On the Completeness of Compositional Reasoning Methods

16

KEDAR S. NAMJOSHI
Bell Laboratories
and
RICHARD J. TREFLER
University of Waterloo

Hardware systems and reactive software systems can be described as the composition of several concurrently active processes. Automated reasoning based on model checking algorithms can substantially increase confidence in the overall reliability of a system. Direct methods for model checking a concurrent composition, however, usually suffer from the explosion in the number of program states that arises from concurrency. Reasoning compositionally about individual processes helps mitigate this problem. A number of rules have been proposed for compositional reasoning, typically based on an assume-guarantee reasoning paradigm. Reasoning with these rules can be delicate, as some are syntactically circular in nature, in that assumptions and guarantees are mutually dependent. This is known to be a source of unsoundness. In this article, we investigate rules for compositional reasoning from the viewpoint of *completeness*. We show that several rules are incomplete: that is, there are properties whose validity cannot be established using (only) these rules. We derive a new, circular, reasoning rule and show it to be sound and complete. We show that the auxiliary assertions needed for completeness need be defined only on the interface of the component processes. We also show that the two main paradigms of circular and noncircular reasoning are closely related, in that a proof of one type can be transformed in a straightforward manner to one of the other type. These results give some insight into the applicability of compositional reasoning methods.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computational logic, modal logic, temporal logic*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Assertions, logics of*

A preliminary version of this work appeared in *Proceedings of the Conference on Computer-Aided Verification* (2000).
K. Namjoshi is supported in part by NSF grant CCR-0341658. R. Trefler is supported in part by an Individual Discovery Grant from the Natural Sciences and Engineering Research Council of Canada.
Authors' addresses: K. S. Namjoshi, Bell Laboratories, Alcatel-Lucent, 600-700 Mountain Avenue, Murray Hill, NJ 07974; email: kedar@research.bell-labs.com; R. J. Trefler, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, N2L 3G1; email: trefler@cs.uwaterloo.ca.

DOI 10.1145/1740582.1740584 http://doi.acm.org/10.1145/1740582.1740584

programs, mechanical verification, specification techniques; B.6.3 [**Logic Design**]: Design Aids—*Verification*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Correctness proofs, formal methods, model checking*

General Terms: Verification, Algorithms, Theory

Additional Key Words and Phrases: Compositional reasoning, syntactically circular reasoning, assume-guarantee reasoning, automated reasoning, concurrent systems

ACM Reference Format:

Namjoshi, K. S. and Trefler, R. J. 2010. On the completeness of compositional reasoning methods. ACM Trans. Comput. Logic, 11, 3, Article 16 (May 2010), 22 pages.
DOI = 10.1145/1740582.1740584 http://doi.acm.org/10.1145/1740582.1740584

1. INTRODUCTION

In a landmark article, Pnueli [1977], advocated the use of temporal logic as a formalism for describing the correct operation of reactive systems. The subsequent development of *model checking* [Clarke and Emerson 1981; Queille and Sifakis 1982] (cf. Clarke et al. [1986] and Vardi and Wolper [1986]) created a fully automated technique for showing that a finite-state reactive system, M, satisfies a temporal logic formula, f, typically denoted as $M \models f$. For many useful specification logics, model checking has complexity linear in the size of the state transition graph of M. However, when M is given as the parallel composition of n finite-state processes, each of size bounded by K, its size may be exponential in n. This problem of "state explosion" is the main limitation to applying model checking in practice.

Compositional reasoning techniques, originally devised in the context of deductive proof methods [de Roever et al. 2001], form a promising approach to ameliorating the state explosion problem. To construct a proof that the parallel composition of M_1 with M_2, written as $M_1//M_2$, satisfies a correctness specification g, a typical compositional technique breaks this into two subproofs, performed in isolation on M_1 and M_2.

For instance, consider the rules in Figure 1. The notation $\{f\}M\{g\}$ indicates that model M satisfies specification g under the assumption f. The first hypothesis shows that M_1 satisfies h under the assumption f. The second shows that M_2 satisfies g under the assumption h. As a consequence of the semantics of synchronous composition, it follows that $M_1//M_2$ satisfies g under the assumption f. The auxiliary assertion h plays a role similar to that of determining an intermediate assertion in a Hoare-style proof for sequential composition, where $\{f\}S_1; S_2\{g\}$ is proved by supplying h such that $\{f\}S_1\{h\}$ and $\{h\}S_2\{g\}$ both hold.

In general, determining the appropriate auxiliary assertion h can be difficult. But once this is done, the hypotheses of the rule can be discharged by model checking properties of the *individual* processes, without reference to the global states that arise in the composition. This can lead to a dramatic drop in the cost of checking a property (cf. McMillan [1997]). In fact, in several instances, model checking a composition as a whole is simply infeasible.

The rule in Figure 1(a) is very similar to the Hoare rule for sequential composition. In effect, M_1 and M_2 can be viewed (for the proof) to be connected in

$$\frac{\{f\}M_1\{h\} \quad \{h\}M_2\{g\}}{\{f\}M_1//M_2\{g\}} \qquad \frac{\{f\}M_1\{\, g_2 \rhd g_1\} \quad \{f\}M_2\{\, g_1 \rhd g_2\}}{\{f\}M_1//M_2\{\mathsf{G}(g_1 \wedge g_2)\}}$$

(a) Noncircular reasoning (b) Syntactically circular reasoning

Fig. 1. Typical forms of compositional rules.

a pipeline. This might go against intuition, especially if M_1 and M_2 exchange a series of messages to achieve a joint result. To match the intuitive back-and-forth reasoning that is typically applied in such cases, several (so-called) *circular* proof rules have been proposed. An example is the rule from Figure 1(b). Several points differentiate this rule from the previous one. First, the form of the postconditions has been restricted to specific operators. The property $q \rhd p$ (read as "q constrains p") is true of a computation if, at all points i on the computation, p is true provided that q holds at all previous points. This can be expressed in linear temporal logic as $\neg(q \ \mathsf{U} \ \neg p)$. The property $\mathsf{G}(p)$ (read as "always p") is true of a computation if p holds at all points of the computation. Second, in the first subgoal, $\{f\}M_1\{g_2 \rhd g_1\}$, g_1 is understood to be the correctness assertion of M_1 while g_2 is a helper assertion. This may be justified by thinking of M_1 as an open system, which interacts with an environment over which it has little control. Thus, the correct operation of M_1 may be dependent on the correct operation of its environment M_2; therefore, g_2 appears as a guarantee of the correct operation of the environment in the proof of the correctness of M_1. The appearance of g_1 in the proof subtask for M_2 can be similarly justified—hence the use of the word *circular* in the name for such proof rules. The circularity helps to more easily encode the back-and-forth handshake protocols that designers typically use for connecting components of a system.

For any proof system, *soundness* is, of course, the most important property—it should not be possible to deduce false facts. A measure of the quality or usefulness of a proof system is obtained from an investigation into *completeness*—is it possible to deduce *all* true facts using the rules of the system? Existing compositional rules, including the circular ones, are known to be sound, but the completeness of compositional rules that refer to temporal properties has not been well studied.

In this article, we first investigate the completeness of compositional reasoning rules, focusing on rules used in practice that are known to be sound for arbitrary linear temporal properties. Surprisingly, several such rules turn out to be *incomplete*; that is, there are compositions whose correctness is not provable by following the proof rules. Typically, the unprovable assertions are liveness properties, but some rules may also be incomplete for safety properties. The counter examples for incompleteness are also quite simple, which indicates that the rules may be inadequate for handling many compositions that arise in practice. We propose a new circular reasoning rule similar to the one above and show that it is both sound and complete. The new rule strengthens the previous rule in a manner analogous to strengthening a proof of invariance by

introducing auxiliary assertions—to prove $\mathsf{G}p$ show that a stronger assertion, $(p \wedge h)$, is inductive. Furthermore, our new rule is backward compatible, in that any proof done using the previous rule is also a proof with the new circular rule.

We then investigate whether circularity is, in itself, essential for reasoning about composed systems. We show that the notion of circularity is a somewhat weak one for LTL properties, in that proofs carried out with circular rules can be easily translated to proofs with non-circular rules, and vice versa.

The article is organized as follows: Section 2 contains preliminary definitions; Section 3 gives the details of several different styles of proof rules and develops our new sound and complete circular proof rule; Section 4 derives our sound and complete circular rule; Section 5 generalizes to n-processes; Section 6 discusses the translations between proofs carried out with circular and noncircular rules. Finally, Section 7 contains a brief conclusion and discusses related work.

2. BACKGROUND

In this section, we define the computational model and provide examples of circular and non-circular rules for compositional reasoning.

2.1 Linear Temporal Logic (LTL)

LTL was first suggested as a protocol specification language in Pnueli [1977]. Formulas in the logic define sets of *infinite* sequences. LTL formula are defined relative to a set of *variable* symbols. As in first-order logic, one can construct *terms* over the set of variables using function symbols from a vocabulary, $\mathcal{F}$, and *atomic predicates* from terms, using relational symbols from a vocabulary, $\mathcal{R}$. A special function symbol, η (read "next") not in $\mathcal{F}$ is assumed to be given: informally, this symbol is used to indicate the value of a variable at a following state.

Atomic predicates and temporal formulas are defined below. A *predicate* is a Boolean combination of atomic predicates. $\mathbb{N}$ denotes the set of natural numbers.

