# Completeness and Incompleteness of Trace-Based Network Proof Systems*

Jennifer Widom
David Gries
Fred B. Schneider

TR 86-766

July 1986
(Revised September 1986)

Department of Computer Science
Cornell University
Ithaca, NY 14853

*This work was supported by the National Science Foundation under grant DCR-8329274. Schneider is also supported by Office of Naval Research Contract N0014-86-K-0092.

In *Proceedings of the Fourteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, January 1987, Munich, West Germany

# Completeness and Incompleteness of Trace-Based Network Proof Systems†

Jennifer Widom, David Gries, and Fred B. Schneider

*Computer Science Department*
*Cornell University*
*Ithaca, New York 14850*

**Abstract.** Most trace-based proof systems for networks of processes are known to be incomplete. Extensions to achieve completeness are generally complicated and cumbersome. In this paper, a simple trace logic is defined and two examples are presented to show its inherent incompleteness. Surprisingly, both examples consist of only one process, indicating that network composition is not a cause of incompleteness. Axioms necessary and sufficient for the relative completeness of a trace logic are then presented.

## 1. INTRODUCTION

Most formalisms for networks in which the specification of a network can be completely deduced from the specifications for its constituent processes are *trace-based*. In them, one specifies and reasons about traces (histories) of the values transmitted along the communication channels of the network. Trace-based proof systems are defined in [CH81, Ho81, Ho85, MC81], but unfortunately they exhibit incompleteness [BA81, Ng85]. Simple trace logics are modified to increase expressiveness in [Jo85, Pr82] and to obtain completeness in [BA81, HH83, NDGO86, ZRE84]. The modifications tend to be extensive and cumbersome; the simplicity of the underlying logic is lost.

This paper explores incompleteness in simple trace-based proof systems and identifies two extensions that are necessary and sufficient for achieving relative completeness. The first source of incompleteness is the inability to state and reason about constraints on the temporal 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. These two properties—the temporal ordering and prefix properties—must be available as reasoning tools in any (relatively) complete proof system.

The need for axiomatizations of these properties is illustrated using two examples, each consisting of a single process. The examples demonstrate that, while compositionality is an important feature of trace-based logics, incompleteness is caused not by network composition but by the inability to express the temporal ordering and prefix properties. We also prove that adding temporal ordering and prefix axioms to a trace logic suffices for achieving relative completeness.

Section 2 describes the class of synchronous process networks used in the remainder of the paper. In Section 3, we define Simple Trace Logic (*STL*), a formalism and proof system for network specification and verification that captures the essence of most trace-based systems. The incompleteness of *STL* is shown in Section 4. To reason about the proof system it is necessary to introduce a computational model; we do this in Section 5. The model is based on the *computation tree*, which captures all possible behaviors of a given process or network. In Section 6, the ideas discussed in Section 4 are formalized, providing axiomatizations of the temporal ordering and prefix properties, along with a proof of their necessity and sufficiency. Finally, in Section 7 we draw conclusions, explain how our results relate to existing proof systems, and discuss future work.

† This work was supported by the National Science Foundation under grant DCR-8320274. Schneider is also supported by Office of Naval Research contract N00014-86-K-0092.

## 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. 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. External channels permit communication with the environment of the network; input or output on an external channel occurs whenever the incident process is ready. Without loss of generality, we assume:

[2.0.1] Message transmission occurs instantaneously.

[2.0.2] Two message transmissions cannot occur simultaneously. Thus, there is a total order on the communication events of a given computation.

[2.0.3] 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 $P_1 \| P_2 \| \ldots \| P_n$, indicating the parallel execution of the component processes. Fig. 1 illustrates a network of three processes and eight communication channels.

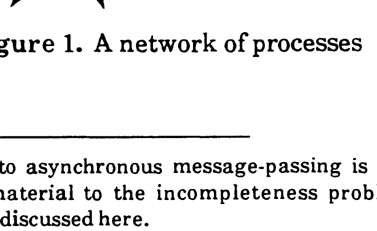


Figure 1. A network of processes

[1] Extension to asynchronous message-passing is straightforward, immaterial to the incompleteness problem, and therefore not discussed here.

## 3. SIMPLE TRACE LOGIC

Our formalism for specifying and verifying networks is called *Simple Trace Logic (STL)*. It concisely captures trace-based reasoning.

### 3.1. Channel Traces

A *specification* is a first-order predicate that is satisfied by every possible execution of the process or network it specifies. The predicate is defined over *channel traces*—the sequences of values transmitted on communication channels during execution.

Let $c$ be a channel. In a specification, $c$ denotes a finite sequence, $\langle c_0, c_1, \ldots, c_k \rangle$, indicating the values transmitted along channel $c$, in order. We use the following notational conventions:

- $\langle\,\rangle$ denotes the empty sequence.
- $|c|$ denotes the length of sequence $c$.
- $c1 \subseteq c2$ denotes that sequence $c1$ is a prefix of sequence $c2$. Note that $\subseteq$ is reflexive.

### 3.2. Process Specifications

A specification for a process $P$ is a predicate $S$ over the traces of $P$'s incoming and outgoing channels. We say that $P$'s behavior *satisfies* $S$, written $P$ **sat** $S$, if, at every point during any computation of $P$, the traces of the values transmitted on channels incident to $P$ satisfy $S$. For example, suppose process $P3$ of Fig. 1 repeatedly reads an integer from $c8$ and writes its successor to $c4$. We can formulate this in $STL$ as

[3.2.1] $P3$ **sat** $(|c8|-1 \le |c4| \le |c8|) \wedge$ $(\forall i\colon 0 \le i < |c4|\colon c4_i = c8_i + 1)$.

### 3.3. Network Specifications and Proof Rules

A specification for a network $N = P_1 \| P_2 \| \ldots \| P_n$ is also a predicate $S$ over the traces of its (internal and external) channels. $N$ **sat** $S$ if, given any behavior of $N$ up to any point in time, the traces of values transmitted along $N$'s channels satisfy $S$.

The axioms of $STL$ consist of all formulas $P$ **sat** $S$, where $S$ is a specification satisfied by every possible execution of process $P$. A specification of a network is to be based solely on specifications for its primitive component processes; how these primitive specifications are obtained—or even how processes are programmed—is not important. This puts $STL$ at a level of abstraction that hides all details except those relevant to the question of completeness.

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

[3.3.1] *Network Composition Rule*:

$$\frac{(\forall i\colon 1 \le i \le n\colon\ P_i \textbf{ sat } S_i)}{P_1 \parallel P_2 \parallel \dots \parallel P_n \textbf{ sat } \wedge_i S_i}$$

Conjoining specifications of processes using [3.3.1] results in "linking" any shared channels because in $\wedge_i S_i$, all $c$'s (say) refer to the same channel trace.

