stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 20/74 MAY

W.P. DE ROEVER

RECURSION AND PARAMETER MECHANISMS:
AN AXIOMATIC APPROACH

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): 02J10, 68A05

ACM - Computing Reviews - category: 5.24

RECURSION AND PARAMETER MECHANISMS: AN AXIOMATIC APPROACH *)

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.

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. In section 2.1 we define a language for binary relations over cartesian products of sets which has minimal fixed point operators, and in section 2.2 a calculus for recursive procedures, the parameters of which are called-by-value, is developed by axiomatizing the semantics of this language.

Chapter 3. The calculus presented in section 2.2 is applied to prove an equivalence due to Morris, and Wright's regularization of linear procedures; then lists are axiomatized, and a correctness proof for a version of the Schorr-Waite marking algorithm is given, first informally and then formally.

*) To appear in *Proceedings of the Second Symposium on Automata, Languages and Programming*, Saarbrücken, July 29 - August 2, 1974, Lecture Notes in Computer Science, Springer-Verlag, Berlin etc.

Chapter 4. Using basepoint preserving relations over cartesian products of sets with unique basepoints, we demonstrate in section 4.1 how a variety of possible parameter mechanisms can be described by using different products of relations. In section 4.2 these relations are axiomatized.

Chapter 5. In section 5.1 we formulate some conclusions and briefly discuss the topic of providing operational, interpreter-based, semantics for the various programming concepts, the mathematical semantics of which we axiomatized in chapters 2 and 4. Finally, section 5.2 is devoted to related work.

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: <u>goto</u> 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 $R_1;R_2 = \{\langle x,y\rangle \mid \exists z[\langle x,z\rangle \in R_1 \text{ and } \langle z,y\rangle \in R_2]\}$. Obviously this operation expresses the composition of statements.

In order to describe the *conditional* <u>if</u> p <u>then</u> S_1 <u>else</u> S_2, one first has to transliterate p: Let D_1 be p^{-1}(<u>true</u>) and D_2 be p^{-1}(<u>false</u>), then the predicate p is uniquely determined by the pair $\langle p,p'\rangle$ of disjoint subsets of the identity relation defined by: $\langle x,x\rangle \in p$ iff $x \in D_1$, and $\langle x,x\rangle \in p'$ iff $x \in D_2$, cf. Karp [18]. 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 $\pi_i\colon D^n \to D$ be the projection function of D^n on its i-th component, i=1,...,n, let the *converse* $\check{R}$ of a relation R be defined by $\check{R} = \{\langle x,y\rangle \mid \langle y,x\rangle \in R\}$, and let $R_1,\ldots,R_n$ be arbitrary relations over D. Consider $R_1;\check{\pi}_1 \cap \ldots \cap R_n;\check{\pi}_n$. This relation consists exactly of those pairs $\langle x,\langle y_1,\ldots,y_n\rangle\rangle$ such that $\langle x,y_i\rangle \in R_i$ for i=1,...,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, $x_i := f(x_1,\dots,x_n)$, is expressed by $\pi_1;\breve{\pi}_1 \cap \dots \cap \pi_{i-1};\breve{\pi}_{i-1} \cap R;\breve{\pi}_i \cap \cap \pi_{i+1};\breve{\pi}_{i+1} \cap \dots \cap \pi_n;\breve{\pi}_n$, 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). □

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 in chapter 2 with minimal fixed point operators, introduced by Scott and De Bakker [29].

Our use of the parameter list as a separate programming concept merits some comment. In ALGOL 60 the evaluation of the parameter list $(f_1(\xi),\dots,f_n(\xi))$ is part of the execution of the procedure call $f(f_1(\xi),\dots,f_n(\xi))$, with ξ denoting the state vector. In case all parameters are called-by-value one might introduce $[f_1(\xi),\dots,f_n(\xi)]$ as a separate programming concept with the following semantics: execution of $[f_1(\xi),\dots,f_n(\xi)]$ amounts to the independent evaluation of the values of $f_1(\xi),\dots,f_n(\xi)$, 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 $f(f_1(\xi),\dots,f_n(\xi))$ can then be replaced by an expression of the form $[f_1(\xi),\dots,f_n(\xi)];P$, where P has no parameters and operates upon a state the components of which are accessed by the projection functions $\pi_1,\dots,\pi_n$.

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 one 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 at all. If one then separates the parameter list from the actual 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 terminate, 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 defined* output, even if its formalized input consists of a pair of two basepoints, signalling 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. 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 relational 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 $\langle p,p'\rangle$ and $\langle q,q'\rangle$ 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 [16], {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 [6] in order to prove (and express) various properties of while statements, and has been independently described in Dijkstra [11] using the term "predicate-transformer". It satisfies $R;p;\check{R} \cap E = \{\langle x,y\rangle \mid \langle x,y\rangle \in E$ and $\langle x,y\rangle \in R;p;\check{R}\} = \{\langle x,y\rangle \mid x=y$ and $\exists z[\langle x,z\rangle \in R, \langle z,z\rangle \in p$, and $\langle z,y\rangle \in \check{R}]\} =$
$= \{\langle x,x\rangle \mid \exists z[\langle x,z\rangle \in R$ and $\langle z,z\rangle \in p]\}$. Thus, if R expresses the input-output behaviour of procedure f, and $\langle p,p'\rangle$ expresses the boolean procedure p, $p(f(x)) =$
$=$ true iff $\langle x,x\rangle \in R\circ p$. If we take for p the identically true predicate, represented by $\langle E,\Omega\rangle$, $\langle x,x\rangle \in R\circ E$ iff R is defined in x, i.e., $R\circ E$ expresses the *domain of convergence* of R. Note that $R;p;\check{R} \cap E = R;p;U \cap E$. □

