Equational Reasoning About Nondeterministic Processes*

Jayadev Misra†
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712
(512) 471-9547
misra@cs.utexas.edu

Abstract

A deterministic message-communicating process can be characterized by a "continuous" function f which describes the relationship between the inputs and the outputs of the process. The operational behavior of a network of deterministic processes can be deduced from the least fixpoint of a function g, where g is obtained from the functions that characterize the component processes of the network. We show in this paper that a nondeterministic process can be characterized by a "description" consisting of a pair of functions. The behavior of a network consisting of such processes can be obtained from the "smooth" solutions of the descriptions characterizing its component processes. The notion of smooth solution is a generalization of least fixpoint. Descriptions enjoy the crucial property that a variable may be replaced by its definition.

1 Introduction

In a pioneering paper, Kahn [1974] showed that a network of deterministic message-communicating processes may be represented by a set of equations, and the appropriate solutions of these equations completely characterize the message sequences sent during a computation of the network. For a deterministic process with input channel b and output channel c, an equation of the form $c = f(b)$ denotes that if a sequence of inputs, say x, is supplied to the process then a sequence of outputs, $f(x)$, will be sent by the process. The inputs may be supplied asynchronously and the outputs will be sent after an arbitrary, but finite, delay following the receipt of the input. Similar equations may be written for processes with multiple inputs and outputs. A network consisting of a finite number of processes may be described by the system of equations corresponding to its component processes. If every function employed in these equations is continuous (see Scott [1970]), then the set of equations has a (unique) least fixpoint which gives the message sequences sent along every channel of the network during a computation.

Several suggestions—Plotkin [1976], Park [1982], Staples and Nguyen [1985], Broy [1987], Panangaden and Shanbhogue [1987], Kok [1987], Rabinovich and Trakhtenbrot [1989]—have been made to extend Kahn's theory to networks that consist of nondeterministic processes (for a nondeterministic process, its outputs cannot be expressed as functions of its inputs). In this paper we propose a very simple mechanism for descriptions of nondeterministic processes and networks composed of them. The essential idea is to use equations in their full generality—a process with input channel b and

*A preliminary version of this paper appeared in *Proc. Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 1989.*

†This material is based in part upon work supported by the Texas Advanced Research Program under Grant No. 003658–065 and by the Office of Naval Research (N00014–90–J–1640).

output channel c is defined by an equation of the form $f(b,c) = g(b,c)$, where f, g are continuous functions; this makes it possible to express nonfunctional relationship between b, c. (We will later show that the left and the right sides of such an equation must be distinguished and hence we will employ a symbol other than the commutative $=$ to separate the left and the right sides.) Analogous to the notion of least fixpoint we define *smooth solutions* of such equations over a complete partial order. Smooth solutions capture the causality constraints between the input and the output of a process. Unlike a unique least fixpoint, there may be an arbitrary number (even zero) of smooth solutions. We show that every smooth solution corresponds to a computation and vice versa. Furthermore, for equations of the form $x = f(x)$, the unique smooth solution is the least fixpoint of f. Our definitional mechanism enjoys the property of referential transparency, i.e., terms can be replaced by their definitions in any other term.

Recently, Abramsky [1989] has generalized the methods of this paper for abstract nondeterministic processes; his work shows how these methods can be extended to domains other than linear traces. In private correspondence, we learned that Pratt has designed a theory based on "acyclic fixpoints" which generalizes Kahn's notion of least fixpoint and avoids the Brock-Ackermann [1981] anomaly; Pratt's method applies to a wide class of systems including continuous ones.

This paper is organized as follows. In Section 2, we give an informal explanation of our approach by using a few examples. We illustrate the need for smooth solutions through an example in which not all solutions correspond to computations. We also sketch the essential idea behind the Brock-Ackermann anomaly and show how it is resolved within our theory. Section 3 contains the basic concepts of our theory; also included are relevant concepts from the theory of complete partial orders. We illustrate these concepts with examples of some typical nondeterministic processes in Section 4. The composition theorem—that the descriptions of component processes of a network together yield a description of the network—is in Section 5. In Section 6, we give an alternative characterization of the least fixpoint of a continuous function in terms of smooth solutions; this constitutes a proof of Kahn's result for deterministic networks. In Section 7 we prove the soundness of substitutions of variables by their definitions. Section 8 contains a number of issues related to the development of this theory.

2 Informal Explanation

2.1 A Small Example Illustrating Kahn's Theory

Consider a deterministic process that has an input channel b and an output channel c; it copies every integer from its input to its output. The Kahn-style definition of such a process is $c = b$. Here a variable represents the sequence of all data *sent* along the correspondingly named channel. Thus, one way to interpret the equation $c = b$ in operational terms is that the process has nothing more to do (i.e., will engage in no further communication, input or output) iff the equation holds. The equation may be falsified by more data being sent to the process, in which case the process will (eventually) send more output. Every deterministic process can be represented by a set of such equations, one equation for each output channel of the process.

Now consider a network consisting of two processes, each of which copies its input to its output; a schematic diagram of the network is shown in Figure 1.
The equations describing the network are

$$c = b \ , \ b = c$$

These equations have many solutions over sequences of integers; b and c could both be $\bar{3}$, ($\bar{3}$ is the sequence consisting of element 3), for instance. However the unique least fixpoint of these equations (over an appropriate cpo) is b, c are the empty sequences. This is in fact the behavior we would expect from this network.

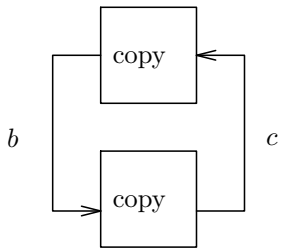

Figure 1: A network with two copy processes

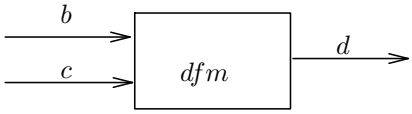

Figure 2: Discriminated fair merge; b is a sequence of even integers, c is a sequence of odd integers

If, however, the second process in Figure 1 first sends a "0" along b and then copies every input to its output, the following equation then describes the process (";" is the concatenation operator).

$$b = 0; c$$

The least solution to

$$c = b \ , \ b = 0; c$$

is $\quad b = 0^\omega \ , \ c = 0^\omega,$

where 0^ω is the infinite sequence of 0's.

This solution describes the "limiting behavior" of the network; every finite computation is some prefix of this solution. Furthermore, it shows that the network computation never terminates.

2.2 Describing a Nondeterministic Process

Consider a nondeterministic process, dfm (for $d$iscriminated $f$air $m$erge), that is shown schematically in Figure 2. Its input channel b carries only even integers and channel c carries only odd integers. The process dfm "merges the inputs fairly" and sends them along d. The precise property of fair merge is this: Every item in d is a unique item from b or c and every finite prefix of b (and of c) is a subsequence of some finite prefix of d.

Of particular interest is the behavior of dfm when both b, c are infinite. In that case output items have to be drawn infinitely often from both b and c. Since we are usually concerned with finite computations, such generality may seem frivolous; however the (limiting) infinite behavior serves to define a whole class of finite behaviors and hence simplifies the reasoning process.

We define dfm by

$$\text{even}(d) = b \ , \ \text{odd}(d) = c$$

where even/odd maps a sequence of integers to its subsequence consisting of even/odd integers.

Observe that all possible values of b, c, d arising in a computation (finite or infinite) of dfm satisfy these equations (the equations, as before, hold at the point where the process has nothing further to do) and conversely every solution of these equations in b, c, d specifies a possible computation of dfm. Also, note that nondeterminism and fairness, both present in dfm, have been handled within this definition. (A more general fair merge process is treated in Sec. 4.10.)

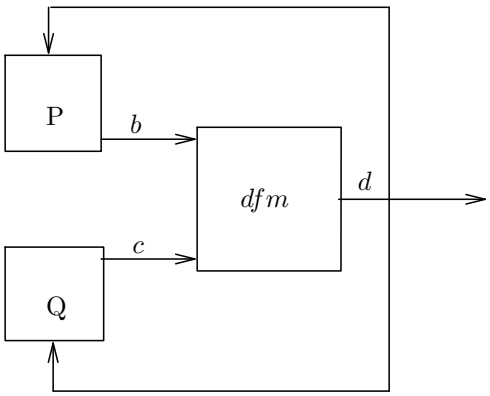

Figure 3: A network with three processes

In the theory of deterministic processes there is an asymmetry between inputs and outputs of a process—outputs are defined as functions of the inputs. We have eliminated this asymmetry; for instance, for dfm the inputs are defined as functions of outputs. Yet, we do need an asymmetry, between what is being defined (the left side) and its definition (the right side). We will introduce this asymmetry by distinguishing the two sides.