—For a relational symbol $r \in \mathcal{R}$ of arity n and terms $t_0, \ldots, t_{n-1}$, $r(t_0, \ldots, t_{n-1})$ is an atomic predicate and a formula;

—for formulae f and g, $(f \wedge g)$ and $\neg(f)$ are formulae;

—for formulae f and g, $\mathsf{X}(f)$, and $(f \mathsf{U} g)$ are formulae.

The temporal operators are X (*next-time*), and U (*until*).

Given an interpretation $\mathcal{I}$ (which is fixed from now on) for the function and relation symbols, temporal formulae are interpreted with respect to infinite sequences of valuations of the variables. For a set of typed variables W, let a *W-state* be a function mapping each variable in W to a value in its type. The set of W-states is denoted by $\Sigma(W)$. A *W-sequence* σ is an infinite sequence of W-states, which is represented as a function $\sigma : \mathbb{N} \to \Sigma(W)$. The notation $\sigma, i \models f$ indicates that the infinite sequence σ *satisfies* the formula f at position i. The *language* of f, denoted by $\mathcal{L}(f)$, is the set $\{\sigma : \sigma, 0 \models f\}$. The satisfaction

relation can be defined by induction on the structure of f. First, the value of a term t at location i on σ, denoted by $t(\sigma, i)$, may be defined by induction on the structure of terms where, specifically, the interpretation of $\eta(t)$ at position i is given by the interpretation of t at position $i + 1$. We omit the detailed formulation. The satisfaction relation for formulas is defined as follows:

—$\sigma, i \models r(t_0, \ldots, t_{n-1})$ iff $(\mathcal{I}(r))(t_0(\sigma, i), \ldots, t_{n-1}(\sigma, i))$ is true.
—$\sigma, i \models \neg(f)$ iff $\sigma, i \models f$ is false.
—$\sigma, i \models (f \wedge g)$ iff both $\sigma, i \models f$ and $\sigma, i \models g$ are true.
—$\sigma, i \models \mathsf{X}(f)$ iff $\sigma, i + 1 \models f$.
—$\sigma, i \models (f \ \mathsf{U}\ g)$ iff there exists j, $j \geq i$, such that $\sigma, j \models g$ and for every k, $i \leq k < j$, $\sigma, k \models f$.

Other connectives can be defined in terms of these basic connectives: $(f \vee g)$ is $\neg(\neg f \wedge \neg g)$, $(f \Rightarrow g)$ is $\neg f \vee g$, $\mathsf{F}g$ (read as "eventually g") is $(true\ \mathsf{U}\ g)$, $\mathsf{G}f$ ("always f") is $\neg\mathsf{F}(\neg f)$, $(f\ \mathsf{W}\ g)$ ("f holds unless g") is $(\mathsf{G}(f) \vee (f\ \mathsf{U}\ g))$, $\overset{\infty}{\mathsf{F}}p$ ("infinitely often p") is $\mathsf{GF}p$, $\overset{\infty}{\mathsf{G}}p$ ("finitely often $\neg p$") is $\mathsf{FG}p$, and $q \rhd p$ ("q constrains p") is $\neg(q\ \mathsf{U} \neg p)$.

The "constrains" operator is used extensively in the formulation of compositional rules. An alternative formulation may give additional insight. The property $q \rhd p$ is true for a sequence σ at position 0 if, for all positions $i, i \geq 0$, p holds at i if q holds at all earlier positions. It follows that (i) p must be true at the initial position, (ii) $true \rhd p$ is equivalent to $\mathsf{G}p$, and (iii) $q \rhd p$ is monotonic in p and antimonotonic in q. We make use of these properties in the correctness proofs.

2.2 Extensions of LTL

The completeness proofs require the use of several extensions of LTL; these are defined below.

2.2.0.1 *LTL+Past.* LTL may be extended with "past" operators [Lichtenstein et al. 1985]. The past temporal operators are B ("back") and S ("since"). Their interpretation is given below.

—$\sigma, i \models \mathsf{B(f)}$ iff $i > 0$ and $\sigma, i - 1 \models f$.
—$\sigma, i \models (f\ \mathsf{S}\ g)$ iff there exists j, $0 \leq j \leq i$, such that $\sigma, j \models g$ and for every k, $j < k \leq i$, $\sigma, k \models f$.

The operator P ("previously") can be defined as $(true\ \mathsf{S}\ g)$. Extending LTL with the past operator does not increase expressive power [Gabbay et al. 1980], but does increase succinctness.

2.2.0.2 *LTL+Quantification.* The expressive power of temporal logic can be enhanced by allowing variable quantification. The formula $(\exists W : f)$ is true of a V-sequence σ iff there is a $V \cup W$-sequence δ that satisfies f, and agrees with σ on the variables in $(V \backslash W)$.

Adding quantification to LTL does increase both expressive power and succinctness. In the finite-state case, LTL+quantification is as expressive as ω-regular expressions—see Thomas [1990] for a survey of these issues.

2.2.0.3 *LTL+Past+Quantification.* In the completeness proofs, we often make use of formulas that involve both extensions. From the preceding, LTL+past+quantification is as expressive as LTL+quantification, but the use of the past operators results in succinct statements of the constructions and proofs.

2.3 Computational Model

We adopt a definition of a process similar to those in Pnueli [1977], Abadi and Lamport [1995], and McMillan [1999]. A process is specified by giving an initial condition, a transition condition and a fairness condition over a set of variables.

Definition 2.1 (*Left-Total Relation*). A binary relation T over a set S is *left-total* if for all $s \in S$ there is a $t \in S$ such that $(s, t) \in T$.

Definition 2.2 (*Process*). A *process* is specified by a tuple (V, I, T, F) where the following hold:

—V is a finite, nonempty set of typed *variables*. Let V' denote the set of terms $\{\eta(v) : v \in V\}$.
—$I(V)$, the *initial condition*, is a predicate on V.
—$T(V, V')$, the *transition condition*, is a left-total predicate on $V \cup V'$.
—$F(V, V')$, the fairness condition, is a Boolean combination of temporal formulas $\overset{\infty}{\mathsf{F}}(p)$ and $\overset{\infty}{\mathsf{G}}(p)$, for predicates p on $V \cup V'$.

Definition 2.3 (*Computation*). For a set of variables W such that $V \subseteq W$, a *W-computation* σ of a process is a W-sequence such that $I(\sigma_0)$, and, for each $i \in \mathbb{N}$, $\sigma, i \models T$.

Definition 2.4 (*Process Language*). For a set of variables W such that $V \subseteq W$, the *W-language* of a process $M = (V, I, T, F)$, denoted by $\mathcal{L}_W(M)$, is the set of W-computations of M that satisfy the fairness condition F. Thus, $\mathcal{L}_W(M)$ can be expressed by the LTL formula $I \wedge \mathsf{G}(T) \wedge F$, interpreted over W-sequences.

We define process composition so that the language of a composition $M_1 // M_2$ is simply the *intersection* of the languages of M_1 and M_2. The semantics of most hardware description languages follows this model. In addition, as shown in Abadi and Lamport [1995], with some reasonable restrictions, it holds also of asynchronous models of computation.

Definition 2.5 (*Synchronous Process Composition*). Given processes $M_1, \ldots, M_n$, with $M_i = (V_i, I_i, T_i, F_i)$, their *composition* is the process denoted by $M = (//i : i \in [1..n] : M_i)$, where $M = (V, I, T, F)$ such that

—$V = (\cup i : V_i)$,
—$I = (\wedge i : I_i)$,
—$T = (\wedge i : T_i)$,
—$F = (\wedge i : F_i)$.

When M is composed of processes M_1 and M_2, we may express this composition as $M = M_1 // M_2$.

Definition 2.6 (*Interface Variable*). For a system M, composed of several processes M_i, we refer to the interface, or interface variables of M_i, as those variables in $V_i \cap (\cup_{j \neq i} V_j)$.

With these definitions of composition, it is possible that $T = T_1 \wedge T_2$ is not left-total even though the T_i's are left-total. For instance, suppose that $x \in V_1$ and $x \in V_2$, where for at least one initial state of $M = M_1 // M_2$, $x = 0$. Furthermore, assume that whenever $x = 0$, then any transition in T_1 forces x to 1 in the next state but T_2 requires that x is 0 in the next state. Then the composed model M does not have a transition from a state where $x = 0$ and T is not left-total. In the rest of the article, we restrict attention to those compositions where T is left-total and nonempty.

THEOREM 2.7 (COMPOSITION THEOREM). *For a composition* $M = (//i : i \in [1..n] : M_i)$ *and a set of variables* W *such that* $(\cup i : V_i) \subseteq W$, *it is the case that* $\mathcal{L}_W(M) = (\cap i : \mathcal{L}_W(M_i))$.

PROOF. Let σ be a computation in $\mathcal{L}_W(M)$. Then $\sigma \models I \wedge F \wedge \mathsf{G}(T)$. So for all i in $[1..n]$, $\sigma \models I_i$, and $\sigma \models F_i$. As G distributes over conjunction, for all i in $[1..n]$, $\sigma \models \mathsf{G}(T_i)$. This implies the right-hand side.

