. □

Propositions 5.1 and 5.2 open the door to purely syntactic reasoning about $\rightarrow$ and $\stackrel{+}{\rightarrow}$. In this syntactic reasoning, we write $\bigwedge$ for $\bigcap$; we also write $P \vdash Q$ for $P \subseteq Q$, and simply Q for $\Sigma \subseteq Q$.

Simple intuitionistic reasoning yields:

Proposition 5.3. *Let I be some index set, and assume that $P, P_i, Q, Q_i, R \in \mathcal{S}$ for $i \in I$. Then:*

$$\begin{aligned} Q &\vdash P \stackrel{+}{\rightarrow} Q && (1)\\ P \stackrel{+}{\rightarrow} Q &\vdash P \rightarrow Q && (2)\\ P \stackrel{+}{\rightarrow} P &\vdash P && (3)\\ (P \rightarrow Q) \wedge (Q \stackrel{+}{\rightarrow} R) &\vdash P \stackrel{+}{\rightarrow} R && (4)\\ (P \stackrel{+}{\rightarrow} (Q \rightarrow R)) \wedge (P \stackrel{+}{\rightarrow} Q) &\vdash P \stackrel{+}{\rightarrow} R && (5)\\ \bigwedge_{i \in I} (P_i \stackrel{+}{\rightarrow} Q_i) &\vdash \bigwedge_{i \in I} P_i \stackrel{+}{\rightarrow} \bigwedge_{i \in I} Q_i && (6) \end{aligned}$$

Theorem 5.1. *For $P, Q, R \in \mathcal{S}$:*

$$(P \wedge Q) \rightarrow R \vdash (R \stackrel{+}{\rightarrow} Q) \rightarrow (P \stackrel{+}{\rightarrow} Q)$$

PROOF: By Proposition 5.1 and simple intuitionistic logic, it suffices to:
ASSUME: 1. $P \wedge Q \rightarrow R$
2. $R \stackrel{+}{\rightarrow} Q$
PROVE: $P \stackrel{+}{\rightarrow} Q$

1. $P \rightarrow (Q \xrightarrow{+} Q)$
   PROOF: It suffices to:
   ASSUME: P
   PROVE: $Q \xrightarrow{+} Q$
   1.1. $Q \rightarrow R$
      PROOF: From the assumptions that $P \wedge Q \rightarrow R$ and P, by simple intuitionistic logic.
   1.2. Q.E.D.
      PROOF: Step 1.1, the assumption that $R \xrightarrow{+} Q$, and Proposition 5.3(4).
2. $P \rightarrow Q$
   PROOF: Step 1, Proposition 5.3(3), and simple intuitionistic logic.
3. $P \rightarrow R$
   PROOF: Step 2, the assumption that $P \wedge Q \rightarrow R$, and simple intuitionistic logic.
4. Q.E.D.
   PROOF: From Step 3 and the assumption that $R \xrightarrow{+} Q$, by Proposition 5.3(4). □

As Corollary 5.1, we obtain a rule for composition in $\mathcal{S}$. The composition rule of this corollary is similar in shape to the unsound rule considered in section 5.4.

Corollary 5.1. *Let I be some index set, and assume that $P, Q, P_i, Q_i \in \mathcal{S}$ for $i \in I$. Then:*

$$\frac{\begin{array}{c} P \wedge \bigwedge_{i \in I} Q_i \rightarrow \bigwedge_{i \in I} P_i \\ P \xrightarrow{+} (\bigwedge_{i \in I} Q_i \rightarrow Q) \end{array}}{\bigwedge_{i \in I}(P_i \xrightarrow{+} Q_i) \rightarrow (P \xrightarrow{+} Q)}$$

PROOF: Theorem 5.1, with the substitutions $R \triangleq \bigwedge_{i \in I} P_i$ and $Q \triangleq \bigwedge_{i \in I} Q_i$ yields

$$(\bigwedge_{i \in I} P_i \xrightarrow{+} \bigwedge_{i \in I} Q_i) \rightarrow (P \xrightarrow{+} \bigwedge_{i \in I} Q_i)$$

The assertion follows by Proposition 5.3(5) and (6), and simple intuitionistic reasoning. □

5.6 Compositional Reasoning in TLA

We apply the algebraic framework of section 5.5 to infer a composition rule for TLA. More precisely, we show that Corollary 5.1 yields a composition rule for safety properties, and then extend this rule to arbitrary properties.

