Expressiveness Bounds for Completeness in Trace-Based Network Proof Systems*

Jennifer Widom
Prakash Panangaden

87-867

September 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

*This work was supported in part by the National Science Foundation under grants DCR-8320274 and DCR-8602072.

Expressiveness Bounds for Completeness in Trace-Based Network Proof Systems*

Jennifer Widom
Prakash Panangaden

Computer Science Department
Cornell University
Ithaca, New York 14853

Abstract

Network proof systems based on first-order specifications over channel traces are incomplete unless reasoning over the interleaving of communication events is permitted. Relatively complete trace-based proof systems using temporal logic have been described, but full temporal logic is more powerful than necessary. Using the interleaving approach, we isolate the expressiveness required of a relatively complete trace logic. A hierarchy of temporal logic subsets is then defined; a certain subset is shown to have necessary and sufficient expressive power for relative completeness.

1 Introduction

Proof systems for networks in which the specification of a network can be completely deduced from specifications for its constituent processes are generally *trace-based.* In them, one specifies and reasons about traces (histories) of the values transmitted along the communication channels of the network. Formalisms based on first-order predicates over channel traces are defined in [CH81,Hoa85,MC81,WGS87], but unfortunately these proof systems are incomplete [BA81,Ngu85,WGS87]. Relative completeness can be achieved by permitting reasoning over the interleaving of communication events in addition to the individual channel traces [Bro84,HH83,ZdRvEB85], but an explicit interleaving introduces more information than necessary.

In [NDGO86], a network proof system is described in which specifications are temporal-logic formulas over channel traces. The interleaving of communication events is implicitly present in the semantics of temporal logic, and relative completeness is therefore achieved. In [WGS87], two properties of network computation are described and it is shown that axiomatization of these properties is sufficient for achieving relative completeness in a trace-based proof system. These properties can be expressed in a subset of temporal logic, indicating that the full power of temporal logic is not needed.

*This work was supported in part by the National Science Foundation under grants DCR-8320274 and DCR-8602072.

This paper explores the exact expressiveness required of a logic if it is to be used for a relatively complete trace-based proof system. Although first-order trace logic is not expressive enough, full temporal logic is more powerful than necessary. We introduce a hierarchy of subsets of temporal logic and show that a subset consisting of first-order trace logic 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. Sections 3–5 describe first-order trace logic as a specification language for processes and networks, illustrating the incompleteness of network proof systems built entirely on such logics. In Section 6, we show that, for relative completeness, a logic must be just expressive enough to distinguish those states reachable by legal network computation from those not. By considering the interleaving of communication events, a formula F (say) can be defined that recognizes all and only states reachable by legal computation. This formula, developed in Section 7, exactly characterizes the expressiveness required of a logic if it is to be used for a relatively complete trace-based proof system.

In Section 8, a hierarchy of temporal logic subsets is introduced. A mapping is defined from temporal logic formulas to formulas over event interleavings (as in formula F). In Section 9, this mapping is used to isolate a subset of temporal logic that is necessary and sufficient to express a formula equivalent to F. This subset thus has the exact expressive power required of a relatively complete trace logic. Finally, in Section 10, we draw conclusions 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 with the environment of the network. 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.

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

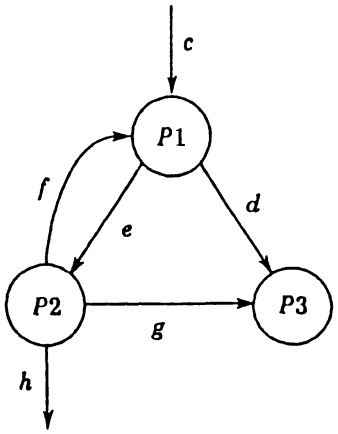

Figure 1: A network of processes

- 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 the parallel execution of the component processes. Fig. 1 illustrates a network of three processes and six communication channels.

3 First-Order Trace Logic

In simple trace-based proof systems, specifications for processes and networks are formulas of *first-order trace logic*—first-order predicates over the sequences of values transmitted on communication channels during execution [CH81,Hoa85,MC81,WGS87]. Such specifications are intended to be satisfied throughout every execution of the process or network they specify.[2] (Thus we are considering only *safety properties* [AS85].)

Let c be a channel. In a formula of first-order trace logic, c denotes a finite sequence, $\langle c.0, c.1, \ldots, c.k \rangle$, indicating the values transmitted along channel c, in order. Sequence $\langle c.0, c.1, \ldots, c.k \rangle$ is called the *channel trace* of c. We use $|c|$ to denote the length of sequence c and $\langle\rangle$ to denote the empty sequence.

[2] 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].

A specification for a process or network N is a first-order formula f over the traces of N's communication channels. We say that f is a *valid* specification for N if, at every point during any computation of N, the traces of the values transmitted on N's channels satisfy f. For example, a specification for a process (or network) that repeatedly reads an integer from incoming channel $c1$ and sends its successor to outgoing channel $c2$ can be formulated as

$$(|c1| - 1 \leq |c2| \leq |c1|) \wedge (\forall i\colon 0 \leq i < |c2|\colon c2.i = c1.i + 1).$$

4 Proof Systems Based on First-Order Trace Logic

A network proof system based on first-order trace logic includes

(a) a system of axioms and inference rules for verifying valid formulas of first-order trace logic, and

(b) a system of axioms and inference rules for verifying formulas of first-order trace logic as valid specifications for given processes and networks.

Since first-order trace logic is a variant of predicate logic, the existence of item (a) is generally assumed.

Some trace-based network proof systems provide facilities for specifying and verifying sequential process programs [CH81,Hoa85,ZdRvEB85] while in others it is assumed that existing logics for sequential programs can be used for this and that process specifications are given [Jon85,MC81,NDGO86]. Without loss of generality, we adopt the latter approach and are simply interested in systems for deducing network specifications from specifications for the network's constituent processes. Typically, such systems are based on

(i) a *network composition rule* for deducing some valid network specification from valid specifications for the network's constituent processes, and

(ii) a *consequence rule* for deducing valid specifications for a network from other valid specifications for that network (since several valid specifications may exist for a given network).

