Communication Traces in the Verification of Distributed Programs

Neelam Soundarajan
Computer and Information Science
The Ohio State University
Columbus, OH 43210, USA.

e-mail: neelam@cis.ohio-state.edu

Abstract:

Two types of communication traces, channel traces and process traces, have been used in the literature in dealing with distributed programs. Specifications and proofs in channel-trace systems are usually considered to be simpler than in systems based on process traces. But channel traces do not contain information about the relative order of communication along different channels of a process which can lead to incompleteness of the channel-trace based axiom systems. Several attempts have been made to overcome the incompleteness by adding new axioms to these systems.

We show with simple examples that these axioms do not by themselves solve the incompleteness problem. Effectively, process traces or some equivalent thereof is necessary to achieve completeness. We also consider the possibility of adding new communications to the processes to include more information in the channel traces.

Keywords and phrases: Communication traces, Program verification, Incompleteness.

1 Introduction

Auxiliary variables [3] are very useful in program verification. They are especially valuable in the verification of distributed programs since they allow us to account for interactions between the different processes of a program.[1] Although auxiliary variables may be used to express many different types of information about the program or process under consideration, perhaps the most useful type of auxiliary variables in dealing with distributed programs are those that represent sequences of communications that the processes engage in. This is not surprising since it is via these communications that all interactions between the processes take place, there being no shared variables. Further, there is also a 'naturalness' about these sequences; while other types of auxiliary variables have to be 'invented' by the program prover based on the particular program in question, communication sequences simply record the sequences of communications between the various processes. This is reflected in the proofs of individual programs; whereas for other types of auxiliary variables the program prover must introduce new (assignment) statements into the program to appropriately update the values of the auxiliary variables, the statements that update communication sequences are already in the program in the form of the input and output commands that the individual processes use to communicate with each other. It is also reflected in the proof systems; whereas systems that allow arbitrary auxiliary variables must include appropriate rules governing the introduction and removal of the variables from programs (see, for example, [1]), systems that include only communication sequences need only ensure that the rules corresponding to the input and output commands capture the effect of these commands on the communication sequences (see, for example, [7]).

Two types of communication sequences have been used in the literature. The first, which we will call *channel traces* or CTs, records the communications that a process P goes through on *one* of its channels, a separate channel trace being used to record P's communications along each of its channels. The second type of sequence, which we call *process traces* or PTs, records all the communications that P engages in along *all* of its channels on a *single* trace variable. The primary advantage of channel traces is ease of use. In particular when composing in parallel a set of processes $P_1, \ldots, P_n$, the channel traces of $[P_1 \parallel \ldots \parallel P_n]$ are obtained directly from the channel traces of the component processes. Thus if c is a channel

[1] Auxiliary variables are also important in dealing with shared-variable parallel programs [6].

along which $[P_1 \parallel \ldots \parallel P_n]$ communicates with an external process, c will also be a channel of exactly one of $P_1, \ldots, P_n$, say, P_i; hence the trace of $[P_1 \parallel \ldots \parallel P_n]$'s communications along c is identical to the trace of P_i's communications along c. The primary disadvantage of channel traces is the lack of information about the relative ordering of communications along different channels of a process.

The goal of this paper is to consider the question of exactly what information is missing from channel traces, discuss in detail some attempts –by adding some axioms to the proof system– that have been made in the literature to handle the resulting problems, and to show via relatively simple examples that channel traces have a fundamental incompleteness about them that cannot be resolved by the addition of such axioms. We should note that for most practical examples, channel traces are entirely adequate. But in all these examples, correctness proofs that use process traces are no more complex than those that use channel traces; indeed, any proof using channel traces can be translated in a straightforward manner into one that uses process traces. Our conclusion is that while channel traces are usually sufficient, (relative) completeness requires us to use process traces as the underlying mechanism in our proof systems.