In the other direction, if σ is not a computation of $\mathcal{L}_W(M)$, then it violates either I or F or $\mathsf{G}(T)$. In either case, it violates one of I_i, F_i, or $\mathsf{G}(T_i)$, for some i; thus, it does not satisfy $\mathcal{L}_W(M_i)$. □

Definition 2.8 (*Model Checking*). Let f be a property defined over a variable set W, and let program M have a variable set that is a subset of W. The *model checking* question is to determine whether f holds for all computations of M, that is, whether $(\forall W : \mathcal{L}_W(M) \Rightarrow f)$ is true.

2.4 Compositional Reasoning

The model checking question for a composition $M_1 // M_2$ may be phrased as $(\forall W : \mathcal{L}_W(M_1 // M_2) \Rightarrow f)$, which is equivalent, by Theorem 2.7, to $(\forall W : \mathcal{L}_W(M_1) \wedge \mathcal{L}_W(M_2) \Rightarrow f)$. Compositional reasoning rules convert this question into two separate model checking questions, one explicitly involving M_1 and the other explicitly involving M_2. This separation is typically required in the proofs of large systems because even the symbolic computation (as BDDs) of the transition relation for $M_1 // M_2$ may be infeasible.

Assume-guarantee rules for composition attempt to generalize the pre- and postcondition reasoning of Hoare logic [Hoare 1969]. Informally, a triple $\{f\}M\{g\}$ asserts the property that every computation of M that satisfies the *assumption* f satisfies the *guarantee* g. Formally, this can be stated as $(\forall W : f \wedge \mathcal{L}_W(M) \Rightarrow g)$. We state below two typical compositional reasoning rules based on the assume-guarantee formulation: the first is syntactically noncircular, while the second is syntactically circular, in that the assumptions of one process form the guarantees of the other and vice versa.

Definition 2.9 (*Noncircular Reasoning*). For a composition $M = (//i : i \in [1..n] : M_i)$, find intermediate assertions h_j such that f implies h_0, h_n implies g, and $\{h_{i-1}\}M_i\{h_i\}$ holds for each i in $[1..n]$. Then $\{f\}M\{g\}$ holds.

PROPOSITION 2.10 (NC SOUNDNESS). *The noncircular reasoning rule is sound.*

PROOF. Suppose that assertions h_j have been found that satisfy the assumptions of the noncircular reasoning rule. The soundness proof is by induction on process indices. The inductive hypothesis is that $\{f\}(//j : j \in [1..i] : M_j)\{h_i\}$ holds at the ith step.

The basis ($i = 1$) holds by the first assumption. The inductive hypothesis, together with the assumption $\{h_i\}M_{i+1}\{h_{i+1}\}$, ensures that the inductive hypothesis holds for $i + 1$. This follows from the general statement that $\{f\}M_1\{h\}$ and $\{h\}M_2\{g\}$ implies $\{f\}M_1//M_2\{g\}$, for any f, h, g, and any M_1, M_2. To see this, note that the first triple is (for W chosen to contain the variables of all formulas involved) $(\forall W : f \wedge \mathcal{L}_W(M_1) \Rightarrow h)$, and the second is $(\forall W : h \wedge \mathcal{L}_W(M_2) \Rightarrow g)$. The result, which is $(\forall W : f \wedge \mathcal{L}_W(M_1//M_2) \Rightarrow g)$, follows by the transitivity of implication and the composition theorem.

The claim follows by instantiating the induction hypothesis for $i = n$. □

PROPOSITION 2.11 (NC COMPLETENESS). *The noncircular reasoning rule* ***NC*** *is complete.*

PROOF. Given that $\{f\}M\{g\}$, where $M = //i \in [1..n] : M_i$, we have to show that there exist assertions $\{h_i\}$ which satisfy the conditions of the rule. Choose $h_i = f \wedge (\wedge j : j \in [1..i] : \mathcal{L}_W(M_j))$. (Note that this is equivalent, by the composition theorem, to $f \wedge \mathcal{L}_W(//j : j \in [1..i] : M_j)$.) Then, $h_0 = f$, and $h_n = f \wedge (\wedge j : j \in [1..n] : \mathcal{L}_W(M_j))$, which implies g by the hypothesis. For any i, consider $\{h_{i-1}\}M_i\{h_i\}$. This expands to $(\forall W : f \wedge (\wedge j : j \in [1..(i-1)] : \mathcal{L}_W(M_j)) \wedge \mathcal{L}_W(M_i) \Rightarrow f \wedge (\wedge j : j \in [1..i] : \mathcal{L}_W(M_j)))$, which is trivially true. □

While the rule is, in some sense, trivially complete, this seems to be the nature of completeness proofs for compositional reasoning. It remains to be shown that the rule is readily applicable in practice. We note that interest in circular reasoning stems from the fact that rules of this type are not seen to be straightforward to apply.

The following rule is derived from the application of a property decomposition theorem to compositional reasoning in McMillan [1999] (cf. Theorem 1 in McMillan [1999]). Below we make use of the following notation: let $B = \{f_1, \ldots, f_k\}$ be a set of LTL formulae, then we use the set B as a shorthand for $(\bigwedge i : f_i)$.

Definition 2.12 (*Syntactically Circular Reasoning (Rule* ***C1****)*). Consider the composition $M = (//j : M_j)$. Let $\{g_i\}$ be a set of properties. To show that $\{f\}M\{\mathsf{G}(\bigwedge i : g_i)\}$ holds,

—for each i, choose a composition $M(i) = (//k : M_k)$ where k ranges over a strict subset of the process indices that includes i, and,

—choose a well founded order $\prec$ and subsets $\{\Theta_i\}$ and $\{\Delta_i\}$ of the set of properties $\{g_i\}$, such that if $g_j \in \Theta_i$, then $j \prec i$.

Then show that $\{f\}M(i)\{\ \Delta_i \rhd (\neg\Theta_i \vee g_i)\}$ holds for all i.

The requirement that $M(i)$ is a strict subcomposition of M is imposed to prevent trivial applications of this rule with every $M(i)$ equal to M.

THEOREM 2.13. *The rule **C1** is sound.*

PROOF. By assumption we have that $\{f\}M(i)\{\Delta_i \rhd (\neg\Theta_i \vee g_i)\}$. That is $\forall W : (f \wedge I_{M(i)} \wedge \mathsf{G}_{M(i)} \wedge F_{M(i)}) \Rightarrow (\Delta_i \rhd (\neg\Theta_i \vee g_i))$. Notice that, for all i, the variable set for $M(i)$ is contained in the variable set for M and that, by definition, the initial condition, transition relation, and fairness conditions for M are more restrictive than those for $M(i)$. Therefore $I_M \Rightarrow I_{M(i)}$, $\mathsf{G}(T_M) \Rightarrow \mathsf{G}(T_{M(i)})$, and $F \Rightarrow F_i$ for all i. Hence, for any computation $\sigma \in \mathcal{L}_W(M)$, it must also be that $\sigma \in \mathcal{L}_W(M(i))$ for all i. Since by assumption $\sigma \models f$, we have that $\sigma \models \Delta_i \rhd (\neg\Theta_i \vee g_i)$ for all i.

Suppose, to the contrary, that there is a computation σ of M which satisfies f but fails to satisfy $\mathsf{G}(\bigwedge i : g_i)$. Consider the first position, k, along σ where $(\bigwedge i : g_i)$ does not hold. Pick a minimal element, m, among the set of failure indices $\{i : \neg g_i\}$. As m is minimal in this set, g_l must be true for all indices l that are below m in the order. Hence, Θ_m is true at position k; therefore, $(\neg\Theta_m \vee g_m)$ is false at position k. By the definition of the constrains operator, Δ_m must be false at some position before k. But this is a contradiction, as Δ_m is a subset of the $\{g_i\}$'s, and k is the first position where $(\bigwedge i : g_i)$ is false. □

3. (IN)COMPLETE PROOF RULES

We have shown in the previous section that the noncircular rule **NC** is both sound and complete. In this section, we consider the proof rule **C1** and the assume-guarantee rule from Abadi and Lamport [1995] and show that these circular rules are incomplete, even for *finite-state* processes. Our choice of these rules is guided by two considerations: (i) these rules, unlike many other compositional rules (as discussed in Section 7), are sound for arbitrary linear temporal properties, and (ii) they have been used successfully (cf. McMillan [1998]) to verify large systems. We then present a new sound and complete circular rule.

3.1 Incompleteness of **C1**

To demonstrate that rule **C1** is incomplete, consider the programs below where $M = M_1 // M_2$. Informally, M_1 and M_2 juggle four tokens by throwing them back and forth in a circular pattern (the l, r variables indicate left and right "hands," respectively). System transitions are described as follows: we use $\eta(l)$ to denote the value of variable l in the *next* state.

program M_1	program M_2
variables $l_1, r_1, r_2 : \mathit{boolean}$	variables $l_2, r_2, r_1 : \mathit{boolean}$
initially $l_1 \wedge r_1$	initially $l_2 \wedge r_2$
transition $(\eta(l_1) \equiv r_2) \wedge (\eta(r_1) \equiv l_1)$	transition $(\eta(l_2) \equiv r_1) \wedge (\eta(r_2) \equiv l_2)$

As can be checked easily, the property $\mathsf{G}(l_1 \wedge l_2)$ holds of the composition $M_1 // M_2$. All the variables, $l_1, l_2, r_1,$ and r_2 are true initially. Since the variable l_i is assigned the current value of $r_{i'}$, in the state after the initial state l_i must be

true. Similarly, the r_i's will be true in the state after the initial one. This pattern continues throughout the computation of M. Applying the substitution $f = true, g_1 = l_1, g_2 = l_2$ to rule **C1**, we obtain the property $\{true\}M_1//M_2\{\mathsf{G}(l_1 \wedge l_2)\}$.

