stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 23/75 DECEMBER

W.P. DE ROEVER

CALL-BY-VALUE VERSUS CALL-BY-NAME: A PROOF-THEORETIC COMPARISON

Prepublication

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 68A05

ACM-Computing Reviews-category: 5.24

Call-By-Value versus Call-By-Name: A proof-theoretic comparison [*)]

by

W.P. de Roever

ABSTRACT

Minimal fixed point operators were introduced by Scott and De Bakker in order to describe the input-output behaviour of recursive procedures. As they considered recursive procedures acting upon a monolithic state only, i.e., procedures acting upon one variable, the problem remained open how to describe this input-output behaviour in the presence of an arbitrary number of components which as a parameter may be either called-by-value or called-by-name. More precisely, do we need different formalisms in order to describe the input-output behaviour of these procedures for different parameter mechanisms, or do we need different minimal fixed point operators within the same formalism, or do different parameter mechanisms give rise to different transformations, each subject to the same minimal fixed point operator? Using basepoint preserving relations over cartesian products of sets with unique basepoints, we provide a single formalism in which the different combinations of call-by-value and call-by-name are represented by different products of relations, and in which only one minimal fixed point operator is needed. Moreover this mathematical description is axiomatized, thus yielding a relational calculus for recursive procedures with a variety of possible parameter mechanisms.

KEY WORDS & PHRASES: *parameter mechanisms, call-by-value, call-by-name, minimal fixed points, relational semantics, recursive procedures*

---

*) This paper is not for review, it is meant for publication elsewhere.

0. STRUCTURE OF THE PAPER

The reader is referred to section 1.2 for a leisurely written motivation of the contents of this paper.

Chapter 1. Section 1.1 deals with the relational description of various programming concepts, and introduces as a separate concept the parameter list each parameter of which may be either called-by-value or called-by-name. In section 1.2 Manna and Vuillemin's indictment of call-by-value as rule of computation is analyzed and refuted by demonstrating that call-by-value is as amenable to proving properties of programs as call-by-name.

Chapter 2. Using basepoint preserving relations over cartesian products of sets with unique basepoints, we demonstrate in section 2.1 how a variety of possible parameter mechanisms can be described by using different products of relations. In section 2.2 these relations are axiomatized.

1. PARAMETER MECHANISMS, PROJECTION FUNCTIONS, AND PRODUCTS OF RELATIONS

1.1 *The relational description of programs and their properties*

The present paper presents an axiomatization of the input-output behaviour of recursive procedures, which manipulate as values neither labels nor procedures, and the parameters of which may be either called-by-value or called-by-name. It will be argued that, in case all parameters are called-by-name, we may confine ourselves, without restricting the generality of our results, to procedures with procedure bodies in which at least one parameter is invoked, describing calls of the remaining ones by suitably chosen constant terms.

The main vehicle for this axiomatization is a language for binary relations, which is rich enough to express the input-output behaviour of programming concepts such as the composition of statements, the conditional, the assignment, systems of procedures which are subject to the restriction stated above and which call each other recursively, and lists of parameters each of which may be either called-by-value or called-by-name.

EXAMPLE 1.1. Let D be a domain of initial states, intermediate values and final states. The *undefined* statement L: goto L is expressed by the *empty* relation Ω over D. The *dummy* statement is expressed by the *identity* relation E over D.

Define the *composition* $R_1;R_2$ of relations R_1 and R_2 by $R_1;R_2 = \{<x,y> \mid \exists z[<x,z> \in R_1 \text{ and } <z,y> \in R_2]\}$. Obviously this operation expresses the composition of statements.

In order to describe the *conditional* if p then S_1 else S_2, one first has to transliterate p: Let D_1 be p^{-1}(true) and D_2 be p^{-1}(false), then the predicate p is uniquely determined by the pair $<p,p'>$ of disjoint subsets of the identity relation defined by: $<x,x> \in p$ iff $x \in D_1$, and $<x,x> \in p'$ iff $x \in D_2$, cf. KARP [6]. If R_i is the input-output behaviour of S_i, $i = 1,2$, the relation described by the conditional above is $p;R_1 \cup p';R_2$.

Let π_i: $D^n \to D$ be the projection function of D^n on its i-th component, $i = 1,\ldots,n$, let the *converse* $\breve{R}$ of a relation R be defined by $\breve{R} = \{<x,y> \mid <y,x> \in R\}$, and let $R_1,\ldots,R_n$ be arbitrary relations over D. Consider $R_1;\breve{\pi}_1 \cap \ldots \cap R_n;\breve{\pi}_n$. This relation consists exactly of those pairs $<x,<y_1,\ldots,y_n>>$ such that $<x,y_i> \in R_i$ for $i = 1,\ldots,n$. *Thus this expression terminates in* x *iff all its components* R_i *terminate in* x. Observe the analogy with the following: The evaluation of a list of parameters called-by-value terminates iff the evaluation of all its parameters terminates.

In case of a state vector of n components, an *assignment* to the i-th component of the state, $x_i := f(x_1,\ldots,x_n)$, is expressed by $\pi_1;\breve{\pi}_1 \cap \ldots \cap \pi_{i-1};\breve{\pi}_{i-1} \cap R;\breve{\pi}_i \cap \pi_{i+1};\breve{\pi}_{i+1} \cap \ldots \cap \pi_n;\breve{\pi}_n$, where the input-output behaviour of f is expressed by R. This description satisfies Hoare's axiom for the assignment (cf. section 2.2.3 of DE ROEVER [3]).

