Complete, Effective and Abstract System For Reasoning About Networks of Processes

Abha Moitra*
Catherine Wagner**

TR 89-977
March 1989

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

*Research supported by NSF grant DCR-86-02307.
**Research supported by NSF grant DCR-87-06851.

Complete, Effective And Abstract System For Reasoning About Networks Of Processes

Abha Moitra*

Catherine Wagner†

Odyssey Research Associates
1301A Harris B. Dates Drive
Ithaca, NY 14850

Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract

We present a complete recursive set of axioms for reasoning about networks of bounded asynchronous processes with finite resources. We also present an effective procedure for hiding internal channels of a network. Our processes use finite resources and bounded asynchronous communication. Such processes and networks can be physically realized. Our processes can be specified by means of regular expressions and do not require the full theory of arithmetic. The assertion language we use exploits this fact and allows us to specify all such processes and networks and obtain a complete and effective proof system. The proofs of completeness are model theoretic and make extensive use of Ehrenfeucht games. An algorithm for channel hiding follows from the way these processes can be modeled.

1 Introduction

The development of proof systems for networks of processes has been studied extensively. Unfortunately, in such proof systems there is a conflict between obtaining completeness and maintaining effectiveness. In most earlier work, no explicit bound was placed on the resources used by the processes. For such systems, complete and effective axiom schemes cannot be given. Completeness either has been obtained by adding essentially all true

*Work done while at Cornell University was supported in part by NSF grant DCR-86-02307

†Research supported in part by NSF grant DRC-87-06851

statements as axioms, or has depended on the existence of a complete proof system for predicate temporal logic. Neither solution is effective. In addition, no adequate procedure has been found for removing references to the internal channels of a network, and no thorough analysis of the expressiveness of the languages has been given.

In this paper we obtain both completeness and effectiveness. By restricting ourselves to bounded asynchronous processes with finite resources, we are able to give (i) a language in which all such processes can be specified, (ii) a complete proof system with a recursive set of axioms, and (iii) an effective procedure for hiding the internal channels. An additional advantage of this restriction is that all our processes can be physically realized. A close match exists between our formal language and the structure of the processes, allowing us to physically realize processes corresponding to all allowable specifications in our language.

The paper is organised as follows. In section 2 we discuss the definition of a precise specification of a process. In section 3 we discuss how previous works have achieved relative completeness, and argue that such an approach does not lead to an effective proof system or allow for channel hiding. In section 4 we introduce the processes and networks that we will be considering in this paper. In section 5 we show the close relationship that exists between our processes and regular expressions. In section 6 we motivate the choice of an assertion language with just the right expressive power that will enable us to obtain an effective and complete proof system and will also allow us to do channel hiding. In section 7 the assertion language is presented. In section 8 we show how network composition with channel hiding can be done. In section 9 we prove that the axiom system is both complete and effective. Finally, in section 10 we state the conclusions and some future work.

2 Precise Specification of a Process

In this section we consider general purpose processes, either deterministic or non-deterministic. We discuss the model of behavior usually used in the definition of a specification of a process and in the analysis of trace-based logics. We investigate the difficulties that arise when using only this model in the definition of a *precise specification* and propose a stronger definition.

The sequence of communication that a process P engages in during one computation is called a *trace*. It is a sequence of elements from the alphabet of the process P. A trace of P gives one possible behaviour of P. The goal of reasoning in trace-based logics is to find a specification (either over the interleaved trace or over individual channel traces) for a process that involves only input and output events, and from which any other specification for the process can be derived. In addition, the logical theory and specifications used should be strong enough to provide compositionality in the building of networks.

The framework normally used in selecting the desired specification relies on the concept of a model of behavior for the process in question. Each such model consists of a sequence of sets of channel traces with the following properties:

1. Each set in the sequence contains a trace for each channel.
2. Each element of the sequence differs from its predecessor by the addition of exactly (or at most) one element to the end of the trace of exactly one channel.
3. The first element of the sequence contains the empty trace for all channels.
4. The sequence of sets of channel traces gives the history of the events occurring on the channels during an initial segment of a possible computation of the process.

If we assume that for a given process no two events occur at the same time, then these models give a complete description of the input-output behavior of the process during a possible computation.

Using this definition of a model of behavior, the concept of a *precise specification* is introduced. Given a formal language, $\mathcal{L}$, and a statement Ψ in the language, Ψ is called a *specification* for a process P if every possible behavior of the process always satisfies Ψ. It is called a *precise specification* if, in addition, every behavior always satisfying Ψ is a possible behavior of P.

Although this is a very natural definition for the notion of a precise specification, it leads to difficulties when we are concerned with the provability of other specifications within a given logical framework.

Ex. 1. Consider the following process that is analyzed in [WGS87]. The process P has two channels, labeled ch_1 and ch_2, and an output alphabet containing only the letter a. An a is produced on ch_1 if and only if an a was already produced on ch_2. Similarly, an a is produced on ch_2 if and only if an a was already produced on ch_1. Note that P never produces any output on either channel. Let Ψ be defined as:

$$\Psi \equiv ((tr(ch_1) = \epsilon) \ \wedge \ (tr(ch_2) = \epsilon)) \vee ((tr(ch_1) = <a>) \ \wedge \ (tr(ch_2) = <a>))$$

According to the above definition, Ψ is a precise specification for P. However, as is pointed out in [WGS87], if the trace logic used for a deductive system does not contain axioms stating that events on different channels cannot occur at the same time, then the system will be too weak to prove that Ψ implies Φ, where Φ is:

$$\Phi \equiv (tr(ch_1) = \epsilon) \ \wedge \ (tr(ch_2) = \epsilon)$$

Since Φ is also a precise specification for P, it is desirable to be able to prove that Ψ implies Φ within the logic.

The problem that arises in this example, however, is due to the model used in the usual definition of a precise specification. The model of behavior described above is a model that is external to the theories developed. If the test for a precise specification is linked more closely to the logical system and its models, the failure to achieve completeness in this example and many others goes away.

We now present a sharper definition of the concept of a precise specification, one that is more closely tied to the languages and theories used in reasoning about networks. The definition we propose eliminates the problems that are due to the inappropriateness of the model of behavior used and instead focuses on those problems that are due to the weakness of the language and/or axiom system itself.

Because we are interested in reasoning about the actual behavior of processes, the models of our formal theory that are of interest are those that we refer to as *standard* models. These are the models which contain the standard model of the natural numbers. With this in mind we make the following definition.

Definition 1. Let the symbol ω denote the usual set of natural numbers. We call a model B of a theory of traces *standard* if the set, N, of natural numbers contained in B is just ω.

Definition 2. Let $\mathcal{L}$ be a language for traces containing a possibly infinite collection of names for channels and a finite input-output alphabet. Let T be a theory of traces in $\mathcal{L}$ that is satisfied by all processes. For a given process P and standard model A of T, we say that A is a T-model of P if the traces in A form a possible history of input-output behaviour of P. Let Ψ be a statement in $\mathcal{L}$. We say that Ψ is a *T-specification* of the process P if every T-model of P satisfies Ψ. It is a *T-precise specification* if, in addition, every *standard* model of $T \cup \{\Psi\}$ is T-model of P.

Note that in our definition of a T-precise specification, the models of behavior of P used are models in the theory T. Thus, in the example given above, the statement Ψ fails to be a T-precise specification whenever T is a theory that orders events on individual channels only.

It is important to note that the theory T we will develop has the following property. For any process P, there is a many-to-one correspondence between the T-models of P and the models of behaviour for P described above. This means that for our theory, the notion of a T-precise specification and the normal definition of a precise specification coincide.

3 Previous Attempts at Relative Completeness

In this section we show how relative completeness for networks of arbitrary processes (without channel hiding) can be achieved in a straightforward manner. However, the resulting proof system will not be effective. We then compare this approach to the techniques used by other researchers to obtain relative completeness.

Consider a process P with alphabet Σ. Let T be any theory of traces in some first order language $\mathcal{L}$. We define the following set of formulas in $\mathcal{L}$.

$A_\Sigma = \{\Theta | \Theta$ is a formula in $\mathcal{L}$ that is true in all standard T-models of processes with alphabet Σ $\}$

If Φ is a T-precise specification of process P, then any valid statement about P can be proved from $A_\Sigma \cup \{\Phi\}$ as follows. If Ψ is true for all standard models of P, then $\Phi \Rightarrow \Psi$ is in A_Σ and hence $A_\Sigma \cup \{\Phi\} \vdash \Psi$.

