Complete, Trace-based, Network Proof Systems: An Advisor's Perspective

David Gries[0]

August 1987

TR 87-865

1. Introduction

Consider a network of processes that communicate solely by synchronous message passing, for example using CSP primitives '!' and '?' [2] (our results hold for asynchronous message passing as well, but for explanatory purposes we restrict attention to synchronous message passing.) We are interested in trace-based, compositional proof systems for such networks, with theorems of the form

(0) $\quad N$ **sat** R,

where N is a network of processes and R is a first-order predicate whose free variables are the names of the channels of the network. Within R, the name of a channel denotes a trace, i.e. the sequence of values that have been delivered along the channel. Sentence (0) is valid iff at all times during any execution of network N predicate R holds.

We consider logics with sentences of form (0). Primitive processes are specified by axioms of form (0), so that we can concentrate our attention on the rules for composing networks and inferring their properties from the properties of their components. Using $N_0 \,||\, \ldots \,||\, N_{n-1}$ to denote the network constructed by combining the N_i into a single network, we can write a typical composition rule as

[0] Work supported by the NSF under grant DCR-8320274.

$$(1) \qquad \frac{N_0 \textbf{ sat } R_0, \ldots, N_{n-1} \textbf{ sat } R_{n-1}}{N_0 \,||\, \ldots \,||\, N_{n-1} \textbf{ sat } \forall(i :: R_i)}$$

We assume that at most two different N_i use the same channel name. If two processes use the same channel name, then one uses it only as an input channel and the other only as an output channel, and in the constructed network these two channels are linked.

In addition, there is the consequence rule

$$(2) \qquad \frac{N \textbf{ sat } Q,\; Q \Rightarrow R}{N \textbf{ sat } R}$$

and a rule to rename a channel:

$$(3) \qquad \frac{N \textbf{ sat } R}{N^n_m \textbf{ sat } R^n_m} \quad \text{(for } m \text{ a fresh channel name)}$$

There are several variations on this theme —for example, one can have inference rules for hiding channels. However, the above description is at the heart of most trace-based proof systems for networks.

Most proof systems based on such axioms have turned out to be relatively incomplete (e.g. the systems in [5] and [3]), and in a not inconsequential manner: several obviously true sentences about simple networks are not provable using these logics. Further, it has been suggested by several authors that this relative incompleteness is due to composition rule (1) (e.g. [0], [4]).

2

Hehner and Hoare [1] were able to achieve relative completeness by allowing reasoning over the interleaving of events on different traces, but using fairly intricate proof rules. Nguyen [6] moved into temporal logic to get implicit reasoning about interleavings and thereby achieved relative completeness. His system requires full temporal logic, even to prove the simple safety properties expressed by (0).

In her Ph.D. thesis [7], Widom determines the cause of the incompleteness in trace-based proof systems, develops simple machinery to overcome it, and then determines just how expressive a proof system must be to remain relatively complete, i.e. determines the minimal amount of extra logic over the predicate calculus needed to achieve relative completeness. The results are somewhat surprising at first, but then obvious.

The purpose of this note is to describe for myself (and others that may want to read it) the essence of Widom's results as simply and clearly as possible, leaving proofs and such to other papers. For this, we rely on the reader's knowledge of the area and good will in understanding terms that have their usual meanings.

2. Incompleteness and its solution

Our notion of relative completeness deals with the notions of *valid* and *precise* specifications, so let us explain these terms. A *computation* is the sequence of states (each giving the traces of the channel variables) assumed by a network during a single execution of it. A computation satisfies the following:

(a) In the initial state all channels are empty.

(b) A channel is changed only by appending a single value to its trace.

(c) At most one event occurs at a time: in any two successive states the trace of at most one channel is changed.

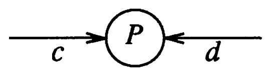
A specification N **sat** R is *valid* iff first-order predicate R holds in all states of all computations of network N. A valid specification is *precise* iff any computation all of whose states satisfy R is actually a computation of N.

