A MATHEMATICAL THEORY OF COMMUNICATING PROCESSES

by A.W. ROSCOE,

ST. EDMUND HALL

Thesis submitted for the degree of D.Phil., Trinity 1982

Contents

Introduction

This thesis is an examination of part of the mathematical theory of communicating processes. These are studied through the medium of C.A.R.Hoare's language C.S.P. (Communicating Sequential Processes). Our main theme is the use of mathematical models to produce correctness proofs for processes, though several mathematical sidelines are also developed. The first seven chapters are broadly concerned with the construction and use of two mathematical models for C.S.P., these being the well-known "traces" model and a related model which is capable of dealing with non-deterministic processes in an adequate way.

In chapter one we meet the version of C.S.P. which is used throughout the thesis. We also meet the first of the two models; this is a model which can adequately cope only with deterministic processes. It has the advantage however that its simple structure allows us to develop the techniques which we later apply to the more general model. We define various operators over the model which represent ways of constructing and combining processes. These are used to produce a formal semantics for the language. Various results are quoted which show how one can to a large extent ignore the distinction between processes defined by the formal semantics and objects defined purely by operators on the model.

Chapter two is an investigation of a class of proof rules which can be applied to the model. These are all derived from the basic notion of "recursion induction". We find that there several different pairs of conditions, which if satisfied by recursive definitions and the predicates we wish to prove of them, guarantee the validity of the proof rules we devize.

Chapter three is a digression into topology. We study the spaces of allowable predicates generated by the previous chapter and the topologies which these induce on the space of processes. We are able to explain many of the results of chapter two and to generalize several of them.

In chapter four we meet the second and more general model for communicating processes, which is used throughout chapters five, six and seven. This is able to represent non-

deterministic processes in a fairly convincing way. This chapter is not an extensive introduction to this model, something which can be found elsewhere ( ),( ), but is rather an examination of two questions which arise from the mathematical foundations of the model. The second of these is especially interesting, since we discover that the well-definedness of our model is a powerful set-theoretic tool, equivalent to the compactness theorem for arbitrary propositional languages.

In chapter five we see how the work of chapter two can be transferred to the non-deterministic model with only a few alterations. We see that many total correctness problems translate naturally into the system we have developed. As an example we take the predicate "is a buffer" and develop a calculus for proving it true of processes. We begin our study of operators defined by parallelism and hiding by discovering the close connections between a "pipe" operator and the space of buffers. Recursions via the pipe operator are investigated and we are able to prove several of them to be correct (buffers).

In chapter six we deal with an operator which models the behaviour of a process operating in parallel with a second, "slave" process, all communications between the two being hidden. We discover by means of examples that this operator is a very useful recursive tool for describing potentially infinite trees of processes. We develop a calculus for proving correct processes which are recursively defined using it. We discover an unfortunate possible type of behaviour in infinite networks and discover methods by which we can prove it absent.

Chapter seven deals with other parallel/hiding combinators in less detail, and as an example shows how to construct a rectangular array of processes. It also deals with the problem of proving a network free from deadlock. We find that hiding has little to do with deadlock, and can often be ignored when studying it. Deadlock is analysed in cases where it is known to be absent from all small portions of a network. This work is shown to have special significance in cases where networks are constructed without loops,

though we are also able to develop various methods for applying it to more general networks.

As a result of the work done in chapters 1-7 we are equipped with a powerful calculus for proving properties of the values assigned to programs in our mathematical models. What the eighth and final chapter seeks to do is to provide a framework for the application of this calculus to the systems which we are attempting to model. We therefore develop a calculus for relating models to the systems which they attempt to model. We see how predicates transfer between systems, and how operators over a model may be held to reliably reflect their implementations in the real world. We construct a plausible though abstract space of "real" processes by which to judge our models for C.S.P. We find that there are several different ways in which we might interpret each of these. We find that in several interpretations some of our definitions of operators do not appear to be readily implementable. Finally we see with the benefit of hindsight what alterations might be made to our models and operators to make them have as many desirable properties as possible.

Much of the mathematics used in this thesis is rather abstract, especially in chapters three and four. The author hopes however that its use is justified by the results achieved, and that it does not obscure too much the techniques developed for the proof of programs.

Acknowledgements

The author would like to thank each of the following, who have all made large or small contributions to this work, either by suggestions or through their lectures.

C.A.R. Hoare
S.D. Brookes
D.S. Scott
J.E. Stoy
P.J. Collins
R. Milner
D.M.R. Park
C.B. Jones
W. Rounds
S.R. Blamey

In addition he would like to thank Miss J. Scheenen for reading the text and pointing out his many typing mistakes.

Finally he would like to thank the Science Research Council and St. Edmund Hall, Oxford for their financial support during the preparation of this thesis.

Chapter 1 :-

An Introduction to CSP and its Deterministic Model

This chapter outlines the version of CSP which is used in this thesis. We also meet the model for deterministic processes upon which chapters 2 and 3 are based, and use it to give a semantics for CSP. This chapter is to a large extent a summary of some of the work of C.A.R.Hoare, in whose papers (e.g. (1),(2)) the reader can find further explanation and motivation for CSP as well as many simple illustrative examples.

A process is to be thought of as an entity which communicates with its environment (possibly other processes) in some alphabet Σ of atomic communications or "events". Communication can take place only with the co-operation of all participants, and is symmetric in that there is no "sender" and no "receiver" of an event. At each point in the history of a process there is clearly a sequence of elements of Σ which is what it has communicated with its environment; this is known as a trace of the process.

We will assume that Σ contains basic symbols such as "a", or "true", or "isempty" and compound or named symbols such as "a.true", "b.a", "?a" or "!a" (we conventionally drop the infix dot with the names "?" and "!" which will usually represent input and output respectively). In addition Σ contains a special symbol "$\surd$" which a process communicates to indicate successful termination, and for which $a.\surd = \surd$.

We denote the set of all strings of symbols by Σ^*. The empty string (containing no symbols) we denote by "$\langle\rangle$". The string containing symbols a,b,...,z (in that order) we denote by $\langle ab...z\rangle$ and the concatenation of strings v & w we denote by vw.
Usually a,b,c,... will represent elements of Σ; v,w,s,t,... elements of Σ^* and x,y,z,... variable elements of Σ. A,B,C,.. will be processes and S,T,.. subsets of Σ. Σ^- will denote $\Sigma - \{\surd\}$. X^* denotes the set of finite strings of elements of X.

Let us suppose that A is a process. We will postulate that at each point in A's history there is a set X of events in which it is willing to co-operate, and will do so (picking one at random) if the environment is also able to execute any of the elements of X. This set will vary with time dependent on both the visible actions of A (namely its communications) and any internal progress which it makes. It is clearly possible that, depending on internal choices which are invisible to the environment, A might behave in one of several different ways after the same visible behaviour. Processes which behave in this way are known as non-deterministic. We will later (in chapter 4) meet models which are sufficiently expressive to deal with this type of behaviour, but for the moment we will only attempt to deal with deterministic processes. The criterion for A to be deterministic is that there be some function F from tr(A) (the traces of A, $\subseteq \Sigma^*$) to $\wp(\Sigma)$ such that for every $s \in tr(A)$ the following two conditions hold.

a) Whenever s is the trace of A then $X \subseteq F(s)$.

b) Whenever s remains the trace indefinitely then for each $a \in F(s)$ & time t there is some $t' > t$ at which $a \in X$.

This means that whenever $s \in tr(A)$ then a sufficiently patient experimenter will be able to achieve it, so long as his current trace is some prefix of s.

It is possible to define a partial order on Σ^* by $v \leqslant w$ if $\exists s.\ w = vs$, or equivalently if v is a prefix of w. If $w \neq v$ we say $v < w$.

The model we choose for deterministic processes is sets of traces (subject to the conditions below), intending to identify a deterministic process with its set of traces. For the moment denote by $\bar{A}$ the representation of A in the model. Define $B \subseteq \Sigma^*$ to be an element of P, the space of deterministic processes, if it satisfies the conditions:

1.1 a) B is non-empty

b) B is prefix closed (i.e. $w \in B \ \& \ v < w \ \Rightarrow \ v \in B$)

c) $w \langle\!\surd\!\rangle v \in B \Rightarrow v = \langle\rangle$

Every "real" process A satisfies these conditions, since:
a) A can at least do nothing (so $\langle\rangle \in \bar{A}$).
b) If $v < w$ and A has performed trace w there must have been some earlier time when it had performed trace v.
c) A cannot do anything after it has terminated.

The strength of this model applied to deterministic processes is that, given a deterministic process A such that $s \in \bar{A}$, then the environment, by applying any set Y s.t. $Y \cap \{ a \mid s\langle a\rangle \in \bar{A} \} \neq \emptyset$ to A (when its trace is s), can be sure of eventually getting some response (in Y). This allows us to express many notions of "total correctness" of a process solely by reference to its value in the model P. For example define deadlock as the state that a process is in when it (a) has not terminated successfully and (b) will never be able to communicate with its environment again. We can express the predicate "is free from deadlock", meaning that deadlock can never arise in a process, in terms of the model:

$$D\text{-}F(A) \equiv (s \in \bar{A} \Rightarrow ((\exists t.\ s = t\langle\surd\rangle) \vee (\exists a.\ s\langle a\rangle \in \bar{A}))) .$$

From now on we will largely be studying the model P in itself, rather than its relation to "real" processes. Thus from now on A,B,C,... will denote elements of P.

1.2 Lemma
a) With the set inclusion order "$\subseteq$" P is a complete lattice with minimal element $\{\langle\rangle\}$ and maximal element $\{w, w\langle\surd\rangle \mid w \in (\Sigma^-)^*\}$.
b) The finite elements of P (in the lattice-theoretic sense) are just the processes with finitely many elements.

The proof of a) follows immediately from the fact that if $\emptyset \neq X \subseteq P$ then $\bigcup X \in P$ and $\bigcap X \in P$. The proof of b) follows from the fact that if $A \in P$ and $F \subseteq A$ is finite (not necessarily $\in P$) then there is some finite size $B \in P$ s.t. $F \subseteq B \subseteq A$.

We will now start to introduce the language and its interpretation (semantics) within P. We first meet a few basic processes:

1.3 abort $= \{\langle\rangle\}$ the process which can do nothing

1.4 skip $= \{\langle\rangle, \langle\surd\rangle\}$ the process which immediately terminates successfully.

1.5 $\underline{\text{run}} = \{w, w\langle\surd\rangle \mid w \in (\Sigma^{-})^{*}\}$, the process which has the ability to do anything at all.

Note that <u>abort</u> corresponds to the minimal element of P, and <u>run</u> to the maximal one. Recall that our interpretation of elements of P is that the environment (external or another process) is at each stage given a choice of what it would like to do next. This choice, on the first step, is between the <u>initials</u> of a process, those elements of which are possible on the first step. We write this set as $A^{o} = \{a \in \Sigma \mid \langle a \rangle \in A\}$. Note that the strength of a process, in terms of the partial order, is measured by the amount of choice it offers.

For any $w \in A$ ($A \in P$) we can define the process which A becomes immediately after it has executed the trace w.

1.6 A <u>after</u> $w = \{v \mid wv \in A\}$

Note that for any process $A \in P$ we have A <u>after</u> $\langle\rangle$ = A, and <u>run after</u> w = <u>run</u> for every $w \in (\Sigma^{-})^{*}$.

Before giving a formal semantics for the rest of our language, we will give an informal explanation and motivation for each of its constructs. These constructs take the form of operators which combine one or more processes together to form more complex ones.

By "a → A" we will mean the process which communicates "a" on its first step and then acts like A. This construct will be the most basic building block of our language. Along the same lines "x:T → A" will represent the process which inputs some value ("b" say) from the set T of unnamed symbols, and substitutes this value for the variable "x" within A. "a.x:T → A" will be the same except that we will expect its input to be named by "a".

The process "a.A" will be the same as A except that all events are given the (possibly additional) name "a".

The process "A $\Box$ B" will give the environment the choice of all the communications offered by A and B.

The sequential composition of A and B, in which B takes over whenever A terminates successfully, is written A;B.

By $(A_X||_Y B)$ we will mean the process which consists of A working in parallel with B. All communications of A

will be in X, all those of B in Y and all communications which lie in $X \cap Y$ must be co-operated in (executed simultaneously) by both A and B.

By "A/X" we will mean the process formed by hiding all occurrences of elements of the set X from the environment, making them into internal actions which happen without the control of the environment.

Finally we will include recursion, for which we include variables representing processes in our syntax, on the understanding that all occurrences of these will be bound by some recursion.

We will now define some of these operators formally.

1.7 $a \to A = \{\langle\rangle\} \cup \{\langle a \rangle w \mid w \in A\}$ (for $a \in \Sigma^-$, $A \in P$)

1.8 $A \,[]\, B = A \cup B$ (for $A, B \in P$)

1.9 $A;B = (A \cap (\Sigma^-)^*) \cup \{wv \mid w\langle\surd\rangle \in A \ \&\ v \in B\}$

1.10 $(A_X \|_Y B) = \{w \mid (w \restriction (X \cup Y) = w) \ \&\ w \restriction X \in A \ \&\ w \restriction Y \in B\} \cap \{w, w\langle\surd\rangle \mid w \in (\Sigma^-)^*\}$
where $A, B \in P$, $X, Y \subseteq \Sigma$
the restriction operator $\restriction X$ is defined
$\langle\rangle \restriction X = \langle\rangle$, $\langle a \rangle w \restriction X = w \restriction X$ if $a \notin X$
$= \langle a \rangle (w \restriction X)$ if $a \in X$

1.11 $A/X = \{w \restriction (\Sigma - X) \mid w \in A\}$

1.12 $a.A = \{a.w \mid w \in A\}$
where the operator "a." is defined on Σ^* by
$a.\langle\rangle = \langle\rangle$, $a.(\langle c \rangle w) = \langle a.c \rangle (a.w)$

We will not be able to define the operators involving variables properly until later. The following is a list of easily proved identities involving the above.

1.13 Theorem

a) $A \,[]\, B = B \,[]\, A$ ([] is commutative)

b) $A \,[]\, (B \,[]\, C) = (A \,[]\, B) \,[]\, C$ ([] is associative)

c) $(A;B);C = A;(B;C)$ (; is associative)

d) $(A \,[]\, B);C = (A;C) \,[]\, (B;C)$ (; distributes to the left over [])

e) $A;(B \,[]\, C) = (A;B) \,[]\, (A;C)$ (; distributes to the right over [])

f) $(a \to B);C = a \to (B;C)$

g) $(A_X \|_Y B) = (B_Y \|_X A)$ (|| is commutative)

h) $(A_X \|_{Y \cup Z} (B_Y \|_Z C)) = ((A_X \|_Y B)_{X \cup Y} \|_Z C)$ (|| is associative)

i) $A \,[]\, A = A$ ([] is idempotent)

j) $(A_X \| _X A) = A$ provided $A \subseteq X^*$

k) $(a \rightarrow A)/X = A/X$ if $a \in X$

$= a \rightarrow A/X$ if $a \notin X$

l) $(A/X_Y \| _Z B) = (A_{X \cup Y} \| _Z B)/X$ provided $X \cap Z = \emptyset$

m) $(A;B)/X = (A/X);(B/X)$ provided $\surd \notin X$

These identities will be used frequently and informally throughout the rest of this chapter and in chapters 2 and 3.

1.14 Theorem

Each of the operators defined in 1.7 - 1.12 represents a monotonic and continuous function of P or P × P. This function is also reverse continuous except in the case /X when X is infinite. (Reverse continuous means continuous on the lattice with reverse order.)

To prove monotonicity it is sufficient to show (for each operator f) that it is monotonic in each parameter.

To prove continuity it is sufficient to show that each operator represents a continuous function of each of its parameters. That is, $f(\bigcup D) = \bigcup_{X \in D} f(X)$ for each nonempty directed set $D \subseteq P$. A function is thus reverse continuous if it satisfies $f(\bigcap D) = \bigcap_{X \in D} f(X)$ for each nonempty reverse directed set $D \subseteq P$.

Each of the cases of the theorem is fairly easily checked. The failure of reverse continuity for infinite hiding is illustrated by the following example.

1.15 Example

We will suppose $a \notin N$ (the natural numbers) and that $\{a\} \cup N \subseteq \Sigma$. For $n \in N$ define the process $A_n = \{\langle\rangle, \langle m\rangle, \langle ma\rangle \mid m \geq n\}$. It is easy to see that $D = \{A_n \mid n \in N\}$ is a reverse directed set with $\bigcap D =$ abort but that $A_n/N = a \rightarrow$ abort for each $n \in N$. We thus have $\bigcap_{A \in D} (A/N) = a \rightarrow$ abort $\neq (\bigcap D)/N =$ abort.

The difficult properties of infinite hiding will appear again in a much more fundamental way when we consider the non-deterministic model in chapter 4. The advantages of having operators which are reverse continuous will be seen in 2.46 et seq.

The monotonicity and continuity of the operators will be very important in defining recursion.

It is necessary here to strike a note of caution. We have happily given definitions in the deterministic model P to each of the operators we have described informally. Inspection of each of the definitions 1.7 - 1.12 will show that these reflect to the greatest possible extent the intentions we set out, in that the resulting sets of traces seem to be the best possible. This still leaves the question of whether it is reasonable to expect an implementor to produce deterministic processes as the result of applying each of the operators to deterministic processes. There are in fact several cases where this does not seem reasonable, in that the "natural" result of an operator is not in general deterministic.

The first and most obvious of these cases is with the hiding operator "/X". Consider for example the process written "$(a \rightarrow b \rightarrow \underline{skip}) \,\Box\, (c \rightarrow d \rightarrow \underline{skip})/\{a,c\}$" (= A, say). (The elements a,b,c,d of Σ are assumed to be distinct.) It is easy to prove the relation $A = (b \rightarrow \underline{skip}) \,\Box\, (d \rightarrow \underline{skip})$. This latter process is one which is invariably willing (given time) to execute either "b" or "d" on its first step, at the environment's choice. It is thus necessary in a correct implementation of the syntax of A that we should take account of its behaviour after <u>both</u> "hidden a" and "hidden c". This varies from what one might consider natural, namely that the process would either carry out hidden "a" or "c" but not both and that the actions of the process would then be independent of the consequences of executing the other one. To implement the hiding operator correctly with respect to definition 1.11 and produce a deterministic process it would in general be necessary to carry out an infinite amount of backtracking. The three examples below help to illustrate other aspects of this problem.

a) $(a \rightarrow b \rightarrow \underline{skip}) \,\Box\, (c \rightarrow \underline{abort})/\{a,c\} = b \rightarrow \underline{skip}$

b) $(a \rightarrow b \rightarrow \underline{abort}) \,\Box\, (b \rightarrow \underline{skip})/\{a\} = b \rightarrow \underline{skip}$

c) $A \,\Box\, (b \rightarrow \underline{skip})/\{a\} = b \rightarrow \underline{skip}$
where $A = \{\langle\rangle, \langle a\rangle, \langle aa\rangle, \langle aaa\rangle, \langle aaaa\rangle, \ldots\}$

The following three features of a process A give rise to implementation problems in A/X.

a) The choice at any stage between several elements of X. (Illustrated in (a) above and also in the example in the text.)

b) The choice between elements and non-elements of X at any stage. (Illustrated in (b) above.)

c) The possibility of infinite sequences of elements of X. (Illustrated in (c) above.)