Note: It is possible to combine the two equations given above into one:

$$(\text{even}(d),\ \text{odd}(d)) = (b, c)$$

where each side of the equation is a pair. This strategy, for converting multiple equations into a single equation, is used throughout this paper; also, see the note in Section 4.

2.3 Description of a Network

Consider the network of processes shown schematically in Figure 3. The process dfm is the one considered in Section 2.2. Process P behaves as follows: it outputs a 0, then repeatedly receives a number, say n, and outputs $2 \times n$. Process Q behaves as follows: it repeatedly receives a number, say m, and outputs $2 \times m + 1$. (For receiving a number a process waits as long as no number is available on its input channel.)

The description of the network is obtained from descriptions of its component processes. In the following $2 \times d$ is the sequence in which every element is double the corresponding element of d and $2 \times d + 1$ is the sequence obtained by doubling each element of d and adding 1 to it.

$b = 0;\ 2 \times d$	{description of P}
$c = 2 \times d + 1$	{description of Q}
$\text{even}(d) = b\ ,\ \text{odd}(d) = c$	{description of dfm}

Eliminating b, c from these equations, we have a description of the output of the network:

$\text{even}(d) = 0;\ 2 \times d$	(1)
$\text{odd}(d) = 2 \times d + 1$	(2)

There are many solutions of equations (1,2) in integer sequences d. We consider a few of these. Let B_i, $i \geq 0$, denote the sequence of integers from 0 through $2^i - 1$. Let x be the concatenations of all B_i, $i \geq 0$, in increasing order of i's. Thus,

$$x = \overline{\underbrace{0}_{B_0}\ \underbrace{0\,1}_{B_1}\ \underbrace{0\,1\,2\,3}_{B_2}\ \underbrace{0\,1\,2\,3\,4\,5\,6\,7}_{B_3}\ \cdots}$$

It can be shown by direct calculation that x is a solution of equations (1,2). (Observe that $\text{even}(B_{i+1}) = 2 \times B_i$ and $\text{odd}(B_{i+1}) = 2 \times B_i + 1$.) Now consider another sequence, y, resulting from the concatenations of $\text{rev}(B_i)$, $i \geq 0$, where $\text{rev}(B_i)$ is the reverse of the sequence B_i; sequence y is also a solution of equations (1,2).

The solution sequences x, y correspond to different computations of the network in Figure 3. Sequence x corresponds to the computation in which dfm first receives 0 from channel b, sends 0 along d and then executes the following loop forever: receive from b; output to d; receive from c; output to d. Sequence y corresponds to the computation in which the above loop is replaced by: receive from c; output to d; receive from b; output to d.

Next consider a sequence z resulting from the concatenations of C_i, $i \geq 0$, where

$$C_0 = \overline{-1} \qquad C_1 = \overline{0\ -2}$$

and, for all $i \geq 1$, C_{i+1} is the sequence obtained by replacing each element m of C_i by two elements: $2m$, $2m + 1$. It can be shown that z is also a solution of equations (1,2). But z corresponds to no computation of the network, because -1 can never be output by the network.

As the above example shows, not all solutions correspond to computations. Therefore, we impose an additional constraint, *smoothness*, on the solutions. We will show that *smooth solutions* (i.e., solutions that also satisfy the smoothness constraint) correspond to computations and vice versa. Intuitively, the smoothness condition formalizes the operational behavior of a machine that an output by a process can depend only on the inputs received *prior to* sending the output, i.e., no output can be caused by (i.e., depend upon) itself as input. Sequence z violates this requirement because process dfm must first receive -1 as input before it can output -1.

For the example, an integer sequence s is a smooth solution iff

- it is a solution, i.e., $\text{even}(s) = 0; 2 \times s,\ \text{odd}(s) = 2 \times s + 1$ and,
- for all finite prefixes u, v of s where length of $v = 1+$ length of u,
  $\text{even}(v) \sqsubseteq 0; 2 \times u,\ \text{odd}(v) \sqsubseteq 2 \times u + 1$ ($\sqsubseteq$ is prefix ordering)

The first condition is met by sequences x, y, z. The latter condition is met by sequences x, y whereas in z, setting u to the empty sequence and v to $\overline{-1}$, the condition

$$\text{odd}(v)\ \sqsubseteq\ 2 \times u + 1$$

is violated.

It is worthwhile to note the advantages of an equational style representation in reasoning about process behavior. For instance from equations (1,2) we can deduce that every natural number n appears in the network's output eventually; the proof is by induction on n. Furthermore, it is easily seen that appearance of $2 \times n$ in the output is preceded by n in the output. These properties are known as progress and safety properties, respectively.

2.4 The Brock-Ackermann Anomaly

We sketch the essence of the Brock-Ackermann anomaly [1981]. (This anomaly was anticipated in Keller [1978] which also contains a "fix" using causality information.) Consider the network of Figure 4. Process A receives only odd numbers from b; it performs a fair merge of the input with (the internally stored) sequence $\overline{0\ 2}$ and outputs the result along c. Process B outputs $n + 1$ where n is the first number it receives; however the output is produced only after receiving two input

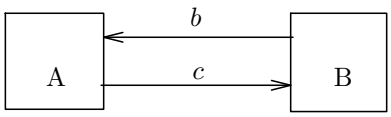

Figure 4: Brock-Ackermann anomaly

items. Thus process B implements a function f given by (where n, m denote integers, x is an integer sequence, and ϵ is the empty sequence),

$$f(\epsilon) = \epsilon \ , \ f(n) = \epsilon \ , \ f(n; m; x) = n + 1$$

The description of the network is

$$\begin{array}{ll} \text{even}(c) = \overline{0\ 2}\ ,\ \text{odd}(c) = b & \{\text{description of A}\} \\ b = f(c) & \{\text{description of B}\} \end{array}$$

Eliminating b,

$$\text{even}(c) = \overline{0\ 2}\ ,\ \text{odd}(c) = f(c)$$

It is not hard to see that there are exactly two solutions to these equations,

$$c = \overline{0\ 1\ 2} \qquad \text{and} \qquad c = \overline{0\ 2\ 1}\ .$$

Brock-Ackermann anomaly arises because $c = \overline{0\ 1\ 2}$ does not correspond to any computation—Process A has to output both 0 and 2 before Process B can produce the output 1. In our theory, $c = \overline{0\ 1\ 2}$ is *not* a smooth solution {because it is not the case that $\text{odd}(\overline{0\ 1}) \ \sqsubseteq \ f(0)$}, whereas $c = \overline{0\ 2\ 1}$ is.

2.5 Notations

Proof Format

We adopt a proof format suggested by Feijen and used extensively in Dijkstra and Scholten [1989] and van Gasteren [1988]. In this format, successive proof steps are separated by justifications enclosed within "{" and "}". Most, though not all, proofs in this paper are written in this way.

Quantification

A (universally or existentially) quantified predicate is written in the following sequence: (1) $\forall$ or $\exists$, (2) names of quantified variables followed by ":", (3) range of quantification followed by "::", (4) the body of the predicate. This is identical to the notation used in Chandy and Misra [1988] except that enclosing brackets "$\langle, \rangle$" are not used. Often the range of quantification (and its preceding ":") are omitted when this is clear from the context.

3 Basic Concepts

We give a few results from the theory of complete partial orders; these results may be found in standard books on the subject; see, for instance, Loeckx and Sieber [1984].

Let $(D, \sqsubseteq)$ be a partially ordered set. An element z of D is an *upper bound* for a nonempty set S if for every x in S, $x \sqsubseteq z$. An upper bound z is a *least upper bound* of S, written as $lub(S)$, if for any upper bound y of S, $z \sqsubseteq y$. The least upper bound of a set, if it exists, is unique. A nonempty set S is a *chain* if for every pair of elements x, y in S, either $x \sqsubseteq y$ or $y \sqsubseteq x$. $(D, \sqsubseteq)$ is a *cpo* (*c*omplete

*p*artial *o*rder) if (1) D has a bottom element, $\perp$, such that for all x in D, $\perp \sqsubseteq x$, and (2) every chain in D has a least upper bound in D.

Notation: We use $\perp$ and $\sqsubseteq$ for the bottom element and the partial ordering relation, respectively, for every cpo; context will tell. In the domain of sequences $\perp$ is the empty sequence; we use $\perp$ and ϵ interchangeably.

Lemma 1: (See Loeckx and Sieber [1984], Lemma 4.11) Let S, T be chains in a cpo. Suppose that for every x in S there is a y in T where $x \sqsubseteq y$. Then $lub(S) \sqsubseteq lub(T)$.

A function from a cpo to a cpo is *continuous* if it preserves the order of elements and the *lub*'s of chains, i.e., for a continuous function f, and chain S,

$$\begin{array}{ll} x \sqsubseteq y & \Rightarrow\ f(x) \sqsubseteq f(y) \qquad \{\text{monotonicity}\} \quad \text{and,} \\ f(lub(S)) = & lub(f(S)), \end{array}$$

where $f(S)$ is the set obtained by applying f to each element of S; using monotonicity, $f(S)$ can be shown to be a chain.