Note that the input-output behaviour of systems of recursive procedures has not been expressed above; this will be taken care of by extending our language for binary relations with minimal fixed point operators, introduced by SCOTT and DE BAKKER [9].

Our use of the parameter list as a separate programming concept merits some comment. In ALGOL 60 the evaluation of the parameter list

$(f_1(\xi),\ldots,f_n(\xi))$ is part of the execution of the procedure call $f(f_1(\xi),\ldots,f_n(\xi))$, with ξ denoting the state vector. In case all parameters are called-by-value one might introduce $[f_1(\xi),\ldots,f_n(\xi)]$ as a separate programming concept with the following semantics: execution of $[f_1(\xi),\ldots,f_n(\xi)]$ amounts to the independent evaluation of the values of $f_1(\xi),\ldots,f_n(\xi)$, and results in the n-tuple consisting of these values. Provided all state components which are accessed in the original procedure body of f are also contained in its parameter list, the procedure call $f(f_1(\xi),\ldots,f_n(\xi))$ can then be replaced by an expression of the form $[f_1(\xi),\ldots,f_n(\xi)];P$, where P has no parameters and operates upon a state the components of which are accessed by the projection functions $\pi_1,\ldots,\pi_n$.

The generalization of this parameter list construct to the case where parameters may also be called-by-name dictates our restriction, that, in case all parameters are called-by-name, we must confine ourselves to procedures with procedure bodies in which at least one parameter is invoked. This will be explained next.

Given a terminating call of a procedure some parameters of which are called-by-value, the remaining ones being called-by-name, the very fact of termination of this call guarantees termination of the evaluation of the parameter expressions which are called-by-value; however, the termination of this call guarantees the termination of the evaluation of a parameter expression which is called-by-name only in case its value is actually needed inside the procedure body. Thus the evaluation of some parameter expressions need not terminate al all. If one then separates the parameter list from the acutal procedure call as above, one is faced with the problem that in the output of the generalized parameter list one has to handle the undefined components. In order to complete an operationally partially defined n-tuple to an output which is a formally well-defined n-tuple, we introduce a formal element, the so-called *basepoint*, whose function is merely to represent the operationally undefined components. Thus, a basepoint represents a nonterminating computation *whose value is simply not asked for*, and hence may not be transformed into any operationally well-defined value, for otherwise the relevance of our theory to actual programming gets lost. On the other hand, in case of a terminating procedure call of which *none* of its parameters terminates, e.g., the call f("L:goto L","L:goto L") of the integer procedure f(x,y);f := 1, the separation of the parameter list from the call results in an expression of the form ["L:goto L","L:goto L"];P with P always producing an *operationally completely undefined* value as input; i.e., P transforms an operationally undefined value into an operationally well-defined value, in violation of the above condition. We resolve this conflict by describing calls of those procedures, which produce an operationally well-defined output by not looking at any component of their input state, by suitably chosen constant terms. E.g., any call $f(f_1(x),f_2(s))$ of f declared above, is described by $U^{2,1};p_1$, with $p_1 = \{<1,1>\}$ and $U^{2,1} = (I\times I) \times I$, where I denotes the set of integers. Hence we may assume that, in case all parameters are called-by-name, a procedure asks for the value of at least one component of its input, and that consequently, in case of a terminating call, the evaluation of the corresponding parameter expression terminates.

Next we demonstrate how certain concepts, which we need in formulating correctness properties of programs, can be expressed within the rational framework.

EXAMPLE 1.2. Let the input-output behaviour of programs S, S_1 and S_2 be described by R, R_1 and R_2, and let the (partial) predicates p and q be represented by the pairs $<p,p'>$ and $<q,q'>$ of disjoint subsets of the identity relation, cf. example 1.1. With D as above let the *universal* relation U be defined by $U = D \times D$. $R_1 \subseteq R_2$ and $R_2 \subseteq R_1$ together express *equality* of R_1 and R_2, and will be abbreviated by $R_1 = R_2$. S_1 and S_2 are called *equivalent* iff $R_1 = R_2$. $p \subseteq R;\check{R}$ and $p \subseteq R;U$ both express *termination* of S provided p is satisfied. $\check{R};R \subseteq E$ expresses *functionality* of R, i.e., R describes the graph of a function.

Correctness in the sense of HOARE [5], {p}S{q}, amounts to: *if* x *satisfies predicate* p *and program* S *terminates for input* x *with output* y, *then* y *satisfies predicate* q, and is expressed by $p;R \subseteq R;q$.

The "∘" operator is defined by $R\circ p = R;p;\check{R} \cap E$. This operator has been investigated in DE BAKKER & DE ROEVER [1] in order to prove (and express) various properties of while statements, and has been independently described in DIJKSTRA [4] using the term "predicate-transformer".*) It satisfies

*) Note added in proof: In case R is functional, Dijkstra's predicate-transformer and our "∘" operator amounts to the same. In case R is non-determinate, however, these operators are different.

