The University of Warwick

THEORY OF COMPUTATION

REPORT . NO.31

ON THE SEMANTICS OF FAIR PARALLELISM

BY

DAVID PARK

TO APPEAR IN PROCEEDINGS OF THE WINTER SCHOOL ON FORMAL SOFTWARE SPECIFICATION, LYNGBY, DENMARK. JAN.1979

Department of Computer Science
University of Warwick
COVENTRY CV4 7AL
ENGLAND.

October 1979

ON THE SEMANTICS OF FAIR PARALLELISM

David Park
Department of Computer Science
University of Warwick
Coventry CV4 7AL
England

1. INTRODUCTION

Suppose that a programming language involves, among other familiar ways of composing commands C_i, a "parallel" construct (C_1 par C_2) . One expects, when using this language, that a sequence such as

x := 0; y := 1; (x := 1 par (while x = 0 do y := y+1))

should be guaranteed to terminate in whatever context it is executed. The command x := 1 must be executed some time, and terminates the while command; both arms of the parallel pair are then complete, and there is nothing else left to be done. Moreover the fact of termination is important enough, one feels, to be deducible from the formal semantics of the programming language. There are obviously many similar contexts in which the termination, and therefore the correctness, of programs may depend on such points. Nevertheless, the appropriate general constraint (the *fairness* or *finite delay* property), whereby commands executed in parallel are each given as large amounts of time as they need to terminate, is notoriously hard to reconcile with methods which suffice to specify other features of programs. There is a crucial distinction involved, between "unbounded but finite" and "potentially infinite" attributes of the abstract objects involved in the specification. The distinction is closely related to the issue of *unbounded nondeterminism*. Because of this we will spend some time discussing such generalities, with a view to exorcism.

The approach to fairness which we will suggest is a development from the use of fixpoints to obtain *relational semantics*, as in Hitchcock & Park [6] and in de Bakker & de Roever [3]. The account is relatively informal, in order to make the algebraic ideas as accessible as possible. A more formal account of nondeterministic relational semantics, including a discussion of the Hitchcock-Park method for proving termination, can be found in de Bakker [2] .

2. BASIC CONCEPTS

Given a set A, $P(A)$ denotes its *power-set*, the set of all subsets of A, and $R_n(A)$ denotes the set $P(A^n)$ of *n-ary relations* on A. The usual mathematical notation is used here to denote combinations of relations and to abbreviate assertions about them. In particular, the *set abstraction* notation

$$\{\tau \mid \text{---------}\}$$

is used to denote the set of all those values of the term τ for which a given property holds of the corresponding values of variables mentioned. The usual conventions as to which variables are bound in this usage will replace the more formal alternatives for variable binding.

A function F: $R_n(A) \to R_m(B)$ is *monotone* if $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$. As is well-known, every monotone function F: $R_n(A) \to R_n(A)$ has a minimal fixpoint

$$\mu F = \cap\{X \mid F(X) \subseteq X\} \qquad (2.1)$$

satisfying $F(\mu F) = \mu F$.

In the informal development here, we have in mind a written form $F(X)$ for the function F, and will use the notation $\mu X.F(X)$ for μF.

Schemes for denotational semantics adopting "the fixpoint approach" use the fixpoint operator as the basic means for describing functions computed using iteration or recursion. In the relational approach the only functions fixpointed are on relation algebras, or on products of relation algebras. For example, the command

<u>while</u> B <u>do</u> C

should denote the relation (presumably the graph of a function)

$$\mu X.\{(x,y) \mid (x \notin S_B \ \underline{\text{and}}\ x = y)\ \underline{\text{or}}\ (x \in S_B\ \underline{\text{and}}\ (\exists z)((x,z) \in R_C\ \underline{\text{and}}\ (z,y) \in X))\} \qquad (2.2)$$

This denotes a binary relation on states, assuming that S_B is the set of states satisfying B, and R_C is the input-output relation corresponding to the command C.

From the form (2.1) for μF follows the principle (sometimes referred to as "recursion induction"):

<u>fixpoint induction principle:</u>

$$\text{If } F(Z) \subseteq Z \quad \text{then } \mu F \subseteq Z \qquad (2.3)$$

Many "inductive" arguments from mathematics can be regarded as instances of this principle, for suitable F and Z. "Induction axioms", in this light, are constraints on μF; typically that it is the whole of the mathematical structure in question.

Slightly more familiar than (2.1) is the alternative in the case that F is ω-continuous as well as monotone.

<u>ω-continuity</u>: $F(X)$ is ω-continuous in X if, whenever $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots$, then

$$F(\bigcup_{i=0}^{\infty} X_i) = \bigcup_{i=0}^{\infty} F(X_i) \qquad (2.4)$$

In this case, there is the alternative expression

$$\mu F = \bigcup_{n=0}^{\infty} F^n(\emptyset) \qquad (2.5)$$

In fact, (2.5) extends to the case that F is only known to be monotone, at the expense of iterating F into the transfinite ordinals. (Hopefully the reader will not take fright at this notion; all that need concern him here are that the ordinals are linearly ordered, that each is either the successor of an ordinal or the least upper bound of its predecessors, and that the first such limit ordinal is ω, not counting 0.)

For each ordinal α, $F^{\alpha}(\emptyset)$ is defined by

$$F^{\alpha}(\emptyset) = F(F^{\beta}(\emptyset)) \text{ if } \alpha = \beta+1$$

$$F^{\alpha}(\emptyset) = \bigcup_{\beta<\alpha} F^{\beta}(\emptyset) \text{ if } \alpha \text{ is a limit ordinal.}$$

Then for any monotone F

$$\mu F = F^{\alpha}(\emptyset) \text{ for some ordinal } \alpha \qquad (2.6)$$

In fact, once (2.6) holds for some α, it holds for all larger α, from the definition above and the fixpoint property; so (2.6) holds *for all sufficiently large* α. In the case that F is ω-continuous, α can be taken as ω; indeed (2.5) is just $F^{\omega}(\emptyset)$.

(2.5) is usually taken as the basis for Scott induction; however (2.6) permits a generalization to fixpoints of monotone, not necessarily continuous, functions. But first the full definition of continuity is needed.

<u>continuity</u>: $G(Y)$ is continuous in Y if, for any ordinal α, and any sequence Y_{λ}, $\lambda<\alpha$, such that $\lambda < \mu \Rightarrow Y_{\lambda} \subseteq Y_{\mu}$, we have

$$G(\bigcup_{\lambda<\alpha} Y_{\lambda}) = \bigcup_{\lambda<\alpha} G(Y_{\lambda})$$

(ω-continuity is just the special case $\alpha = \omega$; the definition here is equivalent to the more usual definition through directed sets - assuming the Axiom of Choice.)

The following is a statement of Scott induction in a notationally tolerable special case.

<u>Scott induction principle</u>:

If (a) $G_i(Y_1, Y_2)$ are continuous in Y_1, Y_2, i=1,2.

(b) $F_i(X)$ are monotone in X, i=1,2.