In this same context, the network composition rule can be formulated as

$$\frac{P_1 \; sat \; \Phi_1, P_2 \; sat \; \Phi_2}{P_1 || P_2 \; sat \; \{\Phi_1'\} \cup \{\Phi_2'\}}$$

where $P_1||P_2$ is the network obtained from combining P_1 and P_2. In this rule, if $\mathcal{L}$ only permits traces over individual channels, then $\Phi_i' \equiv \Phi_i$. If $\mathcal{L}$ permits interleaved traces, then Φ_i' is the projection of Φ_i onto the channel traces of P_i.

Again it is easy to see that if $\Phi_1\,(\Phi_2)$ is a precise specification of $P_1\,(P_2)$ then $A_\Sigma \cup \{\Phi_1\} \cup \{\Phi_2\}$ is a precise specification of $P_1||P_2$. A relatively complete proof system can then be built upon this by adding suitable axioms for renaming and consequence.

The drawback of this approach to relative completeness and network composition is that the proof system is not effective. That is, we do not have a recursive set of axioms and rules of deduction for our theory.

We now describe very briefly the approach taken by other researchers in this area. [ZRB85] consider a first order predicate language and give a relatively complete proof system for dynamic networks of processes where each process is specified in a CSP-like language. An essential component of their proof system is that all valid assertions are added as axioms. This is essentially equivalent to the use of the set A_Σ defined above. [HH83] consider a second order predicate assertion language and hence their system cannot achieve effectiveness. However, they are able to do channel hiding as they can quantify over traces. Both [NDGO86] and [WGS87] use predicate temporal logic over individual channel traces. The relative completeness achieved relies on the premise of a complete proof system for a predicate temporal logic that allows global quantification over the indices of the channel traces. Jonsson [Jon85] suggests the addition of a proof rule of the form

$$\frac{N\ sat\ S_1}{N\ sat\ S_2}$$

to be used whenever any network satisfying S_1 will also satisfy S_2. Such a proof system includes a proof rule corresponding to each element of A_Σ.

As mentioned earlier, the approach using A_Σ to achieve relative completeness does not lead to an effective proof system. Similarly, the past work described above contains an equivalent degree of ineffectiveness. We cannot hope to do much better without restricting the processes suitably.

The obvious question is how we can restrict both the processes and the language so as to obtain a complete and effective proof system. We will later argue that a language containing the full theory of arithmetic is too expressive for the processes that we will be considering. By choosing our assertion language carefully, we will be able to obtain a system for for reasoning about processes that is both complete and effective and which allows us to develop an algorithm for doing network composition with channel hiding.

4 Processes and Networks

We now define both the processes and networks we will be considering and also the terminology needed. Later we will discuss the close correspondence that exists between our processes and finite state automata (fsa's); and in anticipation of that we will call our processes *extended finite state automata* (*efsa's*) and use the terms process and efsa interchangeably.

An efsa is a non-deterministic, finite state, finite resource computing device for which the interactions with the environment are restricted to the communication of elements from a finite alphabet over a finite number of input and output channels. More specifically, each channel of an efsa E is uniquely named, has its own finite alphabet, and is used exclusively for input or output. The efsa E can, at any step, perform exactly one of the following two external actions: read from an input channel or write on an output channel, and based on that action change its state. An efsa cannot engage in two communications at the same time.

More formally, an efsa is a 6-tuple $(Q, Ch_{in}, Ch_{out}, \Sigma, \delta, q_o)$ where Q is a finite set of states, $Ch_{in}(Ch_{out})$ is a finite set of input (output) channel names, Σ is an alphabet consisting of a finite set of 2-tuples for which the first component is from $Ch_{in} \cup Ch_{out}$, q_0 is the start state, and δ is a partial function mapping $Q \times \Sigma$ to Q. The function δ has the restriction that if $\delta(s, < ch_i, y >)$ is defined and $ch_i \in Ch_{in}$, then for all $< ch_i, z >$ in Σ, $\delta(s, < ch_i, z >)$ is defined. For any channel ch_i, $\{y \mid \ < ch_i, y > \in \Sigma\}$ is the alphabet of channel ch_i.

The fsa's and efsa's are similar. The difference between them arises from the fact that an fsa accepts a set of strings over an alphabet, while an efsa engages in communication over input and output channels and hence can be viewed as generating a set of strings over an alphabet. This difference also explains why the transition function in a fsa is a total function, while for an efsa it is a partial function. Finally, the restriction placed on the function δ of an efsa is motivated by the fact that a value read on an input channel can be any value from the alphabet of that input channel, that is, δ is *input total.*

We will permit synchronous as well as a restricted form of asynchronous communication. The type of asynchronous communication that we permit is such that it can be transformed into an equivalent synchronous communication through a bounded size buffer. We term this type of communication to be *bounded asynchronous.* Since bounded asynchronous communication can be transformed into equivalent synchronous communication through the use of bounded buffers, we will restrict our discussion to synchronous communication.

A connection list CL is a set of elements of the form $< E_i.ch_1, E_j.ch_2 >$ where $1 \leq i, j \leq n$. This element signifies that the output channel ch_1 of efsa E_i is connected to the input channel ch_2 of efsa E_j. Communication over this channel is synchronous, that is, a value is transmitted over it only when E_i is ready to write out on ch_1 and E_j is ready to read on

ch_2 at the same time. We assume wlog that channel names are unique and that a channel name appears at most once in a connection list.

Efsa's $E_1, .., E_n$, $n \geq 1$ can be combined according to a connection list CL to yield an *intermediate network* $IntN_{CL} = E_1 \parallel .. \parallel E_n$ or a *network* $N_{CL} = [E_1 \parallel .. \parallel E_n]$. If $< E_i.ch_1, E_j.ch_2 >$ is in CL then it is called an *internal channel* in $IntN_{CL}$; any channel that does not appear in CL is an *external channel* of $IntN_{CL}$. In N_{CL} all the internal channels of $IntN_{CL}$ are *hidden*; communication across these internal channels is no longer visible when N_{CL} is viewed as a black box. The network N_{CL} can now be viewed as an efsa itself. The internal channels are hidden in order to enable us to give abstract specifications for networks. That is, if networks A and B are interchangeable in all contexts, then A and B should satisfy the same specification.

As for the processes used in section 2, the sequence of communication that an efsa E engages in during one computation is called a *trace* and is a sequence of elements from the alphabet of the efsa E. A trace of E gives one possible behaviour of E. The set of all possible behaviours of E is denoted by Tra_E. A trace of E gives a total ordering of communications over its channels, and to highlight that we can refer to our trace as being an interleaved trace.

Now, a word about why we restrict ourselves to networks of efsa's with bounded asynchronous communication. Our main reason is that with these restrictions we are able both to give a complete proof system with a recursive set of axioms for the set of processes and also to present an effective procedure for channel hiding in network composition.

Another reason for restricting ourselves to efsa's is that only finite state, finite resource computing devices can be physically realized. Finally, we do not allow arbitrary asynchronous communication since asynchronous communication can be used to simulate any Turing machine, as shown by Ex. 2 below.

Ex. 2. In this example we sketch how an efsa E with asynchronous communication can simulate any Turing machine T. Our efsa will have 2 input channels and 2 output channels connected together as in Figure 1.

Let the alphabet of T be $\sum$ and \$ and h be 2 symbols not in $\sum$. The alphabets of ch_1, ch_2, ch_3 and ch_4 is $\sum \cup \{\$\}$, $\sum \cup \{\$, h\}$, $\sum \cup \{\$, h\}$ and $\sum$ respectively. The symbols on the input tape provided to T are fed into E through channel ch_1 with \$ appended at the end. E works as follows.

1. E outputs symbol h on ch_2 to mark the current position of read head of T.

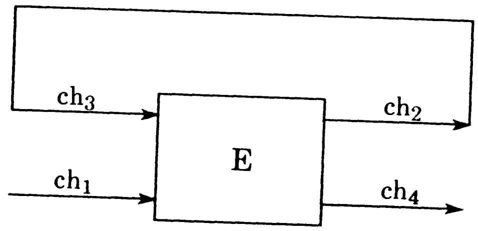

Figure 1: Efsa With Asynchronous Communication.

2. It repeatedly reads in one symbol from ch_1 and outputs it on ch_2 until the symbol \$ has also been processed.

3. For each step of T, E reads in symbols on ch_3 and outputs them onto ch_2 with a lag of 2 symbols until the special symbol h is encountered. On reading in h on ch_3, E changes its internal state, position of h on the 'tape' and the 'tape' just as T would. It then repeatedly reads on ch_3 and outputs it on ch_2 until \$ is processed.

4. When T halts, E simply 'writes out the contents of tape onto ch_4'. □

The essence of Ex. 2 is that we could use asynchronous communication to model an unbounded resource, in this case the input tape of a Turing machine. An incorrect conclusion of this example could be that to restrict ourselves to physically realizable machines we might allow asynchronous communication over certain channels as long as no cycle involving asynchronous communication is formed in the network. This conclusion is incorrect as shown by Ex. 3.

