Trace-Based Network Proof Systems: Expressiveness and Completeness*

David Gries
Fred B. Schneider
Jennifer Widom

TR 89-966
February 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This research was supported in part by the National Science Foundation under grants DCR-8320274 and CCR-8701103.
Dr. Schneider is also supported by Office of Naval Research contract N00014-86-K-0092 and by Digital Equipment Corporation.

Trace-Based Network Proof Systems: Expressiveness and Completeness*

Jennifer Widom
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

David Gries
Fred B. Schneider
Computer Science Department
Cornell University
Ithaca, NY 14853

Abstract

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

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' components. Network proof systems based on first-order predicates over channel traces are given in [CH81,Hoa85,MC81], but unfortunately these logics are incomplete [BA81,Ngu85]. Relative completeness can be achieved by permitting reasoning over the full interleaving of communication events in addition to the individual channel traces [Bro84,HH83,ZdRvEB85] or by using temporal logic predicates so that the interleaving is implicitly present in the semantics of the specifications [NDGO86]. 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, a simple but logically incomplete trace-based proof system for safety properties is defined. This logic 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 while compositionality is an important feature of trace-based systems, network

*This paper is based on portions of 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. 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.

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 we identify 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 always a prefix of that channel's sequence at some 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 explore the exact expressiveness required of a trace logic if it is to be used for a relatively complete network proof system. 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 used in the remainder of the paper. A formal model of computation is introduced to facilitate subsequent reasoning about network behavior and proof systems. In Section 3, we define 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 consider the expressiveness required to achieve relative completeness in any trace-based proof system. We use a logic that permits reasoning over the interleaving of communication events to construct a formula that exactly characterizes the required expressiveness. A hierarchy of temporal logic subsets is then defined, along with a mapping from temporal logic formulas to formulas in the interleaving logic. The mapping is used to identify a subset of temporal logic with exactly the right expressive power for relative completeness. In Section 6, we describe how to generalize our model and proof system for hierarchically structured networks—networks in which component processes may be implemented as subnetworks. We show that our expressiveness and completeness results are not affected by this generalization. Finally, in Section 7, we summarize and explain how our results relate 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. 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, permitting communication between the network and its environment. Channels are unidirectional, and communication along them 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:

- 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.
- There is a fixed domain of values that can be transmitted on communication channels. 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. Fig. 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 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

[1] Extension to asynchronous message-passing is straightforward and does not affect our results [Wid87].

[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 as sub-networks [Wid87]. This generalization is discussed in Section 6.

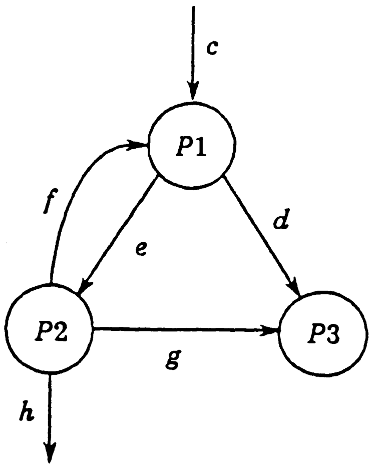

Figure 1: A network of processes

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 can represent 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.

[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 [MP81,Wid87].

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—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 $Proj(\kappa, P_i)$ to be the projection of κ onto those channels of N incident to P_i; that is, $Proj(\kappa, P_i)$ removes from the states of κ all traces of channels in N not incident to P_i. $CS(N)$ is the set of all computations κ built from possible states of N such that

- $Proj(\kappa, P_i) \in CS(P_i)$, $i = 1..n$, 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 [Lam77] 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 [CH81,Hoa85,MC81].[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

[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.

[5] Actually, in [MC81], specifications consist of predicate pairs, but for our purposes it is adequate to consider only single predicates that remain invariant throughout a computation [Wid87].

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

$$N \textbf{ sat } (|c1| - 1 \leq |c2| \leq |c1|) \wedge (\forall i\colon 1 \leq i \leq |c2|\colon c2.i = c1.i + 1).$$

Some network proof systems provide facilities for constructing specifications for sequential primitive processes [CH81, Hoa85, ZdRvEB85]. In others, it is assumed that existing logics for sequential programs can be used for this or that specifications for primitive processes are given [Jon85, MC81, NDGO86]. Without loss of generality, we adopt the latter approach. Thus, we are interested only in deducing specifications for a network from specifications for its component processes. Consequently, we assume that the axioms of SNL contain all formulas P **sat** ϕ, where P is a primitive process and ϕ is a valid specification for P, i.e. ϕ is satisfied by every execution of P.

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 $\bigwedge_i \phi_i$ 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 use the SNL Consequence Rule:

Definition 3.2 (Consequence Rule)

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

(Note that this rule relies on the validity of $\phi_1 \Rightarrow \phi_2$, which is a formula of the underlying trace logic. This aspect of the proof system is discussed below.)

These two rules, or variants thereof, form the the basis of most trace-based systems we know of, including [CH81,Hoa85,MC81,NDGO86,Zwi88].

3.1 Soundness and Completeness

We can use the computational model introduced in Section 2.1 to prove that SNL is *sound* [Apt81]: 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]
- $\kappa.i \models \phi$ holds iff the channel traces in $\kappa.i$ satisfy ϕ. ⊠

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.

Proof: See 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 proven when the specifications for N's component processes are as "strong" as possible [Jon85,NDGO86].

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 $\kappa \in CS(N)$. ⊠

---

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

[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.

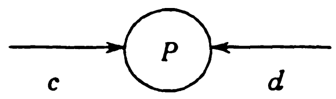

Figure 2: Simple network

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 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 [Sch67], a valid assertion might not be provable in any logic. Therefore, when designing a program proof system, one aims for *relative completeness* (as in [Coo78]): 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

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

4.1 Temporal Ordering Property

Consider the single-process network of Fig. 2. As an informal description of process P we are given four facts: (1) P reads at most one value from channel c; (2) P reads at most one value from channel d; (3) P reads a value from c before reading from d; (4) P reads a value from d before reading from c. By direct translation, a formal specification is

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

[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* [Apt81,Coo78]. We, too, have made an expressiveness assumption in our supposition that precise specifications for primitive processes can be written in SNL. The reader might convince himself that our language is powerful enough to express precise specifications for a large class of processes.

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

$$P \textbf{ sat } \phi_2 \colon (c = \langle\rangle \ \wedge \ d = \langle\rangle) \ \vee \ (c = \langle a \rangle \ \wedge \ d = \langle a \rangle) \tag{2}$$

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

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

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 precise specification ϕ_2 (say) 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 $\phi_2 \Rightarrow \phi_3$ 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—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 proven in Section 5.

4.2 Prefix Property

Consider the network of Fig. 3. Suppose $\{a, b\}$ is the data domain and let a precise specifi-

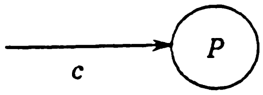

Figure 3: Very simple network

cation for process P be

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

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

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

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

Definition 4.2 (Prefix Property) For any channel c and integers $0 \leq x \leq y$, 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 $N = P_1 \| \cdots \| P_n$. 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** $\bigwedge_i \phi_i$. 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 $\bigwedge_i \phi_i \Rightarrow \phi$ (a formula of the underlying trace logic).

By the following theorem, we know that specification $\bigwedge_i \phi_i$ is precise for N—the conjunction of precise process specifications results in a network specification that is also precise. (A similar theorem for a different logic is proven in [NDGO86].)