In addition, we have the following inference rule:

[3.3.2] *Consequence Rule*: $$\frac{N \textbf{ sat } S1,\ S1 \Rightarrow S2}{N \textbf{ sat } S2}$$

These two inference rules, or variants thereof, underlie all trace-based proof systems we know of, including [CH81, Ho85, MC81, NDGO86].

## 4. Incompleteness of Simple Trace Logic

Specification $S$ is *valid* for a process or network *PN* if every execution of *PN* (up to any point in time) yields channel traces that satisfy $S$. We would like *STL* to be *sound*—i.e. if we use *STL* to prove $N$ **sat** $S$, then indeed $S$ is valid for network $N$. A rigorous soundness proof requires a computational model [Ap81, CK73, Co78], which we give in Section 5.

We would also like *STL* to be *complete*—i.e. if, whenever some specification $S$ is valid for network $N$, then $N$ **sat** $S$ is provable using *STL*. However, a network specification is derived using [3.3.1] from specifications for its component processes. If these specifications are valid, but too weak, then we may not be able to prove a given valid network specification. Thus, what we really want to know is whether we can prove $N$ **sat** $S$ when the specifications given for the primitive processes comprising $N$ are as "strong" as possible.

[4.0.1] *Definition*: A specification $S$ is *precise* for a process or network *PN* iff:
   (1) $S$ is valid for *PN*, and
   (2) Any computation that satisfies $S$ is a possible computation of *PN*.

A precise specification for a process or network, then, exactly characterizes its possible computations. Hence, for completeness, we are merely interested in the provability of $N$ **sat** $S$ when $S$ is valid and the specifications for the processes in $N$ are precise.

*STL* specifications can involve elements of the data domain from which messages are drawn, sequences of such elements, and lengths of sequences. Since number theory itself is incomplete [S67], a valid assertion involving sequence lengths might not be provable in any system. When designing a programming logic, one actually aims for *relative completeness* [Co78]: Assuming that one can prove any valid statement of predicate logic, number theory, and the data domain of the network being considered, is the proof system complete?[2] *STL* is not relatively complete, as we now show.

### 4.1. Temporal Ordering Property

Consider the single-process network of Fig. 2. As an informal description of process $P$ we are

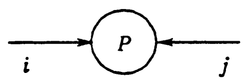


**Figure 2.** Single-process network

given four facts: (1) $P$ reads at most one value from channel $i$; (2) $P$ reads at most one value from channel $j$; (3) $P$ reads a value from $i$ before reading from $j$; (4) $P$ reads a value from $j$ before reading from $i$. A formal specification is

[4.1.1] $P$ **sat** $S1$: $|i| \le 1 \wedge |j| \le 1 \wedge |j| \le |i| \wedge |i| \le |j|$.

Let the data domain for this network be $\{a\}$. The following specification is valid for $P$ and is equivalent to [4.1.1]:

[4.1.2] $P$ **sat** $S2$: $(i=\langle\rangle \wedge j=\langle\rangle) \vee (i=\langle a\rangle \wedge j=\langle a\rangle)$

$P$ is always in one of two states: either no values have been read from $i$ and $j$ or one $a$ has been read from each. However, $P$ can reach a state in which $(i=\langle a\rangle \wedge j=\langle a\rangle)$ only if $i_0$ and $j_0$ are transmitted simultaneously. Since this cannot happen (by assumption [2.0.2]), $P$ can never read a value from $i$ or $j$. Therefore, a third valid specification for $P$ is

[4.1.3] $P$ **sat** $S3$: $i=\langle\rangle \wedge j=\langle\rangle$.

---

[2] Most proof systems make assumptions about both the provability of predicate logic statements and the expressiveness of the specification language involved. This is sometimes referred to as *Cook completeness* [Ap81, Co78]. We, too, have made an expressiveness assumption in our supposition that precise specifications for the component processes can be written in *STL*. The reader should convince himself that our language is powerful enough to express precise specifications for a large class of primitive processes.

All three specifications are valid and, in fact, precise. Any computation satisfying $S1, S2$, or $S3$ is a computation of $P$—no values are ever read on $i$ or $j$. However, consider an attempt at proving [4.1.3] given precise specification $S2$ (say) of [4.1.2]. Since there is only a single process, the network composition rule is irrelevant, and the only inference we can use is the consequence rule. But $S2 \Rightarrow S3$ does not hold. Hence [4.1.3] is unprovable, even though it is valid.

We need a way to formalize the reasoning about event ordering used to obtain [4.1.3]. It must assert the following

[4.1.4] *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) $c1_x$ must be transmitted before $c2_y$, and
(2) $c2_y$ must be transmitted before $c1_x$.

Then $(|c1| \leq x \wedge |c2| \leq y)$ holds throughout any computation of $N$—neither message will be transmitted.

Property [4.1.4] allows $S3$ to be deduced from $S2$, making [4.1.3] provable.

### 4.2. Prefix Property

Consider a network with one process and one communication channel (see Fig. 3). Suppose the

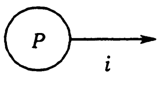

Figure 3. Simplest network of all

network has $\{a, b\}$ as its data domain. Let a precise specification for process $P$ be

[4.2.1] $P$ **sat** $S4$: $i = \langle\rangle \vee i = \langle a \rangle \vee i = \langle b, a \rangle$.

Since $P$ can send only one value at a time on channel $i$, $i = \langle b, a \rangle$ can never be attained—it would be reachable only from $i = \langle b \rangle$, which is prohibited by $S4$. Therefore, [4.2.1] can be simplified to

[4.2.2] $P$ **sat** $S5$: $i \subseteq \langle a \rangle$.

However, $S4$ does not imply $S5$, and therefore [4.2.2] cannot be proved from precise specification [4.2.1]. Here, we need:

[4.2.3] *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 applying the prefix property to $S4$, we can eliminate the disjunct $i = \langle b, a \rangle$ and obtain [4.2.2].

### 4.3. Augmenting the Proof System

Consider any $STL$ proof that establishes $N$ **sat** $S$ for a network $N = P_1 \parallel P_2 \parallel \ldots \parallel P_n$. As axioms, we are given $P_1$ **sat** $S_1$, $P_2$ **sat** $S_2$, ..., $P_n$ **sat** $S_n$, where $S_1, S_2, \ldots, S_n$ are precise. The first rule to be applied in any such proof is necessarily the network composition rule, so we immediately obtain $N$ **sat** $\bigwedge_i S_i$. (In Section 5 we show that $\bigwedge_i S_i$ is in fact a precise specification for $N$.) All remaining steps in the proof must then be applications of the consequence rule. Since any string of consequence rule applications can be collapsed into one, we see that $N$ **sat** $S$ is provable if and only if $\bigwedge_i S_i \Rightarrow S$, a formula of predicate logic. The two examples given, however, demonstrate that such an implication might not hold. By strengthening the antecedent, we can guarantee that the implication will be valid. Thus, we must find a set of axioms such that if $A$ (say) is the conjunction of the axioms in the set, then $(\bigwedge_i S_i \wedge A) \Rightarrow S$ is valid whenever it should be possible to deduce $S$ from $\bigwedge_i S_i$. The temporal ordering and prefix properties are the basis for such a set of axioms.