$R;p;\breve{R} \cap E = \{<x,y> \mid <x,y> \in E$ and $<x,y> \in R;p;\breve{R}\} = \{<x,y> \mid x=y$ and $\exists z[<x,z> \in R, <z,z> \in p$, and $<z,y> \in \breve{R}]\} = \{<x,x> \mid \exists z[<x,z> \in R$ and $<z,z> \in p]\}$. Thus, if R expresses the input-output behaviour of procedure f, and $<p,p'>$ expresses the boolean procedure p, p(f(x)) = true iff $<x,x> \in R \circ p$. If we take for p the identically true predicate, represented by $<E,\Omega>$, $<x,x> \in R \circ E$ iff R is defined in x, i.e., $R \circ E$ expresses the *domain of convergence* of R. Note that $R;p;\breve{R} \cap E = R;p;U \cap E$.

1.2. *Parameter mechainsms and products of relations*

Although in this section mostly partial functions are used, it is stressed that the formalism to-be-developed concerns a calculus of relations.

Given a set D and functions $f\colon D \to D$, $g\colon D \times D \to D$, and $h\colon D \times D \times D \to D$

(*) $\qquad <x,y,z> \longmapsto <f(y),g(x,y),h(x,z,x)>$

certainly describes a function of $D \times D \times D$ into itself. For a relational description this element-wise description is not appropriate. Therefore, when dealing with functions between or with binary relations over finite cartesian products of sets, one introduces projection functions (cf. example 1.1) in order to cope with the notion of coordinates in a purely functional (relational) way, thus suppressing any explicit mention of variables. E.g., (*) describes the function $(\pi_2;f,(\pi_1,\pi_2);g,(\pi_1,\pi_3,\pi_1);h)$. Again, this function has been described component-wise, its third component being $(\pi_1,\pi_3,\pi_1);h$. This does not necessarily imply that

(**) $\qquad (\pi_2;f,(\pi_1,\pi_2);g,(\pi_1,\pi_3,\pi_1);h);\pi_3 = (\pi_1,\pi_3,\pi_1);h$

holds! E.g., consider the following: f, g and h are *partial* functions, and, for some $<a,b,c> \in D \times D \times D$, f(b) is undefined, but g(a,b) and h(a,c,a) are well-defined. Therefore $<f(b),g(a,b),h(a,c,a)>$ is undefined as one of its components is undefined.

The problem whether or not (**) is valid turns out to depend on the particular product of relations one wishes to describe, or, in case of the *input-out behaviour of procedures, on the particular parameter mechanism used.*

In order to understand this, consider the values of fv(1,0) and fn(1,0), with integer procedures fv and fn declared by

<u>integer</u> <u>procedure</u> fv(x,y); <u>value</u> x,y; <u>integer</u> x,y; fv := <u>if</u> x=0 <u>then</u> 0 <u>else</u> fv(x-1,fv(x,y))

and

<u>integer</u> <u>procedure</u> fn(x,y); <u>integer</u> x,y; fn := <u>if</u> x=0 <u>then</u> 0 <u>else</u> fn(x-1,fn(x,y)).

Application of the computation rules of the ALGOL 60 report leads to the conclusion that the value of fv(1,0) is *un*defined and the value of fn(1,0) is *well*-defined and equal to 0.

In order to describe this difference in terms of different products of relations and projection functions, we first discuss two possible products of relations: the *call-by-value* product, which resembles the call-by-value concept from the viewpoint of convergence, and the *call-by-name* product, which incorporates certain properties of the call-by-name concept.

Call-by-value product: Let f_1 and f_2 be partial functions from D to D, then the call-by-value product of f_1 and f_2 is defined by $[f_1,f_2] = f_1;\breve{\pi}_1 \cap f_2;\breve{\pi}_2$, cf. example 1.1.

This product satisfies the following properties:

(1) $[f_1,f_2](x) = \langle y_1,y_2\rangle$ iff $f_1(x)$ and $f_2(x)$ are both defined in x, and $f_1(x) = y_1$ and $f_2(x) = y_2$.

(2) $[f_1,f_2];\pi_1 \subseteq f_1$, as $f_2(x)$, whence $\langle f_1(x),f_2(x)\rangle$, and therefore $\pi_1([f_1,f_2](x))$, may be undefined in x, although $f_1(x)$ is well-defined.

(3) In order to transform $[f_1,f_2];\pi_1$ we therefore need an expression for the domain of convergence of f_2. Using the "$\circ$" operator introduced in example 1.2, this expression is supplied for by $f_2\circ E$, as $f_2\circ E = \{\langle x,x\rangle \mid \exists y[y=f_2(x)]\}$, as follows from example 1.2. Thus we obtain $[f_1,f_2];\pi_1 = f_2\circ E\ ;f_1$.

Call-by-name product: Let f_1 and f_2 be given as above. For the call-by-name product $[f_1\times f_2]$ of f_1 and f_2 we stipulate $[f_1\times f_2];\pi_i$, i = 1,2. Hence $\pi_i([f_1\times f_2](x) = f_i(x)$, even if $f_{3-i}(x)$ is undefined, i = 1,2. The justification of this property orginates from ALGOL 60 call-by-name parameter mechanism for which the requirement of replacing the formal parameters by the corresponding actual parameters within the text of the procedure body prior to its execution leads to a situation in which evaluation of a particular actual parameter takes place *independent* of the convergence of the other actual parameters. Models for this product are given in the next chapter.

Before expressing the difference between f_1 and f_2 in the more technical terms of our relational formalism, we discuss the opinion of MANNA and VUILLEMIN [7] concerning call-by-value and call-by-name. We quote: "In discussing recursive programs, the key problem is: *What is the partial function* f *defined by a recursive program* P?" There are two viewpoints:

(a) *Fixpoint approach*: Let it be the unique least fixpoint f_p.

