Trace-Based Network Proof Systems: Expressiveness and Completeness

JENNIFER WIDOM
IBM Almaden Research Center

and

DAVID GRIES and FRED. B. SCHNEIDER
Cornell University

We consider incomplete trace-based network proof systems for safety properties, identifying extensions that are necessary and sufficient to achieve relative completeness. We investigate the expressiveness required of any trace logic to encode these extensions.

Categories and Subject Descriptors: D 1 3 [**Programming Techniques**]: Concurrent Programming; D.2.4 [**Software Engineering**]: Program Verification—*correctness proofs*; D.3 1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; F.4 1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*proof theory*

General Terms: Verification

Additional Key Words and Phrases: Process networks, safety properties, temporal logics, trace logics

1. INTRODUCTION

In *trace-based network proof systems*, one specifies and reasons about traces (histories) of the values transmitted along the communication channels of the networks under consideration. Such proof systems generally allow specifications for networks to be deduced from specifications for the networks'

This paper is based on portions of J. Widom's Cornell University Ph D. thesis. The work was supported in part by the National Science Foundation under grants DCR-8320274 and CCR-8701103. F. B. Schneider is also supported by the Office of Naval Research under contract N00014-86-K-0092 and by Digital Equipment Corporation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not reflect the views of these agencies.
Authors' addresses: J Widom, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, D Gries and F B. Schneider, Computer Science Department, Cornell University, Ithaca, NY 14853

components. Network proof systems based on first-order predicates over channel traces are given in [4], [7], and [12], but unfortunately, these logics are known to be incomplete [2, 8, 13]. Relative completeness can be achieved by permitting reasoning over the full interleaving of communication events in addition to the individual channel traces [3, 6, 9] or by using temporal logic formulas so that the interleaving is implicitly present in the semantics of the specifications [14]. Both approaches introduce more information than necessary, the modifications tend to be extensive and cumbersome, and the simplicity of the underlying trace logic is lost.

In this paper we analyze the sources of incompleteness in trace-based network proof systems and formalize the extensions needed to achieve relative completeness. A simple but logically incomplete proof system for safety properties is defined. It is representative of other incomplete trace-based network proof systems appearing in the literature. Two example networks are presented to show incompleteness of our logic. Surprisingly, both examples consist of only one process, indicating that, although compositionality is an important feature of trace-based systems, composition is not the cause of incompleteness. Our examples suggest two extensions that are necessary for relative completeness; we show that these two extensions are sufficient as well. The first source of incompleteness is the inability to state and reason about constraints on the ordering of network events. The second source is the inability to assert that the sequence of values transmitted along a communication channel is a prefix of that channels's sequence at each later point. We argue that these two properties—the temporal ordering and prefix properties-must be part of any relatively complete trace-based proof system.

The temporal ordering and prefix properties cannot be expressed in a logic based on first-order predicates over channel traces. However, they can be expressed using only a subset of temporal logic, indicating that the full power of temporal logic (or of explicit reasoning over the interleaving of communication events) is not needed. We introduce a hierarchy of subsets of temporal logic and show that a subset consisting of first-order predicates over traces with a version of the linear-time temporal *Always* operator has necessary and sufficient expressive power for relative completeness.

Section 2 defines the class of synchronous process networks we consider and introduces a formal model of computation. Section 3 defines Simple Network Logic (SNL), a trace-based network proof system that captures the essence of most such systems. In Section 4 we show that SNL is incomplete, define the temporal ordering and prefix axioms, and prove that they are necessary and sufficient for relative completeness. In Section 5 we define a hierarchy of temporal logic subsets and identify a subset of temporal logic with exactly the right expressive power for relative completeness. Finally, in Section 6 we summarize, discuss alternatives and extensions to our work, and relate our work to previous research.

2. PROCESS NETWORKS

Consider networks of processes that communicate and synchronize solely by message passing. Processes and communication channels are uniquely named.

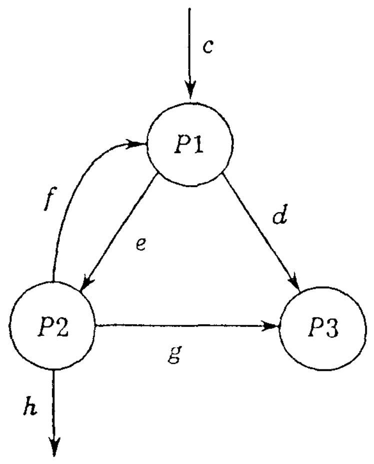

Fig. 1. Network of processes

Each channel is either *internal* or *external* with respect to a network. An internal channel connects two processes of the network; an external channel is connected to only one process, permitting communication between the network and its environment. Channels are unidirectional, and communication along then is synchronous,[1] so both processes incident to an internal channel must be prepared to communicate before a value is actually transmitted. Without loss of generality, we assume that

—input or output on an external channel occurs whenever the single incident process is ready;

—each message transmission occurs instantaneously;

—two message transmissions cannot occur simultaneously, thus there is a total order on the communication events of a given computation; and

—there is a fixed domain of values that can be transmitted on communication channels, and processes send and receive values in this domain only.

A network made up of processes $P_1, P_2, \ldots, P_n$ is denoted by $P_1 \| P_2 \| \cdots \| P_n$ indicating parallel execution of $P_1, P_2, \ldots, P_n$. Figure 1 illustrates a network of three processes and six communication channels. Subsequently, we use the term *network* to refer to either a single process or a network composed of several processes.[2]

2.1 Model of Computation

To reason about proof systems for networks, we introduce a formal model of network behavior. A single point, or state, during the computation of a

[1]Extension to asynchronous message passing is straightforward and does not affect our results [16].

[2]Although we assume here that networks are composed directly of primitive processes, our work easily generalizes to a hierarchical structure in which processes may be implemented by subnetworks [16].

network is represented by the histories of the network's communication channels up to that point. (The model does not include internal process state, since internal computations of processes are unimportant when reasoning about network behavior, except as they affect values sent and received.) A single computation is represented by an infinite sequence of states,[3] subject to constraints on the initial state and between each pair of adjacent states. The behavior of a network is represented by the set of all sequences representing its possible computations.

We now formalize the model. A *trace* of a communication channel c is a finite sequence $\langle c.1, c.2, \ldots, c.k \rangle$ of values that have been transmitted along channel c up to some point in time. An empty trace is denoted by $\langle\ \rangle$. Let N be a network with incident channels $c_1, c_2, \ldots, c_m$. A *state* of N is a tuple containing a trace for each channel $c_1, c_2, \ldots, c_m$. A *computation* of N is an infinite sequence of states of N such that all channel traces are empty in the initial state and each subsequent state extends at most one trace of the preceding state by at most one element (i.e., at most one message transmission occurs between each pair of states). Thus, an infinite sequence of states is a computation iff the following four computation conditions are satisfied:

(CC1) All traces are empty in the initial state.

(CC2) Each trace in each state of the sequence is a prefix of the corresponding trace in the subsequent state.

(CC3) The length of each trace in each state of the sequence (except the first) is at most one more than the length of the corresponding trace in the preceding state.

(CC4) At most one trace changes between every state and its subsequent state.