The remainder of the paper is a formalization of the concepts and results presented thus far.

## 5. Computational Model

Proving soundness and (relative) completeness requires a model of network behavior [Ap81, CK73, Co78]. The model is used to formalize the notions of valid and precise specifications. We can then prove that $STL$ is sound, we can show that the conjunction of precise process specifications results in a precise network specification, and, most importantly, we can formalize the temporal ordering and prefix properties, allowing us to prove that they are necessary and sufficient for relative completeness.

Our model is based on the *computation tree*. Every process or network is represented by one computation tree. The structure of the tree describes all and only potential execution sequences of the process or network; vertices, called *trace-sets*, are sets of communication channel traces, and edges represent a single step of execution. In all computation trees

[5.0.1] The root of the tree is the trace-set in which all channel traces are empty, corresponding to the initial state of a computation.

[5.0.2] The children of a trace-set $TS$ within the computation tree are exactly those trace-sets that extend one channel trace of $TS$ by one element, where the extension corresponds to a communication event that might actually be performed.

Internal computations of a process are irrelevant when reasoning about network behavior, except as they affect the values sent and received. Thus the tree does not include such changes of process state. Since our system allows for reasoning about both finite and infinite computations, trees can be of finite or infinite depth. The domain of communicable values corresponds to the breadth of a tree; it too can be finite or infinite. (There is some similarity here to the *CCS* synchronization tree [Mi80].)

We first describe computation trees for primitive processes and then show how a computation tree for a network is built from trees for its component processes.

### 5.1. Computation Trees for Processes

The behavior of a process $P$ is modeled as a computation tree. As an example, consider the network of Fig.4. *MERGE* repeatedly and

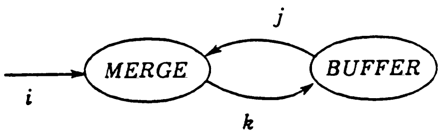


Figure 4. Example network

nondeterministically reads a value from $i$ or $j$ and then writes it on $k$. *BUFFER* simply copies values from $k$ to $j$, with an arbitrary amount of internal buffering. Let the data domain for the network be $\{a\}$. The initial portions of the computation trees for *MERGE* and *BUFFER* are illustrated in Figs. 5 and 6.

### 5.2. Computation Trees for Networks

The computation tree for a network is defined in terms of the computation trees for the network's constituent processes.[3] First, we define compatibility of trace-sets—the criteria for determining when a group of trace-sets from process computation trees can coexist and hence can be combined into a single trace-set of a network

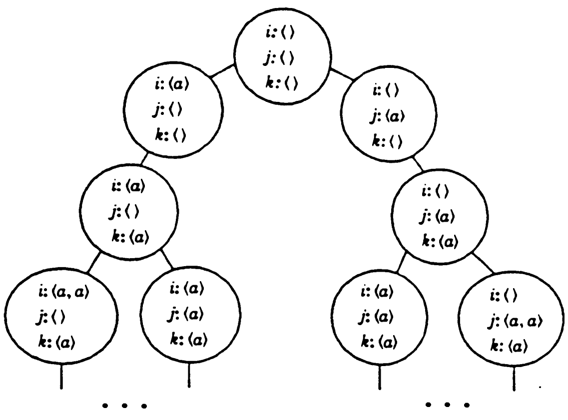


Figure 5. Computation tree for process *MERGE*

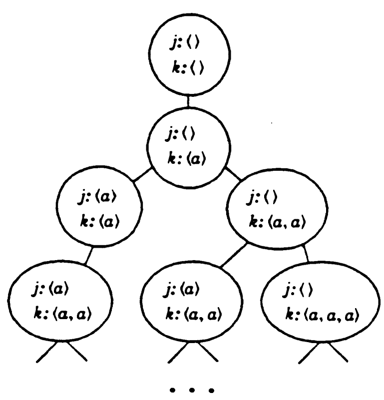


Figure 6. Computation tree for process *BUFFER*

computation tree. Let $TS_1$, $TS_2$, ..., $TS_n$ be trace-sets, one each from the computation trees for processes $P_1, P_2, ..., P_n$ of a network. This group of trace-sets is *compatible* iff for all channels $c$ such that a trace of $c$ appears in both $TS_i$ and $TS_j$, the trace of $c$ in $TS_i$ is identical to the trace of $c$ in $TS_j$. Thus, trace-sets are compatible when the exact same transmissions have occurred on any channels they have in common. When an appropriate set of compatible trace-sets is identified (the identification procedure is described shortly), they are merged into a single trace-set of the network tree being constructed. Merging compatible trace-sets simply consists of forming their union.

Let $T_1$, $T_2$, ..., $T_n$ be the computation trees for processes $P_1$, $P_2$, ..., $P_n$ respectively, and let $N = P_1 \parallel P_2 \parallel ... \parallel P_n$. The tree $T$ for network $N$ is defined by the following construction:

[3] We could alternatively—and equivalently—have chosen to define network trees independently of the component process trees, but the constructive definition given here is both illustrative of the model and useful in subsequent proofs.

[5.2.1] $Combine(T_1, T_2, ..., T_n) \equiv$

the root of $T$ = the result of merging the roots of $T_1, T_2, ..., T_n$;

for each $T_i$, $1 \leq i \leq n$:

let $G_i$ be the group of trace-sets consisting of the root of $T_i$ and all the root's children;

consider every possible group of trace-sets, $G$, where $G$ is constructed by choosing one trace-set from each $G_i$. $G$ is *usable* if:

(1) the trace-sets in $G$ are compatible, and

(2) merging the trace-sets in $G$ results in a new trace-set that extends exactly one trace of $T$'s root by exactly one element;

for each usable $G$:

add a child to the root of $T$, letting this trace-set be the root of the tree defined by *Combine*(set of subtrees whose roots are the trace-sets in $G$).

In each invocation of *Combine*, one set of process tree trace-sets is merged into a single network tree trace-set, followed by the identification of all possible trace-sets the network can achieve in some "next step". The recursive definition then results in the complete network tree, even if some or all of the process trees are infinite (the resulting network tree need not also be infinite). Fig. 7 shows the initial part of the network tree for *MERGE* ‖ *BUFFER*, obtained by combining the process trees pictured in Figs. 5 and 6.