(c) $G_1(\emptyset, \emptyset) = G_2(\emptyset, \emptyset)$

(d) whenever $G_1(Y_1, Y_2) = G_2(Y_1, Y_2)$

$$\text{then } G_1(F_1(Y_1), F_2(Y_2)) = G_2(F_1(Y_1), F_2(Y_2))$$

then we can conclude from (a)-(d) that

$$G_1(\mu X.F_1(X), \mu X.F_2(X)) = G_2(\mu X.F_1(X), \mu X.F_2(X)) \qquad (2.7)$$

The justification for this version of the principle is by a transfinite induction argument, showing that

$$G_1(F_1^{\lambda}(\emptyset), F_2^{\lambda}(\emptyset)) = G_2(F_1^{\lambda}(\emptyset), F_2^{\lambda}(\emptyset))$$

for all ordinals $\lambda \le \alpha$. The important continuity constraint (a) is necessary in the case that λ is a limit ordinal.

Finally, we sketch the generalizations which are needed in the case of simultaneous recursions. We are then dealing with functions on *products* of relation algebras

$$F_j : \prod_{i=1}^{N} R_{n_i}(A_i) \to R_{n_j}(A_j) \qquad 1 \le j \le N$$

with each $F_j(X_1, X_2, \ldots, X_N)$ monotone in each X_i.

Now the N functions $F_1, F_2, \ldots, F_N$ can be thought of as combined into a single function on the product algebra, defined by

$$F(X_1, X_2, \ldots, X_N) = (F_1(X_1, \ldots, X_N), \ldots\ldots, F_N(X_1, \ldots, X_N))$$

Moreover the structure on the product algebra is essentially just that of an algebra of subsets of a disjoint union of the sets $A_i^{n_i}$. We can indicate this notationally:

$$\emptyset_{\Pi} = (\emptyset, \emptyset, \ldots, \emptyset)$$

$$(X_1, X_2, \ldots, X_N) \subseteq_{\Pi} (Y_1, Y_2, \ldots, Y_N) \text{ iff } X_i \subseteq Y_i,\ 1 \le i \le N$$

and so on. With further definitions of $\cup_{\Pi}$, $\cap_{\Pi}$ in this style we can proceed to recast the development for simple recursions into one which applies to the simultaneous case as well. In particular we have

$$\mu F = \cap_{\Pi}\{X \mid F(X) \subseteq_{\Pi} X\}$$
$$= F^{\alpha}(\emptyset_{\Pi}) \text{ for suitable } \alpha.$$

If the functions F_i are defined by forms $F_i(X_1, \ldots, X_N)$, $1 \le i \le N$, we use

$$\mu_i X_1 X_2 \ldots X_N.(F_1(X_1, \ldots, X_N), \ldots, F_N(X_1, \ldots X_N))$$

to denote the i-th component of the fixpoint μF.

We will not write out the generalizations of fixpoint induction and Scott induction principles here. The following principle needs to be emphasized. It allows one either to eliminate simultaneous fixpoints in favour of simple fixpoints, or, used in the reverse direction, to move fixpoint operators outwards from expressions to achieve one simultaneous fixpoint, with no fixpoint operators in the expressions for the functions involved. We state just the special case for N = 2.

Bekić-Scott principle:

If $F_i(X_1, X_2)$ are monotone in X_1, X_2, i=1,2, then

$$\mu_1 X_1 X_2.(F_1(X_1,X_2), F_2(X_1, X_2)) = \mu X_1.F_1(X_1, \mu X_2.F_2(X_1, X_2))$$

$$\mu_2 X_1 X_2.(F_1(X_1,X_2), F_2(X_1, X_2)) = \mu X_2.F_2(\mu X_1.F_1(X_1, X_2), X_2)$$

3. THE PROBLEM OF UNBOUNDED NONDETERMINACY

It is well-known that fair parallelism and unbounded nondeterminacy are interrelated problems. Both the nature of the latter problem and the reason for the relationship can be seen from the introductory example in Section 1. Under fair scheduling the example exhibits unbounded nondeterminacy in the sense that (a) the program terminates whatever; and (b) the final value that y takes is not, in principle, bounded; any value from the *infinite* set {1,2,3,......} is possible. In view of this relationship, any "solution" to the fairness problem would also seem to provide a mechanism to implement unbounded nondeterminacy. Since there are doubts as to the feasibility of such a mechanism, it would be as well to focus on them in order to understand what sort of solution we might expect.

We proceed in the following steps; first, we look at the technical difficulty as it presents itself in an abstract setting, by looking at the "power-domain" constructions available for providing semantics for nondeterminism in the Scott-Strachey style; then we examine the technical difficulties as regards a relational treatment - here we discover that these are not so severe as is implied by the discussion in Dijkstra[4] ; lastly, we discuss the reasons given for considering this sort of feature unfeasible and/or undesirable.

Since we want to contemplate unbounded nondeterminacy by itself, it will be convenient to postulate a programming feature intended to exhibit it. The most natural device in the context of our first example would be an expression which takes, nondeterministically, any positive integer value; writing this as "anyposint", the example program is then to be equivalent to

```
x := 1; y := anyposint ;
```

3.1. Nondeterminism and Power-domains

Power-domain constructions are intended to augment the system of domains used in the Scott-Strachey style of description so as to provide denotations for non-deterministic programs. If D, E are domains, and $\mathcal{P}(E)$ is the power-domain obtained from E by this construction, the "nondeterministic functions" from D to E are to be identified with elements of $(D \to \mathcal{P}(E))$, the domain of continuous functions (in the domain theoretic sense) from D to $\mathcal{P}(E)$. The case we should be immediately interested in is the case in which D, E are the same "flat domain", obtained by adjoining a minimal element $\perp$ ("nontermination") to an otherwise unstructured set of "machine states" .

Quite apart from the issue of continuity (which we will be discussing later, in the context of a relational development), the power-domain construction provides a setting for a slightly different issue. It turns out that the objects in each power-domain $\mathcal{P}(E)$ must be obtained by making identifications between subsets of E. The identifications which concern us occur in all three constructions known to the author; in Plotkin's original proposal [5] , in the variant due to Smyth [8] , and in the construction based on the lattice of Scott-closed subsets - which results from following up the converse to Smyth's ordering. In Plotkin's terminology, the identification involved is that of a set with its *Cantor closure* - the closure with respect to a certain topology on the domain. But we can look at what is involved in a slightly different light - by appealing to constraints on how subsets X of E are to be characterized.

What is needed to specify this constraint is some notion of "finite fact" about elements of E. The collection of such "finite facts" about elements which are consistent with membership in X should then be the *most* we can use to characterize X.

<u>exclusion criterion</u>: for an object x to be definitely excluded from membership in X, there must be some "finite fact" which holds of x but which holds of no element of X.

An object x, therefore, for which there is no such finite fact, can be adjoined to X without affecting its characterization. So at least to this extent distinctions between sets in $\mathcal{P}(E)$ are blurred (and in practice there is more "blurring" before a partial ordering structure can be obtained making $\mathcal{P}(E)$ an acceptable domain).

It remains to choose an appropriate notion of "finite fact". The obvious choice connected with Cantor closure fits in well with other intuitively appealing aspects of domains - relating all computationally relevant structure to properties of a denumerable set of *basis* elements.