(b) *Computational approach*: Let it be the computed function f_C for some given computation rule C (such as call-by-name or call-by-value). We now come to an interesting point: all theory for proving properties of recursive programs in actually based on the assumption that the function defined by a recursive program is exactly the least fixpoint f_p. That is, the fixpoint approach is adopted. *Unfortunately, almost all programming languages are using an implementation of recursion (such as call-by-value) which does not necessarily lead to the least fixpoint.* Hence they conclude: "... existing computer systems whould be modified, and language designers and implementors should look for computation rules which always lead to the least fixpoint. Call-by-name, for example, is such a computation rule ...".

At this point the reader is forced to conclude, that, according to Manna and Vuillemin, call-by-value should be *discarded* (as a computation rule).

Before arguing, that, *quite to the contrary, call-by-value is as suitable for proofs as call-by-name is,* (the latter being accepted by Manna c.s.), we present their argumentation for indictment of the former rule of computation.

Consider again the recursive procedure f defined by

(***) $f(x,y) \Leftarrow \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ f(x-1, f(x,y))$

They observe that evaluation of $f(x,y)$, (1) using call-by-name, results in computation of $\lambda x,y.$ <u>if</u> $x \geq 0$ <u>then</u> 0 <u>else</u> $\perp$, (2) using call-by-value, results in computation of $\lambda x,y.$ <u>if</u> $x = 0$ <u>then</u> 0 <u>else</u> $\perp$, provided y is defined (where $\perp$ is a formal element expressing operational undefinedness). Then they argue that the minimal fixed point of the transformation

$$T = \lambda X \cdot \lambda x,y \cdot \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ X(x-1,X(x,y))$$

according to the rules of the λ-calculus, where, e.g. $(\lambda u,v\cdot u)\langle x,y\rangle = x$ *holds, independent of the value of* y *being defined or not*, can be computed, for k a positive natural number, by a sequence of approximations of the form

$$T^k(\Omega) = \lambda x,y.\ \underline{\text{if}}\ x = 0\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ \ldots\ \underline{\text{if}}\ x = k-1\ \underline{\text{then}}\ 0\ \underline{\text{else}}\ \perp.$$

Hence the minimal fixed point $\bigcup_{i=1}^{\infty} T^i(\Omega)$ of T equals $\lambda x,y.$ <u>if</u> $x \geq 0$ <u>then</u> 0 <u>else</u> $\perp$. The observation that this minimal fixed point coincides with the computation of (***) using call-by-name, but is clearly different from the computation of (***) using call-by-value, then leads them to denounce call-by-value as a computation rule.

We shall demonstrate that computation of the minimal fixed point of the transformation implied by (***) *gives the call-by-value solution, when adopting the call-by-value product, while computation of the minimal fixed point of this transformation using the call-by-name product results in the call-by-name solution.* Hence we come to the conclusion that *the minimal fixed point of a transformation depends on the particular relational product used, i.e., on the axioms and rules of the formal system one applies in order to compute this minimal fixed point.*

We are now in a position to comment upon Manna and Vuillemin's point of view: as it happens they work with a formal system in which minimal fixed points coincide with recursive solutions computed with call-by-name as rule of computation. Quite correctly they observe that within such a system call-by-value does not necessarily lead to computation of the minimal fixed point. Only this observation is too narrow a basis for discarding call-by-value as rule of computation in general, keeping the wide variety of formal systems in mind.

The transformation implied by (***), using call-by-value as parameter mechanism, is expressed within our formalism by

$$\tau_v(X) = [\pi_1;p_0,\pi_2];\pi_1 \cup [\pi_1;\check{S},X];X$$

where (i) p_0 is only defined for 0 with $p_0(0) = 0$, (ii) $\check{S}$ is the converse of the successor function S, whence $\check{S}(n) = n - 1$, $n \in \mathbb{N}$, $n \geq 1$.

It will be demonstrated that the minimal fixed point $\bigcup_{i=1}^{\infty} \tau_v^i(\Omega)$ of this transformation is equivalent with $\pi_1;p_0$, which is in our formalism the expression for the call-by-value solution of (***):

(1) $\tau_v^1(\Omega) = [\pi_1;p_0,\pi_2];\pi_1$ and $[\pi_1;p_0,\pi_2];\pi_1 = \pi_2 \circ E\ ;\pi_1;p_0$, by a property of the call-by-value product; as totality of π_2 implies $\pi_2 \circ E = E$, we obtain $\tau_v^1(\Omega) = \pi_1;p_0$.

(2) $\tau_v^2(\Omega) = \pi_1;p_0 \cup [\pi_1;\check{S},\pi_1;p_0];\pi_1;p_0$. For $[\pi_1;\check{S},\pi_1;p_0]\langle x,y\rangle$ to be defined, both $(\pi_1;\check{S})\langle x,y\rangle$ and $(\pi_1;p_0)\langle x,y\rangle$ must be defined, i.e., both $x \geq 1$ and $x = 0$ have to hold. As these requirements are contradictory, $[\pi_1;\check{S},\pi_1;p_0];\pi_1;p_0 = \Omega$, and therefore $\tau_v^2(\Omega) = \pi_1;p_0$.

(3) Assuming that $\tau_v^k(\Omega) = \pi_1;p_0$, one argues similarly that $\tau_v^{k+1}(\Omega) = \pi_1;p_0$.