Ex. 3. Consider $L = \{a^ib^ic^i \mid i \geq 1\}$. L is not a context free language and hence cannot be accepted by any push down automata (PDA), even though a PDA uses infinite resources. (In fact L is accepted by a two way PDA, for more details see [HU79].) We now sketch how L can be accepted by the network of Figure 2 with asynchronous communication but no channel cycle. The alphabets of the channels ch_1, ch_2, ch_3, ch_4 and ch_5 is $\{a,b,c,d\}, \{a,y,n\}, \{b,y,n\}$, $\{c,y,n\}$ and $\{y,n\}$. The d on ch_1 serves as an endmarker for E_1. The communication on ch_2, ch_3 and ch_4 is asynchronous.

E_1 works as follows. Until a d is read on ch_1 it repeatedly reads in a symbol x on ch_1 and outputs it on ch_2 or ch_3 or ch_4 depending on whether x is a or b or c respectively. Once a d is read in, if the string read in on ch_1 is of the form $a^*b^*c^*d$ then a y is sent on ch_2, ch_3 and ch_4, otherwise an n is sent on ch_2, ch_3 and ch_4. E_1 then stops.

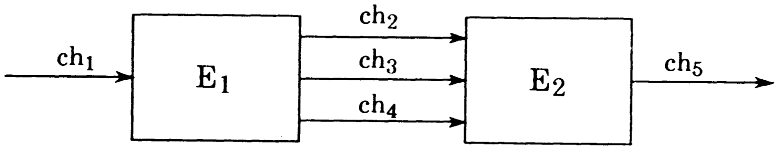

Figure 2: Network With Asynchronous Communication But No Loops.

E_2 works as follows. It proceeds in rounds, in each round it reads in one value each from channels ch_2, ch_3 and ch_4. If the 3 values read in are all y then it outputs a y on ch_5 and stops signifying acceptance of input on ch_1, otherwise if at least one of the 3 values read in is y or n then it sends out an n on ch_5 and stops, otherwise it proceeds to the next round. $\square$

5 Precise Specification of Efsa's

The language and theory, T, we develop for reasoning about efsa's refers to the interleaved trace. Furthermore, it is based on a connection between the efsa's and regular expressions.

Definition 3. Let E be an efsa and R a regular expression. We say that R provides a *T-precise specification* for E if for every trace tr, $tr \in Tra_E$ iff every initial segment of tr is a prefix of an instance of R.

For brevity, throughout the following we will use the terms *specification* and *precise specification* to mean *T-specification* and *T-precise specification.*

The following theorem shows that every efsa can be given a precise specification by means of a regular expression.

Theorem 1. Efsa's can be characterized as follows

(i) for every efsa E there is a prefix closed regular expression R such that $tr \in Tra_E$ iff tr is an initial prefix of an instance of R

(ii) for every regular expression R there is an efsa E for which R gives its specification.

Proof. (i) We will first show that for every efsa E there is a deterministic fsa E'' such that $tr \in Tra_E$ iff every finite initial segment of tr is accepted by E''. Given any efsa E over some alphabet A, transform it to an efsa E' by adding two states, *rej* and *accept* and transitions as follows. For any state s of E let

$$L_s = \{<ch_i, x> \mid \exists q : \delta(s, <ch_i, x>) = q\}$$

Then, for any state s of E and for every $<ch_i, x> \in \Sigma - L_s$, add a transition with label $<ch_i, x>$ from s to *rej* in E'. For each $<ch_i, x> \in \Sigma$, add transition with label $<ch_i, x>$ from *rej* to *rej*. Also, for each state s of E, add an ϵ transition from s to *accept* in E'. Now, efsa E' can be thought of as an ordinary fsa over Σ with start state corresponding to start state of E and a state *accept* as the final state. Each transition labeled $<ch_i, a>$ in E signifies the communication of a over ch_i while in E' it signifies the state transition on reading the input $<ch_i, a>$. The fsa E', in general, is a non-deterministic fsa, but it can be converted to a deterministic fsa E'' as follows: The states of E'' correspond to a set of states of E'; and at every point E'' keeps track of all the states E' could be in having read the same input as E'' (for more details see [HU79]). By the structure of E' and the above transformation, it follows that in E'' only the state corresponding to the set of states $\{rej\}$ of E' does not have an ϵ transition to the final state. Hence, $tr \in Tra_E$ is accepted by E'' follows trivially from the construction.

Now let R be the regular expression corresponding to E'' constructed above. We show that R is prefixed closed. Consider $tr \in R$. For any prefix tr' of tr, we must show that $tr' \in R$. Now, there exists tr'' constructed out of a finite number of symbols such that $tr' \hat{} tr'' = tr$. Since $tr \in R$, tr is a legal trace of E'' and is therefore accepted in E''. In the path traced out by tr in E'', every state on the path has an ϵ transition to the final state; hence every prefix of tr is also accepted and hence every prefix of tr is also in R.

It follows trivially from the construction that $tr \in R$ iff $tr \in Tra_E$ iff tr is accepted by E''.

(ii) Given a regular expression R over some alphabet of 2-tuples Σ, construct an efsa as follows. Let B be the non-deterministic fsa with no ϵ transitions corresponding to R constructed as in [HU79]. Let

$$Rej_B = \{x \mid x \text{ is a state of } B \wedge \text{ no path from } x \text{ leads to the final state of } B\}$$

Construct E from B by dropping all states in Rej_B, and delete all transitions to and from states in Rej_B (note that the final state of B is considered an ordinary state of E.) E can be viewed as an efsa where label $< ch_i, x > \in \Sigma$ corresponds to outputting x on output channel ch_i. The fact that R is a specification of E follows from the construction of E and the first part of this theorem. $\Box$

Note that in the above proof, in (ii) ch_i was taken to be an output channel. ch_i could be an input channel provided the following holds

$$\forall s, t(s\hat{\ }< ch_i, a >\hat{\ }t \in R \Rightarrow \forall b(< ch_i, b > \in A \Rightarrow \exists t_b(s\hat{\ }< ch_i, b >\hat{\ }t_b \in R)))$$

That is, if ch_i is an input channel then when a value is read on it then the value read can be any possible value from its alphabet.

The close relationship between efsa's and regular expressions is not surprising. Such a relationship also exists for other machines, such as the Moore and Mealy machines.

6 Choosing an Assertion Language with Adequate Expressive Power

The choice of an assertion language is a critical step in developing a proof system. On the one hand, the language should be powerful enough to give specifications for our processes. On the other hand, it should not be so powerful that completeness and effectiveness are unobtainable. In this paper we will develop an assertion language based on first order predicate logic over an interleaved channel trace. Our theory will contain some special unary relations, but will not contain the full theory of arithmetic. Our reasons for this choice are as follows.

Notice that even though there are complete and effective proof systems for propositional temporal logic [Kro87], any assertional language based on propositional temporal logic would not be suitable for our purposes since Wolper [Wol81] has shown that it is not expressive enough. For both predicate temporal logic and first order predicate logic with the full theory of arithmetic, there can be no complete and effective proof system for traces of arbitrary processes.

Our choice of predicate logic over predicate temporal logic is based in part on the fact that once the language includes quantification over elements of the trace, temporal formulas do not help in the statement of most properties of interest. In addition, a technical difference between predicate temporal logic and first order predicate logic concerns the models they admit. All the models for predicate temporal logic are standard [Kro87], while first order predicate logic has standard as well as non-standard models. This implies that there are properties that can be stated in predicate temporal logic but which can not be realized by any physical process. An example of such a property is any 'fairness' property for which some action is guaranteed to take place at some finite point in the future, but no bound is placed on how far in the future it may occur.

We now turn to the issue of the expressive power of the language. A first order predicate logic containing only the language of traces and the theory of arithmetic is expressive enough for our purposes, but it is not *suitable* as it is difficult to specify some very simple processes. Even predicate temporal logic suffers from this fate. Whether or not a language is suitable is a qualitative judgment and we therefore give in Ex. 4 an fsa for which the reader is urged to supply a specification.

Ex. 4. Consider fsa A with alphabet $\{a, b, c\}$ and the set of strings it accepts is $S = \{a^* \ (ba^*ba^*)^* \ c\}$, that is, an even number of b's occur before the endmarker c. How do we specify fsa A? We know that it (and in fact any regular expression) can be specified by arguing as follows : (i) we can construct a Turing machine that accepts precisely the set S (in fact, the power of Turing machine is not needed for this example), (ii) the set of strings accepted by a Turing machine is an n-ary recursively enumerable relation R, and (iii) any n-ary recursively enumerable relation R is equivalent to some (quantified) polynomial relation P. □

The argument in Ex. 4 is essentially Gödel's theorem; for more details on this see Rogers [Rog87]. For many processes, it is not too difficult to write out the required polynomial relation, however for fsa A it is not at all obvious what the polynomial relation would be.