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

Proof: See appendix. ⊠

Thus, our proof system would be relatively complete if $\phi_1 \Rightarrow \phi_2$ 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 A (say) is the conjunction of axioms in the set, then $(\phi_1 \wedge A) \Rightarrow \phi_2$ 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, while 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 entire computations, we introduce an operator $\Box$, which converts specifications to being over computations: $\Box\,\phi$ is valid for a computation iff ϕ is true in every state of that computation.[9] That is:

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

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

Definition 4.4 (Modified Consequence Rule)

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

Now we consider the axioms comprising A.

[9]This is a weakened version of the "always", or "henceforth", operator (also $\Box$) of temporal logic [MP81, MP82], since ϕ cannot contain other temporal operators. Temporal operators of varying strengths are discussed briefly in Section 4.4.2 and at length in Section 5.

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 indices x and y, even if $x = y$ or $c1$ and $c2$ are the same channel. We are interested only in ordering of distinct events, so the case in which $c1.x$ 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 $\Box(|c2| < y \Rightarrow |c1| < x)$ holds as well (with $c1$ and $c2$ distinct and $x \neq y$), 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]

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

Allowing $y = 0$ permits the assertion that an empty channel trace temporally precedes all communication 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 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 appendix. $\boxtimes$

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 $\Box$ [MP82].) Now, $\Box\,\phi$ is valid for a computation iff ϕ is valid for every suffix of that computation:

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

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

[11] Suppose, for the sake of a contradiction, that c is non-empty 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.

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, i.e. $\kappa.i \models \phi$. (For more detailed and rigorous discussions of the semantics of temporal operators, see e.g. [MP82,Wid87].)

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

$$\Box\,(c.x = v \;\Rightarrow\; \Box\,(c.x = v)). \quad \boxtimes$$

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 C2 (Section 2.1) and *PREFIX* follows directly. $\boxtimes$

4.4.3 Necessity and Sufficiency of *ORDERING* and *PREFIX*

By letting $A = \mathit{ORDERING} \wedge \mathit{PREFIX}$, 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 $(\Box\,\phi_1 \;\wedge\; A) \Rightarrow \Box\,\phi_2$ (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 $\kappa \models \mathit{ORDERING} \wedge \mathit{PREFIX}$.

Proof: See appendix. $\boxtimes$

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

[12] A different axiomatization of the Prefix Property can be given using the "next" operator of temporal logic in addition to $\Box$ [WGS87]. The definition given here, however, shows that the Prefix Property can be encoded using only $\Box$ operators. 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.

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 $(\Box\,\phi_1 \wedge \mathit{ORDERING} \wedge \mathit{PREFIX}) \Rightarrow \Box\,\phi_2$.

Proof: We show that any infinite sequence of states κ satisfying $\Box\,\phi_1$, *ORDERING*, and *PREFIX* also satisfies $\Box\,\phi_2$. Since $\kappa \models \mathit{ORDERING} \wedge \mathit{PREFIX}$, 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, $\kappa \in \mathit{CS}(N)$. By validity of ϕ_2, every $\kappa \in \mathit{CS}(N)$ satisfies $\Box\,\phi_2$. Hence κ satisfies $\Box\,\phi_2$. ⊠

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 processes. That *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. $\neg\,((\Box\,\phi_1^P \wedge \mathit{ORDERING}) \Rightarrow \Box\,\phi_1^V)$;
2. $\neg\,((\Box\,\phi_2^P \wedge \mathit{PREFIX}) \Rightarrow \Box\,\phi_2^V)$;
3. $\neg\,(\Box\,\phi_3^P \Rightarrow \Box\,\phi_3^V)$.

Proof:

1. Let $N1$ be the example network of Section 4.2.
2. Let $N2$ be the example network of Section 4.1.
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, $\mathit{ORDERING} \wedge \mathit{PREFIX}$ 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, distinguishing those states that are reachable by a computation from those that are not. Thus, we are interested in the expressiveness required of trace logics that encode the notion of a legal network computation.

We formalize this requirement by analyzing the relationship between precise and valid specifications. Suppose ϕ_1 is a precise specification for a network N. Then, by Definition 3.5, a state is reachable by N iff it is reachable by a computation that always satisfies ϕ_1. Now, by Definition 3.3 of validity, a specification ϕ_2 is valid for N iff it is satisfied by every state reachable by N. By transitivity, then, ϕ_2 is valid iff it is satisfied by every state reachable by a computation that always satisfies ϕ_1. That is:

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:

Definition 5.2 (Generalized Consequence Rule)

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

where $K(\phi_1) \Rightarrow \phi_2$ is a formula of logic L. 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 $K(\phi_1) \Rightarrow \phi_2$. Therefore, incorporating the Generalized Consequence Rule yields a proof system that is complete relative to L. Our goal is thus to isolate the power required of a logic to express formula $K(\phi)$—a formula satisfied by exactly those states reachable by a computation that always satisfies ϕ.

5.1 Extending Trace Logic

Formula $K(\phi)$ can be expressed in an extended trace logic in which, in addition to reasoning over traces of individual communication channels, some explicit reasoning over computations is permitted. Without loss of generality, suppose that 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. Recalling the computational model of Section 2.1, a state can be represented by a tuple $\bar{t} = [t_1, \ldots, t_m]$ of channel traces, where t_i is the trace of c_i, $1 \leq i \leq m$. A computation up to some point in time is represented by a finite sequence of such tuples:

$\langle \overline{t^0}, \overline{t^1}, \ldots, \overline{t^n} \rangle$. (Since we are interested only in states reachable by computations, we need not consider the infinite sequences representing full computations.)

It is not surprising that $K(\phi)$ can be expressed in a logic that allows quantification over finite, arbitrarily long, sequences of tuples—the reachable states can be explicitly generated. Let

- $\phi[\bar{c}/\,\overline{t^i}\,]$ denote SNL specification ϕ with channel trace variables $c_1, \ldots, c_m$ replaced by traces $t^i_1, \ldots, t^i_m$;[13]
- $\overline{t^i}\,[t^i_j \,/\, t^i_j \cdot \langle v \rangle]$ denote tuple $\overline{t^i}$ with trace t^i_j extended by value $v \in V$.

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

Definition 5.3 (Formula $K(\phi)$ in Extended Trace Logic)

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

The free variables of $K_{\text{ETL}}(\phi)$ are channel trace variables $c_1, \ldots, c_m$. As illustrated by the annotation, $K_{\text{ETL}}(\phi)$ is satisfied by exactly those states reachable by a computation that always satisfies ϕ. Therefore, Definition 5.3 of $K_{\text{ETL}}(\phi)$ could be used in the Generalized Consequence Rule to obtain relative completeness.

5.2 Using Temporal Logic

Linear-Time Temporal Logic (TL) is a formalism for reasoning over an implicit sequence of states [MP82]. TL is less complicated and more appropriate than the Extended Trace Logic of Section 5.1 for expressing and reasoning with formulas such as $K(\phi)$. As indicated by the definitions of *ORDERING* and *PREFIX* given in Sections 4.4.1 and 4.4.2, however, full temporal logic is more powerful than is needed to express $K(\phi)$. We therefore introduce a

[13] In general, we use $X[old/new]$ to denote entity X with all free occurrences of item *old* replaced by item *new*.

number of temporal operators that can be used to define subsets of TL. We then isolate a TL subset that is necessary and sufficient to express $K(\phi)$ and is appropriate for use as the basis of a relatively complete trace-based proof system.