(4) Hence $\bigcup_{i=1}^{\infty} \tau_v^i(\Omega) = \pi_1;p_0$, which corresponds with $\lambda x,y.$ <u>if</u> $x = 0$ <u>then</u> 0 <u>else</u> $\perp$.

The transformation implied by (***), using call-by-name as parameter mechanism, is expressed by

$$\tau_n(X) = [\pi_1;p_0\times\pi_2];\pi_1 \cup [\pi_1;\check{S}\times X];X.$$

We demonstrate that the minimal fixed point $\bigcup_{i=1}^{\infty} \tau_n^i(\Omega)$ of this transformation corresponds with $\lambda x,y.$ <u>if</u> $x \geq 0$ <u>then</u> 0 <u>else</u> $\perp$, Manna and Vuillemin's call-by-name solution of (***):

(1) $\tau_n^1(\Omega) = [\pi_1;p_0\times\pi_2];\pi_1$ and $[\pi_1;p_0\times\pi_2];\pi_1 = \pi_1;p_0$, by definition of the call-by-name product; clearly $\pi_1;p_0$ corresponds with $\lambda x,y.$ <u>if</u> $x = 0$ <u>then</u> 0 <u>else</u> $\perp$.

(2) $\tau_n^2(\Omega) = \pi_1;p_0 \cup [\pi_1;\check{S}\times\pi_1;p_0];\pi_1;p_0$, by (1); as $[\pi_1;\check{S}\times\pi_1;p_0];\pi_1 = \pi_1;\check{S}$, we have $\tau_n^2(\Omega) = \pi_1;p_0 \cup \pi_1;\check{S};p_0$, corresponding with x,y. <u>if</u> $x = 0$ <u>then</u> 0 <u>else</u> <u>if</u> $x = 1$ <u>then</u> 0 <u>else</u> $\perp$.

(3) Assume $\tau_n^k(\Omega) = \pi_1;p_0 \cup \pi_1;\check{S};p_0 \cup \ldots \cup \pi_1\underbrace{;\check{S};\ldots\check{S}}_{(k-1)\text{times}};p_0$. As $\tau_n^{k+1}(\Omega) =$

$= \pi_1;p_0 \cup [\pi_1;S\times\tau_n^k(\Omega)];\tau_n^k(\Omega)$, it follows form the assumption that $\tau_n^{k+1}(\Omega) = \pi_1;p_0 \cup \pi_1;S;p_0 \cup \ldots \cup {}_1\underbrace{;S;\ldots S;}_{(k)\ \text{times}}p_0$, which corresponds with

$\lambda x,y$. if $x = 0$ then 0 else ... if $x = k$ then 0 else $\perp$.

(4) Hence $\bigcup_{i=1}^{\infty} \tau_n^i(\Omega) = \bigcup_{i=1}^{\infty} \pi_1\underbrace{;S;\ldots S;}_{(i-1)\text{times}}p_0$, corresponding with $\lambda x,y$. if $x \geq 0$ then 0 else $\perp$.

2. A CALCULUS FOR RECURSIVE PROCEDURES WITH VARIOUS PARAMETER MECNANISMS

2.1. *The interpretation of products of relations*

In chapter 1 we demonstrated how the call-by-value and call-by-name parameter mechanisms could be described (from the viewpoint of convergence) within the relational framework by introduction of a call-by-value product of relations, which has been axiomatized in DE ROEVER [2,3], and a call-by-name product of relations, which will be discussed in the present section. In particular, we introduce a product of relations describing a parameter list, some components of which are called-by-value, the remaining ones being called-by-name. Section 2.2.2 contains an axiomatization of *all* these products. By replacing in the axiom system of section 5 of DE ROEVER [2] or chapter 2 of DE ROEVER [3] axioms C_1 and C_2 (the axioms for projection functions upon which our axiomatization of the call-by-value product was based) by the new axioms of section 2.2.2, we obtain a calculus for recursive procedures with various parameter mechanisms.

It has been argued in section 1.1 that the interpretation of the call-by-name product requires the introduction of a special element to each domain, the so-called basepoint, the function of which is merely to complete an operationally partially defined n-tuple to a formally well-defined n-tuple by representing the operationally undefined components, in case these might simply not be invoked within a procedure body (and hence are potentially redundant).

Now the very fact, that the introduction of a basepoint is so closely connected with a relation being undefined in some point, suggests using Scott's undefined value $\perp$, cf. SCOTT [8] as basepoint; an originally partial function then becomes a total function, which assigns the formal value $\perp$ to those elements for which the original function was undefined, and the same applies to relations: formally they become total. However, when considering *converses* of such relations-made-total, we are stuck for the following reason: *an operationally undefined value should never be transformed by any relation into an operationally well-defined value*, since otherwise the relevance to programming of a theory of such relations gets lost, for once a computer initiates an unending computation it will not produce any definite value (if left to itself). Thus we refrain from the transition of basepoints to undefined values in general.