However, as can be checked by enumeration, there is no way to define the well-founded order $\prec$, and the subsets Θ_i and Δ_i such that the subgoals of rule **C1** are satisfied. Intuitively, this is because the next value of l_1 is determined by the current value of r_2, which is unconstrained by the assumptions. Hence, *the original property, which is true of the composition, cannot be shown using the proof rule **C1***. This is shown formally below.

Recall that the goal is to show $\{true\}M(i)\{\ \Delta_i \rhd (\neg\Theta_i \vee g_i)\}$, where $g_i = l_i$. for both $M(1)$ and $M(2)$. Here, $M(1) = M_1$ and $M(2) = M_2$. Consider the case where 1 is a minimal element of the order—the case where 2 is minimal has a symmetric proof. By minimality, Θ_1 must be empty; thus, the goal for $i = 1$ simplifies to $\{true\}M(1)\{\ \Delta_1 \rhd l_1\}$. We show that this must be false by showing that the weaker property $\{true\}M(1)\{\ (l_1 \wedge l_2) \rhd l_1\}$ is false. The new property is weaker as constrains is antimonotonic in its left-hand argument, and $(g_1 \wedge g_2)$ is stronger than any Δ property. Since M_1 does not constrain r_2, consider a computation of M_1 where l_2 is true at the initial state, and r_2 is always false. The first state of this computation satisfies $(l_1 \wedge l_2)$, but l_1 is false in the second state. This shows that the property $\{true\}M(1)\{\ (l_1 \wedge l_2) \rhd l_1\}$ is not valid, and proves that the rule cannot be applied to this instance.

One reason that this rule is incomplete is that it does not permit a choice of auxiliary assertions, as in rule **NC**. If auxiliary assertions were allowed, it is easy to see that the strengthened property $\mathsf{G}(l_1 \wedge r_1 \wedge l_2 \wedge r_2)$ can be shown by properly instantiating rule **C1**. An analogy can be drawn here to the well-known difference between invariance and *inductive* invariance: to establish that a property p is invariant, one often needs to strengthen p to $p \wedge h$, for some h, and show that this strengthened formula is an inductive invariant. We say more about strengthening in Section 4.

3.2 Other Incomplete Rules

The circular proof rule presented in Abadi and Lamport [1995] is given below. This rule is also used to reason about the joint behaviors of $N_1//N_2$ by reasoning about the behaviors of the N_is in isolation. However, there are several key differences, in particular while, for instance, rule **C1** is used to show that the behaviors of $N_1//N_2$ satisfy temporal properties of interest, the Abadi-Lamport rule given below is used in proofs of process refinement; that is, one attempts to show that the behaviors of $N_1//N_2$ are correct by showing that all the behaviors of $N_1//N_2$ are behaviors of an assumed correct system $M_1//M_2$.

For the rule below, taken from Abadi and Lamport [1995], // can be used to represent either synchronous or asynchronous composition and $\hat{\mathcal{L}}(M)$, the language of computations of M, is defined so that it is insensitive to stuttering. A sequence σ is stuttering equivalent to another sequence, δ, if σ can be transformed into δ by repeating or reducing adjacent duplicate states. For example, the sequences *aaaabb* and *abbb* are stuttering equivalent. The language $\hat{\mathcal{L}}(M)$ is insensitive to stuttering if, whenever a sequence σ is in the language,

all sequences that are stuttering equivalent to σ are also in the language. Composition is defined so that $\hat{\mathcal{L}}(M_1//M_2)$ equals $\hat{\mathcal{L}}(M_1) \wedge \hat{\mathcal{L}}(M_2)$.

Although this rule allows the choice of auxiliary assertions E_i, it turns out that it is still incomplete, because of the restricted form of the hypotheses. We recall several of the definitions from Abadi and Lamport [1995] needed to present the results, including the temporal '+' operator, which extends safety properties by allowing them to hold for some finite amount of time.

In the definition below, $\mathcal{C}(f)$, for an LTL property f, is the strongest safety property that is weaker than f while f_{+v} asserts that if the formula f should become false then the variable v becomes constant (for more details see Abadi and Lamport [1995]). The following definitions are all taken from Abadi and Lamport [1995].

Definition 3.1 (*Safety Property*). A property, p, is a safety property if, for every computation σ, if every finite prefix of σ can be extended into a computation in p, then σ must also be in p.

Definition 3.2 (*Process Closure*). Let p be a property; then $\mathcal{C}(p)$ is the strongest safety property that is weaker than p.

Definition 3.3 (*Temporal '+'*). Let p be a temporal formula with variable v and let σ be a computation; then $\sigma \models p_{+v}$ iff $\sigma \models p$, or there exists an $i \in \mathbb{N}$ such that for all $j < i$, $\sigma, j \models p$, and for all $j \geq i$, $\sigma_j(v) = \sigma_{j+1}(v)$.

Definition 3.4 (*Circular Rule* ***C2***). To show that $E \wedge \hat{\mathcal{L}}(N_1//N_2) \Rightarrow \hat{\mathcal{L}}(M_1//M_2)$ holds, pick E_i and show that for each i in $\{1, 2\}$ all the following hold:

—$\mathcal{C}(E) \wedge \bigwedge_{j\in\{1,2\}} \mathcal{C}(\hat{\mathcal{L}}(M_j)) \Rightarrow E_i$, and
—$\mathcal{C}(E_i)_{+v} \wedge C(\hat{\mathcal{L}}(N_i)) \Rightarrow \mathcal{C}(\hat{\mathcal{L}}(M_i))$, and
—$E_i \wedge \hat{\mathcal{L}}(N_i) \Rightarrow \hat{\mathcal{L}}(M_i)$.

This rule was shown to be sound in Abadi and Lamport [1995]. However, consider the programs M_1, M_2, N_1, and N_2 given below, all of which have initial condition *true* and a weak-fairness condition on the actions a_1, a_2.

program M_1	program M_2
variables x : *boolean*	variables y : *boolean*
transition a_1: $\eta(x) = true$, b_1: $\eta(x) = false$	transition a_2: $\eta(y) = true$, b_2: $\eta(y) = false$
fairness $\mathsf{GF}a_1$	fairness $\mathsf{GF}a_2$

Thus, the specification programs M_1 and M_2 define the properties $\mathsf{GF}x$ and $\mathsf{GF}y$, respectively. The implementation programs N_1 and N_2 are as follows:

program N_1	program N_2
variables x, y : *boolean*	variables x, y : *boolean*
transition a_1: $\eta(x) = y$	transition a_2: $\eta(y) = true$, b_2: $\eta(y) = (\neg x \wedge y)$
fairness $\mathsf{GF}a_1$	fairness $\mathsf{GF}a_2$

It is easy to check that $E \wedge \hat{\mathcal{L}}(N_1//N_2) \Rightarrow \hat{\mathcal{L}}(M_1//M_2))$ with $E \equiv \mathit{true}$. By a result in Abadi and Lamport [1995], $\mathcal{C}(\hat{\mathcal{L}}(M))$ is just the temporal formula for process M without the fairness condition. Thus, $\mathcal{C}(\hat{\mathcal{L}}(M_1)) = \mathit{true} \wedge \mathsf{G}(\eta(x) \equiv \mathit{true} \vee \eta(x) \equiv \mathit{false} \vee \eta(x) \equiv x)$, which simplifies to *true*, as does $\mathcal{C}(\hat{\mathcal{L}}(M_2))$. Hence, if the first hypothesis is to hold, both E_1 and E_2 must equal *true*. Therefore, by the third hypothesis, $\hat{\mathcal{L}}(N_1) \Rightarrow \hat{\mathcal{L}}(M_1)$, which is false as N_1 admits computations where x is *false* at every point. Hence, rule **C2** is incomplete.

4. A SOUND AND COMPLETE CIRCULAR RULE

The discussion in the previous section shows that some of the circular reasoning rules that have been proposed in the literature are incomplete. This raises the question: what should a complete circular rule look like?

We answer this question by deriving such a rule from first principles. The rule permits the choice of *auxiliary* properties over process interfaces (Definition 2.6), while retaining the overall style of the proof obligations of rule **C1**. This rule is shown to be sound and complete. For clarity, we restrict the discussion below to the two-process case—we show in Section 5 how to generalize to the n-process case. The derivation of the new rule is carried out in the completeness proof: the new rule is derived, and shown to be complete, in one step. For this reason, we present the completeness proof first.

4.1 Deriving the Rule

There are two notable aspects of the new rule: first, it generalizes **C1**; thus it is "backward compatible." Second, the completeness proof shows that it suffices to pick the auxiliary assertions over the *interface* between the two processes. In a well-designed program, this interface should be "narrow," exposing only a small part of the internal workings of each process, and it must have a simple description. This result makes precise the usual informal claim that compositional reasoning is "easier" than full verification. Of course, this is only true if the interface is available, or can otherwise be easily derived by inspection. We do not, in this work, discuss algorithms for deriving such interface specifications (one such algorithm is implicit in the derivation of the method). This aspect is discussed more fully in Section 7 on related work.