We begin with a version of TL that extends the trace logic of SNL with three standard temporal operators:

- The *Always* operator, $\Box$. Informally, $\Box\,\psi$ is valid iff TL formula ψ is valid at the current point in time and at every point in the future.
- The *Next* operator, $\circ$. Informally, $\circ\,\psi$ is valid iff TL formula ψ is valid at the next point in time.
- The *Until* operator, $\mathcal{U}$. Informally, $\psi_1\ \mathcal{U}\ \psi_2$ is valid for TL formulas ψ_1 and ψ_2 iff ψ_2 is valid either at the current point in time or at some point in the future, and ψ_1 is valid at all points from the current point to the point at which ψ_2 becomes valid.[14]

(We omit the *Eventually* operator, $\Diamond$, since $\Diamond$ is the dual of $\Box$: for any TL formula ψ, $\Box\,\psi \Leftrightarrow \neg\Diamond\neg\psi$ and $\Diamond\psi \Leftrightarrow \neg\Box\neg\psi$.)

TL formulas are interpreted on a network computation by considering a sequence of states as a description of successive points in time. The temporal operators are interpreted in the obvious way, according to the informal descriptions above [MP81,MP82]. (See, e.g., the definitions for $\Box$ given in Section 4.) Our original definition of $\Box$, given in Section 4.4, is for trace logic formulas that do not contain temporal operators. In Section 4.4.2, we consider 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 non-temporal formulas.[15] 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, $\overline{\circ}$
- the *Restricted Until* operator, $\overline{\mathcal{U}}$

[14]Some definitions of TL instead use a *Weak Until* operator, in which ψ_2 need not ever become valid as long as ψ_1 is always valid. In the context of TL, the two versions of $\mathcal{U}$ are expressively equivalent [Wol81].

[15]For example, consider TL formula $\Box\,(\psi_1 \Rightarrow (\Box\,\psi_2\ \vee\ \circ\,\psi_3))$, 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.

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

We want to isolate the TL subset that is both necessary and sufficient to express formula $K(\phi)$. Various subsets of TL can be constructed by choosing different subsets of the six temporal operators; for example, trace logic with $\Box$ and $\bar{\circ}$ is a (strict) subset of TL. Suppose we give an interpretation for TL formulas in the Extended Trace Logic (ETL) of Section 5.1, i.e., we define a mapping $\mathcal{M}$ from formulas in TL to formulas in ETL. Then we can establish expressiveness bounds by proving that certain sets of temporal operators are required in any TL formula that is equivalent (through the mapping) to Definition 5.3 of $K_{\mathrm{ETL}}(\phi)$.

Mapping $\mathcal{M}$ is derived directly from the definitions of the temporal operators. If ψ_{TL} is any formula of temporal logic, then $\mathcal{M}[\![\psi_{\mathrm{TL}}]\!]$ is an ETL formula containing one free variable, σ, which ranges over infinite sequences of tuples (representing states). The mapping is *semantics-preserving*, in that a sequence σ' satisfies formula ψ_{TL} iff substitution $[\sigma/\sigma']$ satisfies formula $\mathcal{M}[\![\psi_{\mathrm{TL}}]\!]$.[16] Using such a mapping, TL formula ψ_{TL} is said to be equivalent to ETL formula ψ_{ETL} iff $\mathcal{M}[\![\psi_{\mathrm{TL}}]\!] \Leftrightarrow \psi_{\mathrm{ETL}}$.

The full definition of $\mathcal{M}$ is given in Table 1. $\sigma = \langle \sigma.0, \sigma.1, \sigma.2, \ldots \rangle$, and $\sigma[i..]$ denotes the suffix of σ starting at $\sigma.i$ $(i \geq 0)$. $\mathcal{M}$ is defined inductively on the structure of TL formulas, and parallels the usual interpretation of temporal logic [MP82]. Note that a mapping $\mathcal{T}$ from TL terms to ETL terms is also needed. We have omitted $\bar{\Box}$, $\bar{\circ}$, and $\bar{\mathcal{U}}$, since the mappings for these operators are identical to the mappings for $\Box$, $\circ$, and $\mathcal{U}$, respectively.

To consider equivalence between $K_{\mathrm{ETL}}(\phi)$ and formulas mapped from TL to ETL, $K_{\mathrm{ETL}}(\phi)$ must be redefined so that its existentially quantified sequence of states is represented by a free variable σ. Define $K_\sigma(\phi)$ as follows.

Definition 5.4 (Modified $K(\phi)$ in Extended Trace Logic)

$$
\begin{aligned}
&K_\sigma(\phi) \equiv \\
&\quad \sigma.0 = [\langle\rangle, \ldots, \langle\rangle] \ \wedge \\
&\quad (\forall i\colon 0 \leq i\colon \phi[\bar{c}/\sigma.i]) \ \wedge \\
&\quad (\forall i\colon 0 \leq i\colon \\
&\qquad \sigma.(i+1) = \sigma.i \ \vee \\
&\qquad (\exists j, v\colon 1 \leq j \leq m, v \in V\colon \\
&\qquad\quad \sigma.(i+1) \ = \ \sigma.i[(\sigma.i)_j/(\sigma.i)_j \cdot \langle v \rangle])))
\end{aligned}
$$

The correspondence between $K_\sigma(\phi)$ and $K_{\mathrm{ETL}}(\phi)$ (Definition 5.3) should be clear. It is easily

[16] A rigorous proof of semantics-preservation requires formal semantics for TL and ETL. We have avoided giving such here, referring the interested reader to [MP82,Wid87].

Table 1: Mapping $\mathcal{M}$ from TL formulas to ETL formulas

$\mathcal{M}[\![p(t_1,\ldots,t_n)]\!]$	$=$	$p(\mathcal{T}[\![t_1]\!],\ldots,\mathcal{T}[\![t_n]\!])$	p a predicate, $t_1,\ldots,t_n$ TL terms
$\mathcal{M}[\![\psi_1 \vee \psi_2]\!]$	$=$	$\mathcal{M}[\![\psi_1]\!] \vee \mathcal{M}[\![\psi_2]\!]$	ψ_1 and ψ_2 TL formulas
$\mathcal{M}[\![\neg\psi]\!]$	$=$	$\neg\mathcal{M}[\![\psi]\!]$	ψ a TL formula
$\mathcal{M}[\![(\exists x :: \psi)]\!]$	$=$	$(\exists k :: \mathcal{M}[\![\psi[x/k]]\!])$	ψ a TL formula, k a constant in V
$\mathcal{M}[\![\Box\,\psi]\!]$	$=$	$(\forall i\colon 0 \le i\colon \mathcal{M}[\![\psi]\!][\sigma/\sigma[i..]])$	ψ a TL formula
$\mathcal{M}[\![\circ\,\psi]\!]$	$=$	$\mathcal{M}[\![\psi]\!][\sigma/\sigma[1..]]$	ψ a TL formula
$\mathcal{M}[\![\psi_1\,\mathcal{U}\,\psi_2]\!]$	$=$	$(\exists i\colon 0 \le i\colon (\mathcal{M}[\![\psi_2]\!][\sigma/\sigma[i..]]) \wedge$ $(\forall j\colon 0 \le j < i\colon \mathcal{M}[\![\psi_1]\!][\sigma/\sigma[j..]]))$	ψ_1 and ψ_2 TL formulas
$\mathcal{T}[\![k]\!]$	$=$	k	k a constant in V
$\mathcal{T}[\![c_i]\!]$	$=$	$(\sigma.0)_i$	c_i a variable in $\{c_1,\ldots,c_m\}$
$\mathcal{T}[\![f(t_1,\ldots,t_n)]\!]$	$=$	$f(\mathcal{T}[\![t_1]\!],\ldots,\mathcal{T}[\![t_n]\!])$	f a function, $t_1,\ldots,t_n$ TL terms

verified [Wid87] that for any SNL specifications ϕ_1 and ϕ_2,

$$K_{\text{BTL}}(\phi_1) \Rightarrow \phi_2 \quad \text{iff} \quad K_\sigma(\phi_1) \Rightarrow (\forall i\colon 0 \le i\colon \phi_2[\bar{c}/\sigma.i]).$$