Before we discuss how to overcome the problem highlighted by fsa A above, let us look at our assertion language and see whether we can weaken it. The language, as it stands, is expressive enough to specify Turing machines, while the processes that we are considering are much simpler; this implies that it should be possible to restrict the assertion language suitably. As usual, our goal in doing this is to aim for a simpler proof system for our processes. Since the assertion language has full arithmetic built in, it allows for specifications in which the indices of the trace are added or multiplied together. Our processes do not have this capability, so this is a logical place in which the assertion language should be restricted. Consequently, we will restrict our assertion language by throwing out most of arithmetic. That is, in our assertion language, the only arithmetic operations permitted on indices is the addition or subtraction by constants.

To make this restricted language suitable, we will add to it a set of unary relations that will allow us to express the concept of a regular expression. Note that such a language will be strictly less expressive than a language containing the full theory of arithmetic. However, it has been carefully designed so that it enables us to describe the key characteristics of any efsa.

7 The Final Assertion Language

For any process P we can construct a specification given by a regular expression R (by Theorem 1). We assume throughout the following that R has been given in a fully parenthesized form. This will allow us to construct the theory and the proof by induction on the complexity of R.

From R we define our formal language as follows. This language contains the constants and relations required to describe a trace of the input-output produced by the process. In addition, the language contains a finite set of unary relation symbols to be used in the specification of the process in terms of the subexpressions of R. To be precise, let L_R be the set containing the symbols as follows:

(i) variables $w,\ x,\ y,\ z,\ w_1,\ w_2, ..., x_1, x_2, ...$

(ii) the unary relation symbol N

(iii) the binary relation symbol $\leq$

(iv) the set of constants 0, 1, 2,...

(v) the unary function symbol Tr

(vi) a finite set of constants we will write as $<ch_1, a_{1,1}>, .., <ch_1, a_{1,k_1}>, ..,$ $<ch_M, a_{M,k_M}>, <ch_1, \alpha>$, where P has M channels and channel ch_i has an alphabet $\{a_{i,1}, .., a_{i,k_i}\}$ and α is not in the alphabet of any channel

(vii) a unary relation symbol Val

(viii) a finite set of unary relation symbols $R_1^B, .., R_G^B, R_1^E, .., R_G^E, R_1^{\epsilon l}, .., R_G^{\epsilon l}, R_1^{\epsilon r}, .., R_G^{\epsilon r}$ where $\{R_1, .., R_G\}$ is the set of subexpressions of the parenthesized R.

The unary relation symbols of the form $R_i^B(x)$ and $R_i^E(x)$ are used to label the beginning and ending of a non-ϵ instance of the subexpression R_i at x on the trace; the symbol $R_i^{\epsilon l}(x)$ ($R_i^{\epsilon r}(x)$) indicates an ϵ instance of R_i to the immediate left (right) of x.

The distinction between $R_i^{\epsilon r}(x)$ and $R_i^{\epsilon l}(x+1)$ is best illustrated through an example. Consider $R = (a(b^*))c$ with $R_1 = a, R_2 = b, R_3 = R_2^*, R_4 = R_1R_3, R_5 = c$ and $R = R_4R_5$. If the first two elements of a trace are ac and all the remaining elements are α values then we would like to label the trace as follows: R_1 begins and ends at the first element of the trace, R begins at the first element and ends at the second element of the trace, and an ϵ instance of R_3 occurs to the immediate right of the first element of the trace and not to the immediate left of the second element of the trace. With this marking, if the trace is chopped after the first element, we still have an occurrence of R_4 on the trace. Furthermore, the labeling has the effect of grouping the subexpression R_3 with R_1 rather than with R_5.

Given L_R, let $\mathcal{L}_R$ be the first order predicate calculus defined from L_R (with the usual axioms and rules of deduction). We now define, for the process P, a theory T_P in the language $\mathcal{L}_R$. A recursive set of axioms for T_P are as follows (the actual axioms appear in the appendix).

(i) a set of axioms stating that $\leq$ is a linear order on N and that under this ordering N is a model of the theory of the natural numbers

(ii) a set of axioms stating that for each natural number n, the constant $\bar{n}$ names the number n

(iii) an axiom stating that every element of Val is named by a constant from (vi) of language L_R

(iv) the axiom $\forall x((N(x) \vee Val(x)) \wedge \neg(N(x) \wedge Val(x)))$

(v) an axiom stating that Tr is a function from N to Val such that

$\forall x\ (\neg(Tr(x) = < ch_1, \alpha >)) \vee \exists x\ \forall y\ ((y \leq x \Rightarrow (\neg Tr(y) = < ch_1, \alpha >)) \wedge (y > x \Rightarrow Tr(y) = < ch_1, \alpha >))$ This axiom guarantees that in each model either there is a last element for which the trace contains a genuine input or output value (something other than $< ch_1, \alpha >$) or that all elements of the trace contain genuine input or output values.

(vi) a set of axioms guaranteeing that for each R_i and every $x \in N$, $Tr(x) = < ch_1, \alpha > \Rightarrow \neg(R_i^{\epsilon l}(x+1) \vee R_i^{\epsilon r}(x) \vee R_i^B(x) \vee R_i^E(x))$ (here $x+1$ is used as an abbreviation.)

(vii) a set of axioms guaranteeing that there is exactly one occurrence of R or its prefix in the 'trace' and that this occurrence starts at the left end of the 'trace'. These axioms also guarantee that the nesting of the labels for the R_i corresponds to the nesting of the subexpressions in R

(viii) For each formula $\theta(x)$ with one free variable and with all other variables restricted to lie below x, we add the axiom

$$\exists x(N(x) \wedge \theta(x)) \Rightarrow \exists x \forall y(N(x) \wedge N(y) \wedge \theta(x) \wedge (y < x \Rightarrow \neg\theta(y)))$$

where y is a variable not occurring in θ. This group of axioms guarantees that for any such formula, if it is satisfied in a model, there is a first point in the model at which it is satisfied.

The only axioms of T_P that are specific to process P are (vii), a finite set, and these axioms form a precise specification for P.

Theorem 2. Let P, T_P and $\mathcal{L}_R$ be as above. There is a many-to-one correspondence between the standard models of T_P and the possible traces of P. This correspondence is such that for every trace there are many (but at least one) standard models of T_P containing this trace.

Proof. By Theorem 1, there is a one-to-one correspondence between the set of possible traces of P and the set of prefixes of R. The standard models of T_P simply contain the prefixes of R with the subformulas labeled and trace values of $< ch_1, \alpha >$ for elements of N that are beyond the last element of the prefix of R. There is at least one such model for each distinct trace. Distinct standard models containing the same trace values can arise from a choice of labeling for a subformula of the form $R_i = R_j + R_k$. □

8 Elementary Equivalence and Ehrenfeucht Games

Before beginning our discussion of channel hiding and the proof of completeness, we need to define some terminology and present a brief description of a technique of proof referred to as Ehrenfeucht games.

The proofs that we give for the completeness of our theory are model theoretic. The goal of each proof is to show that given any model A of T_P and any formula in the language for P that holds in A, there is a standard model in which that formula holds.

Definition 4. Let $\mathcal{L}$ be a first order predicate calculus, and let A and B be models in the language. We say that A is *n-equivalent* to B (in symbols, $A \equiv_n B$) if every formula in $\mathcal{L}$ with no more than n quantifiers that is true in A is also true in B. We say that A is *elementarily equivalent* to B ($A \equiv B$) if they are n-equivalent for all n.

Proofs of the n-equivalence of models can be done in many ways. One very useful technique was developed by Ehrenfeucht. This technique is referred to as Ehrenfeucht games. We make use of these games in the proofs below and so will now describe them briefly. For full details see [Ehr61].

Given a language, $\mathcal{L}$, a finite subset S of constant and relation symbols in $\mathcal{L}$, an $n \geq 0$, and two models A and B, we define an n-game with respect to S, A, and B. The game has two players, P1 and P2. P1 and P2 alternate turns for a sequence of n rounds, with P1 always moving first. In a round, each player selects an element from one of the two models. If P1 selects an element from A (B), then P2 must select an element from B (A).

P2 wins the game if he always succeeds in maintaining a partial isomorphism between the sequence of elements chosen from A and the sequence chosen from B. This partial isomorphism must hold with respect to all of the relations and constants represented in the set S. P1 wins if he can prevent P2 from maintaining a partial isomorphism.

Theorem 3. (Ehrenfeucht [Ehr61]). Given structures A and B, $A \equiv_n B$ if and only if P2 can win the n game for all finite subsets S. $A \equiv B$ if and only if P2 can win the games for all n and all finite S.