### 5.3. Valid and Precise Specifications

We are now ready to define the relationship between *STL* and the computation-tree model. Define a *path* in a computation tree to be any connected sequence of trace-sets beginning with the root and descending through the tree until a trace-set with no children is reached. (If no terminal trace-set is reached then the path is an infinite sequence.) A path corresponds to a computation of the process or network being modeled by the computation tree. For any process or network *PN*, define *Comps(PN)*, the set of possible computations, to be the set of all paths in the computation tree for *PN*.

Denote any sequence of trace-sets by $\sigma = \langle \sigma_0, \sigma_1, \sigma_2, ... \rangle$. A specification $S$ is valid for a process or network *PN* if

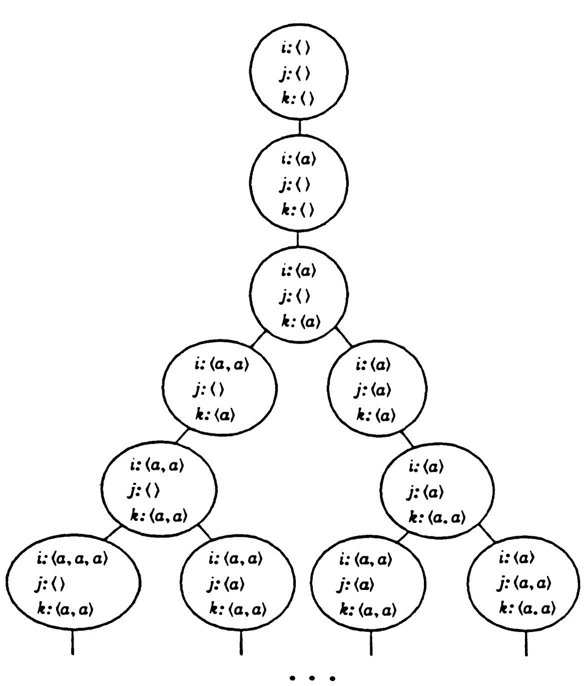


**Figure 7.** Computation tree for *MERGE* ‖ *BUFFER*

[5.3.1] $(\forall \sigma: \sigma \in Comps(PN)$:
$(\forall i: 0 \leq i < |\sigma|: \sigma_i \models S))$.[4]

That is, $S$ is valid for *PN* if every trace-set of every sequence in *Comps(PN)* satisfies $S$. For notational convenience we define an "always" operator , $\Box$:

[5.3.2] $\sigma \models \Box S$ iff $(\forall i: 0 \leq i < |\sigma|: \sigma_i \models S)$.[5]

Definition [5.3.1] of validity can now be written as $(\forall \sigma: \sigma \in Comps(PN): \sigma \models \Box S)$, and we can establish the soundness of *STL*.

[5.3.3] *Theorem (soundness of STL)*: Let $N$ be a network and $S$ a specification such that $N$ **sat** $S$ is provable using *STL*. Then $S$ is valid for $N$.

*Proof*: See appendix.

A sequence of trace-sets is well-formed if it could appear as a path in the computation tree for some process or network because the sequence does not violate [5.0.1] or [5.0.2]. More formally:

---

[4] $\sigma_i \models S$ holds if the channel traces in $\sigma_i$ satisfy $S$.

[5] This version of $\Box$ is consistent with the operator $\Box$ ("henceforth") in temporal logic, see e.g. [MP81]. The temporal logic operator is defined as: $\sigma \models \Box S$ iff $(\forall i: 0 \leq i < |\sigma|: \langle \sigma_i, \sigma_{i+1}, ... \rangle \models S)$, but when $S$ itself contains no temporal operators, then $(\langle \sigma_i, \sigma_{i+1}, ... \rangle \models S) \equiv (\sigma_i \models S)$.

[5.3.4] *Definition*: $\sigma$ is *well-formed* iff:

(1) All channel traces in the initial trace-set of $\sigma$ are empty, and

(2) Each trace-set in $\sigma$, except the first, extends exactly one trace of the preceding set by exactly one element.

We can now formalize Definition [4.0.1] of a precise specification.

[5.3.5] *Definition*: A specification $S$ is *precise* for a process or network $PN$ iff:

(1) $S$ is valid for $PN$, and

(2) Any well-formed sequence of trace-sets $\sigma$ satisfying $\Box S$ is in $Comps(PN)$.

(In part (2) of [5.3.5] we tacitly assume that the trace-sets of $\sigma$ do not include extraneous channel traces—i.e. that all traces in $\sigma$ are histories of channels actually appearing in $PN$.) It turns out that the composition of precise process specifications results in a network specification that is also precise.

[5.3.6] *Theorem (preciseness preservation)*: Let $S_i$ be a precise specification for $P_i$, $1 \leq i \leq n$, and let $N = P_1 \parallel P_2 \parallel \ldots \parallel P_n$. Then $\wedge_i S_i$ is a precise specification for $N$.

*Proof*: See appendix.

## 6. The Temporal Ordering and Prefix Axioms

Consider a network $N = P_1 \parallel P_2 \parallel \ldots \parallel P_n$. Given precise specifications $S_1, S_2, \ldots, S_n$ for the component processes, $N$ **sat** $S$ is provable if and only if $\wedge_i S_i \Rightarrow S$. We now know, by preciseness-preservation theorem [5.3.6], that $\wedge_i S_i$ is a precise specification for $N$. Therefore, $STL$ would be relatively complete if $S1 \Rightarrow S2$ whenever $S1$ is a precise specification for a network $N$ and $S2$ is a valid specification for $N$. The examples of Section 4 showed that the implication does not always hold and suggested that we define a set of axioms whose conjunction $A$ guarantees that $(S1 \wedge A) \Rightarrow S2$. We will prove that axiomatizations of the temporal ordering and prefix properties (from Section 4) are necessary and sufficient for such an $A$.

There is a fundamental difference between any axiomatization of temporal ordering and specifications $S1$ and $S2$, because event ordering is always with respect to an entire computation—a sequence of trace-sets—while $S1$ and $S2$ are with respect to individual trace-sets. We employ $\Box$ to convert a specification to being on entire computations and introduce

[6.0.1] *Revised Consequence Rule*:

$$\frac{N \textbf{ sat } S1,\ \Box S1 \wedge A \Rightarrow \Box S2}{N \textbf{ sat } S2}$$

### 6.1. The Temporal Ordering Axiom

Our first axiom characterizes temporal ordering property [4.1.4]. If some communication $c1_x$ happens before some $c2_y$, then $|c2|$ cannot exceed $y$ until $|c1|$ exceeds $x$. This can be expressed as $\Box(|c2|>y \Rightarrow |c1|>x)$. Note that this assertion captures temporal precedence 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 only interested in temporal 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 excluded. Now, if $\Box(|c1|>x \Rightarrow |c2|>y)$ as well, then neither $c1_x$ nor $c2_y$ can ever happen, equivalently: $\Box(|c1| \leq x \wedge |c2| \leq y)$.

