Assumption/Guarantee Specifications in Linear-Time Temporal Logic* (Extended Abstract)**

Bengt Jonsson and Yih-Kuen Tsay

Department of Computer Systems, Uppsala University, Sweden

Abstract. Previous works on assumption/guarantee specifications typically reason about relevant properties at the semantic level or define a special-purpose logic. We feel it is beneficial to formulate such specifications in a more widely used formalism. Specifically, we adopt the linear-time temporal logic (LTL) of Manna and Pnueli. We find that, with past temporal operators, LTL admits a succinct *syntactic* formulation of assumption/guarantee specifications. This contrasts, in particular, with the work by Abadi and Lamport using TLA, where working at the syntactic level is more complicated. Our composition rules are derived entirely within LTL and can also handle internal variables. We had to overcome a number of technical problems in this pursuit, in particular, the problem of extracting the safety closure of a temporal formula. As a by-product, we identify general conditions under which the safety closure can be expressed in a succinct way that facilitates syntactic manipulation.

1 Introduction

An open system executes in parallel and interacts with its environment. Such a system is best described by an assumption/guarantee (A/G for brevity) specification, which gives the assumptions on the behavior of the environment and the properties guaranteed by the system if the environment obeys the assumptions. In this paper, we explore the use of linear-time temporal logic in writing and reasoning about A/G specifications.

The idea of representing concurrent systems (programs) and their specifications as formulae in temporal logic was first proposed by Pnueli [16]. Lamport later observed that if specifications are invariant under stuttering, then refinement between specifications can be represented by the usual implication in temporal logic [12]. A further advance towards the effective use of temporal logic for the specification of concurrent systems was to notice that hidden, or internal, variables can be represented by existential quantification over program variables [13, 17]. We intend to make a further contribution in the same vein by showing

* This work was supported in part by the Swedish Board for Industrial and Technical Development (NUTEK) as part of ESPRIT BRA project No. 6021 (REACT) and by the Swedish Research Council for Engineering Sciences (TFR).

** The full paper is available as [11]; direct requests to {bengt,tsay}@docs.uu.se.

that A/G specifications can be formulated entirely within linear-time temporal logic. Barringer and Kuiper [6] are, to our knowledge, the first to formulate A/G specifications in temporal logic; however, their formulation does not allow quantification over program variables.

Suppose we wish to specify that the behavior of a system satisfies a guarantee property M if the behavior of its environment satisfies an assumption property E. An obvious formulation would be the usual implication $E \Rightarrow M$. But, this formulation has a problem, for example, when we compose two systems — one satisfying $E \Rightarrow M$ and the other satisfying $M \Rightarrow E$. If E and M are safety properties (stating that "bad things" do not happen [5]), we naturally would expect that the composed system satisfies both E and M. However, we cannot come to this conclusion, since it is not possible to discharge any of the antecedents in the implications.

A solution to the preceding problem was originally proposed by Misra and Chandy [15]. The basic idea is that a system satisfies an A/G specification with assumption E and guarantee M if, in every state of an execution, the system guarantees M up to the current state provided that its environment has respected E up to the preceding state; the definition implies that M is satisfied in the first (initial) state. If both E and M are safety properties, then the circular reasoning encountered when composing two systems can be broken by induction on the length of executions. Misra and Chandy's formulation has been elaborated and extended in various contexts, e.g., [6, 9, 2, 4, 3, 8]. In particular, Abadi and Lamport [3] combine the formulation with the usual implication so that the guarantee part may contain liveness properties (stating that "good things" eventually happen [5]).[3]

We propose to formulate A/G specifications using the linear-time temporal logic (LTL for brevity) of Manna and Pnueli [14] that conform to the aforementioned interpretation. Previous works on A/G specifications, including [15, 10, 9, 2, 4, 3, 7, 8, 18], typically reason about relevant properties at the semantic level or define a special-purpose logic. In contrast, our definition of an A/G specification is syntactic and entirely within LTL, which is relatively well understood and widely applied. We derive inference rules for composing A/G specifications as the main results of the paper.

One important aspect of our formulation is being able to handle assumptions and guarantees with internal variables, which simply are existentially quantified variables in LTL. Internal variables can be very useful in shortening a specification, e.g., the specification of a queue. Abadi and Lamport's work [3] is most closely related to ours. Grønning et al. [9] have also considered internal variables, but in a semantic framework. In addition, their proof rules seem unnecessarily strong and are formulated indirectly in terms of simulations rather than directly in terms of computations. Recently, Collette [8] designed a UNITY-like logic for A/G specifications (with restricted forms of assumptions and guarantees) that can handle hiding.

[3] Abadi and Lamport's formulation of an A/G specification also allows liveness properties in the assumption part. However, their composition rule only works for safety assumptions.

The work of Abadi and Lamport [3] in using TLA [13] for A/G specifications has been one inspiration of our work. Although they have also used temporal logic as the basis, their approach is different from ours. Whereas we try to express and reason about A/G specifications *syntactically*, they work mostly at the *semantic* level (because, for A/G specifications, working at the syntactic level of TLA would be very complicated). In particular, they introduce an operator $\stackrel{+}{\triangleright}$, which is defined semantically (though a syntactic definition is possible), to formulate A/G specifications; inference rules are then derived by semantic arguments. This in effect is introducing a new operator to the logic.