Definition 4.1 (*Circular Reasoning (Rule **C3**)*). For properties g_1 over V_1 and g_2 over V_2, to show $\{f\}M_1//M_2\{\mathsf{G}(g_1 \wedge g_2)\}$, pick properties h_1 and h_2 for which the following two obligations hold:

(1) $\{f\}M_1\{(h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)\}$.
(2) $\{f\}M_2\{(h_1 \wedge g_1) \rhd ((h_1 \Rightarrow g_2) \wedge h_2)\}$.

A natural extension of rule **C1** would be to have the obligations take the symmetric form $\{f\}M_1\{(h_2 \wedge g_2) \rhd (h_1 \wedge g_1)\}$; $\{f\}M_2\{(h_1 \wedge g_1) \rhd (h_2 \wedge g_2)\}$. The completeness proof shows that this is not quite possible. The hypotheses of **C3** are individually weaker than these more symmetric hypotheses as, for instance, the guarantee term $(h_1 \wedge g_1)$ implies the term $((h_2 \Rightarrow g_1) \wedge h_1)$. Thus, a proof using the symmetric hypothesis is also a valid proof according to **C3**.

THEOREM 4.2 (COMPLETENESS OF **C3**). *Rule **C3** is complete. Furthermore, if f is defined over the interface variables $V_1 \cap V_2$, it is always possible to choose h_1 and h_2 as properties over $V_1 \cap V_2$.*

PROOF. In the following, let $V = V_1 \cup V_2$. Assume that $\{f\}M_1//M_2\{\mathsf{G}(g_1 \wedge g_2)\}$ holds.

By definition, this is equivalent to $(\forall V : f \wedge \mathcal{L}_V(M_1) \wedge \mathcal{L}_V(M_2) \Rightarrow \mathsf{G}(g_1 \wedge g_2))$. As G distributes over $\wedge$, this is equivalent to the conjunction of (1) and (2) below.

$$(\forall V : f \wedge \mathcal{L}_V(M_1) \wedge \mathcal{L}_V(M_2) \Rightarrow \mathsf{G}(g_1)), \tag{1}$$

$$(\forall V : f \wedge \mathcal{L}_V(M_1) \wedge \mathcal{L}_V(M_2) \Rightarrow \mathsf{G}(g_2)). \tag{2}$$

Let $\alpha_1 = (\exists V \backslash V_2 : f \wedge \mathcal{L}_V(M_1))$ and $\alpha_2 = (\exists V \backslash V_1 : f \wedge \mathcal{L}_V(M_2))$. Informally, α_i is the language defined by M_i at its interface.

As M_1 and g_1 are defined over V_1, expression (1) can be rewritten as

$$(\forall V_1 : \mathcal{L}_V(M_1) \wedge \alpha_2 \Rightarrow \mathsf{G}(g_1)). \tag{3}$$

By definition of α_1, it is also true that

$$(\forall V : f \wedge \mathcal{L}_V(M_1) \Rightarrow \alpha_1). \tag{4}$$

It would be nice to have a strengthening of **C1** where the first obligation is to show $\{f\}M_1\{(h_2 \wedge g_2) \rhd (h_1 \wedge g_1)\}$. However, this implies that g_1 and h_1 must hold at the initial point of any computation that satisfies $f \wedge \mathcal{L}_V(M_1)$. By (4), such computations satisfy α_1 initially. Thus, it is reasonable to define h_1 as $\mathsf{P}\alpha_1$; informally, h_1 says that α_1 holds either now or in the past. But it is not possible to prove that g_1 holds initially; the best that can be said (by (3)) is that $(\alpha_2 \Rightarrow g_1)$ holds initially. With a symmetric definition of h_2 as $\mathsf{P}\alpha_2$, this becomes $(h_2 \Rightarrow g_1)$. Hence, the form of the first obligation is modified to $\{f\}M_1\{(h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)\}$.

The preceding discussion shows that this assume-guarantee triple holds at the initial point. Now consider an arbitrary position $i+1$ on the sequence such that $(h_2 \wedge g_2)$ holds for all positions j, $j \leq i$. Thus, h_2 holds initially, which implies that α_2 is true initially. By Equation (3), $\mathsf{G}(g_1)$ holds at the origin, so $(h_2 \Rightarrow g_1)$ is true at point $i+1$. By the definition of h_1 and (4), h_1 is true at point $i+1$.

Hence, the first hypothesis is true. In a similar manner, one may argue that the second hypothesis is also true; so the rule is complete.

Note that, if f is defined over $V_1 \cap V_2$, the auxiliary assertions h_1 and h_2 are also defined over the common interface variables $V_1 \cap V_2$. □

THEOREM 4.3 (SOUNDNESS OF **C3**). *Rule **C3** is sound.*

PROOF. Assume that the hypotheses of the rule hold for some choice of h_1 and h_2. Then the guarantees of both hypotheses are true for any computation of $M_1//M_2$ that satisfies f. Consider any such computation σ, and any point i on σ. Since σ is a computation of $M_1//M_2$, and by assumption $\mathcal{L}_W(M_1//M_2) = \mathcal{L}_W(M_1) \cap \mathcal{L}_W(M_2)$, it follows that σ is in $\mathcal{L}_W(M_1)$. Therefore, by the first hypothesis, $\sigma \models (h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)$. Hence, initially

$\sigma, 0 \models ((h_2 \Rightarrow g_1) \wedge h_1)$. Therefore, $\sigma, 0 \models h_1$. Similarly, since σ is also a computation of M_2, and by assumption $\sigma \models (h_1 \wedge g_1) \rhd ((h_1 \Rightarrow g_2) \wedge h_2)$, we must have that $\sigma, 0 \models h_2$. But then it follows that $\sigma, 0 \models g_1$ and that $\sigma, 0 \models g_2$. Putting these facts together, we have that $\sigma, 0 \models h_1 \wedge g_1 \wedge h_2 \wedge g_2$.

Now assume inductively that, for all points j, $j \leq i$, the property $(g_1 \wedge h_1 \wedge g_2 \wedge h_2)$ holds. Together with the first hypothesis of rule **C3**, $\sigma \models (h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)$, it must be the case that $\sigma, i+1 \models (h_2 \Rightarrow g_1) \wedge h_1$. Similarly, by the second hypothesis, $(h_1 \Rightarrow g_2) \wedge h_2$ holds at the $(i+1)$st position. Reasoning similar to the base case shows that $\sigma, i+1 \models h_1 \wedge g_1 \wedge h_2 \wedge g_2$. Hence, the inductive hypothesis holds at point $i+1$. This shows that $\mathsf{G}(g_1 \wedge h_1 \wedge g_2 \wedge h_2)$ holds of σ, from which it follows that the weaker property $\mathsf{G}(g_1 \wedge g_2)$ also holds of σ. □

4.2 Relationship to Rule **C1**

An instance of rule **C1** can be obtained from **C3** by making the substitution $h_1 = \mathit{true}, h_2 = \mathit{true}$. In other words, if the g_i's hold initially, and if g_1 and g_2 hold on the first n states of a computation, then g_1 is guaranteed to hold on the $n+1$st state of the computation. Similarly for g_2. Then $g_1 \wedge g_2$ is an invariant of the computations of $M_1 // M_2$.

For the first example, in Section 3.1, that showed the incompleteness of rule **C1**, the following choices for h_1 and h_2 over the common variables $\{r_1, r_2\}$ ensure that the hypotheses of rule **C3** hold: $h_1 = r_1, h_2 = r_2$. With this substitution, the hypotheses become (1) $\{\mathit{true}\}M_1\{(r_2 \wedge l_2) \rhd ((r_2 \Rightarrow l_1) \wedge r_1)\}$, and (2) $\{\mathit{true}\}M_2\{(r_1 \wedge l_1) \rhd ((r_1 \Rightarrow l_2) \wedge r_2)\}$.

For the second example, in Section 3.2, that showed the incompleteness of rule **C2**, the property to be satisfied by $N_1 // N_2$ may be written as $\mathsf{G}(\mathsf{GF}x \wedge \mathsf{GF}y)$. The following choices for h_1 and h_2 over the common variables x, y ensure that the hypotheses of rule **C3** hold: $h_1 = \mathit{true}, h_2 = \mathsf{GF}y$. With this substitution, the hypotheses simplify to (1) $\{\mathit{true}\}M_1\{\mathsf{GF}y \rhd (\mathsf{GF}y \Rightarrow \mathsf{GF}x)\}$, and (2) $\{\mathit{true}\}M_2\{\mathsf{GF}x \rhd \mathsf{GF}y\}$.

5. GENERALIZING TO *N*-PROCESSES

We present a sound and complete n-process circular reasoning rule, a generalization of the rule **C3**.

Definition 5.1 (*Circular Rule **C4***). Let $M = (//i \in [1..n] : M_i)$ be a composition of processes, and let $\{g_i\}$ be a set of properties such that, for each i, g_i is a property over M_i. Let $g = (\bigwedge i : g_i)$ and $g(i) = (\bigwedge j \neq i : g_j)$. To show that $\{f\}M\{\mathsf{G}(g)\}$ holds, pick an auxiliary property set $\{h_i\}$, one property for each M_i. Then let $h = (\bigwedge i : h_i)$ and show that, for each i, the following holds: $\{f\}M_i\{(h \wedge g(i)) \rhd ((h \Rightarrow g_i) \wedge h_i)\}$.