9 Network Composition with Channel Hiding

Typically, network composition is done without channel hiding, with the specification for the network given essentially as the conjunction of the specifications of the individual processes as stated in section 3. A similar thing could be done in our language by adding projections onto individual channel traces. We will not do that here, however we will give an effective procedure for network composition with and without channel hiding.

For general processes it is not clear whether one can obtain an effective procedure for channel hiding. Ex. 5 below shows that even for efsa's, channel hiding can lead to an increase in the complexity of the specification.

Ex. 5. In this example we show that in a simple first order predicate language of traces

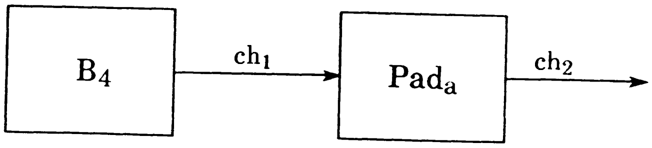

Figure 3: Channel Hiding Can Increase Complexity Of Specification.

without the theory of arithmetic, the specification of a network after channel hiding can require more quantifiers than is required by the specification of its individual components. Consider the network A of Figure 3 where the alphabet of ch_1 is $\{b\}$ and of ch_2 is $\{a, b\}$. Informally, B_4 just outputs $bbbb$ on ch_1. Pad_a reads in input on ch_1 and outputs it on ch_2 after padding it out with a's. More formally, these processes can be specified as follows. (For brevity, we do not mention the channel names in a trace if no confusion results.)

$$
\begin{array}{ll}
B_4 : Tra_{B_4} = \{\epsilon, b, bb, bbb, bbbb\} \\
Pad_a : \text{ all } tr \text{ in } Tra_{Pad_a} \text{ are such that} \\
\quad \forall x \geq 0[\ (tr(x) = <ch_2, a> \Rightarrow (x = 0 \ \vee \ (x > 0 \wedge tr(x-1) = <ch_2, a>) \\
\qquad\qquad \vee (x > 0 \wedge tr(x-1) = <ch_2, b>))) \\
\quad \wedge (tr(x) = <ch_1, b> \Rightarrow (x = 0 \vee (x > 0 \wedge tr(x-1) = <ch_2, a>) \\
\qquad\qquad \vee (x > 0 \wedge tr(x-1) = <ch_2, b>))) \\
\quad \wedge (tr(x) = <ch_2, b> \Rightarrow (tr(x-1) = <ch_1, b>))]
\end{array}
$$

So, the specification of B_4 requires no quantifiers and specification of Pad_a requires one quantifier. Let $\mathcal{L}'$ be $\mathcal{L}$ restricted to some finite set of constants. Consider traces tr_1 and tr_2 where

$$tr_1 = a^k b a^k b a^k b a^k b a^k$$

$$tr_2 = a^k b a^k b a^k b a^k b a^k b a^k$$

and k is twice as large as any constant in our language $\mathcal{L}'$. Now, any specification of the network of Figure 3 in $\mathcal{L}'$ (with internal channels hidden) should distinguish between tr_1 and tr_2 as $tr_1 \in Tra_A$ while tr_2 is not in Tra_A.

No 1-quantifier formula in $\mathcal{L}'$ can distinguish between tr_1 and tr_2 since player II can always win in a 1-move Ehrenfeucht game. In fact, it can be easily verified that at least a 3-move Ehrenfeucht game is required to distinguish between tr_1 and tr_2. This implies that a nesting of at least three quantifiers is needed in specifying A. We can construct a 3-move Ehrenfeucht game that distinguishes between tr_1 and tr_2 such that player I always picks from tr_1. Furthermore, there is a formula with three universal quantifiers that gives a precise specification of A in $\mathcal{L}$. (In the full theory of arithmetic, a formula having a sequence of universal quantifiers can be replaced by an appropriate equivalent formula having just one universal quantifier. However, that is not possible in this language.) □

In this example, we increased the number of quantifiers needed from 1 to 3. In fact, if we replace B_4 by a process generating 8 b's, then the increase in the number of quantifiers could be even larger; and so on.

What Ex. 5 shows is that in an assertion language based on predicate logic without arithmetic, the number of quantifiers required to specify a network with channel hiding can be larger than the number of quantifiers required in the specification of the individual processes. In our assertion language, this would translate into an increase in either the number of quantifiers or the complexity of the regular expression.

Definition 5. Consider an efsa E with alphabet $A = \{a_1, \ldots, a_n\}$. A normal form specification of E is a specification of E of the following form :

$$\begin{aligned}
\forall x[tr(x) = a_1 \Rightarrow\ & (\text{the existence (or non} - \text{existence) of a sequence } m_{1,1} \text{ of patterns in the past)}\\
& \vee(\text{the existence (or non} - \text{existence) of a sequence } m_{1,2} \text{ of patterns in the past)}\\
& \vdots\\
& \vee(\text{the existence (or non} - \text{existence) of a sequence } m_{1,j_1} \text{ of patterns in the past)}\\
\vdots\ & \\
\wedge tr(x) = a_n \Rightarrow\ & (\text{the existence (or non} - \text{existence) of a sequence } m_{n,1} \text{ of patterns in the past)}\\
& \vdots\\
& \vee(\text{the existence (or non} - \text{existence) of a sequence } m_{n,j_n} \text{ of patterns in the past)}]
\end{aligned}$$

The normal form specification allows for a more natural specification of some processes than is possible by regular expressions. However, any normal form specification for E can be translated into an equivalent regular expression specification for E. This is done by constructing a finite state machine equivalent to the normal form specification for E as follows. First of all, identify a set of states by considering all combinations of sequences of patterns (and their prefixes) used in the normal form specification. This set of states is finite. Now for all pairs of states, x and y, and for input/output a_i on ch_j, if a_i on ch_j is permitted by the normal form specification in state x and results in state y, then add the transition $\delta(x, < ch_j, a_i >, y)$ to the transition table.

We can expand the normal form specification by permitting the condition in definition 5 to depend on the **mod** function. Such an extension would still permit the translation specified above. However, to keep things simple, we do not do so here.

We can now present the effective procedure that allows for channel hiding in network composition.

Theorem 4. Let E_1, E_2 be efsa's where each is precisely specified by means of a regular expression or in normal form. For any syntactically valid connection list, CL, there is an effective procedure to obtain the precise specification of $N_{CL} = [E_1 \parallel E_2]$.

Proof. If the specification of E_1 and/or E_2 is given by the normal form specification then we can construct a state machine corresponding to that specification. Otherwise, again we can construct the state machine corresponding to the given regular expression.

Let SM_1 and SM_2 be the state machines corresponding to E_1 and E_2. These two state machine diagrams can be combined to give a new state machine diagram $SM_{1,2}$ according to CL. Note that the states of $SM_{1,2}$ are a subset of the cross product of the states of SM_1 and SM_2. Now, channel hiding can be done as follows: for each internal channel ch_i of $E_1 \parallel E_2$ under CL, replace each edge label of the form $< ch_i, x >$ by the label ϵ. The state machine diagram so constructed corresponds to some efsa $E_{1,2}$ and is characterized by some prefix closed regular expression $R_{1,2}$ by Theorem 1. Now, using $R_{1,2}$, a formal specification language and theory can be defined for reasoning about $E_{1,2}$. □

Corollary 1. Let $E_1, .., E_n$ be efsa's where each efsa is precisely specified either by a regular expression or in normal form. For any syntactically valid connection list CL, there is an effective procedure to obtain the precise specification of $IntN_{CL} = E_1 \parallel ... \parallel E_n$ and $N_{CL} = [E_1 \parallel ... \parallel E_n]$.

Proof. $E_1, \ldots, E_n$ can be composed hierarchically using Theorem 4. □

10 Completeness of the Proof System

We now establish the completeness of the proof system described above. That is, we prove that for any efsa with specification R, and any formula Φ in $\mathcal{L}_R$ that is true in all standard models of T_P, Φ is provable from T_P.

The proof of completeness is model-theoretic. It shows that for any process P and associated theory T_P, if A is a model of T_P and Φ is a formula that is satisfied in A, then there is a *standard* model of T_P also satisfying Φ. Theorems 5 and 6 below are used to establish this completeness result. Theorem 5 treats the case in which the trace in a model has a last element containing a genuine input or output value. Theorem 6 treats the case in which every element of the unbounded trace contains a genuine input or output value.

A key idea used in the proofs of Theorems 5 and 6 is contained in Lemma 1 below. It describes a technique for cutting and pasting together sections of models to obtain larger models while maintaining the property of n-equivalence. This technique was used by Rubin [Rub74].

Definition 6. Let A and B be linearly ordered sets with unary relations. Then $A\,\hat{}\,B$ is the linearly ordered set in which all elements of A are less than all elements of B, and all unary relations are defined as in A and B separately.