The *computation set* for N is the set of all computations representing potential execution sequences of N.[4]

We show that our computational model is compositional; that is, the set representing the behavior of a network can be constructed from the sets representing the behavior of the network's component processes. Let N be a network composed of $P_1, \ldots, P_n$. Let $CS(P_1), \ldots, CS(P_n)$ denote the computation sets for $P_1, \ldots, P_n$, respectively. $CS(N)$, the computation set for N, can be computed directly from $CS(P_1), \ldots, CS(P_n)$ as follows: Suppose κ is a computation of N. Define $Proj(\kappa, P_\iota)$ to be the projection of κ onto those channels of N incident to P_ι; that is, $Proj(\kappa, P_\iota)$ removes from the states of κ all traces of channels in N not incident to P_ι. $CS(N)$ is the set of all

[3]We choose only infinite sequences since, for our purposes, any finite sequence can be converted into an equivalent infinite sequence by indefinitely repeating the final state [10, 16].

[4]Our definition allows arbitrarily many (adjacent) repetitions of any state in a computation sequence, producing a very large computation set for a network. In particular, if a given computation is in a computation set, then so is that computation with any state repeated once, repeated twice, repeated three times, etc. Permitting this repetition facilitates subsequent definitions and proofs without affecting the usefulness of the model.

computations κ built from possible states of N such that

—$Proj(\kappa, P_i) \in CS(P_i)$, $i = 1 \ldots n$, and
—κ satisfies conditions (CC1)–(CC4) above.

Informally, then, the computations of N are all possible well-formed combinations of the computations of N's component processes in which communications on shared channels agree.

3. SIMPLE NETWORK LOGIC

We now introduce a formalism, *Simple Network Logic* (*SNL*), for specifying and verifying safety properties [9] of networks of processes. SNL concisely captures the essence of most trace-based proof systems.

A *specification* is a first-order predicate over channel traces; it is intended to be satisfied throughout every execution of the network it specifies [4, 7, 12].[5] A channel name c appearing as a free variable in a specification represents the trace of c. We use $|c|$ to denote the length of c, and $c1 \sqsubseteq c2$ to denote that trace $c1$ is a prefix of trace $c2$.

A specification for a network N is a predicate ϕ over the traces of N's communication channels. (The only free variables permitted in specifications for N are those corresponding to channels of N.) We say that N *satisfies* ϕ, written N **sat** ϕ, if, at every point during any computation of N, the traces of values transmitted on N's channels satisfy ϕ. For example, consider a process (or network) N that repeatedly reads an integer from channel $c1$ and sends its successor on channel $c2$. This behavior can be specified in SNL as

$$N\ \mathbf{sat}\ \big(|c1| - 1 \le |c2| \le |c1|\big) \wedge \big(\forall i\colon 1 \le i \le |c2| : c2.i = c1.i + 1\big).$$

Some network proof systems provide facilities for constructing specifications for sequential primitive processes [4, 7, 19]. In others, it is assumed that existing logics for sequential programs are used for this, or that specifications for primitive processes are given [8, 12, 14]. Without loss of generality, we adopt the latter approach. Thus, we assume that some programming language and proof system are used initially to construct primitive processes (following our communication model) and to verify trace logic specifications for the behavior of each. We are then interested only in deducing specifications for a network from specifications for its component processes. Hence, let the axioms of SNL be formulas P **sat** ϕ, where P is a primitive process and ϕ is a valid specification for P; that is, ϕ is satisfied by every execution of P. We assume that included is at least one such axiom (but not necessarily all) for each process P of interest.

[5]Actually, in [12], specifications consist of predicate pairs, but for our purposes it is adequate to consider only single predicates that remain invariant throughout a computation [16].

Specifications for networks can be derived from specifications for their component processes using the following inference rule of SNL:

Definition 3.1. *Network Composition Rule*

$$\frac{P_1 \textbf{ sat } \phi_1, P_2 \textbf{ sat } \phi_2, \ldots, P_n \textbf{ sat } \phi_n}{P_1 \| P_2 \| \cdots \| P_n \textbf{ sat } \bigwedge_i \phi_i}$$

Conjoining process specifications using this rule results in "linking" any shared channels in network N, because in $\wedge_i \phi_i$ all c's, say, denote the same channel trace.

We also need a rule for deducing valid specifications for a network from other valid specifications, since several valid specifications may exist for a given network. For this, we rely on the validity of formulas in our underlying trace logic. If ϕ is a specification over channel traces and S is a state (a mapping from channel variables to trace values), let $S \models \phi$ denote that formula ϕ is satisfied by state S. If ϕ is satisfied by every state, we write $\models \phi$. The SNL Consequence Rule is then as follows:

Definition 3.2. *Consequence Rule*

$$\frac{N \textbf{ sat} \phi_1, \models \phi_1 \Rightarrow \phi_2}{N \textbf{ sat } \phi_2}$$

The use of $\models \phi_1 \Rightarrow \phi_2$ in this rule, expressing that every state that satisfies ϕ_1 also satisfies ϕ_2, is further discussed below.

These two rules, or variants thereof, form the basis of most trace-based systems we know of, including those in [4], [7], [12], [14], and [18].

3.1 Soundness and Completeness

We can use the computational model introduced in Section 2.1 to prove that SNL is *sound* [1]: If N **sat** ϕ is a theorem of SNL, then ϕ is indeed valid for network N. To do so, we must first define validity of specifications with respect to computations in the formal model. Recall that for a network N, $CS(N)$ denotes the set of all computations corresponding to N's possible behaviors. A specification ϕ is valid for N iff ϕ is true in every state of every computation in N's computation set.

Definition 3.3. *Validity of Specifications.* Specification ϕ is valid for network N iff $\kappa.i \models \phi$ for all $\kappa \in CS(N)$ and $i \geq 0$, where $\kappa = \langle \kappa.0, \kappa.1, \kappa.2, \ldots \rangle$.[6]

Using this definition we establish the soundness of SNL.

THEOREM 3.4. Soundness of SNL. *Let N be a network and ϕ a specification such that N* **sat** *ϕ is a theorem of SNL. Then ϕ is valid for N.*

[6]Note that indexing of computations begins with 0, while indexing of channel traces begins with 1. This notation facilitates subsequent definitions and proofs.

PROOF. See the Appendix. □

We would also like SNL to be *complete*: If any specification ϕ is valid for a network N, then N **sat** ϕ is provable in SNL. However, an additional assumption must be made. A specification for a network is derived from specifications for its component processes using the Network Composition Rule. If the given process specifications are valid but too weak, then a valid network specification might not be provable. What we really want to know is whether N **sat** ϕ can be proved when the given specifications for N's component processes are as "strong" as possible [14].

Definition 3.5. *Precise Specifications.* A specification ϕ is *precise* for a network N iff

(1) ϕ is valid for N; and

(2) if κ is any computation[7] containing traces for the channels in N and $\kappa.j \models \phi$ for all $j \geq 0$, then $\kappa \in CS(N)$.

Thus, a specification is precise if it is valid and if every computation satisfying the specification is a possible computation of the network being specified. For completeness, we are interested in proving N **sat** ϕ whenever ϕ is valid and whenever the specifications for the processes in N are precise.

The SNL Consequence Rule relies on the validity of formulas in the specification language of SNL. SNL specifications can involve elements of the data domain from which messages are drawn, sequences of such elements (the channel traces), and lengths of sequences. Since arithmetic itself is incomplete [15], when designing a program proof system one aims for *relative completeness* (as in [5]): Assuming that one can prove any valid formula of the underlying logic—which in this case is a trace logic that includes predicate logic, arithmetic, and the data domain of the network being considered—is the proof system complete?[8] SNL is not relatively complete, as we now show.

4. INCOMPLETENESS OF SIMPLE NETWORK LOGIC

Two examples are given to show the incompleteness of SNL. Each example illustrates an inherent property of network behavior that cannot be expressed in SNL, but that is necessary for relative completeness in a trace-based proof system.