THEOREM 5.2. *Rule **C4** is sound.*

PROOF. Assume that the hypotheses of the rule hold for some choice of $\{h_i\}$. That is, $\{f\}M_i\{(h \wedge g(i)) \rhd ((h \Rightarrow g_i) \wedge h_i)\}$ for each i. Then the guarantees of the hypotheses are true for any computation of M that satisfies f.

Consider any such computation of M, say σ, and any point k on σ. Since σ is a computation of M, and by assumption $\mathcal{L}_W(M) = \bigwedge i : \mathcal{L}_W(M_i)$, $\sigma \in \mathcal{L}_W(M_i)$ for all $i \in [1..n]$. Therefore, by the hypothesis, $\sigma \models (h \wedge g(i)) \rhd ((h \Rightarrow g_i) \wedge h_i)$.

Hence, initially $\sigma, 0 \models ((h \Rightarrow g_i) \wedge h_i)$. Therefore, $\sigma, 0 \models h_i$. Similarly, since σ is also a computation of M_j for all $j \neq i$, and by assumption $\sigma \models (h \wedge g_j) \rhd ((h \Rightarrow g_j) \wedge h_j)$, we must have that $\sigma, 0 \models h_j$. But then it follows that $\sigma, 0 \models g_i$. Hence, $\sigma, 0 \models g$. Putting these facts together we have that $\sigma, 0 \models h \wedge g$.

Now assume inductively that for all points l, $l \leq k$, the property $h \wedge g$ holds. Together with the hypotheses of rule **C4**, $\sigma \models (h \wedge g(i)) \rhd ((h \Rightarrow g_i) \wedge h_i)$, this implies that $\sigma, k+1 \models (h \Rightarrow g_i) \wedge h_i$ for each i. Putting these facts together, implies that $\sigma, k \models h$. Hence for each i, $\sigma, k \models h$ and therefore, for each i, $\sigma, k \models g_i$. Hence $\sigma, k \models h \wedge g$ and therefore that $\sigma, k \models g$. This shows that $\sigma \models \mathsf{G}(g)$ and therefore that $\{f\}M\{\mathsf{G}(g)\}$ holds. □

THEOREM 5.3 (COMPLETENESS). *Rule **C4** is complete for every property f that is independent of the local variables X.*

PROOF. The proof is quite similar to the completeness proof for rule **C3**. To simplify the argument, assume that the variables V_i of each process M_i are partitioned into X_i, the local variables, and Y_i, the interface variables (i.e., each variable in Y_i is referred to in some process M_j, for $j \neq i$). The language of M_i, $\mathcal{L}_W(M_i)$, is then given by a formula $L_i(X_i, Y_i) = I_i \wedge \mathsf{G}(T_i) \wedge F_i$. We let L denote the formula representing $\mathcal{L}_W(M)$ built from the intersection of the languages of the individual processes.

Assume that $\{f\}M\{\mathsf{G}(g)\}$ holds. Hence, $\{f\}M\{\mathsf{G}(g_i)\}$ holds, for each i. Using the assume-guarantee definition, this is equivalent to $(\forall V : f \wedge \mathcal{L}(M) \Rightarrow \mathsf{G}(g_i))$. As g_i is defined over V_i, this can be rewritten as

$$\Big(\forall V_i : L_i(X_i, Y_i) \wedge \Big(\exists V \backslash V_i : f \wedge \Big(\bigwedge j \neq i : L_j(X_j, Y_j)\Big)\Big) \Rightarrow \mathsf{G}(g_i)\Big). \tag{5}$$

Define $\beta_i = (\exists V \backslash V_i : f \wedge (\bigwedge j : j \neq i : L_j(X_j, Y_j)))$, which can also be written in terms of the X_i, Y_i variables as: $\beta_i = (\exists X, Y \backslash Y_i : f \wedge (\bigwedge j : j \neq i : L_j(X_j, Y_j))$, where $X = (\cup i : X_i)$ and $Y = (\cup i : Y_i)$. Informally, β_i is the language defined by all other processes at the interface to process M_i.

Let $\alpha_i = (\exists X_i : f \wedge L_i)$. Informally, α_i is the language defined by M_i at its interface. Now define $h_i = \mathsf{P}(\alpha_i)$.

Lemma 5.4 (proved below) is used, in conjunction with Equation (5), to show that $(h \Rightarrow g_i)$ holds at the initial state of a M_i computation. The rest of the argument is similar to that in the earlier completeness proof. It follows that that, for each i, the assume-guarantee triple $\{f\}M_i\{(h \wedge g) \rhd ((h \Rightarrow g_i) \wedge h_i)\}$ holds. □

LEMMA 5.4. $(\bigwedge i : \mathsf{P}(\alpha_i)) \Rightarrow h_j$, *for any* j.

PROOF. From the definitions, this is equivalent to $(\bigwedge i : \mathsf{P}(\alpha_i)) \Rightarrow \mathsf{P}(\beta_j)$, which is equivalent, since this must be true at point 0 of every computation, to $(\bigwedge i : \alpha_i) \Rightarrow \beta_j$. Notice that β_j is defined over Y_j, while the antecedent is defined

over Y. Thus, the implication is equivalent to

$$\Big(\exists Y \backslash Y_j : \Big(\bigwedge i : \alpha_i\Big)\Big) \Rightarrow \beta_j.$$

Expanding the definitions of α and β, we get

$$\Big(\exists Y \backslash Y_j : \Big(\bigwedge i : (\exists X_i : f \wedge L_i)\Big)\Big) \Rightarrow \Big(\exists V \backslash Y_j : f \wedge \Big(\bigwedge k : k \neq j : L_k\Big)\Big),$$

which is true if

$$\Big(\exists Y \backslash Y_j : \Big(\bigwedge i : (\exists X_i : f \wedge L_i)\Big)\Big) \equiv \Big(\exists V \backslash Y_j : f \wedge \Big(\bigwedge i : L_i\Big)\Big)$$

holds for any j. We show this equivalence as follows. Starting with the right-hand side,

$$\begin{array}{ll} & \Big(\exists V \backslash Y_j : f \wedge \Big(\bigwedge i : L_i(X_i, Y_i)\Big)\Big) \\ \equiv & \quad \text{(splitting } V \text{ into } X \text{ and } Y\text{)} \\ & \Big(\exists X, Y \backslash Y_j : f \wedge \Big(\bigwedge i : L_i(X_i, Y_i)\Big)\Big) \\ \equiv & \quad \text{(pushing } f \text{ inside, as the range of } i \text{ is nonempty)} \\ & \Big(\exists X, Y \backslash Y_j : \Big(\bigwedge i : f \wedge L_i(X_i, Y_i)\Big)\Big) \\ \equiv & \quad \text{(rearranging quantifiers)} \\ & \Big(\exists Y \backslash Y_j : \Big(\exists X : \Big(\bigwedge i : f \wedge L_i(X_i, Y_i)\Big)\Big)\Big) \\ \equiv & \quad \text{(the } X_i s \text{ are mutually disjoint, and } f \text{ does not depend on } X\text{)} \\ & \Big(\exists Y \backslash Y_j : \Big(\bigwedge i : \Big(\exists X_i : f \wedge L_i(X_i, Y_i)\Big)\Big)\Big). \quad \square \end{array}$$

6. TRANSLATING PROOFS

In this section we show how proofs derived using the circular rule **C3** can be translated into proofs using the noncircular rule **NC** and vice versa. We also discuss some of the consequences of these translations. In the sequel, when W is clear from the context, we will write $(\forall W : f)$ simply as f.

THEOREM 6.1 (FROM CIRCULAR TO NONCIRCULAR). *Suppose* $\{f\}M_1//M_2\{\mathsf{G}(g_1 \wedge g_2)\}$ *has been derived from the circular rule* ***C3****. Then* $\{f\}M_1//M_2\{\mathsf{G}(g_1 \wedge g_2)\}$ *may be derived by application of the rule* ***NC*** *by letting the intermediate assertion* h *equal* $f \wedge (g_2 \wedge h_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)$.

PROOF. $\{f\}M_1\{f \wedge (g_2 \wedge h_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)\}$, the first requirement of rule **NC**, follows as a direct result of the first proof obligation from **C3** in the premise.

We now show why the second requirement of rule **NC** also holds.

$$
\begin{array}{ll}
 & \{f \wedge (h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)\} M_2 \{\mathsf{G}(g_1 \wedge g_2)\} \\
\equiv & \quad \text{(by the definition of the assume} - \text{guarantee triple)} \\
 & \{f\} M_2 \{(h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1) \Rightarrow \mathsf{G}(g_1 \wedge g_2)\} \\
\Leftarrow & \quad \text{(by the second proof obligation of rule } \mathbf{C3}\text{)} \\
 & (h_1 \wedge g_1) \rhd ((h_1 \Rightarrow g_2) \wedge h_2) \Rightarrow \\
 & [(h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1) \Rightarrow \mathsf{G}(g_1 \wedge g_2)] \\
\equiv & \quad \text{(rearranging)} \\
 & [(h_1 \wedge g_1) \rhd ((h_1 \Rightarrow g_2) \wedge h_2)] \wedge [(h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)] \Rightarrow \\
 & \mathsf{G}(g_1 \wedge g_2) \\
\equiv & \quad \text{(temporal logic (see proof of Theorem 4.3))} \\
 & \mathit{true}. \quad \square
\end{array}
$$

THEOREM 6.2 (FROM NONCIRCULAR TO CIRCULAR). *Suppose* $\{f\}M_1//M_2\{g\}$ *has been derived from the noncircular rule* ***NC*** *using the intermediate assumption* h. *Then the conclusion* $\{f\}M_1//M_2\{g\}$ *may be derived by application of the rule* ***C3*** *using the substitution* $h_1 = \mathsf{P}h$, $h_2 = \mathit{true}$, $g_1 = \mathit{true}$, *and* $g_2 = \mathsf{P}g$.