While we are using the deterministic model we will only use the hiding operator in writing processes where these three conditions are avoided, so that our processes are reasonable to implement deterministically.

Another operator which gives rise to similar problems is the alternative composition operator "$\Box$". This operator is intended to give the environment the choice of the first step symbols of the two processes, the combination then acting like the chosen process. The problem arises when the two processes have first step choices in common. The correct operation of the definition 1.8 would require the operation in parallel of the two processes until any time when one of the processes cannot continue with the executed trace. (An alternative is to back-track when one of the processes is unable to continue.) This is clearly inefficient, for there is the possibility of an exponential growth in effort with the number of "$\Box$"s encountered. There is also the problem (which arises from both the suggested methods of implementation) of detecting just when a process cannot execute a symbol. This would almost certainly be tantamount to solving the halting problem; and in any case it is possible to find processes which do nothing for any given finite time before finally becoming able to communicate (e.g. a process with a very large number of hidden symbols to execute at the start). It would in fact probably be possible to implement "$\Box$" deterministically by combining the two approaches above, running the two processes alternately and trying to make the one (if either) which falls behind catch up. This is unfortunately even less efficient than was expected for the implementation above, for there is now the <u>certainty</u> of exponential growth. Because of these difficulties we must make it a rule that in writing expressions which

we expect to be implementable we should only use "□" when we can be certain that the initials of the two processes which we are combining are disjoint. This can be achieved (for example) by ensuring that the two processes have distinct guards, so that the combination looks like $(a \rightarrow A) \,\Box\, (b \rightarrow B)$ where $a \neq b$.

The last problem of determinacy arises from sequential composition (;). It occurs when it is possible for the first process either to terminate successfully or to do something else ("a", say). If the second process has the ability to communicate "a" on its first step then we have a problem of a very similar nature to the one which arose with "□". To implement 1.9 correctly it would be necessary to run both processes in parallel until it could be detected that only the first or second process has the ability to continue with the trace. This problem of ambiguity gives rise to exactly the same difficulties as arose with "□" above. The rule which avoids this problem is to avoid creating processes which make successful termination anything other than the sole option at any point in their history at which it is possible.

The following illustrate the possible difficulties with the implementation of "□" and ";".

a) $(a \rightarrow A) \,\Box\, (a \rightarrow B)$

Here it is necessary to determine the result of executing both the left and right "a"s to correctly implement this process.

b) $(\underline{skip} \,\Box\, (a \rightarrow b \rightarrow \underline{skip})) ; (a \rightarrow b \rightarrow a \rightarrow b \rightarrow \underline{skip})$

There are two ways in which this process can execute any of the traces $\langle a \rangle$, $\langle ab \rangle$, $\langle aba \rangle$, $\langle abab \rangle$. If the second process were prepared to carry out an infinite sequence of "a"s and "b"s this conflict would never be resolved.

In 1.19 we will meet a more interesting example of a process written without regard for these rules. (For its construction we will need recursion.)

We will find that the model introduced in chapter 4 gives more natural semantics to each of the above operators, avoiding the necessity of imposing strict conditions upon a

process to ensure that implementation is possible. Indeed we will find that the only points at which the semantics in the two models differ fundamentally are those where the difficulties described above occur, and also in the case of ill-defined recursion (which we have yet to meet). This is true in the sense that (except in these cases) if we combine the representations in the more advanced model of one or more deterministic processes the result will be a deterministic process congruent to the result we would have obtained in the deterministic model P.

We must now introduce the various operators involving the use of variables representing processes and elements of Σ. Unfortunately the informal approach we have adopted so far, namely the introduction of our language purely as a collection of operators on P, seems to become inadequate here. We will thus recast our previous definitions and introduce the new ones within the framework of a formal syntax and semantics for C.S.P.

We first meet the various syntactic domains we will need.

1.16 Syntactic Domains

$A \in Exp$	(the C.S.P. expressions)
$a \in \Sigma^-$	(alphabet)
$B \in PV$	(process variables)
$B' \in PV'$	(process variable calls)
$x \in AV$	(alphabet variables)
$T \in IA$	(expressions for sets of unnamed symbols)
$X \in GA$	(expressions for sets of general symbols)
$U \in BA$	(basic subsets of Σ)

$A ::= \underline{skip} \mid \underline{abort} \mid a \to A \mid x \to A \mid a.x \to A \mid x{:}T \to A \mid a.x{:}T \to A \mid$

$A \,[]\, A \mid A;A \mid (A_X \|_X A) \mid A/X \mid a.A \mid B' \mid$

$rec_i(B_1,\ldots,B_k).(A_1,\ldots,A_k) \mid$ "infinite mutual recursion"

(the syntax for infinite mutual recursion will be found later, in 1.18)

$B' ::= B \mid B_g$ (explanation of B_g will be found in the definition of infinite mutual recursion)

$T ::= \emptyset \mid \{a\} \mid \{x\} \mid U \mid T \cup T \mid T \cap T \mid T - T \mid \ldots$

$X ::= \emptyset \mid \{a\} \mid \{x\} \mid U \mid X \cup X \mid X \cap X \mid X - X \mid a.X \mid \ldots$

We will assume that we are initially endowed with the sets Σ, PV, AV and BA and that they satisfy the condition that all the objects created from the above syntax are distinct if they have distinct parse trees.

We can only expect elements of Exp to represent elements of P if they do not depend on the value of any variable. (One would expect such expressions to represent functions of such variables.) One might expect an expression to depend on the value of a variable only if there were some occurrence of it within the expression which was not bound to some construct which assigned it a value. For example we would expect "a → x → skip" to depend on "x" but not "x:T → x → skip". It is easy to construct formal definitions of the notions of free and bound variables in an expression. Informally an occurrence of process variable B will be bound in A ∈ Exp if and only if there is a syntactic subcomponent of A which contains the occurrence and has the form $rec_j(...B...).(A_1,...,A_k)$ or the equivalent in infinite mutual recursion. Alphabet variables will be bound by the constructs "x:T → A" and "a.x:T → A".

1.17 Definition

a) Alphabet variables

Suppose x ∈ AV. It is easy to construct a formal recursive definition of the phrase "x occurs in X" (for X ∈ GA) meaning that the variable "x" occurs in the syntax of "X". The same can be done for "x occurs in T" (T ∈ IA) and "x occurs in B'" (B'∈PV').

Given these definitions one can construct a formal definition of the terms free and bound variables.

- x is neither free nor bound in skip or abort.
- x occurs free (bound) in a → A if it occurs free (bound) in A.
- x occurs free (bound) in a.A if it occurs free (bound) in A.
- x occurs free in x → A and a.x → A.
- x occurs free in y → A and a.y → A (x≠y) if it occurs free in A.
- x occurs bound in y → A and a.y → A if it occurs bound in A.
- x occurs bound in x:T → A and a.x:T → A.
- x occurs free in x:T → A and a.x:T → A if x occurs in T.
- x occurs free in y:T → A and a.y:T → A (y≠x) if x occurs free in A or if x occurs in T.

- x occurs bound in $y:T \to A$ and $a.y:T \to A$ $(y \neq x)$ if x occurs bound in A.
- x occurs free (bound) in $A \,[]\, C$ and $A;C$ if it occurs free (bound) in either A or C.
- x occurs free (bound) in $rec_j(B_1,\ldots,B_k).(A_1,\ldots,A_k)$ if it occurs free (bound) in any A_i.

(similar clause for infinite mutual recursion)

- x occurs free in A/X if x occurs free in A or x occurs in X.
- x occurs bound in A/X if it occurs bound in A.
- x occurs free in $(A_X \| _Y C)$ if x occurs free in A or C or if x occurs in X or Y.
- x occurs bound in $(A_X \| _Y C)$ if it occurs bound in A or C.
- x does not occur bound in B'.
- x occurs free in B' if x occurs in B'.

b) Process variables

The definition of free and bound occurrences of process variables is very much the same as the above. The critical clauses are the following.

- B occurs bound in $rec_j(\ldots B \ldots).(A_1,\ldots,A_k)$
- B occurs free in $rec_j(B_1,\ldots,B_k).(A_1,\ldots,A_k)$ if it occurs free in some of $A_1,\ldots,A_k$ and $B \neq B_1$ & ... & $B \neq B_k$.
- B occurs free in B.
- B occurs free in B_g if $B \in rge(g)$ (see 1.18).

The clause for infinite mutual recursion will be found in 1.18.

We can now define Proc, the space of process expressions, to be the set of those elements of Exp which contain no free variables of either kind. We will expect to be able to construct a semantics for Proc within P.

In order to be able to assign a value to general elements of Exp we will need a state σ which will be a mapping from variables to their values ($\sigma \in S = (AV \to \Sigma) \times (PV \to P)$).

Our semantic functions will be the following.

1.18 Semantic Functions

$\mathcal{E} : Exp \to S \to P$

$\mathcal{V} : GA \to S \to \mathcal{P}(\Sigma)$

$\mathcal{U} : IA \to S \to \mathcal{P}(\Sigma)$

$\mathcal{S} : PV' \to S \to P$

The definitions of $\mathcal{U}$ and $\mathcal{V}$ are fairly obvious and are omitted. The definition of $\mathcal{S}$ is delayed till the section on infinite mutual recursion.

The definition of $\mathcal{E}$ is as follows. Where a definition appears circular it is because we are using the versions of our operators which were defined (over P) in 1.3 - 1.12.

a) $\mathcal{E}[\![\underline{\text{skip}}]\!]\sigma = \underline{\text{skip}}$

b) $\mathcal{E}[\![\underline{\text{abort}}]\!]\sigma = \underline{\text{abort}}$

c) $\mathcal{E}[\![a \to A]\!]\sigma = a \to \mathcal{E}[\![A]\!]\sigma$

d) $\mathcal{E}[\![x \to A]\!]\sigma = \sigma[\![x]\!] \to \mathcal{E}[\![A]\!]\sigma$

e) $\mathcal{E}[\![a.x \to A]\!]\sigma = a.(\sigma[\![x]\!]) \to \mathcal{E}[\![A]\!]\sigma$

f) $\mathcal{E}[\![x:T \to A]\!]\sigma = \bigcup\{ a \to \mathcal{E}[\![A]\!]\sigma[a/x] \mid a \in \mathcal{U}[\![T]\!]\sigma \} \cup \{\langle\rangle\}$
(last term required in case $\mathcal{U}[\![T]\!]\sigma$ is empty)

g) $\mathcal{E}[\![a.x:T \to A]\!]\sigma = \bigcup\{ a.b \to \mathcal{E}[\![A]\!]\sigma[b/x] \mid b \in \mathcal{U}[\![T]\!]\sigma \} \cup \{\langle\rangle\}$

h) $\mathcal{E}[\![A \,[]\, C]\!]\sigma = \mathcal{E}[\![A]\!]\sigma \,[]\, \mathcal{E}[\![C]\!]\sigma$

i) $\mathcal{E}[\![A;C]\!]\sigma = \mathcal{E}[\![A]\!]\sigma ; \mathcal{E}[\![C]\!]\sigma$

j) $\mathcal{E}[\![(A_X \|_Y C)]\!]\sigma = (\mathcal{E}[\![A]\!]\sigma_{X^*} \|_{Y^*} \mathcal{E}[\![C]\!]\sigma)$, where $X^* = \mathcal{V}[\![X]\!]\sigma$ & $Y^* = \mathcal{V}[\![Y]\!]\sigma$

k) $\mathcal{E}[\![A/X]\!]\sigma = (\mathcal{E}[\![A]\!]\sigma)/(\mathcal{V}[\![X]\!]\sigma)$

l) $\mathcal{E}[\![a.A]\!]\sigma = a.(\mathcal{E}[\![A]\!]\sigma)$

m) $\mathcal{E}[\![B']\!]\sigma = \mathcal{S}[\![B']\!]\sigma$

n) $\mathcal{E}[\![\text{rec}_j(B_1,\ldots,B_k).(A_1,\ldots,A_k)]\!]\sigma = (\overset{\infty}{\underset{n=0}{U}} H^n(\{\langle\rangle\}^k))_j$
where $H:P^k \to P^k$ is the function defined
$H(\underline{C}) = (\mathcal{E}[\![A_1]\!]\sigma^*, \ldots, \mathcal{E}[\![A_k]\!]\sigma^*)$ $\quad (\sigma^* = \sigma[C_1/B_1] \ldots [C_k/B_k])$

In the above $\sigma[a/x]$ represents the state which is identical to σ except that it maps x to a. $\sigma[C/B]$ is the state identical to σ except for mapping B to C.

o) Suppose that Λ is some set of indices and that we have an injective mapping from Λ to PV, written B_λ $(\lambda \in \Lambda)$. Suppose further that $(\Gamma_1,\ldots,\Gamma_k)$ is a partition of Λ. Then we allow recursive definitions of the form:

B_g, where $\lambda \in \Gamma_1 \Rightarrow B_\lambda \Leftarrow A_1$
.....
.....
$\lambda \in \Gamma_k \Rightarrow B_\lambda \Leftarrow A_k$

It is assumed that the constants $(\in \Sigma^-)$ used in the construction of the A_i may depend on λ, and that the calls of the B_λ have the form $B_{g'}$ for some function g' of λ and any free alphabet variable(s).

We will assume that these functions carry with them a

syntactic check on their ranges, to enable accurate determination of which process variables are free in an expression of the form B_g. Usually the functions associated with the above recursion will have ranges $\subseteq \Lambda$, though sometimes a call may range over several nested recursions.

We will assume no definite syntax for the space of functions "g" making calls of process variables, since it clearly is heavily dependant on the natures of Λ and Σ. We will however assume that there is some procedure for determining which alphabet variables are free in any B_g, and also that there is some procedure π for evaluating any g in any state. Typical functions will be (i) constant functions; (ii) identity functions on Λ and (if $\Sigma \subseteq \Lambda$) on Σ; and (iii) (if $\Lambda = N$) "+1", "-1" etc. An alphabet variable will occur bound in the above recursion if it occurs bound in any of the A_i, and free if it either occurs free in any of the A_i or in B_g. All the elements of $\{B_\xi | \xi \in \Lambda\}$ occur bound together with any other process variables occurring bound in any A_i. A process variable occurs free if it is not in $\{B_\xi | \xi \in \Lambda\}$ and it either lies in the range of g or occurs free in any A_i.

We can now define the function $\mathcal{S}$ which evaluates procedure calls: $\mathcal{S}[\![B]\!]\sigma = \sigma[\![B]\!]$; $\mathcal{S}[\![B_g]\!]\sigma = \sigma[\![\pi(g)\sigma]\!]$.

Finally we can define $\mathcal{E}[\![B_g$, where.....$]\!]\sigma$

$= \sigma[\![\pi(g)\sigma]\!]$ if $\pi(g)\sigma \notin \{B_\xi | \xi \in \Lambda\}$;

$= (\bigcup_{n=0}^{\infty} H^n(\{\langle\rangle\}^\Lambda))_\xi$ if $\pi(g)\sigma = B_\xi$ $(\xi \in \Lambda)$

where P^Λ is the function space $\Lambda \to P$ (which is a complete lattice because P is); and $H: P^\Lambda \to P^\Lambda$ is defined:

$H(\underline{C})_\xi = \mathcal{E}[\![A_i^\xi]\!]\sigma^*$, where $\xi \in \Gamma_i$, $\sigma^* = \sigma[\underline{C}/\underline{B}]$ is the state identical to σ except for mapping B_ξ to C_ξ for each $\xi \in \Lambda$, and A_i^ξ is the expression obtained by substituting "ξ" for the parameter "λ" in A_i.

Note that to be fully rigorous we should be rather more careful in our treatment of the parameters λ used in this type of recursion. If we had been more pedantic we could have introduced a third class of variables representing these parameters; these would have their free occurrences in expressions representing elements of Σ and in B_gs, and be bound by the recursive definitions of the "schema" kind (1.18(o)).

1.19 Examples

(i) An integer register

Suppose that {iszero, up, down} ∪ set.N ⊆ Σ (N = {0,1,2,..}). Take Λ (the set of indices) to be N∪{u} (u ∉ N). An initially undefined register is represented by

R_u, where

$$R_u \Leftarrow \text{set.x:N} \rightarrow R_x$$

$$R_0 \Leftarrow (\text{iszero} \rightarrow R_0) \,[]\, (\text{up} \rightarrow R_1) \,[]\, (\text{set.y:N} \rightarrow R_y)$$

$$x \in N - \{0\} \Rightarrow R_x \Leftarrow (\text{down} \rightarrow R_{x-1}) \,[]\, (\text{up} \rightarrow R_{1+x}) \,[]\, (\text{set.y:N} \rightarrow R_y)$$

(ii) A stack

Suppose that T is some set of basic symbols, that ?T∪!T ⊆ Σ, and T* = Λ . An initially empty stack of type T is represented by

$S_{\langle\rangle}$, where

$$S_{\langle\rangle} \Leftarrow \text{?x:T} \rightarrow S_{\langle x\rangle}$$

$$w \in T^* - \{\langle\rangle\} \Rightarrow S_w \Leftarrow (\text{?x:T} \rightarrow S_{\langle x\rangle w}) \,[]\, (\text{!y} \rightarrow S_v) \quad \text{where } w = \langle y\rangle v$$

(iii) Palindromes

Suppose that {a,b,/} ⊆ Σ, then the following process is prepared to terminate successfully if and only if its current string is a palindrome of "a"s and "b"s.

$$P \Leftarrow \underline{\text{skip}} \,[]\, (a \rightarrow \underline{\text{skip}}) \,[]\, (b \rightarrow \underline{\text{skip}}) \,[]\, (a \rightarrow P;(a \rightarrow \underline{\text{skip}})) \,[]\, (b \rightarrow P;(b \rightarrow \underline{\text{skip}}))$$

Note that none of the above examples is completely strict in its use of recursive syntax, though it is perfectly clear in each case what the correct syntax is. From here on we will habitually use abbreviations such as the above, on the understanding that we could translate into "correct" syntax if challenged.

Note how the palindromes example breaks two of the conventions designed to make a process implementable in a reasonable way. The difficulties in this case seem to have more to do with the nature of the problem than with "bad programming" since it is possible for some strings to be initial segments of palindromes in several essentially different ways. For example ⟨ababa⟩ , in addition to being a palindrome itself, is an initial segment of ⟨abababa⟩ , ⟨ababababa⟩ and all the longer ones of which ⟨ababa⟩ is less than half.

Many more examples will appear later, especially in chapters two, five and six.

The notation used in conjunction with the above formal semantics for CSP is rather cumbersome to use in practice. In future we will habitually identify the syntax of a process with its value in our model. The following is a sequence of easily proved results which justify this identification (in the case of expressions with no free variables of any type).