3.1 Trace, Process, Network

3.1.1 Informal Explanation

A communication history of a process can be described by listing the sequence of communications—each communication names a channel and the data item transmitted along it—in a computation. For the dfm process of Figure 2, some possible communication histories are ($\perp$ is the empty sequence)

$$\epsilon \qquad \overline{(b,0)} \qquad \overline{(b,0)(d,0)} \qquad \overline{(b,0)(c,1)(c,3)(d,0)(d,1)(b,2)}\ .$$

(Note that a pair (c, m) is included in a history if m is *sent* along c; receipt of a data item is not shown in a history).

Traditionally such a communication history is called a *trace* of the process. We will, however, use *trace* in a more restricted sense in this paper. Some of the communication histories of a process are guaranteed to be extended by outputs of the process; $\overline{(b,0)}$ is such a history for dfm. We call such histories *nonquiescent traces* and the remaining communication histories of the process *quiescent traces*, or simply *traces* of the process (Chandy and Misra [1984], Keller and Panangaden [1986], Jonsson [1985, 1987, 1989]). A process is "quiescent" after engaging in the communications of a quiescent trace. The quiescence of a process is affected by either (1) a message being sent to it, which causes the process to produce further output, or (2) the process deciding, nondeterministically, to produce further output. A process may never be quiescent—for instance, if it always has something to output—and then a quiescent trace is an infinite communication history. A few small examples illustrate this concept, below.

Example:

1. Each of the following communication histories is a quiescent trace for the dfm process (Figure 2).

   $$\epsilon \qquad \overline{(b,0)(d,0)} \qquad \overline{(b,0)(c,1)(c,3)(d,1)(d,3)(d,0)}$$

   Neither of the following is a quiescent trace, though each is a communication history for dfm.

   $$\overline{(b,0)} \qquad \overline{(b,0)(d,0)(c,1)}$$

   The infinite sequence consisting of $(b,0)$ followed $(d,0)$, indefinitely, is also a quiescent trace.

2. A process that halts or, nondeterministically, outputs a "0" along channel b and then halts, has the following two quiescent traces.

$$\epsilon \qquad \overline{(b,0)}$$

3. For a process that outputs all natural numbers consecutively along channel b, the only quiescent trace is the infinite sequence whose i^{th} element, $i \geq 0$, is (b, i).

By taking the prefixes of all traces of a process we can derive all possible communication sequences in which the process may engage. That is why we define a process by its traces. The traces of a network of processes are defined by the component process traces: t is a network trace if and only if for all component processes i, t_i is a trace of i, where t_i is obtained from t by retaining only those pairs (c, m) where c is incident on i.

3.1.2 Definitions

We are given a set called *channels* (each element of *channels* is called a channel). Associated with each channel is an alphabet called its *messages* (each element of *messages* is a message).

Remark: The set of channel names is *channels*. The *messages* associated with a channel is the set of data items that may be sent over that channel.

A *trace* is any sequence whose elements are pairs of the form (c, m) where c is a channel and m is a message associated with c.

A *process* is given by (1) a subset of *channels*, called its *incident channels*, and (2) a set of traces where for every element (c, m) in a trace, c is an incident channel of the process.

The *projection* of trace t on a subset of channels L, denoted by t_L, is the subsequence of t consisting of pairs (c, m) where c is in L.

A *network* consists of a finite number of processes, called its *component* processes. A network may be regarded as a process by defining its incident channels and traces, as follows. The set of incident channels of a network is the union of the incident channels of the component processes. A trace t is a network trace iff t_i is a trace of process i, for every component process i.

For a trace t, u *pre* v in t denotes that u, v are finite prefixes of t and the length of u is 1 less than the length of v. Clearly, for any finite nonempty prefix v of t, there is a unique u such that u *pre* v in t.

Note: The traces that define a process are its "maximal" or quiescent traces.

Notation: We use t_i, for a trace t and process i, to denote the projection of t on the incident channels of process i. It should be clear from the context whether the subscript is a process identifier or a subset of channels.

3.1.3 Facts About Traces and Projections

We need a small number of facts about traces and their projections. Proofs are standard.

- F1. The set of traces is a cpo under prefix ordering.
- F2. For a trace t, the set of its finite prefixes is a chain; *lub* of this chain is t.
- F3. Projection (on any subset of *channels*) is a continuous function from traces to traces.
- F4. For any trace t and process i,

$$u \textit{ pre } v \text{ in } t \quad \Rightarrow \quad (u_i = v_i) \ \vee \ u_i \textit{ pre } v_i \text{ in } t_i$$

- F5. For any trace t and process i,

$$\begin{array}{ll} & x \ pre \ y \text{ in } t_i \\ \Rightarrow & \exists\ u, v\ ::\ u \ pre \ v \text{ in } t\ \wedge\ u_i = x\ \wedge\ v_i = y \end{array}$$

Proof: For each finite prefix of t_i there is a finite prefix of t such that the former is the projection of the latter. Let v be the shortest prefix of t such that $v_i = y$. Since y is not the empty sequence (because $x \ pre \ y$), v is not the empty sequence. Hence, there is a u such that $u \ pre \ v$ in t. We show that $u_i = x$.

$$\begin{array}{ll} & u \ pre \ v \text{ in } t \\ \Rightarrow & \{\text{F4}\} \\ & (u_i = v_i)\ \vee\ u_i \ pre \ v_i \text{ in } t_i \\ \Rightarrow & \{u_i = v_i\ \Rightarrow\ u_i = y, \text{ because } v_i = y. \text{ That is, projection of } u \text{ is } y, \text{ and} \\ & \text{this contradicts our choice of } v \text{ as the shortest prefix whose projection is } y\} \\ & u_i \ pre \ v_i \text{ in } t_i \\ \Rightarrow & \{v_i = y\ ,\ x \ pre \ y \text{ in } t_i\} \\ & u_i = x \end{array}$$

3.2 Description

3.2.1 Informal Explanation

The mechanism that we use to define processes is called a *description*. A description is similar to an equation; however its two sides do *not* commute.

When does a description describe a process? Since a process is a set of traces (over a given set of incident channels) we require that the set of "smooth" solutions of the description be the set of process traces. A trace is a smooth solution of a description if (1) it is a solution of the description, viewed as an equation, and (2) its finite prefixes satisfy a certain "smoothness" condition. The smoothness condition formalizes the operational behavior of a machine that an output data item can depend only on the input received *prior to* sending this output; thus, no output can be caused by (i.e., depend upon) itself as input. (For instance, in the example given in Section 2.3, the sequence z violates the smoothness condition because its first item, -1, can be justified only by viewing it as the input that causes itself to be output.) This property of communicating processes was exploited formally in Chandy and Misra [1981].

3.2.2 Definitions

A *description* is a pair of continuous functions from traces to some cpo. A description consisting of the function pair (f, g) will be written as $f\ \leftarrow\ g$.

Trace t is a *smooth solution* of $f\ \leftarrow\ g$ means

- $f(t) = g(t)$ {limit condition}
- $\forall\ u, v\ :\ u \ pre \ v \text{ in } t\ ::\ f(v) \sqsubseteq g(u)$ {smoothness condition}

A process *is described by* $f\ \leftarrow\ g$ means that the set of smooth solutions of $f\ \leftarrow\ g$, projected on the incident channels of the process, is the set of process traces.

3.2.3 Some Results About Descriptions

Lemma 2: Let trace t be a smooth solution of $f\ \leftarrow\ g$. Then for any finite prefix v of t, $f(v) \sqsubseteq g(v)$.

Proof:

case 1) $t = \perp$: The only finite prefix of t is $\perp$.

$\{t$ satisfies the limit condition of $f \leftarrow g$ and $t = \perp\}$
$f(\perp) = g(\perp)$
$\Rightarrow \quad f(\perp) \sqsubseteq g(\perp)$.

case 2) $t \neq \perp$: Then there is a u such that either v *pre* u in t or u *pre* v in t.

case 2.1) v *pre* u in t:

$f(v)$
$\sqsubseteq \quad \{v \sqsubseteq u$ since v *pre* u in t, f is monotonic$\}$
$f(u)$
$\sqsubseteq \quad \{t$ satisfies the smoothness condition of $f \leftarrow g$ and v *pre* u in $t\}$
$g(v)$

case 2.2) u *pre* v in t:

$f(v)$
$\sqsubseteq \quad \{t$ satisfies the smoothness condition of $f \leftarrow g$ and u *pre* v in $t\}$
$g(u)$
$\sqsubseteq \quad \{u \sqsubseteq v$ since u *pre* v in t , g is monotonic$\}$
$g(v)$

The following theorem provides a simpler characterization of the smooth solutions of a special class of descriptions. It follows from this theorem that for deterministic processes Kahn-style equations can be readily converted to descriptions.