PROOF. Firstly, we note that the conclusion of the rule is the desired one. It is straightforward to show that, at the initial point, any computation satisfies $\mathsf{GP}g$ iff it satisfies g. Therefore, $\{f\}M_1//M_2\{\mathsf{G}(g1 \wedge g2)\}$ is equivalent to $\{f\}M_1//M_2\{g\}$. Consider the first proof obligation.

$$
\begin{array}{ll}
 & \{f\} M_1 \{(h_2 \wedge g_2) \rhd ((h_2 \Rightarrow g_1) \wedge h_1)\} \\
\equiv & \quad \text{(substituting and simplifying)} \\
 & \{f\} M_1 \{\mathsf{P}g \rhd \mathsf{P}h\} \\
\Leftarrow & \quad (\ \rhd \text{ is antimonotone in its first argument)} \\
 & \{f\} M_1 \{\mathit{true} \rhd \mathsf{P}h\} \\
\equiv & \quad (\text{as } \mathit{true} \rhd \mathsf{P}h \equiv \mathsf{GP}h) \\
 & \{f\} M_1 \{\mathsf{GP}h\} \\
\equiv & \quad (\text{as } \mathsf{GP}p \text{ and } p \text{ are equivalent at the initial point}) \\
 & \{f\} M_1 \{h\} \\
\equiv & \quad \text{(by the first premise in the supposition)} \\
 & \mathit{true}.
\end{array}
$$

Now we show that the second premise is also true.

$$
\begin{array}{ll}
 & \{f\} M_2 \{(h_1 \wedge g_1) \rhd ((h_1 \Rightarrow g_2) \wedge h_2)\} \\
\equiv & \quad \text{(substituting and simplifying)} \\
 & \{f\} M_2 \{\mathsf{P}(h) \rhd (\mathsf{P}(h) \Rightarrow \mathsf{P}(g))\}.
\end{array}
$$

This follows by an induction on positions of any sequence satisfying $f \wedge \mathcal{L}(M_2)$. At the initial position, by the second part of **NC**, $(h \Rightarrow g)$ holds; thus, $(\mathsf{P}h \Rightarrow \mathsf{P}g)$ holds at the initial position. At any other position i, by the assumption $\mathsf{P}h$ for 0 up to $i-1$, h must be true initially; hence, g is true initially by the second part of **NC**, so that $\mathsf{P}g$ is true at position i. □

We note that the translations make no use of quantified formulae, which justifies the following corollary.

COROLLARY 6.3. *Compositional Rule **C3** is complete for linear temporal logic.*

PROOF. As in the proof of the previous theorem, one can write a property f as $\mathsf{G}(\mathit{true} \wedge \mathsf{P}(f))$ and apply rule **C3**. Since **C3** was shown to be complete for properties of the form $\mathsf{G}(g_1 \wedge g_2)$, it follows that **C3** is sufficient for proving any linear temporal logic property. We note that the proof of Theorem 4.2, presented above, does make use of quantified temporal properties. This use of quantification is beneficial, in that the properties constructed in the proof of completeness may be restricted to the variables which are mentioned in the interface between M_1 and M_2. However, we show below that it is possible to prove Theorem 4.2 without reference to quantified formulae, and hence the result follows. □

ALTERNATIVE PROOF OF THEOREM 4.2. Suppose $\{f\}M_1//M_2\{g\}$ for some linear temporal logic formulae f and g. This can be written equivalently as $\{f\}M_1//M_2\{\mathsf{G}(\text{true} \wedge \mathsf{P}g)\}$.

Let h_1 encode M_1 by writing $h_1 = \mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1)$ and let $g_2 = \mathsf{P}g$. Set $h_2 = \mathit{true}$ and $g_1 = \mathit{true}$. It is required to show that $\{f\}M_1\{(h_2 \wedge g_2) \rhd (h_2 \Rightarrow g_1) \wedge h_1\}$ and $\{f\}M_2\{(h_1 \wedge g_1) \rhd (h_1 \Rightarrow g_2) \wedge h_2\}$.

Suppose $\sigma \in \mathcal{L}_V(M_1)$ and $\sigma \models f$. Applying the substitutions, the requirement becomes $\sigma \models (\mathit{true} \wedge \mathsf{P}g) \rhd (\mathit{true} \Rightarrow \mathit{true}) \wedge \mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1)$ given that σ is in $\mathcal{L}_V(M_1)$. This requirement on σ can be rewritten as $\sigma \models (f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \Rightarrow (\mathit{true} \wedge \mathsf{P}g) \rhd (\mathit{true} \Rightarrow \mathit{true}) \wedge \mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1)$. This can be simplified to *true*.

Suppose $\sigma \in \mathcal{L}_V(M_2)$ and $\sigma \models f$. Applying the substitutions, the requirement becomes

$$\sigma \models (\mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \wedge \mathit{true}) \rhd (\mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \Rightarrow \mathsf{P}g) \wedge \mathit{true}.$$

Putting these facts together we rewrite the requirement as

$$\sigma \models (f \wedge I_2 \wedge \mathsf{G}(T_2) \wedge F_2) \Rightarrow$$
$$(\mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \wedge \mathit{true}) \rhd (\mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \Rightarrow \mathsf{P}g) \wedge \mathit{true}.$$

This simplifies to

$$\sigma \models (f \wedge I_2 \wedge \mathsf{G}(T_2) \wedge F_2) \Rightarrow$$
$$\mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \rhd (\mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \Rightarrow \mathsf{P}g).$$

Note that this last statement requires that $\sigma, 0 \models \mathsf{P}(f \wedge I_1 \wedge \mathsf{G}(T_1) \wedge F_1) \implies \mathsf{P}g$, which means that if σ is also a computation of M_1 then $\sigma \models g$. Since by

assumption if σ is a computation of $M_1//M_2$ then $\sigma \models g$, therefore $\sigma \models \mathsf{P}g$ and therefore this last requirement is satisfied. □

7. RELATED WORK

There are several proposals for compositional reasoning rules in the literature, but only a few investigations of the completeness of these rules—a good survey of the field appears in de Roever et al. [1997]. de Roever et al. [2001] contains a comprehensive discussion of completeness for compositional proof rules dealing with invariance and termination. The seminal work on assertion based noncompositional reasoning for concurrency was by Owicki and Gries [1976] and Lamport [1977]. This was extended to an assume-guarantee reasoning method by Misra and Chandy [1981] and Jones [1981]. These methods apply to invariance and termination properties. Zwiers [1989] contains much of the groundwork necessary for reasoning about compositional proof systems. Pandya [1988] and de Roever et al. [1997] extended the fundamental work by Zwiers [1989] on compositional partial correctness proofs for synchronously communicating processes to proving more general safety properties such as deadlock freedom, and to a restricted class of liveness properties. Proofs of the completeness of compositional reasoning systems for safety properties are found in Zwiers et al. [1984], Pandya [1988], Pandya and Joseph [1991], and de Roever et al. [2001]. Other assume-guarantee rules for safety properties were proposed in Stark [1985], Pnueli [1985], Kurshan [1988], Alur and Henzinger [1996], and McMillan [1997]. More general rules that apply to both safety and liveness properties were proposed in Pnueli [1985], Josko [1987], Clarke et al. [1989], and Grümberg and Long [1994], Abadi and Lamport [1995], and McMillan [1999].

We have concentrated on the completeness question for general rules that apply to both safety and liveness properties. As shown in Section 3, the circular rules in Abadi and Lamport [1995] and the rule **C1** derived from McMillan [1999] are incomplete. The simplicity of the counterexamples suggests that the incompleteness may indeed impact the verification of systems in practice. We presented a new circular rule, which is a modification of rule **C1**, and show it to be sound and complete for all of LTL—in fact, it is straightforward to generalize these ideas so that the rule is complete for the ω-regular languages. The proofs carried out using rule **C1**, including that of the Tomasulo algorithm [McMillan 1998], can be carried out in exactly the same manner with the new rule. We also investigated whether circularity is, in itself, essential for reasoning about composed systems, and showed that, for assume-guarantee reasoning in LTL, the notion of circularity is a somewhat weak one, in that proofs carried out with circular rules are easily translated into proofs with noncircular rules, and vice versa.

There are a number of ways one could choose to strengthen the circular proof rules found in the literature in order to make them complete. We have chosen one in particular, rule **C3**. Our choice was motivated by a desire to remain as close as possible to the spirit of the original circular proof rule—namely, to avoid the "direct" use of $M_1//M_2$ when proving properties about this composition.