<u>finite fact about x</u>: any boolean combination of assertions of the form $e \sqsubseteq x$, where x is the given element and e may be any basis element of E.

The general constraint that results from these considerations is the identification of sets with their Cantor closures. In our case E is a "flat" domain, satisfying

$x \sqsubseteq y \Leftrightarrow (x = \bot$ <u>or</u> $x = y)$

Every element is a basis element. The difficulty over unbounded nondeterminism appears when we apply the exclusion criterion to $\bot$, with X an infinite set. For consider any finite fact true of $\bot$, and suppose it to be in disjunctive normal form. At least one disjunct must be true of $x = \bot$, and each such disjunct must be equivalent to a conjunction of the form

$$e_1 \not\sqsubseteq x \text{ \underline{and} } e_2 \not\sqsubseteq x \text{ \underline{and}} \ldots\ldots \text{\underline{and} } e_n \not\sqsubseteq x \qquad (3.1.1)$$

for some n, since $\bot \sqsubseteq x$ is true of all x, and no other assertions $e \sqsubseteq x$ are true of $\bot$. But any statement of the form (3.1.1) is true of all but a finite number of elements of E, so must be true of some element of X. $\bot$ can therefore not be excluded from X - and this is essentially the difficulty we expected.

<u>Note</u>: Plotkin invokes the notion of "finitely generable" set to put an initial constraint on $\mathcal{P}(E)$, and it is tempting look for significance in this aspect of the construction rather than elsewhere. But in fact every Cantor closed set is finitely generable, as Plotkin points out; so the initial constraint was not, after all, essential. The same structure is obtained by partitioning all subsets with the help of the closure operator.

This difficulty in the context of power-domains makes it apparently impossible to find a reasonable semantics reflecting fair parallelism in the way one would like, i.e. to find a function mapping parallel programs to denotations which are elements of some domain, of "nondeterministic functions" perhaps . This does not at all rule out, however, denotations obtained as *subsets* (in the conventional sense) of domains - and research in this direction seems called for. What seems to require careful formulation is the problem of choosing appropriate domains for use in continuation semantics cast in such a style, since analogs of the reflexive domains needed may not be constructible.

3.2. Relational semantics for nondeterminism.

The outcome of a relational semantics for a deterministic program can be quite straightforward. One characterizes a set S of "machine states", and produces, for each program C, a denotation $\mathcal{M}[\![C]\!] \in \mathcal{R}_2(S)$, which is to be the relation between input and corresponding output states of C. But in the nondeterministic case there is an extra complication concerned with termination. The input-output relation by itself will not distinguish between the following two trivial programs

skip (3.2.1)

skip or (while true do skip) (3.2.2)

but the second of these has a possible non-termination, whereas the first does not. But one clearly wants to distinguish between such programs. One can cope with this problem either by adding an element $\perp$ to S, to signify nontermination in the style of Scott - this is the device adopted by de Bakker [3]- or, as we do here, by adding an additional semantic function. For each program C, we will aim to give a denotation composed of two sets:

$M[\![C]\!] \in R_2(S)$ the *relation computed* by C

$T[\![C]\!] \subseteq S$ the *termination domain* of C

The intended significance is that all executions of C terminate iff C is started from a state in $T[\![C]\!]$. The two programs (3.2.1), (3.2.2) are then distinguished by T, which yields S for the former and $\emptyset$ for the latter, presumably. In the deterministic case T is unnecessary , since it may be derived from M - unless it is intended to attach some meaning to termination without a value as opposed to nontermination. At the end of this article we will give definitions for M, T for a "toy" language in which commands are permitted to exhibit unbounded nondeterminism. But we should first discuss the arguments that seem to imply this is an unreasonable goal.

The most influential critique of unbounded nondeterminacy appears to be that embodied in Chapter 9 of Dijkstra [4]. In this section we will concentrate on the technical difficulties discussed by him, setting aside conceptual problems for the moment. Our observations here follow lines largely developed by de Bakker[3]and de Roever [7].

There are some awkward superficial obstacles first of all. Dijkstra talks of "predicates" and "conditions", where we are inclined to talk in terms of sets, so the reader should be prepared to substitute propositional notions (and, or, $\Rightarrow$ etc.) for the corresponding set-theoretic notions ($\cap$, $\cup$, $\subseteq$ etc.) in order to translate between our principles and those enunciated by Dijkstra, and vice versa. Moreover, we will want to regard the fixpoint operator μ as applicable in either context, so that the principles sketched in Section 2 should be regarded in that light as well. For example, there will be a fixpoint induction principle applicable to predicate transformers $F(R)$

if $F(R) \Rightarrow R$ then $\mu F \Rightarrow R$ (3.2.3)

Secondly, Dijkstra is intent on obtaining axioms for the wp predicate transformer, and not on obtaining denotations for programs. But his criticisms must apply to our efforts as well, since wp can be characterized in terms of our M and T.

$$wp(C, R)(s) \Leftrightarrow (s \in \mathcal{T}[\![C]\!] \ \underline{\text{and}}\ (\forall s')((s,s') \in \mathcal{M}[\![C]\!] \Rightarrow R(s'))) \qquad (3.2.4)$$

Thirdly, we prefer to talk about programs constructed from while statements, and to achieve nondeterminacy either through an or construct on commands or through a possibly nondeterministic basic command, rather than through the guarded command formalism. We hope these details are only superficial obstacles to understanding.

Dijkstra proceeds by enunciating a continuity property - that wp(C, R) is ω-continuous in R, ~~in~~ just the sense of Section 2, though translated into propositional terms. He then shows that the language he has developed to that point has this property, but that the command which we have written x := anyposint does not, since

$$wp(x := \text{anyposint}, \bigvee_{i=0}^{\infty}(x < i))$$

is true of all states, but

$$wp(x := \text{anyposint}, x < i)$$

is true of none. This is quite correct, and an important example; failures of the continuity property are inconvenient - many proofs depend critically on continuity assumptions, which are often embedded in principles for reasoning about programs (e.g. condition (a) of our version of Scott induction). Dijkstra, however, appears to overestimate the inconvenience. The reason for this is that there is, in effect, a hidden continuity assumption in his rules for reasoning about the repetitive construct. He notices this, but fails to notice that it can be fixed elegantly. In our notation, his example is the following command:

$$\underline{\text{while}}\ x \neq 0\ \underline{\text{do}}\ (\underline{\text{if}}\ x < 0\ \underline{\text{then}}\ x := \text{anyposint}\ \underline{\text{else}}\ x := x - 1) \qquad (3.2.5)$$