The rest of the paper is organized as follows. In the next section we briefly describe the fairly standard notation that we use for processes, traces, and specifications. In the third section we present two examples that will illustrate the types of information that channel traces do not provide. The first example exhibits behavior like that of the classic Brock-Ackerman [2] example and we show in section 4 how the CT-based approach can, when strengthened with axioms such as those proposed by Widom et al [9], deal with it. (The original Brock-Ackerman example and even the simplified version in [5] are rather involved. Our example is simpler but illustrates the same point.) Our second example is new to this paper and we show in section 4 why general axioms such as those of [9] are inadequate to deal with the problem illustrated by this example. This seems to contradict the results of [9] and we discuss the apparent contradiction. Briefly, the problem is as follows: Widom et al [9] essentially prove that if we have 'sufficient information' about the component processes $P_1, \ldots, P_n$ of a program, then all valid conclusions about $[P_1 \parallel \ldots \parallel P_n]$, that are expressible in terms of the channel traces of $[P_1 \parallel \ldots \parallel P_n]$ can be derived using their axioms. Our example shows, however, that in some cases the required information about the component processes $P_1, \ldots, P_n$ cannot be expressed in terms of the channel traces of those processes although the result about $[P_1 \parallel \ldots \parallel P_n]$ can be expressed in tems of its channel traces. And this is not a problem of the *assertion language* used to express these properties; rather it is a result of the nature of channel traces. In section 4 we also briefly relate our discussion to the models of Jonsson and Kok [4]. Jonsson and Kok introduce the notion of *full abstractness* to represent whether or not a particular model contains enough information to avoid Brock-Ackerman type of problems.

In section 5 we reiterate our conclusions and briefly mention an extension to channel traces that may allow us to handle the incompleteness problem in a very different manner. Interestingly, this approach would require us to introduce new statements into the processes to include additional information in the channel traces in the spirit of the original auxiliary variables. We should note that we have not fully explored the ramifications of including such additional information in the traces, nor have we formalized axioms and rules corresponding to this extension. We hope to do that in a future paper.

2 Processes, Traces, and Specifications

We use a standard CSP-like notation with processes communicating along named channels. A process may either be a sequential process or the parallel composition $[P_1 \parallel \ldots \parallel P_n]$ of n other processes. A sequential process is made up of the standard statements: **skip**, assignment, sequential composition, selection, and repetition. Selection and repetition may have input as well as output guards.

Consider a parallel composed process $[P_1 \parallel \ldots \parallel P_n]$. Let I_i, O_i be the sets of input and output channels of P_i. We will make the standard assumption that $P_1, \ldots, P_n$ are 'compatible', i.e., that no channel appears in more than one I_i or in more than one O_i. (We have borrowed some of our terminology and notation from [4].) If c appears in $I_i \cap O_j$ then it is a channel along which P_j can send values to P_i; such a channel is an *internal* channel of $[P_1 \parallel \ldots \parallel P_n]$, communications along which are, of course, not visible to an external observer of $[P_1 \parallel \ldots \parallel P_n]$. If c appears in O_j but not in $\cup I_i$, it is an *external* channel of $[P_1 \parallel \ldots \parallel P_n]$ and is used by $[P_1 \parallel \ldots \parallel P_n]$ for outputting values to the 'environment' – more precisely to the process at the other end of c. Similar remarks may be made if $c \in I_i \wedge c \notin \cup O_j$.

Let P be a process, sequential or parallel composed. An external observer of P will see all the communications that P engages in on its external channel but none of its internal activities including communications on any internal channels (in the case of parallel composed processes). The observer can record the entire sequence of P's communications with external agents and this record h is the process trace of P. Thus h is a sequence of elements each representing a communication between P and an external process, an element of h being of the form (c, d) where c is the channel and d the value communicated. Communications in which P receive a value and those in which it sends a value are both recorded in the same manner. Alternately, we may associate a separate channel trace h_c with each channel c of P and record all communications that P engages in along this channel in h_c. Each element of h_c is just a value there being no need to record the identity of the channel. Given the value of the process trace h at any time, we can obtain the corresponding channel traces of P, say h_c in particular, by simply projecting out those elements of h whose first component is c and then omitting this first component from each element. On the other hand, given the values of all the channel traces of P at some time we cannot, in general, obtain the corresponding h since we do not know the relative order of communications along the different channels.

Consider next the specification of a process. One possibility is to specify the set of possible values that the process trace h, in the case of the PT-based approach, or the various sets of possible values that the channel traces of P in the case of the CT-based approach, may have when P finishes. A better alternative, especially for non-terminating processes, would be an *invariant* that specifies the possible values that h or the set of values that the channel traces of P may have at *any* time during execution. We will use invariants not only because of their ability to deal with non-terminating processes, but also because, as we will soon see, they add considerable power to the CT-based system even for terminating processes. We will not formally define a language of assertions for the invariants. Something along the lines of the STL (Simple Trace Logic) of Widom [8] would be appropriate.