For item (i), the given inference rule usually states that a specification for a network is the logical conjunction of specifications for the network's constituent processes:

Rule 4.1 (Network Composition Rule) If f_1, f_2, ..., f_n are valid specifications for processes P_1, P_2, ... P_n, respectively, then $\bigwedge_i f_i$ is a valid specification for network $N = P_1||P_2||\cdots||P_n$.

Conjoining process specifications using the Network Composition Rule results in "linking" any shared channels in network N because in $\bigwedge_i f_i$ all c's (say) refer to the same channel trace. Formalisms for reasoning over the interleaving of communication events require more complicated network composition rules [Bro84,HH83], although such rules are also built on the conjunction of process specifications.

The inference rule for item (ii) usually relies on theorems of the underlying logic, as follows.

Rule 4.2 (Consequence Rule) If $f1$ is a valid specification for network N and $f1 \Rightarrow f2$ is a theorem of first-order trace logic, then $f2$ is a valid specification for N.

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

5 Completeness and Incompleteness

A network proof system is *complete* if whenever some specification f is valid for network N, then the validity of f can be proven using the axioms and inference rules of the proof system. In a compositional proof system, a network specification is derived from specifications for its component processes (as in Rule 4.1). If the given process specifications are valid, but too weak, then a valid network specification may not be provable. Thus, what we are really interested in is the provability of network specifications when the specifications given for the primitive processes comprising the network are as "strong" as possible [Jon85,NDGO86, WGS87].

Definition 5.1 (Precise Specifications) A specification f is *precise* for a process or network N iff:

1. f is valid for N, and
2. any computation always satisfying f is a possible computation of N.

A precise specification for a process or network characterizes its possible computations. Hence, for completeness, we are interested in the provability of valid specifications for a network given precise specifications for its constituent processes.

Since first-order trace logic includes arithmetic, which itself is incomplete [Sch67], a valid assertion might not be provable in any system. When designing a programming logic, one aims for *relative completeness* (as in [Coo78]): Assuming that any valid statement of first-order trace logic can be proven, is the proof system complete? Proof systems based entirely on first-order trace logic are not relatively complete, as we now illustrate.

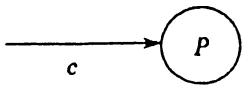

Figure 2: Example network

Consider a simple example network consisting of one process P and one incoming communication channel c (see Fig. 2).[3] Let the data domain for this network be $\{a\}$. Suppose, as an informal description of process P, we know that at all times P has read either no values on channel c or has read two values on c. A valid specification for process P is thus

$$c = \langle\rangle \vee c = \langle a, a\rangle. \tag{1}$$

However, P can reach a state in which $c = \langle a, a\rangle$ only by passing through a state in which $c = \langle a\rangle$, which is prohibited by specification (1).[4] Therefore, a second valid specification is

$$c = \langle\rangle, \tag{2}$$

i.e. P never reads from channel c.

Both specifications are valid and, in fact, precise. Any computation always satisfying (1) or (2) is a computation of P. However, consider an attempt at proving valid specification (2) given precise specification (1). Since there is only a single process, the network composition rule is irrelevant—the only inference we can use is the consequence rule. But

$$(c = \langle\rangle \vee c = \langle a, a\rangle) \Rightarrow (c = \langle\rangle) \tag{3}$$

is not a theorem of first-order trace logic. Hence specification (2) is unprovable, even though it is valid.

Since incompleteness is exhibited here in a single-process network, we know that network composition is not a necessary ingredient of incompleteness. In fact, the following theorem tells us that the network specification deduced from precise process specifications using Rule 4.1 is itself precise [NDGO86,WGS87].

Theorem 5.2 (Preciseness-Preservation) If f_i is a precise specification for process P_i, $1 \leq i \leq n$, then $\bigwedge_i f_i$ is a precise specification for network $N = P_1 \| P_2 \| \cdots \| P_n$.

[3] Other examples—both one- and multi-process—are given in [WGS87] and [Wid87].

[4] We use the term *state* here and subsequently to denote a correspondence between channel trace variables and sequences of transmitted values—a point in network computation.

Proof: See [WGS87] or [Wid87].

Therefore, a trace-based proof system is relatively complete iff every valid specification for a network can be deduced from a precise specification for that network. Our example has shown, however, that if specifications $f1$ and $f2$ are precise and valid, respectively, for a given network, it is not necessarily the case that $f1 \Rightarrow f2$. Thus, Consequence Rule 4.2 is not strong enough to conclude every valid specification. To obtain a relatively complete system, this rule must be strengthened to characterize the exact relationship between precise and valid specifications.[5]

6 Strengthening the Proof System

In the example network of Section 5, disjunct $c = \langle a, a \rangle$ cannot be eliminated from specification $c = \langle\rangle \vee c = \langle a, a \rangle$ using the consequence rule even though a state satisfying $c = \langle a, a \rangle$ is not reachable by any legal network computation always satisfying $c = \langle\rangle \vee c = \langle a, a \rangle$. The $\Rightarrow$ relationship of first-order trace logic does not capture the notion of "states reachable by legal network computation", yet it is exactly this that must be expressed in any relatively complete system.

Suppose $f1$ is a precise specification for a network N. Then, by Definition 5.1, a state is reachable by N iff it is reachable by a computation always satisfying $f1$. Now, by the definition of validity, a specification $f2$ is valid for N iff it is satisfied by every state reachable by N. By transitivity, then, $f2$ is valid iff it is satisfied by every state reachable by a computation always satisfying $f1$. That is:

Observation 6.1 A specification is valid for a network iff it is satisfied by every state reachable by a computation always satisfying a precise specification for that network.