Call this command WHILE, and the component conditional IF; what is wp(WHILE, true)? Following Dijkstra, we have

$$wp(\text{WHILE}, \underline{\text{true}}) = \bigvee_{i=0}^{\infty} H_i \qquad (3.2.6)$$

where $H_0 = (x = 0)$

$$H_{i+1} = (wp(\text{IF}, H_i)\ \underline{\text{or}}\ H_i)$$

by induction on i, then (assuming we know all about wp(IF, R))

$$H_i = (0 \leq x \leq i)$$

but then, plugging back into (3.2.6)

$$wp(\text{WHILE}, \underline{\text{true}}) = (x \geq 0)$$

and this is clearly wrong - WHILE terminates for all x, positive or negative. But there is another version of (3.2.6), using μ:

$$wp(\text{WHILE}, \underline{\text{true}}) = \mu X.(x = 0\ \underline{\text{or}}\ wp(\text{IF}, X)) \qquad (3.2.7)$$

i.e. wp(WHILE, true) is the *strongest condition* X such that

$$X = (x = 0\ \underline{\text{or}}\ wp(\text{IF}, X))$$

To see that (3.2.7) should give the correct result for wp(WHILE, true), it may help to move to the formulation using sets. The analogous function is

$$F(X) = (\{0\} \cup \{x | x < 0\ \underline{\text{and}}\ N^+ \subseteq X\} \cup \{x | x-1 \in X\})$$

where $N^+ = \{y | y > 0\}$; (we are assuming that "states" are just values of x - but this is not essential to the result of (3.2.7)); now we have

$$F(\emptyset) = 0$$
$$F^2(\emptyset) = \{0, 1\} \text{ etc., and}$$
$$F^\omega(\emptyset) = \bigcup_{i=0}^{\infty} F^i(\emptyset) = \{0\} \cup N^+$$

And we must now go transfinite - but for just one step; putting $N^- = \{y | y < 0\}$

$$F^{\omega+1}(\emptyset) = \{0\} \cup N^- \cup N^+$$

which is now correct. So this was a case where a fixpoint of a *non-continuous* F is called for - moreover an F with $\mu F \neq F^\omega(\emptyset)$.

The nature of the error and the way to fix it now begins to be clear. The general formulation for the while statement should be

$$\text{wp}(\underline{\text{while}}\ B\ \underline{\text{do}}\ C,\ R) = \mu X.((\underline{\text{non}}\ B\ \underline{\text{and}}\ R)\ \underline{\text{or}}\ (B\ \underline{\text{and}}\ \text{wp}(C, X))) \qquad (3.2.8)$$

If the right hand side of this is abbreviated $\mu X.F_R(X)$, the analog of Dijkstra's rule would be

$$\text{wp}(\underline{\text{while}}\ B\ \underline{\text{do}}\ C,\ R) = F_R{}^\omega(\underline{\text{false}}) \qquad (3.2.9)$$

This latter formulation is quite correct provided F_R is continuous, which in turn is true provided there is no possibility of unbounded nondeterminism.

This is not the end of the story, but to continue would take us too far from the theme of this article. Clearly (3.2.8) is also correct in the continuous case, provided (3.2.9) captures this special case of Dijkstra's axiom correctly. But it also remains to justify (3.2.8) in the monotone case, and to verify that the other main principle underlying Dijkstra's discipline, with the exception of the continuity property, are still valid. The fixpoint principles of Section 2 can, in fact, be used to justify Dijkstra's properties 1-3 ("excluded miracle", monotonicity and multiplicativeness of the wp function). And the "loop invariance theorem" also turns out to be a nice instance of the fixpoint induction principle when the fixpoint version of wp is adopted for the repetitive construct.

3.3 Continuity and implementability.

Many theoreticians accept the thesis that objects which are "computable" or "effectively given" may be obtained by considering only continuous operations on a tightly constrained system of domains and/or relation algebras. On the other hand, they would be inclined to accept the constraints that our "exclusion principle" puts on what sets can be regarded as effectively given - and insist on sets which are "Cantor closed" with respect to some domain structure. But the denotations of programs invoking fair parallelism or unbounded nondeterminism appear to violate both these constraints - and, superficially, nothing could be more effectively given than the denotation of a computer program. But there is one observation which appears to dodge the conflict, which concentrates on the role actually played by nondeterminism in the languages being discussed. One can ask the question - what is it that makes an implementation of such a language correct? The answers classify into one of two categories:

<u>tight nondeterminism</u>: each correct implementation must, according to some precise sense of "possible result", produce all and only those possible results which the semantics of the language prescribes.

<u>loose nondeterminism</u>: there may or may not be a sense in which the implementation can produce more than one result; the only constraint is that every result produced is one of those prescribed by the semantics.

In loose nondeterminism the requirement that all prescribed outputs be "possible" disappears. In this sense, any implementation of

```
          x := 1
```

is , loosely, an implementation of

```
          x := anyposint
```

Now it is immediately clear that the sense in which fair parallelism is nondeterministic is a loose one. Noone requires of a correct implementation for parallelism that there be an appropriate sense in which <u>all</u> scheduling algorithms be possible in it, only that there be one such scheduling algorithm, and if fairness be required that the scheduler be fair. On considering other 'nondeterministic' features of actual languages, the reader should also be able to convince himself that the nondeterminism intended is in the loose sense (the only exceptions seem to be in the realm of probabilistic algorithms).

With this observation the conflict begins to fade; there is still perplexity, nevertheless. Firstly, if the class of possible implementations is in no sense computable, there may be no effective test that an implementation is correct. As regards the fairness of schedulers, this is quite true - and can be derived as an unsolvability result in the classical style. Nevertheless, fairness is a property that can be formally established, just as totality can for recursive functions on the integers - a property with just the same status as fairness.

Secondly, one can doubt that 'fairness' and similar properties are really useful, since no very useful consequences can be deduced from them alone, and/or since reliance on them alone for correctness is not good programming practice. This view bases itself on scepticism as regards the value of guaranteed termination in the absence of bounds on termination time. If there is a defense to the view as applied to practical programs, it depends on the utility of separating termination proofs from the derivation of more detailed time estimates. In the case of deterministic programs, this separation appears to be real enough, if only because termination is usually just logic, and easy - whereas obtaining termination bounds (even quite loose ones) is mathematics, uses (so far) rather different methods and standards of rigour, and may be just that much harder to do. One might expect the same experience to recur as regards properties deducible from fairness alone, as against properties deducible from more detailed knowledge of scheduling. But to support the defense would need more experience in using fairness, etc. in proofs than we now have.

4. STRANGE FIXPOINTS AND THEIR USES

4.1 Maximal Fixpoints.

Over relation algebras the properties of the minimal fixpoint operator μ dualise in a straightforward way, to properties of the *maximal fixpoint operator* ν; thus

$$\nu X.F(X) = \cup\{X | X \subseteq F(X)\} \qquad (4.1.1)$$

for $F(X)$ any function monotone in X. Or we can, in effect, turn the algebra upside-down to obtain an expression with three nested complement signs

$$\nu X.F(X) = \mu X.(\overline{\overline{F(\overline{X})}}) \qquad (4.1.2)$$

Similarly, if we let $\mathcal{U}$ stand for the maximal *universal set* in the algebra, we can proceed downwards from the top of the algebra to obtain a maximal fixpoint:

$$F_0(\mathcal{U}) = \mathcal{U}$$

$$F_\alpha(\mathcal{U}) = F(F_\beta(\mathcal{U})) \quad \text{if } \alpha = \beta + 1$$

$$F_\alpha(\mathcal{U}) = \bigcap_{\beta<\alpha} F_\beta(\mathcal{U}) \quad \text{if } \alpha \text{ is a limit ordinal}$$

With these definitions, we have

$$\nu X.F(X) = F_\alpha(\mathcal{U}) \quad \text{for all sufficiently large } \alpha$$

and $$\nu X.F(X) = F_\omega(\mathcal{U}) \quad \text{in the case that } F \text{ is } \omega\textit{-cocontinuous}$$

<u>ω-cocontinuity</u>: $F(X)$ is ω-cocontinuous in X if, whenever $X_0 \supseteq X_1 \supseteq X_2 \supseteq \ldots$, then

$$F(\bigcap_{i=0}^{\infty} X_i) = \bigcap_{i=0}^{\infty} F(X_i)$$

Similarly, we can proceed to define general cocontinuity, and to derive dual inference principles to those of Section 2. Indeed they can be deduced from the principles of that section using the expression (4.1.2). We will not explicitly formulate these duals here.