Let Σ denote the set of finite sequences of states over some fixed structure $\mathcal{M}$. The prefix ordering, which we write $\sqsubseteq$, is a well-founded pre-order on Σ. Let $\mathcal{S}$ be the set of sets of finite sequences of states closed under $\sqsubseteq$ and under stuttering equivalence. The results of section 5.5 immediately apply to $\mathcal{S}$. In particular, Corollary 5.1 is a semantic composition rule for $\mathcal{S}$.

To obtain a syntactic composition rule, we reinterpret the results of section 5.5 in terms of temporal formulas. Logical conjunction and disjunction correspond to finite intersection and union over $\mathcal{S}$. Similarly, for safety properties, the logical operators $-\!\!\rhd$ and $\overset{+}{-\!\!\rhd}$ of section 5.2 correspond to the semantic operators $\rightarrow$ and $\overset{+}{\rightarrow}$, respectively. Thus, we obtain:

Theorem 5.2. *Let I be some finite index set, and assume that E, E_i, M, M_i are safety properties. Then:*

$$\frac{\begin{array}{c} E \wedge \bigwedge_{i \in I} M_i \Rightarrow \bigwedge_{i \in I} E_i \\ E \overset{+}{-\!\!\rhd} (\bigwedge_{i \in I} M_i -\!\!\rhd M) \end{array}}{\bigwedge_{i \in I}(E_i \overset{+}{-\!\!\rhd} M_i) \Rightarrow (E \overset{+}{-\!\!\rhd} M)}$$

Here the index set I is required to be finite simply because TLA offers only finite conjunctions and disjunctions. An infinitary form of Theorem 5.2 would be sound.

For safety properties, this rule is a sound variant of the rule discussed in section 5.4. It suffices to treat the first example of section 5.4, since there M_X and M_Y are safety properties. For that example, we take $I = \{X, Y\}$, $E = \mathsf{true}$, $E_X = M_Y$, $E_Y = M_X$, and $M = M_X \wedge M_Y$, and observe that $\mathsf{true} \overset{+}{-\!\!\rhd} F$ is equivalent to F.

Looking beyond safety properties, we recall that the following equivalence is valid for arbitrary temporal formulas F and G:

$$F \overset{+}{-\!\!\rhd} G \quad \equiv \quad (\mathcal{C}(F) \overset{+}{-\!\!\rhd} \mathcal{C}(G)) \wedge (F \Rightarrow G) \tag{7}$$

We obtain the following composition rule, which is essentially the same as that of [4]:

Theorem 5.3. *Let I be some finite index set, and assume that E, E_i, M, M_i are temporal formulas. Then:*

$$\frac{\begin{array}{l} \mathcal{C}(E) \wedge \bigwedge_{i \in I} \mathcal{C}(M_i) \Rightarrow \bigwedge_{i \in I} E_i \\ \mathcal{C}(E) \overset{+}{-\!\!\rhd} (\bigwedge_{i \in I} \mathcal{C}(M_i) -\!\!\rhd \mathcal{C}(M)) \\ E \wedge \bigwedge_{i \in I} M_i \Rightarrow M \end{array}}{\bigwedge_{i \in I}(E_i \overset{+}{-\!\!\rhd} M_i) \Rightarrow (E \overset{+}{-\!\!\rhd} M)}$$

PROOF:

1. $\bigwedge_{i \in I}(\mathcal{C}(E_i) \overset{+}{-\!\!\rhd} \mathcal{C}(M_i)) \Rightarrow (\mathcal{C}(E) \overset{+}{-\!\!\rhd} \mathcal{C}(M))$

   PROOF: The first hypothesis implies that

$$\mathcal{C}(E) \wedge \bigwedge_{i \in I} \mathcal{C}(M_i) \Rightarrow \bigwedge_{i \in I} \mathcal{C}(E_i)$$

   is valid, since $E_i \Rightarrow \mathcal{C}(E_i)$ is valid. The assertion follows by Theorem 5.2, since $\mathcal{C}(E)$, $\mathcal{C}(E_i)$, $\mathcal{C}(M_i)$, and $\mathcal{C}(M)$ are all safety properties.

2. $\bigwedge_{i\in I}(E_i \stackrel{+}{-\!\triangleright} M_i) \Rightarrow (E \Rightarrow M)$

PROOF: Theorem 5.2, with the substitution $M \triangleq \bigwedge_{i\in I} \mathcal{C}(M_i)$, yields

$$\bigwedge_{i\in I}(\mathcal{C}(E_i) \stackrel{+}{-\!\triangleright} \mathcal{C}(M_i)) \Rightarrow (\mathcal{C}(E) \stackrel{+}{-\!\triangleright} \bigwedge_{i\in I} \mathcal{C}(M_i))$$

from which we obtain

$$\bigwedge_{i\in I}(\mathcal{C}(E_i) \stackrel{+}{-\!\triangleright} \mathcal{C}(M_i)) \Rightarrow (\mathcal{C}(E) \Rightarrow \bigwedge_{i\in I} \mathcal{C}(M_i))$$

by (7). The first hypothesis implies

$$\bigwedge_{i\in I}(\mathcal{C}(E_i) \stackrel{+}{-\!\triangleright} \mathcal{C}(M_i)) \Rightarrow (\mathcal{C}(E) \Rightarrow \bigwedge_{i\in I} E_i)$$

Using (7) and the validity of $E \Rightarrow \mathcal{C}(E)$, we infer

$$\bigwedge_{i\in I}(E_i \stackrel{+}{-\!\triangleright} M_i) \Rightarrow (E \Rightarrow \bigwedge_{i\in I} E_i)$$

Using (7) again, we infer

$$\bigwedge_{i\in I}(E_i \stackrel{+}{-\!\triangleright} M_i) \Rightarrow (E \Rightarrow \bigwedge_{i\in I} M_i)$$

The assertion follows by the third hypothesis.

3. Q.E.D.

From Steps 1 and 2 by the equivalence (7). □

To apply this composition rule, the low-level assumptions E_i have to be deduced from a conjunction of safety properties. Hence, in practice, they commonly are safety properties themselves; this restriction to safety properties is consistent with the informal discussion and the examples of section 5.4. When E_i is not a safety property, we can use the equivalence (7) to apply the composition rule to the safety part of the specification, and use ordinary temporal reasoning for the liveness part.

The first and third hypotheses of the composition rule can be established using standard TLA proof rules, such as those of section 4, and some auxiliary rules to deal with closures. A strategy for reducing the proof of the second hypothesis to the proof of ordinary implications is discussed in [4]. An alternative strategy consists in using the following rule:

$$\frac{\begin{array}{c} F \wedge G \Rightarrow \Box P \\ \Box[P]_v \wedge G \Rightarrow H \end{array}}{F \stackrel{+}{-\!\triangleright} (G -\!\triangleright H)}$$

where P is a state predicate, v a state function, and F, G, and H arbitrary temporal formulas. This rule enables us to derive a formula with $-\!\triangleright$ and $\stackrel{+}{-\!\triangleright}$ from two classical implications. The state predicate P plays the role of an invariant, guaranteed by F and G according to the first hypothesis. The second hypothesis says, roughly, that H follows from G together with the invariant.