Prior to interpreting the call-by-name product, we first define the cartesian product of domains with basepoints: The product of domains $D_1,\dots,D_n$ with basepoints $\underline{pt}_1,\dots,\underline{pt}_n$, which are contained in $D_1,\dots D_n$ respectively, is the cartesian product of $D_1,\dots,D_n$ with basepoint $<\underline{pt}_1,\dots,\underline{pt}_n>$. □

Next we define our admissable relations. The requirement that a basepoint should not be transformed into an operationally defined value, implies conversely that, due to the presence of the conversion operator, an operationally well-defined value should never be transformed into a basepoint. Hence we must observe the following two restrictions when interpreting relations over domains with basepoints:

(1) *A basepoint should be transformed into a basepoint.* ...(2.1.1)

(2) *Only a non-basepoint can be transformed into a non-basepoint.* ...(2.1.2)

EXAMPLE 2.1. Let $D_1,\dots,D_n$ be domains with basepoints, $\underline{pt}_1,\dots,\underline{pt}_n$, respectively, then the projection function $\pi_i: D_1 \times\dots\times D_n \to D_i$ is defined as follows:

$$\pi_i(<x_1,\dots,x_n>) = \begin{cases} x_i, \text{ provided } x_i \neq \underline{pt}_i, \\ \underline{pt}_i, \text{ in case } x_j = \underline{pt}_j,\ j=1,\dots,n, \\ \text{undefined, otherwise,} \end{cases} \quad \dots(2.1.3)$$

for $i = 1,\dots,n$. □

At last we are in a position to discuss the interpretation of the call-by-name product:

Let D, $D_1,\dots,D_n$ be domains with basepoints $\underline{pt}$, $\underline{pt}_1,\dots,\underline{pt}_n$, and $R_1,\dots R_n$ be binary relations such that $R_i \subseteq D \times D_i$, for $i = 1,\dots,n$, which satisfy

(2.1.1) and (2.1.2). Then $[R_1 \times \ldots \times R_n]$ is interpreted as follows:

$$[R_1 \times \ldots \times R_n] = \bigcup_{\substack{I \subseteq \{1,\ldots,n\} \\ \text{and } I \neq \emptyset}} \{<x,<y_1,\ldots,y_n>> \mid xR_jy_j \text{ for } j \in I, \text{ and } y_j = \underline{pt}_j \text{ for } j \in \{1,\ldots,n\}-I\}.$$

For example, $[R_1 \times R_2] = \{<x,<y_1,\underline{pt}_2>> \mid xR_1y_1\} \cup \{<x,<\underline{pt}_1,y_2>> \mid xR_2y_2\} \cup$ $\cup \{<x,<y_1,y_2>> \mid xR_iy_i,\ i=1,2\}$. In particular, $[E \times \Omega] = \{<x,<x,\underline{pt}>> \mid x \in D\}$. The reader should verify himself, using the interpretation of π_i in example 2.1, that $[R_1 \times \ldots \times R_n];\pi_i = R_i$, $i=1,\ldots,n$. Notice also that $[R_1 \times \ldots \times R_n];(\pi_{j_1};\breve{\pi}_1 \cap \ldots \cap \pi_{j_k};\breve{\pi}_k) = (R_{j_1};\breve{\pi}_1 \cap \ldots \cap R_{j_k};\breve{\pi}_k)$, for $1 \leq j_1 < \ldots < j_k \leq n$, i.e., a list of n parameters called-by-name, of which only the j_1-st,...,j_k-th components are invoked, is equivalent with the list of k invoked parameters which are called-by-value.

Nevertheless, for a relational calculus this element-wise description is not appropriate. Therefore we introduce the following constants:
Let D, $D_1,\ldots,D_n$ be as above, then the relation constants $*_1,\ldots,*_n$ are defined by

$$<<x_1,\ldots,x_n>,x> \in *_i \text{ iff } \begin{cases} x=\underline{pt}, \text{ in case } x_j=\underline{pt}, j=1,\ldots,n, \\ x\ D-\{\underline{pt}\}, \text{ provided } x_i=\underline{pt}_i, \text{ and} \\ \quad x_j \neq pt_j \text{ for at least one } j,\ j \neq i, \\ \text{undefined, otherwise,} \end{cases} \quad \ldots (2.1.4)$$

for i = 1,...,n. □

The introduction of these constant is motivated by the following property: $\breve{*}_i$ transforms any non-basepoint into any n-tuple, the i-th component of which is $\underline{pt}_i$, provided this n-tuple is not composed out of basepoints altogether. Hence we have

$$[R_1 \times \ldots \times R_n] = \bigcup_{i=1}^{n} (R_i;\breve{\pi}_i \cup \breve{*}_i).$$

In general, the ALGOL 60 parameter mechanism allows within the same parameter list for a combination of parameters called-by-value and called-by-name. This combination of parameter mechanisms results in a product of relations, which reflects this mixed structure.

Let procedure f have for simplicity a parameter list of n components, the first k components of which are called-by-value, and the last n-k components of which are called-by-name. Let ξ denote a statevector. As in our formal model of description the parameter list is separated from the procedure call, cf. section 1.1, the separation of $(f_1(\xi),\dots,f_n(\xi))$ from the call $f(f_1(\xi),\dots,f_n(\xi))$ results in an expression of the form $[f_1(\xi),\dots,f_n(\xi)]^{\underline{\text{value}}\{1,\dots,k\}};P$, where the value of $[f_1(\xi),\dots,f_n(\xi)]^{\underline{\text{value}}\{1,\dots,k\}}$ is only defined in case the evaluation of the first k parameters, the call-by-value parameters $f_1(\xi),\dots,f_k(\xi)$, terminates. Therefore a relational description of this parameter list is obtained by introducing a product of relations $[R_1\times\dots\times R_n]^{\underline{\text{value}}\{1,\dots,k\}}$, which satisfies

$$[R_1\times\dots\times R_n]^{\underline{\text{value}}\{1,\dots,k\}};\pi_i = R_1\circ E;\dots;R_k\circ E;R_i,$$

for $i = 1,\dots,n$.