By formalizing this observation, we obtain a relatively complete system. Unfortunately, the observation cannot be expressed in first-order trace logic. (This is rigorously proven in Section 9.) We are interested, then, in the power required of a logic to formally express Observation 6.1.

Let f be a variable representing an arbitrary formula of first-order trace logic. Suppose $C(f)$ is a formula in some logic L such that a state satisfies $C(f)$ iff the state is reachable by a legal network computation always satisfying f. Consider the following modified consequence rule.

[5] Actually, a similar effect can be obtained by strengthening the network composition rule or adding an additional consequence rule for deriving stronger precise specifications, but all such changes are equivalent [Wid87]. Without loss of generality, we only consider modification of the consequence rule.

Rule 6.2 (Modified Consequence Rule) If $f1$ is a valid specification for network N and $C(f1) \Rightarrow f2$ is a theorem of logic L, then $f2$ is a valid specification for N.

Now suppose $f1$ is a precise specification for N. By Observation 6.1 and the definition of $C(f)$, $f2$ is valid for N iff $C(f1) \Rightarrow f2$. Therefore, an inference rule based on Rule 6.2 yields a proof system that is complete relative to logic L. Our goal is thus to isolate the power required of a logic to express formula $C(f)$—a formula satisfied by all and only those states reachable by a computation always satisfying f.

7 Extending First-Order Trace Logic

Under the assumptions of Section 2, a legal network computation is modeled by a finite or infinite sequence of states satisfying

1. all channel traces in the initial state are empty, and
2. each subsequent state extends at most one trace of the preceding state by at most one element.[6]

Let the communication channels of all networks under consideration be $c_1, c_2, \ldots, c_k$, and let the domain of transmittable values be a set V. A state can be represented by a tuple $\bar{t} = [t_1, \ldots, t_k]$ of channel traces, where t_j is the trace of c_j, $1 \leq j \leq k$. We extend first-order trace logic to allow reasoning over *state-sequences*—arbitrarily long sequences $\langle \overline{t^0}, \overline{t^1}, \ldots, \overline{t^n} \rangle$ of such tuples. State-sequences model network computation and permit reasoning over the interleaving of communication events.

Formula $C(f)$ is expressible in the extended logic. Let

- $f[\overline{t^i}/\bar{c}]$ denote formula f with channel trace variable c_j replaced by trace t_j^i, $1 \leq j \leq k$;[7]
- $\overline{t^i}[t_j^i \cdot \langle v \rangle / t_j^i]$ denote tuple $\overline{t^i}$ with trace t_j^i extended by value $v \in V$;

We then define $C(f)$ as follows.

---

[6] Strictly speaking, one needs second-order logic to limit computations to such sequences. Our results are relative to the assumption that we can restrict sequences in this way.

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

Definition 7.1 (Formula C(f) in Extended Trace Logic)

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

The free variables of $C_{ETL}(f)$ are channel trace variables $c_1, \ldots, c_k$. As illustrated by the annotation, $C_{ETL}(f)$ is satisfied by exactly those states reachable by a legal network computation always satisfying f. Therefore, Definition 7.1 allows us to formalize Consequence Rule 6.2 and attain relative completeness.

It would be unwise, however, to use $C_{ETL}(f)$ explicitly in a proof system. Verifying $C_{ETL}(f1) \Rightarrow f2$ for application of the consequence rule could require construction of arbitrarily long sequences drawn from a large pool of possibilities—i.e. the enumeration of all possible computations.

8 Using Temporal Logic

Linear-Time Temporal Logic (TL) is a formalism for reasoning over an implicit sequence of states [MP82,RU71]. Therefore, for expressing and reasoning with formulas such as $C(f)$, TL is more appropriate than the extended first-order trace logic of Section 7. However, the full power of temporal logic is more than is needed to express $C(f)$. We want to isolate a subset of TL that is necessary and sufficient to express $C(f)$ and is thus appropriate for use as the basis of a relatively complete trace-based proof system.

We begin with a version of TL consisting of first-order trace logic and three (standard) temporal operators:

1. The *Always* operator, $\Box$. Informally, $\Box f$ is valid iff TL formula f is valid at the current point in time and at every point in the future.

2. The *Next* operator, $\circ$. Informally, $\circ f$ is valid iff TL formula f is valid at the next point in time.

3. The *Until* operator, $\mathcal{U}$. Informally, $f1\ \mathcal{U}\ f2$ is valid for TL formulas $f1$ and $f2$ iff $f2$ is valid either at the current point in time or at some point in the future, and $f1$ is valid at all points from the current point to the point at which $f2$ becomes valid.[8]

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

TL formulas can be defined over network computation by considering a sequence of states as a description of successive points in time. The temporal operators are then formalized with respect to state-sequences (in the obvious way) according to their informal descriptions above [MP81,MP82].

When we assert that a (first-order) specification f is valid for some network N, by definition we are asserting that $\Box f$ holds for every computation of N. The *Always* operator used in this context—over first-order formulas only—is a restricted version of $\Box$ as defined above. Permitting nested temporal operators yields significantly more expressive power than restricting temporal operators to operate over non-temporal formulas.[9] Since we are interested in finding a TL subset to express a formula $C(f)$ in which all f's under consideration are non-temporal, we consider an additional set of temporal operators that operate over first-order formulas only:

1. the *Restricted Always* operator, $\bar{\Box}$
2. the *Restricted Next* operator, $\bar{\circ}$
3. the *Restricted Until* operator, $\bar{\mathcal{U}}$

These operators are strictly weaker than their fully temporal counterparts—which we refer to as *Unrestricted* operators—and thus enrich the subset structure of TL.