Functions f, g are *independent* if there exist disjoint subsets of *channels*, L and M, such that for every trace t,

$$f(t) = f(t_L) \qquad \text{and} \qquad g(t) = g(t_M) \ .$$

In syntactic terms, no variable (i.e., channel) is named in both f, g. Such is the case for the description of a deterministic process whose outputs can be written as functions of its inputs. This is also the case for a nondeterministic process like *dfm* (Section 2.2). However the description of the network of Section 2.3 (when viewed as a process with output channel d), does not meet this criterion because d is named on both sides of the description.

Theorem 1: Let f, g be independent. Then

t is a smooth solution of $f \leftarrow g$
$\equiv \quad f(t) = g(t) \quad \wedge$
$\forall\ s\ :\ s$ a finite prefix of $t\ ::\ f(s) \sqsubseteq g(s)$.

Proof: Proof is by mutual implication.

1. Let t be a smooth solution of $f \leftarrow g$. Then, from the limit condition for smooth solutions, $f(t) = g(t)$. The second condition (about finite prefixes) follows from Lemma 2.

2. Assume that $f(t) = g(t)$ and for all finite prefixes s, $f(s) \sqsubseteq g(s)$. In order to prove that t is a smooth solution of $f \leftarrow g$, the only nontrivial task is to prove the smoothness condition, i.e.,

$$u \text{ pre } v \text{ in } t \ \Rightarrow \ f(v) \sqsubseteq g(u) \ .$$

From u *pre* v in t, v is an extension of u by a pair (c, m). From the independence of f, g, channel c does not belong to both L and M. Therefore,

either $u_L = v_L$ or $u_M = v_M$

case 2.1) $u_L = v_L$:

$f(v)$

$=$ $\{f(t) = f(t_L)$ for all $t\}$

$f(v_L)$

$=$ $\{u_L = v_L$ from the assumption of case 2.1$\}$

$f(u_L)$

$=$ $\{f(t) = f(t_L)$ for all $t\}$

$f(u)$

$\sqsubseteq$ {assumption in 2, with s replaced by u}

$g(u)$

case 2.2) $u_M = v_M$:

$f(v)$

$\sqsubseteq$ {assumption in 2, with s replaced by v}

$g(v)$

$=$ {similar to the proof steps in case 2.1}

$g(u)$

3.3 An Operational View of Smooth Solutions

Given a description, how do we find its smooth solutions? This question is not relevant to understanding the theory in this paper. However, it does provide some intuition for understanding the notion of description. The method, outlined below, is a generalization of the method for obtaining least fixpoints of continuous functions.

For the least fixpoint computation of a continuous function f, we start with $\perp$ as the initial approximation, and we compute the chain $f(\perp), f^2(\perp) \ldots$, etc. The least fixpoint is the *lub* of this chain (from the fixpoint theorem). A description $g \leftarrow h$ may have many smooth solutions. Hence we employ a tree, rather than a linear chain, for obtaining the smooth solutions. The root of the tree is labelled $\perp$. For finite sequences u, v, a vertex labelled u has a son labelled v iff $u \; pre \; v \; \wedge \; g(v) \sqsubseteq h(u)$. Hence the sons of a node labelled u can be computed by looking at all 1-step extensions, v, of u and identifying the ones that satisfy $g(v) \sqsubseteq h(u)$. We say that s is a smooth solution of $g \leftarrow h$ iff $g(s) = h(s)$ and s is a leaf node, i.e., the *lub* of a path, in the tree. (Note that some leaf nodes may not satisfy the limit condition, $g(s) = h(s)$).

4 Examples of Process Descriptions

We consider a number of processes, most of them nondeterministic, and we show their descriptions. One strategy that we employ to obtain the description of a procss is to show an implementation of it using processes whose descriptions are already known; the description of the former is then obtained from the descriptions of the latter processes. We prove the validity of this approach in the following sections.

Note on Auxiliary Channels: Whenever we obtain the description of a process by using an implementation of it with predefined processes, we introduce auxiliary channels, i.e., channels that were not originally incident on the process. Auxiliary channels can be shown to be essential; see Sec. 8.2 for further discussion.

In this section, we consider a variety of processes by introducing different kinds of nondeterminism and/or fairness constraints. The last example we treat is the Fair-Merge. It has been folklore that any nondeterministic process can be implemented by a network consisting of deterministic processes

and Fair-Merges. In this sense, we can claim that any process can be defined using our method of descriptions. For more rigorous treatment of this claim, and for applications in domains other than linear traces, see Abramsky [1989].

Note on Multiple Descriptions: For some processes we write several descriptions rather than just one. Multiple descriptions can be written as a single description as follows. The two descriptions $f' \leftarrow g'$ and $f'' \leftarrow g''$ may be combined into the description $f \leftarrow g$ where f is the pair (f', f'')—that is, $f(t)$ for any t is the pair $(f'(t), f''(t))$—and similarly, g is the pair (g', g''). The range of f is the cartesian product of the ranges of f' and f''. (Similarly for g). If f', f'' (and g', g'') are continuous then so is f (and g). Define,

$$f(v) \sqsubseteq g(u) \;\equiv\; f'(v) \sqsubseteq g'(u) \;\wedge\; f''(v) \sqsubseteq g''(u) \;.$$

In the following examples we use a channel name, c, to denote the function that maps a trace to the sequence associated with c in the trace.

4.1 CHAOS

This process has a single output channel, b. It sends any sequence of messages, from the associated message set of b, along b. Hence every trace (over b and its messages) is a trace of this process. This process is called CHAOS in Hoare [1985].

This is an example for which it is possible to synthesize a description, $f \leftarrow g$, using the fact that all traces are smooth solutions. Let u, v be two finite traces where u *pre* v in v. Then,

$$\begin{array}{ll} & f(v) \\ \sqsubseteq & \{v \text{ is a smooth solution. Applying the smoothness condition to } u \textit{ pre } v \text{ in } v\} \\ & g(u) \\ = & \{u \text{ is a smooth solution. Applying the limit condition to } u\} \\ & f(u) \\ \sqsubseteq & \{u \sqsubseteq v \text{ and } f \text{ is monotonic}\} \\ & f(v) \end{array}$$

Hence $f(u) = f(v)$. Let $f(u) = K$. Applying induction on trace length, it follows that $f(s) = K$, for all finite traces s. Then using the continuity of f, it follows that $f(t) = K$, for all traces t. Since any t is a smooth solution of $f \leftarrow g$, from the limit condition, $f(t) = g(t)$, i.e., $g(t) = K$.

Hence a description of CHAOS is $K \leftarrow K$, where K is any constant function.

4.2 Ticks

This process has a single output channel, b. The process outputs an unending stream of T's. ("T" is for tick.)

This is a deterministic process. Its only trace is $(b, T)^\omega$, the sequence consisting of an infinite number of pairs (b, T). A description for this process is,

$$b \leftarrow T; b$$

The reader is urged to prove the smoothness condition.

4.3 Random Bit

This process has an output channel b along which it outputs a single bit, T or F, and then halts. To describe this process, we introduce a function, R, where

$$R \;:\; \{T, F, \bot\} \;\rightarrow\; \{T, \bot\}$$

In the domain of R, $\bot \sqsubseteq T$, $\bot \sqsubseteq F$; in the range, $\bot \sqsubseteq T$. R is defined by the following table.

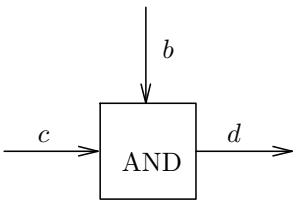

Figure 5: An implementation for Section 4.5

x:	T	F	$\perp$
$R(x)$:	T	T	$\perp$

Let $R(b)$ be the function that yields a sequence by applying R pointwise to b. (It can be shown that this function is continuous.) In the following we use $\overline{T}$ to denote the constant function whose value is the sequence $\overline{T}$. A description of the process is

$$R(b) \ \leftarrow \ \overline{T}$$

4.4 Random Bit Sequence

This process is a variation of the one in Section 4.3. It has additionally an input channel c along which it receives a sequence of T's. The goal is to output a sequence of random bits along b, one bit for each tick in c. In particular, an infinite sequence of random bits is obtained by supplying an infinite sequence of ticks (see Section 4.2) to this process. A description for this process is

$$R(b) \ \leftarrow \ c$$

4.5 Implication

This process has an input channel c and an output channel d. The process receives at most one bit, T or F, and it outputs a bit after receiving the input. The output bit is F if the input is F; it is arbitrary (T or F) otherwise.