There are two main technical problems that arise in using linear-time temporal logic for A/G specifications: The first problem is to be able to talk about the satisfaction of (the safety part of) a formula with respect to a finite prefix of an infinite sequence (temporal formulae in LTL are interpreted over *infinite* sequences of states). Barringer and Kuiper [6] have pioneered the use of past temporal operators in overcoming the problem, while considering particular forms of assumptions and guarantees. Their idea intuitively is to regard the end of the prefix as the present and look at the past. We observe that Barringer and Kuiper's idea can be applied to any safety formulae of the form $\Box H$, where the truth value of H depends only on the present and the past states. This brings us to the second problem, which is to extract the safety part, or safety closure, of the guarantee part (i.e., the strongest safety property implied by the guarantee) in the form of $\Box H$. Although this is always possible, the obtained formula may be too complicated to reason about. We give a result showing that, if the guarantee is specified by a formula in a canonical form that satisfies certain general conditions, then its safety closure can be expressed in the desired form and in a more succinct way, which greatly facilitates syntactic manipulation. We actually could avoid calculating the safety closure, since our formulation of A/G specifications is applicable as long as the guarantee is given in the form of $\exists x : \Box H \wedge L$ with H depending only on the present and the past states (no other conditions on H or L are required). This would be similar to the approach of considering the guarantee as a pair of a safety property and an arbitrary supplementary property, e.g., [6, 18].

We give a brief, informal review of LTL in Section 2, where a canonical form of formulae is defined for expressing the assumption and the guarantee of an A/G specification. In Section 3, we show how the safety closure of a canonical formula can be expressed in a succinct way. Section 4 describes our formulation of A/G specifications in LTL. A general composition rule is derived and compared with that of Abadi and Lamport. An example is elaborated in Section 5. Section 6 is the conclusion.

2 Preliminaries

This section gives a brief and informal introduction to linear-time temporal logic (LTL) as defined in [14]. The use of the logic in specifying closed systems

is explained, leading to the definition of canonical formulae. Canonical formulae are also suitable for expressing the assumption and the guarantee of an A/G specification, each of which, when standing alone, is a specification of some closed system.

2.1 LTL

LTL is a logic for expressing properties of infinite sequences of states, where each state is an assignment to a predefined universe of variables. An LTL formula is interpreted with respect to a position $i \geq 0$ in a sequence of states. We assume a set of constant, function, and predicate symbols with fixed interpretations. A *state formula* is a particular LTL formula built only from variables, constants, functions, and predicates using the usual first-order logic connectives. The interpretation of a state formula (such as "$y > 5 \wedge (\exists n : y = n^2)$") in position i is performed as usual using the particular interpretation of variables in state i (plus the fixed interpretations of constants, functions, and predicates). LTL also contains extra temporal operators; in this paper, we will use the following:

- $\bigcirc$ means "in the next state". The formula $\bigcirc\varphi$ is true in position i of a sequence σ (denoted $(\sigma, i) \models \bigcirc\varphi$) iff φ is true in position $i+1$ of σ (i.e., $(\sigma, i+1) \models \varphi$).
- $\Box$ means "always in the future (including the present)"; $(\sigma, i) \models \Box\varphi$ iff $\forall k \geq i : (\sigma, k) \models \varphi$.
- $\Diamond$ means "sometime in the future"; $(\sigma, i) \models \Diamond\varphi$ iff $\exists k \geq i : (\sigma, k) \models \varphi$.
- $\tilde{\ominus}$ means "in the previous state, if there is any"; $(\sigma, i) \models \tilde{\ominus}\varphi$ iff $(i > 0) \Rightarrow ((\sigma, i-1) \models \varphi)$.
- $\boxminus$ means "always in the past (including the present)"; $(\sigma, i) \models \boxminus\varphi$ iff $\forall k : 0 \leq k \leq i : (\sigma, k) \models \varphi$.
- $\blacklozenge$ means "sometime in the past"; $(\sigma, i) \models \blacklozenge\varphi$ iff $\exists k : 0 \leq k \leq i : (\sigma, k) \models \varphi$.
- For a variable u, the interpretation of u^- (the previous value of u) in position i is the same as the interpretation of variable u in position $i-1$; by convention, the interpretation of u^- in position 0 is the same as the interpretation of u in position 0.[4]
- *first* is an abbreviation for $\tilde{\ominus}$*false* which is true only in position 0.

We say that a sequence σ satisfies a formula φ (or φ is true for σ) if $(\sigma, 0) \models \varphi$, which will be abbreviated as $\sigma \models \varphi$. A formula φ is *valid*, denoted $\models \varphi$ (or simply φ when it is clear that validity is intended), if φ is satisfied by every sequence.

Quantification deserves special attention. Each variable is either *rigid* (having the same interpretation in all states of a sequence) or *flexible* (with no restrictions

[4] In contrast to Lamport and others who use "+"-superscribed (or primed) variables to denote their values in the next state, we use "−"-superscribed variables to denote their values in the previous state. The reason is that (for conformity) we wish to use only past operators in the safety part of a specification. The introduction of "−"-superscribed variables is convenient but not essential, since they can be encoded by the $\tilde{\ominus}$ operator.

on interpretation in different states). When specifying a system, flexible variables can represent program or control variables, whose value may change over time. For a formula $\exists u : \varphi$ we have $(\sigma, i) \models \exists u : \varphi$ iff there is another sequence σ', which differs from σ in at most the interpretation given in each state to u, such that $(\sigma', i) \models \varphi$;[5] note that the restrictions of rigid variables must be observed. Intuitively, this means that the truth of $\exists u : \varphi$ for a flexible variable u depends on the existence of an infinite sequence of u-values (one for each state), rather than just a single value, such that φ can be satisfied.