Let us consider a simple example. Suppose B is a bounded buffer of size n that reads values on channel c and outputs them on d. $A[0..(n-1)]$ is the internal array in which B stores the values it has input but not yet output. in, out keep track of how many values have been input and output respectively:

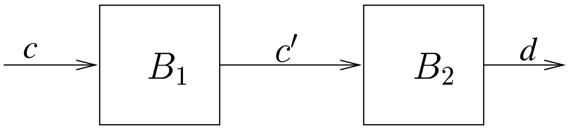
$$
\begin{array}{l}
in := 0; out := 0; \\
*\ [in < (out + n);\ c?A[in \bmod n] \rightarrow in := in + 1 \\
\quad |\ out < in;\ d!A[out \bmod n] \rightarrow out := out + 1]
\end{array}
$$

A CT-invariant r for B would be:

$$r :: \quad [h_d \preceq h_c \wedge \#h_c - n \leq \#h_d \leq \#h_c]$$

where $\preceq$ denotes 'prefix of', and $\#h$ is the number of elements in h. Thus r says that the sequence of values B has output on channel d is a prefix of the sequence of values it has input on c, with the length of the former being no more than n less than the length of the latter.

Suppose we have two bounded buffers B_1, B_2; B_1 is exactly like B except it has a capacity n_1, inputs on channel c, and outputs on channel c'; B_2 is also like B except it has a capacity n_2, inputs on c', and outputs on d. Pictorially



The invariant for B_1 is:

$$r_1 :: \quad [h_{c'} \preceq h_c \wedge \#h_c - n_1 \leq \#h_{c'} \leq \#h_c]$$

and for B_2:

$$r_2 :: \quad [h_d \preceq h_{c'} \wedge \#h_{c'} - n_2 \leq \#h_d \leq \#h_{c'}]$$

The invariant for $[B_1 \parallel B_2]$ can be obtained by just conjuncting r_1 and r_2 and eliminating references to communications on the internal channel c':

$$\begin{aligned} r_1 \wedge r_2 :: \quad & [h_d \preceq h_{c'} \preceq h_c \wedge \#h_c - n_1 \leq \#h_{c'} \leq \#h_c \\ & \wedge \#h_{c'} - n_2 \leq \#h_d \leq \#h_{c'}] \end{aligned}$$

Simplifying and eliminating references to channel c', we get, as invariant for $[B_1 \parallel B_2]$:

$$r_{12} :: \quad [h_d \preceq h_c \wedge \#h_c - (n_1 + n_2) \leq \#h_d \leq \#h_c]$$

This invariant shows that $[B_1 \parallel B_2]$ behaves like a buffer, copying values from c to d as we would expect. Note that we have used the same name $h_{c'}$ for the trace of values output by B_1 on c' and the trace of values input by B_2 on c'. This is a fairly standard trick but, of course, it is ok only if i/o is synchronous (so that these traces are, in fact, always equal). If i/o were asynchronous, we would have to use distinct names, say, $h^{out}_{c'}, h^{in}_{c'}$ for these traces and add the condition $h^{in}_{c'} \preceq h^{in}_{c'}$ at parallel composition, i.e., when combining r_1, r_2 to obtain the invariant for $[B_1 \parallel B_2]$. For convenience we will assume synchronous communications in our discussion.

In a PT-based system, since the invariants r_1, r_2 would be expressed in terms of

h_1, h_2 the process traces of B_1, B_2, things would seem to be more complex since to obtain the process trace h of $[B_1 \parallel B_2]$, we would have to 'merge' h_1, h_2, and then omit the elements corresponding to communications on the internal channel c'. (For details, see for example [7].) In fact though, in examples like $[B_1 \parallel B_2]$, the situation in the PT-based system is actually not very different than in the CT-based system since r_1, r_2 can be expressed in terms of projections of h_1 on the channels c, c' and the projections of h_2 on c', d respectively. Once this is done, the invariants in the PT-based system look exactly like the invariants in the CT-based system and combining these to obtain the invariant for the parallel composition is no harder than in the CT-based system since the 'merge' operation in this case reduces to identifying, for instance, the projection of h_1 on c' with the projection of h_2 on c', the projection of the trace h of the parallel composition on the channel c with the projection of h_1 on c etc. In general, given a CT-based proof, we can similarly translate it into an equivalent PT-based proof.