4.1 Temporal Ordering Property

Consider the single-process network of Figure 2. As an informal description of process P, we are given four facts: (1) P reads at most one value from

[7]Recall from Section 2.1 that a computation is an infinite sequence of states such that the initial state contains only empty channel traces and each subsequent state extends at most one trace of the preceding state by at most one element.

[8]Most proof systems make assumptions about both the provability of statements in the underlying logic and the expressiveness of the specification language involved. This is sometimes referred to as *Cook completeness* [1, 5]. We, too, have made an expressiveness assumption in our supposition that precise specifications for primitive processes can be written in SNL. See Section 6 for further discussion.

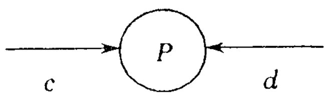

Fig. 2. Simple network.

channel c, (2) P reads at most one value from channel d, (3) P reads a value from c before reading from d, and (4) P reads a value from d before reading from c. By direct translation, a formal specification is

$$P \textbf{ sat } \phi_1\text{: } |c| \leq 1 \wedge |d| \leq 1 \wedge |d| \leq |c| \wedge |c| \leq |d|. \tag{1}$$

Let the data domain for this network be $\{a\}$. The following specification is valid for P and is equivalent to (1):

$$P \textbf{ sat } \phi_2\text{: } (c = \langle\ \rangle \wedge d = \langle\ \rangle) \wedge (c = \langle a \rangle \wedge d = \langle a \rangle). \tag{2}$$

P is always in one of two states: Either no values have been read from c or d, or a value a has been read from each. However, since two values cannot be transmitted simultaneously (condition (CC4) from Section 2.1), P can reach a state in which $(c = \langle a \rangle \wedge d = \langle a \rangle)$ only by being in a state in which either $(c = \langle a \rangle \wedge d = \langle\ \rangle)$ or $(c = \langle\ \rangle \wedge d = \langle a \rangle)$, neither of which is permitted by specification (2) (or specification (1)). Thus, P will never read a value from either c or d, so a third valid and equivalent specification for P is

$$P \textbf{ sat } \phi_3\text{: } c = \langle\ \rangle \wedge d = \langle\ \rangle. \tag{3}$$

All three specifications are valid and, in fact, precise. Any computation satisfying ϕ_1, ϕ_2, or ϕ_3 is a computation of P: no values are ever read on c or d. However, consider an attempt at proving (3) given, say, precise specification ϕ_2 of (2). Since there is only a single process, the Network Composition Rule is irrelevant; the only SNL inference rule applicable is the Consequence Rule. But $\models \phi_2 \Rightarrow \phi_3$ does not hold. Hence, (3) is unprovable from (2), even though it is valid.

We need a way to formalize the reasoning about event ordering used to obtain specification (3). It must assert the following property:

Definition 4.1. *Temporal Ordering Property.* Suppose $c1$ and $c2$ are channels of a network N, $c1.x$ and $c2.y$ are transmitted as a result of distinct communication events, and in any computation of N

(1) $c1.x$ is transmitted before $c2.y$, and

(2) $c2.y$ is transmitted before $c1.x$.

Then $(|c1| < x \wedge |c2| < y)$ holds throughout any computation of N, i.e., neither message is ever transmitted.

Formalizing this property would allow ϕ_3 to be deduced from ϕ_2, making (3) provable from (2). Unfortunately, the Temporal Ordering Property cannot be expressed in the trace logic underlying SNL. This is discussed further in Section 4.4.1 and is proved in Section 5.

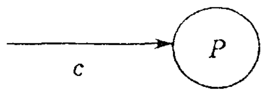

Fig. 3. Very simple network.

4.2 Prefix Property

Consider the network of Figure 3. Suppose $\{a, b\}$ is the data domain, and let a precise specification for process P be

$$P \textbf{ sat } \phi_4 \colon c = \langle\ \rangle \vee c = \langle a \rangle \vee c = \langle b, a \rangle. \tag{4}$$

Since P sends one value at a time on channel c (condition (CC3) from Section 2.1), disjunct $c = \langle b, a \rangle$ can never hold. (It would hold only if $c = \langle b \rangle$ held first, and this is prohibited by ϕ_4.) Therefore, (4) can be simplified to

$$P \textbf{ sat } \phi_5 \colon c \sqsubseteq \langle a \rangle. \tag{5}$$

However, ϕ_4 does not imply ϕ_5, so (5) cannot be proved from precise specification (4). Here we need the following property:

Definition 4.2. *Prefix Property.* For any channel c and integers $0 \le x \le y$, the trace of c after x values have been transmitted is always a prefix of the trace of c after y values have been transmitted.

By using this property in conjunction with ϕ_4, we could eliminate disjunct $c = \langle b, a \rangle$ and obtain (5). Like the Temporal Ordering Property, however, the Prefix Property cannot be expressed in the trace logic underlying SNL.

4.3 Incorporating the Properties in SNL

Consider any SNL proof that establishes N **sat** ϕ for a network $N = P_1 \| \cdots \| P_n$. We assume that as axioms we are given P_1 **sat** $\phi_1, \ldots, P_n$ **sat** ϕ_n, where $\phi_1, \ldots, \phi_n$ are precise. The first rule to be applied in any such proof is the Network Composition Rule, so we immediately infer N **sat** $\wedge_i \phi_i$. All remaining steps in the proof have to be applications of the Consequence Rule. By transitivity of implication, any string of Consequence Rule applications can be collapsed into one, so N **sat** ϕ is provable iff $\models \wedge_i \phi_i \Rightarrow \phi$.

By the following theorem, we know that specification $\wedge_i \phi_i$ is precise for N: the conjunction of precise process specifications results in a network specification that is also precise. (A similar theorem for temporal logic is proved in [14].)

THEOREM 4.3. Preciseness Preservation. *Let ϕ_i be a precise specification for P_i, $1 \le i \le n$, and let $N = P_i \| \cdots \| P_n$. Then $\bigwedge_i \phi_i$ is a precise specification for N.*

PROOF. See the Appendix. □

Thus, our proof system would be relatively complete if $\models \phi_1 \Rightarrow \phi_2$ whenever ϕ_1 is a precise specification for a network N and ϕ_2 is a valid specification for N. However, the examples given in Sections 4.1 and 4.2 illustrate that this implication does not always hold.

To obtain a relatively complete system, the implication in the hypothesis of the Consequence Rule must be modified so that all valid specifications can be deduced from precise specifications. We do this by strengthening the antecedent of the implication, adding a set of axioms such that if, say, A is the conjunction of axioms in the set, then $\models (\phi_1 \wedge A) \Rightarrow \phi_2$ whenever ϕ_1 and ϕ_2 are precise and valid, respectively, for a given network. The Temporal Ordering and Prefix Properties are the basis for such a set of axioms.