By the definition of $K_\sigma(\phi)$, we see that any TL subset that can express $K_\sigma(\phi)$ can also express $(\forall i\colon 0 \le i\colon \phi_2[\bar{c}/\sigma.i])$. Therefore, we can revise the Generalized Consequence Rule to use the equivalent implication $K_\sigma(\phi_1) \Rightarrow (\forall i\colon 0 \le i\colon \phi_2[\bar{c}/\sigma.i])$, adopting Definition 5.4 of $K_\sigma(\phi)$ as our measure of the required expressiveness.

5.3 Expressiveness Bounds for Relative Completeness

We want to determine which combination of the six temporal operators defined in Section 5.2 is necessary and sufficient in any formula $K_{\text{TL}}(\phi)$ such that $\mathcal{M}[\![K_{\text{TL}}(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$. By the definition of mapping $\mathcal{M}$ on formulas of the form $\psi_1\ \mathcal{U}\ \psi_2$, and by the fact that $K_\sigma(\phi)$ contains no eventuality components of the form $(\exists i\colon 0 \le i\colon f(\sigma.i))$, we see that operator $\mathcal{U}$ is not needed in any TL formula $K_{\text{TL}}(\phi)$ such that $\mathcal{M}[\![K_{\text{TL}}(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$. Similarly, there is no need to consider operator $\overline{\mathcal{U}}$. Therefore, the TL subsets of interest correspond to the subsets of $\{\Box, \circ, \overline{\Box}, \overline{\circ}\}$. The partial ordering of the expressive power of these subsets is given in Fig. 4.

For each non-empty subset S in Fig. 4, let Tr_S denote the trace logic of SNL extended to include the operators in S. For example, subset 4 of Fig. 4 is denoted by $\text{Tr}_\Box$ and

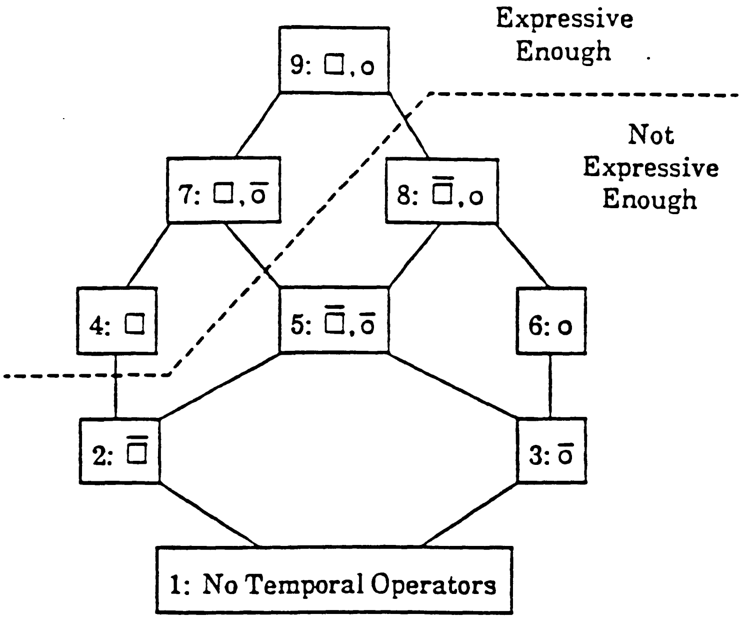

Figure 4: Temporal logic subsets and expressiveness bounds

subset 8 by $\mathrm{Tr}_{\overline{\Box}\circ}$. We prove that $\mathrm{Tr}_{\Box}$ has the necessary and sufficient expressive power to encode $K_\sigma(\phi)$. We show that $\mathrm{Tr}_{\Box}$ is sufficient by using trace logic with *Unrestricted Always* operators (only) to write a formula equivalent to $K_\sigma(\phi)$. (Not surprisingly, this formula is based on axioms *ORDERING* and *PREFIX.*) We then prove that the subset is an absolute lower bound: each subset lower than or incomparable to $\mathrm{Tr}_{\Box}$ (subset 4) in the hierarchy of Fig. 4 is not expressive enough to encode $K_\sigma(\phi)$. This is proven by showing that no formula equivalent to $K_\sigma(\phi)$ can be expressed in $\mathrm{Tr}_{\overline{\Box}\circ}$ (subset 8). Consequently, all subsets except 4, 7, and 9 are insufficient. The resulting division of the subset hierarchy is shown in Fig. 4.

5.3.1 Sufficiency

We give a formula $K_{\Box}(\phi)$ in $\mathrm{Tr}_{\Box}$ such that $\mathcal{M}[\![K_{\Box}(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$:

Definition 5.5 (Formula $K(\phi)$ in $\mathrm{SNL}_{\Box}$)

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

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.

Theorem 5.6 $\mathcal{M}[\![K_{\Box}(\phi)]\!] \Leftrightarrow K_{\sigma}(\phi)$.

Proof: See appendix. ⊠

Thus, the expressive power of $\mathrm{Tr}_{\Box}$ is sufficient to encode a formula equivalent to $K_{\sigma}(\phi)$.

5.3.2 Necessity

We now show that, with respect to our hierarchy of TL subsets, $\mathrm{Tr}_{\Box}$ is necessary to encode $K_{\sigma}(\phi)$—any TL subset weaker than or incomparable to $\mathrm{Tr}_{\Box}$ cannot be used to express a formula equivalent to $K_{\sigma}(\phi)$. This requires proving that no formula equivalent to $K_{\sigma}(\phi)$ can be expressed in $\mathrm{Tr}_{\overline{\Box}\bigcirc}$.

First, we prove a key lemma, that there is no formula $\psi_{\overline{\Box}\bigcirc}$ in $\mathrm{Tr}_{\overline{\Box}\bigcirc}$ such that $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!]$ and $K_{\sigma}(true)$ are satisfied by the same set of substitutions for free variable σ. (Note that *true* is an SNL specification satisfied by every network.) The final result—that there is no formula $K_{\overline{\Box}\bigcirc}(\phi)$ in $\mathrm{Tr}_{\overline{\Box}\bigcirc}$ such that $\mathcal{M}[\![K_{\overline{\Box}\bigcirc}(\phi)]\!] \Leftrightarrow K_{\sigma}(\phi)$—then follows directly. To prove that there is no formula $\psi_{\overline{\Box}\bigcirc}$ in $\mathrm{Tr}_{\overline{\Box}\bigcirc}$ such that $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!]$ and $K_{\sigma}(true)$ are satisfied by the same set of substitutions, we show, for every potential $\psi_{\overline{\Box}\bigcirc}$, that there is some state-sequence σ' such that either $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma']$ but not $K_{\sigma}(true)[\sigma/\sigma']$, or $K_{\sigma}(true)[\sigma/\sigma']$ but not $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma']$.