A formula without temporal operators but possibly with "$-$"-superscribed variables is called a *transition formula* (the definition is slightly different from that in [14]). A formula without any future operator $\bigcirc$, $\Box$, or $\Diamond$ is called a *past formula*; in particular, a transition formula is a past formula. A property is said to be a safety property if the following condition holds: for any sequence σ, if each prefix of σ is a prefix of some sequence that satisfies the property, then σ also satisfies the property. A property is said to be a liveness property if every finite sequence is a prefix of some sequence that satisfies the property. A *safety formula* is one that specifies a safety property and a *liveness formula* is one that specifies a liveness property. Of particular importance, formulae of the form $\Box H$ are safety formulae if the truth value of H depends only on the present and the past states, e.g., if H is a past formula. The form of a liveness formula is not important for our purposes. The *safety closure* of a given property is the strongest safety property implied by the given property, i.e., a safety property satisfied by exactly those sequences σ such that each prefix of σ is a prefix of some sequence that satisfies the given property. The safety closure of a property specified by an arbitrary temporal formula can also be specified by a temporal formula (see the next paragraph), making it meaningful to talk about the safety closure of a formula.

Notations We use $\Rightarrow$ (instead of $\rightarrow$) for implication. Square brackets "[" and "]" are sometimes used in place of parentheses "(" and ")" for readability. A formula φ is sometimes written as $\varphi(x)$ to indicate that the free variables of φ are among the tuple of variables x. We write $\varphi[t/u]$, where t is a state formula and u a variable, to denote the formula obtained from φ by simultaneously substituting t for every free occurrence of u, assuming no free variables in t become bound in φ; when no confusion may arise, we also use the same notation for changing the name of a bound variable or the value of a constant. $\varphi[t/u]$ is simply written as $\varphi[t]$ if the intended substitution is clear from context. The notation for substitutions generalizes to tuples in the obvious way. To avoid ambiguity (since there are different but equivalent formulae), the safety closure $\mathcal{C}(\varphi)$ of a formula $\varphi(x)$ is defined as $\Box[\exists \bar{x} : \boxdot(\bar{x} = x) \wedge \Diamond(\mathit{first} \wedge \varphi[\bar{x}/x])]$ (which

[5] We note that, in contrast to TLA advocated by Lamport, the semantics of existential quantification in LTL does not preserve invariance under stuttering, i.e., $\exists u : \varphi$ is not necessarily invariant under stuttering even if φ is. However, the composition rules derived in this paper will still be valid if we adopt the semantics of TLA for existential quantification.

is satisfied by exactly those sequences σ such that each prefix of σ is a prefix of some sequence that satisfies φ).

2.2 Canonical Formulae

Using LTL, we can specify the behavior of a closed system ([13, 17]). Each variable of the system is represented by a flexible variable in LTL. Each execution of the system is represented by an infinite sequence of states. The system can thus be specified by an LTL formula in the sense that the executions of the system are exactly those satisfying the formula.

In general, a closed system can be specified by the conjunction of a safety formula and a liveness formula. The safety formula can be put in the form of $\Box H$ with H being a past formula, particularly $\Box((\mathit{first} \Rightarrow \mathit{Init}) \wedge (\neg \mathit{first} \Rightarrow N))$, where *Init* is a state formula and N a transition formula. The exact form of the liveness formula is not important for our purposes. Existential quantifications may be introduced to hide internal variables. Thus, a complete specification can be put into the form

$$\exists x : \Box H \wedge L$$

where x is a tuple of flexible variables, H is a past formula (so that $\Box H$ is a safety formula), and L is a liveness formula. Formulae of this form are called *canonical formulae*; the condition that H is a past formula will be further relaxed.

As we will show in Section 3, it is desirable that the pair of $\Box H$ and L be "*machine-closed*" [1], i.e., $\mathcal{C}(\Box H \wedge L) \Leftrightarrow \Box H$; this condition can always be enforced as shown in the following paragraphs. One way of understanding machine-closedness is that L does not rule out safety properties that are allowed by $\Box H$. For example, "$\Box[\boxminus(x = 0) \vee \boxminus(x = 1)]$" (which equals "$\Box(x = 0) \vee \Box(x = 1)$") and "$\Box\Diamond(x = 1)$" is not machine-closed, since $\Box\Diamond(x = 1)$ rules out the possibility of $\Box(x = 0)$.

We have required H in a canonical formula "$\exists x : \Box H \wedge L$" to be a past formula so that the truth value of H depends only on the present and the past states and thus $\Box H$ is a safety formula. The requirement can be replaced by a more general but less practical one, since the fact that "the truth value of a formula φ depends only on the present and the past states" can be precisely expressed as "$\Box[\boxminus(x = y) \Rightarrow (\varphi[x] \Leftrightarrow \varphi[y])]$ is valid". A formula that meets the more general requirement is called a *historical* (or semantically past) formula.[6] Obviously, every past formula is historical. From the definition of a historical formula it follows that, if H is historical, then $\Box H$ specifies a safety property and thus is a safety formula. The defining formula for $\mathcal{C}(\varphi)$, for instance, is in this form.

Once H is generalized to be historical, the desirable condition of machine-closedness — $\mathcal{C}(\Box H \wedge L) \Leftrightarrow \Box H$, can always be enforced (at least in theory). In fact, any formula φ is equivalent to the formula $\mathcal{C}(\varphi) \wedge (\neg\mathcal{C}(\varphi) \vee \varphi)$, which is in the form of $\Box H_\varphi \wedge L_\varphi$ with H_φ being a historical formula and L_φ a liveness formula such that $\mathcal{C}(\Box H_\varphi \wedge L_\varphi) \Leftrightarrow \Box H_\varphi$. (This is analogous to the well-known

[6] The definition of historical formulae is due to Amir Pnueli.

result that every property is the conjunction of a safety property and a liveness property [5].) We may also apply the result to φ in the context of a specification of a closed system $\exists x : \varphi$, translating the specification into another canonical formula that satisfies the machine-closedness condition.