4.4 Temporal Ordering and Prefix Axioms

We now prove that axiomatizations of the Temporal Ordering and Prefix Properties are necessary and sufficient for deducing ϕ_2 from ϕ_1 whenever ϕ_1 and ϕ_2 are precise and valid, respectively, for a given network. There is a fundamental difference, however, between any axiomatization of the Temporal Ordering (or Prefix) Property and specifications ϕ_1 and ϕ_2: Event ordering is considered with respect to an entire computation, whereas ϕ_1 and ϕ_2 are considered with respect to the individual states of a computation. Since $(\phi_1 \wedge A) \Rightarrow \phi_2$ must be considered with respect to computations, we resolve the discrepancy by introducing an operator $\Box$, which converts single-state specifications to operate over sequences of states: $\Box\phi$ is valid for a sequence of states κ, denoted by $\kappa \models \Box\phi$, iff ϕ is true in every state of the sequence.[9] That is,

$$\kappa \models \Box\phi \qquad \text{iff} \quad \kappa.i \models \phi \text{ for all } i \geq 0.$$

(Note that by Definition 3.3 of validity of specifications, specification ϕ is thus valid for a network N iff $\kappa \models \Box\phi$ for every κ in the computation set of N.) Using $\Box$ and $\models$, we modify the Consequence Rule as follows:

Definition 4.4. *Modified Consequence Rule*

$$\frac{N \textbf{ sat } \phi_1,\ \models (\Box\phi_1 \wedge A) \Rightarrow \Box\phi_2}{N \textbf{ sat } \phi_2}$$

Notice that this rule requires that the underlying logic is extended to include the operator $\Box$ and that formula $(\Box\phi_1 \wedge A) \Rightarrow \Box\phi_2$ is interpreted over sequences of states rather than over single states. That is, $\models (\Box\phi_1 \wedge A) \Rightarrow \Box\phi_2$ holds iff every sequence of states satisfying $(\Box\phi_1 \wedge A)$ also satisfies $\Box\phi_2$. Next, we consider the axioms comprising A.

4.4.1 *Temporal Ordering Axiom.* The Temporal Ordering Axiom will formalize the Temporal Ordering Property. Suppose some communication $c1.x$ must happen before some $c2.y$. Then $\Box(|c1| < x \Rightarrow |c2| < y)$. This assertion captures ordering of communication events for any channels $c1$ and $c2$ and any indexes x and y, even if $x = y$ or $c1$ and $c2$ are the same channel. We are interested only in the ordering of distinct events, so the case in which

[9]This is a weakened version of the "always," or "henceforth," operator (also $\Box$) of temporal logic [10, 11], since here ϕ cannot contain other temporal operators. Temporal operators of varying strengths are discussed in Sections 4.4.2 and 5.

$c1.x$ and $c2.y$ are produced by the same event (i.e., $x = y$ and $c1$ and $c2$ are the same channel) is not of interest. Now, if $\Box(|c2| < y \Rightarrow < y \Rightarrow |c1| < x)$ holds as well (with $x \neq y$ or $c1$ and $c2$ distinct), then neither $c1.x$ nor $c2.y$ can ever occur; equivalently, $\Box(|c1| < x \wedge |c2| < y)$. Hence, we state the Temporal Ordering Axiom as follows:

Definition 4.5. *ORDERING.* If $c1$ and $c2$ are channels, $x \geq 1$ and $y \geq 0$ are integers, and either $x \neq y$ or $c1$ and $c2$ are distinct; then[10]

$$\Box(|c1| < x \Leftrightarrow |c2| < y) \Rightarrow \Box(|c1| < x \wedge |c2| < y).$$

Allowing $y = 0$ permits the assertion that an empty channel trace temporally precedes all communications events on that channel.[11] We disallow $x = y = 0$, however, since this results in a pathological situation in which the antecedent of the implication is trivially true but the consequent is trivially false.

We can now prove soundness of *ORDERING* with respect to our computational model.

THEOREM 4.6. Soundness of *ORDERING*. *If κ is a computation (recall Section 2.1) then $\kappa \models$ ORDERING.*

PROOF. See the Appendix. $\Box$

4.4.2 *Prefix Axiom.* To formulate an axiom for the Prefix Property, we introduce a more powerful version of $\Box$ in which $\Box$ may be applied to formulas that themselves contain $\Box$'s. (This is the usual linear-time temporal logic interpretation for $\Box$ [11].) Now, $\Box\phi$ is valid for a sequence of states iff ϕ is valid for every suffix of that sequence:

$$\kappa \models \Box\phi \quad \text{iff} \quad \langle \kappa.i, \kappa.(i+1), \kappa.(i+2), \ldots \rangle \models \phi \text{ for all } i \geq 0.$$

When ϕ contains no $\Box$ operators, $\langle \kappa.i, \kappa.(i+1), \kappa.(i+2), \ldots \rangle \models \phi$ is usually interpreted to be true iff ϕ is true in the first state, that is, iff $\kappa.i \vDash \phi$. (For more detailed and rigorous discussions of the semantics of temporal operators, see, e.g., [11] and [16].)

The Prefix Axiom can be stated using $\Box$ as follows:

Definition 4.7. *PREFIX.* If c is a channel, $x \geq 1$ is an integer, and v is a value in the data domain of messages, then

$$\Box(c.x = v \Rightarrow \Box(c.x = v)).$$

[10]Technically, this is an axiom scheme rather than an axiom, since substitution for metasymbols $c1$, $c2$, x, and y is permitted.

[11]Suppose, for the sake of contradiction, that c is nonempty in the initial state of some computation satisfying *ORDERING*. Then $\Box(|c| < x \Leftrightarrow |c| < 0)$ for some $x \geq 1$. However, $\Box(|c| \geq 0)$, so the conclusion of *ORDERING* does not hold.

This axiom (scheme) asserts that, once a value has been transmitted as $c.x$, $c.x$ remains unchanged. This is equivalent to the Prefix Property as stated in Section 4.2.[12]

THEOREM 4.8. Soundness of *PREFIX*. *If κ is a computation, then $\kappa \models$ PREFIX.*

PROOF. Let κ be any computation. Then κ satisfies condition (CC2) (Section 2.1), and *PREFIX* follows directly. $\Box$

4.4.3 *Necessity and Sufficiency of ORDERING and PREFIX.* By letting A be $ORDERING \wedge PREFIX$, we can prove that, if ϕ_1 is a precise specification for a network N and ϕ_2 is a valid specification for N, then $\models (\Box \phi_1 \wedge A) \Rightarrow \phi_2$ (from the hypothesis of Modified Consequence Rule 4.4) holds. Thus, *ORDERING* and *PREFIX* are sufficient for achieving relative completeness. In addition, we will argue that *ORDERING* and *PREFIX* are necessary: If either axiom is removed from A, then there is a network N with precise and valid specifications ϕ_1 and ϕ_2, respectively, such that $\Box \phi_1$ and A do not imply $\Box \phi_2$.

We begin with a key lemma.

LEMMA 4.9. *Let κ be any infinite sequence of states. κ represents a computation iff $\kappa \models ORDERING \wedge PREFIX$.*

PROOF. See the Appendix. $\Box$

With this lemma in hand, we can easily prove that our two axioms are sufficient for relative completeness.

THEOREM 4.10. Sufficiency of the Axioms. *If ϕ_1 is a precise specification for network N and ϕ_2 is a valid specification for N, then $\models (\Box \phi_1 \wedge ORDERING \wedge PREFIX) \Rightarrow \Box \phi_2$.*

PROOF. We show that any infinite sequence of states κ satisfying $\Box \phi_1$, *ORDERING*, and *PREFIX* also satisfies $\Box \phi_2$. Since $\kappa \models ORDERING \wedge PREFIX$, by Lemma 4.9 we know that κ is a computation. By Definition 3.5 of preciseness, since $\kappa \models \Box \phi_1$ and ϕ_1 is precise, $\kappa \in CS(N)$. By validity of ϕ_2, every $\kappa \in CS(N)$ satisfies $\Box \phi_2$. Hence, κ satisfies $\Box \phi_2$. $\Box$

Thus, with *ORDERING* and *PREFIX*, we ensure that any valid network specification is implied by a precise specification for the network; by Preciseness-Preservation Theorem 4.3, a precise network specification is obtainable from precise specifications for the network's component process. That

[12]A different axiomatization of the Prefix Property can be given using the "next" operator of temporal logic in addition to $\Box$ [17]. The definition given here, however, shows that the Prefix Property can be encoded using only $\Box$. This is of importance in Section 5, where we consider the minimal expressiveness required of any trace logic used as the basis of a relatively complete proof system.