The traces of this process are

$$\perp \qquad \overline{(c,T)(d,T)} \qquad \overline{(c,T)(d,F)} \qquad \overline{(c,F)(d,F)}$$

A description for it is obtained by employing the implementation shown in Figure 5. Here b is a random bit (see Section 4.3) and 'AND' is a function,

$$\text{AND} \ : \ \{T,F,\perp\}^2 \ \rightarrow \ \{T,F,\perp\}$$

where the result of AND is $\perp$ if either argument is $\perp$, T if both arguments are T, and F otherwise. (Function AND can be applied to a pair of sequences by applying AND pointwise to the corresponding elements of the two sequences.)

The complete description, including the description for b, is

$R(b) \ \leftarrow \ \overline{T}$ {see Section 4.3}
$d \ \leftarrow \ b$ AND c

Note: This description introduces an auxiliary channel b. The meaning of auxiliary channels is taken up in Section 8.2.

Note: We leave it to the reader to decide why

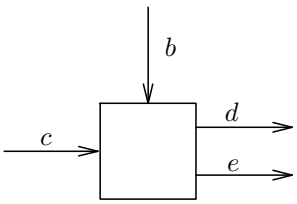

Figure 6: Implementation of fork process of Section 4.6

$$d \leftarrow c \text{ AND } d$$

is not a description for this process. Also, can a nonstrict AND—result of AND is T if both arguments are T, it is F if either argument is F and $\perp$, otherwise—be used in the original description?

4.6 Fork

This process has an input channel c and output channels d and e. Every item it receives is sent along one of the output channels. There is no fairness requirement.

An implementation for this process is shown in Figure 6. Here b is an infinite sequence of random bits (see Section 4.4). For each item received along c, the next element of b, T or F, determines the output channel, d or e, respectively, on which the item is to be sent. {Sequence b is called an *oracle* (Park [1982]).} Below, function g extracts the subsequence of c corresponding to the positions where sequence b has T; similarly, function h is defined.

$$d \leftarrow g(c, b)$$
$$e \leftarrow h(c, b)$$

4.7 Fair Random Sequence

This process has an output channel c along which it outputs a sequence consisting of an infinite number of T's and infinite number of F's. Let TRUE, FALSE be functions from sequences of $\{T, F\}$ to sequences of $\{T, F\}$ where TRUE(s) is the subsequence of s consisting of T's only; similarly FALSE is defined. The description of this process is

$$\text{TRUE}(c) \leftarrow \textit{trues}$$
$$\text{FALSE}(c) \leftarrow \textit{falses}$$

where *trues* is the infinite sequence of T's and *falses* is the infinite sequence of F's (for their descriptions see Section 4.2).

4.8 Finite Sequence of Ticks

This process has an output channel d along which it sends a finite number of T's and then halts. Note that a fairness property is being implemented by this process: $(d, T)^\omega$ is *not* a trace of this process though $(d, T)^i$, for every i, $i \geq 0$ is a trace.

We implement this process by introducing an auxiliary input channel c along which a fair random sequence (see Section 4.7) is supplied. Each input item in c is reproduced in d until the first F. Let g be the function which extracts the prefix of a sequence until the first F (more formally, $g(s)$ is the longest prefix of s that contains no F; it can be shown that g is continuous). A description for the process is,

$$d \leftarrow g(c)$$

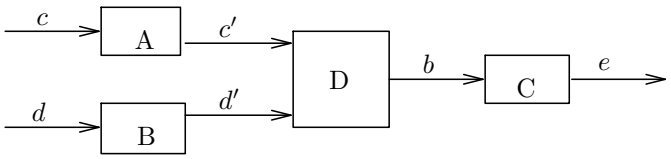

Figure 7: Implementing fair merge (Section 4.10)

4.9 Random Number

This process outputs a natural number on its output channel d and then halts. This process can be implemented as follows: receive a fair random sequence (see Section 4.7) along an input channel c; count the number of T's as they are received; upon receiving the first F, output the count and then halt. The counting process can be described by a function h. The process description is

$$d \leftarrow h(c)$$

where the description of c is as in Section 4.7.

4.10 Fair Merge

This process has input channels c, d along which it receives integers. It outputs a sequence along channel e that satisfies:

- every item sent along e can be identified uniquely as an item of c or d, and
- every finite prefix of c (and of d) is a subsequence of some finite prefix of e.

We have seen a simpler version of this process, dfm, in Section 2.2. We implement fair merge using a variation of dfm. The schema of the implementation is in Figure 7.

The idea of the implementation is as follows. Process A outputs the pair $(0, n)$ for each input n it receives; process B outputs $(1, n)$ upon receiving n. Thus, processes A,B apply "tags"—0 or 1—to the inputs they receive. Process D performs a fair merge; a description for it is

$$\text{ZERO}(b) \leftarrow c' \ , \ \text{ONE}(b) \leftarrow d'$$

where ZERO/ONE applied to a sequence of pairs yields the subsequence whose first component is 0/1. Process C outputs the second component of every pair it receives. Processes A,B,C are deterministic. Let $t0, t1, r$ denote the functions implemented by A,B,C, respectively. Then a description of fair merge is

$$c' \leftarrow t0(c) \ , \ d' \leftarrow t1(d) \ ,$$
$$\text{ZERO}(b) \leftarrow c' \ , \ \text{ONE}(b) \leftarrow d' \ ,$$
$$e \leftarrow r(b)$$

Eliminating c', d' {justification for such elimination is in Section 7}

$$\text{ZERO}(b) \leftarrow t0(c) \ , \ \text{ONE}(b) \leftarrow t1(d) \ , \ e \leftarrow r(b)$$

We have introduced an auxiliary channel b in this description.

To appreciate some of the difficulties of dealing with unbounded nondeterminacy and fair merge, we refer the reader to Apt and Plotkin [1986] and Panangaden and Stark [1987].

5 Composition Theorem

We show in this section that descriptions of the component processes of a network constitute a description of the network. This provides part of the justification for the strategy we used in Section 4 to obtain descriptions of processes from their implementations. (The other part of the justification—eliminations of variables—is left to Section 7.)

Consider a network whose component process i has a description $f_i \leftarrow g_i$. We expect f_i, g_i to depend only on the traces of the process; that is, $f_i(t) = f_i(t_i)$ and $g_i(t) = g_i(t_i)$. We term this constraint dc (for *d*escription *c*onstraint).

Theorem 2: (Composition Theorem) Consider a network N whose i^{th} component process is described by $f_i \leftarrow g_i$, where f_i, g_i satisfy dc. Then $f \leftarrow g$ describes network N where f is the tuple whose i^{th} component is f_i (and similarly g).

Proof: First we prove a sublemma.

Sublemma: For trace t,

t is a smooth solution of $f \leftarrow g \;\equiv\; \forall\, i \;::\; t_i$ is a smooth solution of $f_i \leftarrow g_i$

Proof of the Sublemma: The proof consists of three parts, corresponding to (1) the limit condition and (2,3) the smoothness condition, which is proved by mutual implication.

1.

$f(t) = g(t)$
$\equiv$ {definitions of f and g}
$\forall\, i \;::\; f_i(t) = g_i(t)$
$\equiv$ {using $dc: \; f_i(t) = f_i(t_i) \;,\; g_i(t) = g_i(t_i)$}
$\forall\, i \;::\; f_i(t_i) = g_i(t_i)$

2. The smoothness condition for $f \leftarrow g$ is satisfied by t
$\Rightarrow$ $\forall\, i$ :: the smoothness condition for $f_i \leftarrow g_i$ is satisfied by t_i.

That is, we have to show for every i,
$x \; pre \; y$ in $t_i \;\Rightarrow\; f_i(y) \sqsubseteq g_i(x)$
assuming that the smoothness condition is satisfied by t.
Given $x \; pre \; y$ in t_i, using F5, there exist u, v where
$u \; pre \; v$ in $t \;\wedge\; u_i = x \;\wedge\; v_i = y$. Then,

$u \; pre \; v$ in t
$\Rightarrow$ {t satisfies the smoothness condition for $f \leftarrow g$}
$f(v) \sqsubseteq g(u)$
$\Rightarrow$ {definitions of f, g}
$f_i(v) \sqsubseteq g_i(u)$
$\Rightarrow$ {using $dc: \; f_i(v) = f_i(v_i) \;,\; g_i(u) = g_i(u_i)$}
$f_i(v_i) \sqsubseteq g_i(u_i)$
$\Rightarrow$ {$v_i = y \;,\; u_i = x$}
$f_i(y) \sqsubseteq g_i(x)$

3. $\forall\, i$ :: The smoothness condition for $f_i \leftarrow g_i$ is satisfied by t_i
$\Rightarrow$ the smoothness condition for $f \leftarrow g$ is satisfied by t.