3 Calculating Safety Closures

We define the *stuttering-extensibility* of a safety formula and show that, under machine-closedness and stuttering-extensibility, the safety closure of a canonical formula can be expressed in a succinct way and in the desired form.

First, we invoke a result in [3]:

Lemma 1. *If* $\exists x : \Box H \wedge L$ *is a canonical formula such that* $\mathcal{C}(\Box H \wedge L) \Leftrightarrow \Box H$, *then* $\mathcal{C}(\exists x : \Box H \wedge L) \Leftrightarrow \mathcal{C}(\exists x : \Box H)$.

Note that $\exists x : \Box H$ may or may not be a safety formula; even if it is, the formula is not in the desired form of $\Box H'$ such that H' is past or historical.

Let $\sigma|_i$ denote the prefix of σ ($\equiv s_0, s_1, s_2, \ldots$) of length i, i.e., the finite sequence $s_0, s_1, \ldots, s_i$ (which contains $i+1$ states and i transitions). The *stuttering-extension* of $\sigma|_i$ is the infinite sequence obtained from $\sigma|_i$ by indefinitely repeating its last state s_i, i.e., the infinite sequence $s_0, s_1, \ldots, s_{i-1}, s_i, s_i, s_i, \ldots$. We say that a safety formula $\Box H$ (with H being a past or historical formula) is *stuttering-extensible* if the following condition holds: for every σ and every $i \geq 0$, if $\boxdot H$ is true at position i of σ, then $\Box H$ is true for the stuttering-extension of $\sigma|_i$. Equivalently, $\Box H$ is stuttering-extensible if $\Box[\boxdot H(w) \Rightarrow \exists \bar{w} : \boxdot(\bar{w} = w) \wedge \bigcirc\Box(\bar{w} = \bar{w}^-) \wedge \Diamond(\mathit{first} \wedge \Box H[\bar{w}/w])$ is valid. The intuition is that, if a system has satisfied a stuttering-extensible safety property so far, then it will continue to satisfy the safety property simply by doing nothing. We note that stuttering-extensibility is conjunctive in the sense that the conjunction of two stuttering-extensible formulae is also stuttering-extensible.

Lemma 2. *If* $\exists x : \Box H$ *is a canonical formula such that* $\Box H$ *is stuttering-extensible, then* $\mathcal{C}(\exists x : \Box H) \Leftrightarrow \Box(\exists x : \boxdot H)$.

A weaker condition than stuttering-extensibility is sufficient for the above lemma to hold. However, the weaker condition lacks the conjunctivity property enjoyed by stuttering-extensibility. From Lemmas 1 and 2, it follows:

Theorem 3. *If* $\exists x : \Box H \wedge L$ *is a canonical formula such that* $\mathcal{C}(\Box H \wedge L) \Leftrightarrow \Box H$ *and* $\Box H$ *is stuttering-extensible, then* $\mathcal{C}(\exists x : \Box H \wedge L) \Leftrightarrow \Box(\exists x : \boxdot H)$.

4 Assumption/Guarantee Specifications

In this section, we describe our formulation of A/G specifications in LTL and derive inference rules for composing such specifications. In light of the results in Section 3, we consider A/G specifications where the assumption and the guarantee parts are expressed in the canonical form with the additional conditions of machine-closedness and stuttering-extensibility:

1. The assumption E is expressed as $\Box(\exists x : \boxdot H_E)$. Given that $\Box H_E$ is stuttering-extensible, $\Box(\exists x : \boxdot H_E)$ equals the safety closure of $\exists x : \Box H_E$, which is a specification of the safety properties of a system with the tuple x of internal variables hidden.
2. The guarantee M is expressed as $\exists y : \Box H_M \wedge L_M$ where $\mathcal{C}(\Box H_M \wedge L_M) \Leftrightarrow \Box H_M$ and $\Box H_M$ is stuttering-extensible so that the safety closure of M is equivalent to $\Box(\exists y : \boxdot H_M)$.

In fact, the conditions of machine-closedness and stuttering-extensibility are not absolutely necessary. The derived composition rules will still be valid without the two conditions if we regard M as a pair of the safety formula $\Box(\exists y : \boxdot H_M)$ and some supplementary formula and interpret the definition of $\triangleright$ (Section 4.2) as a purely syntactical expansion.

4.1 Basic Formulation

Following Misra and Chandy [15], an A/G specification of a system should capture the following property: For every execution of the system, the guarantee M is satisfied initially and, for every $i \geq 1$, if the assumption E has been satisfied by the prefix of length $i-1$, then M is satisfied by the prefix of length i (recall that the length of an execution is the number of transitions, which is one less than the number of states in the execution). The satisfaction of a formula with respect to prefixes of an execution depends only the safety part of the formula (since each of the prefixes is finite and therefore, by definition, can be extended to an infinite execution satisfying the liveness part of the formula). Therefore, we first concentrate on the safety closure of E (which is E itself) and that of M; the liveness part of M will be taken into account in the next subsection. In other words, we consider the formulation of an A/G specification with assumption $\Box(\exists x : \boxdot H_E)$ and guarantee $\Box(\exists y : \boxdot H_M)$.