4.2 Extended Languages.

Given any set Σ, the set $\Sigma^\dagger$ of *extended sequences over* Σ is the set Σ^* of finite sequences over Σ together with the set Σ^ω of infinite sequences over Σ. So $\Sigma^\dagger = \Sigma^* \cup \Sigma^\omega$. We will consider subsets of $\Sigma^\dagger$ as *extended languages*, a pretext for borrowing notation by analogy with standard notation for finite sequences. So, for $\sigma \in \Sigma$, $x,y \in \Sigma^\dagger$, $X,Y \subseteq \Sigma^\dagger$:

λ	denotes the null sequence
$\langle\sigma\rangle$	denotes the corresponding sequence of unit length
σx	denotes the result of prefixing σ to x
xy	denotes the concatenation of x and y, if $x \in \Sigma^*$ and denotes x, if $x \in \Sigma^\omega$.
XY	denotes $\{xy \mid x \in X, y \in Y\}$

This extended notation of concatenation, while degenerate on infinite sequences, can easily be seen to have the elementary algebraic properties of the standard notion:

$$\lambda x = x\lambda = x$$
$$x(yz) = (xy)z$$
$$X(YZ) = (XY)Z$$
$$X(Y \cup Z) = XY \cup XZ$$
$$(X \cup Y)Z = XZ \cup YZ$$
$$X\emptyset = \emptyset X = \emptyset$$

etc.

The star-closure operator on languages (subsets of Σ^*) can be defined using a minimal fixpoint. For extended languages, the same expression provides the expected generalisation of the notion. But we can talk now of omega-closure, dagger-closure as well. For an extended language $A \subseteq \Sigma^\dagger$, we have

star-closure of A:	$A^* = \mu X.(AX \cup \{\lambda\})$
omega-closure of A:	$A^\omega = \nu X.AX$
dagger-closure of A:	$A^\dagger = \nu X.(AX \cup \{\lambda\})$

It is easy to see that

$$A^* = \{\lambda\} \cup A \cup A^2 \cup \ldots$$

since the right-hand side has the form $F^\omega(\emptyset)$ and is a fixpoint of $F(X)$, taking $F(X) = AX \cup \{\lambda\}$. In fact this $F(X)$ is easily seen to be continuous in X - concatenation of extended languages is continuous. Notice that the above definitions of "omega-closure", "star-closure" do not correspond to intuition in the cases that the null string $\lambda \in A$, since in this case $A^\dagger = A^\omega = \Sigma^\dagger$ is the universal set. We should verify the following basic properties:

1. If $\lambda \notin A$, then $A^\omega = \{w_0w_1w_2 \ldots \mid w_i \in A\}$:

   Abbreviate the right-hand side as A^∞ for the moment. $A^\infty = AA^\infty$ satisfies the fixpoint equation defining A^ω, so $A^\infty \subseteq A^\omega$ - by the dual of the fixpoint rule (2.3) - if $A^\infty \subseteq AA^\infty$ then $A^\infty \subseteq$ X.AX $= A^\omega$. Conversely, suppose $w \in A^\omega$, then $w \in AA^\omega$ - so $w = w_0w'$, with $w_0 \in A$ and $w' \in A^\omega$; similarly $w' = w_1w''$ with $w_1 \in A$, and so on. So $w \in A^\omega$.

2. $A^\dagger = A^\omega \cup A^*$: this is the special case of (3) below in which $B = \{\lambda\}$.

3. $\nu X.(AX \cup B) = A^\omega \cup A^*B$: this gives an opportunity to exhibit the dual to the Scott principle (2.7) in an elementary context. Consider the equation:

   $$X = Y \cup A^*B$$

   (a) both sides of the equation are cocontinuous in X,Y;

   (b) the functions $AX \cup B$, AX are monotone in Y;

   (c) $\mathcal{V} = \mathcal{V} \cup A^*B$

   (d) whenever $X = Y \cup A^*B$, then

   $$\begin{aligned} AX \cup B &= A(Y \cup A^*B) \cup B \\ &= AY \cup AA^*B \cup B \\ &= AY \cup (AA^* \cup \{\lambda\})B \\ &= AY \cup A^*B \end{aligned}$$

   The conclusion of the dual principle follows, viz.

   $$\nu X.(AX \cup B) = \nu X.AX \cup A^*B$$

4. $\mu X.(AX \cup B) = A^{*}B$: follows directly from a conventional Scott induction.

5. If $\lambda \notin A$ and $B \neq \emptyset$, then $A^{\omega}B = A^{\omega}$, $A^{\dagger}B = A^{\omega} \cup A^{*}B$: that $A^{\omega}B = A^{\omega}$ follows from (1) above. The other equation is derived from this and from (2) as follows

$$A^{\dagger}B = (A^{\omega} \cup A^{*})B = A^{\omega}B \cup A^{*}B = A^{\omega} \cup A^{*}B$$

Note: while (1) above looks straightforward, it should be noticed that the maximal fixpoint $A^{\omega} = \nu X.AX$ is not necessarily reached in ω steps, from which it is clear that the function AX is not always cocontinuous. The sort of example which shows this was noticed by Tiuryn [9] in the context of algebras of infinite trees.

Example (Paterson): take $\Sigma = \{0, 1, 2\}$, $a \in \Sigma^{\dagger}$ defined by

$$a = 2101001000100001\ldots\ldots = 21010^{2}10^{3}10^{4}1\ldots$$

$$A = \{21010^{2}1\ldots.10^{n}1 \mid n \geq 0\} \cup \{0\}$$

then $a \in (21010^{2}1\ldots.0^{n-1}1)0^{n-1}\Sigma^{\dagger} \subseteq A^{n}\Sigma^{\dagger}$ for each n; so $a \in F_{\omega}(\Sigma^{\dagger})$; but $a \notin A^{\omega} = \nu F$.

In fact the concatenation function $F(X, Y) = XY$ on extended languages is not cocontinuous in either X or Y, (though it is cocontinuous on languages of finite sequences). To see that cocontinuity fails in the first argument position consider just

$$X_i = \{1^{j}/j>i\}$$

then $X_i1^{\omega} = \{1^{\omega}\}$, $\cap(X_i1^{\omega}) = \{1^{\omega}\}$

But $\cap X_i = \emptyset$, so $(\cap X_i)1^{\omega} = \emptyset$.

