POSSIBLE FUTURES, ACCEPTANCES, REFUSALS, AND COMMUNICATING PROCESSES

W.C. Rounds* and S.D. Brookes

Computer and Communication Sciences
University of Michigan

Programming Research Group
Oxford University

Abstract

Two distinct models for the notion of communicating processes are introduced, developed and related. The first, called the possible-futures model, is a generalization to nondeterministic systems of the familiar derivative (Nerode equivalence class) construction. The second, called the acceptance-refusals model, is a slight strengthening of a model introduced by Hoare, Brookes, and Roscoe. The PF model can be mapped onto the AR model homomorphically, and the equivalence classes of this map can be characterized by imposing a very natural equivalence relation on the PF model. The resulting quotient algebra admits a complete partial order structure in which the algebraic operations are continuous.

I. Introduction

We propose two mathematical models for the algebra of communicating sequential processes introduced by Hoare [H]. We think of CSP 'programs' as expressions denoting processes with concurrent and nondeterministic behavior, and assign meanings to these expressions using the methods both of formal language theory and denotational semantics. Starting with the primitive notions of event and sequence of events, we are able to define certain features of nondeterministic transition systems without having to introduce the notion of states. This makes it possible to introduce recursive expressions into the language and to assign them meanings as well.

Our first model is called the possible-futures model. It is intended to abstract a nondeterministic transition system, and was invented as the result of an attempt to define the so-called Nerode equivalence for nondeterministic systems. The second model, called the acceptance-refusal machine (ARM) model, was directly inspired by the refusal machines of Hoare, Brookes, and Roscoe [HBR]. ARMS are a slight strengthening of refusal machines and arose as the result of investigating the connections between refusal machines and possible futures. The results of that investigation will appear in a companion paper [BR].

*Research supported in part by sabbatical leave from University of Michigan and NSF Grant MCS-8102286.

The most interesting results from a theoretical point of view are, of course, the connections between the two models. We spend some time developing the first one, whose elements, called processes, are defined as certain relations on prefix-closed sets of strings. An algebra of processes is introduced based on the operations in CSP. In particular, we study the operations of autonomous choice, controllable choice, guarding, parallel composition, renaming, hiding, sequential composition, and inverse image. The possible-futures model, however, does not lend itself well to the general solution of recursive equations in these operators. A method is needed to introduce a partial ordering on processes which is complete in the sense of denotational semantics, and for which the operations are continuous. This we do by defining an equivalence relation-called testable equivalence-on the possible-futures model, which is a congruence with respect to the algebra of processes, and which induces a (complete) partial order on the quotient space in a very natural way.

At this point, the ARM machines are introduced. We show that each equivalence class of testable equivalence is naturally respresented by a machine. In fact, we demonstrate that testable equivalence is a congruence by constructing a homomorphism from the set of processes onto the set of machines. The algebra of machines is very much the same as that given by Hoare, Brookes, and Roscoe. This paper thus vindicates their definitions by showing how they can be derived from a more general model.

Several authors have attributed meanings to CSP([FHLD], [FLP]). The present work, as well as that of Hoare, Brookes, and Roscoe, focuses on an abstract version of the language, which has no identifier conventions, labeling conventions, provision for variables, or scoping rules. We have attempted rather to make the mathematical semantics as universal and precise as possible, in hopes that it can be adapted to particular language designs.

We should also contrast these models with other models of parallelism in the theoretical literature.

We make a general distinction between the notion of a state-based and an event-based model. The former have been well-studied (see Keller [K] for a good representative) and are useful for many correctness properties. The latter models are newer (see Milner [M] for the most comprehensive treatment) and are oriented toward algebraic and denotational methods. Our models are of this second kind.

Event-based models of parallelism consider the notion of action or event, and sequences thereof, to be primary. The temporal ordering of actions, and the occurrence of non-occurrence of certain events are the central idea which they try to capture. In contrast, the notion of shared variables or shared storage is primary for the state-machine models. There is an emphasis on the values computed by the program, and consequently on the notion of internal state, which is explicitly given for these models. This fact implies that the techniques of program correctness can be used (or at least generalized) to deal with properties of concurrent programs, and that theoretical constructions, like nondeterministic automata, can be used to understand behavior.

This paper is organized as follows. Section II develops the possible-futures model; Section III presents the relevant algebraic operations, and Section IV makes the connection between the two models.

II. The Possible-Futures Model

To begin, let us assume that a universal set Σ of events is predefined. The elements of Σ are presumed to be individual actions which are indivisible and observable in the outside world. As in automata theory, events may be the names of functions on an internal state space. More generally, they may contain explicit information about values-for example, transmission of a value along a wire might be considered an event. The set Σ should be regarded as large.

If we assume that a system has internal states, it becomes a problem to define them or to discuss them using only the notion of event. Fortunately, however, the problem has already been treated in automata theory. Suppose that L is a regular subset of Σ^*. We can define the Nerode equivalence relation on Σ^* determined by L as follows:

$$x \equiv_L y \text{ if and only if } L \text{ after } x = L \text{ after } y,$$

where $L \text{ after } x = \{z \mid xz \in L\}$. The equivalence classes [x] of this relation serve as internal states for a minimal machine accepting the language L. If q and q' are states, and $s \in \Sigma^*$, we denote by $q \xrightarrow{s} q'$ the passage from state q to state q' by applying the input string s. Phrased in terms of equivalence classes, this passage is written

$$[x] \xrightarrow{s} [xs]$$

from which it follows that the transition relation is a function.

Now consider another way of writing this function. Each string x such that $L \text{ after } x \neq \emptyset$ determines an equivalence class, which can be represented by L after x. The strings w such that $L \text{ after } w = \emptyset$ form one extra class which is a dead state in the minimal machine. The set of pairs

$$f_L = \{\langle x, L \text{ after } x\rangle \mid L \text{ after } x \neq \emptyset\}$$

is a partial function on Σ^* whose domain is the prefix-closure of L. The set L after x represents the future behavior of the machine on its way to accepting some extension of x.

Since nondeterministic choice should be captured in our model, we will replace the function f_L by a relation. This relation will now assign a set of future behaviors, each itself a prefix-closed set of strings, to each string in the domain D of the relation. If T is a future behavior assigned to the string x in this way, then we want $T \subseteq D \text{ after } x$. In addition, certain consistency constraints must be met between futures assigned to strings x and to extensions xy of x; these should generalize the equation $L \text{ after } xy = (L \text{ after } x) \text{ after } y$. These constraints will be precisely stated later on.