Since H_E and H_M (and hence $\exists x : \boxdot H_E$ and $\exists y : \boxdot H_M$) are historical, "$\Box(\exists x : \boxdot H_E)$ is satisfied by the prefix of length $i-1$ of σ" can be formally stated as "$(\sigma, i) \models \tilde{\ominus}\boxdot(\exists x : \boxdot H_E)$" or equivalently "$(\sigma, i) \models \tilde{\ominus}(\exists x : \boxdot H_E)$" (recall that the states of σ are numbered from 0) and "$\Box(\exists y : \boxdot H_M)$ is satisfied by the prefix of length i of σ" as "$(\sigma, i) \models (\exists y : \boxdot H_M)$". The property that we intended to capture can, therefore, be expressed as $\Box[\tilde{\ominus}(\exists x : \boxdot H_E) \Rightarrow (\exists y : \boxdot H_M)]$. This formula also implies that $(\exists y : \boxdot H_M)$, or simply $(\exists y : H_M)$, has to be satisfied initially, since $\tilde{\ominus}(\exists x : \boxdot H_E)$ always holds in the initial state.

In summary, we take $\Box[\tilde{\ominus}(\exists x : \boxdot H_E) \Rightarrow (\exists y : \boxdot H_M)]$ to be the A/G specification with assumption $\Box(\exists x : \boxdot H_E)$ and guarantee $\Box(\exists y : \boxdot H_M)$. The following lemma states that A/G specifications can be composed in a straightforward way:

Lemma 4. *Assuming that $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ are pairwise disjoint,*

$$\bigwedge_{i=1}^{n} \Box[\tilde{\ominus}(\exists x_i : \boxdot H_{E_i}) \Rightarrow (\exists y_i : \boxdot H_{M_i})]$$
$$\Rightarrow \Box[\tilde{\ominus}(\exists x_1 \ldots x_n : \boxdot \bigwedge_{i=1}^{n} H_{E_i}) \Rightarrow (\exists y_1 \ldots y_n : \boxdot \bigwedge_{i=1}^{n} H_{M_i})].$$

From Lemma 4 and a simple induction, we obtain

Theorem 5.

$$\Box[\tilde{\ominus}(\exists y_1 : \boxdot H_{M_1}) \Rightarrow (\exists y_2 : \boxdot H_{M_2})] \wedge \Box[\tilde{\ominus}(\exists y_2 : \boxdot H_{M_2}) \Rightarrow (\exists y_1 : \boxdot H_{M_1})] \Rightarrow \Box(\exists y_1 : \boxdot H_{M_1}) \wedge \Box(\exists y_2 : \boxdot H_{M_2}).$$

A special case of the theorem is the composition principle essentially formulated by Misra and Chandy:

$$\Box(\tilde{\ominus}\boxdot H_{M_1} \Rightarrow \boxdot H_{M_2}) \wedge \Box(\tilde{\ominus}\boxdot H_{M_2} \Rightarrow \boxdot H_{M_1}) \Rightarrow \Box H_{M_1} \wedge \Box H_{M_2}.$$

4.2 Adding Liveness

To allow liveness properties in the guarantee, we simply strengthen the specification by conjoining it with the usual implication between the assumption and the entire guarantee. As the complete formulation of an A/G specification with assumption $E \equiv \Box(\exists x : \boxdot H_E)$ and guarantee $M \equiv \exists y : \Box H_M \wedge L_M$, we define $E \triangleright M$ as follows:

$$E \triangleright M \stackrel{\Delta}{=} \Box[\tilde{\ominus}(\exists x : \boxdot H_E) \Rightarrow (\exists y : \boxdot H_M)] \wedge [E \Rightarrow M]$$

This is consistent with our taking "$\Box[\tilde{\ominus}(\exists x : \boxdot H_E) \Rightarrow (\exists y : \boxdot H_M)]$" to be the A/G specification with assumption $\Box(\exists x : \boxdot H_E)$ and guarantee $\Box(\exists y : \boxdot H_M)$ in the previous subsection, since if M is a safety formula, the implication $E \Rightarrow M$ is subsumed by $\Box[\tilde{\ominus}(\exists x : \boxdot H_E) \Rightarrow (\exists y : \boxdot H_M)]$. Therefore, $[\Box(\exists x : \boxdot H_E) \triangleright \Box(\exists y : \boxdot H_M)] \Leftrightarrow \Box[\tilde{\ominus}(\exists x : \boxdot H_E) \Rightarrow (\exists y : \boxdot H_M)]$.

The general composition rule is as follows (assuming that x, y, $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ are pairwise disjoint); when M is a safety formula, Premise 2(b) can be dropped.

Theorem 6.

$$\frac{\begin{array}{ll} 1. & \Box\Big[(\exists x : \boxdot H_E) \wedge (\exists y_1 \ldots y_n : \boxdot \bigwedge_{i=1}^{n} H_{M_i}) \Rightarrow (\exists x_1 \ldots x_n : \boxdot \bigwedge_{i=1}^{n} H_{E_i})\Big] \\ 2.\ (a) & \Box\Big[\tilde{\ominus}(\exists x : \boxdot H_E) \wedge (\exists y_1 \ldots y_n : \boxdot \bigwedge_{i=1}^{n} H_{M_i}) \Rightarrow (\exists y : \boxdot H_M)\Big] \\ (b) & E \wedge \bigwedge_{i=1}^{n} M_i \Rightarrow M \end{array}}{\bigwedge_{i=1}^{n} (E_i \triangleright M_i) \Rightarrow (E \triangleright M)}$$