Maximal, rather than minimal, fixpoints are appropriate when defining functions, predicates recursively on $\Sigma^{\dagger}$. The following two "definitions", of concatenation and equality respectively, could well serve as axioms for $\Sigma^{\dagger}$ under concatenation

$$\{(x,y,xy) \mid x,y\in\Sigma^{\dagger}\} = \nu X.(\{(\lambda,y,y) \mid y\in\Sigma^{\dagger}\} \cup \{(\sigma x,y,\sigma z) \mid (x,y,z)\in X, \sigma \in \Sigma\})$$

$$\{(x,x) \mid x\in\Sigma^{\dagger}\} = \nu X.(\{(\lambda,\lambda)\} \cup \{(\sigma x,\sigma y) \mid (x,y)\in X, \sigma\in \Sigma\})$$

Notice that if μ is used rather than ν, what are obtained are concatenation and identity restricted so that $x\in\Sigma^{*}$, and undefined for $x\in\Sigma^{\omega}$.

4.3 The "fair merge" problem.

One can approach the fair scheduling problem by first of all tackling the problem of defining a "fair" merge on $\Sigma^{\dagger}$, a nondeterministic function which interleaves pairs of possibly infinite sequences in such a way that all of any infinite sequence provided is absorbed. A functional relevant to finding the fair merge relation is the following

$$\begin{aligned} \mathcal{F}m(X) = {} & \{(\lambda,x,x) \mid x\in\Sigma^{\dagger}\} \cup \{(x,\lambda,x) \mid x\in\Sigma^{\dagger}\} \\ & \cup \{(\sigma x,y,\sigma z) \mid \sigma\in\Sigma,\ (x,y,z)\in X\} \\ & \cup \{(x,\sigma y,\sigma z) \mid \sigma\in\Sigma,\ (x,y,z)\in X\} \qquad (4.3.1) \end{aligned}$$

Unfortunately, neither $\mu\mathcal{F}m$ nor $\nu\mathcal{F}m$ is the required relation. For $\mu\mathcal{F}m$ produces no merges from pairs of infinite sequences,while $\nu\mathcal{F}m$ produces merges which are not fair. Thus

$$\begin{aligned} & (0^{\omega},1^{\omega},x)\in\mu\mathcal{F}m \qquad \text{for } \underline{\text{no}}\ x\in\{0,1\}^{\dagger} \\ \text{while } & (0^{\omega},1^{\omega},0^{\omega})\in\nu\mathcal{F}m, \qquad \text{for example.} \end{aligned}$$

A clue to the appropriate solution is given by concentrating just on the fair merges of 0^{ω} and 1^{ω}, which is the set of all sequences with infinite numbers of both 0's and 1's. This set can be specified nicely using the operators of 4.2 as

$$\begin{aligned} & (0^{*}11^{*}0)^{\omega} \\ \text{or } & (1^{*}00^{*}1)^{\omega} \end{aligned}$$

Now

$$\begin{aligned} (0^{*}11^{*}0)^{\omega} & = \nu X.(0^{*}11^{*}0X) \\ & = \nu X.(0^{*}1\mu Y.(0X\cup 1Y)) \\ & = \nu X.(\mu Z(0Z\cup 1\mu Y.(0X\cup 1Y))) \\ & = \nu X.(\mu Z(\mathcal{F}(Z,\mu Y.\mathcal{F}(X,Y)))) \end{aligned}$$

where $\mathcal{F}(X,Y) = 0X \cup 1Y$.

Proceeding by analogy, we first distinguish the two occurrences of X in $\mathcal{F}m(X)$:

$$\begin{aligned} \mathcal{F}m(X,Y) = {} & \{(\lambda,x,x) \mid x\in\Sigma^{\dagger}\} \cup \{(x,\lambda,x) \mid x\in\Sigma^{\dagger}\} \\ & \cup\{(\sigma x,y,\sigma z) \mid \sigma\in\Sigma,\ (x,y,z)\in X\} \\ & \cup\{(x,\sigma y,\sigma z) \mid \sigma\in\Sigma,\ (x,y,z)\in Y\} \qquad (4.3.2) \end{aligned}$$

Then define

$$\text{fairmerge} = \nu X.\mu Z.(\mathcal{F}m(Z,\mu Y.\mathcal{F}m(X,Y))) \qquad (4.3.3)$$

The dual of the Bekić-Scott principle allows us to rewrite this, after noting that $\nu X.A = A$ if A is independent of X. Thus

$$\text{fairmerge} = \nu X_1.\mu Z.(\mathit{Fm}(Z,\nu X_2.\mu Y.\mathit{Fm}(X_1,Y))) = \nu_1$$

where, for i=1,2, $$\nu_i = \nu_i X_1X_2.\langle \mu Z.\mathit{Fm}(Z,X_2),\mu Y.\mathit{Fm}(X_1,Y)\rangle. \qquad (4.3.4)$$

We can now note that fairmerge = ν_2 also, using three applications of the fixpoint property:

$$\nu_1 = \mu Z.\mathit{Fm}\ (Z,\nu_2) \qquad (4.3.5)$$
$$\nu_2 = \mu Y.\mathit{Fm}\ (\nu_1,Y) \qquad (4.3.6)$$

applying the fixpoint property to the simultaneous fixpoint (4.3.4)

Hence $$\nu_1 = \mathit{Fm}(\nu_1,\nu_2) \qquad (4.3.7)$$
and $$\nu_2 = \mathit{Fm}(\nu_1,\nu_2) \qquad (4.3.8)$$

applying the fixpoint property to the two μ-expressions, (4.3.5) (4.3.6). This symmetry allows us to reverse the Bekić-Scott argument to obtain the fourth form

$$\text{fairmerge} = \nu X.\mu Y.\mathit{Fm}(\mu Z.\mathit{Fm}(Z,X),Y) \qquad (4.3.9)$$

We now have four forms for fairmerge. Note incidentally, that we now have also, from (4.3.7) that

$$\text{fairmerge} = \mathit{Fm}(\text{fairmerge},\text{fairmerge})$$

so that our relation is certainly a fixpoint of the original Fm - but neither its maximal nor its minimal fixpoint.