Consider again the buffer process B. Consider the value $\langle (d,0),\ (c,0),\ (c,1),\ (d,1),\ (c,2)\rangle$ for the process trace h of B. The corresponding values for h_c and h_d are $\langle 0,1,2\rangle$ and $\langle 0,1\rangle$ respectively. These values satisfy the invariant r, but this particular trace h cannot in fact arise during the execution of B (since the value 0 appears on the channel d before it has been input on the channel c). In a process trace based approach we can easily fix this problem by strengthening the invariant as follows:

$$r' :: \ \ \forall h' \preceq h.\ [h'_{/d} \preceq h'_{/c} \wedge \#h'_{/c} - n \leq \#h'_{/d} \leq \#h'_{/c}]$$

where $h'_{/c}$ is the sequence obtained from h' by retaining only the elements communicated along the channel c; in other words, it is the channel trace on c corresponding to the process trace h'; $h'_{/d}$ is similar. This invariant allows us to conclude that the given trace $\langle (d,0),\ (c,0),\ (c,1),\ (d,1),\ (c,2)\rangle$ cannot arise during the execution of B since if we take h' to be $\langle (d,0)\rangle$, it will not satisfy the conditions in r'.

Interestingly, Widom et al [9] showed that there is no need to resort to process traces to solve this problem. The intuition behind their approach was that although the original invariant r does not directly rule out the trace specified in the last paragraph, we can still argue that it is not an acceptable value for h according to r: Since r is an *invariant*, not only must it be satisfied at the *current* moment, but it must also have been satisfied at all moments in the *past*. But this would not have been the case immediately following the communication recorded in the first element of the h in question since at this point the corresponding values of the channel traces h_c, h_d would have been $\langle\rangle$ (the empty trace) and $\langle 0\rangle$ respectively and this pair of values would not have satisfied the invariant r.

Using this type of reasoning, one can in general extract, from the CT-based invariant, considerable amount of information on the relative ordering of communications along different channels. Widom et al proposed two general axioms, *prefix* and *ordering*, to capture such properties that follow from the fact that we are dealing with invariants over (channel) traces; the idea being that these axioms would allow us to derive, on the basis of invariants over channel traces, all the information that we would need about the communications, and that there would never be any need to resort to process traces. One of the examples in the next section will show that this is not possible in general.

3 The Missing Information

We present two examples in this section. The first one is similar, as far as the problem it demonstrates is concerned, to the Brock-Ackerman [2] example. This problem, as we will see in the next section, is handled by the [9] axioms. Our second example will demonstrate a different type of problem with the CT-based systems that cannot be handled by such axioms.

Before we present the examples, we should make some general remarks. The bounded buffer example of section 2 is not really a good example of incompleteness in the CT-based system. This is because the particular property of the buffer we considered –that the output of 0 on channel d must have taken place after the input of 0 of channel c– is not expressible using channel traces because once the communications on c and d have taken place, there is no way to talk about the order of those communications. So one could validly argue that there is no need to worry about this property or about the fact that it is not implied by the CT-based invariant r. The importance of the Brock-Ackerman example is that it presents a property p of a process $[P_1 \parallel P_2]$ that *is* expressible in terms of its channel traces and that also intuitively follows from the (channel-trace) invariants r_1 and r_2 of P_1 and P_2, but is not implied by $r_1 \wedge r_2$. The axioms of [9] solves this problem since in conjunction with those axioms $r_1 \wedge r_2$ does imply p.

Our second example behaves differently. In that example a process $[P_3 \parallel P_4]$ has a property q that is expressible in terms of its channel traces. But the properties of P_3 and P_4 that lead to $[P_3 \parallel P_4]$ having the property q are not themselves expressible in terms of their channel traces. It is this kind of example that general axioms such as those of [9] cannot help with.

Let us now turn to the actual examples. The first is the parallel composition of 2 processes $[P_1 \parallel P_2]$ where

$P_1 :: c_1!0; c_2!0$, and

$P_2 :: c_2?x; c_1?y; d!0$

The invariants r_1, r_2 for P_1, P_2 (in r_1, r_2 below as well as in the rest of the paper, we will use c rather than h_c to denote the trace of communications on the channel c; this should cause no confusion since the context will make it clear whether we are talking about the channel or the corresponding channel trace):

$$r_1 : [(c_1 = \langle\rangle \wedge c_2 = \langle\rangle) \vee (c_1 = \langle 0 \rangle \wedge c_2 = \langle\rangle) \vee (c_1 = \langle 0 \rangle \wedge c_2 = \langle 0 \rangle)\ ]$$

$$\begin{aligned} r_2 : [&(c_1 = \langle\rangle \wedge c_2 = \langle\rangle \wedge d = \langle\rangle) \vee (c_1 = \langle\rangle \wedge \#c_2 = 1 \wedge d = \langle\rangle) \\ &\vee (\#c_1 = 1 \wedge \#c_2 = 1 \wedge d = \langle\rangle) \vee (\#c_1 = 1 \wedge \#c_2 = 1 \wedge d = \langle 0 \rangle)\ ] \end{aligned}$$