We now show how one goes about establishing the premises of the composition rule. Premise 2(b) is in the usual form of refinement between two canonical formulae, which can be proven by standard "refinement mapping" techniques [13, 17]. Premises 1 and 2(a) can respectively be rewritten as follows:

$$1.\ \Box\Big[\boxdot H_E \wedge \boxdot \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow \exists x_1 \ldots x_n : \boxdot \bigwedge_{i=1}^{n} H_{E_i}\Big]$$

2. $\Box\Big[\tilde{\ominus}\boxminus H_E \wedge \boxminus \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow \exists y\colon \boxminus H_M\Big]$

Both formulae are in a weaker form than the usual refinement between two canonical formulae, because the existential quantifications occur inside the $\Box$ operator. Like in the usual case, we may find appropriate functions $f_1, \ldots, f_n, g$ that map from the free variables of $H_E, H_{M_1}, \ldots, H_{M_n}$ to the domains of $x_1, \ldots, x_n, y$ respectively, and prove the following:

1. $\Box\Big[\boxminus H_E \wedge \boxminus \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow \boxminus \bigwedge_{i=1}^{n} H_{E_i}[f_i/x_i]\Big]$
2. $\Box\Big[\tilde{\ominus}\boxminus H_E \wedge \boxminus \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow \boxminus H_M[g/y]\Big]$

The first premise can be proven by showing

$$\Box\Big[H_E \wedge \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow \bigwedge_{i=1}^{n} H_{E_i}[f_i/x_i]\Big].$$

The second premise needs more work. It can be shown that $\Box(\tilde{\ominus}\boxminus H_E \wedge \boxminus \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow \boxminus H_M[g/y])$ is implied by $\Box(\tilde{\ominus} I_E \wedge \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow H_M[g/y])$, where I_E is a state formula such that $\Box(\boxminus H_E \Rightarrow I_E)$. The validity of $\Box(\boxminus H_E \Rightarrow I_E)$ can be established by the usual proof of an invariant. Let I_E^- denote the transition formula obtained from I_E by superscribing every flexible variable with "$-$". It follows that the second premise can be proven by showing

$$\Box\Big[(\neg \mathit{first} \Rightarrow I_E^-) \wedge \bigwedge_{i=1}^{n} H_{M_i} \Rightarrow H_M[g/y]\Big].$$

It is sometimes possible, e.g., in the example of Section 5, to prove a stronger and simpler formula $\Box(\bigwedge_{i=1}^{n} H_{M_i} \Rightarrow H_M[g/y])$.

4.3 Comparison with Abadi and Lamport's Work

In [3], Abadi and Lamport further develop their earlier work on A/G specifications [2] using the framework of TLA [13]. Although they have also used temporal logic as the basis, their approach is different from ours. Whereas we express and reason about A/G specifications *syntactically*, they work mostly at the *semantic* level. Technically, due to the absence of past operators in TLA, they extend the semantic model for TLA formulae to include *finite* sequences so that the assertion "a finite prefix of an execution satisfies a formula" (which plays a central role in A/G specifications) can be stated directly, which otherwise would require extensive usage of auxiliary variables that result in long and complicated formulae.

Abadi and Lamport formulate an A/G specification by means of the operator $\stackrel{+}{\triangleright}$. They define the formula $E \stackrel{+}{\triangleright} M$ to be true of a behavior σ iff (a) $E \Rightarrow M$

is true of σ, (b) M holds initially, and (c) for every $n \geq 0$, if E holds for the prefix of σ of length n, then M holds for the prefix of σ of length $n+1$. Part (a) is also present in our formulation of $E \triangleright M$. Parts (b) and (c) presuppose a definition of when a formula is satisfied by a prefix of a behavior; the definition of satisfaction by Abadi and Lamport depends only on the safety closure of a formula (i.e., parts (b) and (c) can be equivalently formulated using the safety closures $\mathcal{C}(E)$ and $\mathcal{C}(M)$ of E and M). Consequently, their general composition theorem, shown below, makes extensive use of safety closures.

$$\frac{\begin{array}{ll} 1. & \mathcal{C}(E) \wedge \bigwedge_{i=1}^{n} \mathcal{C}(M_i) \Rightarrow \bigwedge_{i=1}^{n} E_i \\ 2. & (a)\ \mathcal{C}(E)_{+v} \wedge \bigwedge_{i=1}^{n} \mathcal{C}(M_i) \Rightarrow \mathcal{C}(M) \\ & (b)\ E \wedge \bigwedge_{i=1}^{n} M_i \Rightarrow M \end{array}}{\bigwedge_{i=1}^{n} (E_i \stackrel{+}{\triangleright} M_i) \Rightarrow (E \stackrel{+}{\triangleright} M)}$$

where $\mathcal{C}(E)_{+v}$, for some tuple v of variables, intuitively means that the values of v may change in a state transition only if the prefix up to the current state satisfies $\mathcal{C}(E)$.

Aside from using the novel (and not very easy to reason about) operator "$_{+v}$", the theorem explicitly uses safety closures, which are a semantic notion. In contrast, our composition theorem is presented purely syntactically within LTL. It can be shown [11] that, under some general assumptions, the definitions of $\stackrel{+}{\triangleright}$ and $\triangleright$ are equivalent and the composition theorem of Abadi and Lamport can be derived from Theorem 6.