We should show now how to verify that fairmerge corresponds to our original idea. One way to simplify the manipulations needed to verify this is, temporarily, to extend even further the notion of concatenation used in the previous section, so as to apply to triples (x,y,z) of words from $\Sigma^\dagger$, so that $(x,y,z)\ (u,v,w,) \overset{D}{=} (xu,yv,zw)$, to replace λ by the triple $(\lambda,\lambda,\lambda)$ and to define corresponding closure operations on sets of triples. The properties of $*,\dagger,\omega$ given in (4.3) generalise, except that the condition "$\lambda\notin A$" in (1), (5) of (4.3) must be strengthened to "$x,y,z\in A \Rightarrow x \neq \lambda$ and $y \neq \lambda$ and $z \neq \lambda$". With this device, $\mathit{Fm}(Y,Y)$ simplifies to

$$\mathit{Fm}(X,Y) = A \cup BX \cup CY$$
$$\text{where } A = \{(\lambda,x,x) \mid x\in\Sigma^\dagger\} \cup \{(x,\lambda,x) \mid x\in\Sigma^\dagger\}$$
$$B = \{(\sigma,\lambda,\sigma) \mid \sigma\in\Sigma\}$$
$$C = \{(\lambda,\sigma,\sigma) \mid \sigma\in\Sigma\}$$

then
$$\begin{aligned}\text{fairmerge} &= \nu X.\mu Z.(A \cup BZ \cup C\,\mu Y.(A \cup BX \cup CY))\\ &= \nu X.\mu Z.(A \cup BZ \cup CC^*(A \cup BX))\\ &= \nu X.B^*(A \cup CC^*(A \cup BX))\\ &= \nu X.((B^* \cup CC^*)A \cup B^*CC^*BX)\\ &= (B^*CC^*B)^\dagger(B^* \cup CC^*)A\\ &= (B^*CC^*B)^\omega \cup (B^*CC^*B)^*(B^* \cup CC^*)A\end{aligned}$$

since B^*CC^*B contains no triple with a null component. The final expressions simplifies to

$$\text{fairmerge} = (B^*CC^*B)^\omega \cup (B \cup C)^*A \tag{4.3.10}$$

With the given definition of A,B,C, the reader should see that this is indeed the expected fair merge. The term $(B \cup C)^*A$ defines merges on pairs one of which is finite - in which case fairmerge is not in question. The term $(B^*CC^*B)^\omega$ provides the <u>fair</u> merges of pairs of infinite sequences.

Fairmerge is, of course, commutative and associative, in the sense that

$$(x,y,z) \in \text{fairmerge} \Leftrightarrow (y,x,z) \in \text{fairmerge} \tag{4.3.11}$$
$$(x,y,u) \in \text{fairmerge} \;\&\; (u,z,v) \in \text{fairmerge} \tag{4.3.12}$$
$$\Rightarrow (y,z,w) \in \text{fairmerge} \;\&\; (x,w,v) \in \text{fairmerge, for some } w.$$

An easy proof of commutativity springs from the two symmetric forms (4.3.3) and (4.3.9). But it is not clear how best to prove associativity formally. In view of the fact that concatenation is not cocontinuous, dual Scott induction seems to be of limited used in proofs, and other algebraic insights seem to be called for to obtain general principles applicable to expressions involving occurrences of ν as well as μ. Recent work by Tiuryn [9] by Arnold and Nivat [1] and by Wadge and Faustini [10] seem to suggest some directions in which such insights might be sought.

5. AN EXAMPLE LANGUAGE

In the appendix is specified a relational semantics for a toy language with a fair parallel constraint, using the notational devices introduced in this article and a format adapted from various models in the Scott-Strachey style. In order to accommodate parallelism, the semantic functions $\mathcal{M}[\![C]\!]$ <u>(relational meaning)</u>, $\mathcal{T}[\![C]\!]$ <u>(termination domain)</u> are factored, so that

$$\mathcal{M}[\![C]\!] = \mathrm{RM}(\mathcal{N}[\![C]\!])$$
$$\mathcal{T}[\![C]\!] = \mathrm{TD}(\mathcal{N}[\![C]\!])$$

where $\mathcal{N}[\![C]\!]$ is an intermediate entity, the set of abstract paths corresponding to the command C

$$\mathcal{N}[\![C]\!] \subseteq \mathcal{R}_2(S)^{\dagger}$$

Abstract paths are therefore finite or infinite sequences of state relations, with the intention that each element of a sequence corresponds to an atomic ("uninterruptible") action. The choice of abstract paths, rather than name sequences (or trees), is comparatively arbitrary, and is motivated by the wish to factor out inessential syntactic (and other) detail at the earliest opportunity.

It should now be possible to foresee how proofs might be shaped in a properly formulated formal system. We can consider the task of proving that the command

C : (D <u>par</u> <u>while</u> B <u>do</u> <u>skip</u>) (5.1)

always terminates, assuming D is atomic and makes B false.

1. Let ATOMIC (Ξ) abbreviate

   $$(\forall\xi)\ \xi \in \Xi \Rightarrow \mathrm{length}\ (\xi) \leqslant 1 \qquad (5.2)$$

   and TOT (Ξ) abbreviate

   $$\mathrm{TD}\ (\Xi) = S \qquad (5.3)$$

   We must establish:

   $$\text{if ATOMIC } (\mathcal{D}[\![D]\!]) \text{ and } \mathrm{RM}(\mathcal{D}[\![D]\!]\mathcal{E}[\![B]\!]) = \emptyset \text{ then } \mathrm{TOT}(\mathcal{N}[\![C]\!]) \qquad (5.4)$$

2. The following general results should be available if ATOMIC (Ξ_i), i=1,2,3, and $\lambda \notin \Xi_2$

   $$\text{then } \mathrm{FM}(\Xi_1, \Xi_2{}^{\dagger}\Xi_3) = \Xi_2{}^{*}\ (\Xi_1(\Xi_2{}^{\dagger}\Xi_3) \cup \Xi_3\Xi_1) \qquad (5.5.)$$

   $$\mathrm{TOT}(\Xi_1)\ \&\ \mathrm{RM}\ (\Xi_1) = \emptyset \Rightarrow \mathrm{TOT}\ (\Xi_1\Xi_2) \qquad (5.6.)$$

   $$\mathrm{TOT}\ (\Xi_1)\ \&\ \mathrm{TOT}\ (\Xi_2) \Rightarrow \mathrm{TOT}\ (\Xi_1\Xi_2) \qquad (5.7)$$

   $$\&\ \mathrm{TOT}\ (\Xi_1 \cup \Xi_2) \qquad (5.8)$$

   $$\&\ \mathrm{TOT}\ (\Xi_1{}^{*}\Xi_2) \qquad (5.9)$$

   $$\mathrm{ATOMIC}\ (\Xi) \Rightarrow \mathrm{TOT}\ (\Xi) \qquad (5.10)$$

   $$\mathrm{ATOMIC}\ (\mathcal{E}[\![C]\!]) \qquad (5.11)$$

   $$\mathrm{ATOMIC}\ (\overline{\mathcal{E}}[\![C]\!]) \qquad (5.12)$$

3. From the semantics

$$\mathcal{N}\,[\![C]\!] = \mathcal{N}\,[\![(D\ \underline{par}\ \underline{while}\ B\ \underline{do}\ skip)]\!]$$

$$= FM(\mathcal{D}[\![D]\!],\ \mathcal{E}[\![B]\!]^{\dagger}\bar{\mathcal{E}}[\![B]\!]\ )$$

$$= \mathcal{E}[\![B]\!]^{*}(\mathcal{D}[\![D]\!](\mathcal{E}[\![B]\!]^{\dagger}\ \bar{\mathcal{E}}[\![B]\!])\ \cup\ \bar{\mathcal{E}}[\![B]\!]\mathcal{D}[\![D]\!]) \qquad (5.13)$$

from (5.5.)

Expand $\mathcal{E}[\![B]\!]^{\dagger}\ \bar{\mathcal{E}}[\![B]\!] = (\mathcal{E}[\![B]\!]\mathcal{E}[\![B]\!]^{\dagger}\ \bar{\mathcal{E}}[\![B]\!]\ \cup\ \bar{\mathcal{E}}[\![B]\!])$ (5.14)

TOT ($\mathcal{N}[\![C]\!]$) then follows, using, in turn, (5.6), (5.7), (5.8) and (5.9) to work the TOT property outwards.