A subset of TL is constructed by choosing any subset of the six temporal operators; for example, trace logic with $\Box$ and $\bar{\circ}$ is a (strict) subset of TL. We want to isolate the TL subset that is both necessary and sufficient to express formula $C(f)$. If we define a mapping from TL to the extended trace logic (ETL) of Section 7, then we can establish expressiveness bounds by proving that a certain set of temporal operators is required in any TL formula that is equivalent (through the mapping) to Definition 7.1 of $C_{ETL}(f)$.

---

[8] Some definitions of TL instead use a *Weak Until* operator, in which $f2$ need not ever become valid as long as $f1$ is always valid. In the context of TL, the two versions of $\mathcal{U}$ are expressively equivalent [Wol81].

[9] For example, consider TL formula $\Box(f1 \Rightarrow (\Box f2 \vee \circ f3))$, which asserts that whenever $f1$ is valid, either $f2$ is valid thereafter or $f3$ is valid at the next point in time. This property cannot be expressed using temporal operators only over first-order formulas.

Although the free variables of a TL formula may be identical to the free variables of an ETL formula—some set of channel trace variables—TL formulas are interpreted over a sequence of states while ETL formulas are interpreted over a single state. Note, however, that the ETL formula under consideration, $C_{ETL}(f)$, is based on quantification over a variable representing a sequence of states. We redefine $C_{ETL}(f)$ so that its state-sequence is represented by a free variable—this requires an adjustment to the modified consequence rule—and we define a mapping $\mathcal{M}$ from arbitrary TL formulas to ETL formulas with a free state-sequence variable. (As seen below, $\mathcal{M}$ is straightforward and derived directly from the definitions of the temporal operators.) This mapping can then be used to establish equivalence between TL formulas and the modified version of $C_{ETL}(f)$.

Let σ represent an infinite sequence of tuples.[10] Define $C_\sigma(f)$ as follows.

Definition 8.1 (Modified C(f) in Extended Trace Logic)

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

There is a clear correspondence between $C_\sigma(f)$ and Definition 7.1 of $C_{ETL}(f)$. It is easily verifiable [Wid87] that for any specifications $f1$ and $f2$,

$$C_{ETL}(f1) \Rightarrow f2 \ \text{ iff } \ C_\sigma(f1) \Rightarrow (\forall i{:}\ i \geq 0{:}\ f2[\sigma.i/\bar{c}\,]).$$

By the definition of $C_\sigma(f)$, we see that any TL subset that can express $C_\sigma(f)$ can also express $(\forall i{:}\ i \geq 0{:}\ f[\sigma.i/\bar{c}\,])$. Therefore, we can revise the modified consequence rule (Rule 6.2) to use the equivalent implication $C_\sigma(f1) \Rightarrow (\forall i{:}\ i \geq 0{:}\ f2[\sigma.i/\bar{c}\,])$, adopting Definition 8.1 of $C_\sigma(f)$ as our measure of required expressiveness.

Next, we must define the mapping $\mathcal{M}$ from TL formulas to ETL formulas. $\mathcal{M}$ is defined such that if f_{TL} is a formula of temporal logic, then $\mathcal{M}[\![f_{TL}]\!]$ is an ETL formula containing a single free state-sequence variable σ. The mapping is *semantics-preserving*, in that a

[10] We switch from finite to infinite sequences at this point to eliminate tedious details in proofs. In our context, any finite state-sequence can be converted to an equivalent infinite state-sequence by indefinitely repeating the final state [MP81,Wid87].

state-sequence σ' satisfies formula f_{TL} iff substitution $[\sigma'/\sigma]$ satisfies formula $\mathcal{M}[\![f_{TL}]\!]$.[11] Using such a mapping, TL formula f_{TL} is said to be equivalent to ETL formula f_{ETL} iff $\mathcal{M}[\![f_{TL}]\!] \Leftrightarrow f_{ETL}$.

We define $\mathcal{M}$ inductively on the structure of TL formulas [MP82]; the mapping parallels the usual semantics of temporal logic [MP82,RU71]. Note that a mapping $\mathcal{T}$ from TL terms to ETL terms is also needed.

- $\mathcal{M}[\![p(t_1, \ldots, t_n)]\!] = p(\mathcal{T}[\![t_1]\!], \ldots, \mathcal{T}[\![t_n]\!])$, p a predicate, $t_1, \ldots, t_n$ TL terms
- $\mathcal{M}[\![f1 \vee f2]\!] = \mathcal{M}[\![f1]\!] \vee \mathcal{M}[\![f2]\!]$, $f1$ and $f2$ TL formulas
- $\mathcal{M}[\![\neg f]\!] = \neg\mathcal{M}[\![f]\!]$, f a TL formula
- $\mathcal{M}[\![(\exists \boldsymbol{x}:: f)]\!] = (\exists k:: \mathcal{M}[\![f[k/\boldsymbol{x}]]\!])$, f a TL formula, k a constant in V
- $\mathcal{M}[\![\Box f]\!] = \mathcal{M}[\![\bar{\Box} f]\!] = (\forall i: i \geq 0: \mathcal{M}[\![f]\!][\sigma[i..]/\sigma])$, f a TL formula
- $\mathcal{M}[\![\circ f]\!] = \mathcal{M}[\![\bar{\circ} f]\!] = \mathcal{M}[\![f]\!][\sigma[1..]/\sigma]$, f a TL formula
- $\mathcal{M}[\![f1\ \mathcal{U}\ f2]\!] = \mathcal{M}[\![f1\ \bar{\mathcal{U}}\ f2]\!] =$
  $(\exists i: i \geq 0: (\mathcal{M}[\![f2]\!][\sigma[i..]/\sigma]) \wedge (\forall j: 0 \leq j < i: \mathcal{M}[\![f1]\!][\sigma[j..]/\sigma]))$,
  $f1$ and $f2$ TL formulas
- $\mathcal{T}[\![k]\!] = k$, k a constant in V
- $\mathcal{T}[\![c_i]\!] = (\sigma.0)_i$, c_i a variable in $\{c_1, \ldots, c_k\}$
- $\mathcal{T}[\![fn(t_1, \ldots, t_n)]\!] = fn(\mathcal{T}[\![t_1]\!], \ldots, \mathcal{T}[\![t_n]\!])$, fn a function, $t_1, \ldots, t_n$ TL terms

Theorem 8.2 (Correctness of the Mapping) For any TL formula f_{TL} and state-sequence σ', σ' satisfies f_{TL} iff $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$.