1.20 Lemma

(i) If $A \in Exp$, $a \in \Sigma^-$ and x is an alphabet variable which does not occur free in A then $\mathcal{E}[\![A]\!]\sigma = \mathcal{E}[\![A]\!]\sigma[a/x]$ for every state σ.

(ii) If $A \in Exp$, $\underline{C} \in P^\lambda$ and $\underline{B} \in PV^\lambda$ is a vector none of whose components occurs free in A then $\mathcal{E}[\![A]\!]\sigma = \mathcal{E}[\![A]\!]\sigma[\underline{C}/\underline{B}]$ for every state σ.

1.21 Corollary

If $A \in Exp$ has no free variables then $\mathcal{E}[\![A]\!]\sigma = \mathcal{E}[\![A]\!]\rho$ for all states σ and ρ.

If A_1 and A_2 are elements of Exp let us say that $A_1 \equiv A_2$ if $\mathcal{E}[\![A_1]\!]\sigma = \mathcal{E}[\![A_2]\!]\sigma$ for all states σ.

1.22 Lemma

Clauses a - i of 1.13 remain true if elements of Exp are substituted for elements of P (etc.) and $\equiv$ is substituted for = throughout. Clauses j - m remain true if the stated conditions are true for all states (e.g. clause l remains true provided $\mathcal{V}[\![X]\!]\sigma \cap \mathcal{V}[\![Z]\!]\sigma = \emptyset$ for all σ).

Additionally:

n) $x \to (B;C) \equiv (x \to B);C$, $a.x \to (B;C) \equiv (x \to B);C$

o) $x{:}T \to (A;C) \equiv (x{:}T \to B);C$ if x does not occur free in C

p) $a.x{:}T \to (A;C) \equiv (a.x{:}T \to B);C$ " " " " " "

1.23 Lemma

If $A_1, A_2, C \in Exp$, $A_1 \equiv A_2$ and A_1 is a syntactic subcomponent of C, then the result of substituting A_2 for A_1 in C (C', say) satisfies $C' \equiv C$.

1.24 Lemma

If $A \in Exp$, $\sigma \in S$ and $\underline{B} \in PV^{\Lambda}$ then the function $H: P^{\Lambda} \to P^{\Lambda}$ defined $H(\underline{C}) = \mathcal{E}[\![A]\!]\sigma[\underline{C}/\underline{B}]$ is monotonic and continuous.

1.25 Corollary

Each recursively defined process satisfies its defining equation in the following senses:

a) $\mathcal{E}[\![rec_j(B_1,\ldots,B_k).(A_1,\ldots,A_k)]\!]\sigma = \mathcal{E}[\![A_j]\!]\sigma[C_1/B_1]\ldots[C_k/B_k]$
where $C_r = \mathcal{E}[\![rec_r(B_1,..,B_k).(A_1,..,A_k)]\!]\sigma$.

b) In the recursion

$\lambda \in \Gamma_1 \Rightarrow B_\lambda \Leftarrow A_1$
. . . . . .
. . . . . .
$\lambda \in \Gamma_k \Rightarrow B_\lambda \Leftarrow A_k$

($\{\Gamma_1,..,\Gamma_k\}$ a partition of some indexing set Λ)

we have $\mathcal{E}[\![B_g$, where.....$]\!]\sigma = \mathcal{E}[\![A_i^\kappa]\!]\sigma[\underline{C}/\underline{B}]$ if $\pi(g)\sigma = B_\kappa \,\&\, \kappa \in \Gamma_i$
where $C_\mu = \mathcal{E}[\![B_\mu$, where.....$]\!]\sigma$ and A_i^κ is the element of Exp obtained by substituting the value "κ" into the parameter "λ".

1.26 Lemma

If all free occurrences of $B_1,..,B_k$ have the form of simple calls of process variables (i.e. "B" rather than "B_g") and if A_j has no bound occurrence of any variable which occurs free in any of the A_i then

$rec_j(B_1,..,B_k).(A_1,..,A_k) \equiv A_j[C_1'/B_1]\ldots[C_k'/B_k]$

where $C_r' = rec_r(B_1,..,B_k).(A_1,..,A_k)$ and syntactic substitution for (free occurrences of) simple calls of process variables is defined in the obvious way.

1.27 Lemma

If in the recursion of 1.25(b) we have

(i) for all states σ, $\pi(g)\sigma \in \{B_\mu \mid \mu \in \Gamma_i\}$;

(ii) A_i contains no bound occurrence of any variable which occurs free in any of $A_1,..,A_k$;

(iii) the only occurrences of B_μ ($\mu \in \Lambda$) in A_j are free and of the form B_h for some h s.t. $rge(h) \subseteq \Lambda$;

then 'B_g, where....' $\equiv A_i^*$, where A_i^* is the expression obtained by substituting 'B_h, where... (usual clauses)' for all calls of process variables s.t. $rge(h) \subseteq \Lambda$.

The conditions on the form of recursive calls within processes could be relaxed somewhat in 1.26 and 1.27, but this would be at the expense of simplicity and the extra cases included would very rarely arise in practice.

From here on we will identify elements of Exp with no free variables with their unique (by 1.21) counterparts in P. For this type of expression we thus identify "$\equiv$" (equivalence over Exp) with actual equality over P. We are enabled by 1.22 - 1.27 to manipulate these expressions and their subcomponents with a great deal of freedom. Subject to a few conditions on the use of variables we can treat the components of an expression very much as though they were elements of P, and recursive definitions very much like fixed point equations over P and its product spaces. We can thus practically forget the formal semantic definition of our language when doing manipulations, confident that we could fully justify all of them if challenged.

There is a clear ambivalence in our attitude to constructs in our language, which is brought about by the identification of elements of Exp and elements of P. One the one hand we can think of an expression as a formal piece of syntax which can only be given a value by reference to a semantic function and a state. On the other hand we can think of it as being an operator dependent on its free variables; there being a class of results (1.20 - 1.27) showing these two approaches to be congruent. Although from here on it is the second of these two ideas which will dominate, there are several important uses for the more formal approach. The most obvious of these is that it enables us to define precisely what we mean by any construct, however involved.

A second reason is that there are other, similar languages which are almost impossible to define without "states" and some kind of formal denotational semantics. The language thich we have adopted is very rich in its potential for parallelism and recursion, but lacks such features as assignment to variables (other than by communication with other processes) and conditional statements. The choice of language made in this chapter largely results from the fact that most of the work in the subsequent chapters chapters is chiefly concerned with parallelism, recursion, and the connection between the two. The introduction of further constructs such as those mentioned above would have tended

to intr duce extra cases into our arguments (many of which are complex enough already). It is not difficult, either in the deterministic model P or the nondeterministic model we meet in chapter four, to devize semantics for languages with more conventional constructs (e.g. assignment, "if, then, else", less rich recursion) by simple adaptation of the formal semantics we have met in this chapter. Having done this one can prove congruences between the languages and thereby apply results obtained about one to the other.

1.28 Example

In the language we use assignment to variables is often modelled by elaborate subscripting in mutual recursion, and conditional statements by "□". For examples of the former see 1.19 (i, ii); for examples of the latter see 6.9. The following example represents a process which inputs a succession of symbols from some set T and after each one outputs the largest one it has received so far. (We assume that T is endowed with some total order $<$, and that the syntax for IA is extended to encompass the usage below.)

A, where $A \Leftarrow ?x{:}T \to !x \to B_x$

$x \in T \Rightarrow B_x \Leftarrow (?y{:}\{y \mid y>x\} \to !y \to B_y)$
$\quad \square\ (?y{:}\{y \mid y<x\} \to !x \to B_x)$

Additional combinators

From time to time we will want to use combinators additional to those which we have already met. These will be defined over P on the understanding that if the syntax introduced in 1.16 were extended to include them then the semantic function $\mathcal{E}$ would be extended in the natural way: if op is a (unary) operator over P and op is the syntactic object intended to model it then $\mathcal{E}[\![op(A)]\!]\sigma = op(\mathcal{E}[\![A]\!]\sigma)$. A few operators are introduced in this way below.

(i) If $f: \Sigma - \Sigma \cup \{*\}$ is a function such that $f^{-1}(\surd) \subseteq \{\surd\}$ then one can extend f to Σ^* by the rule:

$f(\langle\rangle) = \langle\rangle$; $f(w\langle a\rangle) = f(w)$ if $f(a) = *$;
$f(w\langle a\rangle) = f(w)\langle f(a)\rangle$ otherwise .

One can now define an operator on P by

$$f^{-1}(A) = \{w \in \underline{run} \mid f(w) \in A\} .$$

For example if $X \subseteq \Sigma$ and f_X is the function defined

$f(a) = a$ if $a \in X$; $f(a) = *$ otherwise

then $f_X^{-1}(A)$ is the process which behaves like A in the set X and is prepared to do anything outside X; one can think of $f_X^{-1}(A)$ as "A ignoring $(\Sigma - X)$". Note that $(A_X \|_Y B) = (X \cup Y)^* \cap f_X^{-1}(A) \cap f_Y^{-1}(B)$.

(ii) If f is any function of the above type it can be extended to an operator over P directly:

$f(A) = \{f(w) \mid w \in A\}$.

For example if "a" is any name used for elements of Σ we can define a function strip(a) on Σ as follows:

$\text{strip}(a)(c) = c$ if $c \notin a.\Sigma$

$= d$ if $c = a.d$ for some $d \in \Sigma$.

Regarded as an operator over P strip(a) is the operator which removes the name "a" from any of its operand's communications which have it. This operator can be combined with others to construct some useful combinators. For example if T is some set of unnamed symbols and $T \cup ?T \cup !T \subseteq \Sigma$ we can define a "pipe" operator "$\gg$" which is intended to take two processes which communicate in $?T \cup !T$, join the inputs of one to the outputs of the other, and hide the resulting internal communications:

$A \gg B = (\text{strip}!(A)_{T \cup ?T} \|_{T \cup !T} \text{strip}?(B))/T$.

This and similar operators will be studied in some detail in the later chapters.

The two types of operators described above (f and f^{-1}) are known as alphabet transformers. Note that for all allowable functions f we have $f^{-1}(\underline{\text{run}}) = \underline{\text{run}}$ and $f(\underline{\text{abort}}) = \underline{\text{abort}}$.

Chapter 2 :- Recursion Induction in the Deterministic Model

Recursion induction is the proof rule:

> If in a recursive definition of a process we can prove some property true of the process by assuming it true of all recursive calls, then we can infer the property true of the process.

In the model described in chapter 1 the above rule can be formalised thus:

2.1 In the (mutual) recursive definition $\underset{\sim}{A} = F(\underset{\sim}{A})$, where $\underset{\sim}{A}$ is a vector of processes in $P^{\wedge}$ and $F: P^{\wedge} \to P^{\wedge}$, if R is a predicate on $P^{\wedge}$ such that $\forall \underset{\sim}{B}.\ R(\underset{\sim}{B}) \Rightarrow R(F(\underset{\sim}{B}))$ then we can infer that $R(A)$ holds (where $\underset{\sim}{A} = fix(F)$).

In this chapter we will examine the validity of this rule, give examples of its use and show how it can be extended in several directions. The rule as it stands is not universally valid. Below are some counter-examples which will motivate conditions upon F and R to ensure validity.

2.2 The rule 2.1 is completely without the base case usually found in inductive proofs. It is therefore possible to "prove" the predicate "false" true of any process whatsoever.

2.3 Of the process $A \Leftarrow A$ we could prove any predicate.

2.4 Of the process $A \Leftarrow A \,\square\, B$ (B some fixed process) we could prove "A = C" for any process $C \supseteq B$.

2.5 Of the process $A = a \to A$ we could prove the predicate "every trace of A can be extended to include a "b" ".

In 2.2 it is clearly the predicate which is at fault. Any predicate which is not equivalent to "false" is <u>satisfiable</u> in the sense $\exists \underset{\sim}{A}.\ R(\underset{\sim}{A})$. It is this condition that will represent the base case of our inductions. In 2.3 it is clearly the recursion which is at fault, and under the assumption that "=C" is a valid predicate this must also be the case in 2.4. In 2.5 the recursion is very straightforward and well-defined so we must assume that the predicate (with its implicit existential quantifier) is at fault.

There are several possible pairs of conditions on predicates and functions which, along with satisfiability, can be shown to make the rule 2.1 valid.

We firstly examine the most commonly useful of these. Define the notion of restriction of a deterministic process to strings of length n or less as follows:

2.6 $A \restriction n = \{w \in A \mid |w| \leq n\}$

This can be extended to vectors of processes in P^{Λ} by pointwise restriction:

2.7 $(\underset{\sim}{A} \restriction n)_{\lambda} = (A_{\lambda}) \restriction n$ for $\underset{\sim}{A} \in P^{\Lambda}$

These definitions give rise to some obvious results:

2.8 Lemma

a) $A \in P \Rightarrow A \restriction o =$ abort
b) $\underset{\sim}{A} \in P^{\Lambda} \Rightarrow (\underset{\sim}{A} \restriction n) \restriction m = \underset{\sim}{A} \restriction \min(n,m)$
c) $(a \rightarrow A) \restriction n+1 = a \rightarrow (A \restriction n)$ if $a \in \Sigma$, $A \in P$
d) $\underset{\sim}{A} \in P^{\Lambda} \Rightarrow A_{\lambda} = \bigcup_{n=0}^{\infty} ((\underset{\sim}{A} \restriction n)_{\lambda})$ if $\lambda \in \Lambda$

Suppose $F: P^{\Lambda} \rightarrow P^{\Lambda}$, define F to be constructive if it satisfies

2.9 $\forall n \in N. \forall \underset{\sim}{A} \in P^{\Lambda}.\ F(\underset{\sim}{A}) \restriction n+1 = F(\underset{\sim}{A} \restriction n) \restriction n+1$

and non-destructive if it satisfies

2.10 $\forall n \in N. \forall \underset{\sim}{A} \in P^{\Lambda}.\ F(\underset{\sim}{A}) \restriction n = F(\underset{\sim}{A} \restriction n) \restriction n$

Informally these definitions mean that by looking at the result of applying F to the n-step behaviour of $\underset{\sim}{A}$ we can deduce the n-step behaviour of $F(\underset{\sim}{A})$ in the case of a non-destructive F and the n+1 step behaviour in the case of a constructive F. From the point of view of recursion a constructive F would seem to be a desirable thing, since then one can always guarantee to be able to deduce the behaviour of fix(F) up to any finite time from only finitely many iterations.

2.11 Lemma

a) If $F: P^{\Lambda} \rightarrow P^{\Lambda}$ and $G: P^{\Lambda} \rightarrow P^{\Lambda}$ are both non-destructive then so are FoG and GoF.
b) If a function is constructive then it is non-destructive.
c) If $F: P^{\Lambda} \rightarrow P^{\Lambda}$ is constructive and $G: P^{\Lambda} \rightarrow P^{\Lambda}$ is non-destructive then FoG and GoF are both constructive.
d) If $F: P^{\Lambda} \rightarrow P^{\Lambda}$ is constructive then it has a unique fixed point.

proof

a) $F(G(\underset{\sim}{A}))\restriction n = F(G(\underset{\sim}{A})\restriction n)\restriction n$ (by nondestructiveness of F)

$= F(G(\underset{\sim}{A}\restriction n)\restriction n)\restriction n$ (" " " " " G)

$= F(G(\underset{\sim}{A}\restriction n))\restriction n$ (" " " " " F)

b) $F(\underset{\sim}{A})\restriction n = (F(\underset{\sim}{A})\restriction n+1)\restriction n$ (by 2.8(b))

$= (F(\underset{\sim}{A}\restriction n)\restriction n+1)\restriction n$

$= F(\underset{\sim}{A}\restriction n)\restriction n$

c) $F(G(\underset{\sim}{A}))\restriction n+1 = F(G(\underset{\sim}{A})\restriction n)\restriction n+1$

$= F(G(\underset{\sim}{A}\restriction n)\restriction n)\restriction n+1$

$= F(G(\underset{\sim}{A}\restriction n))\restriction n+1$

GoF is very similar to this.

d) Suppose F has two distinct fixed points $\underset{\sim}{A}$ & $\underset{\sim}{B}$. Then by 2.8 (a) & (d) we have $\underset{\sim}{A}\restriction o = \underset{\sim}{B}\restriction o$ and $\exists n.\ \underset{\sim}{A}\restriction n \neq \underset{\sim}{B}\restriction n$. Hence there is some m such that $\underset{\sim}{A}\restriction m = \underset{\sim}{B}\restriction m$ but $\underset{\sim}{A}\restriction m+1 \neq \underset{\sim}{B}\restriction m+1$.

But then $\underset{\sim}{A}\restriction m+1 = F(\underset{\sim}{A})\restriction m+1$ (as $\underset{\sim}{A} = F(\underset{\sim}{A})$)

$= F(\underset{\sim}{A}\restriction m)\restriction m+1$ (as F is constructive)

$= F(\underset{\sim}{B}\restriction m)\restriction m+1$ ($\underset{\sim}{B}\restriction m = \underset{\sim}{A}\restriction m$)

$= F(\underset{\sim}{B})\restriction m+1$ (as F is constructive)

$= \underset{\sim}{B}\restriction m+1$ contradicting assumption

Thus F must only have one fixed point (it has at least one by Tarski).

Now suppose that $R:P^{\omega} \to \{\text{true, false}\}$ is a predicate. Define R to be <u>continuous</u> if it satisfies

2.12 $\forall \underset{\sim}{A}.(\neg R(\underset{\sim}{A}) \Rightarrow \exists n.(\forall \underset{\sim}{B}.(\underset{\sim}{B}\restriction n = \underset{\sim}{A}\restriction n) \Rightarrow \neg R(\underset{\sim}{B})))$

or equivalently

2.13 $\forall \underset{\sim}{A}.(\forall n.\exists \underset{\sim}{B}.(\underset{\sim}{A}\restriction n = \underset{\sim}{B}\restriction n)\ \&\ R(\underset{\sim}{B})) \Rightarrow R(\underset{\sim}{A})$

(Later this will sometimes be termed <u>weak continuity</u> to contrast with another notion which will be introduced at the end of this chapter.)

Informally a predicate is continuous if, for every $\underset{\sim}{B}$ such that $R(\underset{\sim}{B})$ does not hold we can be sure of this after some finite time. Observe that the predicate of 2.5 does not satisfy this as at any finite time the "b"s might be "just around the corner".

2.14 Theorem

If $F:P^{\checkmark} \to P^{\checkmark}$ is a constructive function and if R is a continuous satisfiable predicate then rule 2.1 is valid.