Assume that the smoothness condition is satisfied by every t_i. We have to show,
$u \; pre \; v$ in $t \;\Rightarrow\; f(v) \sqsubseteq g(u)$

u *pre* v in t
$\Rightarrow$ {using F4}
$(u_i = v_i) \ \vee \ (u_i$ *pre* v_i in $t_i)$
$\Rightarrow$ $\{(u_i = v_i) \ \Rightarrow \ g_i(u_i) = g_i(v_i)\}$
$[g_i(u_i) = g_i(v_i)] \ \vee \ [u_i$ *pre* v_i in $t_i]$
$\Rightarrow$ {t_i is a smooth solution of $f_i \ \leftarrow \ g_i$ and v_i is a finite prefix of t_i. Hence, from Lemma 2, $f_i(v_i) \sqsubseteq g_i(v_i)$. Using this fact on the first disjunct}
$[f_i(v_i) \sqsubseteq g_i(u_i)] \ \vee \ [u_i$ *pre* v_i in $t_i]$
$\Rightarrow$ {t_i satisfies the smoothness condition for $f_i \ \leftarrow \ g_i$.
Hence, u_i *pre* v_i in $t_i \ \Rightarrow \ f_i(v_i) \sqsubseteq g_i(u_i)$}
$[f_i(v_i) \sqsubseteq g_i(u_i)] \ \vee \ [f_i(v_i) \sqsubseteq g_i(u_i)]$
$\Rightarrow$ $f_i(v_i) \sqsubseteq g_i(u_i)$
$\Rightarrow$ {using *dc*: $f_i(v_i) = f_i(v) \ , \ g_i(u_i) = g_i(u)$}
$f_i(v) \sqsubseteq g_i(u)$

Since i is arbitrary, the above holds for all i. Using definitions of f, g
$f(v) \sqsubseteq g(u)$

This completes the proof of the sublemma. To complete the proof of the theorem we have to show that $f \ \leftarrow \ g$ is a description of network N, i.e., for any trace t,

t is a smooth solution of $f \ \leftarrow \ g \ \equiv \ t$ is a trace of network N.

The proof is as follows.

t is a smooth solution of $f \ \leftarrow \ g$
$\equiv$ {from the sublemma proven above}
$\forall \ i \ :: \ t_i$ is a smooth solution of $f_i \ \leftarrow \ g_i$
$\equiv$ {definition of description of a process; see Section 3.2.2}
$\forall \ i \ :: \ t_i$ is a trace of process i
$\equiv$ {definition of network trace; see Section 3.1.2}
t is a trace of network N

6 Least Fixpoint as a Smooth Solution

We show that for deterministic networks the only smooth solution is the least fixpoint. This is another proof of Kahn's result, and it also establishes that the notion of smooth solution is a generalization of the notion of least fixpoint. (See Lynch and Stark [1988] for another proof of Kahn's result based on an operational view of a network of communicating processes.)

Let h be a continuous function from a cpo to itself. An element z of the cpo is a *fixpoint* of h if it satisfies $z = h(z)$. A fixpoint z is a *least fixpoint* (of h) if $z \sqsubseteq y$ for any fixpoint y. The least fixpoint, if it exists, is unique.

Theorem 3: Fixpoint Theorem (see Loeckx and Sieber [1984])
Let h be a continuous function and $T = \{h^i(\perp) \mid i \geq 0\}$. Then the least fixpoint of h is $lub(T)$.

Let *id* denote the identity function over some cpo. We show that the only smooth solution of $id \ \leftarrow \ h$ is the least fixpoint of h. However, we have defined smooth solutions only in the cpo of traces. We extend our definition of smooth solution to arbitrary cpo's and then prove this result.

Let $(D, \sqsubseteq)$ be a cpo. A *countable* chain S in D is a chain whose elements can be indexed with natural numbers such that for x^n, x^{n+1} in S, $x^n \sqsubseteq x^{n+1}$; also $x^0 = \perp$. We write u *pre* v in S to mean there exist x^n, x^{n+1} in S, where $u = x^n$ and $v = x^{n+1}$.

Let f, g be continuous functions from D to some cpo. An element z of D is a *smooth solution* of $f \leftarrow g$ if it is the *lub* of a countable chain S (in D) that satisfies,

- $f(z) = g(z)$ {limit condition} and,
- $u \; pre \; v$ in $S \Rightarrow f(v) \sqsubseteq g(u)$ {smoothness condition}

Note: This definition of smooth solution when restricted to the cpo of traces can be shown to be the same as the previous definition.

Theorem 4: The only smooth solution of $id \leftarrow h$, for a continuous function h, is the least fixpoint of h.

Proof: Let T be the set,

$$T = \{h^i(\perp) \mid i \geq 0\}$$

Using the fixpoint theorem, T is a chain and the least fixpoint of $h = lub(T)$. Note that T is a countable chain.

Proof of the theorem is in two parts, by mutual implication.

1. Let z be the least fixpoint of h. We show that z is a smooth solution of $id \leftarrow h$. From the fixpoint theorem, $z = lub(T)$. We next show that T satisfies the limit and smoothness conditions.

   1.1 Limit condition, $id(z) = h(z)$: $z = h(z)$, because z is a fixpoint of h.
   1.2 Smoothness Condition:
   $u \; pre \; v$ in T
   $\Rightarrow$ {the definition of T}
   $\exists \; n \; :: \; u = h^n(\perp) \; \wedge \; v = h^{n+1}(\perp)$
   $\Rightarrow$ $v = h(u)$
   $\Rightarrow$ $id(v) \sqsubseteq h(u)$

2. Let z be a smooth solution of $id \leftarrow h$. We show that z is the least fixpoint of h.

   Since z is a smooth solution it is the *lub* of a countable chain S that satisfies the limit and smoothness conditions. Let the elements of S be $x^0, \ldots x^n, \ldots$, where $x^0 = \perp$. We show that $x^n \sqsubseteq h^n(\perp)$, for all n, $n \geq 0$. Proof is by induction on n.

   $n = 0$: $x^0 \sqsubseteq h^0(\perp)$, , because $x^0 = \perp$, and $h^0(\perp) = \perp$.
   $n + 1, n \geq 0$:
   {induction hypothesis}
   $x^n \sqsubseteq h^n(\perp)$
   $\Rightarrow$ {h is monotonic}
   $h(x^n) \sqsubseteq h^{n+1}(\perp)$
   $\Rightarrow$ {$x^n \; pre \; x^{n+1}$ in S and S satisfies the smoothness condition for $id \leftarrow h$.
   Hence $x^{n+1} \sqsubseteq h(x^n)$.}
   $x^{n+1} \sqsubseteq h^{n+1}(\perp)$.

   Thus for every element, x^n, of S there is an element, $h^n(\perp)$, in T, such that $x^n \sqsubseteq h^n(\perp)$. Using Lemma 1,

   $lub(S) \sqsubseteq lub(T)$
   i.e., $z \sqsubseteq$ least fixpoint of h.

   From the limit condition, $id(z) = h(z)$, i.e., $z = h(z)$. Since $z \sqsubseteq$ least fixpoint of h, z is the least fixpoint of h.

7 Variable Elimination

In Sections 2.3 and 4, we eliminated certain variables from descriptions by replacing them by their right-hand sides. The intuitive understanding we had is that all smooth solutions are preserved by this operation. More precisely, if t is a smooth solution of the original description then t', the projection of t on the retained variables, is a smooth solution of the transformed description. Conversely (and this is more important) for any smooth solution, t', of the transformed description there is a smooth solution t of the original, where t and t' are similarly related. We prove this result, formally, in this section.

Let b be a variable, i.e., channel, to be eliminated (b could be a list of variables; the same arguments will apply). Suppose,

$$b \leftarrow h \ , \ f \leftarrow g$$

are two descriptions. Variable b can be eliminated provided neither h nor f mentions b; we replace b by h in g. To be able to do the latter, it should be possible to write g as a function of two arguments—one is b, and the other, the remaining variables.

Notation: Let c be the subset of *channels* excluding b. Let t_b, t_c denote the projections of trace t on $\{b\}$ and c, respectively. We regard b as a function on traces where $b(t) = t_b$. (In the previous examples it was more convenient to assume that the range of b is sequences; for formal manipulations, and also for the general case where b is a list of variables, it is better to assume that b's range is the set of traces. The difference is merely technical as the two ranges are isomorphic.)

f is *independent* of b means

$$f(t) = f(t_c).$$

Let D1 be the descriptions,

$$b \leftarrow h \ , \ f \leftarrow g$$

where,

(1) h, f are independent of b, and
(2) $g(t) = r(t_b, t_c)$, for some continuous function r, and
(3) $f(\perp) = \perp$

Let D2 be the following description, obtained by replacing b by h in g:

$$f \leftarrow g'$$

where $g'(t) = r(h(t), t_c)$

Theorem 5: Trace t is a smooth solution of D1 $\Rightarrow$ t_c is a smooth solution of D2

Theorem 6: Trace s is a smooth solution of D2, where $s_c = s$
$\Rightarrow$ $\exists\, t \ :: \ s = t_c \ \wedge \ t$ is a smooth solution of D1.