ORDERING and *PREFIX* are necessary (as well as sufficient) for the implication to always hold is shown in the following theorem:

THEOREM 4.11. Necessity of the Axioms. *There exist networks $N1$, $N2$, and $N3$, with precise specifications ϕ_1^P, ϕ_2^P, and ϕ_3^P, respectively, and valid specifications ϕ_1^V, ϕ_2^V, and ϕ_3^V, respectively, such that*

(1) $\not\models (\Box \phi_1^P \wedge \mathit{ORDERING}) \Rightarrow \Box \phi_1^V)$

(2) $\not\models (\Box \phi_2^P \wedge \mathit{PREFIX} \Rightarrow \Box \phi_2^V)$, *and*

(3) $\not\models (\Box \phi_3^P \Rightarrow \Box \phi_3^V)$.

PROOF. For (1), let $N1$ be the example network of Section 4.2. For (2), let $N2$ be the example network of Section 4.1. Fact (3) follows directly from (1) and (2). □

5. STRENGTHENING THE PROOF SYSTEM

We have demonstrated that an axiomatization of the Temporal Ordering and Prefix Properties is necessary and sufficient for relative completeness of SNL. However, *ORDERING* ∧ *PREFIX* is not the only way to formalize the properties of computation that must be encoded in the logic. From the proof of Theorem 4.10, we see that the function of *ORDERING* and *PREFIX* is to characterize legal network computations. Thus, we are interested in the expressiveness required of a logic in order to encode the notion of a legal network computation.

We formalize this expressiveness requirement by analyzing the relationship between valid and precise specifications. By Definition 3.3 of validity, a specification is valid for a network N iff it is satisfied by every state reachable by N. By Definition 3.5 of preciseness, a state is reachable by N iff it is reachable by a computation that always satisfies a precise specification for N. Hence, we observe the following:

Observation 5.1. A specification is valid for a network iff it is satisfied by every state reachable by a computation that always satisfies a precise specification for that network.

Formalizing this observation results in a relatively complete proof system. One such formalization is implication $(\Box \phi_1 \wedge A) \Rightarrow \Box \phi_2$ of Modified Consequence Rule 4.4. We establish bounds on the expressive power required of any trace logic that formalizes Observation 5.1.

Let ϕ range over trace logic formulas (i.e., over formulas in the specification language of SNL). Suppose $K(\phi)$ is a formula in some logic L such that, for any ϕ, a state satisfies $K(\phi)$ iff the state is reachable by a computation that always satisfies ϕ. Consider the following Generalized Consequence Rule, where $K(\phi_1) \Rightarrow \phi_2$ is a formula of logic L:

Definition 5.2. *Generalized Consequence Rule*

$$\frac{N \textbf{ sat } \phi_1,\ \models K(\phi_1) \Rightarrow \phi_2}{N \textbf{ sat } \phi_2}$$

(Note that $K(\phi_1) \Rightarrow \phi_2$ is being interpreted over states, not over state sequences.) By Observation 5.1 and the definition of $K(\phi)$, if ϕ_1 and ϕ_2 are precise and valid specifications, respectively, for N, then $\models K(\phi_1) \Rightarrow \phi_2$. Therefore, incorporating the Generalized Consequence Rule yields a proof system that is complete relative to logic L. Our goal is to isolate the power required to express $K(\phi)$, a formula satisfied by exactly those states reachable by a computation that always satisfies ϕ. Here we provide an outline of our expressiveness results; the reader is referred to [16] for technical details.

First, we observe that formula $K(\phi)$ can be expressed in an extended trace logic in which some explicit reasoning over computations is permitted. Suppose the communication channels of all networks under consideration are $c_1, c_2, \ldots, c_m$, and let the data domain of transmittable values be a set V. A state can be represented by a tuple $\bar{t} = [t_1, \ldots, t_m]$ of channel traces, where each t_i is the trace of c_i. A computation up to some point in time is represented by a finite sequence $\langle \overline{t^0}, \overline{t^1}, \ldots, \overline{t^n} \rangle$ of such tuples. (Since we are interested only in states reachable by computations, we need not consider the infinite sequences representing full computations.) $K(\phi)$ can be expressed in a logic that allows quantification over such sequences. Let

—$\phi[\bar{c}/\overline{t^i}]$ denote SNL specification ϕ with channel trace variables $c_1, \ldots, c_m$ replaced by traces $t_1^i, \ldots, t_m^i$; and

—$\overline{t^i}[t_j^i / t_j^i \cdot \langle v \rangle]$ denote tuple $\overline{t^i}$ with trace t_j^i extended by value $v \in V$.

We then define $K(\phi)$ as follows:

Definitions 5.3. *Formula $K(\phi)$ in Extended Trace Logic*

$K_{\mathrm{ETL}}(\phi) \equiv$

$(\exists \langle \overline{t^0}, \overline{t^1}, \ldots, \overline{t^n} \rangle$:	*There exists a sequence of states*
$\overline{t^0} = [\langle\ \rangle, \ldots, \langle\ \rangle] \wedge$	*such that: in the first state*
$\overline{t^n} = [c_1, \ldots, c_k] \wedge$	*all traces are empty, ϕ is*
$(\forall i: 0 \le i \le n: \phi[\bar{c}/\overline{t^i}]) \wedge$	*satisfied in every state, and in*
$(\forall i: 0 \le i < n$:	*every pair of adjacent states:*
$\overline{t^{i+1}} = \overline{t^i} \vee$	*either the states are identical or the*
$(\exists j, v: 1 \le j \le m, v \in V$:	*second state extends exactly one trace*
$\overline{t^{i+1}} = \overline{t^i}[t_j^i / t_j^i \cdot \langle v \rangle])))$	*of the first state by exactly one element.*

The free variables of $K_{\mathrm{ETL}}(\phi)$ are channel trace variables $c_1, \ldots, c_m$. As illustrated by the annotation, $K_{\mathrm{ETL}}(\phi)$ is satisfied by exactly those states reachable by a computation that always satisfies ϕ. Therefore, $K_{\mathrm{ETL}}(\phi)$ could be used in the Generalized Consequence Rule to obtain relative completeness.

Linear-Time Temporal Logic (TL) can be used for similar but implicit reasoning over sequences of states [11] and is generally considered more convenient for such reasoning. As indicated by the definitions of *ORDERING* and *PREFIX* given in Sections 4.4.1 and 4.4.2, however, full temporal logic and *PREFIX* given in Sections 4.4.1 and 4.4.2, however, full temporal logic is more powerful than is needed to express $K(\phi)$. By considering equivalence of TL formulas and formula $K_{\mathrm{ETL}}(\phi)$, we isolate a subset of TL that is necessary and sufficient to express $K(\phi)$.