Combining r_1, r_2, we get

$$r_1 \wedge r_2 : [(c_1 = \langle\rangle \wedge c_2 = \langle\rangle \wedge d = \langle\rangle) \vee (c_1 = \langle 0 \rangle \wedge c_2 = \langle 0 \rangle \wedge d = \langle 0 \rangle)\ ]$$

Eliminating references to the internal channels c_1, c_2, we get, as the invariant for $[P_1 \parallel P_2]$:

$$r_{12} : [(d = \langle\rangle) \vee (d = \langle 0 \rangle)]$$

From this there is no way to see that $[P_1 \parallel P_2]$ will not output any values on d. But as we will see in the next section, $r_1 \wedge r_2$, in conjunction with the axioms of [9], will let us derive this.

Our next example is more involved. It is the parallel composition of 2 processes $[P_3 \parallel P_4]$ where

$P_3 :: [c_3!0 \rightarrow c_4!0;\ e!0 \| c_4!0 \rightarrow c_3!0;\ e!1]$, and

$P_4 :: [c_3?x \rightarrow c_4?y;\ f!0 \| c_4?y \rightarrow c_3?x;\ f!1]$

The invariants r_3, r_4 for P_3, P_4 are:

$$\begin{aligned} r_3 : [&(c_3 = \langle\rangle \wedge c_4 = \langle\rangle \wedge e = \langle\rangle) \vee (c_3 = \langle 0 \rangle \wedge c_4 = \langle\rangle \wedge e = \langle\rangle) \\ &\vee (c_3 = \langle 0 \rangle \wedge c_4 = \langle 0 \rangle \wedge e = \langle\rangle) \vee (c_3 = \langle 0 \rangle \wedge c_4 = \langle 0 \rangle \wedge e = \langle 0 \rangle) \\ &\vee (c_3 = \langle\rangle \wedge c_4 = \langle 0 \rangle \wedge e = \langle\rangle) \vee (c_3 = \langle 0 \rangle \wedge c_4 = \langle 0 \rangle \wedge e = \langle 1 \rangle)\ ] \end{aligned}$$

The reader might already see the beginnings of a problem in r_3. There is no way to see the relation between the value output on e and the order of the previous communications on c_3, c_4. And there is no way to strengthen r_3 to include information about this relation since each combination of values for c_3, c_4, e allowed by r_3 can actually arise during the execution of P_3. We may, for now, simply ignore this relation but when we consider $[P_3 \parallel P_4]$, we will see that without information about this relation (and a similar relation for P_4) we will be unable to establish a property of that process although the property is easily stated in terms of its channel traces.

$$\begin{aligned}
r_4 : [&(c_3 = \langle\rangle \wedge c_4 = \langle\rangle \wedge f = \langle\rangle) \vee (\#c_3 = 1 \wedge c_4 = \langle\rangle \wedge f = \langle\rangle) \\
&\vee (\#c_3 = 1 \wedge \#c_4 = 1 \wedge f = \langle\rangle) \vee (\#c_3 = 1 \wedge \#c_4 = 1 \wedge f = \langle 0\rangle) \\
&\vee (c_3 = \langle\rangle \wedge \#c_4 = 1 \wedge f = \langle\rangle) \vee (\#c_3 = 1 \wedge \#c_4 = 1 \wedge f = \langle 1\rangle)\ ]
\end{aligned}$$

Combining the two:

$$\begin{aligned}
r_3 \wedge r_4 : [&(c_3 = \langle\rangle \wedge c_4 = \langle\rangle \wedge e = \langle\rangle \wedge f = \langle\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle\rangle \wedge e = \langle\rangle \wedge f = \langle\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle\rangle \wedge f = \langle\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle 0\rangle \wedge f = \langle\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle 1\rangle \wedge f = \langle\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle\rangle \wedge f = \langle 0\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle\rangle \wedge f = \langle 1\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle 0\rangle \wedge f = \langle 0\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle 1\rangle \wedge f = \langle 1\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle 0\rangle \wedge f = \langle 1\rangle) \\
&\vee (c_3 = \langle 0\rangle \wedge c_4 = \langle 0\rangle \wedge e = \langle 1\rangle \wedge f = \langle 0\rangle)\ ]
\end{aligned}$$