The first theorem is straightforward. The second theorem's proof is harder, requiring an explicit construction of t from s. (The necessity for condition (3), $f(\perp) = \perp$, was discovered during this construction.)

Observation: g' is independent of b, i.e., $g'(t) = g'(t_c)$, because h is independent of b.

Proof of Theorem 5:

Let t be a smooth solution of D1. We show that t_c satisfies the limit and smoothness conditions for D2.

1. Proof of the limit condition, i.e., $f(t_c) = g'(t_c)$:

$f(t_c)$
$=$ $\{f$ is independent of $b\}$
$f(t)$
$=$ $\{t$ satisfies the limit condition of D1; hence, $f(t) = g(t)\}$
$g(t)$
$=$ {rewriting, using r}
$r(t_b, t_c)$
$=$ $\{t$ satisfies the limit condition of D1; hence, $b(t) = h(t)$. Since $b(t) = t_b$, $t_b = h(t)\}$
$r(h(t), t_c)$
$=$ {definition of g'}
$g'(t)$
$=$ {from the above observation, $'$ is independent of $b\}$
$g'(t_c)$

2. Proof of the smoothness condition, i.e., x *pre* y in $t_c \Rightarrow f(y) \sqsubseteq g'(x)$:
From F5,
x *pre* y in $t_c \Rightarrow \exists\, u, v :: u$ *pre* v in $t \wedge x = u_c \wedge y = v_c$
In the following proof u, v are as given above.

$f(y)$
$=$ $\{y = v_c\}$
$f(v_c)$
$=$ $\{f$ is independent of $b\}$
$f(v)$
$\sqsubseteq$ $\{t$ satisfies the smoothness condition in D1 and u *pre* v in t; hence, $f(v) \sqsubseteq g(u)\}$
$g(u)$
$=$ {from condition (2), $g(u) = r(u_b, u_c)\}$
$r(u_b, u_c)$
$\sqsubseteq$ {since t is a smooth solution of D1, and u is a finite prefix of t, $b(u) \sqsubseteq h(u)$,
using Lemma 2. Since $b(u) = u_b$, we get $u_b \sqsubseteq h(u)$.
Note that r is monotonic in both arguments}
$r(h(u), u_c)$
$=$ {definition of g'}
$g'(u)$
$=$ $\{g'$ is independent of $b\}$
$g'(u_c)$
$=$ $\{u_c = x\}$
$g'(x)$

Proof of Theorem 6:

Let s be a smooth solution of D2. Using the finite prefixes of s we construct a chain of traces whose *lub* is a smooth solution of D1.

From s^n, the prefix of length n of s, we construct $(2n + 2)$ traces $t^0, t^1, \ldots, t^{2n+1}$. Here $t^0 = \perp$. For $i \geq 0$, t^{2i+1} is obtained by extending t^{2i} by (zero or more) pairs (b, m) such that t_b^{2i+1} is $h(s^i)$; clearly t_c^{2i+1} is same as t_c^{2i}. Then t^{2i+2} is obtained by extending t^{2i+1} by pairs (d, m) where $d \in c$, such that $t_c^{2i+2} = s^{i+1}$; clearly t_b^{2i+2} is same as t_b^{2i+1}. Rewriting,

$$t^0 = \perp$$

and for $i \geq 0$:

$t_b^{2i+1} = h(s^i)$ $\quad$ $t_c^{2i+1} = s^i$
$t_b^{2i+2} = h(s^i)$ $\quad$ $t_c^{2i+2} = s^{i+1}$

Clearly $t^j \sqsubseteq t^{j+1}$, for all j and hence the t's form a chain. Let t be the *lub* of this chain. We show,

1. $s = t_c$,
2. t satisfies the limit condition for D1,
3. t satisfies the smoothness condition for D1.

Notation: We write $i/2$ for the largest integer that does not exceed the real number $i/2$, in this proof.

Proof of 1: Function c applied to the chain of traces $t^0, t^1, \ldots, t^i, \ldots$ yields the chain of traces $s^0, s^1, \ldots, s^j, \ldots$ because $t_c^i = s^{i/2}$. Since c is a continuous function, the *lub* of the first chain is t and of the second chain is s, we have:

$$s = t_c$$

Proof of 2: We have to show,

$$2.1) \quad b(t) = h(t) \qquad\qquad 2.2) \quad f(t) = g(t)$$

Proof of 2.1) Consider the chains obtained by applying b to the t-chain and h to the s-chain. The two chains are identical because for every s^i there is a t^j such that $h(s^i) = b(t^j)$ (choose $j = 2i+1$) and similarly, for every t^i there is an s^j such that $h(s^j) = b(t^i)$ {choose $j = (i-1)/2$}. Let z be the *lub* of this (common) chain.

$z = h(s)$, since h is continuous and s is the *lub* of s-chain
$z = b(t)$, since b is continuous and t is the *lub* of t-chain
$h(s) = h(t_c)$, $s = t_c$, from 1
$h(t_c) = h(t)$, h is independent of b
$h(t) = b(t)$, from the above four

Proof of 2.2)

$f(t)$
$=$ {f is independent of b}
$f(t_c)$
$=$ {$s = t_c$}
$f(s)$
$=$ {s satisfies the limit condition for D2, i.e., $f(s) = g'(s)$. And $g'(s) = r(h(s), s_c)$}
$r(h(s), s_c)$
$=$ {From 2.1, $h(s) = b(t)$, i.e., $h(s) = t_b$. Also $s_c = s$ and $s = t_c$, from (1)}
$r(t_b, t_c)$
$=$ {definition of g}
$g(t)$

Before proving the next result, we note the following facts which follow directly from the definitions of t^j's.

Observation 1: For $j \geq 0$, $t_b^{j+1} = h(s^{j/2})$

Observation 2: For $j \geq 0$, $t_c^j = s^{j/2}$

Proof of 3) We have to show

$$u \; pre \; v \text{ in } t \;\Rightarrow\; b(v) \sqsubseteq h(u) \;\wedge\; f(v) \sqsubseteq g(u)$$

Note that $u \; pre \; v$ in $t \;\Rightarrow\; \exists \; j \;:\; t^j \sqsubseteq u \sqsubseteq v \sqsubseteq t^{j+1}$. In the following proofs t^j, t^{j+1} refer to the traces as above.

3.1) Proof of $b(v) \sqsubseteq h(u)$:

$$
\begin{array}{ll}
 & b(v) \\
\sqsubseteq & \{v \sqsubseteq t^{j+1}\ ,\ b \text{ is monotonic}\} \\
 & t_b^{j+1} \\
= & \{\text{using Observation 1}\} \\
 & h(s^{j/2}) \\
= & \{\text{using Observation 2, } s^{j/2} = t_c^j\} \\
 & h(t_c^j) \\
= & \{h \text{ is independent of } b\} \\
 & h(t^j) \\
\sqsubseteq & \{t^j \sqsubseteq u\ ,\ h \text{ is monotonic}\} \\
 & h(u)
\end{array}
$$

3.2) Proof of $f(v) \sqsubseteq g(u)$:

$$
\begin{array}{ll}
 & f(v) \\
\sqsubseteq & \{v \sqsubseteq t^{j+1}\ ,\ f \text{ is monotonic}\} \\
 & f(t^{j+1}) \\
= & \{f \text{ is independent of } b\} \\
 & f(t_c^{j+1}) \\
= & \{\text{using Observation 2 to replace the argument}\} \\
 & f(s^{(j+1)/2}) \\
\sqsubseteq & \{\text{For } j = 0, f(s^{(j+1)/2}) = f(s^0). \text{ Since } s^0 = \perp \text{ and } f(\perp) = \perp \text{ from condition (3) for D1,} \\
 & \text{we have } f(v) \sqsubseteq g(u), \text{ for } j = 0. \\
 & \text{For the rest of this proof, } j > 0. \text{ Since } s \text{ satisfies the smoothness condition for D2}\} \\
 & g'(s^{(j-1)/2}) \\
= & \{\text{using the definition of } g'\} \\
 & r(h(s^{(j-1)/2}), s^{(j-1)/2}) \\
\sqsubseteq & \{\text{From Observation 1, } h(s^{(j-1)/2}) = t_b^j. \text{ Also, } s^{(j-1)/2} \sqsubseteq s^{j/2} \text{ and } s^{j/2} = t_c^j, \text{ from} \\
 & \text{Observation 2. Note that } r \text{ is monotonic in both arguments}\} \\
 & r(t_b^j\ ,\ t_c^j) \\
= & \{\text{using the definition of } g\} \\
 & g(t^j) \\
\sqsubseteq & \{t^j \sqsubseteq u\ ,\ g \text{ is monotonic}\} \\
 & g(u)
\end{array}
$$

Note: The following example shows why we need the condition $f(\perp) = \perp$. Let

D1: $b \leftarrow f\ ,\ f \leftarrow b\ .$

Eliminating variable b, we get

D2: $f \leftarrow f.$

D2 has a smooth solution, $\perp$. However, if $f(\perp) \neq \perp$ then D1 has no smooth solution—any nonempty trace violates the smoothness condition for the second description in D1 and the empty trace, $\perp$, violates the limit condition.