Proof: See [Wid87].

9 Expressiveness Bounds for Relative Completeness

We want to isolate the subset of temporal logic needed to express a formula equivalent to Definition 8.1 of $C_\sigma(f)$. That is, we want to determine which combination of the six temporal operators defined in Section 8 is necessary and sufficient in any formula $C_{TL}(f)$ such that $\mathcal{M}[\![C_{TL}(f)]\!] \Leftrightarrow C_\sigma(f)$.

[11] 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].

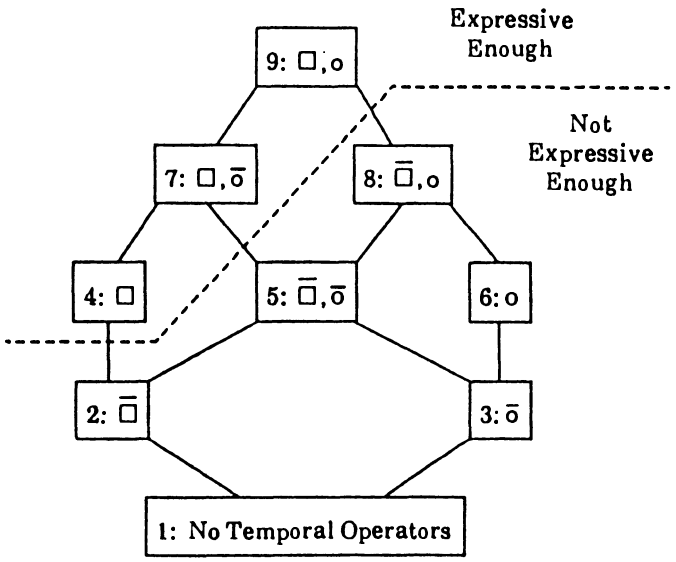

Figure 3: Temporal logic subsets and expressiveness bounds

By the definition of mapping $\mathcal{M}$ on formulas of the form $f1\ \mathcal{U}\ f2$ (Section 8) and the fact that $C_\sigma(f)$ contains no eventuality components of the form $(\exists i\colon i \geq 0\colon fn(\sigma.i))$, we see that operator $\mathcal{U}$ is not needed in any TL formula $C_{TL}(f)$ such that $\mathcal{M}[\![C_{TL}(f)]\!] \Leftrightarrow C_\sigma(f)$. Similarly, there is no need to consider operator $\bar{\mathcal{U}}$. Therefore, the TL subsets of interest correspond to the subsets of $\{\Box, \mathrm{o}, \bar{\Box}, \bar{\mathrm{o}}\}$. The partial ordering of these subsets is illustrated in Figure 3.

For each non-empty subset S in Figure 3, let TL_S denote first-order trace logic extended to include the operators in S. For example, subset 4 of Figure 3 is denoted by $\mathrm{TL}_\Box$ and subset 8 by $\mathrm{TL}_{\bar{\Box}\mathrm{o}}$ We prove that $\mathrm{TL}_\Box$ has the necessary and sufficient expressive power to encode $C_\sigma(f)$. We show that $\mathrm{TL}_\Box$ is sufficient by using trace logic with *Unrestricted Always* operators (only) to write a formula equivalent to $C_\sigma(f)$. We then prove that the subset is an absolute lower bound: each subset lower than or incomparable to subset 4 in the hierarchy of Figure 3 is not expressive enough to encode $C_\sigma(f)$. This is proven by showing that no formula equivalent to $C_\sigma(f)$ can be expressed in $\mathrm{TL}_{\bar{\Box}\mathrm{o}}$, subset 8. Consequently, all subsets except 4, 7 and 9 are insufficient. The resulting division of the subset hierarchy is shown in Figure 3.

9.1 Sufficiency

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

Definition 9.1 (C(f) in $\mathrm{TL}_{\Box}$)

$$
\begin{aligned}
&C_{\Box}(f) \equiv \\
&\quad \Box f \wedge \\
&\quad (\forall m, n, x, y\colon 1 \leq m \leq k, 1 \leq n \leq k, x \geq 1, y \geq 0, m \neq n \vee x \neq y\colon \\
&\qquad (\Box(|c_m| \geq x \Leftrightarrow |c_n| \geq y)) \Rightarrow \Box(|c_m| < x \wedge |c_n| < y)) \wedge \\
&\quad (\forall m, x, v\colon 1 \leq m \leq k, x \geq 0, v \in V\colon \\
&\qquad \Box((c_m.x = v) \Rightarrow \Box(c_m.x = v)))
\end{aligned}
$$

Informally, the first conjunct of $C_{\Box}(f)$ restricts state-sequences satisfying $C_{\Box}(f)$ to always satisfy f. The second and third conjuncts are derived from the temporal ordering and prefix axioms of [WGS87], restricting state-sequences satisfying $C_{\Box}(f)$ to represent legal network computations. Formally, then:

Theorem 9.2 $\mathcal{M}[\![C_{\Box}(f)]\!] \Leftrightarrow C_{\sigma}(f)$.

Proof: See appendix.