Simplifying and omitting references to the internal channels c_3, c_4, we get

$$r_{34} :\ [\,[(e \preceq \langle 0\rangle) \vee (e \preceq \langle 1\rangle)] \wedge [(f \preceq \langle 0\rangle) \vee (f \preceq \langle 1\rangle)]\,]$$

But we can write a stronger invariant for $[P_3 \parallel P_4]$ to express the equality of values output on e, f:

$$R_{34} :\ [\,[(e \preceq \langle 0\rangle) \wedge (f \preceq \langle 0\rangle)] \vee [(e \preceq \langle 1\rangle) \wedge (f \preceq \langle 1\rangle)]\,]$$

but, as we will see in the next section, there is no way to derive this from $r_3 \wedge r_4$ even with the help of the [9] axioms.

4 The Incompleteness of Channel Traces

Consider the first example $[P_1 \parallel P_2]$ from section 3. The conjunction of r_1, r_2, the invariants of P_1, P_2 is:

$$r_1 \wedge r_2 : [(c_1 = \langle\rangle \wedge c_2 = \langle\rangle \wedge d = \langle\rangle) \vee (c_1 = \langle 0 \rangle \wedge c_2 = \langle 0 \rangle \wedge d = \langle 0 \rangle)\ ]$$

This assertion does, using the special kind of reasoning from section 2, let us conclude that no value will be output on d since for that to happen a value must have also been output on c_1 and on c_2. But if we consider the situation when only one or two of these three values have been output, the corresponding values of the channel traces c_1, c_2, d will not satisfy $r_1 \wedge r_2$ and hence, since there is no way for the process to *simultaneously* communicate on all three channels, no values will be output on any of the channels. In particular no value will be output on d.

But this type of argument does not apply once we abstract away the internal channels c_1, c_2 and obtain the 'proper' invariant (i.e., the invariant that refers only to $[P_1 \parallel P_2]$'s external channels:

$$r_{12} : [(d = \langle\rangle) \vee (d = \langle 0 \rangle)]$$

To solve this problem, [9] introduce two new axioms. Their *prefix* axiom says that the value of a channel trace at any time will be a proper prefix of its value at the 'next' moment. Essentially this says that communications cannot be 'taken back' once they have occurred. Their second axiom, the *ordering axiom*, essentially says that if (an invariant asserts that) two or more communications must take place simultaneously (or not at all) then they will not take place at all. They also go on to show that there is no way to express these axioms without using temporal operators and Widom [8] analyses in depth exactly 'how much' of temporal notations are needed to express these axioms (the point being that if we simply allow all the power of temporal logic (as do [5]), we would lose the simplicity of the CT-based approach and such a system would be even more complex than the PT-based system). For our purposes though, it is not necessary to look at the precise formal expressions of these axioms. All we need is to note that using these axioms we can formalize the type of informal arguments we have considered. This essentially takes care of our first example $[P_1 \parallel P_2]$ since we can conjunct the ordering axiom to $r_1 \wedge r_2$ and derive the stronger invariant:

$$[(c_1 = \langle\rangle \wedge c_2\langle\rangle \wedge d = \langle\rangle)]$$

We can then omit the references to the internal channels and derive the required invariant $[d = \langle\rangle]$.

Consider now our second example. By looking at r_3 (and r_4) there is, as we saw, no way to tell what the relation between the value output on e (and f) and the the order of prior communications on c_3, c_4. We can see this by noting that the process

$$P'_3 :: [c_3!0 \rightarrow c_4!0 [\!] c_4!0 \rightarrow c_3!0];\ [e!0 \rightarrow \textbf{skip} [\!] e!1 \rightarrow \textbf{skip}\,]$$

also satisfies r_3 and for this process there is indeed no relation between the value output on e and the order of communications on c_3, c_4. A similar process P'_4 satisfies r_4 again with no relation between the value output on f and the order or communications on c_3, c_4. Given these facts it is clear that the [9] axioms cannot be used to strengthen $r_3 \wedge r_4$ to allow us to establish the equality of the values output on e and f by $[P_3 \parallel P_4]$ since otherwise we will be able to do the same for $[P'_3 \parallel P'_4]$, and this equality does not hold for $[P'_3 \parallel P'_4]$.