Informally, the argument proceeds as follows. For every $\psi_{\overline{\Box}\bigcirc}$ in $\mathrm{Tr}_{\overline{\Box}\bigcirc}$ there is some $n \geq 0$ (n is the nesting depth of *Next* operators in $\psi_{\overline{\Box}\bigcirc}$) such that $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!]$ can refer to states of σ beyond $\sigma.n$ only by universal quantification (resulting from *Restricted Always* operators in $\psi_{\overline{\Box}\bigcirc}$). If no state-sequences σ' satisfy $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma']$, or the only satisfying sequences have all repeated states beyond $\sigma'.n$, then it is straightforward to construct a sequence σ'' such that $K_{\sigma}(true)[\sigma/\sigma'']$ but not $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma'']$. Otherwise, we construct a sequence σ'' that does not represent a computation due to an irregularity beyond state $\sigma''.n$ (e.g. the length of a trace decreases from one state to the next). $K_{\sigma}(true)[\sigma/\sigma'']$ does not hold, since σ'' does not represent a computation; however, $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma'']$ does hold, as long as σ'' is constructed by rearranging states from a sequence σ' know to satisfy $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma']$.

Lemma 5.7 For any formula $\psi_{\overline{\Box}\bigcirc}$ in $\mathrm{Tr}_{\overline{\Box}\bigcirc}$, there exists a state-sequence σ' such that either

$\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma']$ and not $K_{\sigma}(true)[\sigma/\sigma']$, or

$K_{\sigma}(true)[\sigma/\sigma']$ and not $\mathcal{M}[\![\psi_{\overline{\Box}\bigcirc}]\!][\sigma/\sigma']$.

Proof: See appendix. ⊠

Theorem 5.8 There is no formula $K_{\overline{\Box}\circ}(\phi)$ in $\mathrm{Tr}_{\overline{\Box}\circ}$ such that $\mathcal{M}[\![K_{\overline{\Box}\circ}(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$.

Proof: Consider an arbitrary formula $K_{\overline{\Box}\circ}(\phi)$ in $\mathrm{Tr}_{\overline{\Box}\circ}$. $\mathcal{M}[\![K_{\overline{\Box}\circ}(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$ iff for all trace-logic formulas ϕ and state-sequences σ': $\mathcal{M}[\![K_{\overline{\Box}\circ}(\phi)]\!][\sigma/\sigma']$ iff $K_\sigma(\phi)[\sigma/\sigma']$. Suppose $\phi \equiv \mathit{true}$. By Lemma 5.7, there exists a state-sequence σ' such that either $\mathcal{M}[\![K_{\overline{\Box}\circ}(\mathit{true})]\!][\sigma/\sigma']$ and not $K_\sigma(\mathit{true})[\sigma/\sigma']$ or $K_\sigma(\mathit{true})[\sigma/\sigma']$ and not $\mathcal{M}[\![K_{\overline{\Box}\circ}(\mathit{true})]\!][\sigma/\sigma']$. Hence $\mathcal{M}[\![K_{\overline{\Box}\circ}(\mathit{true})]\!] \not\Leftrightarrow K_\sigma(\mathit{true})$ and consequently $\mathcal{M}[\![K_{\overline{\Box}\circ}(\phi)]\!] \not\Leftrightarrow K_\sigma(\phi)$. ⊠

This result can be strengthened by refining the subset hierarchy. Rather than distinguishing only between *Restricted* and *Unrestricted* temporal operators—operators that may be nested zero or arbitrarily many times, respectively—consider an infinite set of temporal operators based on allowable nesting depth. For any x, $0 \leq x \leq \infty$, let $\Box_x$ denote a version of $\Box$ restricted to operate over formulas with at most x nested $\Box$'s; similarly define operator $\circ_x$. From these infinite sets of temporal operators we obtain an infinite hierarchy of TL subsets. Given the results of Sections 5.3.1 and 5.3.2, 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 $K_\sigma(\phi)$ [Wid87].

6 Hierarchically Structured Networks

Thus far, we have restricted attention to networks constructed directly from primitive processes. A *hierarchically structured network* is a network in which the component processes may be either primitive processes or sub-networks. For simplicity, assume that all process and channel names are unique throughout the hierarchy. Our model and proof system easily generalize to such hierarchically structured networks; our results remain unchanged.

Recall from Section 2 that our model of network computation is compositional—the set representing the behavior of a network can be constructed from the sets representing the behavior of the network's component processes. To adapt this model for hierarchically structured networks, we construct the set representing the behavior of a network by inductively applying the construction to the network's components (assuming that the sets for the base processes are given). Internal channels of a process implemented as a sub-network can be hidden by simply eliminating all traces of the process's internal channels when constructing the set for the network.

No modifications to the SNL inference rules are needed for hierarchically structured networks. If a component process is implemented as a sub-network, the proof system itself is used to verify a specification for the sub-network. Furthermore, by inductive application of Preciseness Preservation Theorem 4.3, from precise specifications for base processes, precise specifications for networks at any level of a hierarchy can be obtained. Thus, our assumption regarding availability of precise specifications for a network's component processes remains valid. Finally, if one wishes to simplify specifications by hiding the existence of internal channels in component sub-networks, this can be done using Consequence Rule 3.2.

Since our model and proof system adapt directly to hierarchically structured networks, it should be clear that our fundamental results regarding expressiveness and completeness are still valid. In fact, all definitions, examples, lemmas, and theorems of Sections 4 and 5 are left unchanged.

7 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 [Bro84, CH81, HH83, Hoa85, Jon85, MC81, NDGO86, Zwi88]. 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 Temporal Ordering and Prefix Properties is necessary and sufficient to achieve relative completeness. The Temporal Ordering and Prefix Axioms characterize legal network computation; thus, we investigated the expressiveness needed in any relatively complete system by considering logics that can perform this function. We found that the power of an unrestricted temporal logic *Always* operator is an upper and lower bound.

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, it is interesting to look at existing complete systems and identify how this expressive power is encoded. No encoding is needed in [NDGO86], since the proof system is based directly on temporal logic. However, as we have shown, the full power of temporal logic present in that proof system is not necessary for proving safety properties. Several proof systems allow explicit reasoning over all possible computations [Bro84,HH83,ZdRvEB85], as in the Extended Trace Logic of Section 5.1. 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 [ZdRvEB85], the incompleteness of the proof system in [MC81] is discussed and a rule is suggested that would render it relatively complete. (A similar rule is proposed in [Ngu85].) 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 other (such a prefix must exist). Then τ' will not satisfy ϕ, since only one of $c1.x$ and $c2.y$ appears in τ'. Hence no computation of N can include either event.

In [Jon85], the fact that valid specifications do not always follow from precise specifications is identified, but no actual solution is proposed. 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 will also always satisfy 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 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:

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

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 $\phi_1 \Rightarrow \phi_2$ 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, $1 \leq i \leq n$, 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 proven in Soundness Theorem 3.4.