Lemma 1. [Rub74] Let A, B, C and D be linearly ordered sets with unary relations in some language. If $A \equiv_n B$ and $C \equiv_n D$, then $A\,\hat{}\,C \equiv_n B\,\hat{}\,D$.

Proof. The proof follows easily by means of Ehrenfeucht games . □

Note that although our language contains the unary function Tr, it can be viewed as simply assigning elements of N to disjoint unary relations. Hence Lemma 1 may be applied to the models of our theory.

Throughout the following, we assume that P is an efsa with specification given by R, and that T_P is defined as in section 7. For Theorems 5 and 6 below, $\mathcal{L}'_R$ is $\mathcal{L}_R$ restricted to some finite set of constants for elements of N. We also define the following notation that will be used throughout the proofs of Theorems 5 and 6. We use the notation $|B|$ to represent the set of elements of B. If D is any model of a linear ordering, and $z \in |D|$, let $D\,|_{\leq z}$ ($D\,|_{\geq z}$) be the substructure of D with elements $\{y \mid y \leq z\}$ ($\{y \mid y \geq z\}$).

Theorem 5. Let A be any model of T_P and $x \in A$ such that $Tr(y) \neq <ch_1, \alpha>$ for $y \leq x$ and $Tr(y) = <ch_1, \alpha>$ for $y > x$. For any $n \geq 0$ and any $\mathcal{L}'_R$, there is a standard model B of T_P, such that $A \equiv_n B$ with respect to $\mathcal{L}'_R$ and such that B has a largest trace element representing a genuine input-output value.

Proof. We proceed by induction on the complexity of the regular expression R specifying P.

base case: If $R = \epsilon$, then any model A of T_P has $Tr(x) = <ch_1, \alpha>$ for all x. In this case there is only one standard model of T_P. It is easy to show by means of Ehrenfeucht games that if A is a non-standard model, then it is n-equivalent to the unique standard model of T_P for all n and for all $\mathcal{L}'_R$.

If $R = <ch_i, a_{i,j}>$ for some i and j, then in any model A, $Tr(0) = <ch_i, a_{i,j}>$ and $Tr(x) = <ch_1, \alpha>$ for $x > 0$. Again there is a unique standard model, B, and any model A of T_P is n-equivalent to B for all n and all $\mathcal{L}'_R$.

inductive case: Let n and $\mathcal{L}'_R$ be given. We assume wlog that $n \geq 1$. Assume that the result holds for all $m \geq 0$ and all $\mathcal{L}'_{R'}$, where the expression R' has complexity less than R.

case 1. $R = R_i + R_j$ for some i and j. Let A be a model of T_P. By Theorem 1, there are processes P_i and P_j for which the specifications are given by R_i and R_j respectively. By the definition of R, A is a model of T_{P_i} or T_{P_j} when the relations R^B, R^E $R^{\epsilon l}$ and $R^{\epsilon r}$ are ignored. Assume, wlog, that $A \models T_{P_i}$. By induction, there is a standard model B of T_{P_i} that is n-equivalent to A, and has an element x such that $\forall y(y > x \Rightarrow Tr(y) = <ch_1, \alpha>)$. B can now be turned into a model of T_P by defining the relations $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E on it. We define them as follows:

(i) $\forall x(R^{\epsilon l}(x) \Leftrightarrow R_i^{\epsilon l}(x))$

(ii) $\forall x(R^{\epsilon r}(x) \Leftrightarrow R_i^{\epsilon r}(x))$

(iii) $\forall x(R^B(x) \Leftrightarrow R_i^B(x))$

(iv) $\forall x(R^E(x) \Leftrightarrow R_i^E(x))$

It is now easy to show that B with these added relations is n-equivalent to A in the language containing $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E.

case 2. $R = R_iR_j$ for some R_i and R_j. Let $A \models T_P$.

If $A \models R_i^{\epsilon l}(0) \vee \ \forall x(\neg R_j^B(x))$, then A can be viewed as a model of T_{P_i} or T_{P_j}. In this case we argue as in *case* 1 to get a standard B that is n-equivalent to A in the language including $R^{\epsilon l}, R^{\epsilon r}, R^B$, and R^E. Furthermore, B will again have a largest element x such that $Tr(x) \neq \, < ch_1, \alpha >$.

If this statement does not hold in A, then there is an $x_0 \in |A|$ such that $A \models R_i^E(x_0) \wedge R_j^B(x_0 + 1)$. Let A_{x_0} be the model A with the trace values for all $x > x_0$ changed to $< ch_1, \alpha >$ and no unary relation holding for $x > x_0$. By induction, there are standard structures $B \models T_{P_i}$ and $C \models T_{P_j}$ such that B is n-equivalent to A_{x_0} and C is n-equivalent to $A|_{\geq x_0+1}$ when the relations $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E are ignored. In addition, B and C both have largest elements for which the trace is not $< ch_1, \alpha >$. Note that in order to obtain this equivalence, the constants in $\mathcal{L}_R'$ need to be reassigned appropriately in the structure $A|_{\geq x_0+1}$.

Define a model D of T_P as follows. Let x_1 be the largest element of B such that $Tr(x_1) \neq \, < ch_1, \alpha >$. Let D be the structure $B|_{\leq x_1} \hat{\ } C$ with appropriate reassigning of the constants in N. Note that we we must have $R_i^B(0) \wedge R_i^E(x_1) \wedge R_j^B(x_1 + 1)$ true in D, where $x_1 + 1$ is the successor of x_1 in D. If we also have $C \models R_j^E(x_2)$ for some x_2, then we have also $R_j^E(x_3)$ holding in D, where x_3 is the copy of x_2 in D. In this case, x_2 (x_3) must be the largest element in C (D) for which the trace is not $< ch_1, \alpha >$.

Since $B \equiv_n A_{x_0}$ with respect to the language without $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E and $n \geq 1$, we have $B|_{\leq x_1} \equiv_n A_{x_0}|_{\leq x_0}$ with respect to this restricted language. It is easy to see that, if we enrich B to form the model B' by adding the relations corresponding to R and letting $B' \models R^B(0) \wedge R^E(x_1)$, we have $B' \equiv_n A_{x_0}$ in the larger language.

Similarly, if we let $A_C = A|_{\geq x_0+1}$ then we have $C \equiv_n A_C$ (where an appropriate renaming of the constants in N is done in A_C) when $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E are ignored. Since $n \geq 1$, if there is an $x_2 \in |C|$ such that $C \models R_j^E(x_2)$, then there must be an $x_4 \in |A_C|$ such that $A_C \models R_j^E(x_4)$. Furthermore, x_2 and x_4 must be the last elements in C and A_C, respectively, with trace values that are not $< ch_1, \alpha >$. Since $R = R_i R_j$, it must also be the case that $A_C \models R^E(x_4)$. If we form the model C' by enriching C with the addition of the relations corresponding to R and let $C' \models R^E(x_2)$, we have $C' \equiv_n A_C$ in the larger language.

Let D' be $B'|_{\leq x_1} \hat{\ } C'$. By Lemma 1, since $B'|_{\leq x_1} \equiv_n A|_{\leq x_0}$ and $C' \equiv_n A|_{\geq x_0+1}$, we conclude that $D' \equiv_n A$ in the language $\mathcal{L}_R'$.

case 3. $R = R_i^*$ for some R_i. In this case we again do an argument by cutting and pasting. In addition, we make use of the family of axioms in section 7 (viii).

Let A be a non-standard model of T_P. Let x_0 be the largest element in A such that $Tr(x_0) \neq\ <ch_1, \alpha>$. If x_0 is a standard point in N, the model B, consisting of A restricted to its standard part, is n-equivalent to A. So we assume x_0 is a non-standard point.

For $x \in |A|$, let A_x be the model A modified by letting $Tr(y) = \ <ch_1, \alpha>$ for all $y > x$, with none of the relations $R_j^{cl}, R_j^{cr}, R_j^B, R_j^E$ holding for $y > x$ for any R_j. We will now prove the following claim: for any $x \in |A|$ there is a standard B, such that $B \models T_P$ and $B \equiv_n A_x$.

Suppose the claim is not true. For any $x \in |A|$, let Ψ_x be the conjunction of all formulas of complexity $\leq n$ that are true in $A|_{\leq x}$. Let $\Phi^{\leq x}(y)$ be Ψ_x with all quantifiers restricted to $\leq y$, where y is a new variable name. For $x \in |A|$ let $< A, x >$ be the structure A with distinguished element x. Then, for each $x \in |A|$, $< A, x > \models \Phi^{\leq x}(x)$.

For each formula of the form $\Phi^{\leq x}(y)$, there is (by axiom (viii)) a smallest x_0 such that $< A, x_0 > \models \Phi^{\leq x}(x_0)$. Thus, since the set $\{\Phi^{\leq x}(y) \mid x \in |A|\}$ contains only a finite number of distinct elements, there must be a smallest element x_1 such that no standard B_{x_1} is n-equivalent to A_{x_1}.