(i.e. $(\forall \underset{\sim}{B}.\ R(\underset{\sim}{B}) \Rightarrow R(F(\underset{\sim}{B}))) \Rightarrow R(fix(F))$)

proof

Since R is satisfiable there is some $\underset{\sim}{B}$ such that $R(\underset{\sim}{B})$ and (by 2.8 (a)) $\underset{\sim}{B}\restriction o = \underset{\sim}{A}\restriction o$ (where $\underset{\sim}{A} = fix(F)$). Claim that for all n we have $R(F^n(\underset{\sim}{B})) \ \&\ (F^n(\underset{\sim}{B})\restriction n = \underset{\sim}{A}\restriction n)$. This is true when n=0, so assume it true for n.

Then
$$\begin{aligned}
\underset{\sim}{A}\restriction n+1 &= F(\underset{\sim}{A})\restriction n+1 && \text{(as } F(\underset{\sim}{A}) = \underset{\sim}{A}\text{)}\\
&= F(\underset{\sim}{A}\restriction n)\restriction n+1 && \text{(as F is constructive)}\\
&= F(F^n(\underset{\sim}{B})\restriction n)\restriction n+1 && \text{(by induction)}\\
&= F(F^n(\underset{\sim}{B}))\restriction n+1 && \text{(as F is constructive)}\\
&= F^{n+1}(\underset{\sim}{B})\restriction n+1 && \text{as desired}
\end{aligned}$$

also $R(F^n(\underset{\sim}{B})) \Rightarrow R(F(F^n(\underset{\sim}{B})))$ completing induction

Hence the above holds for all n. Thus by 2.13 (since R is continuous) we must have $R(\underset{\sim}{A})$, as desired.

This pair of conditions is useful since they normally hold of well-defined recursions and predicates we wish to prove of them and also because there are some simple rules for checking that they hold.

2.15 Lemma

The following are all non-destructive functions of their variables: $a \to A$, $A \square B$, $A;B$, $(A_X||_Y B)$, $a.A$.

The following are constructive functions of A:

$a \to A$

$B;A$ when $\checkmark \notin B^o$

$(B_X||_Y A)$ when $Y \subseteq X$ and $B^o \cap Y = \emptyset$

The proofs of all these results are elementary and follow directly from the definitions of the operators.

For example $(B;A)\restriction n = \{w \mid \neg\exists v.w=v\checkmark \ \&\ w\in B \ \&\ |w|\leq n\}$
$$\cup \{wv \mid w\checkmark\in B \ \&\ v\in A \ \&\ v \neq \langle\rangle \ \&\ |wv|\leq n\}$$
(in line 2 the case $v=\langle\rangle$ is included in line 1)
$$\subseteq \{w \mid \neg\exists v.w=v\checkmark \ \&\ w\in B\restriction n\} \cup \{wv \mid w\checkmark\in B\restriction n \ \&\ v\in A\restriction n\}$$
$$\subseteq (B\restriction n);(A\restriction n)$$
$$\Rightarrow B;A\restriction n \subseteq ((B\restriction n);(A\restriction n))\restriction n \quad (\supseteq \text{ follows by monotonicity})$$

We can now combine this lemma with 2.11 a&c to give the following result:

2.16 Theorem

Suppose that the function $F:P^{\wedge} \to P^{\wedge}$ is such that each component of $F(\underset{\sim}{A})$ is a syntactic expression involving only expressions independent of $\underset{\sim}{A}$ (such as skip and abort), "$A_{\sim}$"s and the combinators $a \to B$, $a?x:T \to B(x)$, $?x:T \to B(x)$, $B;C$, $B \,\Box\, C$, $a.B$ and $(B_X \,||_Y C)$. Then provided that every recursive call of an A_λ is guarded directly (as in $a \to A_\lambda$) or indirectly (as in $a \to (A_\lambda \,\Box\, B)$) the function F is constructive.

The hiding operator A/X is not non-destructive so we must be careful when we use it in a recursive definition. There are important cases where the $(A \,||\, a::B)$ and $(A \gg B)$ operators are constructive and non-destructive. These will be examined later on.

2.17 Examples

The following recursions are all constructive:

a) $A \Leftarrow (a \to A) \,\Box\, (b \to \underline{skip})$

b) $B \Leftarrow (?t \to T) \,\Box\, (?f \to F)$

$T \Leftarrow (?t \to T) \,\Box\, (?f \to F) \,\Box\, (\ t \to T)$

$F \Leftarrow (?t \to T) \,\Box\, (?f \to F) \,\Box\, (\ f \to F)$

c) $A \Leftarrow ((a \to (A \,\Box\, (b \to \underline{skip})))_X \,||_Y A) \,\Box\, b \to A$

where $X = \{a, b, \surd\}$, $Y = \{b, \surd\}$

None of the following is constructive:

d) $A \Leftarrow (A \,\Box\, a \to \underline{skip})$

e) $A \Leftarrow a.A \,\Box\, a \to A$

f) $A \Leftarrow (a \to A) \,\Box\, (B;A)$

$B \Leftarrow (a \to \underline{skip}) \,\Box\, A$

We will come back to the case of iterated recursions (i.e the body of a recursive definition including a recursion) later.

We now turn our attention to the classification of continuous predicates. The following is a list of the classes of predicates which can easily be shown to be continuous. Some of these classes can clearly be derived from others. In the next chapter we will examine the continuous predicates as a topology of the space P which will yield more insight on them.

2.18 Theorem

The following predicates are all (weakly) continuous:

$R(\underset{\sim}{A})$ =

(i) $A_\lambda = B$

(ii) $A_\lambda = A_\mu$

(iii) $A_\lambda \subseteq B$

(iv) $A_\lambda \supseteq B$

(v) $A_\lambda \subseteq A_\mu$

(vi) $A_\lambda \neq B$ if there is an upper bound on $\{|w| \mid w \in B\}$

(vii) A_λ is deadlock-free

(viii) $w \in A_\lambda \Rightarrow p(w)$ where p is any predicate on Σ^*

(ix) $w \in A_\lambda \Rightarrow p_w((A_\lambda \text{ after } w)^o)$ if p_w are predicates on $\mathcal{P}(\Sigma)$

(x) $R_1(F(\underset{\sim}{A}))$ if $\exists g:N \to N$ s.t. $\forall \underset{\sim}{B}.\forall n.F(\underset{\sim}{B})\restriction n = F(\underset{\sim}{B}\restriction g(n))\restriction n$
(F monotonic $P^\lambda \to P^\Gamma$ and R_1 predicate on P^Γ)

(xi) $F(\underset{\sim}{A})_\lambda \subseteq B$ for any continuous $F:P^\lambda \to P^\Gamma$

(xii) $\bigwedge_{\gamma \in \Gamma} R_\gamma(\underset{\sim}{A})$ for any set Γ

(xiii) $R_1(\underset{\sim}{A}) \vee R_2(\underset{\sim}{A})$

where B is any constant process and R_1, R_2 & R_γ are all continuous predicates.

example proofs

(i) If $(A_\lambda \neq B)$ then either $\exists w.\ w \in A_\lambda$ & $w \notin B$
or $\exists w.\ w \notin A_\lambda$ & $w \in B$
In either case if we put $n = |w|$ then $C_\lambda\restriction n = A_\lambda\restriction n \Rightarrow C_\lambda \neq B$
so $\underset{\sim}{C} \in P^\lambda \Rightarrow \underset{\sim}{C}\restriction n = \underset{\sim}{A}\restriction n \Rightarrow \neg(C_\lambda = B)$

(vi) Observe that $C \in P \Rightarrow (C = B \Leftrightarrow C\restriction n+1 = B)$ where n is the upper bound upon $\{|w| \mid w \in B\}$, since either $C\restriction n \neq B$ or C contains some string of length n+1 if $C \neq B$.
Thus $A_\lambda = B \Rightarrow \forall C.((C\restriction n+1 = (A_\lambda)\restriction n+1) \Rightarrow (C = B))$

(x) $\neg R_1(F(\underset{\sim}{A})) \Rightarrow \exists n.\ \underset{\sim}{B}\restriction n = F(\underset{\sim}{A})\restriction n \Rightarrow \neg R(\underset{\sim}{B})$
$\Rightarrow F(\underset{\sim}{B})\restriction n = F(\underset{\sim}{A})\restriction n \Rightarrow \neg R(F(\underset{\sim}{B}))$
but $\underset{\sim}{B}\restriction g(n) = \underset{\sim}{A}\restriction g(n) \Rightarrow F(\underset{\sim}{B})\restriction n = F(\underset{\sim}{A})\restriction n$
so $\underset{\sim}{B}\restriction g(n) = \underset{\sim}{A}\restriction g(n) \Rightarrow \neg R(F(\underset{\sim}{B}))$

The satisfiability of predicates is very often obvious. The examples (i) - (viii) on the previous page fall into this category (provided, in case (viii) that $p(\langle\rangle)$ holds, though if not the predicate is not satisfiable as every process contains $\langle\rangle$). Also (xiii) is satisfiable if one of its components is, and (xii) is if each of its components is and they all refer to different components of $\underline{A}$. A method for getting round this last requirement (that all the R_x should be independent) will be indicated later. (xi) will be satisfied by $\underline{abort}^{\wedge}$ if it is satisfiable. The cases (ix) and (x) are more difficult and it is necessary to examine their individual instances. Often when it is not easy to prove a given predicate of these forms satisfiable it is possible to break it down into two or more sections and prove each of these to be true of fix(G) (where G is the function of the recursion). For example to prove $F(fix(G)) = B$ it suffices to prove $F(fix(G)) \supseteq B$ and $F(fix(G)) \subseteq B$.

It is now possible to give some provably valid applications of rule 2.1.

2.19 Example (C.A.R. Hoare)

We attempt to model an integer counter in two ways. An integer counter will be expected to communicate in the alphabet $\{up, down, iszero\}$. It will always be non-negative, so when it has value zero it will not accept a "down" instruction, and it will only communicate "iszero" when it has value zero. The instruction "up" will increase its value by one, the instruction "down" will reduce its value by one and "iszero" will not affect its value.

Our first attempt is by mutual recursion (with $\Lambda = N$)

$$COUNT_0 \Leftarrow (iszero \rightarrow COUNT_0) \,\square\, (up \rightarrow COUNT_1)$$

$$COUNT_{n+1} \Leftarrow (down \rightarrow COUNT_n) \,\square\, (up \rightarrow COUNT_{n+2})$$

Of these processes in isolation it is possible to prove several properties without too much difficulty, notably that each $COUNT_n$ is deadlock-free and that $w \in COUNT_n \Rightarrow ups(w)+n \geq downs(w)$ (where ups and downs have the obvious interpretation). This second property is just the conjunction of independent cases of 2.18 (viii), as the first is of cases of (vii).

For the second attempt we will first construct a process which terminates successfully after inputting one more "down" than "up"s.

$$POS \Leftarrow (down \rightarrow \underline{skip}) \,\square\, (up \rightarrow POS;POS)$$

(If POS inputs "down" on its first step it terminates, but if it inputs an "up" it must subsequently input two more "down"s than "up"s.)

We can now use this process to define an integer counter with initial value zero.

$$ZERO \Leftarrow (iszero \rightarrow ZERO) \,\square\, (up \rightarrow POS;ZERO)$$

(ZERO will accept "iszero" and remain unaltered, or accept "up" and become ZERO again after inputting one more "down" than "up"s.)

It is an immediate consequence of 2.16 that all the recursions defined above (COUNT, POS & ZERO) are constructive. Also if we define $C_o = ZERO$, $C_{n+1} = POS;ZERO$ then the predicate $R(\underset{\sim}{A}) = \forall n.\ A_n = C_n$ is continuous and satisfiable (by 2.18 et seq).

Claim that COUNT satisfies R. To prove this it is now sufficient to prove $R(\underset{\sim}{A}) \Rightarrow R(F(\underset{\sim}{A}))$ where F is the function of the COUNT recursion.

Suppose $R(\underset{\sim}{A})$

Then
$$\begin{aligned} F(\underset{\sim}{A})_o &= (iszero \rightarrow A_o) \,\square\, (up \rightarrow A_1) \\ &= (iszero \rightarrow ZERO) \,\square\, (up \rightarrow POS;ZERO) \quad \text{by assumption} \\ &= ZERO = C_o \quad \text{by definition of ZERO} \end{aligned}$$

$$\begin{aligned} F(\underset{\sim}{A})_{n+1} &= (down \rightarrow A_n) \,\square\, (up \rightarrow A_{n+2}) \\ &= (down \rightarrow skip;C_n) \,\square\, (up \rightarrow POS;POS;C_n) \quad \text{by assumption} \\ &= ((down \rightarrow skip) \,\square\, (up \rightarrow POS;POS));C_n \\ &= POS;C_n \quad \text{by definition of POS} \\ &= C_{n+1} \end{aligned}$$

This completes the proof. Thus in particular we have proved that $ZERO = COUNT_o$, so the two processes we have defined are essentially equivalent.

2.20 Example: Buffers

Define a buffer to be a process which accepts input from some set T and later outputs it in the same order. We can express this condition formally as: B is a buffer if $B \subseteq (?T \cup !T)^*$ and $w \in B \Rightarrow ins(w) \geq outs(w)$, where $ins(w) = strip?(w \restriction ?T)$ and

$outs(w) = strip^{!}(w\lceil^{!}T)$. This condition, being of type (viii) only represents a form of partial correctness (it is satisfied by abort). If we were in addition to demand that B be free from deadlock this would eliminate abort but not other pathological cases like $B \Leftarrow ?a \rightarrow B$ which can only input "a"s and can never output at all. One condition which seems to be satisfactory in all respects is the following:

$$w \in B \Rightarrow w \in (?T \cup T)^{*} \;\&\; ins(w) \geqslant outs(w) \qquad (i)$$

$$\&\; ins(w) = outs(w) \Rightarrow (B \text{ after } w)^{o} = ?T \qquad (ii)$$

$$\&\; ins(w) > outs(w) \Rightarrow (B \text{ after } w)^{o} \cap {}^{!}T \neq \emptyset \qquad (iii)$$

The interpretation of line (ii) is that an empty buffer must be prepared to accept any input, and line (iii) means that a non-empty buffer must be prepared to output. The fact that this output is correct is implied by line (i).

That the above predicate is satisfiable is by no means obvious. However each of the lines individually is ((i) & (iii) by abort and (ii) by $?x{:}T \rightarrow$ abort) so if for any given process B we can prove all three independently we will have established that they are simultaneously satisfiable. The process we choose to do this job for us is the one-place buffer

$$B \Leftarrow ?x{:}T \rightarrow ({}^{!}x \rightarrow B) \quad .$$

The function associated with this recursion can be written

$$F(A) = \{\langle\rangle\} \cup \{\langle ?x \rangle \mid x \in T\} \cup \{\langle ?x\, {}^{!}x \rangle w \mid w \in A\}$$

Since each of the three predicates is continuous and satisfiable and this recursion is constructive, to prove them true of B it will suffice to show $\forall A.\ R(A) \Rightarrow R(F(A))$ for each R.

Suppose (i) holds of A $\quad w \in F(A) \Rightarrow w = \langle\rangle$ $(\Rightarrow ins(w) = outs(w))$

or $w = \langle ?x \rangle$ for some $x \in T$

$(\Rightarrow ins(w) > outs(w))$

or $w = \langle ?x\, {}^{!}x \rangle v$ for some $x \in T$, $v \in A$

$(\Rightarrow ins(w) = \langle x \rangle ins(v')$

$\geqslant \langle x \rangle outs(v)$

$\geqslant outs(w))$

Thus (i) holds of F(A).

Suppose (ii) holds of A. Then $w \in F(A)$ & $(ins(w)=outs(w))$

$\Rightarrow$ (a) $w=\langle\rangle$ or (b) $w=\langle ?x\,!\,x\rangle v$ where $x \notin T$, $v \in A$ & $ins(v)=outs(v)$

(a) $\Rightarrow$ $(F(A)\ \underline{after}\ w)^{o} = ?T$

(b) $\Rightarrow$ $(F(A)\ \underline{after}\ w)^{o} = (A\ \underline{after}\ v)^{o} = ?T$

as desired

Thus (ii) holds of F(A).

Suppose (iii) holds of A. Then $w \in F(A)$ & $(ins(w) > outs(w))$

$\Rightarrow$ (a) $w=\langle ?x\rangle$ for some $x \in T$

or (b) $w=\langle ?x\,!\,x\rangle v$ for some $x \in T$, $v \in A$ s.t. $ins(v) > outs(v)$

(a) $\Rightarrow$ $!x \in (F(A)\ \underline{after}\ w)^{o}$ since $\langle ?x\,!\,x\rangle \in F(A)$ for all $x \in T$

(b) $\Rightarrow$ $(F(A)\ \underline{after}\ w)^{o} = (A\ \underline{after}\ v)^{o}$

$\Rightarrow$ $(F(A)\ \underline{after}\ w)^{o} \cap\, !T \neq \emptyset$ as A satisfies (iii)

This completes the proof that B is a buffer. Therefore when in future we want to use the predicate "is a buffer" (in the definition on the previous page) we will be entitled to assume that it is satisfiable.

Extensions of our rule

Firstly it is possible to weaken the definition of constructive function to F non-destructive and

2.21 $\forall \underset{\sim}{A}.\forall n.\exists m.(F^{m}(\underset{\sim}{A})\restriction n+1 = F^{m}(\underset{\sim}{A}\restriction n)\restriction n+1)$

Under this revised condition it is possible to prove the validity of 2.1 with exactly the same conditions upon the predicate as before. The proof of this is very similar to that of 2.14. This result does not get us much further in dealing with single recursions, but can sometimes be of use in mutual recursions.

e.g. $A \Leftarrow (a \rightarrow A) \,[]\, (b \rightarrow B) \,[]\, (c \rightarrow \underline{skip})$

$B \Leftarrow A;B$

this is not constructive in the sense of 2.9, but is in the sense of 2.21 with m=2 for all $\underset{\sim}{A}$,n

A more useful technique is (for mutual recursions) the concept of constructiveness relative to a partial order on Λ, the set of indices of the recursion. Define a partial order to be well founded if every subset of it contains some minimal elements (or equivalently if it contains no infinite descending chains). Suppose $<$ is a well founded partial order.

A function will be constructive relative to $<$ if it is non-destructive and at any point is constructive in all variables except those strictly less than the point in question.
If $F:P^{\Lambda} \to P^{\Lambda}$ this condition can be formally expressed:

2.22 $\forall \underset{\sim}{A}.\forall n.\forall \lambda.(F(\underset{\sim}{A})_\lambda)\restriction n+1 = (F(\underset{\sim}{A}^*)_\lambda)\restriction n+1$
where $A^*_\mu = A_\mu \restriction n+1$ if $\mu<\lambda$
$= A_\mu \restriction n$ if $\neg(\mu<\lambda)$.

A function will be non-destructive relative to $<$ if

2.23 $\forall \underset{\sim}{A}.\forall n.\forall \lambda.(F(\underset{\sim}{A})_\lambda)\restriction n+1 = (F(\underset{\sim}{A}^*)_\lambda)\restriction n+1$
where $A^*_\mu = A_\mu \restriction n+1$ if $\mu \leq \lambda$
$= A_\mu \restriction n$ if $\neg(\mu \leq \lambda)$

Notice that to be non-destructive a function must not, in some sense, work against the partial order by pulling information from high points to low points.