We begin with a version of TL that extends the trace logic of SNL with three standard temporal operators: the *Always* operator $\Box$, the *Next* operator $\bigcirc$, and the *Until* operator $\mathcal{U}$. We omit operator $\Diamond$ since it is the dual of $\Box$. (For definition and further discussion of these temporal operators, see, e.g., [11] and [16].) Our original definition of $\Box$, in Section 4.4, is for trace logic formulas that do not contain temporal operators. In Section 4.4.2 we considered a version for formulas containing other $\Box$s. The weaker version of $\Box$ is used to define the Temporal Ordering Axiom, while the stronger version is needed to define the Prefix Axiom. In general, allowing nested temporal operators yields significantly more expressive power than restricting temporal operators to operate over nontemporal formulas.[13] Hence, we also consider use of an additional set of temporal operators that operate over trace logic formulas only:

—the *Restricted Always* operator, $\overline{\Box}$;
—the *Restricted Next* operator, $\overline{\bigcirc}$; and
—the *Restricted Until* operator, $\overline{\mathcal{U}}$.

It is easy to show that these operators are strictly weaker than their fully temporal counterparts, which we refer to as *Unrestricted* operators.

Various subsets of TL can be constructed by choosing different subsets of the six temporal operators. For example, trace logic with $\overline{\Box}$ and $\overline{\bigcirc}$ is a (strict) subset of TL. It is easily seen that operators $\mathcal{U}$ and $\overline{\mathcal{U}}$ are not needed to encode $K_{\mathrm{ETL}}(\phi)$. Hence, the TL subsets of possible interest in expressing a formula equivalent to $K_{\mathrm{ETL}}(\phi)$ correspond to the subsets of $\{\Box, \bigcirc, \overline{\Box}, \overline{\bigcirc}\}$. The partial ordering of the expressive power of these subsets is given in Figure 4. For each nonempty subset S in Figure 4, let Tr_S denote the trace logic of SNL extended to include the operators in S. For example, subset 4 of Figure 4 is denoted by $\mathrm{Tr}_{\Box}$ and subset 8 by $\mathrm{Tr}_{\overline{\Box}\bigcirc}$. We have proved that $\mathrm{Tr}_{\Box}$ has the necessary and sufficient expressive power to encode $K_{\mathrm{ETL}}(\phi)$.

Sufficiency of $\mathrm{Tr}_{\Box}$ is shown by exhibiting a formula in $\mathrm{Tr}_{\Box}$ that is equivalent to $K_{\mathrm{ETL}}(\phi)$:

Definition 5.4. *Formula* $K(\phi)$ *in* $\mathrm{Tr}_{\Box}$

$$
\begin{aligned}
K_{\Box}(\phi) \models\ & \Box\phi \wedge \\
& (\forall\, i, j, x, y\colon 1 \le i \le m, 1 \le j \le m, 1 \le x, 0 \le y, i \ne j \vee x \ne y\colon \\
& \quad \Box(|c_i| < x \Leftrightarrow |c_j| < y) \Rightarrow \Box(|c_i| < x \wedge |c_j| < y)) \wedge \\
& (\forall\, i, x, v\colon 1 \le i \le m, i \le x, v \in V\colon \\
& \quad \Box(c_i.x = v \Rightarrow \Box(c_i.x = v)))
\end{aligned}
$$

[13]For example, consider TL formula $\Box(\psi_1 \Rightarrow (\Box\psi_2 \vee \bigcirc\psi_3))$, which asserts that whenever ψ_1 is valid, either ψ_2 is valid thereafter or ψ_3 is valid at the next point in time This property cannot be expressed using temporal operators only over first-order formulas.

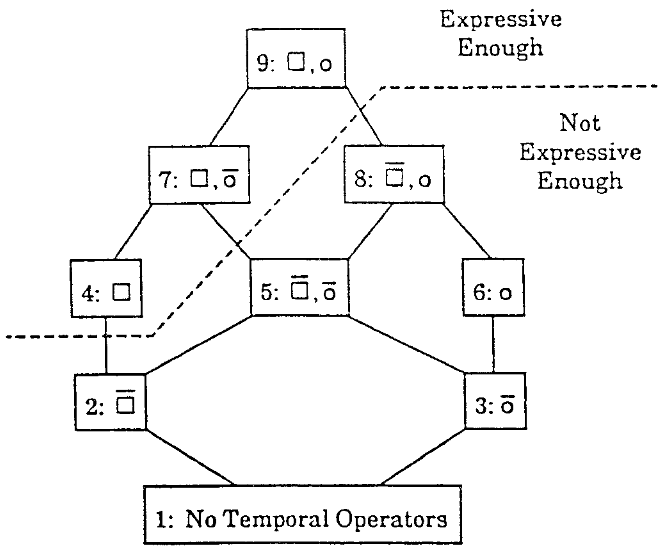

Fig. 4. Temporal logic subsets and expressiveness bounds.

The first conjunct of $K_{\Box}(\phi)$ restricts state-sequences satisfying $K_{\Box}(\phi)$ to always satisfy ϕ. The second and third conjuncts encode the Temporal Ordering and Prefix Properties, restricting state-sequences satisfying $K_{\Box}(\phi)$ to represent computations. Equivalence of $K_{\Box}(\phi)$ and $K_{\mathrm{ETL}}(\phi)$ is shown using a semantics-preserving mapping from Temporal Logic to Extended Trace Logic [16]. Necessity of $\mathrm{Tr}_{\Box}$ is proved by showing that no formula in $\mathrm{Tr}_{\bar{\Box}\circ}$ can be equivalent (through the mapping) to $K_{\mathrm{ETL}}(\phi)$. Hence, all subsets except 4, 7, and 9 are insufficient.

Note that our results can be strengthened by refining the subset hierarchy. Consider an infinite set of temporal operators based on allowable nesting depth. For any $x, 0 \leq x$, let $\Box_x$ denote a version of $\Box$ restricted to operate over formulas with at most x nested $\Box$s; similarly define operator $\circ_x$. We then obtain an infinite hierarchy of TL subsets. Given the previous results, it is easy to show that, with respect to this refined subset hierarchy, $\mathrm{Tr}_{\Box_1}$ is necessary and sufficient to express a formula equivalent to $K_{\mathrm{ETL}}(\phi)$.

6. CONCLUSIONS

We have considered a simple trace-based proof system for networks of processes, SNL, with a specification language and inference rules similar to those in most trace-based systems [3, 4, 6, 7, 8, 12, 14, 18]. Through examples that are single-process networks, we showed that SNL is incomplete because it is not expressive enough to encode properties of computation that are needed to verify certain valid network specifications. We then showed that axiomatization of the Temporal Ordering and Prefix Properties is necessary and sufficient to achieve relative completeness. Finally, we showed that the power of an unrestricted temporal logic *Always* operator is necessary and sufficient to express these axioms.

Our extensions to the consequence rule of SNL could have been encoded elsewhere in the proof system. For example, the network composition rule could have been extended so that Temporal Ordering and Prefix Axioms would be added whenever the rule is applied [16]; then, however, the network composition rule would have to be used even in the case of one-process networks (such as the examples in this paper). Another approach to obtaining relative completeness is to use a different interpretation for logical implication in the consequence rule, treating $\phi_1 \Rightarrow \phi_2$ as valid iff every legal computation (rather than every state) that always satisfies ϕ_1 also always satisfies ϕ_2. This has the disadvantage of requiring a new underlying logic, rather than relying on a first-order logic of states, as we have done in this paper.