2.1 Notation

An event is an element $\sigma \in \Sigma$, where Σ is a universal alphabet (generally infinite). A trace is an element of Σ^*, the set of all finite strings over Σ. Λ is the null string. Variables x, y, s, t... range over traces. A tree is a nonempty prefix-closed subset of Σ^*. Variables S, T... range over trees. The set of all trees over Σ is denoted TREES (Σ). Concatenation of strings and sets of strings is denoted as usual in formal language theory:

$$LM = \{xy \mid x \in L,\ y \in M\};\ xT = \{xy \mid y \in T\}.$$

There is a special symbol $\checkmark$ which can only occur at the end of traces, in any set used in the definition of process. We define the semicolon operator on strings as follows:

$$x\checkmark;y = xy$$
$$x;y = x$$

where x contains no occurrence of $\checkmark$. For languages:

$$L;M = \{z;y \mid z \in L,\ y \in M\}.$$

(in this case z may end in $\checkmark$.) Thus

$$L;M = \{xy \mid x\checkmark \in L,\ y \in M, \text{ and } x \text{ is } \checkmark\text{-free}\} \cup \{x \mid x \in L,\ x \text{ is } \checkmark\text{-free, and } x\checkmark \notin L\}$$

In the case that L and M are trees, the semicolon operator is actually a substitution operator, replacing occurrences of $\checkmark$ at the ends of paths by a tree.

We define for $L \subseteq \Sigma^*$ and $x \in \Sigma^*$

$$L \text{ after } x = \{y \mid xy \in L\}.$$

Let h be a function from Σ^* to Σ^*. We write

$h[h] = \{h(x) \mid x \in L\}$, and $h^{-1}[L] = \{z \mid h(z) \in L\}$. The function h is a homomorphism if $h(xy) = h(x)h(y)$ for $x,y \in \Sigma^*$. We stipulate that for homomorphisms, $h(\sigma) = \surd$ iff $\sigma = \surd$.

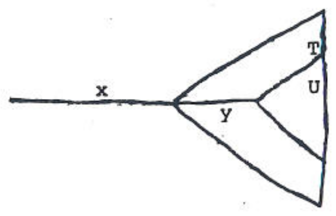
2.2 Definition of Process

A process is a nonempty relation f on $\Sigma^* \times$ TREES (Σ) satisfying

(1) Consistency. For all x, y $\in \Sigma^*$, and U $\in$ TREES(Σ), if $\langle xy, U\rangle \in f$ then $\exists T \in$ TREES(Σ) such that $\langle x, T\rangle \in f$ and $yU \subseteq T$.

(2) Persistence. For all x, y $\in \Sigma^*$, and T $\in$ TREES(Σ), if $\langle x, T\rangle \in f$ and y $\in$ T, then $\exists U \in$ TREES(Σ) such that $yU \subseteq T$ and $\langle xy, U\rangle \in f$.

The figure below shows the relations between x, T, y, and U.



If $\langle x, T\rangle \in f$, we say that T is a possible future for the process f after executing x. The idea is that at subsequent stages, all execution traces will lie in one of the sets T such that $\langle x, T\rangle \in f$. In a sense, the set T is a prediction about the behavior of the process, given that some nondeterministic choices may occur (internally) after the execution of s. Axiom 1 then states that each "later" prediction must be somehow anticipated at all earlier stages. Axiom 2 is a sort of converse property which states that if t is a predicted trace, in a set T, then after execution of st, there will be other predictions made, consistent with the earlier prediction T. These ideas can be clarified with the aid of an example; the one which follows is a modification by W. Ogden of an example due to Hoare.

Imagine a soft-drink machine which communicates with its environment (a customer) using three types of event: insertion of a 25 cent piece (quarter), denoted by "q"; pushing the coin return button, denoted by "r", and pushing the "soda" button, denoted by "s". We are thus using the 3-letter alphabet $\Sigma = \{q, r, s\}$. In the initial state, the machine will always accept a quarter, but will respond to no other event. After the "q" event has occurred, one of two possibilities may occur nondeterministically and without being subject to control by the customer. The quarter may fall into a coin-return slot, in which case the machine will only respond to the "r" event, or the coin may fall into the "give soda" slot, in which case the machine will only respond to the "s" event. Following any of these possibilities, the machine enters a permanent state of breakage.

The situation can be described by the following process:

$$\begin{aligned} SDM = \{&\langle\Lambda, \{\Lambda, q, qr, qs\}\rangle, \\ &\langle q, \{\Lambda, r\}\rangle, \\ &\langle q, \{\Lambda, s\}\rangle, \\ &\langle qr, \{\Lambda\}\rangle, \\ &\langle qs, \{\Lambda\}\rangle\}. \end{aligned}$$

Since no autonomous nondeterministic choices are made in the initial state, there is just one future prediction made after execution of the null trace Λ: the entire set of possible traces. Notice, however, that there are two possible predictions about the future made after executing q. Finally, the only predictions made after executing qr or qs are null; no further execution is possible.

This machine should now be contrasted with a more friendly machine which will always respond to the "r" or "s" event after execution of "q". This machine is subservient to, or controllable by, the customer. It can be "graphed" as follows:

$$\begin{aligned} F\_SDM = \{&\langle\Lambda, \{\Lambda, q, qr, qs\}\rangle, \\ &\langle q, \{\Lambda, r, s\}\rangle, \\ &\langle qr, \{\Lambda\}\rangle, \\ &\langle qs, \{\Lambda\}\rangle\}. \end{aligned}$$

Notice that F_SDM is a function; only one prediction is made at any stage. Such processes are said to be controllable or deterministic. We may introduce a whole class of such processes, which correspond to the Nerode machines at the beginning of this section.

2.2.1 Definition

If T is a prefix-closed set (tree) let

$$\text{proc}(T) = \{\langle x, T \text{ after } x\rangle \mid x \in T\}$$

A process is controllable if it is proc(T) for some tree T. A controllable process has only one possible future at any stage; it has a minimal degree of autonomous nondeterminism.