2. (If κ is any computation containing traces for the channels in N, and $\kappa.j \models \bigwedge_i \phi_i$ for all $j \geq 0$, then $\kappa \in CS(N)$.) Consider any κ containing traces for the channels in N in which $\kappa.j \models \bigwedge_i \phi_i$ for all $j \geq 0$. Recall that $Proj(\kappa, P_i)$ denotes the projection of κ onto those channels of N incident to P_i. Since $\kappa.j \models \bigwedge_i \phi_i$ for all $j \geq 0$, $Proj(\kappa, P_i).j \models \phi_i$ for all $j \geq 0$ and $1 \leq i \leq n$. Thus, by the preciseness of the ϕ_i, $Proj(\kappa, P_i) \in CS(P_i)$, $1 \leq i \leq n$. 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 $x \geq 1$, $y \geq 0$. 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 $x \geq 1$ 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 $(|c1| \geq x \vee |c2| \geq y)$. Then, by the antecedent of *ORDERING*, $\kappa.i$ also satisfies $(|c1| \geq x \wedge |c2| \geq y)$. 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 $x \neq y$, 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

$$(\neg \mathrm{CC2}) \vee (\mathrm{CC2} \wedge \neg \mathrm{CC1}) \vee (\mathrm{CC2} \wedge \neg \mathrm{CC3}) \vee (\mathrm{CC2} \wedge \mathrm{CC3} \wedge \neg \mathrm{CC4}). \qquad (6)$$

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 non-empty 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. ⊠

Theorem 5.6 $\mathcal{M}[\![K_\Box(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$.

Proof: Proving $\mathcal{M}[\![K_\Box(\phi)]\!] \Leftrightarrow K_\sigma(\phi)$ requires showing that for all trace logic formulas ϕ and state-sequences σ', $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$ iff $K_\sigma(\phi)[\sigma/\sigma']$. First, applying the definition of $\mathcal{M}$, we obtain[17]

$$
\begin{aligned}
&\mathcal{M}[\![K_\Box(\phi)]\!] \equiv \\
&\quad (\forall i\colon 0 \le i\colon \phi[\bar{c}/\sigma.i]) \wedge \\
&\quad (\forall i,j,x,y\colon 1 \le i \le m, 1 \le j \le m, 1 \le x, 0 \le y, i \ne j \ \vee\ x \ne y\colon \\
&\qquad (\forall k\colon 0 \le k\colon |(\sigma.k)_i| < x \Leftrightarrow |(\sigma.k)_j| < y) \Rightarrow \\
&\qquad\quad (\forall k\colon 0 \le k\colon |(\sigma.k)_i| < x \ \wedge\ |(\sigma.k)_j| < y)) \wedge \\
&\quad (\forall i,x,v\colon 1 \le i \le m, 1 \le x, v \in V\colon \\
&\qquad (\forall k\colon 0 \le k\colon ((\sigma.k)_i.x = v) \Rightarrow (\forall l\colon 0 \le l\colon (\sigma.(k+l))_i.x = v))).
\end{aligned}
$$

Now consider an arbitrary trace-logic formula ϕ and state-sequence σ'.

[$\Rightarrow$] If $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$ then $K_\sigma(\phi)[\sigma/\sigma']$:

We prove the contrapositive: the falsity of $K_\sigma(\phi)[\sigma/\sigma']$ implies the falsity of $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$. Suppose $K_\sigma(\phi)[\sigma/\sigma']$ is false, and consider the three conjuncts of $K_\sigma(\phi)$. The second conjunct of $K_\sigma(\phi)$ is identical to the first conjunct of $\mathcal{M}[\![K_\Box(\phi)]\!]$. Therefore, if the second conjunct of $K_\sigma(\phi)[\sigma/\sigma']$ is false then $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$ is also false. Suppose the first or third conjunct of $K_\sigma(\phi)[\sigma/\sigma']$ is false. Then some trace in the initial state of σ' is non-empty or some subsequent state does not extend at most one trace of the preceding state by at most one element, i.e. σ' does not represent a computation. Note that the second and third conjuncts of $\mathcal{M}[\![K_\Box(\phi)]\!]$ are derived directly from the Temporal Ordering and Prefix Axioms, and recall Lemma 4.9: a state-sequence κ represents a computation iff $\kappa \models$ *ORDERING* $\wedge$ *PREFIX*. Therefore, if σ' does not represent a computation, then the second or third conjunct of $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$ is false. We have considered all three conjuncts of $K_\sigma(\phi)$, so we conclude that if $K_\sigma(\phi)[\sigma/\sigma']$ is false then $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$ is also false, hence $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$ implies $K_\sigma(\phi)[\sigma/\sigma']$.

[$\Leftarrow$] If $K_\sigma(\phi)[\sigma/\sigma']$ then $\mathcal{M}[\![K_\Box(\phi)]\!][\sigma/\sigma']$:

[17] To understand the first conjunct, note that for any non-temporal formula ϕ, $\mathcal{M}[\![\phi]\!] = \phi[c_1,\ldots,c_m/(\sigma.0)_1,\ldots,(\sigma.0)_m] = \phi[\bar{c}/\sigma.0]$.

We must show that each of the three conjuncts of $\mathcal{M}[\![K_{\Box}(\phi)]\!][\sigma/\sigma']$ follows from $K_{\sigma}(\phi)[\sigma/\sigma']$. The first conjunct of $\mathcal{M}[\![K_{\Box}(\phi)]\!][\sigma/\sigma']$ is identical to the second conjunct of $K_{\sigma}(\phi)[\sigma/\sigma']$ and therefore follows directly. The second and third conjuncts of $\mathcal{M}[\![K_{\Box}(\phi)]\!]$ encode axioms *ORDERING* and *PREFIX*, respectively. By the definition of $K_{\sigma}(\phi)$, if $K_{\sigma}(\phi)[\sigma/\sigma']$ then σ' must represent a computation. Recall from Theorems 4.6 and 4.8 that *ORDERING* and *PREFIX* are sound, i.e. if σ' represents a computation then $\sigma' \models ORDERING \wedge PREFIX$. Therefore, if $K_{\sigma}(\phi)[\sigma/\sigma']$ then σ' represents a computation and consequently satisfies the second and third conjuncts of $\mathcal{M}[\![K_{\Box}(\phi)]\!]$. Since each of the three conjuncts of $\mathcal{M}[\![K_{\Box}(\phi)]\!][\sigma/\sigma']$ follows from $K_{\sigma}(\phi)[\sigma/\sigma']$, we conclude that $K_{\sigma}(\phi)[\sigma/\sigma']$ implies $\mathcal{M}[\![K_{\Box}(\phi)]\!][\sigma/\sigma']$. ⊠

Lemma 5.7 For any formula $\psi_{\overline{\Box}\circ}$ in $\mathrm{Tr}_{\overline{\Box}\circ}$, there exists a state-sequence σ' such that either

$\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and not $K_{\sigma}(true)[\sigma/\sigma']$, or

$K_{\sigma}(true)[\sigma/\sigma']$ and not $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$.

Proof: To prove this, we will need to consider the nesting depth of *Next* operators in formulas of $\mathrm{Tr}_{\overline{\Box}\circ}$. For any formula $\psi_{\overline{\Box}\circ}$ in $\mathrm{Tr}_{\overline{\Box}\circ}$, let $\circ\text{-}nesting(\psi_{\overline{\Box}\circ})$ denote the maximum nesting of *Next* operators in $\psi_{\overline{\Box}\circ}$. (For example, $\circ\text{-}nesting(\circ(\phi_1 \vee \circ\phi_2)) = 2$.) Since every formula $\psi_{\overline{\Box}\circ}$ has only a finite number of $\circ$'s, $\circ\text{-}nesting(\psi_{\overline{\Box}\circ})$ is a well-defined non-negative integer.