5 An Example

To illustrate the application of our formulation of A/G specifications, we consider the same example as used by Abadi and Lamport [3] and others, e.g., [6, 7], where two queues for storing natural numbers are composed in series to obtain a larger one. A queue is an open system whose environment is the user of the queue. In the most straightforward A/G specification of a queue, the guarantee part contains an internal variable that is used to store a finite sequence of natural numbers. To make the example more interesting, we stipulate that numbers are sent by the environment in ascending order. We deliberately introduce an internal variable (which could have been avoided) in the assumption part that stores the next number to be sent by the environment to the queue and we specify that the next number be greater than the previous one sent. It should be noted that internal variables are often introduced for the ease of writing specifications and need not correspond to actual program variables in an implementation. We take the liberty to borrow notions from [3] for our exposition of the example. We shall concentrate only on safety properties; liveness properties can be handled in a similar way as in [3]. Moreover, we shall adopt a noninterleaving representation

of the queue, permitting the addition of a number to and the removal of another from the queue to occur simultaneously; an interleaving representation can also be treated in a similar way as in [3].

A queue interacts with its environment via two single-buffered channels: an input channel for receiving a number from the environment and an output channel for sending a number to the environment, as depicted in Figure 1. A channel c consists of two variables $c.sig$ and $c.ack$ of type boolean and a third variable (the single buffer) $c.val$ of type natural number. Boolean variables $c.sig$ and $c.ack$ are used for the synchronization between the sender and the receiver of channel c. The channel is ready for sending iff $c.sig = c.ack$. When $c.sig = c.ack$, the sender may deposit a number in $c.val$ and at the same time negate the value of $c.sig$ so that $c.sig \neq c.ack$, indicating that a value has been sent. Subsequently, the receiver may read the value of $c.val$ and negate $c.ack$, which makes $c.sig$ and $c.ack$ equal indicating that the channel is ready for sending again. This synchronization mechanism is usually referred to as the (two-phase) handshake protocol. We write c to denote $\langle c.sig, c.ack, c.val \rangle$ and $c.snd$ to denote $\langle c.sig, c.val \rangle$.

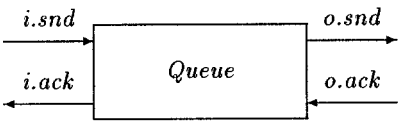

Fig. 1. A queue as an open system.

We introduce abbreviations for the initial condition and two relevant transition formulae on a channel:

$$
\begin{array}{lcl}
CInit(c) & \triangleq & \neg c.sig \wedge \neg c.ack \wedge c.val = 0 \\
Send(v, c) & \triangleq & c.sig^- = c.ack^- \wedge c.ack = c.ack^- \wedge c.snd = \langle \neg c.sig^-, v \rangle \\
Ack(c) & \triangleq & c.sig^- \neq c.ack^- \wedge c.ack = \neg c.ack^- \wedge c.snd = c.snd^-
\end{array}
$$

In accordance with the handshake protocol, the environment of a queue with input channel i and output channel o is expected to obey the following assumption:

$$
\begin{array}{lcl}
E(i, o, x) & \triangleq & \Box(\exists x : \boxdot H_E(i, o, x)) \\
H_E(i, o, x) & \triangleq & (first \Rightarrow Init_E) \wedge (\neg first \Rightarrow (Put \vee Get \vee GetPut \vee Skip_E))
\end{array}
$$

where the initial condition and relevant transition formulae are defined as follows.

$$
\begin{array}{lcl}
Init_E & \triangleq & CInit(i) \\
Put & \triangleq & Send(x^-, i) \wedge x > x^- \wedge o.ack = o.ack^- \\
Get & \triangleq & Ack(o) \wedge i.snd = i.snd^- \\
GetPut & \triangleq & Ack(o) \wedge Send(x^-, i) \wedge x > x^- \\
Skip_E & \triangleq & i.snd = i.snd^- \wedge o.ack = o.ack^- \wedge x = x^-
\end{array}
$$

The queue in return should guarantee M as defined below:

$$M(i,o,q,max) \triangleq \Box(\exists q : \boxdot H_M(i,o,q,max))$$
$$H_M(i,o,q,max) \triangleq (first \Rightarrow Init_M) \wedge (\neg first \Rightarrow (Enq \vee Deq \vee DeqEnq \vee Skip_M))$$

where the relevant formulae are defined as follows.

$$\begin{aligned}
Init_M &\triangleq CInit(o) \wedge q = \langle\rangle \\
Enq &\triangleq |q^-| < max \wedge Ack(i) \wedge (q = q^- \circ \langle i.val^- \rangle) \wedge o.snd = o.snd^- \\
Deq &\triangleq |q^-| > 0 \wedge Send(head(q^-), o) \wedge q = tail(q^-) \wedge i.ack = i.ack^- \\
DeqEnq &\triangleq |q^-| > 0 \wedge Send(head(q^-), o) \wedge Ack(i) \wedge (q = tail(q^-) \circ \langle i.val^- \rangle) \\
Skip_M &\triangleq o.snd = o.snd^- \wedge i.ack = i.ack^- \wedge q = q^-
\end{aligned}$$

Note that H_E and H_M are indeed past and hence historical formulae as required. Both $\Box H_E$ and $\Box H_M$ are apparently stuttering-extensible. From Theorem 3, $\Box(\exists x : \boxdot H_E(i,o,x))$ and $\Box(\exists q : \boxdot H_M(i,o,q,max))$ express the safety closures of $\exists x : \Box H_E(i,o,x)$ and $\exists q : \Box H_M(i,o,q,max)$, respectively.

The assertion that the composition of two queues in series results in a larger one can be expressed by the following formula.

$$\begin{aligned}
&(E[i,m,x_1] \triangleright M[i,m,q_1,N_1]) \wedge (E[m,o,x_2] \triangleright M[m,o,q_2,N_2]) \\
&\Rightarrow (E[i,o,x] \triangleright M[i,o,q,(N_1+N_2+1)])
\end{aligned}$$

Figure 2 shows the statement in picture.