2.2.2 Further Examples

STOP = $\{\langle\Lambda, \cdot\rangle\}$ (where $\cdot$ is the null tree)
STOP = $\{\langle\Lambda, \cdot\}$. $\{\langle\Lambda, \cdot\rangle\}$
This is the process which does nothing at all.
CHAOS = $\Sigma^* \times$ TREES(Σ).
CHAOS exhibits the least amount of controllability of any process.
RUN = $\{\langle x, \Sigma^*\rangle \mid x \in \Sigma^*\}$.
RUN is a controllable process which never refuses to continue.
CHOOSE = $\{\langle x, \sigma \rightarrow \Sigma^*\rangle \mid x \in \Sigma^*, \sigma \in \Sigma\}$
where $\sigma \rightarrow \Sigma^*$ is the tree $\{\Lambda\} \cup \sigma\Sigma^*$
CHOOSE is a process which at any step can choose to do exactly one action and refuse any other events besides the one it has chosen to do.

SKIP = $\{\langle \Lambda, \underline{\surd} \rangle, \langle \surd, \bullet \rangle\}$.
SKIP does nothing but terminate successfully.

2.2.3 Restriction

Let S be any tree, and let f be a process. We wish to restrict f to the set S. This is accomplished by the definition $f \restriction S =$ $\{\langle x, T \cap (S \text{ after } x)\rangle \mid x \in S \text{ and } \langle x, T\rangle \in f\}$.

It is easy to check that $f \restriction S$ is a process. The processes CHAOS $\restriction$ S, RUN $\restriction$ S, are relativised versions of CHAOS, RUN, etc. Clearly STOP $\restriction$ S = STOP for any S.

2.2.4 A Two-Person Game

Define

$$\text{ALTERNATE} = \{\langle x, \Sigma^*\rangle \mid |x| \text{ is odd}\} \cup \{\langle x, \sigma \to \Sigma^*\rangle \mid |x| \text{ is even}\}$$

ALTERNATE mixes controllable with uncontrollable steps, with an uncontrollable step happening first. We can use this definition together with the notion of restriction to describe the play of a game.

Let C be a set of board configurations. (If the game board is infinite then C will be infinite in general.) Let $c_0 \in C$ be the initial configuration of the board. We wish to model the play of a game G from the point of view of the first player to move. Thus the first move will be an autonomous choice by the first player, but the second will be determined by the second player, and so forth. We define a move $m = \langle c, d\rangle$, where c and $d \in C$, and d follows from c by the rules of the game. A move is an event.

A sequence $m_0\, m_1 \ldots m_k$ is legitimate if $m_0 = \langle c_0, d\rangle$ for some d, and for each $i < k$, if $m_i = \langle c, d\rangle$ then $m_{i+1} = \langle d, e\rangle$ for some e. Now let T_G be the tree $\{\Lambda\} \cup \{\mu \mid \mu$ is a legitimate move sequence$\}$.

The tree T_G is the game tree of G, and the required process has the simple definition

$$\text{ALTERNATE} \restriction T_G.$$

The familiar alternating Turing machines of [CKS] can be defined as a special case of such a game.

III. Operators

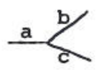
We introduce several operators on processes, with examples, and develop an algebra of these operators. The particular operators we have defined were suggested both by Hoare in CSP and by analogous operators in Hoare, Brookes, and Roscoe. The definitions we give, however, were chosen to be natural with respect to the possible-futures model. The next section vindicates our definitions by establishing that our operators map homomorphically into the corresponding operators for the ARM model.

3.1 Autonomous choice

Let f and g be processes. Define

$$f \sqcap g = f \cup g = \{\langle s, T\rangle \mid \langle s, T\rangle \in f \text{ or } \langle s, T\rangle \in g\}$$

Example. We will use a graphical representation for trees, or prefix-closed sets:

 denotes $\{\Lambda, a, ab, ac\}$

$\underline{a\,b}$ denotes $\{\Lambda, a, ab\}$

$\underline{a}$ denotes $\{\Lambda, a\}$

$\bullet$ denotes $\{\Lambda\}$.

Let $\Sigma = \{q, r, s\}$.

$$f = \{\langle \Lambda, \underline{qr}\rangle, \langle q, \underline{r}\rangle, \langle qr, \bullet\rangle\}$$

$$g = \{\langle \Lambda, \underline{qs}\rangle, \langle q, \underline{s}\rangle\ \ \langle qs, \bullet\rangle\}$$

Then $f \sqcap g$ is a (non-controllable) version of the soft-drink machine discussed earlier, where the autonomous choice is made before insertion of the quarter. In some sense this machine is equivalent to SDM; the equivalence relation will be defined in Section IV. We will be ordering processes by increasing controllability, so the 'meet' symbol $\sqcap$ is used to show that $f \sqcap g$ is less controllable then either f or g.

We wish to have another choice operator which will allow the environment to resolve choices created by alternatives in the component processes, instead of having those choices resolved autonommously as stated earlier. This will make it possible to define guarded commands in which selection of a guarded alternative can be made by the environment.

$$f \,\Box\, g = \{\langle \Lambda, T \cup V\rangle \mid \langle \Lambda, T\rangle \in f; \langle \Lambda, V\rangle \in g\} \cup \{\langle x, Y\rangle \mid \langle x, Y\rangle \in f \cup g \text{ and } x \neq \Lambda\}$$

The meaning of this definition is perhaps clearest when f and g are controllable. $f \,\Box\, g$ has then one future at stage Λ, and is otherwise like $f \sqcap g$. The environment has a chance to resolve choices only at the first step.

Example. Let f and g be as in 3.1; then $f \,\Box\, g$ is exactly SDM.

3.2 Guarding

We wish to define a process which must perform a single event "a" $\in \Sigma$ and then behave like a given process f. The process $a \to f$ is defined as:

$$a \to f = \{\langle \Lambda, a \to T\rangle \mid \langle \Lambda, T\rangle \in f\} \cup \{ax, y\rangle \mid \langle x, y\rangle \in f\}$$

where $a \rightarrow T = \{\Lambda\} \cup a\,T$ prefixes a to every string in T.

3.3 Intersection (pure parallel product)

$$f \,||\, g = \{\langle s, T \cap U\rangle \mid \langle s, T\rangle \in f \text{ and } \langle s, U\rangle \in g\}$$

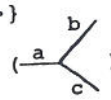
The product $f \,||\, g$ should have traces in common to both processes, and both processes should agree on their common future.