In general, such products are interpreted as follows:
Let $D, D_1,\dots,D_n$ be given as above. Let $J\subseteq\{1,\dots,n\}$ and let $I=\{1,\dots,n\}-J$. Then $[R_1\times\dots\times R_n]^{\underline{\text{value}}\ J}$ is defined by:

$$(\bigcap_{j\in J} R_j;\breve{\pi}_j)\cap(\bigcap_{i\in I}(R_i;\breve{\pi}_i\cup\breve{*}_i)).\quad\Box\qquad\dots(2.1.5)$$

Observe finally that both the call-by-value and the call-by-name product can be obtained as special case of the product defined above by taking $J=\{1,\dots,n\}$ and $J=\emptyset$, respectively.

2.2. *A calculus for recursive procedures with various parameter mechanisms*

2.2.1. *Language*

The language MU^* for basepoint preserving relations over cartesian products of domains with unique basepoints, which has minimal fixed point operators, is a simple extension of the language MU, defined in DE ROEVER [2,3].

The syntax of MU^* is obtained from the syntax of MU by adding for $n\geq 2$ the logical relation constants $*_i^{\eta_1\times\dots\times\eta_n,\eta}$, for $i = 1,\dots,n$, and all

$\eta_1,\dots,\eta_n$ and η, to the elementary terms of *MU*.

The semantics of MU^* is determined by considering binary relations over domains with unique basepoints only, observing restrictions (2.1.1) and (2.1.2), and interpreting $\pi_i^{\eta_1\times\dots\times\eta_n,\eta_i}$ and $*_i^{\eta_1\times\dots\times\eta_n,\eta}$ as in (2.1.3) and (2.1.4), for $i = 1,\dots,n$, and all $\eta_1,\dots,\eta_n$, and η. Hence

(1) $m(\Omega^{\eta,\theta}) = \{<\underline{pt}_\eta,\underline{pt}_\theta> \mid \underline{pt}_\eta \in D_\eta,\ \underline{pt}_\theta \in D_\theta\}$, $(E^{\eta,\eta}) = \{<x,x> \mid x \in D_\eta\}$,
$m(U^{\eta,\theta}) = \{<x,y> \mid x \in D_\eta - \{\underline{pt}_\eta\},\ y \in D_\theta - \{\underline{pt}_\eta\}\} \cup \{<\underline{pt}_\eta,\underline{pt}_\theta>\}$,

(2) interpretations of elementary relation constants $A^{\eta,\theta}$ satisfy
$m(\Omega^{\eta,\theta}) \subseteq m(A^{\eta,\theta}) \subseteq m(U^{\eta,\theta})$,

(3) interpretations of pairs $<p^{\eta,\eta},p'^{\eta,\eta}>$ of boolean constants satisfy
$m(\Omega^{\eta,\eta}) \subseteq m(p^{\eta,\eta}) \subseteq m(E^{\eta,\eta})$, $m(\Omega^{\eta,\eta}) \subseteq m(p'^{\eta,\eta}) \subseteq m(E^{\eta,\eta})$, and
$m(p^{\eta,\eta}) \cap m(p'^{\eta,\eta}) = m(\Omega^{\eta,\eta})$,

(4) interpretations of relation variables $X^{\eta,\theta}$ satisfy $m(\Omega^{\eta,\theta}) \subseteq m(X^{\eta,\theta}) \subseteq$
$\subseteq m(U^{\eta,\theta})$,

(5) the operators "$\cup$", "$\cap$", "$;$", "$\smile$" are interpreted as usual, and the "$-$" operator is interpreted by $m(\overline{X^{\eta,\theta}}) = (m(U^{\eta,\theta}) - m(X^{\eta,\theta})) \cup m(\Omega^{\eta,\theta})$,

(6) $\mu_i X_1\dots X_n[\sigma_1,\dots,\sigma_n]$ is interpreted as the i-th component of the (unique) minimal fixed point of the transformation $<m(\sigma_1),\dots,m(\sigma_n)>$ acting on n-tuples of relations satisfying (2.1.1) and (2.1.2), $i = 1,\dots,n$.
Observe that it follows from the definitions that any fixed point of $<m(\sigma_1),\dots,m(\sigma_n)>$ acting on these relations satisfies (2.1.1) and (2.1.2); hence the minimal fixed point of this transformation, being the intersection of all these fixed points, satisfies (2.1.1) and (2.1.2) also.