The only other possibility would be to try to arrive at stronger invariants for P_3 and P_4; P'_3 and P'_4 will presumably not satisfy these stronger invariants. But if we are using a CT-based approach, no stronger invariant than r_3 and r_4 can be valid for P_3 and P_4 since every one of the combinations of values for the traces c_3, c_4, and e (f) allowed by r_3 (r_4) can actually arise during execution. So no strengthening of the invariants of P_3, P_4 is possible. If we cannot strengthen r_3, r_4 and we should not even try to strengthen $r_3 \wedge r_4$ (given the existence of $[P'_3, P'_4]$), the conclusion has to be that there is no way to establish the relation between the values output on channels e and f by $[P_3 \parallel P_4]$ in a CT-based system.

Widom et al introduce the notion of 'precise invariants' in their paper. Informally, an invariant is precise if every combination of (channel) trace values that satisfies the invariant either can actually arise during the execution of the process or is such that 'somewhere along the way to reaching this set of values' the traces would have to go through one or more combinations of values that do not satisfy the invariant. (The idea being that in that case we can use their axioms to strengthen the invariant to be satisfiable only by those combinations of trace values that can actually arise during execution.) They then proceed to show that given precise invariants of component processes, we can derive a precise invariant for the parallel composed process. This seems to be contradicted by our example. r_3, r_4 are precise invariants for P_3, P_4 since each combination of trace values allowed by r_3, r_4 can arise in practice, yet $r_3 \wedge r_4$ is not precise for the process $[P_3 \parallel P_4]$ as we saw.

The problem is that though the examples that they consider suggest the above definition of preciseness, Widom et al's formal definition of preciseness is stronger. To see this definition let us introduce the notion of a 'state' of a process as the set of values of all the channel traces of the process at any given time. Consider now a sequence of states such that each state can arise from the previous one by performing one communication on one channel, in other words there are no violations of the axioms of [9] in going from state to the next in the sequence. Suppose also that each state in the sequence satisfies the invariant r in question. Then we will say that r is 'path precise' (this is not a term that [9] use; we are using this term to distinguish it from the notion introduced in the last paragraph) if this sequence of states can actually arise during the execution of the process. Note that it is not enough that each *state* can arise; the entire *sequence* must be capable of arising during execution. It is clear that path preciseness implies preciseness but the converse is not true. In particular r_3, r_4 are *not* path precise for P_3, P_4 since r_3 for instance is satisfied by each state in the following sequence:

$$\begin{aligned}&\langle\ (c_3 = \langle\rangle, c_4 = \langle\rangle, e = \langle\rangle), (c_3 = \langle 0\rangle, c_4 = \langle\rangle, e = \langle\rangle),\\ &\quad (c_3 = \langle 0\rangle, c_4 = \langle 0\rangle, e = \langle\rangle), (c_3 = \langle 0\rangle, c_4 = \langle 0\rangle, e = \langle 0\rangle)\ \rangle,\end{aligned}$$

and each state in this sequence can be obtained from the previous one by a single communication, but this sequence cannot arise during the execution of P_3.

In their proof that from precise invariants of component processes one can derive precise invariants of the parallel composition it is this stronger definition of preciseness that [9] use, not the weaker one we considered earlier. But if we assume that we must start with specifications of component processes which are (path) precise, we have a serious problem. There is no way to specify, in first order logic, such invariants for P_3 and P_4; as we saw r_3, r_4 are the strongest such invariants these processes satisfy. We would have to instead move to temporal assertions which means that much of the promise that the [9] approach held, that of retaining the simplicity of the first order CT-based system, will not be met. Indeed once we have such temporal assertions, there would be no need for the axioms proposed by Widom et al because once we allow such temporal invariants, r_3 and r_4 can include all the ordering information that are specifiable using process trace invariants.[2]

Before concluding this section we briefly consider the models that Jonsson

[2]Zwiers [10] introduces a notion of *adaptation completeness* which can be achieved using the axioms of [9]. But as we just saw, these axioms do not allow us deal with the lack of information in channel traces that is manifested in processes like $[P_3 \parallel P_4]$.

and Kok [4] prove are 'fully abstract'. Informally, a model is fully abstract if it contains all (but no more) information necessary to distinguish between processes that exhibit differences in their communications with the outside world.[3] It is important for the model to contain this information since otherwise, as we saw in the examples, these differences can be translated into effects that can show up in the channel trace specifications of processes constructed out of these processes. Jonsson and Kok [4] establish the full abstractness of three models. The first is the process trace model. The remaining two models are the ones of interest to us. Both of these use channel traces but the semantics of a process is not just the set of values of its channel traces; rather it is a set of functions whose values are essentially all the legal sequences of states, where by a state we mean, as before, the set of values of all the channel traces. But given such a legal sequence, we can directly map it to the corresponding process trace and this is exactly why these models are fully abstract. (The main difference between the two models that [4] consider are in the details of how the functions are specified.) The problem, of course, is that the simplicity which was the original advantage of the CT-based system over the PT-based system is lost since specifying these legal sequences, or combining the sets of legal sequences of component processes to obtain the set of legal sequences of the parallel composition, is no easier than the corresponding task in the PT-based system.