Clearly x_1 must be a non-standard point. If there is no $y < x_1$, such that $A \models R_i^E(y)$ then A_{x_1} can be viewed as a model of T_{P_i} and we would have the desired B_{x_1} by induction. Thus, by axiom (viii), there must be a largest $y_1 < x_1$ such that $A \models R_i^E(y_1)$.

Since $y_1 < x_1$, there is a standard model B of T_P that is n-equivalent to A_{y_1}. Let y_2 be the element in B corresponding to y_1.

Now, with an appropriate renaming of the constants in N, we see that $C = A_{x_1}|_{>y_1}$ forms a model of $T_{P'}$, where $T_{P'}$ is the process associated with R_i. Hence, by induction, there is a standard model $D \equiv_n C$. By Lemma 1 we have $B|_{\leq y_2} \hat{} D \equiv_n A_{x_1}$ in $\mathcal{L}'_R$, and the claim is proved .

Having proved this claim, it is true for the largest value x for which $Tr(x) \neq\ <ch_1, \alpha>$. Hence for that value of x we obtain a standard model n-equivalent to A. $\square$

Theorem 6: Let P be a process and A a non-standard model of T_P with the property that for all $x \in |A|$, $Tr(x) \neq\ <ch_1, \alpha>$. Let $n \geq 0$ and $\mathcal{L}'_R$ be given. Then there is a standard model B of T_P such that $B \equiv_n A$ wrt $\mathcal{L}'_R$.

Proof. The proof of Theorem 6 uses the same kind of cutting and pasting techniques as those used in the proof of Theorem 5. Let A be as in the statement of the theorem. We again proceed by induction on the complexity of the regular expression R used to define P.

Wlog we assume that $n \geq 1$.

base case: $R = \epsilon$ or $R =< ch_i, a >$, for some element a in the alphabet of some channel ch_i. In this case, if $A \models T_P$, A cannot satisfy the hypothesis, so there is nothing to prove.

inductive case: Assume the theorem holds for all regular expressions of complexity less than R. We show that it holds for R.

case 1. $R = R_i + R_j$. As in the proof of Theorem 5, A can be seen as a model of T_{P_i} or T_{P_j} when the relations $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E are ignored. Thus by induction we get a B n-equivalent to A (ignoring those relations). Letting $B \models R^B(0)$ we obtain $B \equiv_n A$ in $\mathcal{L}'_R$.

case 2. $R = R_iR_j$. Again, if $A \models R_i^{\epsilon l}(0) \vee \forall x(\neg R_j^B(x))$ then the proof is trivial.

Assume that there is an x_0 such that $A \models R_i^E(x_0) \wedge R_j^B(x_0+1)$. By Theorem 5, there is a standard $B \models T_{P_i}$ and $x_1 \in |B|$ such that $B\,|_{\leq x_1} \equiv_n A\,|_{\leq x_0}$. Since $n \geq 1$, we must have $B \models R_i^E(x_1)$.

By induction, there is a standard $C \models T_{P_j}$ such that $C \equiv_n A\,|_{\geq x_0+1}$ with some reassigning of the constants. Again, we must have $C \models R_j^B(0)$.

By Lemma 1, we can conclude that $B\,\hat{}\,C \equiv_n A$ when $R^{\epsilon l}, R^{\epsilon r}, R^B$ and R^E are ignored. Letting $B \models R^B(0) \wedge \forall x(\neg R^E(x) \wedge \neg R^{\epsilon l}(x) \wedge \neg R^{\epsilon r}(x))$ we get $B\,\hat{}\,C \equiv_n A$ in $\mathcal{L}'_R$.

case 3. $R = R_j^*$. By the downward Löwenheim-Skolem Theorem, we may assume that A is countable. It is then possible to define a sequence $\{x_i\}_{i<\omega}$ such that (i) this sequence is strictly increasing, (ii) $R_j^B(x_i)$ holds for all i, (iii) the sequence is cofinal in A, and (iv) $x_0 = 0$.

For each $i \geq 0$, define B_i and C_i as follows. Let B_i be the model of T_P consisting of $A\,|_{\geq x_i}$ with the trace values for all $y \geq x_{i+1}$ changed to $< ch_1, \alpha >$. Note that to be a model of T_P, this requires a new assignment of the constants in $\mathcal{L}'_R$. B_i satisfies the conditions of Theorem 5, so we obtain a standard model $C_i \models T_P$ such that $B_i \equiv_n C_i$ with respect to $\mathcal{L}'_R$.

For $i < \omega$, let y_i (z_i) be the largest element in B_i (C_i) such that $Tr(y_i) \neq< ch_1, \alpha >$ ($Tr(z_i) \neq< ch_1, \alpha >$). Then we can conclude that $B_i\,|_{\leq y_i} \equiv_n C_i\,|_{\leq z_i}$.

By a simple extension to Lemma 1 it follows that $A = B_0\,\hat{}\,B_1\,\hat{}\cdots\hat{}\,B_k\,\hat{}\,B_{k+1}\,\hat{}\,\cdots \equiv_n C_0\,\hat{}\,C_1\,\hat{}\cdots\hat{}\,C_k\,\hat{}\,C_{k+1}\,\hat{}\,\cdots = C$. But since each C_i is finite, C is a standard model and the theorem is proved. □

Corollary 2. (Completeness). If Φ is any statement in the original language $\mathcal{L}_R$ that is true in all standard models of T_P, then Φ is provable from the axioms for T_P.

Proof. By the axioms in (v) of section 7, Theorems 5 and 6 cover all cases. □

Corollary 3. (Effectiveness). The set of all statements in the original language $\mathcal{L}_R$ that are true in all standard models of T_P is recursively enumerable.

Proof. By Corollary 2 and the fact that the set of axioms for T_P and the set of rules of deduction are both recursive sets. □

11 Conclusions

The goal of this paper is to provide a formal language together with a recursive set of axioms that is complete and sufficiently expressive and to develop an algorithm for doing network composition with channel hiding. With this goal in mind, we have restricted the processes and communications to ones for which effective results are possible. In addition, the language that we have defined in this paper has been carefully designed so that we are able to use it to provide a specification for any of these processes.

The close relationship between the processes and the language enables us to express most properties of interest about these processes, while eliminating those that are inherently undecidable. In addition, it enables us to combine processes into a network and *effectively* obtain a specification for the network that does not refer to the internal channels. The only results of this kind obtained until now have required the use of second order arithmetic.

The ability to specify all processes and do the channel hiding would be of only limited usefulness if we did not at the same time have a system that was complete with respect to the models representing actual computations. Fortunately we were able to obtain this completeness while still retaining a recursive set of axioms. In addition, by using interleaved traces we obtain a one-to-one correspondence between our standard T-models and the usual models of behaviour used.

A question that is still open concerns the decidability of this theory. In addition, the following questions would be interesting to explore. Can we effectively do channel hiding within the language? Can we enlarge our set of processes and still maintain both effectiveness and completeness? When doing channel hiding, is there an effective bound on the increase in complexity of the specification? And finally, is there a larger class of processes for which we can do effective channel hiding?

References

[**Ehr61**] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math. 49,* 129-141, 1961.

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

[**HU79**] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[**Jon85**] B. Jonsson. A model and proof system for asynchronous networks. In *Proc. of the 4th ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing.* 49-59, 1985.

[**Kro87**] F. Kröger. *Temporal Logic of Programs.* Springer-Verlag, Berlin, F.D.R., 1987.

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

[**Rog87**] H. Rogers. *Theory of Recursive Functions and Effective Computability.* MIT Press, Cambridge, MA, 1987.

[**Rub74**] M. Rubin. Theories of linear order. *Israel J. Math. 17,* 392-443, 1974.

[**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 Symp. on Princ. of Prog. Langs.,* 27-38, 1987.

[**Wol81**] P. Wolper. Temporal logic can be more expressive. *Information and Control 56,* 72-99, 1983.

[**ZRB85**] J. Zwiers, W.P. de Roever and P. v. E. Boas. Compositionality and concurrent networks: soundness and completeness of a proofsystem. In *Proc. of ICALP*, LNCS vol. 194, Springer-Verlag, 509-519, 1985.

Appendix: Axioms

Throughout the following we use the notation $S(x) = y$ as an abbreviation for the formula stating that "y is the successor of x". We will also use $\oplus$ as the symbol for exclusive or.