Thus, a precise specification N **sat** R exactly characterizes N: all states of all computations of N satisfy R, and each computation that satisfies the specification can indeed be produced by some execution of N.

We consider a proof system to be relatively complete iff any valid specification for a network can be proved in the system (which includes as axioms precise specifications for the primitive processes).

It turns out that composition rule (1) preserves precision: if the specifications N_i **sat** R_i are precise, then so is $N_0 \mid\mid \dots \mid\mid N_{n-1}$ **sat** $\forall(i :: R_i)$. Therefore, the composition rule is not the basis of incompleteness, as previously thought. We now turn to examples that illustrate the cause of incompleteness.

Consider the network



with the informal description: P reads at most one value from channel c, P reads at most one value from channel d, P reads from c first, and P reads from d first.

The following precise specification is given for P:

(4) P **sat** $S0$: $\#c \le \#d \le 1 \wedge \#d \le \#c \le 1$

Now, it is impossible to read from c first and from d first unless both reads occur simultaneously, and the model of execution does not allow this. Therefore, P cannot read *any* values, so another precise specification for P is

(5) P **sat** $S1$: $\#c = \#d = 0$

However, given (4) one cannot prove (5) in many earlier trace-based proof systems because $S0 \Rightarrow S1$ does not hold.

Although this example looks contrived, it illustrates a primary cause of the relative incompleteness in network proof systems. Since the network has only one process, the composition rule can't be at fault here. However, the use of the composition rule does tend to generate predicates that describe some states that cannot be reached in any computation of the network (as $S0$ does), and there is no way to eliminate these states from the specification. The problem is that properties of the model of execution of a network —e.g. that at most one event can occur at a time— have not been encoded in the proof system.

Widom [7] chooses the following axiom scheme, written in temporal logic, to eliminate the incompleteness shown by this example:

(6) *ORDERING*. Let c and d be channels and x and y integers satisfying the following: $0<x$, $0 \le y$, and either $x \ne y$ or c and d are distinct channels. Then

$$(\Box(\#c \ge x \equiv \#d \ge y)) \equiv (\Box(\#c < x \wedge \#d < y))$$

ORDERING can be interpreted as follows: constraining $c.x$ and $d.y$ to be written (and read) at the same time means they will never be written.

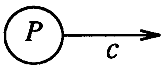
Since the length of a trace is at least 0, in the case $y=0$ and $x=1$ (6) reduces to $(\Box \#c \ge 1) \equiv (\Box \mathit{false})$, which means there is a state in which $\#c < 1$. Hence, axiom (6) includes the statement that traces are initially empty.

Suppose c and d are the same channel and $0 \le y < x$. Then *ORDERING* reduces to

$$(\Box(\#c \ge x \equiv \#c \ge y)) \equiv (\Box(\#c < x \wedge \#c < y))$$

This implies that more than one value cannot be appended to c at a single time.

Now consider the even simpler process



with given precise specification

(7) P **sat** $c \sqsubseteq [a,a] \vee c=[b,a]$

where $\sqsubseteq$ denotes 'prefix of'. Only one event can occur at a time, and it is not possible to change a trace that has been written, so state $c=[b,a]$ is never attained. Therefore, another precise specification for this network is

(8) P **sat** $c \sqsubseteq [a,a]$

However, (8) cannot be proved from (7) in the proof system unless the fact that written traces cannot be changed is encoded in the system. Widom [7] captures this in the axiom scheme

(9) *PREFIX*. For any channel c, $\Box(c \sqsubseteq \bigcirc c)$

where $\bigcirc$ is a restricted form of the next operator of temporal logic; $\bigcirc c$ denotes the value of c in the next state of the computation. This axiom scheme reads: it is always the case that a trace in a state is a prefix of the trace in the next state.