Now, consider an arbitrary $\psi_{\overline{\Box}\circ}$ in $\mathrm{Tr}_{\overline{\Box}\circ}$, and let $n = \circ\text{-}nesting(\psi_{\overline{\Box}\circ})$. Three cases must be considered. Case 3 is the general case, in which we use a σ' known to satisfy $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!]$ to construct a σ'' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma'']$ holds but $K_{\sigma}(true)[\sigma/\sigma'']$ does not. σ'' is constructed by rearranging states of σ' beyond state $\sigma'.n$ so that $K_{\sigma}(true)[\sigma/\sigma'']$ is false. However, $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma'']$ follows from $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ since $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!]$ can refer to states beyond $\sigma.n$ only by universal quantification. Cases 1 and 2 cover the situations in which there is no σ' satisfying $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!]$ or every such σ' has only repeating states after state $\sigma'.n$.

Case 1. There is no σ' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$:

We must then show that there is some σ' such that $K_{\sigma}(true)[\sigma/\sigma']$. One such σ' is the state-sequence in which every channel trace is always empty:

$$\sigma' = \langle\ [\langle\rangle, \ldots, \langle\rangle],\ [\langle\rangle, \ldots, \langle\rangle],\ [\langle\rangle, \ldots, \langle\rangle],\ \ldots\ \rangle$$

$\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ is false, since there there is no σ' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$, but, by Definition 5.4 of $K_{\sigma}(\phi)$, $K_{\sigma}(true)[\sigma/\sigma']$ is true.

Case 2. There exists a σ' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and every such σ' has only repeating states after state $\sigma'.n$. (Recall that $n = \circ\text{-}nesting(\psi_{\overline{\Box}\circ})$, so $n \geq 0$.) That is, for every σ' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and every $i > n$, $\sigma'.i = \sigma'.(i-1)$:

Consider an arbitrary σ' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$. If $K_\sigma(true)[\sigma/\sigma']$ is false, we are done. Suppose, then, that $K_\sigma(true)[\sigma/\sigma']$ is true. Construct σ'' from σ' by extending one trace by one element between states $\sigma'.n$ and $\sigma'.(n+1)$:

$$\sigma'' = \langle\, \sigma'.0,\ \sigma'.1,\ \ldots,\ \sigma'.n,\ \sigma'.n[(\sigma'.n)_1 \,/\, (\sigma'.n)_1 \cdot \langle v\rangle],\ \ldots \rangle$$

for an arbitrary $v \in V$. $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma'']$ is false, since $\sigma''.(n+1) \neq \sigma''.n$. But, by $K_\sigma(true)[\sigma/\sigma']$ and the definition of $K_\sigma(\phi)$, $K_\sigma(true)[\sigma/\sigma'']$ is true.

Case 3. There exists a σ' such that $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and σ' has a non-repeating state after $\sigma'.n$ (i.e. $\sigma'.i \neq \sigma'.(i-1)$ for some $i > n$):

Consider such a σ', and consider the smallest $i > n$ satisfying $\sigma'.i \neq \sigma'.(i-1)$. By definition,

$$\sigma' = \langle\, \sigma'.0,\ \sigma'.1,\ \ldots,\ \sigma'.n,\ \sigma'.(n+1),\ \ldots,\ \sigma'.(i-1),\ \sigma'.i,\ \ldots \rangle$$

such that for all k, $n \leq k < i$, $\sigma'.k = \sigma'.n$, but $\sigma'.i \neq \sigma'.n$. If $K_\sigma(true)[\sigma/\sigma']$ is false, we are done. Suppose, then, that $K_\sigma(true)[\sigma/\sigma']$ is true. Since $\sigma'.i \neq \sigma'.n$, there must exist a j and a v, $1 \leq j \leq m$, $v \in V$, such that $\sigma'.i = \sigma'.n[(\sigma'.n)_j \,/\, (\sigma'.n)_j \cdot \langle v\rangle]$. Let σ'' be constructed from σ' by repeating state $\sigma'.n$ and inserting a copy of state $\sigma'.i$ between the repetition:

$$\sigma'' = \langle\, \sigma'.0,\ \sigma'.1,\ \ldots,\ \sigma'.n,\ \sigma'.i,\ \sigma'.n,\ \sigma'.(n+1),\ \sigma'.(n+2),\ \ldots \rangle$$

Note that $\sigma''.(n+1) = \sigma'.i$ and $\sigma''.(n+2) = \sigma'.n$. Let x be the index of the last element in channel trace $(\sigma''.(n+1))_j$. (We know $(\sigma''.(n+1))_j$ is non-empty since $\sigma''.(n+1) = \sigma'.i = \sigma'.n[(\sigma'.n)_j \,/\, (\sigma'.n)_j \cdot \langle v\rangle]$.) $K_\sigma(true)[\sigma/\sigma'']$ is false because its third conjunct is contradicted (recall Definition 5.4): $(\sigma''.(n+1))_j.x = (\sigma'.i)_j.x = v$, but $(\sigma''.(n+2))_j.x = (\sigma'.n)_j.x$ is undefined. Therefore, by showing $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma'']$, the proof is complete.

Recall that we are assuming $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$. We prove $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma'']$ by structural induction on $\psi_{\overline{\Box}\circ}$.

Base Case:

- $\psi_{\overline{\Box}\circ} = p(t_1, \ldots, t_n)$: No temporal operators can appear in $p(t_1, \ldots, t_n)$, so by the definition of $\mathcal{M}$ (Table 1), the only references to σ in $\mathcal{M}[\![\, p(t_1, \ldots, t_n)]\!]$ are references to $\sigma.0$. Since $\sigma''.0 = \sigma'.0$, $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma'']$ follows from $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$.

Induction:

- $\psi_{\overline{\Box}\circ} = \psi_1 \vee \psi_2$: By $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and the definition of $\mathcal{M}$, $\mathcal{M}[\![\psi_1]\!][\sigma/\sigma']$ or $\mathcal{M}[\![\psi_2]\!][\sigma/\sigma']$. If $\mathcal{M}[\![\psi_1]\!][\sigma/\sigma']$, then by the induction hypothesis, $\mathcal{M}[\![\psi_1]\!][\sigma/\sigma'']$. If $\mathcal{M}[\![\psi_2]\!][\sigma/\sigma']$, then by the induction hypothesis, $\mathcal{M}[\![\psi_2]\!][\sigma/\sigma'']$. Therefore $\mathcal{M}[\![\psi_1]\!][\sigma/\sigma'']$ or $\mathcal{M}[\![\psi_2]\!][\sigma/\sigma'']$ and consequently $\mathcal{M}[\![\psi_1 \vee \psi_2]\!][\sigma/\sigma'']$.

- $\psi_{\overline{\Box}\circ} = \neg\psi$: By $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and the definition of $\mathcal{M}$, $\mathcal{M}[\![\psi]\!][\sigma/\sigma']$ is false. Then by the induction hypothesis, $\mathcal{M}[\![\psi]\!][\sigma/\sigma'']$ is false. Therefore $\mathcal{M}[\![\neg\psi]\!][\sigma/\sigma'']$.