Thus, the techniques for verifying the hypotheses of the composition rule are rather specific to TLA. On the other hand, the composition rule is general, and follows fairly directly from the algebraic arguments of section 5.5.

Acknowledgements

Leslie Lamport contributed to most of the ideas presented here, and encouraged this work.

References

1. Martín Abadi. An axiomatization of Lamport's Temporal Logic of Actions. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 57–69. Springer-Verlag, August 1990. Also appeared as SRC Research Report 65, revised in March 1993.
2. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
3. Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.
4. Martín Abadi and Leslie Lamport. Conjoining specifications. Research Report 118, Digital Equipment Corporation, Systems Research Center, 1993. To appear in *ACM Transactions on Programming Languages and Systems*.
5. Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.
6. Martín Abadi, Leslie Lamport, and Stephan Merz. Refining specifications. To appear.
7. Martín Abadi and Gordon Plotkin. A logical view of composition and refinement. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 323–332, January 1991.
8. Martín Abadi and Gordon Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, June 1993.
9. Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.
10. Pierre Collette. *Design of Compositional Proof Systems Based on Assumption-Commitment Specifications: Application to UNITY*. PhD thesis, Université Catholique de Louvain, June 1994.
11. Urban Engberg, Peter Gronning, and Leslie Lamport. Mechanical verification of concurrent systems with TLA. In *Computer-Aided Verification*, Lecture Notes in Computer Science, pages 44–55. Springer-Verlag, June 1992. Proceedings of the Fourth International Conference, CAV'92.
12. Limor Fix and Fred B. Schneider. Reasoning about programs by exploiting the environment. In *Annual International Colloquium on Automata, Languages and Programming*, 1994.
13. M.J.C. Gordon. *Introduction to HOL: A Theorem Proving Environment*. Cambridge University Press, 1993.
14. C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
15. Cliff B. Jones. Specification and design of (parallel) programs. In R. E. A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 321–332. IFIP, North-Holland, September 1983.

16. R.P. Kurshan and Leslie Lamport. Verification of a multiplier: 64 bits and beyond. In Costas Courcoubetis, editor, *Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 166–179. Springer-Verlag, June 1993. Proceedings of the Fifth International Conference, CAV'93.
17. Leslie Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Information Processing 83: Proceedings of the IFIP 9th World Congress*, pages 657–668. IFIP, North-Holland, September 1983.
18. Leslie Lamport. Hybrid systems in TLA$^+$. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 77–102. Springer-Verlag, 1993.
19. Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
20. Leslie Lamport and Stephan Merz. Specifying and verifying fault-tolerant systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 41–76. Springer-Verlag, September 1994.
21. Zohar Manna and Amir Pnueli. Verification of concurrent programs: A temporal proof system. In J.W. de Bakker and J. van Leeuwen, editors, *Foundations of Computer Science IV, Distributed Systems: Part 2*, pages 163–255. Mathematical Center Tracts 159, Center for Mathematics and Computer Science, Amsterdam, 1983.
22. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.
23. Michael Merritt. Completeness theorems for automata. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 544–560. Springer-Verlag, 1990.
24. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
25. Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1981.
26. Susan Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Transactions on Programming Languages and Systems*, 4(3):455–495, July 1982.
27. Amir Pnueli. In transition from global to modular temporal reasoning about programs. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, NATO ASI Series, pages 123–144. Springer-Verlag, October 1984.
28. Amir Pnueli. System specification and refinement in temporal logic. In R.K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 1992.
29. Y.S. Ramakrishna. On the satisfiability problem for Lamport's Propositional Temporal Logic of Actions and some of its extensions. *Fundamenta Informaticæ*, 1995. To appear. A preliminary version appears in the Proceedings of the ICTL'94 Workshop (editor H.J. Ohlbach), Technical Report MPI-I-94-230, Max-Planck-Institut für Informatik, Saarbrücken, pp. 12–21, June 1994.
30. A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*, volume 2. North Holland, 1988.
31. Steven Vickers. *Topology Via Logic*, volume 5 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.