Example. Consider the process

$$f_1(\Lambda) = \{\underline{ab}, \underline{ac}\}$$

$$f_1(a) = \{\underline{b}, \underline{c}\}$$

$$f_1(ab) = f_1(ac) = \{\cdot\}$$

and the process f_2 = proc (). Let g be the process proc $(\underline{a.b})$. Then

$$(g \,||\, f_1)\ (\Lambda) = \{\underline{a.b}, \underline{a}\}$$

$$(g \,||\, f_1)\ (a) = \{\underline{b}, \cdot\}$$

$$(g \,||\, f_1)\ (ab) = \{\cdot\}$$

whereas

$$(g \,||\, f_2)\ (\Lambda) = \{\underline{a.b}\}$$

$$(g \,||\, f_2)\ (a) = \{\underline{b}\}$$

$$(g \,||\, f_2)\ (ab) = \{\cdot\}$$

This example shows that running g in parallel with the autonomous process f_1 can result in a deadlock after a, but that no such deadlock can occur when g runs in parallel with the controllable process f_2.

3.4 Inverse Image

Let h be a string homomorphism. The process $h^{-1}f$ is supposed to be able to execute a trace y whenever f could have executed h(y). A future of $h^{-1}f$ at y will then be the inverse image of a future of f at h(y). If h maps events to Λ, then this future will in general be infinite.

$$h^{-1}f = \{\langle y, h^{-1}[T]\rangle \mid \langle h(y), T\rangle \in f\}.$$

3.5 Direct Image

An admissible homomorphism is one which maps a subset of Σ to Λ, and otherwise maps symbols to symbols. A typical class of admissible homomorphisms are the characteristic homomorphisms of subsets $\Delta \subseteq \Sigma$:

$$\Delta(\sigma) = \begin{cases} \sigma & \text{if } \sigma \in \Delta \\ \Lambda & \text{if } \sigma \notin \Delta \end{cases}$$

Let $\Gamma \subseteq \Sigma$. The characteristic homomorphism of $\Sigma - \Gamma$ is said to be the hiding of Γ.

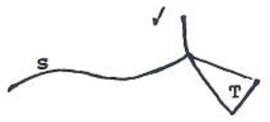
If h is admissible, and f is a process, then we define

$$hf(s) = \{h[X] \mid \exists z.\ h(z) = s \text{ and } X \in f(z)\}$$

We call this the fair direct image of f under h. There is another operator taking into account potential infinite executions of hidden symbols, which replaces these by a null future; this is discussed in the full version of the paper.

3.6 Sequential Composition

We want to define a process from f and g which will start the process g at every point in a trace where f could have terminated successfully, and then maintains the execution of g. In addition, if f had the choice of terminating or continuing, we wish to make sure that f can still continue: in the picture f could have the futures $\underline{\checkmark}$ and T after



executing s. g should start at this point, and f should continue in the future T. We have a situation, in this case, where we need something like the nondeterministic machine accepting the concatenation of two regular sets.

The definition we give is a simple one meeting the above criteria. More complicated definitions are possible.

$$f;g = \{\langle s, W;Y\rangle \mid \langle s\checkmark, W\rangle \in f \wedge \langle \Lambda, Y\rangle \in g\}$$
$$\cup \{\langle st, X\rangle \mid s\checkmark \in \mathrm{dom}(f) \wedge \langle t, X\rangle \in g\}.$$

The first line says that a future for f;g after execution of the trace s can be obtained by 'substituting' Y (initial for g) for occurrences of $\checkmark$ in the future W which f has after s. The second line maintains execution of g once it has started. It also allows g to start autonomously (when $t=\Lambda$).

Example. Let

$$T = \underline{a\overset{\checkmark}{/}b\ \checkmark} \text{ and } T' = \underline{b.c}$$

$$f = \mathrm{proc}(T),\ g = \mathrm{proc}(T')$$

Then f;g assigns the following futures:

$$\Lambda \rightarrow \{\underline{a.b\overset{c}{/}b.c}\} = \{\mathrm{dom}(f;g)\}$$

$$a \rightarrow \{\underline{b.c}, \underline{b.b.c}\}$$

$$ab \rightarrow \{\underline{c}, \underline{bc}\}$$

$$abc \rightarrow \{\cdot\}$$

$$abb \rightarrow \{\underline{c}\}$$

$$abbc \rightarrow \{\cdot\}.$$

3.7 Ordering

It is natural to consider the set inclusion relation on the space of processes, which then becomes a poset. For technical reasons, the reverse inclusion relation

$$f \sqsubseteq g \text{ iff } g \subseteq f$$

will be considered in the rest of the paper. The relation $f \sqsubseteq g$ should be interpreted as "g is more controllable than f" or "f is more autonomous than g".

If $\mathcal{F}$ is an arbitrary nonempty family of processes, then the union $\cup\mathcal{F}$ is a process, which is the greatest lower bound of the family under reverse inclusion. Thus

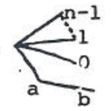
$$\sqcap\mathcal{F} = \cup\mathcal{F}.$$

In case $\mathcal{F}$ is the set of all processes, (denoted PROCESSES(Σ)) we have

$$\sqcap\mathcal{F} = \text{CHAOS}.$$

Unfortunately, suprema do not exist in general in the set PROCESSES(Σ), even for linearly ordered families. This fact is shown by the following example.

Example. Let Σ be the countable alphabet $\mathbb{N} \cup \{a,b\}$. For each $n \in \mathbb{N}$ define the tree

$T_n =$  $= \{\Lambda, a, ab\} \cup \{j | j \le n-1\}$

Let $f_n = \{<\Lambda, T_i> \mid i \ge n\}$
$\cup\{<i, \cdot> \mid i \in \mathbb{N}\}$
$\cup\{<a, b>, <ab, \cdot>\}$

It is easy to check that $\bigcap_n f_n$, which contains the supremum of the f_i's, does not contain Λ in its domain, so the supremum could not be a process.

In order to use the least-fixed-point method to solve recursive equations, it is necessary to establish existence of such limits, in addition to showing that operators are continuous or monotonic in a given partial ordering. We are not able to give a general method for solving recursive equations in the possible-futures model. However, we will find a way to remedy this situation in the next section.