A further question of expressiveness arises: Using SNL, can precise specifications be given for all processes of interest? There are some processes for which precise specifications cannot be given, for the same reason that SNL is incomplete: SNL specifications for such processes would require expressing Temporal Ordering or Prefix Properties. However, since we have proved that extension of SNL is necessary for completeness of the proof system, the same extension can be made to the specification language, allowing a wider class of precise specifications to be formulated. It would be interesting to characterize those processes for which precise specifications can be given in various trace-based formalisms. In addition, it might be interesting to consider how our results can be adapted to a network logic that allows *mixed terms* [18], a logic in which processes and specifications are interchangeable.

Since the expressive power of the *Always* operator, or of Temporal Ordering and Prefix Axioms, is an essential component of a relatively complete proof system, we briefly review existing complete systems and identify how this expressive power is encoded. No encoding is needed in [14], since the proof system is based directly on temporal logic. We have shown, however, that the full power of temporal logic used in [14] is unnecessary for proving safety properties. Several proof systems allow explicit reasoning over all possible computations [3, 6, 19]. As we have seen, this gives at least the expressive power of temporal logic, since the states of every computation can be directly and individually referenced.

In [19] the incompleteness of the proof system in [12] is discussed, and a rule is suggested that would render it relatively complete. (A similar rule is proposed in [13].) Informally, the rule asserts the following: Let ϕ be a valid specification for a network N, and let τ be an interleaved trace of all communication events during any computation of N. Then every prefix of τ satisfies ϕ. This rule certainly captures our Prefix Property. The Temporal Ordering Property is encoded as well. To see this, suppose specification ϕ constrains two communication events $c1.x$ and $c2.y$ to occur simultaneously. Any trace τ including only one of $c1.x$ and $c2.y$ will not satisfy ϕ and, thus, cannot correspond to a computation of N. Suppose, then, that both events are included in τ. Consider any prefix $\tau' \subseteq \tau$ that contains one event but not the

$c1.x$ and $c2.y$ appears in τ'. Hence, no computation of N can include either event.

In [8] the fact that valid specifications do not always follow from precise specifications is identified, and the author suggests adding a proof rule of the form

$$\frac{N \textbf{ sat } \phi_1}{N \textbf{ sat } \phi_2},$$

which can be applied whenever ϕ_1 and ϕ_2 are such that any network that always satisfies ϕ_1 always satisfies ϕ_2. No formal method is given, however, for determining when a pair of specifications is a candidate for application of the rule. Our work has exactly characterized those pairs that qualify and has isolated the expressiveness required of a logic that can recognize them.

APPENDIX

THEOREM 3.4. Soundness of SNL. *Let N be a network and ϕ a specification such that N* **sat** *ϕ is a theorem of SNL. Then ϕ is valid for N.*

PROOF. Since we are assuming validity of process specifications, soundness requires showing that whenever the hypothesis of an SNL inference rule is valid, so is the conclusion.

Network Composition Rule 3.1

$$\frac{P_1 \textbf{ sat } \phi_1, \ldots, P_n \textbf{ sat } \phi_n}{P_1 \| \cdots \| P_n \textbf{ sat } \bigwedge_i \phi_i}$$

Assume that each ϕ_i is valid for P_i, so $\kappa.j \models \phi_i$ for all $\kappa \in CS(P_i)$ and $j \geq 0$. We must show that $\kappa.j \models \bigwedge_i \phi_i$ for all $\kappa \in CS(N)$ and $j \geq 0$, where $N = P_1 \| \cdots \| P_n$. Recall that $Proj(\kappa, P_i)$ denotes the projection of κ onto those channels of N incident to P_i. Consider an arbitrary conjunct ϕ_i, an arbitrary $\kappa \in CS(N)$, and an arbitrary $j \geq 0$. By the definition of $CS(N)$ (Section 2.1), $Proj(\kappa, P_i) \in CS(P_i)$; hence, by assumption, $Proj(\kappa, P_i).j \models \phi_i$. Therefore, $\kappa.j \models \phi_i$ as well, since the traces that are removed from κ in the projection cannot be for channels mentioned in ϕ_i. Since κ, j, and ϕ_i were chosen arbitrarily, we conclude that $\kappa.j \models \bigwedge_i \phi_i$ for all $\kappa \in CS(N)$ and $j \geq 0$. Thus, $\bigwedge_i \phi_i$ is valid for N.

Consequence Rule 3.2

$$\frac{N \textbf{ sat } \phi_1, \models \phi_1 \Rightarrow \phi_2}{N \textbf{ sat } \phi_2}$$

Let ϕ_1 be valid for N, so $\kappa.j \models \phi_1$ for all $\kappa \in CS(N)$ and $j \geq 0$. Then, by $\models \phi_1 \Rightarrow \phi_2$ and predicate logic, $\kappa.j \models \phi_2$ for all $\kappa \in CS(N)$ and $j \geq 0$. Thus, ϕ_2 is valid for N. □

THEOREM 4.3. Preciseness-Preservation. *Let ϕ_i be a precise specification for P_i, $1 \le i \le n$, and let $N = P_1 \| \cdots \| P_n$. Then $\bigwedge_i \phi_i$ is a precise specification for N.*

PROOF. We must show that $\bigwedge_i \phi_i$ satisfies both parts of Definition 3.5 of precise specifications.

(1) ($\bigwedge_i \phi_i$ is valid for N.) Since the ϕ_i are precise specifications for their respective P_i, they are valid. That $\bigwedge_i \phi_i$ is then valid for N was proved in Soundness Theorem 3.4.

(2) (If κ is any computation containing traces for the channels in N and if $\kappa.j \vDash \bigwedge_i \phi_i$ for all $j \ge 0$, then $\kappa \in CS(N)$.) Consider any κ containing traces for the channels in N in which $\kappa.j \vDash \bigwedge_i \phi_i$ for all $j \ge 0$. Recall that $Proj(\kappa, P_i)$ denotes the projection of κ onto those channels of N incident to P_i. Since $\kappa.j \vDash \bigwedge_i \phi_i$ for all $j \ge 0$, $Proj(\kappa, P_i).j \vDash \phi_i$ for all $j \ge 0$ and $1 \le i \le n$. Thus, by the preciseness of the ϕ_i, $Proj(\kappa, P_i) \in CS(P_i)$, $1 \le i \le n$. Then, by the definition of $CS(P_1 \| \cdots \| P_n)$ (Section 2.1), $\kappa \in CS(N)$. □

THEOREM 4.6. Soundness of *ORDERING*. *If κ is a computation, then $\kappa \models$ ORDERING.*

PROOF. Let κ be any computation, and suppose that the antecedent of *ORDERING*, holds: $\kappa \models \Box(|c1| < x \Leftrightarrow |c2| < y)$ for some $x \ge 1$, $y \ge 0$. We prove by induction that each state of κ satisfies $(|c1| < x \wedge |c2| < y)$, so $\kappa \models \Box(|c1| < x \wedge |c2| < y)$ and, therefore, $\kappa \models$ *ORDERING*.

Base Case. We show that $\kappa.0$ satisfies $(|c1| < x \wedge |c2| < y)$. From $x \ge 1$ and computation condition (CC1) (Section 2.1), $\kappa.0$ satisfies $|c1| < x$. Then, by the antecedent of *ORDERING*, $\kappa.0$ also satisfies $|c2| < y$. Thus, $\kappa.0$ satisfies $(|c1| < x \wedge |c2| < y)$.