- $\psi_{\overline{\Box}\circ} = (\exists x :: \psi)$: By $\mathcal{M}[\![\psi_{\overline{\Box}\circ}]\!][\sigma/\sigma']$ and the definition of $\mathcal{M}$, there exists a $v \in V$ such that $\mathcal{M}[\![\psi[x/v]]\!][\sigma/\sigma']$. Then by the induction hypothesis, there exists a $v \in V$ such that $\mathcal{M}[\![\psi[x/v]]\!][\sigma/\sigma'']$. Therefore $\mathcal{M}[\![(\exists x :: \psi)]\!][\sigma/\sigma'']$.

- $\psi_{\overline{\Box}\circ} = \overline{\Box}\,\psi$: We know $\mathcal{M}[\![\overline{\Box}\,\psi]\!][\sigma/\sigma']$. Therefore, by the definition of $\mathcal{M}$ and substitution, $\mathcal{M}[\![\psi]\!][\sigma/\sigma'[i..]]$ for all $i \geq 0$. Since $\overline{\Box}$ is the restricted version of *Always*, ψ contains no temporal operators. Thus the only references to σ in $\mathcal{M}[\![\psi]\!]$ are references to $\sigma.0$, and consequently $\mathcal{M}[\![\psi]\!][\sigma/[\sigma'[i..]]$ is equivalent to $\mathcal{M}[\![\psi]\!][\sigma.0/\sigma'.i]$. Now, by the definition of σ'', for all $\sigma''.k$, $k \geq 0$, there exists some $\sigma'.i$, $i \geq 0$ such that $\sigma''.k = \sigma'.i$. Therefore, $\mathcal{M}[\![\psi]\!][\sigma.0/\sigma''.k]$ for all $k \geq 0$ follows from $\mathcal{M}[\![\psi]\!][\sigma.0/\sigma'.i]$ for all $i \geq 0$. Hence $\mathcal{M}[\![\psi]\!][\sigma/[\sigma''[k..]]$ for all $k \geq 0$ and $\mathcal{M}[\![\overline{\Box}\,\psi]\!][\sigma/\sigma'']$.

- $\psi_{\overline{\Box}\circ} = \circ\,\psi$: We know $\mathcal{M}[\![\circ\,\psi]\!][\sigma/\sigma']$. Since $\circ$-*nesting*$(\circ\,\psi) \leq n$, by the definition of $\mathcal{M}$ every occurrence of σ in $\mathcal{M}[\![\circ\,\psi]\!]$ is either

  1. $\sigma.k$, for some $0 \leq k \leq n$, resulting from at most n nested *Next* operators, or
  2. $\sigma.(k+i)$, for some some universally quantified i and some $0 \leq k \leq n$, resulting from a $\overline{\Box}$ operator nested within at most n $\circ$ operators. (No temporal operators can be nested within $\overline{\Box}$, since $\overline{\Box}$ operates only over non-temporal formulas.)

  We need to show $\mathcal{M}[\![\circ\,\psi]\!][\sigma/\sigma'']$. Since $\mathcal{M}[\![\circ\,\psi]\!][\sigma/\sigma']$, we can prove $\mathcal{M}[\![\circ\,\psi]\!][\sigma/\sigma'']$ by showing that substitutions $[\sigma/\sigma']$ and $[\sigma/\sigma'']$ yield the same values for all occurrences of σ in $\mathcal{M}[\![\circ\,\psi]\!]$. Consider the two types of occurrences of σ, as defined above:

  1. $\sigma.k$, $0 \leq k \leq n$. By the definition of σ'', $\sigma''.k = \sigma'.k$ for all $0 \leq k \leq n$.
  2. $\sigma.(k+i)$, i universally quantified and $0 \leq k \leq n$. Under substitution $[\sigma/\sigma']$, the $\sigma.(k+i)$'s range over the set $S_{\sigma'} = \{\sigma'.k, \sigma'.(k+1), \sigma'.(k+2), \ldots\}$. Under substitution $[\sigma/\sigma'']$, the $\sigma.(k+i)$'s range over the set $S_{\sigma''} = \{\sigma''.k, \sigma''.(k+1), \sigma''.(k+2), \ldots\}$. By the definition of σ'', $S_{\sigma'} = S_{\sigma''}$ for every possible k, $0 \leq k \leq n$.

  Therefore, from $\mathcal{M}[\![\circ\,\psi]\!][\sigma/\sigma']$ we conclude $\mathcal{M}[\![\circ\,\psi]\!][\sigma/\sigma'']$. ⊠

Acknowledgements

We are grateful to Prakash Panangaden for many useful discussions regarding the results described in Section 5.

References

[Apt81] K.R. Apt. Ten years of Hoare's logic: a survey — part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, October 1981.

[BA81] J.D. Brock and W.B. Ackerman. Scenarios: a model of non-determinate computation. In *Formalization of Programming Concepts, Lecture Notes in Computer Science 107*, pages 252–259, Springer-Verlag, Berlin, 1981.

[Bro84] S.D. Brookes. A semantics and proof system for communicating processes. In *Logics of Programs, Lecture Notes in Computer Science 164*, pages 68–85, Springer-Verlag, Berlin, 1984.

[CH81] Z.C. Chen and C.A.R. Hoare. Partial correctness of communicating sequential processes. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 1–12, Paris, April 1981.

[Coo78] S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, February 1978.

[HH83] E.C.R. Hehner and C.A.R. Hoare. A more complete model of communicating processes. *Theoretical Computer Science*, 26:105–120, September 1983.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *Proceedings of the Fourth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 49–58, August 1985.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, March 1977.

[MC81] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(7):417–426, July 1981.

[MP81] Z. Manna and A. Pnueli. Verification of concurrent programs: the temporal framework. In R.S. Boyer and J.S. Moore, editors, *The Correctness Problem in Computer Science*, pages 215–273, International Lecture Series in Computer Science, Academic Press, London, 1981.

[MP82] Z. Manna and A. Pnueli. Verification of concurrent programs: a temporal proof system. In *Proceedings of the Fourth School on Advanced Programming*, pages 163–255, Amsterdam, June 1982.

[NDGO86] V. Nguyen, A. Demers, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. *Distributed Computing*, 1(1):7–25, January 1986.

[Ngu85] V. Nguyen. The incompleteness of Misra and Chandy's proof systems. *Information Processing Letters*, 21:93–96, August 1985.

[Sch67] J.R. Schoenfield. *Mathematical Logic.* Addison-Wesley, Reading, Massachusetts, 1967.

[WGS87] J. Widom, D. Gries, and F.B. Schneider. Completeness and incompleteness of trace-based network proof systems. In *Proceedings of the Fourteenth ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 27–38, January 1987.

[Wid87] J. Widom. *Trace-Based Network Proof Systems: Expressiveness and Completeness.* PhD thesis, Cornell University, Ithaca, New York, May 1987.

[Wol81] P. Wolper. Temporal logic can be more expressive. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, pages 340–348, October 1981.

[ZdRvEB85] J. Zwiers, W.P. de Roever, and P. van Emde Boas. 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 194*, pages 509–519, Springer-Verlag, Berlin, 1985.

[Zwi88] J. Zwiers. *Compositionality, Concurrency, and Partial Correctness: Proof Theories for Networks of Processes, and Their Connection.* PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, February 1988.