2.24 Suppose $\Lambda = \{0,1\}$ with partial order $0<1$
Then the function $F(A_0,A_1) = (a \to (A_0 \,[]\, A_1)\ ,\ b.A_0)$
is constructive,
the function $F(A_0,A_1) = ((a \to A_1)\,[]\,A_0\ ,\ A_1)$
is non-destructive,
but the function $F(A_0,A_1) = (A_1,A_0)$ is neither.

It is possible to prove a calculus of these definitions in very much the same manner as for the old kind:

2.25 Theorem
If $F,G:P^{\Lambda} \to P^{\Lambda}$ are functions and $<$ is a well-founded partial order on Λ then (relative to $<$)
a) If F & G are both non-destructive then so is FoG .
b) If F is constructive then it is non-destructive.
c) If F is constructive and G is non-destructive then both FoG and GoF are constructive.
d) If F is constructive then it has a unique fixed point.

proof
a) $F(G(\underset{\sim}{A})_\lambda)\restriction n+1 \supseteq F(G(\underset{\sim}{A}^*)_\lambda)\restriction n+1$ by monotonicity.
$F(G(\underset{\sim}{A})_\lambda)\restriction n+1 = F(\underset{\sim}{B}^*)_\lambda\restriction n+1$ where $B^*_\mu = G(\underset{\sim}{A}_\mu)\restriction n$ if $\neg(\mu \leq \lambda)$
$= G(\underset{\sim}{A}\restriction n)_\mu \restriction n$
$\subseteq G(\underset{\sim}{A}^*)_\mu \restriction n$ as $\underset{\sim}{A}^* \supseteq \underset{\sim}{A}\restriction n$
$\subseteq G(\underset{\sim}{A}^*)_\mu$

$$B^*_\mu = G(\underset{\sim}{A})_\mu \lceil n+1 \quad \text{if } \mu \leq \lambda$$
$$= G(\underset{\sim}{A}')_\mu \lceil n+1 \quad \text{where } A'_\gamma = A_\gamma \lceil n \quad \text{if } \neg(\gamma \leq \mu)$$
$$= A_\gamma \lceil n+1 \text{ if } \gamma \leq \mu$$

but $\gamma \leq \mu \Rightarrow \gamma \leq \lambda$ so $\underset{\sim}{A}' \subseteq \underset{\sim}{A}^*$

$$\Rightarrow \quad B^*_\mu \subseteq G(\underset{\sim}{A}^*)_\mu \lceil n+1 \subseteq G(\underset{\sim}{A}^*)_\mu$$

Thus $\underset{\sim}{B}^* \subseteq G(\underset{\sim}{A}^*)$

so $F(\underset{\sim}{B}^*)_\lambda \lceil n+1 \subseteq F(G(\underset{\sim}{A}^*))_\lambda \lceil n+1$

which completes the proof of part a)

b) This follows by monotonicity since the $\underset{\sim}{A}^*$ of 2.23 is greater than that of 2.22.

c) The proofs of both parts of this are almost identical to that of part a).

d) Suppose F has two distinct fixed points $\underset{\sim}{A}$ & $\underset{\sim}{B}$. As in the proof of 2.11 there must be some n such that $\underset{\sim}{A}\lceil n = \underset{\sim}{B}\lceil n$ but $\underset{\sim}{A}\lceil n+1 \neq \underset{\sim}{B}\lceil n+1$. Now $\underset{\sim}{A}\lceil n+1 \neq \underset{\sim}{B}\lceil n+1$ implies there must be some λs such that $(A_\lambda)\lceil n+1 \neq (B_\lambda)\lceil n+1$. Thus there is at least one λ which is minimal with respect to this property. For such a λ we must have:

$(A_\lambda)\lceil n+1 = (F(\underset{\sim}{A})\lambda)\lceil n+1$ (as $\underset{\sim}{A}$ is a fixed point of F)

$= (F(\underset{\sim}{A}^*)_\lambda)\lceil n+1$ where $A^*_\mu = A_\mu \lceil n$ if $\neg(\mu<\lambda)$

$(= B_\mu \lceil n$ as $A\lceil n = B\lceil n)$

$= A_\mu \lceil n+1$ if $\mu<\lambda$

$(= B_\mu \lceil n+1$ as λ is minimal)

[The above remarks show that $\underset{\sim}{A}^* = \underset{\sim}{B}^*$, the corresponding restriction of $\underset{\sim}{B}$.]

$= (F(\underset{\sim}{B}^*)_\lambda)\lceil n+1$

$= (F(\underset{\sim}{B})_\lambda)\lceil n+1$ (as F is constructive)

$= (B_\lambda)\lceil n+1$ (as $\underset{\sim}{B}$ is a fixed point of F)

which contradicts our choice of λ. Thus F must only have a single fixed point.

Note that in the case of Λ finite (or any other case where the partial order has a bound on the lengths of its ascending chains) this condition is implied by the earlier extension, for then by recursing a fixed number of times it is possible to refer only to $\underset{\sim}{A}\lceil n$ when calculating each $F^m(\underset{\sim}{A})_\mu \lceil n+1$.

For a general well founded partial order it is necessary to restrict the types of predicates which may be used in proofs.

2.26 Example

Let Λ = N, then the recursion

$$A_0 \Leftarrow a \rightarrow \underline{skip}$$

$$A_{n+1} \Leftarrow A_n$$

is constructive relative to the standard order on N, and the predicate $R(\underset{\sim}{A}) = \exists n.\ A_n = \underline{abort}$ is continuous under definition 2.12 and is clearly satisfiable. But $R(\underset{\sim}{A}) \Rightarrow R(F(\underset{\sim}{A}))$ since $A_n = \underline{abort} \Rightarrow F(\underset{\sim}{A})_{n+1} = \underline{abort}$ and R does not hold of fix(F). Thus rule 2.1 is not justified in this case.

2.27 Theorem

Suppose F is constructive relative to the well founded partial order $<$ on Λ and the predicate R on P^{Λ} is satisfiable. Then rule 2.1 is justified provided R has the form $\forall K.R_K(\underset{\sim}{A})$ $(K \in \mathcal{K}$, say) where each R_K is continuous and is independent of all components of $\underset{\sim}{A}$ except for a finite set $\Gamma_K \subseteq \Lambda$, and all the Γ_K are disjoint.

(i.e. each R_K is a predicate on finitely many components of $\underset{\sim}{A}$ and no component occurs in more than one R_K.)

proof (technical)

As in the proof of 2.14 set $\underset{\sim}{A}$ = fix(F). Since R is satisfiable there must be some $\underset{\sim}{B}$ s.t. $\underset{\sim}{B}\lceil o = \underset{\sim}{A}\lceil o$ and $R(\underset{\sim}{B})$, and since R is continuous, if we assume $\neg R(\underset{\sim}{A})$ there must be some n s.t. $\exists \underset{\sim}{B}^*.\ \underset{\sim}{B}^*\lceil n = \underset{\sim}{A}\lceil n \ \&\ R(\underset{\sim}{B}^*)$ and $\forall \underset{\sim}{B}.\ \underset{\sim}{B}\lceil n+1 = \underset{\sim}{A}\lceil n+1 \Rightarrow \neg R(\underset{\sim}{B})$.

If $\Gamma \subseteq \Lambda$ is non-empty, denote the set of its least elements by $\mu(\Gamma)$. Define a map from the ordinal numbers to $\mathcal{P}(\Lambda)$ as follows:

$$f(0) = \emptyset$$

$$f(\alpha+1) = \mu(\Lambda - f(\alpha)) \cup f(\alpha)$$

$$f(\alpha) = \bigcup_{\beta<\alpha} f(\beta) \quad \text{if } \alpha \text{ is a limit ordinal}$$

This map is clearly monotonic, and is strictly monotonic unless $\Lambda = f(\alpha)$ for some α. But a strictly monotonic function is one-one, so under the assumption that $\forall \alpha.\ f(\alpha) \neq \Lambda$ we have produced a 1-1 function from On to the set $\mathcal{P}(\Lambda)$. This is impossible by Hartog's theorem (see for example Enderton p195) so we must have $\Lambda = f(\alpha^*)$ for some α^*.

Claim that each $f(\alpha)$ $(\alpha \leq \alpha^*)$ satisfies $\lambda \in f(\alpha)\ \&\ \mu < \lambda \Rightarrow \mu \in f(\alpha)$.

This true if $\alpha = 0$.

Assume true of α and that $\lambda \in f(\alpha+1)\ \&\ \mu < \lambda$

if $\lambda \in f(\alpha)$ then we must have $\mu \in f(\alpha)$ by assumption

if $\lambda \in \mu(\Lambda - f(\alpha))$ then $\mu \notin f(\alpha)$ would contradict the minimality of λ in $(\Lambda - f(\alpha))$

hence in either case $\mu \in f(\alpha) \subseteq f(\alpha+1)$, so the result is true of $\alpha+1$.

If α is a limit ordinal then if we assume the result true of all $\beta < \alpha$ the result for α follows immediately since if $\mu < \lambda$ then

$$\lambda \in f(\alpha) \Rightarrow \exists \beta \text{ s.t. } \lambda \in f(\beta)\ \&\ \beta < \alpha$$
$$\Rightarrow \mu \in f(\beta) \subseteq f(\alpha)$$

Thus by transfinite induction the result holds of all $\alpha \leq \alpha^*$.

Claim next that for each $\alpha \leq \alpha^*$ there is some $\underset{\sim}{B} \in P^{\Lambda}$ satisfying

$$\underset{\sim}{B}\lceil n = \underset{\sim}{A}\lceil n \ \&\ \lambda \in f(\alpha) \Rightarrow B_\lambda \lceil n+1 = A_\lambda \lceil n+1 \ \&\ R(\underset{\sim}{B})$$

This is true for $\alpha = 0$ (put $\underset{\sim}{B} = \underset{\sim}{B}^*$)

Suppose $\underset{\sim}{B}$ satisfies the above for α, claim $F(\underset{\sim}{B})$ satisfies it for $\alpha+1$.

Firstly $F(\underset{\sim}{B})\lceil n = F(\underset{\sim}{B}\lceil n)\lceil n = F(\underset{\sim}{A}\lceil n)\lceil n = F(\underset{\sim}{A})\lceil n = \underset{\sim}{A}\lceil n$

and also $R(F(\underset{\sim}{B}))$ holds since $R(\underset{\sim}{B})$ does.

Now $\lambda \in f(\alpha+1) \Rightarrow (F(\underset{\sim}{B})_\lambda)\lceil n+1 = (F(\underset{\sim}{C})_\lambda)\lceil n+1$

where $C_\mu = B_\mu \lceil n+1$ if $\mu < \lambda$

$C_\mu = B_\mu \lceil n$ if $\neg(\mu < \lambda)$

But $B_\mu \lceil n = A_\mu \lceil n$ by assumption

and we established above that $\mu < \lambda \in f(\alpha+1)$ implies $\mu \in f(\alpha)$, so $\mu < \lambda \Rightarrow B_\mu \lceil n+1 = A_\mu \lceil n+1$.

Thus $C_\mu = A_\mu \lceil n+1$ if $\mu < \lambda$

$= A_\mu \lceil n$ if $\neg(\mu < \lambda)$

Hence $(F(\underset{\sim}{B})_\lambda)\lceil n+1 = (F(\underset{\sim}{A})_\lambda)\lceil n+1$ as F is constructive

$= \underset{\sim}{A}_\lambda \lceil n+1$

Finally suppose α is a limit ordinal and that for each $\beta < \alpha$ there is some $\underset{\sim}{B}^\beta$ satisfying the above. For each $\kappa \in K$ we can write the set Γ_κ as $\{\lambda_1, \lambda_2, \ldots, \lambda_r, \lambda_{r+1}, \ldots, \lambda_s\}$ where $\lambda_1 - \lambda_r \in f(\alpha)$ and $\lambda_{r+1} - \lambda_s \notin f(\alpha)$

As there are only finitely many "λ_j"s in $f(\alpha)$ there must be some $\beta < \alpha$ s.t. $f(\beta) \cap \Gamma_\kappa = f(\alpha) \cap \Gamma_\kappa$. Put $B_{\lambda_j} = (B^\beta)_{\lambda_j}$ for $j = 1 - s$.

If we do this for every $\kappa \in K$ there will be no clashes in the B_λ's so derived since the Γ_κ are disjoint.

Formally define $\underset{\sim}{B}$ as follows:

$B_\lambda = (B^\beta)_\lambda$ if $\exists K.\, \lambda \in \Gamma_K$ (β dependent on K)

$= A_\lambda$ otherwise .

Each R_K holds of the $\underset{\sim}{B}$ so defined since it holds of $\underset{\sim}{B}^\beta$ and is independent of all $\{B_\lambda | \lambda \in \Gamma_K\}$. Also $\underset{\sim}{B}\lceil n = \underset{\sim}{A}\lceil n$ since this is true of each $\underset{\sim}{B}^\beta$ by assumption. Finally if $\lambda \in f(\alpha)$ then

$\exists K.\, \lambda \in \Gamma_K \Rightarrow B_\lambda = (B^\beta)_\lambda$

$\Rightarrow B_\lambda\lceil n+1 = (B^\beta)_\lambda\lceil n+1$

$= A_\lambda\lceil n+1$ as $\lambda \in f(\beta)$ and B^β satisfies our hypothesis for β.

otherwise $B_\lambda = A_\lambda$

$\Rightarrow B_\lambda\lceil n+1 = A_\lambda\lceil n+1$

This completes the proof that the above $\underset{\sim}{B}$ satisfies the condition for α.

Hence by transfinite induction on $\alpha \leq \alpha^*$ there must be some $\underset{\sim}{B}$ corresponding to α^*. But then this $\underset{\sim}{B}$ satisfies $R(\underset{\sim}{B})$ and $\underset{\sim}{B}\lceil n+1 = \underset{\sim}{A}\lceil n+1$ (since $\Lambda = f(\alpha^*)$) which contradicts the assumption that $\forall \underset{\sim}{B}.\ \underset{\sim}{B}\lceil n+1 = \underset{\sim}{A}\lceil n+1 \Rightarrow \neg R(\underset{\sim}{B})$.

This result admits rather easier proofs in less general cases, such as predicates of the form $\forall \lambda.R_\lambda(A_\lambda)$ and also when each $\lambda \in \Lambda$ has some limit (dependent on λ) in N to the lengths of chains ascending to it.

Below is a simple example involving a finite partial order. See 2.3 for a more general example.

2.28 Example (after David Park)

Define four processes by mutual recursion:

$A_0 \Leftarrow (a \rightarrow A_1) \,[]\, (b \rightarrow A_2)$

$A_1 \Leftarrow (a \rightarrow A_3) \,[]\, A_0$

$A_2 \Leftarrow A_0 \,[]\, (b \rightarrow A_3)$

$A_3 \Leftarrow A_2 \,[]\, A_1$

It is easily verified that these equations define a recursion which is constructive relative to the standdard order on $\{0,1,2,3\}$. The predicate $\forall i.C_i = B$, where B is defined $B \Leftarrow (a \rightarrow B) \,[]\, (b \rightarrow B)$ is clearly satisfiable and of the correct form, so the rule is justified in this case.

Assume $R(\underset{\sim}{C})$

Then $F(\underset{\sim}{C})_0 = (a \to C_1) \sqcap (b \to C_2)$

$= (a \to B) \sqcap (b \to B)$ by assumption

$= B$ by definition of B

$F(\underset{\sim}{C})_1 = (a \to C_3) \sqcap C_0$

$= (a \to B) \sqcap B$ by assumption

$= (a \to B) \sqcap ((a \to B) \sqcap (b \to B))$ by definition of B

$= (a \to B) \sqcap (b \to B)$ by properties of $\sqcap$

$= B$ by definition of B

(case 2 is almost identical to case 1)

$F(\underset{\sim}{C})_3 = C_1 \sqcap C_2$

$= B \sqcap B$ by assumption

$= B$

Hence $R(F(\underset{\sim}{C}))$ holds, so we can infer $\forall i.A_i = B$.

The next extension to our rule will allow us, amongst other things, to deal with "partial predicates" which are independent of one or more components of the mutual recursion.

2.29 Theorem

Suppose F is a constructive function (possibly relative to some partial order) and H is non-destructive. $(F,H:P^{\wedge} \to P^{\wedge})$ Suppose further that $H(fix(F)) = fix(F)$, then

(i) HoF & FoH are both constructive

(ii) $fix(HoF) = fix(FoH) = fix(F)$

(iii) If R is continuous (or satisfies the conditions of 2.27 in the $<$ case) and satisfiable then

$(\forall \underset{\sim}{A}.\ R(\underset{\sim}{A}) \Rightarrow R(H(F(\underset{\sim}{A})))) \Rightarrow R(fix(F))$ and

$(\forall \underset{\sim}{A}.\ R(\underset{\sim}{A}) \Rightarrow R(F(H(\underset{\sim}{A})))) \Rightarrow R(fix(F))$

proof

(i) is just a restatement of 2.11(c) and 2.25(c)

(ii) We have $H(F(fix(F))) = F(H(fix(F))) = fix(F)$ by assumption, so fix(F) is a fixed point of HoF and of FoH. But by (i) and 2.11(d) or 2.25(d) these functions have unique fixed points, so the result follows.

(iii) This follows since, by (ii), anything which is true of fix(HoF) or fix(FoH) is also true of fix(F)

This result has the following application to the partial predicates described on the last page:

2.30 Theorem

Suppose $F:P^{\wedge} \to P^{\wedge}$ is a constructive function and that R_1 & R_2 are two predicates on $P^{\wedge}$ which satisfy the following conditions:

a) R_1 is continuous and satisfiable and is independent of some subset Γ of $\wedge$.

b) R_2 has the form $R(\underset{\sim}{A}) \equiv ((\underset{\sim}{A}\lceil\Gamma) = H(\underset{\sim}{A}))$ for some non-destructive function $H:P^{\wedge} \to P^{\Gamma}$ which is constructive in its Γ-component (in a sense to be made clear below).

c) $R_2(fix(F))$ holds.

d) $\forall \underset{\sim}{A}.(R_1(\underset{\sim}{A}) \;\&\; R_2(\underset{\sim}{A})) \Rightarrow R_1(F(\underset{\sim}{A}))$

Then $R_1(fix(F))$ holds.

proof

Write every element of $P^{\wedge}$ in the form $(\underset{\sim}{A},\underset{\sim}{B})$, where $\underset{\sim}{A} \in P^{\wedge-\Gamma}$ & $\underset{\sim}{B} \in P^{\Gamma}$.

A function will now be constructive in its Γ-component if it satisfies $\forall \underset{\sim}{A}.\forall \underset{\sim}{B}.\forall n.H(\underset{\sim}{A},\underset{\sim}{B})\lceil n+1 = H(\underset{\sim}{A},\underset{\sim}{B}\lceil n)\lceil n+1$.