We hope these very sketchy ideas will serve as a stimulus towards a more detailed and rigorous development of such a formal system. Our prime aim in this article has been to show that the fairness property can be reasonably characterised in a relational style, and this, we feel, has now been achieved.

References.

[1] Arnold, A., Nivat, M. : Metric interpretations of infinite trees and semantics of nondeterministic recursive programs. Rapport I.T.-3-78. Equipe Lilloise d'Informatique Theorique, 1978.

[2] de Bakker, J.W.: Semantics and termination of nondeterministic recursive programs. pp 435-477 in *Automata, Languages and Programming*, ed. Michaelson, Milner, Edinburgh University Press, 1976.

[3] de Bakker, J.W., de Roever, W.P.: A calculus for recursive program schemes, pp. 167-196 in *Automata, Languages and Programming*, ed. Nivat, North Holland 1973.

[4] Dijkstra, E.W., *A Discipline of Programming*. Prentice-Hall, 1976.

[5] Plotkin, G., A powerdomain construction. 452-487, SIAM J. Comp. 5 (1976).

[6] Hitchcock, P., Park, D., Induction rules and termination proofs. pp. 225-251 in *Automata, Languages and Programming*, ed. Nivat, North Holland 1973.

[7] de Roever, W.P., Dijkstra's predicate transformer, nondeterminism, recursion and termination, in *Mathematical Foundations of Computer Science*, Springer Lecture Notes 45, 1976.

[8] Smyth, M., Power domains. 23-26 J. Comp. Sys. Sci. 16 (1978)

[9] Tiuryn, J., Continuity problems in the power-set algebra of infinite trees. (presented at 4e Colloque de Lille, 1979), Warsaw, 1979.

[10] Wadge, W.W.: An extensional treatment of dataflow deadlock. 285-299 in *Semantics of Concurrent Computation*, Springer Lecture Notes 70, 1979.

APPENDIX.

<u>Relational Semantics for a toy language involving fair parallelism, and permitting unbounded nondeterminism:</u>

Syntactic Classes and Notation:

$C \in \mathrm{Cmd}$	(Commands)
$B \in \mathrm{BCond}$	(Basic Conditions)
$D \in \mathrm{BCmd}$	(Basic Commands)

Syntax:

$C ::= D \mid \underline{\text{skip}} \mid \underline{\text{abort}} \mid (C_1 \,;\, C_2) \mid \underline{\text{if}}\ B\ \underline{\text{then}}\ C_1\ \underline{\text{else}}\ C_2 \mid \underline{\text{while}}\ B\ \underline{\text{do}}\ C \mid$
$(C_1\ \underline{\text{or}}\ C_2) \mid (C_1\ \underline{\text{par}}\ C_2)$

Basic Semantic Notions and Notation:

$x \in S$	(States)
$X \in P(S)$	(State-sets)
$R \in R_2(S)$	(State relations)
$\xi \in R_2(S)^\dagger$	(Relation sequences, Paths)
$\Xi \in P(R_2(S)^\dagger)$	(Path sets)
$\Phi \in P(R_2(S)^\dagger \times S^2)$	
$\Psi \in P(R_2(S)^\dagger \times S)$	
$D : \mathrm{BCmd} \to R_2(S)^\dagger$	(Basic command specs)
$B : \mathrm{BCond} \to P(S)$	(Basic condition specs)

Derived Semantic Notions:

$N : \mathrm{Cmd} \to P(R_2(S)^\dagger)$	(Abstract paths)
$M : \mathrm{Cmd} \to R_2(S)$	(Denoted relation)
$T : \mathrm{Cmd} \to P(S)$	(Termination domain)
$E, \overline{E} : \mathrm{BCond} \to P(R_2(S)^\dagger)$	

Auxiliary Relations:

$\mathrm{run} = \mu\Phi.(\{(\lambda, x, x) \mid x \in S\} \cup \{(R\xi, x, y) \mid (\exists z)((x, z) \in R \;\&\; (\xi, z, y) \in \Phi)\})$

$\mathrm{RM}(\Xi) = \{(x, y) \mid (\exists\xi)\xi \in \Xi \;\&\; (\xi, x, y) \in \mathrm{run}\}$

$\mathrm{tdom} = \mu\Psi.(\{(\lambda, x) \mid x \in S\} \cup \{(R\xi, x) \mid (\forall z)((x, z) \in R \Rightarrow (\xi, z) \in \Psi)\})$

$\mathrm{TD}(\Xi) = \{x \mid (\forall\xi)(\xi \in \Xi \Rightarrow (\xi, x) \in \mathrm{tdom})\}$

fairmerge -- see main text (4.3.2), (4.3.3).

$FM(\Xi_1, \Xi_2) = \{\xi \mid (\exists\xi_1)(\exists\xi_2)\xi_1 \in \Xi_1, \xi_2 \in \Xi_2, (\xi_1, \xi_2, \xi) \in \text{fairmerge}\}$

<u>Semantic Equations</u>:

$\mathcal{E}[\![B]\!] = \{<\{(x, x) \mid x \in \mathcal{B}[\![B]\!]\}>\}$

$\overline{\mathcal{E}}[\![B]\!] = \{<\{(x, x) \mid x \notin \mathcal{B}[\![B]\!]\}>\}$

$\mathcal{M}[\![C]\!] = RM(\mathcal{N}[\![C]\!])$

$\mathcal{T}[\![C]\!] = TD(\mathcal{N}[\![C]\!])$

$\mathcal{N}[\![D]\!] = \mathcal{D}[\![D]\!]$

$\mathcal{N}[\![\underline{\text{skip}}]\!] = \{\lambda\}$

$\mathcal{N}[\![\text{abort}]\!] = \mathcal{R}_2(S)^\omega$

$\mathcal{N}[\![(C_1; C_2)]\!] = \mathcal{N}[\![C_1]\!]\mathcal{N}[\![C_2]\!]$

$\mathcal{N}[\![\underline{\text{if}}\ B\ \underline{\text{then}}\ C_1\ \underline{\text{else}}\ C_2]\!] = \mathcal{E}[\![B]\!]\mathcal{N}[\![C_1]\!] \cup \overline{\mathcal{E}}[\![B]\!]\mathcal{N}[\![C_2]\!]$

$\mathcal{N}[\![\underline{\text{while}}\ B\ \underline{\text{do}}\ C]\!] = (\mathcal{E}[\![B]\!]\mathcal{N}[\![C]\!])^\dagger\ \overline{\mathcal{E}}[\![B]\!]$

$\mathcal{N}[\![(C_1\ \underline{\text{or}}\ C_2)]\!] = \mathcal{N}[\![C_1]\!] \dot{\cup} \mathcal{N}[\![C_2]\!]$

$\mathcal{N}[\![(C_1\ \underline{\text{par}}\ C_2)]\!] = FM(\mathcal{N}[\![C_1]\!], \mathcal{N}[\![C_2]\!])$