IV. Relations Between Machines and Processes

We show in this section how to collapse the large space PROCESSES(Σ) into one where we can solve recursive equations. We do this by introducing an equivalence relation on processes in such a way that processes are identified if no finite 'test' can distinguish them. We then introduce machines - our version of the processes defined in Hoare, Brookes, and Roscoe - and an onto map

$$M: \text{PROCESSES} \rightarrow \text{MACHINES}$$

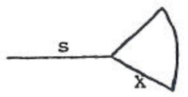
which is such that if f is testably equivalent to g, then M(f) = M(g) and conversely. The map M is shown later to be a homomorphism with respect to the algebra defined in III, and an analogous one on machines directly taken from Hoare, Brookes, and Roscoe.

4.1 The Relation of 'Testable Equivalence'

We can define our equivalence relation directly in terms of processes and the operators already given. The proof that the relation has the desired properties relies on the machine characterization, which we will develop after stating the main theorem.

Our definition is based on the idea of indistinguishability by means of finite tests, and can be viewed as equivalence in all finite environments of a strictly limited kind; a test consists of running a given processes in parallel with a finite test process.

The class of test process we use is called the class of probes. A probe is a controllable process of the form proc($s \rightarrow X$) where X is a finite subset of Σ, $s \in \Sigma^*$, and $s \rightarrow X$ is the tree formed by taking the prefix-closure of the set $s(X \cup \{\Lambda\})$. The tree picture of a probe looks like

 .

We will run probes in parallel with a given process f, and obtain test results by observing the result of intersecting the finite set X at the 'tip' of the probe with the futures that f offers at s. No other information about earlier steps will be used in the test.

We need a preliminary definition: if $s \in \Sigma^*$, and f is a process,

$$f \text{ after } s = \{<t, T> \mid <st, T> \in f\}.$$

Since s is not required to be in dom(f), f after s can be empty and therefore not always a process.

Definition. f is testably equivalent to g, written $f \equiv g$, iff for all probes $s \rightarrow X$,

$$[f \,||\, (s \rightarrow X)] \text{ after } s = [g \,||\, (s \rightarrow X)] \text{ after } s.$$

The "after s" part of this definition corresponds to looking only at the tip X of the probe.

We can now state the main result.

Theorem 4.1

The relation of testable equivalence is a congruence relation on PROCESSES(Σ) with respect to the operations of autonomous choice, conditional choice, guarding, intersection, and sequential composition. If homomorphisms map Σ to Σ, the same holds for inverse images. The induced ordering on equivalence classes

$$[f] \sqsubseteq [g] \text{ iff } [f \sqcap g] = [f]$$

makes the quotient algebra into a complete partial order such that all the induced operations are continuous.

4.2 Refusals and Acceptances

Theorem 4.1 will follow from a concrete representation of the equivalence classes of testable equivalence, in the same way that {0, 1, 2, 3, 4} is a representation of congruence modulo 5 on the integers. Each equivalence class will be represented by a machine, which is a strengthened version of the model proposed by Hoare, Brookes, and Roscoe. The class of machines admits algebraic structure; we generalize the algebraic operations in [HBR] in a straightfoward manner. We then show that testable equivalence is a congruence relation by exhibiting a homomorphism from the "futures" model onto the "machine" model such that testable equivalence is exactly the equivalence determined by this homomorphism. (The analogous situation in the integers is the mapping $i \to i \bmod 5$.)

It turns out that the "collapsed" version of the partial ordering $\subseteq$ from Sec. 3.9 makes the space of machines complete in the sense of denotational semantics. The relevant operators then can be shown to be continuous in this ordering, and so we may apply the fixed-point theory appropriate to showing existence of machines defined recursively.

Definition. Let pΣ denote the set of finite subsets of Σ. A machine is a relation M on $\Sigma^* \times (p\Sigma \times p\Sigma)$ satisfying

(1) dom(M) is nonempty and prefix-closed;

(2) $\langle s, A, R\rangle \in M \Rightarrow A \cap R = \emptyset$;

(3) $\{a \in \Sigma \mid sa \in \text{dom}(M)\} = \cup\{A \mid \exists R.\ \langle s, A, R\rangle \in M\}$; this set is denoted I(M, s).

(4) $\langle s, A, R\rangle \in M$, $B \subseteq A$, $S \subseteq R$ imply $\langle s, B, S\rangle \in M$;

(5) $\langle s, A, R\rangle \in M$, $\sigma \in \Sigma$ imply either $\langle s, A \cup \{\sigma\}, R\rangle \in M$ or $\langle s, A, R \cup \{\sigma\}\rangle \in M$. (Both these conditions could hold.)

The sets A and R such that $\langle s, A, R\rangle \in M$ are called acceptance and refusal sets respectively. We may justify the above definition as follows. Imagine that a machine M can execute the trace s, after which it could act like a machine N. We may formalize this using the notation

$$M \xrightarrow{s} N.$$

This is of course just another way of expressing the state transitions of a nondeterministic machine, with Λ-moves allowed. Now define

$$\text{init}(N) = \{a \in \Sigma \mid \exists P.\ N \xrightarrow{s} P\}.$$

The set init(N) is the set of events in which N could participate on its first step. If $A \subseteq \text{init}(N)$, and $R \subseteq \Sigma - \text{init}(N)$, we say that N may accept A, and refuse R. That is, N may progress on any events from A, and deadlock on events from R.

The reasons for conditions (3), (4) and (5) should now be evident. Since M can evolve into several machines after executing s, and since we want I(M, s) to be the totality of events in which M could participate after executing s, we must have

$$I(M, s) = \cup\{\text{init}(N) \mid M \xrightarrow{s} N\}.$$

Condition (3) states that the union of the A sets must be this same union. The meaning of (4) is easily understood, because $B \subseteq A \subseteq \text{init}(N)$ and $S \subseteq R \subseteq \Sigma - \text{init}(N)$ imply $B \subseteq \text{init}(N)$ and $s \subseteq \Sigma - \text{init}(N)$. Condition (5) follows because if $A \subseteq \text{init}(N)$ and $R \subseteq \Sigma - \text{init}(N)$, then one of $A \cup \{\sigma\}$ or $R \cup \{\sigma\}$ must again have the same property. Since M could evolve to another machine Q, it might be that $\langle s, A, R \cup \{\sigma\}\rangle$ and $\langle s, A \cup \{\sigma\}, R\rangle$ were both allowed triples.