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

9.2 Necessity

We now show that, with respect to our hierarchy of TL subsets, $\mathrm{TL}_{\Box}$ is necessary to encode $C_{\sigma}(f)$: any TL subset weaker than or incomparable to $\mathrm{TL}_{\Box}$ cannot be used to express a formula equivalent to $C_{\sigma}(f)$. By the hierarchy of Figure 3, proving the necessity of $\mathrm{TL}_{\Box}$ only requires proving that no formula equivalent to $C_{\sigma}(f)$ can be expressed in $\mathrm{TL}_{\overline{\Box}\circ}$.

First, we will prove a key lemma: there is no formula f_{TL} in $\mathrm{TL}_{\overline{\Box}\circ}$ such that $\mathcal{M}[\![f_{TL}]\!]$ and $C_{\sigma}(\mathit{true})$ are satisfied by the same set of substitutions for free variable σ. The final result—that there is no formula $C_{\overline{\Box}\circ}(f)$ in $\mathrm{TL}_{\overline{\Box}\circ}$ such that $\mathcal{M}[\![C_{\overline{\Box}\circ}(f)]\!] \Leftrightarrow C_{\sigma}(f)$—then follows directly. Let C_{σ} be an abbreviation for $C_{\sigma}(\mathit{true})$. (Recall that *true* is a legal specification of first-order trace logic.) To prove that there is no formula f_{TL} in $\mathrm{TL}_{\overline{\Box}\circ}$ such that $\mathcal{M}[\![f_{TL}]\!]$ and C_{σ} are satisfied by the same set of substitutions, we show, for every potential f_{TL}, that there is some state-sequence σ' such that either $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ but not $C_{\sigma}[\sigma'/\sigma]$, or $C_{\sigma}[\sigma'/\sigma]$ but not $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$.

Informally, the argument proceeds as follows. For every f_{TL} in $\mathrm{TL}_{\overline{\Box}\circ}$ there is some $n \geq 0$ (n is the nesting depth of *Next* operators in f_{TL}) such that $\mathcal{M}[\![f_{TL}]\!]$ can refer to states of σ

beyond $\sigma.n$ only by universal quantification (resulting from *Restricted Always* operators in f_{TL}). If no state-sequences σ' satisfy $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$, or the only satisfying sequences have all repeated states beyond $\sigma'.n$, then it is straightforward to construct a sequence σ'' such that $C_\sigma[\sigma''/\sigma]$ but not $\mathcal{M}[\![f_{TL}]\!][\sigma''/\sigma]$. Otherwise, we construct a sequence σ'' that represents an illegal computation due to an irregularity beyond state $\sigma''.n$ (e.g. the length of a trace decreases from one state to the next). $C_\sigma[\sigma''/\sigma]$ does not hold; however, $\mathcal{M}[\![f_{TL}]\!][\sigma''/\sigma]$ does hold, as long as σ'' is constructed by rearranging states from a sequence σ' known to satisfy $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$.

Lemma 9.3 For any formula f_{TL} in $\mathrm{TL}_{\overline{\Box}_o}$, there exists a state-sequence σ' such that either

$\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ and not $C_\sigma[\sigma'/\sigma]$, or

$C_\sigma[\sigma'/\sigma]$ and not $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$.

Proof: See appendix.

Theorem 9.4 There is no formula $C_{\overline{\Box}_o}(f)$ in $\mathrm{TL}_{\overline{\Box}_o}$ such that $\mathcal{M}[\![C_{\overline{\Box}_o}(f)]\!] \Leftrightarrow C_\sigma(f)$.

Proof: Consider an arbitrary formula $C_{\overline{\Box}_o}(f)$ in $\mathrm{TL}_{\overline{\Box}_o}$. $\mathcal{M}[\![C_{\overline{\Box}_o}(f)]\!] \Leftrightarrow C_\sigma(f)$ iff for all trace formulas f and state-sequences σ': $\mathcal{M}[\![C_{\overline{\Box}_o}(f)]\!][\sigma'/\sigma]$ iff $C_\sigma(f)[\sigma'/\sigma]$. Consider the case in which $f \equiv \mathit{true}$. By Lemma 9.3, there exists a state-sequence σ' such that either $\mathcal{M}[\![C_{\overline{\Box}_o}(\mathit{true})]\!][\sigma'/\sigma]$ and not $C_\sigma(\mathit{true})[\sigma'/\sigma]$, or $C_\sigma(\mathit{true})[\sigma'/\sigma]$ and not $\mathcal{M}[\![C_{\overline{\Box}_o}(\mathit{true})]\!][\sigma'/\sigma]$. Hence $\mathcal{M}[\![C_{\overline{\Box}_o}(\mathit{true})]\!] \not\Leftrightarrow C_\sigma(\mathit{true})$ and consequently $\mathcal{M}[\![C_{\overline{\Box}_o}(f)]\!] \not\Leftrightarrow C_\sigma(f)$. QED

10 Conclusions

We have proven that a formula equivalent to Definition 8.1 of $C_\sigma(f)$ cannot be written in $\mathrm{TL}_{\overline{\Box}_o}$. Since we have also shown that a formula equivalent to $C_\sigma(f)$ must be expressible in any relatively complete trace-based proof system (Sections 6–8), we conclude that $\mathrm{TL}_{\overline{\Box}_o}$, as well as any weaker logic, is not expressive enough for relative completeness. In Section 9.1, a formula equivalent to $C_\sigma(f)$ is defined using $\mathrm{TL}_\Box$. Therefore, with respect to the hierarchy of TL subsets pictured in Figure 3, first-order trace logic with an *Unrestricted Always* operator is both an upper and a lower bound on the expressive power required for relative completeness in a trace-based network proof system.

The result can be strengthened slightly by refining the TL subset hierarchy. Rather than distinguishing only between *Restricted* and *Unrestricted* temporal operators—operators that may be nested zero and 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 Section 9, it is easy to show that, with respect to this refined subset hierarchy, $\mathrm{TL}_{\Box_1}$ is necessary and sufficient to express a formula equivalent to $C_\sigma(f)$ [Wid87].