The formalization differs slightly from the preceding discussion, however. All $>$'s are changed to $\geq$'s in the antecedent of the rule and all $\leq$'s are changed to $<$'s in the consequent. Doing so allows channel traces of length 0 in the antecedent, thereby asserting that an empty channel trace temporally precedes all communication events on that channel. Hence we state the temporal ordering axiom as

[6.1.1] *ORDERING*: If $c1$ and $c2$ are channels, $x \geq 1$ and $y \geq 0$ are indices, and either $x \neq y$ or $c1$ and $c2$ are distinct, then $\Box(|c1| \geq x \equiv |c2| \geq y) \Rightarrow \Box(|c1| < x \wedge |c2| < y)$.

We require $x \geq 1$, rather than $x \geq 0$, because allowing $x=y=0$ results in a pathological situation in which the antecedent is trivially true (since trace lengths are always at least 0), but the consequent is trivially false (since lengths cannot be less than 0).

We must prove that the axiom is sound.

[6.1.2] *Theorem (soundness of ORDERING)*: $\sigma \models ORDERING$ for any well-formed sequence of trace-sets $\sigma$.

*Proof*: See appendix.

### 6.2. The Prefix Axiom

An additional bit of notation is necessary in order to formulate an axiom for prefix property [4.2.3]. For any $i \geq 0$ and trace-set sequence $\sigma$, let

$\circ c$ ("the next value of $c$") be defined with respect to trace-set $\sigma_i$ as the trace of channel $c$ in trace-set $\sigma_{i+1}$.[6] If $\sigma$ is finite, in the last trace-set let $\circ c = c$ (since there is no next trace-set). In effect, we convert finite sequences to infinite ones by repeating the final trace-set. Thus, for any sequence $\sigma$, every channel $c$ appearing in $\sigma$ has a corresponding and well-defined value $\circ c$ in each trace-set of the sequence. Intuitively, the value of $\circ c$ at any given time is the value that channel trace $c$ will have after the next computation step.

We now state the prefix axiom.

[6.2.1] *PREFIX*:
If $c$ is any channel, then: $\Box(c \subseteq \circ c)$.

The axiom asserts that the value of a channel trace $c$ at any point in time is a prefix of $c$'s trace at any later time. The axiom is thus equivalent to the prefix property as stated in Section 4.2.

[6.2.2] *Theorem (soundness of PREFIX)*:
$\sigma \models PREFIX$ for any well-formed sequence of trace-sets $\sigma$.

*Proof*: Let $\sigma$ be any well-formed sequence of trace-sets. $\sigma \models PREFIX$ follows directly from the definition of well-formedness: Since $\sigma_{i+1}$ extends exactly one trace of $\sigma_i$ by exactly one element (for all $0 \leq i < |\sigma| - 1$), every channel trace $c$ in $\sigma_i$ is a prefix of the corresponding trace in $\sigma_{i+1}$. If $i = |\sigma| - 1$, then by definition $c = \circ c$. Therefore *PREFIX* is a sound axiomatization of the prefix property. ⊠

### 6.3. Necessity and Sufficiency of the Axioms

By letting $A = ORDERING \wedge PREFIX$, we can prove that if *S1* is a precise specification for network $N$ and *S2* is a valid specification for $N$, then $\Box S1 \wedge A \Rightarrow \Box S2$. In addition, we will argue that *ORDERING* and *PREFIX* are necessary axioms for this—if either axiom is removed from $A$ then we can find a network $N$ with precise and valid specifications *S1* and *S2* (respectively) such that $\Box S1$ and $A$ do not imply $\Box S2$. We begin with a key lemma.

[6.3.1] *Lemma (well-formedness)*: Let $\sigma$ be any sequence of trace-sets. Define *Compress*($\sigma$) to be the sequence obtained from $\sigma$ by eliminating each trace-set that duplicates its immediate predecessor in the sequence. *Compress*($\sigma$) is well-formed if and only if $\sigma \models ORDERING \wedge PREFIX$.

*Proof*: See appendix.

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

[6.3.2] *Theorem (sufficiency of the axioms)*: If *S1* is a precise specification for network $N$ and *S2* a valid specification for $N$, then $\Box S1 \wedge ORDERING \wedge PREFIX \Rightarrow \Box S2$.

*Proof*: We show that any sequence of trace-sets $\sigma$ satisfying $\Box S1$, *ORDERING*, and *PREFIX*, also satisfies $\Box S2$. Since $\sigma \models ORDERING \wedge PREFIX$, by Lemma [6.3.1] we know that *Compress*($\sigma$) is well-formed. Furthermore, $Compress(\sigma) \models \Box S1$ since $\sigma \models \Box S1$. (Note that for any trace-set $\sigma$ and specification $S$, $\sigma \models \Box S$ iff $Compress(\sigma) \models \Box S$.) Now recall from the formal definition of preciseness ([5.3.5]) that any well-formed sequence satisfying a precise specification is a path in the computation tree for the corresponding process or network. Thus, since *S1* is precise, $Compress(\sigma) \in COMPS(N)$. By the validity of *S2*, every sequence in *COMPS*($N$) satisfies $\Box S2$. Hence $Compress(\sigma) \models \Box S2$ and consequently $\sigma \models \Box S2$. ⊠

Thus with *ORDERING* and *PREFIX*, we ensure that any valid network specification follows from a precise specification for the network. (In fact, by preciseness-preservation theorem [5.3.6], only precise specifications for the component processes are needed.) Both axioms are necessary for the implication to always hold, as well as sufficient, as is shown in our final theorem:

[6.3.3] *Theorem (necessity of the axioms)*: There exist networks *N1*, *N2*, and *N3*, with precise specifications $S1_P$, $S2_P$, $S3_P$ (respectively) and valid specifications $S1_V$, $S2_V$, $S3_V$ (respectively), such that:

(1) $\neg(\Box S1_P \wedge ORDERING \Rightarrow \Box S1_V)$
(2) $\neg(\Box S2_P \wedge PREFIX \Rightarrow \Box S2_V)$
(3) $\neg(\Box S3_P \Rightarrow \Box S3_V)$

*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). ⊠

[6] Operator $\circ$ corresponds to the "next" operator of temporal logic [MP81]. Do not confuse this with a second use of $\circ$ in temporal logic, where $\circ$ operates over formulas: $\sigma_i \models \circ S$ iff $\sigma_{i+1} \models S$.