We now identify the nondeterministic state - transition system with the set of triples $\langle s, A, R\rangle$. The system is now specified by a collection of finite entities which represent possible observable aspects of its behavior. We may define

$$\text{dom}(M) = \{s \mid \exists A, R.\ \langle s, A, R\rangle \in M\}$$

and

$$M \text{ after } s = \{\langle t, A, R\rangle \mid \langle st, A, R\rangle \in M\}.$$

4.3 Mapping "futures" to Acceptances and Refusals

Let f be a process. If $\langle s, T\rangle \in f$, then the initial segments of T of length 1 should represent a collection of initial events from some state reachable by s. That is, $T \cap \Sigma$ is a set of the form init(N) as in 4.2. Accordingly, we define

$$M(f) = \{\langle s, A, R\rangle \mid A \subseteq \Sigma, R \subseteq \Sigma, |A \cup R| < \infty, \text{ and } \exists T.\ \langle s, T\rangle \in f, \text{ with } A \subseteq T \text{ and } R \subseteq \Sigma - T\}.$$

The following two theorems are stated without proof.

Theorem 4.3.0

If f is a process, then M(f) is a machine.

Theorem 4.3.1

f is testably equivalent to g <=> M(f) = M(g).

The mapping $\lambda f.\ M(f)$ is going to be the homomorphism mentioned in 4.1. We need to show that M maps onto the space of machines, and that it preserves the operations of 4.1. The latter result is straightforward (once we have defined the relevant operations on machines.) First we establish that M is surjective.

4.4 Mapping Machines to Processes

We show that every machine is M(f) for some process f, thus establishing a 1-1 correspondence between equivalence classes and machines. We do this by defining a map F(N) from machines to processes such that for all N, M(F(N)) = N. The definition of F(N), due to S. Brookes, has independent interest.

Definition. Let N be a machine.

$$\mathrm{Imp}(N) = \{T \in \mathrm{TREES}(\Sigma) \mid M(\mathrm{proc}(T)) \subseteq N\};$$
$$F(N) = \{\langle s, T\rangle \mid T \in \mathrm{Imp}(N \text{ after } s)\}.$$

The set Imp(N) is the set of deterministic processes (thought of as trees) such that their own acceptances and refusals agree with those of N. So these processes are in some sense implementations of N.

By definition of the map M(f),

$T \in \mathrm{Imp}(N) \Leftrightarrow (\forall t \in T)(\forall X \in p\Sigma)$: $\langle t, X \cap (T \text{ after } t), X \cap (\Sigma - (T \text{ after } t))\rangle \in N$.

We must show that F(N) is a process; it turns out that the most difficult job is showing that $F(N) = \emptyset$. This is of course entailed by $\mathrm{Imp}(N) \neq \emptyset$, so we turn out attention to this task.

If $s \in \mathrm{dom}(N)$, then the sets $\langle A, R\rangle$ such that $\langle s, A, R\rangle \in N$ satisfy properties (2), (4), and (5). A collection π of such finite pairs will be called an approximate split of Σ: if $\pi \neq \emptyset$ and $\pi \subseteq p\Sigma \times p\Sigma$, then π must satisfy

(i) $\langle A, R\rangle \in \pi \Rightarrow A \cap R = \emptyset$ (disjointness)

(ii) $\langle A, R\rangle \in \pi \wedge B \subseteq A \wedge S \subseteq R \Rightarrow \langle B, S\rangle \in \pi$ (left-closure)

(iii) $\langle A, R\rangle \in \pi \wedge \sigma \in \Sigma \Rightarrow \langle A \cup \{\sigma\}, R\rangle \in \pi$ or $\langle A, R \cup \{\sigma\}\rangle \in \pi$ (additivity).

Lemma 4.4.1

Let π be an approximate split of Σ, and let $\langle A, R\rangle \in \pi$. Then there is a set $U \subseteq \Sigma$ such that $A \subseteq U$, $R \subseteq \Sigma - U$, and for all finite $X \subseteq \Sigma$,

$$\langle U \cap X, (\Sigma - U) \cap X\rangle \in \pi.$$

The conclusion of this lemma justifies the term "approximate split". If $\langle A, R\rangle \in \pi$, then A is obtained as a finite subset of a set U, and $R \subseteq \Sigma - U$, in such a way that if A' and R' are any other finite subsets of U and $\Sigma - U$, then $\langle A', R'\rangle \in \pi$. Furthermore every $\langle A, R\rangle \in \pi$ determines some U like this; conceivably, different $\langle A, R\rangle$ pairs could determine different U sets.

Proof. Extend the definition of inclusion to pairs of sets by $\langle X, Y\rangle \subseteq \langle Z, W\rangle$ iff $X \subseteq Z$ and $Y \subseteq W$. A pair of (not necessarily finite) sets $\langle X, Y\rangle$ is a full extension of the finite pair $\langle A, R\rangle$ if $\langle A, R\rangle \subseteq \langle X, Y\rangle$ and for all finite $\langle B, S\rangle \subseteq \langle X, Y\rangle$, $\langle B, S\rangle \in \pi$. Let FE(A, R) be the collection of all full extensions of $\langle A, R\rangle$. Since $\langle A, R\rangle \in FE(A, R)$, the collection is nonempty, and is partially ordered by pairwise inclusion. We will apply Zorn's lemma to FE(A, R).

To do this, let C be a subset of FE(A, R) which is a chain: linearly ordered by $\subseteq$. Let $X_c = \cup\{X \mid (\exists Y)\ \langle X, Y\rangle \in C\}$, and $Y_c =$

$\cup\{Y \mid (\exists X)\ \langle X, Y\rangle \in C\}$. Then $\langle X_c, Y_c\rangle \in FE(A, R)$.

Suppose $\langle B, S\rangle \subseteq \langle X_c, Y_c\rangle$. Then $B \subseteq X_c$ and B is finite. Since the X sets in C are increasing, there must be an X_1 such that $B \subseteq X_1$, and a Y_1 such that $\langle X_1, Y_1\rangle \in C$. Similarly there must be an $\langle X_2, Y_2\rangle$ in C such that $S \subseteq Y_2$. Since C is a chain, either $\langle X_1, Y_1\rangle \subseteq \langle X_2, Y_2\rangle$ or the reverse. In the first case, which occurs without loss of generality, we have $B \subseteq X_1 \subseteq X_2$, and $S \subseteq Y_2$; so $\langle B, S\rangle \in \pi$ because $\langle X_2, Y_2\rangle \in FE(A, R)$.