5 Discussion

Is it likely that stronger axioms than those [9] can be found using which we can achieve completeness in a CT-based system while using only first order assertions for invariants of processes? The answer, as indicated by our $[P_3 \parallel P_4]$ and $[P_3' \parallel P_4']$, is almost certainly no. The problem is that even a 'small amount of missing

[3] It is worth noting here that the term 'fully abstract' is perhaps a bit confusing. The problem with the channel trace models and some of the others analyzed by [4] is not that they are not abstract enough but that they are *too abstract*; the problem is not that they contain inappropriate information (such as, for instance, the details of the internal workings of a process) but that they do not contain enough information about the external behavior of the processes in question. In order for a model to be fully abstract according to the definition of [4], the model is required not only not to contain too much information (such as about the internals of the process) but also not contain too little information; it is this latter aspect that we are concerned with in this paper. Perhaps something like *sufficiently expressive* rather than fully abstract might be a better term for this.

information' about the ordering between communications on separate channels can be converted by composing the given process with an appropriately cleverly designed process to manifest itself as influencing the actual values that are output on specific channels. In fact, [4] (in an appendix to their paper) show how to construct such a process quite simply. If we could recover this information using general axioms, then we would also be able to apply these axioms to a differently designed example where the missing information ought not to be derived (because it is not valid) and we would have a serious problem.

But there may be another possible approach. The problem with channel traces is that the information they contain is not sufficient to allow us to discriminate between P_3 and P_3' and similarly between P_4 and P_4'. But what if we *added* the required information to the channel traces? This would be like introducing appropriate auxiliary variables and introducing assignment statements to update them at appropriate points so that they contain the required information. Similarly we may be able to introduce additional communication commands to include additional information in the individual elements of the (channel) traces. For instance, in the particular example, we could include information immediately following a communication on channel c_3 whether or not the communication on c_4 has taken place, *and* specify how the value communicated later on the channel e depends on that earlier information. We might then be able to combine these pieces of information from P_3 and P_4 to show that the values communicated by $[P_3 \parallel P_4]$ on channels e and f are equal to each other. This is obviously very preliminary and many details remain to be worked out but it looks promising. One problem we must deal with is the possibility that the additional communication commands we introduce do not influence the behavior of the processes; in particular, we certainly do not want to introduce deadlocks into the system! One easy way of ensuring this would be to allow such auxiliary communication commands to be only 'piggy-backed' onto existing communication commands. We plan to investigate this possibility and report on the results in a future paper.

6 References

1. K. Apt, N. Francez, W. deRoever, A proof system for CSP, ACM TOPLAS, vol. 2, pp. 359-385, 1980.

2. J. Brock, W.B. Ackerman, Scenarios: a model of non-deterministic compu-

tations, LNCS 107, 1981, pp. 252-259.

3. M. Clint, Program proving: coroutines, Acta Informatica, vol. 2, pp. 50-63, 1972.

4. B. Jonsson, J.N. Kok, Comparing two fully abstract dataflow models, PARLE 1989, LNCS, pp. 217-234.

5. V. Nguyen, A. Demers, D. Gries, S. Owicki, A model and temporal proof system for networks of processes, Distributed Computing, 1, 1986.

6. S. Owicki, D. Gries, An axiomatic proof technique for parallel programs, Acta Informatica, vol.6, pp. 319-339, 1976.

7. N. Soundararajan, Axiomatic semantics of CSP, ACM TOPLAS, 6, pp. 647-662, 1984.

8. J. Widom, Trace based network proof systems, Ph.D. thesis, Cornell University, 1987.

9. J. Widom, D. Gries, F. Schneider, Completeness and incompleteness of network proof systems, 14th Symp. on Principles of Prog. Languages, 1987, pp. 27-38.

10. J. Zwiers, Compositionality, concurrency, and partial correctness, LNCS 321, 1987.