Specifically, we have decided not to allow the use of temporal implication, $h \Rightarrow g$, as a proof rule [Manna and Pnueli 1995]. Our results show that implication is not necessary in order to obtain a sound and complete rule. Furthermore, rules that include implication may allow the proof of $f \wedge \mathcal{L}_W(M_1) \wedge \mathcal{L}_W(M_2) \Rightarrow g$ to be instantiated directly as an implication without use of the, hopefully, better rules mentioning only M_1 or M_2. That this goal has been, to some extent, mitigated against in our proof of completeness should not come as a surprise in light of the difficulty of the problem.

This article has concentrated on an assume-guarantee style of compositional reasoning for synchronously communicating processes. Amla et al. [2001] described a related style of assume-guarantee, circular reasoning where it was shown that this style of reasoning fits in well with specifications written as timing-diagrams. Such diagrams are commonly used to specify hardware systems and it was shown that they can offer particular benefits in terms of the complexity of compositional reasoning over the use of LTL. In particular, that work gave polynomial time model checking algorithms for a class of timing diagram specifications. The authors showed how those algorithms could be used to perform compositional reasoning in time linear in the size of a composed system and its composed specification provided that the specification was structured appropriately.

Those authors then extended their work on synchronous systems to asynchronous systems in Amla et al. [2002]. Furthermore, Amla et al. [2002] described a timing diagram-based modular specification language that is as expressive as the ω-regular languages, thus substantially increasing the expressive power of the specification formalism that is applicable with a circular, assume-guarantee style of reasoning. Amla et al. [2003] then showed how, in the context of a more general mathematical framework, many of the existing circular-style compositional reasoning rules could be seen as instances of a common pattern of mutual induction over the computations of several variables.

Several different approaches have recently been developed to incorporate learning-based algorithms into assume-guarantee reasoning frameworks. The key idea is to learn appropriate assumptions needed to perform the component verification tasks. For instance, Chaki and Sinha [2006] have used learning algorithms in the context of compositional reasoning for deadlock detection. Cobleigh et al. [2003] and Barringer et al. [2003] have used learning algorithms to perform assume-guarantee reasoning for safety-property verification. Alur et al. [2005] concentrated on learning algorithms for compositional verification in a symbolic setting. Blundell et al. [2005], have looked at assume-guarantee-based approaches for testing. Related work by Chaki et al. [2005] has looked at automated techniques for reasoning about simulation conformance.

ACKNOWLEDGMENTS

The authors thank Willem-Paul de Roever and the anonymous referees for their many interesting and helpful comments.

REFERENCES

ABADI, M. AND LAMPORT, L. 1995. Conjoining specifications. *ACM Trans. Programm. Lang. Syst. 17*, 3 (May), 507–535.

ALUR, R. AND HENZINGER, T. 1996. Reactive modules. In *Proceedings of IEEE LICS*. 207–218.

ALUR, R., MADHUSUDAN, P., AND NAM, V. 2005. Symbolic compositional verification by learning assumptions. In *Proceedings of the 17th International Conference on Computer Aided Verification* (CAV).

AMLA, N., EMERSON, E. A., NAMJOSHI, K. S., AND TREFLER, R. J. 2001. Assume-guarantee based compositional reasoning for synchronous timing diagrams. In *Proceedings of the 2001 European Joint Conferences on Theory and Practice of Software, Tools and Algorithms for the Construction and Analysis of Systems*.

AMLA, N., EMERSON, E. A., NAMJOSHI, K. S., AND TREFLER, R. J. 2002. Visual specifications for modular reasoning about asynchronous systems. In *Proceedings of the IFIP TC6 WG 6.1 Joint International Conference on Formal Techniques for Networked and Distributed Systems*.

AMLA, N., EMERSON, E. A., NAMJOSHI, K. S., AND TREFLER, R. J. 2003. Abstract patterns of compositional reasoning. In *Proceedings of the 14th International Conference on Concurrency Theory* (CONCUR).

BARRINGER, H., GIANNAKOPOULOU, D., AND PĂSĂREANU, C. 2003. Proof rules for automated compositional verification through learning. In *Proceedings of the Second Workshop on Specification and Verification of Component-Based Systems*.

BLUNDELL, C., GIANNAKOPOULOU, D., AND PĂSĂREANU, C. 2005. Assume-guarantee testing. In *Proceedings of the Workshop on Specification and Verification of Component-Based Systems*.

CHAKI, S., CLARKE, E. M., SINHA, N., AND THATI, P. 2005. Automated assume-guarantee reasoning for simulation conformance. In *Proceedings of the Conference on Computer Aided Verification*. Lecture Notes in Computer Science, vol. 3576. Springer, Berlin, Germany, 534–547.

CHAKI, S. AND SINHA, N. 2006. Assume-guarantee reasoning for deadlock. In *Proceedings of the Conference on Formal Methods in Computer-Aided Design*. 134–144.

CLARKE, E., EMERSON, E., AND SISTLA, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic. *ACM Trans. Program. Lang. Syst. 8,* 2(Apr.), 244–263.

CLARKE, E. AND EMERSON, E. A. 1981. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs*.

CLARKE, E., LONG, D., AND MCMILLAN, K. 1989. Compositional model checking. In *Proceedings of the IEEE LICS*.

COBLEIGH, J. M., GIANNAKOPOULOU, D., AND PĂSĂREANU, C. S. 2003. Learning assumptions for compositional verification. In *Proceedings of the European Joint Conferences on Theory and Practice of Software, Tools and Algorithms for the Construction and Analysis of Systems*.

DE ROEVER, W.-P., DE BOER, F., HANNEMANN, U., HOOMAN, J., LAKHNECH, Y., POEL, M., AND ZWIERS, J. 2001. *Concurrency Verification: Introduction to Compositional and Noncompositional Proof Methods*. Cambridge University Press, Cambridge, U.K.

DE ROEVER, W.-P., LANGMAACK, H., AND PNUELI, A., EDS. 1997. *Compositionality: The Significant Difference*. Lecture Notes in Computer Science, vol. 1536. Springer, Berlin, Germany.

GABBAY, D. M., PNUELI, A., SHELAH, S., AND STAVI, J. 1980. On the temporal basis of fairness. In *Proceedings of POPL*. 163–173.

GRÜMBERG, O. AND LONG, D. 1994. Model checking and modular verification. *ACM Trans. Program. Lang. Syst. 16*, 3(May), 843–871.

HOARE, C. 1969. An axiomatic basis for computer programming. *Commun. ACM 12,* 10(Oct.), 576–580.

JONES, C. 1981. Development methods for computer programs including a notion of interference. Ph.D. dissertation, Oxford University, Oxford, U.K.

JOSKO, B. 1987. Model checking of CTL formulae under liveness assumptions. In *Proceedings of ICALP* 280–289.

KURSHAN, R. 1988. Reducibility in analysis of coordination. In *Discrete Event Systems: Models and Applications*. Lecture Notes in Control and Information Sciences, vol. 103. Springer, Berlin, Germany, 19–39.

LAMPORT, L. 1977. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng. 3,* 2(Mar.), 125–143.

LICHTENSTEIN, O., PNUELI, A., AND ZUCK, L. 1985. The glory of the past. In *Proceedings of the Conference on Logics of Programs*.

MANNA, Z. AND PNUELI, A. 1995. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, Berlin, Germany.

MCMILLAN, K. 1997. A compositional rule for hardware design refinement. In *Proceedings of CAV*.

MCMILLAN, K. 1998. Verification of an implementation of Tomasulo's algorithm by compositional model checking. In *Proceedings of CAV*.

MCMILLAN, K. 1999. Circular compositional reasoning about liveness. In *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. Lecture Notes in Computer Science, vol. 1703. Springer, Berlin, Germany, 342–345.

MISRA, J. AND CHANDY, K. 1981. Proofs of networks of processes. *IEEE Trans. Softw. Eng. 7,* 4 (July), 417–426.

OWICKI, S. S. AND GRIES, D. 1976. Verifying properties of parallel programs: An axiomatic approach. *Commun. ACM 19,* 5(May), 279–285.

PANDYA, P. 1988. Compositional verification of distributed programs. Ph.D. dissertation, University of Bombay, Mumbai, India.

PANDYA, P. AND JOSEPH, M. 1991. P-A logic—a compositional proof system for distributed programs. *Distrib. Comput. 5*, 1, 37–54.

PNUELI, A. 1977. The temporal logic of programs. In *Proceedings of FOCS*.

PNUELI, A. 1985. In transition from global to modular reasoning about programs. In *Logics and Models of Concurrent Systems*. NATO ASI Series. NATO, Brussels, Belgium.

QUEILLE, J. AND SIFAKIS, J. 1982. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*.

STARK, E. 1985. A proof technique for rely/guarantee properties. In *Proceedings of FST&TCS*. 369–391.

THOMAS, W. 1990. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, J. van Leeuwen, ed. Elsevier, Amsterdam, The Netherlands, and MIT Press, Cambridge, MA.

VARDI, M. AND WOLPER, P. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the IEEE Symposium on Logic in Computer Science*.

ZWIERS, J. 1989. *Compositionality, Concurrency and Partial Correctness*.Springer-Verlag, Berlin, Germany.

ZWIERS, J., DE ROEVER, W., AND VAN EMDEBOAS, P. 1984. Compositionality and concurrent networks: Soundness and completeness of a proof system. Tech. rep. University of Nijmegen, Nijmegen, The Netherlands.

Received May 2007; accepted July 2008