In [WGS87], two properties of network computation are axiomatized and it is shown that incorporating these axioms into a proof system is sufficient for achieving relative completeness. It is not surprising, then, that we were able to derive Definition 9.1 of $C_\Box(f)$ from the axioms in [WGS87]. The axioms are given in [WGS87] using temporal operators $\Box$ and $\circ$, however. In this work, we prove that although the use of *Next* operators can be eliminated, the use of nested *Always* operators cannot.

Appendix

Theorem 9.2 $\mathcal{M}[\![C_\Box(f)]\!] \Leftrightarrow C_\sigma(f)$.

Proof: Proving $\mathcal{M}[\![C_\Box(f)]\!] \Leftrightarrow C_\sigma(f)$ requires showing that for all trace formulas f and state-sequences σ', $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ iff $C_\sigma(f)[\sigma'/\sigma]$.[12] First, applying the definition of $\mathcal{M}$, we obtain:

$$
\begin{aligned}
&\mathcal{M}[\![C_\Box(f)]\!] \equiv \\
&\quad (\forall i{:}\ i \geq 0{:}\ f[\sigma.i/\bar{c}])^{13} \wedge \\
&\quad (\forall m, n, x, y{:}\ 1 \leq m \leq k, 1 \leq n \leq k, x \geq 1, y \geq 0, m \neq n \vee x \neq y{:} \\
&\qquad (\forall i{:}\ i \geq 0{:}\ |(\sigma.i)_m| \geq x \Leftrightarrow |(\sigma.i)_n| \geq y) \Rightarrow \\
&\qquad\quad (\forall i{:}\ i \geq 0{:}\ |(\sigma.i)_m| < x \wedge |(\sigma.i)_n| < y)) \wedge \\
&\quad (\forall m, x, v{:}\ 1 \leq m \leq k, x \geq 0, v \in V{:} \\
&\qquad (\forall i{:}\ i \geq 0{:}\ ((\sigma.i)_m.x = v) \Rightarrow (\forall j{:}\ j \geq 0{:}\ (\sigma.(i{+}j))_m.x = v))).
\end{aligned}
$$

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

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

We prove the equivalent contrapositive: the falsity of $C_\sigma(f)[\sigma'/\sigma]$ implies the falsity of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$. Suppose $C_\sigma(f)[\sigma'/\sigma]$ is false, and consider the three conjuncts of $C_\sigma(f)$. The second conjunct of $C_\sigma(f)$ is identical to the first conjunct of $\mathcal{M}[\![C_\Box(f)]\!]$. Therefore, if the second conjunct of $C_\sigma(f)[\sigma'/\sigma]$ is false then $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is also false. Suppose the third conjunct of $C_\sigma(f)[\sigma'/\sigma]$ is false. This can happen in four ways:

[12] This proof parallels the proof of Lemma 6.3.1 in [WGS87].

[13] Note that for any non-temporal formula f, $\mathcal{M}[\![f]\!] = f[(\sigma.0)_1, \ldots, (\sigma.0)_k/c_1, \ldots, c_k] = f[\sigma.0/\bar{c}]$.

1. There is some $j \geq 0$ and m, $1 \leq m \leq k$, such that $|(\sigma'.j)_m| > |(\sigma'.(j+1))_m|$. In this case, the third conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is false.

2. There is some $j \geq 0$ and m, $1 \leq m \leq k$, such that $|(\sigma'.j)_m| = x$, $|(\sigma'.(j+1))_m| = x+y$, and $y > 1$. If σ' does not satisfy case 1 as well, then $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| \geq x+1 \Leftrightarrow |(\sigma'.i)_m| \geq x+y)$. But it is not the case that $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| < x+1 \wedge |(\sigma'.i)_m| < x+y)$, so the second conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is false.

3. There is some $j \geq 0$, m, $1 \leq m \leq k$, and $n \neq m$, $1 \leq n \leq k$, such that $|(\sigma'.j)_m| < |(\sigma'.(j+1))_m|$ and $|(\sigma'.j)_n| < |(\sigma'.(j+1))_n|$. Let $x = |(\sigma'.j)_m|$ and $y = |(\sigma'.j)_n|$. If σ' does not satisfy case 1 as well, then $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| \geq x+1 \Leftrightarrow |(\sigma'.i)_n| \geq y+1)$. However $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| < x+1 \wedge |(\sigma'.i)_n| < y+1)$ does not hold. Hence the second conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is false.

4. There is some $j \geq 0$, $x \geq 0$ and m, $1 \leq m \leq k$, such that $(\sigma'.j)_n.x = v$, $(\sigma'.(j+1))_n.x = w$, and $v \neq w$. In this case, the third conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is false.

These are the only possibilities, so if the third conjunct of $C_\sigma(f)[\sigma'/\sigma]$ is false, then $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is also false. Finally, suppose the first conjunct of $C_\sigma(f)[\sigma'/\sigma]$ is false, i.e. $\sigma'.0 \neq [\langle\rangle, \ldots, \langle\rangle]$. If σ' does not satisfy case 1 above as well, then there must be some $x \geq 1$ and m, $1 \leq m \leq k$, such that $|(\sigma'.i)_m| \geq x$ for all $i \geq 0$. Therefore $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| \geq x \Leftrightarrow |(\sigma'.i)_m| \geq 0)$. But it is not the case that $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| < x \wedge |(\sigma'.i)_m| < 0)$. Hence the second conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is false. We have considered all three conjuncts of $C_\sigma(f)$, so we conclude that $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ implies $C_\sigma(f)[\sigma'/\sigma]$.