## 7. Conclusions, Comparisons, and Future Work

*STL* is a simple trace-based proof system for networks of processes, with specification language and inference rules similar to those in most existing trace logics [Br84, CH81, HH83, Ho81, Ho85, Jo85, MC81, Mi80, NDGO86, ZRE84]. Like other simple trace logics [CH81, Ho81, Ho85, MC81], *STL* is incomplete, and we have proved that axiomatizations of the temporal ordering and prefix properties are necessary to achieve relative completeness. Since these two axioms are essential components of a relatively complete proof system, it is interesting to look at existing complete systems and identify how the axioms are represented.

Several proof systems involve explicit reasoning about every possible interleaving of communication events [Br84, HH83, Mi80]; within the system all possible computations must actually be listed. It is clear that such a logic will be complete, since an exhaustive list of potential computations is an exact characterization of process or network behavior, including (implicitly) the constraints of the temporal ordering and prefix properties. Naturally, the difficulty is the exponential number of possible computations. Verifying the specification of any but very simple networks could be a formidable task with such a formalism.

The proof system in [ZRE84] is designed both for the specification of sequential processes and for the verification of their behavior when connected into a network. Thus, Hoare-style triples and inference rules are given (in the style of [AFR80, LG81]), as well as a means for reasoning about specifications over channel traces. The logic includes a statement of the prefix property, written essentially as $\{ \mathrm{Tr}=c \}$ *Pgm* $\{\mathrm{Tr} \subseteq c \}$, where *Pgm* is any program segment. (The interpretation is: If execution of *Pgm* begun in any state in which channel trace $c$ has value Tr terminates, then upon termination Tr is a prefix of $c$.) Reasoning about the temporal ordering property, however, is achieved only by enumerating all possible interleavings of the communication events of interest. Again, this can result in an exponential number of cases to consider.

In [ZRE84], the authors also discuss the incompleteness of [MC81] and suggest a rule that would render it relatively complete. (A similar rule is proposed in [Ng85].) Informally, the rule asserts the following: Let $S$ be a valid specification for network $N$ and $t$ be an interleaved trace of all communication events during any computation of $N$. Then every prefix of $t$ satisfies $S$. This rule certainly captures the prefix property, and the temporal ordering property is encoded as well. To see this, suppose specification $S$ constrains two communication events $c1_x$ and $c2_y$ (say) to occur simultaneously. Any trace $t$ including only one of $c1_x$ and $c2_y$ will not satisfy $S$, and thus cannot be a computation of $N$. Suppose, then, that both events are included in $t$. Consider any prefix $p$ of $t$ that contains one event but not the other. (Such a prefix must exist.) Then $p$ will not satisfy $S$, since only one of $c1_x$ and $c2_y$ appears in $p$. Hence no computation of $N$ can include either event.

In [Jo85], the fact (and problem) 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

$$\frac{N \textbf{ sat } S1}{N \textbf{ sat } S2},$$

which can be used whenever $S1$ and $S2$ are such that any network satisfying $S1$ will also satisfy $S2$. With a rule of inference like this, the issues of behavioral properties such as temporal ordering can essentially be ignored, but consequently there is no formal method for deciding when a pair of specifications is a candidate for an application of the above rule.

The proof system of [NDGO86] is based on temporal logic, so it is straightforward to formulate ordering constraints between network events in the logic. In addition, a number of axioms for behaviors are defined, including assertions that all traces are initially empty, that only one communication event can occur in a single time-step, that the prefix property holds, etc. These axioms for behaviors are also stated in temporal logic.

Our *ORDERING* and *PREFIX* axioms could be formulated in temporal logic, since the operators $\Box$ and $\circ$ are subsumed by the corresponding operators of temporal logic. However, we have actually drawn upon only a relatively small subset of temporal logic. In particular, we use $\circ c$, but do not need the formula version of $\circ$; we use $\Box S$, but only in the special case when $S$ is non-temporal. Although temporal logic is a convenient language in which to perform the types of reasoning needed for our axioms, temporal logic may be far more powerful than is necessary. Our contribution here is to identify the subset of temporal logic needed to achieve relative completeness.

The next step in our work is to extend the language of $STL$ to enable our two axioms to be expressed. Our goal is to create as simple a trace logic as possible, but one that is still relatively complete. Since we have shown that *ORDERING* and *PREFIX* are necessary and sufficient property axiomatizations, they will be our guide in devising such a proof system.

## Appendix

[5.3.3] *Theorem (soundness of STL)*: Let $N$ be a network and $S$ a specification such that $N$ **sat** $S$ is provable using $STL$. Then $S$ is valid for $N$.

*Proof*: Since we're assuming validity of process specifications, proving this theorem consists of showing that whenever the antecedent of an $STL$ inference is valid, so is the consequent.

[3.3.1] Network Composition Rule:

$$\frac{(\forall i: 1 \le i \le n:\ P_i \textbf{ sat } S_i)}{P_1 \parallel P_2 \parallel \ldots \parallel P_n \textbf{ sat } \wedge_i S_i}$$

Assume each $S_i$ is valid for $P_i$, so ($\forall \sigma$: $\sigma \in Comps(P_i)$: $\sigma \models \Box S_i$). We must show that ($\forall \sigma$: $\sigma \in Comps(N)$: $\sigma \models \Box \wedge_i S_i$), where $N = P_1 \parallel P_2 \parallel \ldots \parallel P_n$. Consider an arbitrary conjunct $S_i$ and an arbitrary $\sigma \in Comps(N)$. Let $\sigma_j$ be any trace-set of $\sigma$. If we construct $\sigma_j'$ by removing from $\sigma_j$ all traces of channels that are not incident to process $P_i$ then—by the method of constructing network trees from component process trees—we obtain a trace-set that must appear in some $\sigma \in Comps(P_i)$. Therefore, $\sigma_j' \models S_i$, because $S_i$ is valid, and $\sigma_j \models S_i$ as well, since the traces that were removed from $\sigma_j$ cannot appear in $S_i$. Since $\sigma_j$ is an arbitrary trace-set of an arbitrary sequence in $Comps(N)$, we know ($\forall \sigma$: $\sigma \in Comps(N)$: $\sigma \models \Box S_i$). The conjunct $S_i$ was also chosen arbitrarily, so we can conclude that ($\forall \sigma$: $\sigma \in Comps(N)$: $\sigma \models \Box \wedge_i S_i$). Thus $\wedge_i S_i$ is valid for $N$.