For any $\underset{\sim}{A} \in P^{\wedge-\Gamma}$ we can now define a function $G_{\underset{\sim}{A}}: P^{\Gamma} \to P^{\Gamma}$ by $G(\underset{\sim}{B}) = H(\underset{\sim}{A},\underset{\sim}{B})$. This $G_{\underset{\sim}{A}}$ is constructive and so has a unique fixed point. We can write $H^*(\underset{\sim}{A},\underset{\sim}{B}) = (\underset{\sim}{A}, fix(G_{\underset{\sim}{A}}))$. H^* is a non-destructive function, as will be shown in 2.36 and since each $G_{\underset{\sim}{A}}$ has a unique fixed point we must have $H^*(fix(F)) = fix(F)$ (condition (c) above implies that $fix(F)\lceil\Gamma$ is a fixed point of $G_{\underset{\sim}{C}}$ where $\underset{\sim}{C} = fix(F)\lceil(\wedge - \Gamma)$).

Now apply 2.29 with F as above, $H = H^*$ and $R = R_1$ (if the FoH case is used) or $R = R_1 \;\&\; R_2$ (if the HoF case is used). Note that H^* does not alter the truth of R_1 since it is the identity on all components upon which R_1 depends.

Informally this result allows us to make certain additional assumptions about recursive calls which lie outside the domain of the predicate we are trying to prove, though we must be able to justify these assumptions independently.

In the example we will see shortly R_2 will have the form $\gamma \in \Gamma \Rightarrow A_\gamma = fix(F)_\gamma$ (so that H is a constant function).

Other suitable "H"s for use in 2.29 are F^k, $H(A) = A \sqcap B$ for any $B \sqsubseteq fix(F)$ and the device below for taking fixpoints one at a time.

2.31 If $F:P^{\Lambda} \to P^{\Lambda}$ is a constructive function defining a mutual recursion and $\Lambda = \Gamma \cup \Delta$ is a partition of the indexing set then we can write every element of P^{Λ} in the form $(\underset{\sim}{A},\underset{\sim}{B})$, where $\underset{\sim}{A} \in P^{\Gamma}$ & $\underset{\sim}{B} \in P^{\Delta}$ and there are some constructive $G:P^{\Lambda} \to P^{\Gamma}$ & $K:P^{\Lambda} \to P^{\Delta}$ s.t. $F(\underset{\sim}{A},\underset{\sim}{B}) = (G(\underset{\sim}{A},\underset{\sim}{B}),K(\underset{\sim}{A},\underset{\sim}{B}))$. A suitable H to use with 2.29 is $H(\underset{\sim}{A},\underset{\sim}{B}) = (\underset{\sim}{A},\mathrm{fix}(\lambda \underset{\sim}{B}.K(\underset{\sim}{A},\underset{\sim}{B})))$. This could be useful if one wanted to do induction on the variables of a recursion one at a time. This topic of iterated fixed points will be treated in more detail later.

2.32 Example (illustrating 2.27 & 2.30)
For some $T \subseteq \Sigma$ take $\Lambda = T^* - \{\langle\rangle\}$.
We seek to model "stacks" which terminate successfully as soon as they first become empty.

Define $S_{\langle a\rangle} \Leftarrow (!a \to \underline{\mathrm{skip}}) \,[]\, (?b{:}T \to S_{\langle ba\rangle})$ $\quad a \in T$

$S_{\langle a\rangle w} \Leftarrow (!a \to S_w) \,[]\, (?b{:}T \to S_{\langle ba\rangle w})$ $\quad a \in T,\ w \in \Lambda$

A second version of this:

$Q_{\langle a\rangle} \Leftarrow (!a \to \underline{\mathrm{skip}}) \,[]\, (?b{:}T \to Q_{\langle ba\rangle})$ $\quad a \in T$

$Q_{w\langle a\rangle} \Leftarrow Q_w ; Q_{\langle a\rangle}$

Theorem: $w \in \Lambda \Rightarrow S_w = Q_w$
In proving this we first show that $a \in T$ & $w \in \Lambda \Rightarrow S_{w\langle a\rangle} = S_w ; S_{\langle a\rangle}$.
This is done using 2.30, the partial predicate used being $R_1(\underset{\sim}{A}) \equiv (\forall a \in T.\forall w \in \Lambda.A_{w\langle a\rangle} = S_w ; S_{\langle a\rangle})$ (this is independent of $A_{\langle a\rangle}, a \in T$). The secondary predicate we use is $R_2(\underset{\sim}{A}) \equiv (\forall a \in T.\ A_{\langle a\rangle} = S_{\langle a\rangle})$ which arises from the constant $H(\underset{\sim}{A})_{\langle a\rangle} = S_{\langle a\rangle}$ $(a \in T)$. By construction the $\underset{\sim}{S}$-recursion is constructive, R_1 is satisfiable and continuous and $R_2(\underset{\sim}{S})$ holds.
Thus it only remains to show clause (d) of 2.30.
Assume $R_1(\underset{\sim}{A})$ & $R_2(\underset{\sim}{A})$
Then $F(\underset{\sim}{A})_{\langle ab\rangle} = (!a \to A_{\langle b\rangle}) \,[]\, (?c{:}T \to A_{\langle cab\rangle})$ $\quad a,b \in T$

$= (!a \to S_{\langle b\rangle}) \,[]\, (?c{:}T \to S_{\langle ca\rangle} ; S_{\langle b\rangle})$ by assumption

$= ((!a \to \underline{\mathrm{skip}}) \,[]\, (?c{:}T \to S_{\langle ca\rangle})) ; S_{\langle b\rangle}$

$= S_{\langle a\rangle} ; S_{\langle b\rangle}$ as desired (by definition of $\underset{\sim}{S}$)

$F(\underset{\sim}{A})_{\langle ba\rangle w\langle a\rangle} = (!b \to A_{w\langle a\rangle}) \,[]\, (?c{:}T \to A_{\langle cb\rangle w\langle a\rangle})$ $\quad (a,b \in T,\ w \in \Lambda)$

$= (!b \to S_w ; S_{\langle a\rangle}) \,[]\, (?c{:}T \to S_{\langle cb\rangle w} ; S_{\langle a\rangle})$ by assumption

$= ((!b \to S_w) \,[]\, (?c{:}T \to S_{\langle cb\rangle w})) ; S_{\langle a\rangle}$

$= S_{\langle b\rangle w} ; S_{\langle a\rangle}$ as desired

This establishes $R_1(F(\underset{\sim}{A}))$, so we can infer $R_1(\underset{\sim}{S})$ by 2.30.

Having established this result, we can now prove $\forall w.Q_w = S_w$ by induction on $\underset{\sim}{Q}$. Observe that the $\underset{\sim}{Q}$-recursion is constructive relative to the partial order induced by the length of $w \in \Lambda$. Let R' be the predicate $\forall w.A_w = S_w$. This is clearly satisfiable and is allowable in terms of 2.27.

Suppose $R'(\underset{\sim}{A})$, then if F is the function of the $\underset{\sim}{Q}$-recursion

$$\begin{aligned} F(\underset{\sim}{A})_{\langle a\rangle} &= (!a \to \underline{skip}) \,[]\, (?b{:}T \to A_{\langle ba\rangle}) \\ &= (!a \to \underline{skip}) \,[]\, (?b{:}T \to S_{\langle ba\rangle}) \quad \text{by assumption} \\ &= S_a \quad \text{by definition of } \underset{\sim}{S} \end{aligned}$$

$$\begin{aligned} F(\underset{\sim}{A})_{w\langle a\rangle} &= A_w ; A_{\langle a\rangle} \\ &= S_w ; S_{\langle a\rangle} \quad \text{by assumption} \\ &= S_{w\langle a\rangle} \quad \text{by earlier result} \end{aligned}$$

This establishes $R'(F(\underset{\sim}{A}))$, so we can infer $R'(\underset{\sim}{Q})$ as desired.

The following two extensions are also valid, their proofs are omitted in this model but are given in chapter 5 for the non-deterministic model.

2.33 Suppose $A \in \mathcal{P}(P^\wedge)$, define $A\lceil n = \{B\lceil n \mid B \in A\}$
Define a function $F: \mathcal{P}(P^\wedge) \to \mathcal{P}(P^\wedge)$ to be constructive if it satisfies $\forall A.\forall n.F(A)\lceil n+1 = F(A\lceil n)\lceil n+1$.
Suppose the predicate R on $P^\wedge$ is satisfiable and continuous.
Then $(\forall A'.(\forall B.(B \in A' \Rightarrow R(B))) \Rightarrow (\forall B.(B \in F(A') \Rightarrow R(B)))) \;\&\; (A \subseteq F(A))$ implies $(\forall B.(B \in A \Rightarrow R(B)))$.

This result is chiefly of use in proving general theorems about processes, for example the fundamental buffer theorem of chapter 5 (which is also true in this deterministic model).

2.34 Suppose $R_1 \ldots R_n$ are predicates which are individually satisfiable and all continuous but which may not have been shown to be simultaneously satisfiable. Suppose $F: P^\wedge \to P^\wedge$ is a constructive function which can be written as $F_i^* \circ D_n$ for $i=1,2,\ldots n$, where $D_n: P^\wedge \to (P^\wedge)^n$ and $F_i^*: (P^\wedge)^n \to P^\wedge$, $D_n(\underset{\sim}{A}) = (\underset{\sim}{A}, \underset{\sim}{A}, \ldots, \underset{\sim}{A})$.
Then if $\forall(\underset{\sim}{A}_1 \ldots \underset{\sim}{A}_n).\ R_1(\underset{\sim}{A}_1) \;\&\; \ldots \;\&\; R_n(\underset{\sim}{A}_n) \Rightarrow R_1(F_1^*(\underset{\sim}{A}^*)) \;\&\ldots\&\; R_n(F_n^*(\underset{\sim}{A}^*))$ (where $\underset{\sim}{A}^* = (\underset{\sim}{A}_1, \ldots, \underset{\sim}{A}_n)$) we can infer $R_1(\mathrm{fix}(F)) \&\ldots\& R_n(\mathrm{fix}(F))$.

Informally this rule represents
uctive hypotheses for each recursive call of a process. The method of defining F_i^* depends on which assumption we wish to make of each call in proving $R_i(F(\underset{\sim}{A}))$.

Iterated recursions

As has been indicated earlier it is possible to use subsiduary recursions within the body of a recursive definition.

e.g. $A = (a \to \underline{skip}) \,\square\, (b \to \text{rec } B.(b \to A;B))$

It is possible to analyse this type of construction in several ways. The first of these is the well-known result below which relates mutual recursions to iterated fixed points.

2.35 If $\Lambda = \Gamma \cup \Delta$ is a partition of Λ then as in 2.31 we can write elements of P^{Λ} and $P^{\Lambda} \to P^{\Lambda}$ as pairs $(\underset{\sim}{A},\underset{\sim}{B})$ and (G,K) respectively. It can quite easily be shown that

$$fix(G,K) = (fix(\lambda \underset{\sim}{A}.G(\underset{\sim}{A},fix(\lambda \underset{\sim}{B}.K(\underset{\sim}{A},\underset{\sim}{B})))),fix(\lambda \underset{\sim}{B}.K(fix(\lambda \underset{\sim}{A}.G(\underset{\sim}{A},\underset{\sim}{B})),\underset{\sim}{B})))$$

This gives a method for converting mutual recursions into iterated ones and vice-versa. Thus the example above is equal to A, where

$$A \Leftarrow (a \to \underline{skip}) \,\square\, (b \to B)$$
$$B \Leftarrow b \to A;B$$

It is also possible to analyse recursions with free process variables (such as rec B.(b → A;B) above), to decide if they are constructive, non-destructive etc. The need to do this has already arisen in 2.30 & 2.31.

2.36 Theorem (in the same notation as 2.35)

Suppose $F:(P^{\Gamma} \times P^{\Delta}) \to P^{\Gamma}$ is continuous and is non-destructive in its Γ component. Then the function $G(\underset{\sim}{B}) = fix(\lambda \underset{\sim}{A}.F(\underset{\sim}{A},\underset{\sim}{B}))$ is non-destructive if F is non-destructive in its Δ component and constructive if F is in its Δ component.

proof

We have the identity $fix(H) = \bigcup_{n=0}^{\infty} H^n(\underset{\sim}{\bot})$ for any continuous function H on a complete lattice of which $\underset{\sim}{\bot}$ is the minimal element.

Thus $G(\underset{\sim}{B}) = \bigcup_{n=0}^{\infty} K^n(\underset{\sim}{\bot})$ where $K(\underset{\sim}{A}) = F(\underset{\sim}{A},\underset{\sim}{B})$ and $\gamma \in \Gamma \Rightarrow \bot_{\gamma} = \underline{abort}$

We prove first the non-destructive case.

It is sufficient to show for arbitrary $\underset{\sim}{B}$ & n that $G(\underset{\sim}{B}\lceil n)\lceil n = G(\underset{\sim}{B})\lceil n$.

Claim that $\forall m.K^m(\underset{\sim}{\bot})\lceil n = L^m(\underset{\sim}{\bot})\lceil n$ where $L(\underset{\sim}{A}) = F(\underset{\sim}{A},\underset{\sim}{B}\lceil n)$

This is true when m=0 (both sides $= \underset{\sim}{\bot}$) so for induction assume true of m.

Then $K^{m+1}(\underset{\sim}{\bot})\lceil n = F(K^m(\underset{\sim}{\bot}),\underset{\sim}{B})\lceil n$

$= F(K^m(\underset{\sim}{\bot})\lceil n,\underset{\sim}{B}\lceil n)\lceil n$ as F is non-destructive

$= F(L^m(\underset{\sim}{\bot})\lceil n,\underset{\sim}{B}\lceil n)\lceil n$ by assumption

$= F(L^m(\underset{\sim}{\bot}),\underset{\sim}{B}\lceil n)\lceil n$ as F is non-destructive

$= L^{m+1}(\underset{\sim}{\bot})\lceil n$ as desired

Hence $K^m(\underset{\sim}{\bot})\lceil n = L^m(\underset{\sim}{\bot})\lceil n$ for all m.

Thus $G(\underset{\sim}{B})\lceil n = (\bigcup_{m=0}^{\infty} K^m(\underset{\sim}{\bot}))\lceil n$

$= (\bigcup_{m=0}^{\infty} (K^m(\underset{\sim}{\bot})\lceil n))$ as $\lceil n$ is continuous

$= (\bigcup_{m=0}^{\infty} (L^m(\underset{\sim}{\bot})\lceil n))$

$= (\bigcup_{m=0}^{\infty} L^m(\underset{\sim}{\bot}))\lceil n$

$= G(B\lceil n)\lceil n$

The proof of the constructive case is very similar (merely change some of the "n"s into "n+1"s).

It is now possible to strengthen 2.16 to include the possibility of iterated recursions.

2.37 Theorem

Suppose that the function $F:\hat{P} \to \hat{P}$ is such that each component of $F(\underset{\sim}{A})$ is a syntactic expression involving only expressions independent of all process variables, process variables, the combinators $a \to B$, $a?x:T \to B(x)$, $?x:T \to B(x)$, $B;C$, $B \,[]\, C$, $a.B$ and $(B_X \,||_Y C)$ and iterated recursions which bind all instances of process variables which are not A_j^s. Then provided that every (free) recursive call of an A_j is guarded directly (e.g. rec B.$(a \to A_j;B)$) or indirectly (e.g. $a \to$ rec B.$(A_j;B)$) the function F is constructive.

The proof of this is a straightforward structural induction using 2.11, 2.15, 2.36.

Operators involving hiding in their definition

The following two operators are both very useful in defining processes by recursion:

a) The Master/Slave operator $(A \| a::B)$ $(A,B \in P)$ is intended to model process A working in parallel with process B and using it as a slave. All communications with the slave will be in some $T \subseteq \Sigma$ which does not include $\surd$, and may either take the form "t" $(t \in T)$, "?t" $(t \in T)$ or "!t" $(t \in T)$. The last two of these will represent input and output respectively. The inputs of the slave B will be connected to the "a"outputs of A (all communications with the slave by A will be labelled "a") and the outputs of B will be connected to the "a"inputs of A. To avoid confusion we will insist that T, ?T and !T are all disjoint (T will be an inplicit parameter of this operator, as it will be of the next). Finally all communications of B must be with A and all communications with the slave are hidden.

$(A \| a::B) = ((A_{\Sigma} \|_{a.\Gamma} a.(swap!?(B))) / a.\Gamma)$

where $\Gamma = T \cup ?T \cup !T$ and swap is defined (on Σ, Σ^* and P):

$c \in \Sigma \Rightarrow$ swapab(c) = c if $c \notin (a.T \cup b.T)$

= b.d if c = a.d for some $d \in T$

= a.d if c = b.d for some $d \in T$

swapab(w) $(w \in \Sigma^*)$ is the natural elementwise extension of the definition on Σ.

swapab(A) $(A \in P)$ is the natural elementwise extension of the definition on Σ^*.

b) The pipe operator $(A \gg B)$ is intended to model the behaviour of process A accepting input from the environment, processing it in some way, passing its output to the inputs of B down a hidden channel, and B using this input to produce outputs to the environment. This is effected by transforming all outputs of A (in !T) to T and all the inputs of B (in ?T) to T, thus identifying them. All internal communication is then hidden by hiding T. Because of this device, for the operator to be meaningful, it is important that A and B cannot themselves use T for their communications.

$(A \gg B) = (strip!(A)_{T \cup ?T} \|_{T \cup !T} strip?(B)) / T$,

where strip is defined in a similar way to swap:

$c \in \Sigma \Rightarrow$ stripa(c) = c if $c \notin a.T$, = d if c = a.d for some $d \in T$

etc.

As was mentioned earlier the hiding operator A/X is not non-destructive. Thus the two derived operators $A \gg B$ and $(A \| a::B)$ are not obviously non-destructive either. This is unfortunate since both are useful tools in defining recursive processes.

e.g. $B^{\infty} \Leftarrow ?x:T \to (B^{\infty} \gg (!x \to B))$ (where $B \Leftarrow ?x:T \to !x \to B$)

represents an infinite capacity buffer

$S \Leftarrow (X \| a::S)$

where $X \Leftarrow ?x:T \to Y_x$

$Y_x \Leftarrow (!x \to Z) \,[]\, (?y:T \to a!x \to Y_y)$

$Z \Leftarrow (a?x:T \to Y_x) \,[]\, (?x:T \to Y_x)$

represents an infinite stack

We deal first with the master/slave operator $(A \| a::B)$. It is only possible to treat this as a function of its second variable, as it is certain (if $B \neq$ abort) to be "destructive" in A. It is not too difficult to see that it is not always constructive in B (e.g. $A = a!x \to a!x \to !x \to$ skip) but sometimes is constructive (e.g. if A never communicates with its slave $(A \| a::B) = A$ is a constant function of B). It turns out that the fundamental issue is the respective numbers of communications A makes with its environment and its slave. If A must at all times have made at least as many external as internal communications then $(A \| a::B)$ is non-destructive and if A must at all times (except initially) have made more external than internal communications then $(A \| a::B)$ is constructive. This is formally expressed in the following result:

2.38 Theorem

For $n \in Z$ (positive and negative integers) define the predicate

$C_n^a(A) = \forall w.\ w \in A \Rightarrow |w\lceil\Gamma| - |w\lceil a.\Gamma| \geq \min(n, |w\lceil\Gamma \cup a.\Gamma|)$

($\Gamma \subseteq \Sigma$ is assumed to be the alphabet of "atomic" communications, so that $\Sigma = \{\sqrt{}\} \cup \Gamma \cup \bigcup \{a.\Gamma \mid a \in N\}$ where N is the set of process names and $\forall a.\ \Gamma \cap a.\Gamma = \emptyset$.)