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

We must show that each of the three conjuncts of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ follows from $C_\sigma(f)[\sigma'/\sigma]$. The first conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is identical to the second conjunct of $C_\sigma(f)[\sigma'/\sigma]$, and therefore follows directly. The third conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ is a straightforward consequence of the third conjunct of $C_\sigma(f)[\sigma'/\sigma]$. Now consider the second conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$. Suppose that this conjunct is false—that there is some m, n, x, and y such that $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| \geq x \Leftrightarrow |(\sigma'.i)_n| \geq y)$ but not $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| < x \wedge |(\sigma'.i)_n| < y)$. Consider three cases:

1. $y = 0$. Then $(\forall i\colon i \geq 0\colon |(\sigma'.i)_m| \geq x)$ and consequently $\sigma'.0 \neq [\langle\rangle, \ldots, \langle\rangle]$.

2. $y > 0$ and $m = n$. Then there must be some $j \geq 0$ such that $|(\sigma'.(j+1))_m| - |(\sigma'.j)_m| \geq 2$.

3. $y > 0$ and $m \neq n$. Then there must be some $j \geq 0$ such that $|(\sigma'.j)_m| < |(\sigma'.(j+1))_m|$ and $|(\sigma'.j)_n| < |(\sigma'.(j+1))_n|$.

In each case, at least one conjunct of $C_\sigma(f)[\sigma'/\sigma]$ is false. Hence the second conjunct of $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ must hold, and $\mathcal{M}[\![C_\Box(f)]\!][\sigma'/\sigma]$ follows from $C_\sigma(f)[\sigma'/\sigma]$. QED

Lemma 9.3 For any formula f_{TL} in $\mathrm{TL}_{\overline{\Box}\circ}$, there exists a state-sequence σ' such that either

$\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ and not $C_\sigma[\sigma'/\sigma]$, or

$C_\sigma[\sigma'/\sigma]$ and not $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$.

Proof: To prove this, we will need to consider the nesting depth of *Next* operators in formulas of $\mathrm{TL}_{\overline{\Box}\circ}$. For any formula f_{TL} in $\mathrm{TL}_{\overline{\Box}\circ}$, let $\circ$-*nesting*(f_{TL}) denote the maximum nesting of *Next* operators in f_{TL}. (For example, $\circ$-*nesting*$(\circ(f1 \vee \circ f2) = 2$.) Since every formula f_{TL} has only a finite number of $\circ$'s, $\circ$-*nesting*(f_{TL}) is a well-defined nonnegative integer.

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

Case 1. There is no σ' such that $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$:

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

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

$\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ is false, since there is no σ' such that $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$, but, by Definition 8.1 of $C_\sigma(f)$, $C_\sigma[\sigma'/\sigma]$ is true.

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

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

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

for an arbitrary $v \in V$. Then $\mathcal{M}[\![f_{TL}]\!][\sigma''/\sigma]$ is false, since $\sigma''.(n+1) \neq \sigma''.n$. But, by $C_\sigma[\sigma'/\sigma]$ and the definition of $C_\sigma(f)$, $C_\sigma[\sigma''/\sigma]$ is true.

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

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

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

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

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

Note that $\sigma''.(n+1) = \sigma'.m$ and $\sigma''.(n+2) = \sigma'.n$. Let x be the index of the last element in $(\sigma''.(n+1))_j$. (We know $(\sigma''.(n+1))_j$ is non-empty, since $\sigma''.(n+1) = \sigma'.m = \sigma'.n[(\sigma'.n)_j \cdot \langle v \rangle / (\sigma'.n)_j]$.) $C_\sigma[\sigma''/\sigma]$ is false because its third conjunct is contradicted (recall Definition 8.1): $(\sigma''.(n+1))_j.x = (\sigma'.m)_j.x = v$, but $(\sigma''.(n+2))_j.x = (\sigma'.n)_j.x$ is undefined. Therefore, by showing $\mathcal{M}[\![f_{TL}]\!][\sigma''/\sigma]$, the proof is complete.

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

Base Case:

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

Induction:

- $f_{TL} = f1 \vee f2$: By $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ and the definition of $\mathcal{M}$, $\mathcal{M}[\![f1]\!][\sigma'/\sigma]$ or $\mathcal{M}[\![f2]\!][\sigma'/\sigma]$. If $\mathcal{M}[\![f1]\!][\sigma'/\sigma]$, then by the induction hypothesis, $\mathcal{M}[\![f1]\!][\sigma''/\sigma]$. If $\mathcal{M}[\![f2]\!][\sigma'/\sigma]$, then by the induction hypothesis, $\mathcal{M}[\![f2]\!][\sigma''/\sigma]$. Therefore $\mathcal{M}[\![f1]\!][\sigma''/\sigma]$ or $\mathcal{M}[\![f2]\!][\sigma''/\sigma]$ and consequently $\mathcal{M}[\![f1 \vee f2]\!][\sigma''/\sigma]$.

- $f_{TL} = \neg f$: By $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ and the definition of $\mathcal{M}$, $\mathcal{M}[\![f]\!][\sigma'/\sigma]$ is false. Then by the induction hypothesis, $\mathcal{M}[\![f]\!][\sigma''/\sigma]$ is false. Therefore $\mathcal{M}[\![\neg f]\!][\sigma''/\sigma]$.

- $f_{TL} = (\exists x:: f)$: By $\mathcal{M}[\![f_{TL}]\!][\sigma'/\sigma]$ and the definition of $\mathcal{M}$, there exists a k such that $\mathcal{M}[\![f[k/x]]\!][\sigma'/\sigma]$. Then by the induction hypothesis, there exists a k such that $\mathcal{M}[\![f[k/x]]\!][\sigma''/\sigma]$. Therefore $\mathcal{M}[\![(\exists x:: f)]\!][\sigma''/\sigma]$.

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

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

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

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

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

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

Acknowledgements

We are grateful to David Gries, Abha Moitra, and Fred Schneider, for valuable discussions and useful comments.

References

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[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.

[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.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic. Library of Exact Philosophy*, Springer-Verlag, Vienna, 1971.

[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.