(i) A set of axioms stating that the theory of the natural numbers holds for all elements of N:

$$\forall x \forall y((N(x) \wedge N(y) \wedge x \neq y) \Leftrightarrow ((x > y) \vee (y > x)))$$

$$\forall x(N(x) \Rightarrow (\exists y((y > x) \wedge \forall z(z > x \Rightarrow z \geq y))))$$

$$\forall x(N(x) \Rightarrow (x \neq 0 \Leftrightarrow \exists y(S(y) = x)))$$

$$\forall x \forall y(x > y \Rightarrow \neg(y > x))$$

$$\forall x \forall y \forall z((x > y) \wedge (y > z) \Rightarrow (x > z))$$

(ii) For each natural number n, the constant $\bar{n}$ corresponds to n. For each such n we have:

$S^n(0) = \bar{n}$, where $S^n(0)$ is an abbreviation for the nth successor of 0

(iii) Let A be the set of all pairs of channel and channel output values plus the element $< ch_1, \alpha >$, then the following holds:

$$\forall x(Val(x) \Longleftrightarrow \bigvee_{c \in A} (x = c))$$

(iv) $\forall x(N(x) \oplus Val(x))$

(v) To establish that Tr is a function from N to Val we need the following two axioms:

$$\forall x(N(x) \Leftrightarrow \exists y(Val(y) \wedge Tr(x) = y))$$

$$\forall x \forall y \forall z((Tr(x) = y \wedge Tr(x) = z) \Rightarrow (y = z))$$

We also need the following axiom

$$\forall x\ (N(x) \Rightarrow Tr(x) \neq\ <ch_1, \alpha>)$$

$$\vee \exists x\ \forall y\ ((N(x) \wedge N(y) \wedge y \leq x \Rightarrow Tr(y) \neq\ <ch_1, \alpha>)$$

$$\wedge (N(x) \wedge N(y) \wedge y > x \Rightarrow Tr(y) =\ <ch_1, \alpha>))$$

In all the following axioms we assume that all variables x, y and z are elements in N and $x+1$ and $x-1$ etc. are used as a shorthand.

(vi) For all R_i in $\{R_1, \ldots, R_G\}$ add

$$\forall x(Tr(x) =< ch_1, \alpha >\Rightarrow \neg(R_i^B(x) \vee R_i^E(x) \vee R_i^{\epsilon l}(x+1) \vee R_i^{\epsilon r}(x)))$$

(vii) As a reminder, we are using $R_i^{\epsilon l}(x)$ ($R_i^{\epsilon r}(x)$) to indicate an ϵ instance of R_i to the immediate left (right) of x and that R is given in a fully parenthesized form. We assume the following structure of R and its constituents: if $R_i = R_j^*$ then $R_j \neq R_k^*$ for some R_k. This is simply a notational convenience. For each R_i in $\{R_1, \ldots, R_G\}$ we define as a shorthand the formulas

$$\begin{aligned}\Phi^{R_i}(x, y) \equiv\ & (x \leq y) \wedge R_i^B(x) \wedge R_i^E(y)\\ & \wedge \forall z(R_i^B(z) \Rightarrow z \leq x \vee z > y) \wedge \forall z(R_i^E(z) \Rightarrow z < x \vee z \geq y)\\ & \wedge \forall z(R_i^{\epsilon l}(z) \Rightarrow z < x \vee z > y+1) \wedge \forall z(R_i^{\epsilon r}(z) \Rightarrow z < x-1 \vee z > y)\end{aligned}$$

and

$$\begin{aligned}\Psi^{R_i}(x) \equiv\ & R_i^B(x) \wedge \forall z(R_i^B(z) \Rightarrow z \leq x) \wedge \forall z(R_i^E(z) \Rightarrow z < x)\\ & \wedge \forall z(R_i^{\epsilon l}(z) \Rightarrow z < x) \wedge \forall z(R_i^{\epsilon r}(z) \Rightarrow z < x-1)\end{aligned}$$

The first formula states that x and y are endpoints for a segment which our axioms will ensure is a non-ϵ instance of R_i. The second formula states that x is the beginning of such a segment that has no matching endpoint. The axioms below will ensure that this segment contains a prefix of an instance of R_i.

For regular expression R add the following axiom. (It guarantees that there is exactly one occurrence of R or its prefix.)

$$\begin{aligned}&(\exists y(\Phi^R(0, y) \wedge \forall z(z > y \Rightarrow Tr(z) =< ch_1, \alpha >)))\\ &\vee \Psi^R(0) \vee (R^{\epsilon l}(0) \wedge \forall y(Tr(y) =< ch_1, \alpha >))\end{aligned}$$

Add the following axioms for each R_i in $\{R_1, \ldots, R_G\} - \{R\}$.

$$\forall x(R_i^B(x) \Rightarrow (\exists y(y \geq x \wedge \Phi^{R_i}(x,y)) \vee \Psi^{R_i}(x)))$$
$$\forall y(R_i^E(y) \Rightarrow \exists x(x \leq y \wedge \Phi^{R_i}(x,y)))$$

For each R_i in $\{R_1, \ldots, R_G\}$, add axioms as follows:

If $R_i = < ch_j, a >$ for some $< ch_j, a >$ in Val we add

$$\forall x(R_i^B(x) \Rightarrow Tr(x) = < ch_j, a >)$$
$$\forall x(R_i^B(x) \Leftrightarrow R_i^E(x))$$
$$\forall x(\neg R_i^{\epsilon l}(x) \wedge \neg R_i^{\epsilon r}(x))$$

If $R_i = (R_j + R_k)$ for some j and k, we add the following

$$\forall x \forall y(\Phi^{R_i}(x,y) \Leftrightarrow (\Phi^{R_j}(x,y) \oplus \Phi^{R_k}(x,y)))$$
$$\forall x(\Psi^{R_i}(x) \Leftrightarrow (\Psi^{R_j}(x) \oplus \Psi^{R_k}(x)))$$
$$\forall x(R_i^{\epsilon l}(x) \Leftrightarrow (R_j^{\epsilon l}(x) \oplus R_k^{\epsilon l}(x)))$$
$$\forall x(R_i^{\epsilon r}(x) \Leftrightarrow (R_j^{\epsilon r}(x) \oplus R_k^{\epsilon r}(x)))$$

If $R_i = R_j R_k$ for some j and k, we add

$$\begin{aligned}
\forall x \forall y(\Phi^{R_i}(x,y) \Leftrightarrow (&\exists z(x \leq z < y \wedge \Phi^{R_j}(x,z) \wedge \Phi^{R_k}(z+1,y)) \\
&\vee (R_j^{\epsilon l}(x) \wedge \Phi^{R_k}(x,y)) \\
&\vee (\Phi^{R_j}(x,y) \wedge R_k^{\epsilon r}(y))))
\end{aligned}$$
$$\begin{aligned}
\forall x(\Psi^{R_i}(x) \Leftrightarrow (&\exists z(x \leq z \ \wedge \ \Phi^{R_j}(x,z) \wedge \Psi^{R_k}(z+1)) \\
&\vee (\exists y(y \geq x \wedge \Phi^{R_j}(x,y) \wedge \neg R_k^{\epsilon r}(y) \wedge \neg R_k^{\epsilon l}(y+1) \\
&\quad \wedge Tr(y+1) = < ch_1, \alpha >)) \\
&\vee (R_j^{\epsilon l}(x) \wedge \Psi^{R_k}(x)) \vee \Psi^{R_j}(x)))
\end{aligned}$$
$$\forall x(R_i^{\epsilon l}(x) \Leftrightarrow R_j^{\epsilon l}(x) \wedge R_k^{\epsilon l}(x))$$
$$\forall x(R_i^{\epsilon r}(x) \Leftrightarrow R_j^{\epsilon r}(x) \wedge R_k^{\epsilon r}(x))$$

If $R_i = R_j^*$, we add

$$\begin{aligned}
\forall x \forall y(\Phi^{R_i}(x,y) \Leftrightarrow\ & R_j^B(x) \wedge R_j^E(y) \wedge (x > 0 \Rightarrow \neg R_j^E(x-1)) \wedge \neg R_j^B(y+1) \\
&\wedge \forall z(x \leq z < y \wedge (R_j^E(z) \Leftrightarrow R_j^B(z+1))))
\end{aligned}$$

$$\forall x(\Psi^{R_i}(x) \Leftrightarrow R_j^B(x) \wedge (x > 0 \Rightarrow \neg R_j^E(x-1))$$
$$\wedge \forall z(x \leq z \wedge (R_j^E(z) \Rightarrow R_j^B(z+1) \vee Tr(z+1) =< ch_1, \alpha >)$$
$$\wedge (R_j^B(z+1) \Rightarrow R_j^E(z))))$$

(viii) For each formula $\theta(x)$ with one free variable and with all other variables restricted to being $\leq x$, we add the axiom

$$\exists x \theta(x) \Rightarrow \exists x \forall y(\theta(x) \wedge (y < x \Rightarrow \neg\theta(y)))$$

where y is a variable that does not occur in θ.