Induction. Suppose that $\kappa.(i-1)$ satisfies $(|c1| < x \wedge |c2| < y)$ for some $i > 0$. We show that $\kappa.i$ satisfies $(|c1| < x \wedge |c2| < y)$. Assume, for the sake of a contradiction, that $\kappa.i$ satisfies $(|c1| \ge x \vee |c2| \ge y)$. Then, by the antecedent of *ORDERING*, $\kappa.i$ also satisfies $(|c1| \ge x \wedge |c2| \ge y)$. Since $\kappa.(i-1)$ satisfies $(|c1| < x \wedge |c2| < y)$, if $c1$ and $c2$ are distinct then two channel traces change between $\kappa.(i-1)$ and $\kappa.i$, contradicting computation condition (CC4) (Section 2.1). If $c1$ and $c2$ are the same channel, then $x \ne y$, a channel trace increases in length by more than one between $\kappa.(i-1)$ and $\kappa.i$, and condition (CC3) is contradicted. Hence, $\kappa.i$ satisfies $(|c1| < x \wedge |c2| < y)$, and the induction is complete. □

LEMMA 4.9. *Let κ be any infinite sequence of states. κ represents a computation iff $\kappa \models$ ORDERING $\wedge$ PREFIX.*

PROOF. [$\Rightarrow$] (If κ represents a computation, then $\kappa \models$ *ORDERING* $\wedge$ *PREFIX*.) This follows directly from soundness of *ORDERING* and *PREFIX* (Theorems 4.6 and 4.8).

[$\Leftarrow$] (If $\kappa \models ORDERING \wedge PREFIX$, then κ represents a computation.) We prove the contrapositive: If κ does not represent a computation, then $\kappa \not\models ORDERING \wedge PREFIX$. By the definition of a computation (Section 2.1), if κ does not represent a computation then κ satisfies $\neg$(CC1 $\wedge$ CC2 $\wedge$ CC3 $\wedge$ CC4). Formula $\neg$(CC1 $\wedge$ CC2 $\wedge$ CC3 $\wedge$ CC4) can be rewritten as

$$(\neg CC2) \vee (CC2 \wedge \neg CC1) \vee (CC2 \wedge \neg CC3) \vee (CC2 \wedge CC3 \wedge \neg CC4). \quad (6)$$

We show that if κ satisfies any of the disjuncts in (6), then κ does not satisfy both *ORDERING* and *PREFIX*.

(1) κ does not satisfy (CC2): Some trace in some state is not a prefix of the corresponding trace in the subsequent state. Therefore, *PREFIX* does not hold.

(2) κ satisfies (CC2) but not (CC1): Some trace is nonempty in the initial state, so let $|c| = x$ in $\kappa.0$ with $x \geq 1$. Since (CC2) holds, κ satisfies $\Box(|c| \geq x)$, and therefore, $\Box(|c| < x \Leftrightarrow |c| < 0)$. However, κ does not satisfy $\Box(|c| < x \wedge |c| < 0)$, so *ORDERING* does not hold.

(3) κ satisfies (CC2) but not (CC3): Some trace increases in length by more than one between states, so let $|c| = x$ in some $\kappa.i$ and $|c| = x + y$ in $\kappa.(i + 1)$, with $y > 1$. Since (CC2) holds, κ satisfies $\Box(|c| < x + 1 \Leftrightarrow |c| < x + y)$. However, κ does not satisfy $\Box(|c| < x + 1 \wedge |c| < x + y)$ so *ORDERING* does not hold.

(4) κ satisfies (CC2) and (CC3) but not (CC4): More than one trace changes between some state and its subsequent state, so let $|c1| = x$ and $|c2| = y$ in some $\kappa.i$, and let $|c1| = x + 1$ and $|c2| = y + 1$ in $\kappa.(i + 1)$. Since (CC2) holds, κ satisfies $\Box(|c1| < x + 1 \Leftrightarrow |c2| < y + 1)$. However, κ does not satisfy $\Box(|c1| < x + 1 \wedge |c2| < y + 1)$, so *ORDERING* does not hold. $\Box$

ACKNOWLEDGMENTS

We are grateful to Prakash Panangaden for many useful discussions regarding the results described in Section 5, and to Stephen Brookes for his comments on an earlier version of the paper.

REFERENCES

1. Apt, K. R. Ten years of Hoare's logic: A survey—Part I. *ACM Trans. Program. Lang. Syst. 3*, 4 (Oct. 1981), 431-483.
2. Brock, J. D., and Ackerman, W. B. Scenarios: A model of non-determinate computation. In *Formalization of Programming Concepts*. Lecture Notes in Computer Science, vol. 107. Springer-Verlag, New York, 1981, pp. 252-259.
3. Brookes, S. D. A semantics and proof system for communicating processes. In *Logics of Programs*. Lecture Notes in Computer Science, vol. 164. Springer-Verlag, New York, 1984, pp. 68-85
4. Chen, Z. C., and Hoare, C. A. R. Partial correctness of communicating sequential processes. In *Proceedings of the IEEE International Conference on Distributed Computing Systems* (Paris, Apr. 1981), IEEE, New York, pp. 1-12.

5. Cook, S. A. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7*, 1 (Feb. 1978), 70-90.
6. Hehner, E. C. R., and Hoare, C. A. R. A more complete model of communicating processes. *Theor. Comput. Sci. 26* (Sept 1983), 105-120.
7. Hoare, C. A. R. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, N.J., 1985.
8. Jonsson, B. A model and proof system for asynchronous networks. In *Proceedıngs of the 4th ACM SIGACT-SIGOPS Symposium on Prınciples of Distributed Computing* (Aug. 1985) ACM, New York, pp. 49-58.
9. Lamport, L. Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng. SE-3*, 2 (Mar. 1977), 125-143.
10. Manna, Z., and Pnueli, A. Verification of concurrent programs: The temporal framework. In *The Correctness Problem in Computer Science*, R. S. Boyer and J. S. Moore, Eds. International Lecture Series in Computer Science. Academic Press, London, 1981, pp. 215-273
11. Manna, Z., and Pnueli, A. Verification of concurrent programs: A temporal proof system. In *Proceedings of the 4th School on Advanced Programming* (Amsterdam, June 1982), pp. 163-255.
12. Misra, J., and Chandy, K. M Proofs of networks of processes. *IEEE Trans. Softw. Eng. 7*, 7 (July 1981), 417-426.
13. Nguyen, V. The incompleteness of Misra and Chandy's proof systems *Inf. Process. Lett. 21* (Aug. 1985), 93-96.
14. Nguyen, V., Demers, A., Gries, D., and Owicki, S. A model and temporal proof system for networks of processes. *Distrib. Comput. 1*, 1 (Jan. 1986), 7-25.
15. Schoenfield, J. R. *Mathematical Logic.* Addison-Wesley, Reading, Mass., 1967.
16. Widom, J. Trace-based network proof systems: Expressiveness and completeness. Ph.D. thesis, Computer Science Dept. Cornell Univ., Ithaca, N.Y., May 1987.
17. Widom, J., Gries, D., and Schneider, F. B Completeness and incompleteness of trace-based network proof systems. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposıum on Prıncıples of Programmıng Languages* (Jan. 1987), ACM, New York, pp. 27-38.
18. Zwiers, J. *Compositionality, Concurrency, and Partial Correctness.* Lecture Notes ın Computer Science, vol 321 Springer-Verlag, Berlin, 1989.
19. Zwiers, J., de Roever, W. P., and van Emde Boas, P. Compositionality and concurrent networks: Soundness and completeness of a proofsystem. In *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming.* Lecture Notes in Computer Science, vol. 194. Springer-Verlag, Berlin, 1985, pp. 509-519.

Received February 1989; revised January 1990, November 1990, and August 1991; accepted August 1991