Note that in any process which satisfies C_n^a every trace must contain at least n+1 "Γ"s before the first "$a.\Gamma$".

a) $C_n^a(A)$ is a continuous predicate

b) $C_k^a(A) \Rightarrow \forall n. \forall B. (A \| a::B)\lceil n+k+1 = (A \| a::B\lceil n)\lceil n+k+1$

c) $C_k^a(A) \Rightarrow C_k^a(A \| b::B)$

proof

a) This follows immediately from 2.18(viii)

b) Define the notation "$(u,v) \Leftrightarrow w$ in $(A \| a::B)$" to mean
$u \in A$, $v \in B$, $u \restriction \Sigma - a.\Gamma = w$ & $swap?!(v) = strip_a(u \restriction a.\Gamma)$
so that $w \in (A \| a::B) \Leftrightarrow \exists u.\exists v.\ ((u,v) \Leftrightarrow w \text{ in } (A \| a::B))$.

Suppose $w \in (A \| a::B) \restriction n+k+1$. If $w = \langle\rangle$ then certainly $w \in (A \| a::B \restriction n)$
so suppose $w = w' \langle b \rangle$. There must be some u of minimal length
and corresponding v s.t. $(u,v) \Leftrightarrow w$ in $(A \| a::B)$.

By construction u has the form $u' \langle b \rangle$ for some u' and (by $C_k^a(A)$)
we have $|u' \restriction \Gamma| - |u' \restriction a.\Gamma| \geqslant \min(k, |u' \restriction \Gamma \cup a.\Gamma|)$

$= |u' \restriction (\Sigma - a.\Gamma)| - |u' \restriction a.\Gamma| \geqslant \min(k, |u'|)$

$= |u \restriction (\Sigma - a.\Gamma)| - |u \restriction a.\Gamma| \geqslant \min(k+1, |u|)$ (add one to both sides)

so <u>either</u> $|u| < k+1$ and $|u \restriction (\Sigma - a.\Gamma)| - |u \restriction a.\Gamma| \geqslant |u|$

$\Rightarrow u \restriction a.\Gamma = \langle\rangle$ & $u = w$

$\Rightarrow (u, \langle\rangle) \Leftrightarrow w$ in $(A \| a::B \restriction n)$ (as certainly $\langle\rangle \in B \restriction n$)

<u>or</u> $|u| \geqslant k+1$ and $|u \restriction (\Sigma - a.\Gamma)| - |u \restriction a.\Gamma| \geqslant k+1$

$= |w| - |v| \geqslant k+1$

$= |v| \leqslant |w| - (k+1)$

$= |v| \leqslant (n+k+1) - (k+1)$ as $|w| \leqslant n+k+1$

Thus in either case $w \in (A \| a::B \restriction n)$

Hence $(A \| a::B) \restriction n+k+1 \subseteq (A \| a::B \restriction n) \restriction n+k+1$, the reverse inclusion
following by monotonicity.

c) Suppose $w \in (A \| b::B)$ and $C_k^a(A)$ where $b \neq a$ (the case $b = a$
is trivial). Then there are some u,v s.t. $(u,v) \Leftrightarrow w$ in $(A \| b::B)$.
But then $w \restriction \Gamma = u \restriction \Gamma$, $w \restriction a.\Gamma = u \restriction a.\Gamma$ and $w \restriction \Gamma \cup a.\Gamma = u \restriction \Gamma \cup a.\Gamma$
so $|w \restriction \Gamma| - |w \restriction a.\Gamma| = |u \restriction \Gamma| - |u \restriction a.\Gamma| \geqslant \min(k, |u \restriction \Gamma \cup a.\Gamma|)$

$\min(k, |w \restriction \Gamma \cup a.\Gamma|)$ as desired

Methods for the determining of these conditions will be given
in chapter 6.

2.39 <u>Examples</u>

It is possible to model ZERO (= $COUNT_0$) (of 2.19) using $(A \| a::B)$.

$Z \Leftarrow (X \| a::Z)$
where $X \Leftarrow (iszero \rightarrow X) \,[]\, (up \rightarrow Y)$
$Y \Leftarrow (up \rightarrow a.up \rightarrow Y)$
$[]\,(down \rightarrow (a.down \rightarrow Y)$
$[]\,(a.iszero \rightarrow X)\ )$

It is an easy induction to show that X satisfies C_1^a and Y satisfies C_0^a (use of 2.1 on their joint definition). Thus the recursion $Z \Leftarrow (X \,||\, a::Z)$ is constructive.

Claim that $COUNT_0 = (X \,||\, a::COUNT_0)$

and that $COUNT_{n+1} = (Y \,||\, a::COUNT_n) \quad n \in N$

This can be proved by induction on the definition of COUNT.

The predicate R on P^N defined

$$R(\underset{\sim}{C}) \equiv C_0 = (X \,||\, a::COUNT_0) \;\&\; \forall n.C_{n+1} = (Y \,||\, a::COUNT_n)$$

is clearly satisfiable and continuous. Attempt to prove it true of COUNT .

Assume $R(\underset{\sim}{C})$ and that F is the (constructive) function of the COUNT recursion.

Then
$$\begin{aligned}(X \,||\, a::COUNT_0) &= ((iszero \to X) \,[]\, (up \to Y) \,||\, a::COUNT_0)\\ &= iszero \to (X \,||\, a::COUNT_0)\\ &\quad []\, up \to (Y \,||\, a::COUNT_0)\\ &= iszero \to C_0\\ &\quad []\, up \to C_1 \qquad \text{by assumption}\\ &= F(\underset{\sim}{C})_0 \qquad \text{as desired}\end{aligned}$$

$$\begin{aligned}(Y \,||\, a::COUNT_0) &= ((up \to a.up \to Y) \,[]\, (down \to Y^*) \,||\, a::COUNT_0)\\ &\quad (\text{where } Y^* = (a.iszero \to X) \,[]\, (a.down \to Y)\;)\\ &= up \to (a.up \to Y \,||\, a::COUNT_0)\\ &\quad []\, down \to (Y^* \,||\, a::COUNT_0)\\ &= up \to (a.up \to Y \,||\, a::(up \to COUNT_1 \,[]\, iszero \to COUNT_0))\\ &\quad []\, down \to (Y^* \,||\, a::(iszero \to COUNT_0 \,[]\, up \to COUNT_1))\\ &= up \to (Y \,||\, a::COUNT_1)\\ &\quad []\, down \to (X \,||\, a::COUNT_0)\\ &= up \to C_2\\ &\quad []\, down \to C_0 \qquad \text{by assumption}\\ &= F(\underset{\sim}{C})_1 \qquad \text{as desired}\end{aligned}$$

The final proof of $(Y \,||\, a::COUNT_{n+1}) = F(\underset{\sim}{C})_{n+2}$ is very similar and is ommitted. The manipulations of the $(A \,||\, a::B)$ operator used above are formally justified in chapter 6. We have thus established $R(F(\underset{\sim}{C}))$, so we can infer $R(COUNT)$.

It is now simple to show $Z = COUNT_0$, for if S is the predicate "$= COUNT_0$" we have
$$\begin{aligned}S(U) &\Rightarrow (X \,||\, a::U)\\ &= (X \,||\, a::COUNT_0) = COUNT_0\\ &\Rightarrow S(X \,||\, a::U)\end{aligned}$$

and the result follows since the Z-recursion is constructive.

The intuition behind this definition of Z is that on receiving an "up" or a "down" other than the first up or last down Z passes it on to its slave, which is a copy of itself. It knows when its slave has been brought back to zero by the slave's willingness to communicate "iszero". One might try some alternative definitions, such as on every "up" or "down" other than the first sending it twice to the slave. This is what is intended in the example below.

$$Z^* \Leftarrow (X \,||\, a::Z^*)$$
$$\text{where } X^* \Leftarrow (up \to Y^*) \,\Box\, (iszero \to X^*)$$
$$Y^* \Leftarrow (up \to a.up \to a.up \to Y^*)$$
$$\Box\, (down \to (a.iszero \to X^*)$$
$$\Box\, (a.down \to a.down \to Y^*) \quad)$$

It is indeed possible to show (as in the previous example) that $COUNT_0$ is a fixpoint of this recursion. One would use the earlier proof as a model for showing $(X^* \,||\, a::COUNT_0) = COUNT_0$ and $\forall n.(Y^* \,||\, a::COUNT_{2n}) = COUNT_{2n+2}$. But this X^* does not satisfy C_0^a (nor does it satisfy C_z^a for any $z \in Z$) so we are not justified in thinking that this recursion is constructive. This is brought out by the fact that it does have several other fixed points:

e.g. $Z_1 \Leftarrow (iszero \to Z_1) \,\Box\, (up \to (down \to Z_1) \,\Box\, (up \to MANY))$

where $MANY \Leftarrow (up \to MANY) \,\Box\, (down \to MANY)$

(This Z_1 can be regarded as only understanding clearly the ideas "zero" and "one" after which it gets confused.)

$Z_2 \Leftarrow (iszero \to Z_2) \,\Box\, (up \to (down \to Z_2) \,\Box\, (up \to \underline{abort} \,\Box\, down \to \underline{abort}))$

which is the minimal fixed point.

The conditions for the $A \gg B$ operator to be non-destructive in either of its variables are intuitively very similar to those for $(A \,||\, a::B)$. This operator will be treated at some length in the non-deterministic model (chapter 5) so we just state the conditions here.

For $w \in \Sigma^*$ let ins(w) and outs(w) have the same definitions as in 2.20.

2.40 Theorem

Suppose that $C \subseteq (?T \cup !T)^*$, then

a) If $w \in C \Rightarrow |outs(w)| \geq |ins(w)|$ then $(A \gg C)$ is a non-destructive function of A.

b) If $w \in C \Rightarrow |outs(w)| \leq |ins(w)|$ then $(C \gg A)$ is a non-destructive function of A.

The proof of this for the non-deterministic model will be found in 5.30.

By this result it can be shown without too much trouble that if C is any buffer (in either the strong or the weak sense of 2.20) then so is the process recursively defined

$$A \Leftarrow ?x{:}T \rightarrow (C \gg (!x \rightarrow A))$$

as any buffer plainly satisfies condition (b) above. There wi will be several worked examples similar to this in chapter 5.

Alternative conditions for the validity of 2.1

As has been indicated earlier, there are other pairs of conditions on functions and predicates which guarantee validity of 2.1 along with satisfiablity. Firstly we can vary our interpretation of the restriction operator $\lceil n$. The only thing we assume about this operator in the proof of 2.16 is that for all $\underset{\sim}{A},\underset{\sim}{B}\in P^{\wedge}$ we have $A\lceil o = B\lceil o$. Thus if for any class of operators $\{\lceil n \mid n\in N\}$ we define continuity of predicates and constructiveness of functions we can prove a theorem which corresponds to 2.16 provided that the above condition holds. Exactly how useful this result is will clearly depend on the resulting classes of continuous predicates and constructive functions, and the difficulty of proving a given recursion to be constructive. Provided that we wish equality to be continuous we must insist $(\forall n.\underset{\sim}{A}\lceil n = \underset{\sim}{B}\lceil n) \Rightarrow \underset{\sim}{A} = \underset{\sim}{B}$ and to give any "meaning" to constructive functions we must require that $(\underset{\sim}{A}\lceil n)\lceil m = \underset{\sim}{A}\lceil \min(n,m)$. Examples of such constructions are given below:

We could turn the old definition upside down and make

$A\lceil n = \{w\in A \mid |w| < n\} \cup \{wv \mid w\in A \ \& \ |w| = n\}$

or restrict attention to a subset of Σ

$A\lceil n = \{w\in A \mid |w\sqcap| < n\}$

or nest allowable symbols

$A\lceil n = \{w\in A \mid w\in \Gamma_n^*\}$ where $\Gamma_0 = \emptyset$, $\Gamma_n \subseteq \Gamma_{n+1}$ and $\bigcup_n \Gamma_n = \Sigma$

Lastly we examine a theory which is in several ways more elegant than that which has gone before. In some ways it represents a highest common factor (or greatest lower bound) of the theories which involve a restriction operator and its study therefore gives us insight into these. It is rarely, however, that we will wish to apply it directly to a problem since it usually turns out that the easiest method is through one of the less abstract theories. Therefore it is stated here only in terms of single recursion since this makes the notation rather easier and the proofs easier to understand. Also (unlike the earlier conditions) it does not seem to generalise naturally to the non-deterministic model, because it relies critically on the existence of a top element.

We firstly observe that any reasonable (under the above conditions) restriction operator will give rise to a definition of constructiveness which implies unique fixed points (like 2.11).

In many of the proofs we have performed using 2.1 it would have been sufficient to know that the recursive equations involved had a unique fixed point (for example in the cases where we showed that another process satisfied the equation, so the two must be equal). It is possible to formulate a condition on predicates which, along with UFP (unique fixed point) guarantees the validity of 2.1 . We have already shown that a constructive function satisfies UFP, so we would expect this condition to be at least as strong as the old one. That the condition needs to be strictly stronger is demonstrated by the following example

2.41 Of the recursion $A \Leftarrow (a.A_{\Sigma} - ||_{\Sigma} a.A)$ (which has UFP abort)
we have $(B \neq \underline{\text{abort}} \wedge B \neq \underline{\text{skip}}) \Rightarrow (F(B) \neq \underline{\text{abort}} \wedge F(B) \neq \underline{\text{skip}})$
so by applying 2.1 we could deduce $A \neq \underline{\text{abort}}$

A predicate was weakly continuous if its truth of a sequence of processes implied the truth of its limit (2.13). We only looked, however, at a specific sort of convergence of processes. It is possible to define a more general sort of convergence.

2.42 Suppose $\langle A_i \mid i \in N \rangle$ is a sequence of processes

Define $\text{limsup}(A_i) = \bigcap_{i=0}^{\infty} (\bigcup_{j=i}^{\infty} (A_j))$

and $\text{liminf}(A_i) = \bigcup_{j=0}^{\infty} (\bigcap_{i=j}^{\infty} (A_i))$

(these correspond to the usual notions in real analysis)

Say that a sequence $\langle A_i \rangle$ is <u>convergent</u> to limit B
iff $\text{limsup}(A_i) = \text{liminf}(A_i) = B$ (and that then $\lim(A_i) = B$).

$\text{Limsup}(A_i)$ is the set of traces which are contained in infinitely many A_i and $\text{liminf}(A_i)$ is the set of traces contained in all but finitely many A_i.

2.43 <u>Lemma</u>

a) If $\langle A_i \rangle$ is a sequence of processes then both $\text{limsup}(A_i)$ and $\text{liminf}(A_i)$ are processes.

b) $\text{limsup}(A_i) \supseteq \text{liminf}(A_i)$

c) If $\forall i.\ B_i \lceil i = A \lceil i$ then the B_i converge to A .

<u>proof</u>

a) follows since the space of processes is closed under both infinite intersections and infinite unions.

b) Clearly $i \leqslant j \Rightarrow \bigcup_{k=i}^{\infty}(A_k) \supseteq \bigcap_{k=j}^{\infty}(A_k)$

$\Rightarrow \bigcup_{k=i}^{\infty}(A_k) \supseteq \bigcup_{j=i}^{\infty}(\bigcap_{k=j}^{\infty}(A_k)) = \bigcup_{j=o}^{\infty}(\bigcap_{k=j}^{\infty}(A_k))$

$\Rightarrow \bigcap_{i=o}^{\infty}(\bigcup_{k=i}^{\infty}(A_k)) \supseteq \bigcup_{j=o}^{\infty}(\bigcap_{k=j}^{\infty}(A_k))$ as desired

c) $w \in A \;\&\; k \geqslant w \Rightarrow w \in B_k$ (by definition of $A\lceil k$)

$\Rightarrow w \in \bigcup_{k=i}^{\infty}(B_k)$ for each i

$\Rightarrow w \in \text{limsup}(B_k)$

Hence $\text{limsup}(B_k) \subseteq A$ (*)

Also $w \in \text{liminf}(B_k) \Rightarrow \exists i.\ w \in \bigcap_{j=i}^{\infty}(B_j)$

$\Rightarrow w \in B_k$ where $k = \max(i, |w|)$

$\Rightarrow w \in B_k\lceil k$

$\Rightarrow w \in A\lceil k$

$\Rightarrow w \in A$

Hence $A \subseteq \text{liminf}(B_k)$ (+)

But now (*) & (+) & (b) give the desired result.

Now define a predicate to be <u>strongly continuous</u> if it satisfies:

2.44 For every convergent sequence $\langle A_i \rangle$, $\forall i.R(A_i) \Rightarrow R(\lim(A_i))$

2.45 <u>Theorem</u>

a) If a predicate is strongly continuous then it is weakly continuous.

b) If Σ is finite and a predicate is weakly continuous then it is strongly continuous.

<u>proof</u>

a) Suppose R is strongly continuous and that A_i is a sequence of processes satisfying $\forall i.R(A_i)$ and $\forall i.(A_i)\lceil i = B\lceil i$
Then by 2.43(c) $\langle A_i \rangle$ is a convergent sequence with limit B so R(B) follows by the strong continuity of R.

b) Suppose R is weakly continuous and Σ is finite.
Then if $\langle A_i \rangle$ is any convergent sequence with limit B s.t. $\forall i.R(A_i)$ we have:

$w \in B \Rightarrow \exists j.\ i \geqslant j \Rightarrow w \in A_j$ (as w liminf(A_j))

$w \notin B \Rightarrow \exists j.\ i \geqslant j \Rightarrow w \notin A_j$ (as w liminf(A_j))

Since Σ is finite, for any fixed n there is only a finite number of elements of Σ^* with length n or less.

For each of these "w"s we can thus find a j_w s.t.

$$i \geqslant j_w \Rightarrow (w \in B \Leftrightarrow w \in A_i)$$

Hence if $j_n = \max \{ j_w \mid w \leqslant n \}$ we have

$$i \geqslant j_n \Rightarrow A_i \lceil n = B \lceil n$$

and in particular we thus have $(A_{j_n}) \lceil n = B \lceil n \ \& \ R(A_{j_n})$

Since we can carry out this procedure for each n these A_{j_n}s satisfy the left hand side of 2.13, so we can deduce R(B) by the weak continuity of R.