Note: The results of this section apply to more general substitutions. If $p \leftarrow h$ is a description in D1 (instead of $b \leftarrow h$) where p depends only on b and p is surjective—i.e., for every y in the range of p there exists an x such that $p(x) = y$—then p can be replaced by h.

Note: It may seem that the following descriptions

D1: $b \leftarrow h\ ,\ f \leftarrow g$
D2: $b \leftarrow h\ ,\ f \leftarrow g'$

have the same set of smooth solutions (where h, f are independent of b and g' is obtained from g by replacing b by h). This is untrue. To see this consider (where v is replaced by w to obtain D2 from D1)

D1: $v \leftarrow w\ ,\ u \leftarrow v$
D2: $v \leftarrow w\ ,\ u \leftarrow w$

The following trace is a smooth solution of D2 but not of D1.

$\overline{(w,0)\quad (u,0)\quad (v,0)}$

8 Some Remarks About This Approach

8.1 Using Arbitrary Predicates to Define Processes

Arbitrary predicates over its input-ouput channels may be used to define a process. We have limited ourselves to a very specific type of predicate in the form of a description. What is the advantage of such a restriction? We can manipulate descriptions as easily as equations—substituting the right side for any occurrence of a left side (Section 7)—to eliminate variables; such manipulations have been applied many times in Section 4. Manipulations of arbitrary predicates have proved cumbersome, at best.

8.2 Auxiliary Channels

In describing certain processes we were forced to introduce *auxiliary channels*, i.e., channels that were not originally incident on those processes. It can be shown that auxiliary channels are essential; some processes cannot be described otherwise (consider a process that outputs a finite number of ticks, Section 4.9).

To introduce auxiliary channels into the theory we add more structure to the set *channels*. Some channels are specified to be auxiliary. A restriction on processes is that no auxiliary channel is incident on two processes; i.e., each auxiliary channel is internal to one process. Suppose a process is defined by its traces that mention only its incident nonauxiliary channels. Then $f \leftarrow g$ is a process description means that for any trace t,

t is a process trace
$\equiv$ $\exists\ s\ ::$ s is a smooth solution of $f \leftarrow g$ and the projection
of s on the incident, nonauxiliary channels of the process is t.

8.3 Functional Programs and Nondeterminism

The methods suggested in this paper are not limited to defining process networks; arbitrary non-functional modules may be so defined. We hesitate to claim that this is a viable approach because a description in the general form does not prescribe a computation procedure. Therefore, we recommend using descriptions as specifications. In traditional functional programming, a description also suffices as a computation procedure.

8.4 Smooth Solution Induction

An induction rule for smooth solutions, similar to fixpoint induction, is the following (refer to Section 6 for a definition of smooth solution over an arbitrary cpo).

For any admissible predicate ϕ (see Loeckx and Sieber [1984] for a definition of admissible predicate) and description $f \leftarrow g$, if,

$\phi(\perp)$ and,
$[u \sqsubseteq v \ \wedge \ f(v) \sqsubseteq g(u) \ \wedge \ \phi(u)] \ \Rightarrow \ \phi(v)$

then, $\phi(z)$ holds for every smooth solution z of $f \leftarrow g$.

For the cpo of traces, this rule can be strengthened by replacing $u \sqsubseteq v$ by $u \ pre \ v$ in v.

Unfortunately—as noted by Trakhtenbrot in personal communication—this induction rule does not exploit the limit condition, and hence may be too weak.

Acknowledgment: I am indebted to Samson Abramsky, Edsger W. Dijkstra, C.A.R. Hoare, Amir Pnueli, and Boris Trakhtenbrot for their suggestions and comments. Special thanks go to Prakash Panangaden, Jim Russell, and the Austin Tuesday afternoon club for thorough readings of the manuscript and constructive criticisms. I am grateful to the referees and C. B. Jones for their careful reading of an earlier manuscript, and for many insightful suggestions. The earlier works on which this paper is based were carried out jointly with K. M. Chandy. I am indebted to Nancy E. Lawler for her excellent typing.

References

Abramsky, S., "A Generalized Kahn Principle for Abstract Asynchronous Networks," *Proc. Symposium on Mathematical Foundations of Programming Language Semantics, Tulane, New Orleans, May 1989.*

Apt, K. R., and G. D. Plotkin, "Countable Nondeterminism and Random Assignment," *J. ACM 33*, **4**, pp. 727–767, 1986.

Brock, J. D., and W. B. Ackerman, "Scenarios: a model of nondeterminate computation," in *Formalization of Programming Concepts* (eds. J. Diaz and I. Ramos), *Lecture Notes in Computer Science*, **107**, New York: Springer-Verlag, pp. 252–259, 1981.

Broy, M., "Semantics of finite and infinite networks of concurrent communicating agents," *Distributed Computing 2*, pp. 13–31, 1987.

Chandy, K. M. and J. Misra, "Proofs of networks of processes," *IEEE*, **SE-7:4**, pp. 417–426, July 1981.

Chandy, K. M. and J. Misra, "Reasoning about networks of communicating processes," *Unpublished*, presented at INRIA Advanced Nato Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, La Colle-sur-Loupe, France, 1984.

Dijkstra, E. W., and C. S. Scholten, *Predicate Calculus and Program Semantics*, Springer-Verlag, New York, 1989.

Gasteren, A. J. M. van, *On the Shape of Mathematical Arguments*, Ph.D. Dissertation, Technische Universiteit Eindhoven, 1988.

Hoare, C. A. R., *Communicating Sequential Processes*, London: Prentice-Hall International, 1985.

Jonsson, B., "A model and proof system for asynchronous networks," in *Proc. 4th/ ACM Symp. on Principles of Distributed Computing*, pp. 49–58, Minaki, Canada, 1985.

Jonsson, B., "Modular verification of asynchronous networks," in *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pp. 152–166, Vancouver, Canada, 1987.

Jonsson, B., "A fully abstract trace model for dataflow networks," *Proc. 16th Annual ACM Symp. on Principles of Programming Languages, Austin, Texas, January 1989*.

Kahn, G., "The semantics of a simple language for parallel programming," *Proc. of IFIP Congress 74*, Amsterdam: North Holland, 1974.

Keller, R. M., "Denotational Models for Parallel Programs with Indeterminate Operators," in *Formal Descriptions of Programming Concepts*, ed. E. J. Neuhold, Amsterdam: North-Holland, pp. 337–365, 1978.

Keller, R. M., and P. Panangaden, "Semantics of Networks Containing Indeterminate Operators," *Distributed Computing*, **1**:235–245, 1986.

Kok, J. N., "A fully abstract semantics for data flow nets," in *Proc. PARLE, LNCS 259*, Springer-Verlag, pp. 351–368, 1987.

Loeckx, J., and K. Sieber, *The Foundations of Program Verification*, John Wiley & Sons Ltd. and B. G. Teubner, Stuttgart, 1984.

Lynch, N. A., and E. W. Stark, "A proof of the Kahn principle for input/output automata," *MIT/LCS/TM-349*, January 1988.

Panangaden, P., and E. W. Stark, "Computations, residuals, and the power of indeterminacy," *TR87–883, Cornell University*, November 1987.

Panangaden, P., and V. Shanbhogue, "On the expressive power of indeterminate network primitives," *TR87–891, Cornell University*, December 1987.

Park, D., "The 'fairness' problem and nondeterministic computing networks," *Foundations of Computer Science IV Part 2, Amsterdam, Math. Centre Tracts 159*, pp. 133–161, 1982.

Plotkin, G., "A power domain construction," *SIAM Journ. on Com.*, **5**, pp. 452–487, 1976.

Rabinovich, A., and B. A. Trakhtenbrot, "Nets and Data Flow Interpreters," *Proc. 4th Annual Symp. on Logic in Computer Science*, Asilomar, California, June 5–8, 1989.

Scott, D., "Outline of a mathematical theory of computation," *4th Annual Princeton Conf. Inform. Sc. and Systems*, pp. 169–176, 1970.

Staples, J., and V. N. Nguyen, "A fixpoint semantics for nondeterministic data flow, *J. ACM 32*, **2**, pp. 411–444, 1985.