[3.3.2] Consequence Rule: $\dfrac{N \textbf{ sat } S1,\ S1 \Rightarrow S2}{N \textbf{ sat } S2}$

Let $S1$ be valid for $N$. From ($\forall \sigma$: $\sigma \in Comps(N)$: $\sigma \models \Box S1$) and $S1 \Rightarrow S2$, by predicate logic we conclude ($\forall \sigma$: $\sigma \in Comps\ (N)$: $\sigma \models \Box S2$). Therefore $S2$ is also valid for $N$. ⊠

[5.3.6] *Theorem (preciseness preservation)*: Let $S_i$ be a precise specification for $P_i$, $1 \le i \le n$, and let $N = P_1 \parallel P_2 \parallel \ldots \parallel P_n$. Then $\wedge_i S_i$ is a precise specification for $N$.

*Proof*: We must show that $\wedge_i S_i$ satisfies both parts of Definition [5.3.5].

(1) ($\wedge_i S_i$ is valid for $N$.) Since the $S_i$ are precise specifications for their respective $P_i$, they are valid. We must then show that $\wedge_i S_i$ is valid for $N$. This was proven in part (1) of Theorem [5.3.3] (the soundness theorem).

(2) (If $\sigma$ is any well-formed sequence of trace-sets such that $\sigma \models \Box \wedge_i S_i$, then $\sigma \in Comps(N)$.) For any process $P$, define $Project(\sigma, P)$ to be the sequence of trace-sets $\sigma'$ that results from restricting the trace-sets in $\sigma$ to those channels that are incident to $P$ and then eliminating each trace-set that duplicates its immediate predecessor in the sequence. Using $Project(\sigma, P)$ we can take a path representing a computation of a network and extract the trace-set sequence that shows how a single process behaves during this computation. Now, let $\sigma$ be any well-formed sequence of trace-sets such that $\sigma \models \Box \wedge_i S_i$. We must show that $\sigma \in Comps(N)$. Let $\sigma_1 = Project(\sigma, P_1)$, $\sigma_2 = Project(\sigma, P_2)$, etc. By definition, $\sigma_i \models \Box S_i$, $1 \le i \le n$. Thus, by the preciseness of each of the $S_i$, $\sigma_i \in Comps(P_i)$. Lastly, we use the algorithm for network tree construction to conclude that $\sigma \in Comps(N)$. ⊠

[6.1.2] *Theorem (soundness of ORDERING)*: If $\sigma$ is any well-formed sequence of trace-sets, then $\sigma \models ORDERING$.

*Proof*: Let $\sigma$ be an arbitrary well-formed sequence of trace-sets. We must show that if $\sigma \models \Box(|c1| \ge x \equiv |c2| \ge y)$ then $\sigma \models \Box(|c1| < x \wedge |c2| < y)$. Assume that $\Box(|c1| \ge x \equiv |c2| \ge y)$ holds for $\sigma$, and suppose, for the sake of a contradiction, that $\Box(|c1| < x \wedge |c2| < y)$ does not. Thus, there is a trace-set of $\sigma$ in which $(|c1| \ge x \vee |c2| \ge y)$. Let $i$ be the smallest index for which this is true: $(|c1| \ge x \vee |c2| \ge y)$ is true in $\sigma_i$, but does not hold in any $\sigma_j$ for $j < i$. Since $(|c1| \ge x \vee |c2| \ge y)$ is true in $\sigma_i$, by $\sigma \models \Box(|c1| \ge x \equiv |c2| \ge y)$ we know that $(|c1| \ge x \wedge |c2| \ge y)$ holds in $\sigma_i$. By $x \ge 1$ (recall Definition [6.1.1]), $i > 0$, since all traces in $\sigma_0$ are empty. So consider trace-set $\sigma_{i-1}$. By the definition of a well-formed sequence, $\sigma_i$ extends exactly one trace of $\sigma_{i-1}$ by exactly one element. Therefore since $(|c1| \ge x \wedge |c2| \ge y)$ holds in $\sigma_i$, $(|c1| \ge x \vee |c2| \ge y)$ must hold in $\sigma_{i-1}$. This contradicts the assumption that $i$ is the smallest index for which $\sigma_i \models (|c1| \ge x \vee |c2| \ge y)$. Thus, $\sigma \models \Box(|c1| < x \wedge |c2| < y)$ and $\sigma \models ORDERING$. ⊠

[6.3.1] *Lemma (well-formedness)*: Let $\sigma$ be any sequence of trace-sets. Define $Compress(\sigma)$ to be the sequence obtained from $\sigma$ by eliminating each trace-set that duplicates its immediate predecessor in the sequence. *Compress*($\sigma$) is well-formed if and only if $\sigma \models ORDERING \wedge PREFIX$.

*Proof*: [$\Rightarrow$] (If *Compress*($\sigma$) is well-formed then $\sigma \models ORDERING \wedge PREFIX$.) Since *Compress*($\sigma$) is well-formed, by the soundness of *ORDERING* and *PREFIX* we know *Compress*($\sigma$) $\models$ $ORDERING \wedge PREFIX$. Repetition of trace-sets cannot invalidate *ORDERING* or *PREFIX*, hence $\sigma \models ORDERING \wedge PREFIX$.

[$\Leftarrow$] (If $\sigma \models ORDERING \wedge PREFIX$ then *Compress*($\sigma$) is well-formed.) Consider any $\sigma$ that satisfies *ORDERING* and *PREFIX*; we must show that *Compress*($\sigma$) is well-formed. We prove the (equivalent) contrapositive: If *Compress*($\sigma$) is not well-formed, then $\sigma$ does not satisfy $ORDERING \wedge PREFIX$. In fact, it suffices to show that *Compress*($\sigma$)—rather than $\sigma$ itself—does not satisfy $ORDERING \wedge PREFIX$. That $\sigma$ does not satisfy $ORDERING \wedge PREFIX$ then follows by the fact that repetition of trace-sets cannot cause *ORDERING* or *PREFIX* to become valid. Thus, without loss of generality, we need only show that any $\sigma$ that is not well-formed and has no repeated trace-sets does not satisfy $ORDERING \wedge PREFIX$. By Definition [5.3.4] of well-formedness, such a $\sigma$ must exhibit at least one of the following conditions:

[A.1] In the initial trace-set all channel traces are not empty.
[A.2] Some channel trace decreases in length.
[A.3] Some channel trace increases in length by more than 1.
[A.4] Two channel traces increase in length at the same step.
[A.5] Some channel trace element takes on more than one value. (A value changes spontaneously between trace-sets on a path).

(The negation of well-formedness condition (1) from Definition [5.3.4] is [A.1], while negating condition (2) results in [A.2] through [A.5].) We must show that in every case, one of *ORDERING* and *PREFIX* is violated. The proof proceeds by induction on the length of $\sigma$.