We have shown that every chain in FE(A, R) has an upper bound in FE(A, R).

Let $\langle U, S\rangle$ be the maximal element of FE(A, R) guaranteed by Zorn's lemma. If $\tau \in U \cap S$, then $\langle\{\tau\}, \{\tau\}\rangle \in \pi$, so $U \cap S = \emptyset$. We claim that $S = \Sigma - U$; this will prove the lemma.

Suppose not; then there is an element $\sigma \notin U \cup S$, because $U \cap S = \emptyset$. The pair $\langle U \cup \{\sigma\}, S\rangle$ properly extends $\langle U, S\rangle$, so cannot be in FE(A, R) by maximality of $\langle U, S\rangle$. This implies the existence of a finite pair of the form $\langle A_1 \cup \{\sigma\}, R_1\rangle \notin \pi$, where $A_1 \subseteq U$ and $R_1 \subseteq S$.

Similarly, $\langle U, S \cup \{\sigma\}\rangle$ is not maximal, so there is a pair $\langle A_2, R_2 \cup \{\sigma\}\rangle \notin \pi$. Now the pair $\langle A_1 \cup A_2, R_1 \cup R_2\rangle$ is finite and included in $\langle U, S\rangle$, so must be in π because $\langle U, S\rangle \in FE(A, R)$. By additivity, $\langle A_1 \cup A_2 \cup \{\sigma\}, R_1 \cup R_2\rangle \in \pi$, or else $\langle A_1 \cup A_2, R_1 \cup R_2 \cup \{\sigma\}\rangle \in \pi$. But by left closure, either of these two cases gives a contradiction.

The set U of symbols given by 4.4.1 can now be used in the construction of Imp(N). It will form the initial events in a deterministic process implementing N.

Lemma 4.4.2. Let N be a machine and $\langle \Lambda, A, R\rangle \in N$. There is a $T \in \mathrm{Imp}(N)$ such that $A \subseteq T$ and $R \subseteq \Sigma - T$.

The proof constructs the desired tree in inductive stages, using the sets guaranteed by 4.4.1.

Corollary 4.4.3. Let N be a machine, and $<s, A, R> \in N$. Then $T \in Imp(N \text{ after } s)$ with $A \subseteq T$, and $R \subseteq \Sigma - T$.

Proof. N after s is again a machine; apply 4.4.2 to N after s.

Theorem 4.4.4. If N is a machine, then F(N) is a process.

Proof

By definition, $dom(F(N)) \subseteq dom(N)$. By 4.4.3, the reverse inclusion holds. Since $dom(N) \neq \emptyset$, we have $F(N) \neq \emptyset$. It remains to show consistency and persistence.

Suppose $<s\sigma, U> \in F(N)$. Then $s\sigma \in dom(N)$ and $s \in dom(N)$ as well. By condition(3) in the definition of machines, there is an $<A, R>$ with $\sigma \in A$ and $<s, A, R> \in N$. By Cor. 4.4.3, there is a T' with $<s, T'> \in F(N)$, $A \subseteq T'$, and $R \subseteq \Sigma - T'$. Let $T = (T' - \sigma \cdot \Sigma^*) \cup \sigma U$. Then $T \cup \Sigma = T' \cap \Sigma$; if $\tau \neq \sigma$, then T after τt = T' after τt; and T after σu = U after u. These facts imply $T \in Imp(N \text{ after } s)$, so $<s, T> \in F(N)$, and by construction $\sigma U \subseteq T$.

Persistence is easy: $T \in Imp(N \text{ after } s)$ and $t \in T$ imply T after $t \in Imp(N \text{ after } st)$.

We are now ready for the last theorem of this section.

Theorem 4.4.5. For all N, $M(F(N)) = N$.

Proof. We already know $dom(F(N)) = dom(N)$, so $dom(M(F(N))) = dom(N)$ as well. Let $<s, A, R> \in M(F(N))$. Then $\exists X \in p\Sigma$ and T such that $<s, T> \in F(N)$ and $A = T \cap X$, $R = (\Sigma - T) \cap X$. But $<s, (T \text{ after } \Lambda) \cap X, (\Sigma - (T \text{ after } \Lambda) \cap X> \in N$ by definition of F(N), so $<s, A, R> \in N$. For the other inclusion, let $<s, A, R> \in N$. By 4.4.3, $\exists T$ such that $<s, T> \in F(N)$ and $A \subseteq T$, $R \subseteq \Sigma - T$. Let $X = A \cup R$. Then $<s, A, R> = <s, T \cap X, (\Sigma - T) \cap X> \in M(F(N))$.

Corollary. The map $f \rightarrow M(f)$ is onto, and the map $N \rightarrow F(N)$ is one-to-one.

4.4.5 Algebra of Machines

The space of machines over Σ admits an algebraic structure similar to the space of processes. Here we show that it is in fact a homomorphic image with respect to a reasonably large set of operations. We have chosen the operations defined in Hoare, Brookes, and Roscoe and have strengthened their definitions to handle acceptances as well as refusals. We have not, however, treated the hiding operator in this section. In the companion paper we will show a continuity result, but the homomorphic property does not hold of this operator.

Definition 4.5.

(1) $STOP = \{<\Lambda, \emptyset, R> \mid R \in p\Sigma\}$
STOP is the machine which can accept nothing and must refuse everything.

(2) $RUN = \{<s, A, \emptyset> \mid s \in \Sigma^*, A \in p\Sigma\}$
RUN can refuse nothing and must accept everything.

(3) $CHAOS = \{<s, A, R> \mid s \in \Sigma^*, A \cap R = \emptyset\}$
CHAOS may make any choice whatsoever.

(4) Autonomous choice.
$M \sqcap N = M \cup N$.