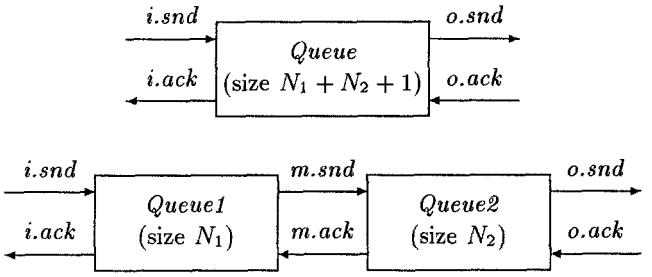

Fig. 2. Implementation of a queue by two composed in series.

We now apply Theorem 6 to show the validity of the above statement. Following the discussion in Section 4.2, the premises of the theorem translate into the following two formulae:

1. $\Box\Big[\boxdot H_E[i,o,x] \wedge \boxdot(H_M[i,m,q_1,N_1] \wedge H_M[m,o,q_2,N_2])$
$\Rightarrow \exists x_1 x_2 : \boxdot(H_E[i,m,x_1] \wedge H_E[m,o,x_2])\Big]$

2. $\Box\Big[\tilde{\ominus}\boxminus H_E[i,o,x] \wedge \boxminus(H_M[i,m,q_1,N_1] \wedge H_M[m,o,q_2,N_2])$
$\Rightarrow \exists q : \boxminus H_M[i,o,q,(N_1+N_2+1)]\Big]$

The required mappings can be defined as follows:

1. For variable x_1, the required mapping $\bar{x}_1$ is simply $\bar{x}_1(x) = x$. For variable x_2, we define the mapping $\bar{x}_2$ as follows:

$$\bar{x}_2(x,i,q_1) \triangleq \begin{array}{ll} head(q_1) & \text{if } q_1 \neq \langle\rangle \\ i.val & \text{if } q_1 = \langle\rangle \wedge i.sig \neq i.ack \\ x & \text{otherwise} \end{array}$$

Intuitively, for the second queue, the next number to be sent by its environment is either the value at the head of q_1 (if not empty), $i.val$ (if q_1 is empty and channel i not empty), or x.
2. The mapping from q_1, m, and q_2 to q is the same as in [3]:

$$\bar{q}(q_1,m,q_2) \triangleq \begin{array}{ll} q_2 \circ q_1 & \text{if } m.sig = m.ack \\ q_2 \circ \langle m.val\rangle \circ q_1 & \text{otherwise} \end{array}$$

Using these mappings, it is straightforward to prove

1. $\Box\Big[H_E[i,o,x] \wedge (H_M[i,m,q_1,N_1] \wedge H_M[m,o,q_2,N_2])$
$\Rightarrow H_E[i,m,\bar{x}_1] \wedge H_E[m,o,\bar{x}_2]\Big]$
2. $\Box\Big[H_M[i,m,q_1,N_1] \wedge H_M[m,o,q_2,N_2] \Rightarrow H_M[i,o,\bar{q},(N_1+N_2+1)]\Big]$

which implies the needed premises.

6 Conclusion

We have developed a formulation of A/G specifications within linear-time temporal logic. We have shown that, by using past temporal operators, it is possible to give a natural and general formulation of A/G specifications and to derive needed inference rules in a simple and straightforward way. We have also shown how internal variables can be handled. It is our intention that the results will further advance the use and applicability of temporal logic in specifying concurrent systems.

Acknowledgment

We thank Amir Pnueli for suggesting the notion of historical formulae. We also thank Martín Abadi, Pierre Collette, Leslie Lamport, and Qiwen Xu for comments and suggestions on earlier drafts of the paper.

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1991.
2. M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.
3. M. Abadi and L. Lamport. Conjoining specifications. Technical Report 118, SRC DEC, 1993. Part of the paper appears in PODC '94 as "Open Systems in TLA".
4. M. Abadi and G.D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, June 1993.
5. B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 24(4):181–185, October 1985.
6. H. Barringer and R. Kuiper. Hierarchical development of concurrent systems in a temporal logic framework. In *Seminar on Concurrency, LNCS 197*, pages 35–61. Springer-Verlag, 1984.
7. P. Collette. Application of the composition principle to Unity-like specifications. In *TAPSOFT '93, LNCS 668*, pages 230–242. Springer-Verlag, 1993.
8. P. Collette. *Design of Compositional Proof Systems Based on Assumption-Guarantee Specifications — Application to UNITY.* PhD thesis, Université Catholique de Louvain, June 1994.
9. P. Grønning, T.Q. Nielsen, and H.H. Løvengreen. Refinement and composition of transition-based rely-guarantee specifications with auxiliary variables. In *FST&TCS, LNCS 472*, pages 332–348. Springer-Verlag, 1991.
10. C.B. Jones. Tentative steps towards a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, October 1983.
11. B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. Technical Report DoCS 95/58, Department of Computer Systems, Uppsala University, January 1995.
12. L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Information Processing 83*, pages 657–668. IFIP, North-Holland, 1983.
13. L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
14. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.
15. J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, July 1981.
16. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1982.
17. A. Pnueli. System specification and refinement in temporal logic. In *FST&TCS, LNCS 652*, pages 1–38. Springer-Verlag, 1992.
18. Q. Xu, A. Cau, and P. Collette. On unifying assumption-commitment style proof rules for concurrency. In *CONCUR '94, LNCS 836*, pages 267–282. Springer-Verlag, 1994.