*Base case*: $|\sigma| = 1$. Since $\sigma$ has only one trace-set, $\sigma$ must be ill-formed due to case [A.1]—all channel traces are not empty in $\sigma_0$. Let $|c| = x$ in $\sigma_0$ for some channel $c$ and some $x \geq 1$. Then $\sigma \models \Box(|c| \geq 0 \Rightarrow |c| \geq x)$. Trivially, $\sigma \models \Box(|c| \geq x \Rightarrow |c| \geq 0)$, so $\sigma \models \Box(|c| \geq 0 \equiv |c| \geq x)$. By *ORDERING* we conclude $\sigma \models \Box(|c| < x \wedge |c| < 0)$. This last assertion is not true, and thus *ORDERING* does not hold for $\sigma$.

*Induction*: $|\sigma| = n+1$, $n \geq 1$. Suppose, as the induction hypothesis, that any $\sigma'$ of length $n$ that is not well-formed and has no repeated trace-sets violates *ORDERING* and/or *PREFIX*. Now consider $\sigma$. If $\langle \sigma_0 .. \sigma_{n-1} \rangle$ is not well-formed, then by the induction hypothesis we are done. So assume that $\langle \sigma_0 .. \sigma_{n-1} \rangle$ is well-formed. Then the ill-formedness of $\sigma$ must occur between trace-sets $\sigma_{n-1}$ and $\sigma_n$ and must be of type [A.2], [A.3], [A.4], or [A.5] above. By cases:

[A.2] (Some channel trace decreases in length.) Let $|c| = x$ in $\sigma_{n-1}$ and $|c| = y$ in $\sigma_n$, for some $c$ and $x > y$. Then $c \sqsubseteq \circ c$ does not hold in $\sigma_n$, $\Box(c \sqsubseteq \circ c)$ is not valid for $\sigma$, and hence *PREFIX* is violated.

[A.3] (Some channel trace increases in length by more than 1.) Suppose $|c| = x$ in $\sigma_{n-1}$ and $|c| = x+y$ in $\sigma_n$, for some $c$, $x$, and $y \geq 2$. Recall that $\langle \sigma_0 .. \sigma_{n-1} \rangle$ is well-formed (by hypothesis), so we know $\langle \sigma_0 .. \sigma_{n-1} \rangle \models \Box(|c| \leq x)$, since $|c| \leq x$ in $\sigma_{n-1}$. Therefore $\sigma \models \Box\ (|c| \geq x+1 \ \Rightarrow\ |c| \geq x+y)$. Now since $\Box(|c| \geq x+y \Rightarrow |c| \geq x+1)$ holds trivially, we obtain $\sigma \models \Box(|c| \geq x+1 \equiv |c| \geq x+y)$. It is not the case, however, that $\sigma \models \Box(|c| < x+1 \wedge |c| < x+y)$. Thus *ORDERING* does not hold.

[A.4] (Two channel traces increase in length at the same step.) Let $|c1| = x$ and $|c2| = y$ in $\sigma_{n-1}$, and let $|c1| = x+1$ and $|c2| = y+1$ in $\sigma_{n-1}$, for some $c1$, $c2$, $x$, and $y$. Since $\langle \sigma_0 .. \sigma_{n-1} \rangle$ is well-formed, $\sigma \models \Box(|c1| \geq x+1 \equiv |c2| \geq x+y)$. Then by *ORDERING* it should be the case that $\sigma \models \Box(|c1| < x+1 \wedge |c2| < x+y)$. This assertion is not valid, so *ORDERING* is violated.

[A.5] (A channel trace element takes on more than one value.) Suppose there is a channel trace element $c_x$ such that $c_x = a$ in $\sigma_{n-1}$, $c_x = b$ in $\sigma_n$, and data items $a$ and $b$ are not identical. Then $c \sqsubseteq \circ c$ does not hold in $\sigma_n$, $\Box(c \sqsubseteq \circ c)$ is not valid for $\sigma$, and *PREFIX* does not hold.

We have shown that if $\sigma$ exhibits one of the five cases above, then $\sigma$ does not satisfy both of *ORDERING* and *PREFIX*. Suppose that in fact $\sigma$ is ill-formed in more than one way. Then consider a condition that involves a single channel—only case (4) involves two channels—and reasoning as above guarantees that one of *ORDERING* and *PREFIX* is still violated.

Thus we have shown that for any $\sigma$ satisfying *ORDERING* and *PREFIX*, *Compress*($\sigma$) is well-formed. Together with the first half of the proof: if $\sigma$ is a sequence of trace-sets then *Compress*($\sigma$) is well-formed if and only if $\sigma \models ORDERING \wedge PREFIX$. ☒

## Acknowledgments

We are grateful to Abha Moitra and Prakash Panangaden for valuable discussions.

### References

[Ap81] K.R. Apt. Ten years of Hoare's logic: a survey — part I. *Trans. on Programming Languages and Systems 3* (October 1981), 431-483.

[AFR80] K.R. Apt, N. Francez, and W.P. de Roever. A proof system for communicating sequential processes. *Trans. on Programming Languages and Systems 2* (July 1980), 359-385.

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

[Br84] S.D. Brookes. A semantics and proof system for communicating processes. *Logics of Programs, Lecture Notes in Computer Science 164*, Springer Verlag, New York, 1984, 68-85.

[CK73] C.C. Chang and H. J. Keisler. *Model Theory*. North-Holland, Amsterdam, 1973.

[CH81] Z.C. Chen and C.A.R. Hoare. Partial correctness of communicating processes and protocols. Technical monograph PRG-20, Programming Research Group, Oxford University Computing Laboratory, May 1981.

[Co78] S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing 7* (February 1978), 70-90.

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

[Ho81] C.A.R. Hoare. A calculus of total correctness for communicating processes. *Science of Computer Programming 1* (October 1981), 49-72.

[Ho85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[Jo85] B. Jonsson. A model and proof system for asynchronous networks. *Proc. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing* (August 1985), 49-58.

[LG81] G.M. Levin and D. Gries. A proof technique for communicating sequential processes. *Acta Informatica 15, 3* (1981), 281-302.

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

[Mi80] R. Milner. A calculus of communicating systems. *Lecture Notes in Computer Science 92*, Springer Verlag, New York, 1980.

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

[Ng85] V. Nguyen. The incompleteness of Misra and Chandy's proof systems. *Information Processing Letters 21* (August 1985), 93-96.

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

[Pr82] V.R. Pratt. On the composition of processes. *Proc. 9th ACM Symp. on Principles of Programming Languages* (January 1982), 213-223.

[S67] J.R. Schoenfield. *Mathematical Logic*. Addison-Wesley, Reading, Mass., 1967.

[ZRE84] J. Zwiers, W.P. de Roever, and P. van Emde Boas. Compositionality and concurrent networks: soundness and completeness of a proofsystem. Report 57, Informatica/Computer Graphics, Faculty of Science, Nijmegen University, The Netherlands, December 1984.