(5) Controllable choice.
$M \,\Box\, N =$
$\{<\Lambda, A \cup B, R \cap S> \mid <\Lambda, A, R> \in M, <\Lambda, B, S> \in N\}$
$\cup \{<s, X, Y> \mid s \neq \Lambda \wedge <s, X, Y> \in M \sqcap N\}$

(6) Guarding.
$a \rightarrow M = \{<\Lambda, \{a\}, R - \{a\} \mid R \in p\Sigma\}$
$\cup \{<ax, A, R> \mid <x, A, R> \in M\}$

(7) Synchronized parallelism.
$M \,||\, N = \{<s, A \cap B, R \cup S> \mid <s, A, R> \in M$ and $<s, B, S> \in N\}$

(8) Sequential composition.
$M:N = \{<s, A, R> \mid <s, A, R \cup \{\checkmark\}> \in M$
$\cup \{<s, A \cup B, R \cap X> \mid \checkmark \notin A, <s, A \cup \{\checkmark\}, R> \in M,$ and $<\Lambda, B, X> \in N\}$
$\cup \{<st, B, X> \mid s\checkmark \in dom(M) \wedge (t, B, X) \in N\}$

(9) Inverse renaming. Let $h: \Sigma \rightarrow \Sigma$.

$h^{-1}[M] = \{<s, A, R> \,||\, A \cup R| < \infty$ and $<h(s), h[A], h[R]> \in M\}$

The mapping $f \rightarrow M(f)$ from processes to machines provides the required homomorphism. Since this map is surjective, establishing its homomorphic properties also shows that the space of machines is closed under operations (4-9).

Theorem 4.5.1.

(1) $M(STOP) = STOP$

(2) $M(RUN) = RUN$

(3) $M(CHAOS) = CHAOS$

(4) $M(f \sqcap g) = M(f) \sqcap M(g)$

(5) $M(f \,\Box\, g) = M(f) \,\Box\, M(g)$

(6) $M(a \rightarrow f) = a \rightarrow M(f)$

(7) $M(f \,||\, g) = M(f) \,||\, M(g)$

(8) $M(f;g) = M(f); M(g)$

(9) $M(h^{-1}f) = h^{-1}(M(f))$.

The proof is straightforward, except for (8), and is omitted here.

4.6 Continuity

We close the paper with a discussion of the limit properties of machines and continuity of various operators. This section completes the proof of Theorem 4.1 as well, since the ordering defined on machines is derived from the same one on processes.

Definition: $M \sqsubseteq N$ iff $N \subseteq M$.

The 'reverse' ordering makes the space of machines into a complete partial order with bottom element CHAOS, as we now check.

Clearly CHAOS $\sqsubseteq M$ for any M. If $\mathfrak{F}$ is a non-empty set of machines, then $\bigcup \mathfrak{F}$ is a machine which, under reverse ordering, becomes an infimum for the set. We must show that if $<M_i>$ is a chain with respect to $\sqsubseteq$, then $\bigcap \{M_i \mid i \in I\}$ is a machine, which will then be a least upper bound.

Theorem 4.6.0. If $M_{i+1} \sqsubseteq M_i$ for i = 1, 2... then $\bigcap_{i \in \mathbb{N}} M_i$ is a machine.

Proof. This theorem is straightforward except for the additivity condition, so we include the proof that the intersection machine has this property.

Let $<s, A, R> \in M_i$ for every i, and let $\sigma \in \Sigma$ be arbitrary. For each i, either $<s, A \cup \{\sigma\}, R> \in M_i$ or $<s, A, R \cup \{\sigma\}> \in M_i$. As $i \to \infty$, one of these two possibilities occurs infinitely often. Suppose without loss the first occurs. Then $<s, A \cup \{\sigma\}, R> \in M_i$ for infinitely many i. But $M_i \supseteq M_j$ if $i \leq j$. Therefore $<s, A \cup \{\sigma\}, R> \in M_i$ for all i, which is what we needed.

Theorem 4.6.1. The operations autonomous choice, controllable choice, guarding, parallel composition, sequential composition, and inverse renaming are continuous in all their arguments.

Proof. We show the result for parallel composition. We are to show first: if M_i is a decreasing sequence of machines (in the subset ordering then

$$\bigcap_i (M_i \parallel N) = (\bigcap_i M_i) \parallel N.$$

The inclusion from riqht to left is again easy, so suppose $<s, A, R> \in \bigcap_i (M_i \parallel N)$. Then for each i $<s, A, R> = <s, A, R_1^i \cup R_2^i>$ where $<s, A, R_1^i> \in M_i$ and $<s, A, R_2^i> \in N$. Now $R = R_1^i \cup R_2^i$ is a finite set, so among the pairs $<R_1^i, R_2^i>_{i \in \mathbb{N}}$, one must occur infinitely often, and so $\exists <s, A, R_1> \in M_i$ for each i, and $<s, A, R_2> \in N$ with $R = R_1 \cup R_2$, by the chain property. Therefore $<s, A, R> \in \bigcap M_i \parallel N$.

Our proof of 4.1 can now be said to be complete, and we have established all the connections promised between the two models. We have not, however, investigated all the possible operators on processes and machines. nor have we shown how to connect the HBR model itself to ours. This latter job is not difficult, and we plan to do it in a companion note. The other investigations will be reported in future papers.

Acknowledgements

Professor Tony Hoare was directly responsible for making this paper possible. The PF model represents Rounds' efforts to understand his latest ideas. A special debt is owed to Jim Bodwin, who suggested the elegant additivity condition on machines. The Programming Research Group at Oxford was the site of much of this work, and Rounds found conversations with Bill Roscoe, Cliff Jones, Gordon Plotkin, and Robin Milner very helpful. Finally, credit is due to Roxianne Carbary for excellent typing.

References

[BR] Brookes, S. and Rounds, W. Refusals and possible futures (in preparation).

[CKS] Chandra, A., Kozen, D., and Stockmeyer, L. Alternation. JACM 28, 1, January 1981.

[FHLD] Francez, N., Hoare, C.A.R., Lehmann, D.J., and de Roever, W.P. Semantics of nondeterminism, concurrency, and communication. JCSS 19 (1979).

[FLP] Francez, N., Lehmann, D., and Pnueli, A. A linear history semantics for distributed languages. Proc. 21st Focs, 1980.

[H] Hoare, C.A.R. Communicating sequential processes. Comm. ACM. 21, 8, August 1978.

[HBR] Hoare, C.A.R., Brookes, S., and Roscoe, A.W. A mathematical model of communicating processes. T. Mon. PRG-20, Programming Research Group, Oxford University, 1981.

[K] Keller, R.M. Formal verification of parallel programs. CACM 19, 7, July 1976.

[M] Milner, R. A calculus of communicating systems. Lecture Notes in CS, Springer, 1981.