Another formulation that does not require $\bigcirc$ is

(10) *PREFIX*. For any channel c,
$$\Box \forall(i,v\colon 0 \le i < \#c\colon c.i=v \equiv \Box c.i=v)$$

Let us now change the rule of consequence (2) to

$$(11)\quad \frac{N \textbf{ sat } Q,\quad (\Box Q) \wedge \mathit{ORDERING} \wedge \mathit{PREFIX} \Rightarrow \Box R}{N \textbf{ sat } R}$$

The result is a relatively complete proof system. Further, as we have shown, the encoding of *ORDERING* and *PREFIX* is *necessary* if relative completeness is desired.

The relatively complete system of [1] (which we discovered after this work was done) is expressed in terms of conventional predicate logic extended to allow quantification over the infinite sequence of states produced during a computation (so that the interleaving of events on different traces could be expressed). Our simpler temporal-logic description allows us to pinpoint precisely the cause of incompleteness and its solution.

3. A minimal proof system

Our desire for relative completeness has forced us to consider temporal logic, and the question is: how much of it do we really need? Nguyen [6] went to the full temporal logic to achieve completeness. We have just shown that temporal logic restricted to the operator $\Box$ suffices.

In [7], Widom gives a certain formula F ((6.3.2) in [7]) —which is essentially *ORDERING* and *PREFIX*— and proves that it must be expressed in some fashion in order to achieve relative completeness. She proves that the only temporal operator (of $\Box$, $\Diamond$, $\bigcirc$ in linear-time temporal logic) that does the job is $\Box$; no combinations of the others suffice.

She goes one step further; (11) expresses formula F, and from (11), (10), and (6), one sees that to express F one needs only $\Box$ nested at most once. (That is, no formula contains $\Box(\ldots\Box(\ldots\Box\ldots))$.) She then proves that relative completeness can be achieved with a proof system whose axioms and inference rules consider only the predicate calculus on states and the temporal operator $\Box$ nested at most once.

4. Conclusion

In retrospect, it all seems obvious. The restrictions on computations in the model of execution must be encoded in the proof system. In other words, a

4

relatively complete logic must include enough reasoning power to discriminate between computations and sequences of states that look like computations but are not. These restrictions are expressed by *ORDERING* and *PREFIX*.

Since a specification N **sat** P (for P a first-order predicate on a single state) means that P holds in all states of all possible computations, which is expressed in temporal logic as $\Box P$, it should have been clear that any relatively complete proof system would require something like the power of temporal logic restricted to this operator. Widom was to able to show that a proof of N **sat** P requires at most one level of nesting of $\Box$ in any formula used in the proof.

Besides Widom's thesis [7], the reader can turn to [8] for a more detailed description of the material described in Sect. 2, and Widom is in the process of writing a paper describing the results mentioned in Sect. 3.

5. Acknowledgements

Hats off to Jennifer Widom for being such a pleasant, creative, and willing Ph.D. student. Thanks also go to Fred Schneider and Prakash Panangaden, who helped her when I was not able. Widom and Schneider commented on an earlier version of this note.

6. References

[0] Brock, J.D., and W.B. Ackerman. Scenarios: a model; of non-determinate computation. In *Formalization of Programming Concepts, LNCS 107*, 252-259, Springer Verlag, New York, 1981.

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

[2] Hoare, C.A.R. Communicating sequential processes. *CACM 21*, 8 (Aug. 1978), 666-677.

[3] Hoare, C.A.R. *Communicating sequential processes.* Prentice Hall, Englewood Cliffs, New Jersey, 1985.

[4] Keller, R.M. Denotational models for parallel programs with indeterminate operators. In *Formal Description of Programming Concepts* (E.J. Neuhold, ed.), 337-366, North Holland, New York, 1977.

[5] Misra, J., and K.M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng. 7*, 7 (July 1982), 417-426.

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

[7] Widom, J. *Trace-based network proof systems: expressiveness and completeness.* Ph.D. Thesis, Computer Science Department, Cornell University, May 1987.

[8] Widom, J., D. Gries, and F.B. Schneider. Completeness and incompleteness of trace-based network proof systems. *Proc. Fourteenth Ann. ACM SIGACT-SIGPLAN Symp. Princ. of Programming Languages*, January 1987, Munich.