2.2.2. *Axiomatization*

MU^* is axiomatized by replacing in the axiom system for *MU*, as contained in chapter 2 of DE ROEVER [3] or section 5 of DE ROEVER [2], axioms C_1 and C_2 by BP_1, BP_2, BP_3, BP_4 and BP_5 below: For $n \geq 2$,

$$BP_1 : \vdash *_1;\breve{*}_1 \cap \dots \cap *_n;\breve{*}_n = \Omega^{\eta_1\times\dots\times\eta_n,\eta_1\times\dots\times\eta_n}$$

$$BP_2 : \vdash *_i^{\eta_1\times\dots\times\eta_n,\xi} = *_i^{\eta_1\times\dots\times\eta_n,};U^{\theta,\xi},\ i = 1,\dots,n,$$

$$BP_3 : \vdash \pi_i;\breve{\pi}_i \cap *_i;\breve{*}_i = \Omega^{\eta_1\times\dots\times\eta_n,\eta_1\times\dots\times\eta_n},\ i = 1,\dots,n,$$

$$BP_4 : \vdash (\pi_1;\breve{\pi}_1 \cup *_1;\breve{*}_1) \cap \dots \cap (\pi_n;\breve{\pi}_n \cup *_n;\breve{*}_n) = E^{\eta_1\times\dots\times\eta_n,\eta_1\times\dots\times\eta_n}$$

BP_5 : *For all* $I \neq \{1,\dots,n\}$ s.t. $I \supsetneq \emptyset$:

$$\vdash \bigcap_{i\in I} X_i;Y_i = \{(\bigcap_{i\in I} X_i;\breve{\pi}_i) \cap (\bigcap_{i\in\{1,\dots,n\}-I} \breve{*}_i^{\eta_1\times\dots\times\eta_n,\theta})\};$$
$$\{(\bigcap_{i\in I} \pi_i;Y_i) \cap (\bigcap_{i\in\{1,\dots,n\}-I} *_i^{\eta_1\times\dots\times\eta_n,\xi})\},$$

and for $I = \{1,\dots,n\}$:

$$\vdash \bigcap_{i\in I} X_i;Y_i = (\bigcap_{i\in I} X_i;\breve{\pi}_i);(\bigcap_{i\in I} \pi_i;Y_i),$$

with π_i of type $(\eta_1\times\dots\times\eta_n,\eta_i)$, and X_i and Y_i of types (θ,η_i) and (η_i,ξ), respectively, $i = 1,\dots,n$.

The following lemma is proved in DE ROEVER [3]:

<u>LEMMA</u>. *Let* $n \geq 2$, $i = 1,\dots,n$, *and* $j = 1,\dots,n$, *and* $*_i$ *of type* $(\eta_1\times\dots\times\eta_n,\eta_i)$, *then*

a. $\vdash \breve{*}_i;\pi_j = U$, $i\neq j$, *and* $\vdash \breve{*}_i;\pi_i = \Omega$.

b. *For* $n=2$: $\vdash \breve{*}_i;*_j = \Omega$, $i\neq j$, *and* $\vdash \breve{*}_i;*_i = U$.

For $n\geq 3$: $\vdash \breve{*}_i;*_j = U$.

c. $\vdash \breve{\pi}_i;\pi_j = U$, $i\neq j$, *and* $\vdash \breve{\pi}_i;\pi_i = E$.

Let $[X_1\times\dots\times X_n]\xrightarrow{\text{value}} J$ be defined as in (2.1.5). Then corollaries 2.1 and 2.2 follow from the above lemma and the definitions.

COROLLARY 2.1. $\vdash [X_1\times\dots\times X_n]\xrightarrow{\text{value}}\{j_1,\dots,j_k\};\pi_i =$
$= X_{j_1}\circ E;\dots;X_{j_k}\circ E;X_i$, $i = 1,\dots,n$.

COROLLARY 2.2. $\vdash [X_1\times\dots\times X_n]\xrightarrow{\text{value}}\{j_1,\dots,j_m\};(\pi_{k_1};\breve{\pi}_1 \cap\dots\cap \pi_{k_p};\breve{\pi}_p)=$
$= X_{j_1}\circ E;\dots;X_{j_m}\circ E;(X_{k_1};\breve{\pi}_1\cap\dots\cap X_{k_p};\breve{\pi}_p)$.

REFERENCES

[1] DE BAKKER, J.W., & W.P. DE ROEVER, *A calculus for recursive program schemes*, <u>in</u> Proc. IRIA Symposium on Automata, Formal Languages and Programming, M. Nivat (ed.), North-Holland, Amsterdam, 1972.

[2] DE ROEVER, W.P., *Operational and mathematical semantics for recursive polyadic program schemata (extended abstract)*, <u>in</u> Proceedings of Symposium and Summer School "Mathematical Foundations of Computer

Science", 3-8 September 1973, High Tatras, Czechoslovakia, pp. 298-298.

[3] DE ROEVER, W.P., *Recursion and parameter mechanisms: an axiomatic approach*, in Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken, July 29 - August 2, 1974, Edited by Jacques Loeckx, Lecture Notes in Computer Science no. 14, Springer-Verlag, Berlin etc., 1974.

[4] DIJKSTRA, E.W., *A simple axiomatic basis for programming languages constructs*, Indagationes Mathematicae, 36 (1974) 1-15.

[5] HOARE, C.A.R., *An axiomatic basis for computer programming*, Comm. ACM, 12 (1969) 576-583.

[6] KARP, R.M., *Some applications of logical syntax to digital computer programming*, Thesis, Harvard University, 1959.

[7] MANNA, Z., & J. VUILLEMIN, *Fixpoint approach to the theory of computation*, Comm. ACM, 15 (1972) 528-536.

[8] SCOTT, D., *Lattice theory, data types, and semantics*, in NYU Symposium on formal semantics, pp. 64-106, Princeton, 1972.

[9] SCOTT, D., & J.W. DE BAKKER, *A theory of programs*, Unpublished notes, IBM Seminar, Vienna, 1969.