2.46 Theorem

The following predicates are all strongly continuous:

$R(A) \equiv$

(i)	$A = B$	
(ii)	$A \subseteq B$	
(iii)	$A \supseteq B$	
(iv)	$w \in A \Rightarrow p(w)$	where p is any predicate on Σ^*
(v)	$F(A) \subseteq B$	if F is continuous
(vi)	$F(A) \supseteq B$	if F is rev-continuous
(vii)	$R_1(F(A))$	if F is doubly continuous
(viii)	$\forall \gamma . R_\gamma(A)$	$(\gamma \in \Gamma)$
(ix)	$R_1(A) \vee R_2(A)$	,

where B is any constant process and R_1, R_2 & R_is are all strongly continuous, and a function is doubly continuous if it is both continuous and rev-continuous.

example proofs

(i) If $\langle A_i \rangle$ is any convergent sequence s.t. $\forall i . R(A_i)$ then $\forall i . A_i = B$. So certainly $\lim(A_i) = B$.

(v) Suppose $\langle A_i \rangle$ is a convergent sequence s.t. $\forall i . R(A_i)$. Set $A = \lim(A_i)$ and suppose E is any finite process $E \subseteq A$. For each $w \in E$ there is some j_w s.t. $i \geqslant j \Rightarrow w \in A_i$ (as $A = \text{liminf}(A_i)$) Set $j_E = \max \{ j_w \mid w \in E \}$, we then have $E \subseteq A_{j_E}$. But then $F(E) \subseteq F(A_{j_E}) \subseteq B$ and by continuity of F we have

$F(A) = \bigcup_{\substack{E \subseteq A \\ \text{finite}}} F(E) \subseteq B$ as desired.

(vii) The proof of this will be given later.

(ix) Suppose $\langle A_i \rangle$ is any convergent sequence s.t. $\forall i . R_1(A_i) \vee R_2(A_i)$. Then there is either an infinite subsequence satisfying R_1 or one satisfying R_2. It is easy to show that if $\langle B_i \rangle$ is a subsequence of $\langle A_i \rangle$ then $\text{liminf}(A_i) \subseteq \text{liminf}(B_i) \subseteq \text{limsup}(B_i) \subseteq \text{limsup}(A_i)$. Thus when $\langle A_i \rangle$ is convergent so must be $\langle B_i \rangle$ with the same limit.

Hence we either have an infinite subsequence (with the same limit) which satisfies R_1 ($\Rightarrow R_1(\lim(A_i))$ by continuity of R_1) or the same for R_2. We hence have $R_1(\lim(A_i)) \vee R_2(\lim(A_i))$ as desired.

2.47 Theorem

Suppose $F:P \to P$ is doubly continuous and has a unique fixed point and that R is a satisfiable and strongly continuous predicate. Then rule 2.1 is valid.

i.e $(\forall A.R(A) \Rightarrow R(F(A))) \Rightarrow R(fix(F))$

proof

Let A_o be such that $R(A_o)$ (by satisfiability)

Then $\bot \sqsubseteq A_o \sqsubseteq \top$ where $\bot$ = abort and $\top$ = run

Claim that for each n we have $R(A_n)$ and $F^n(\bot) \sqsubseteq A_n \sqsubseteq F^n(\top)$, where $A_n = F^n(A_o)$.

This is true for n=o by the above.

Suppose it true for n.

Then $R(A_{n+1})[= R(F(A_n))]$ follows by supposition on R&F.

Also we then have $F^n(\bot) \sqsubseteq A_n \sqsubseteq F^n(\top)$

$\Rightarrow F^{n+1}(\bot) \sqsubseteq F(A_n) \sqsubseteq F^{n+1}(\top)$ by monotonicity

$\Rightarrow F^{n+1}(\bot) \sqsubseteq A_{n+1} \sqsubseteq F^{n+1}(\top)$ as desired.

Since F is doubly continuous and has a unique fixed point we have

$$\bigsqcup_{n=o}^{\infty} F^n(\bot) = fix(F) = \bigsqcap_{n=o}^{\infty} F^n(\top) \quad .$$

As monotonic sequences both $F^n(\bot)$ and $F^n(\top)$ are convergent and have limit $fix(F)$.

We therefore have $fix(F) = limsup(F^n(\bot)) \sqsubseteq limsup(A_n)$ (by (*))

and $fix(F) = liminf(F^n(\top)) \sqsupseteq liminf(A_n)$

thus $fix(F) \sqsubseteq liminf(A_n) \sqsubseteq limsup(A_n) \sqsubseteq fix(F)$

so $\langle A_n \rangle$ converges to $fix(F)$

The result then follows by the strong continuity of R.

We now examine the way in which this theory fits in with the others we have already seen.

2.48 Lemma

Suppose $\langle A_n \rangle$ is a convergent sequence and that the function $F:P \to P$ is doubly continuous. Then the sequence $\langle F(A_n) \rangle$ is convergent with limit $F(\lim(A_n))$.

proof

By monotonicity we have for any $k \in N$

$$i \geq k \Rightarrow A_i \subseteq \bigcup_{j=k}^{\infty}(A_j) \quad \& \quad A_i \supseteq \bigcap_{j=k}^{\infty}(A_j)$$

$$\Rightarrow F(A_i) \subseteq F(\bigcup_{j=k}^{\infty}(A_j)) \ \& \ F(A_i) \supseteq F(\bigcap_{j=k}^{\infty}(A_j))$$

$$\Rightarrow \bigcup_{j=k}^{\infty}(F(A_j)) \subseteq F(\bigcup_{j=k}^{\infty}(A_j)) \ \& \ \bigcap_{j=k}^{\infty}(F(A_j)) \supseteq F(\bigcap_{j=k}^{\infty}(A_j))$$

Thus $\text{limsup}(F(A_j)) = \bigcap_{k=o}^{\infty}(\bigcup_{j=k}^{\infty}(F(A_j)))$

$$\subseteq \bigcap_{k=o}^{\infty}(F(\bigcup_{j=k}^{\infty}(A_j)))$$

$$\subseteq F(\bigcap_{k=o}^{\infty}(\bigcup_{j=k}^{\infty}(A_j))) \quad \text{by rev-continuity of F}$$

$$\subseteq F(\lim(A_n)) \quad \text{as } \langle A_n \rangle \text{ is convergent}$$

Also $\text{liminf}(F(A_j)) = \bigcup_{k=o}^{\infty}(\bigcap_{j=k}^{\infty}(F(A_j)))$

$$\supseteq \bigcup_{k=o}^{\infty}(F(\bigcap_{j=k}^{\infty}(A_j))) \quad \text{by the above}$$

$$\supseteq F(\bigcup_{k=o}^{\infty}(\bigcap_{j=k}^{\infty}(A_j))) \quad \text{by continuity of F}$$

$$\supseteq F(\lim(A_n))$$

We have thus shown $\text{limsup}(F(A_n)) \subseteq F(\lim(A_n)) \subseteq \text{liminf}(F(A_n))$ so the desired result follows by 2.43(b).

One immediate corollary to this result is the delayed proof of 2.46(vii).

2.49 Define a class of restriction operators to be normal if it each of its members is doubly continuous and:

a) $\forall A.\forall B.\ A\restriction o = B\restriction o$

b) $\forall A.\forall n.\ m.(A\restriction n)\restriction m = A\restriction \min(m,n)$

c) $\forall w.\exists n.\ A.\ w \in A \Leftrightarrow w \in A\restriction n$

Note that each of the examples on page is normal.

2.50 Theorem

If a predicate is strongly continuous then it is continuous relative to every normal class of restriction operators.

proof

Suppose $\{\restriction n \mid n \in N\}$ is such a class, $A \in P$ and $\langle A_i \rangle$ is a sequence s.t. $\forall i.A_i\restriction i = A\restriction i \ \& \ R(A_i)$ for some strongly continuous predicate R. It will clearly suffice to show that $\langle A_i \rangle$ is a convergent sequence with limit A, for then we have R(A) by continuity of R.

Suppose $w \in \text{limsup}(A_i)$ then by 2.49 (b)&(c) there is some n_w s.t. $\forall B.\ w \in B \restriction m \Leftrightarrow w \in B$ if $m \geqslant n_w$

Now $w \in \bigcap_{i=0}^{\infty}(\bigcup_{j=i}^{\infty}(A_j)) \Rightarrow w \in \bigcup_{j=n_w}^{\infty}(A_j)$

$\Rightarrow \exists m \geqslant n_w\ w \in A_m$

$\Rightarrow w \in A_m \restriction m$

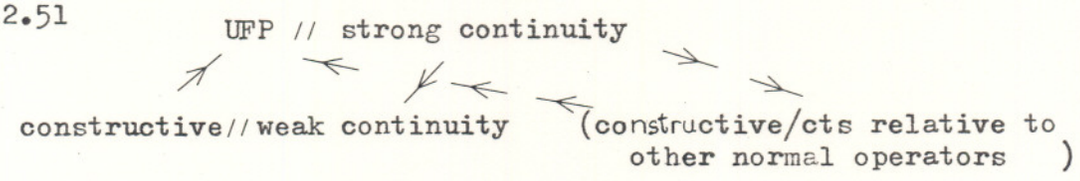
$\Rightarrow w \in A \restriction m$ (as $A_m \restriction m = A \restriction m$)

$\Rightarrow w \in A$

Hence $\text{limsup}(A_i) \subseteq A$ and in a similar way we can prove that $A \supseteq \text{liminf}(A_i)$ which gives us the desired result by 2.43(b).

This result is just a generalization of 2.45(a) (it would be necessary to impose stricter conditions upon the class of restriction operators to generalise 2.45(b)). It does however give us a ready-made class of predicates (2.46) to use with any normal class of restriction operators. We can draw the following diagram of conditions for validity of 2.1 showing their relative strengths.

2.51



Note that this assumes the double continuity of the function.

Postscript: Countable alphabets

It is possible to drop the condition of double continuity in 2.47 if we restrict our attention to countably infinite or finite alphabets. This has applications to the hiding operator, which as we have seen is not always reverse continuous. The fundamental fact used is that in either of these cases Σ^* is countable.

2.52 Theorem

If Σ is countable then every infinite sequence $\langle A_i \rangle$ of P has a convergent subsequence $\langle A_{n_i} \rangle$ where $i<j \Rightarrow n_i<n_j$.

proof

Enumerate Σ^* as $\{w_1, w_2, \ldots, w_i, \ldots\}$.

Claim that for each j we can find a sequence $\langle A_{n_{ji}} \rangle$ such that $n_{ji} < n_{j,i+1}$ and $n_{ji} \leq n_{j+1,i}$ and $j'<j \Rightarrow$ each n_{ji} is an $n_{j'i'}$ and $\forall i.\ A_{n_{ji}} \cap \{w_1, \ldots, w_{j-1}\} = A_{n_{j1}} \cap \{w_1, \ldots, w_{j-1}\}$

Suppose we have constructed such sequences for each $j'<j$.

If j=o we can simply set $n_{ji} = i$.

If $j=j'+1$ then either there is an infinite subsequence of $A_{n_{j'i}}$ s.t. each contains $w_{j'}$, or one s.t. each does not contain $w_{j'}$. Put n_{ji} equal to the ith $n_{j'i}$ of this subsequence.

It is easy to check that the sequence $A_{n_{ji}}$ thus formed satisfies the required conditions.

Thus the required sequence exists for each j.

Now set $m_i = n_{ii}$. We have $m_i < m_{i+1}$ by construction.

We know that $\text{limsup}(A_{m_i}) \supseteq \text{liminf}(A_{m_i})$ by 2.43(b) so suppose $w \in \text{limsup}(A_{m_i})$. $w = w_j$ for some j, and as $w_j \in \text{limsup}(A_m)$ we must have $w_j \in \bigcap_{k=j+1}^{\infty} (A_m)$. Thus there is some $k>j$ s.t. $w_j \in A_{m_k}$. But by construction each A_{m_k} $(k>j)$ is a $A_{n_{j+1,i}}$, and these have constant intersection with $\{w_1, \ldots, w_j\}$ so if $\forall k>j.\ w_j \in A_{m_{k\infty}}$.

Thus $w_j \in \bigcap_{k=j+1} (A_{m_k})$

$\Rightarrow w_j \in \text{liminf}(A_{m_k})$

Hence liminf(A_{m_i}) = limsup(A_{m_i}) and so $\langle A_{m_i} \rangle$ is the required convergent subsequence.

Given this result we can now prove the stronger version of 2.47 for countable alphabets.

2.53 Theorem

If Σ is countable and $F:P \to P$ is monotonic and if R is a strongly continuous satisfiable predicate then

$$(\forall A.(R(A) \Rightarrow R(F(A)))) \Rightarrow R(fix(F)) \qquad (*)$$

if F has a unique fixed point.

proof

We can define $F^{\alpha}(\bot)$ and $F^{\alpha}(\top)$ for arbitrary ordinal α as follows

$$F^{0}(\bot) = \bot \qquad F^{0}(\top) = \top$$
$$F^{\alpha+1}(\bot) = F(F^{\alpha}(\bot)) \quad F^{\alpha+1}(\top) = F(F^{\alpha}(\top))$$
$$F^{\mu}(\bot) = \bigcup_{\sigma<\mu} F^{\sigma}(\bot) \quad F^{\mu}(\top) = \bigcap_{\sigma<\mu} F^{\sigma}(\top) \quad \text{if } \mu \text{ is a limit ordinal.}$$

There must be some countable ordinal κ s.t.

$$F^{\kappa}(\bot) = F^{\kappa}(\top) = fix(F)$$

The fact that κ is countable can be derived in a similar way to 4.13.

We can now prove by transfinite induction on $\rho \leqslant \kappa$ that

$$\forall \rho.\ \exists A_{\rho}.\ R(A_{\rho}) \ \&\ F^{\rho}(\bot) \subseteq A_{\rho} \subseteq F^{\rho}(\top)$$

This is true for $\rho=0$ since R is satisfiable.

Suppose true for ρ

then $F^{\rho}(\bot) \subseteq A_{\rho} \subseteq F^{\rho}(\top)$ & $R(A_{\rho})$

$\Rightarrow F(F^{\rho}(\bot)) \subseteq F(A_{\rho}) \subseteq F(F^{\rho}(\top))$ & $R(F(A_{\rho}))$ by monotonicity and (*)

$\Rightarrow F^{\rho+1}(\bot) \subseteq A_{\rho+1} \subseteq F^{\rho+1}(\top)$ & $R(A_{\rho+1})$ where $A_{\rho+1} = F(A_{\rho})$

as desired.

Suppose true of all $\beta<\rho$ and that ρ is a limit ordinal.

Because ρ is countable there must be some ascending sequence $\langle \beta_i \mid i \in N \rangle$ of ordinals less than ρ s.t. $\bigcup_{i=1}^{\infty} \beta_i = \rho$.

Put $B_i = A_{\beta_i}$ This is an infinite sequence which satisfies $\forall i.R(B_i)$ and by 2.52 has an infinite convergent subsequence $\langle B_{n_i} \rangle$ s.t. $\forall i.n_i \geqslant i$. We then have $\forall i.F^{\beta_{n_i}}(\bot) \subseteq B_{n_i} \subseteq F^{\beta_{n_i}}(\top)$.

$\Rightarrow \forall i.F^{\beta_i}(\bot) \subseteq B_n \subseteq F^{\beta_i}(\top)$ as $\beta_i \leqslant \beta_{n_i}$

$\Rightarrow F^{\rho}(\bot) = \text{limsup}(F^{\beta_i}(\bot)) \subseteq \lim(B_{n_i}) \subseteq \text{limsup}(F^{\beta_i}(\top)) = F^{\rho}(\top)$

$R(\lim(B_{n_i}))$ holds by strong continuity of R, so we can put $A_{\rho} = \lim(B_{n_i})$.

This establishes the desired result for all $\rho \leq \kappa$.
Hence in particular it holds for κ, and so there is some
A s.t. R(A) and $fix(F) = F^{\kappa}(\top) \supseteq A \supseteq F^{\kappa}(\bot) = fix(F)$.
Thus $A = fix(F)$, which completes the proof that R(fix(F)) holds.

This result is <u>not</u> true when Σ is uncountable, as is demonstrated by the following example.

If Σ is uncountable there is some (uncountable) initial ordinal λ equinumerous with it; suppose $\{a_\kappa \mid \kappa \in \lambda\}$ is any enumeration of $\Sigma - \{\surd\}$ by λ. Define $F:P \to P$ by $F(A) = \{\langle\rangle, \langle b, \langle a_\kappa \rangle w \mid \forall \nu \in \kappa \langle a_\nu \rangle \in A\}$. It is easy to see that F is monotonic but not continuous, and that F has unique fixed point <u>run</u>. Now consider the predicate R defined $R(A)$ = "A^o is countable"; R is plainly satisfiable and it is easy to show that $R(A) \Rightarrow R(F(A))$ for all $A \in P$. That R is strongly continuous follows from the fact that a countable union of countable sets is countable, so that whenever $\langle A_i \rangle$ is any sequence of processes s.t. $R(A_i)$ for each i we have that $(\text{limsup}(A_i))^o$ is countable also. However, R(<u>run</u>) plainly does not hold, demonstrating that the rule is not valid in this case.

In the appendix to the next chapter we will see that if we strengthen still further our definition of continuity of predicates 2.51 does hold for general alphabets. In addition we will see there that over uncountable alphabets neither strong continuity nor the even stronger continuity can be represented as continuity relative to any normal class of restriction operators. Such a class of operators does exist for countable alphabets, though:

2.52 <u>Theorem</u>
If Σ is countable then strong continuity is equivalent to continuity relative to the following normal class of restriction operators:

$A \restriction i = \{w_j \mid j \leq i \ \& \ w_j \in A\}$
where $\{w_0, w_1, \ldots\}$ is any fixed enumeration of Σ^* with the property that $w_i < w_j \Rightarrow i < j$ (such enumerations are easy to construct).

<u>proof</u>
It is very easy to show that the above is indeed a normal class of restriction operators. By 2.50 we know that every predicate which is strongly continuous is continuous relative to any normal class of restriction operators, so it will be sufficient

to show that continuity relative to the above class implies strong continuity. To this end let us suppose that R is a predicate continuous relative to our class of operators, and that $\langle A_i \rangle$ is a sequence of processes which converges to A and such that $R(A_j)$ holds for each $j \in N$.

Since $\text{limsup}(A_i) = \text{liminf}(A_i) = A$ it is easy to see that for all $i \in N$ there exists some minimal $k(i)$ with the property that $m \geqslant k(i) \Rightarrow ((w_i \in A_m) \Leftrightarrow (w_i \in A))$. Define $r(i) = \max\{k(j) \mid j \leqslant i\}$: for each $s \geqslant r(i)$ we have $A_s \lceil i = A \lceil i$ by definition of $\lceil i$.

Hence for each i there exists some B s.t. $B \lceil i = A \lceil i$ and $R(B)$; thus $R(A)$ follows by continuity of R relative to this class.

Thus $\{A \mid R(A)\}$ is closed under the limits of its convergent sequences, so R is strongly continuous as claimed.