1.2. *Parameter mechanisms 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 $h\colon D \times D \times D \to D$,

$$(*) \qquad \langle x,y,z\rangle \longmapsto \langle f(y),g(x,y),h(x,z,x)\rangle$$

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 $\langle a,b,c\rangle \in D \times D \times D$, $f(b)$ is undefined, but $g(a,b)$ and $h(a,c,a)$ are well-defined. Therefore $\langle f(b),g(a,b),h(a,c,a)\rangle$ 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-output 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

```
integer procedure fv(x,y); value x,y; integer x,y; fv:= if x=0 then 0 else
                                                    fv(x-1,fv(x,y))
```

and

```
integer procedure fn(x,y); integer x,y; fn:= if x=0 then 0 else 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 $[f_1,f_2] = f_1;\check{\pi}_1 \cap f_2;\check{\pi}_2$, 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 $f_1(x) = y_1$, $f_2(x) = y_2$.

(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 $[f_1 \times f_2];\pi_i = f_i$, 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 originates from the 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. Possible models for this product are given in chapter 4. □

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 [20] 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 the theory for proving properties of recursive programs is 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 should 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

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

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

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

according to the rules of the λ-calculus, where, e.g. $(\lambda u,v.u)\langle x,y\rangle = x$ *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

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

Hence the minimal fixed point $\bigcup_{i=1}^{\infty} T^i(\Omega)$ of T equals $\lambda x,y.\ \underline{if}\ x\geq 0\ \underline{then}\ 0\ \underline{else}\ \bot$. 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

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

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

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

(2) $\tau_v^2(\Omega) = \pi_1;p_0 \cup [\pi_1;\breve{S},\pi_1;p_0];\pi_1;p_0$. For $[\pi_1;\breve{S},\pi_1;p_0]\langle x,y\rangle$ to be defined, both $(\pi_1;\breve{S})\langle x,y\rangle$ 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, $[\pi_1;\breve{S},\pi_1;p_0];\pi_1;p_0 = \Omega$, and therefore $\tau_v^2(\Omega) = \pi_1;p_0$.

(3) Assuming $\tau_v^k(\Omega) = \pi_1;p_0$, 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 $\lambda x,y$. if x=0 then 0 else $\perp$. □

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

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

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

(1) $\tau_n(\Omega) = [\pi_1;p_0\times\pi_2];\pi_1$ 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 $\lambda x,y$. if x=0 then 0 else $\perp$.

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

(3) Assume $\tau_n^k(\Omega) = \pi_1;p_0 \cup \pi_1;\breve{S};p_0 \cup \ldots \cup \pi_1;\underbrace{\breve{S};\ldots\breve{S}}_{(k-1)\text{times}};p_0$. As $\tau_n^{k+1}(\Omega) = \pi_1;p_0 \cup$ $\cup\, [\pi_1;\breve{S}\times\tau_n^k(\Omega)];\tau_n^k(\Omega)$, it follows from the assumption that $\tau_n^{k+1}(\Omega) = \pi_1;p_0 \cup$ $\cup\, \pi_1;\breve{S};p_0 \cup \ldots \cup \pi_1;\underbrace{\breve{S};\ldots\breve{S}}_{k\text{ times}};p_0$, which corresponds with $\lambda x,y$. if x=0 then 0 else ... if x=k then 0 else $\perp$.

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

2. A CALCULUS FOR RECURSIVE PROCEDURES, THE PARAMETERS OF WHICH ARE CALLED-BY-VALUE

2.1. *Language*

In this section we define *MU*, a language for binary relations over cartesian products of sets, which has minimal fixed point operators in order to characterize the input-output behaviour of recursive procedures.

As the binary relations considered are subsets of the cartesian product of one domain D_η or cartesian product of domains $D_{\eta_1}\times\ldots\times D_{\eta_n}$, and another domain D_θ or cartesian product of domains $D_{\theta_1}\times\ldots\times D_{\theta_n}$, terms $\sigma^{\eta,\theta}$ or $\sigma^{\eta_1\times\ldots\times\eta_n,\theta_1\times\ldots\times\theta_n}$ denoting these relations are typed. Types will not be mentioned or discussed unless explicitly needed, and are formally defined in De Roever [9].

Elementary terms are the individual relation constants $A^{\eta,\theta}, A_1^{\eta,\theta},\ldots$, boolean relation constants $p^{\eta,\eta}, p'^{\eta,\eta},\ldots,q^{\eta,\eta},q'^{\eta,\eta},\ldots$, logical relation constants $\Omega^{\eta,\theta}$, $E^{\theta,\theta}$, $U^{\eta,\theta}$ and $\pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$, i=1,...,n, for the empty, identity and universal relations, and the projection functions, and the relation variables $X^{\eta,\theta}, X_1^{\eta,\theta},\ldots,Y^{\eta,\theta},\ldots$.

Compound terms are constructed by means of the operators ";" (relational or Peirce product), "∪" (union), "∩" (intersection), "$\smile$" (converse and "—" (complementation) and the minimal fixed point operators "μ_i", which bind for i=1,...,n, n different relation variables $X_1^{\eta_1,\theta_1},\ldots,X_n^{\eta_n,\theta_n}$ in n-tuples of terms $\sigma_1^{\eta_1,\theta_1},\ldots,\sigma_n^{\eta_n,\theta_n}$, provided none of these variables occurs in any complemented subterm.

Terms of *MU* are elementary or compound terms. The well-formed formulae of *MU* are called *assertions*, and are of the form $\Phi \vdash \Psi$, where Φ and Ψ are sets of inclusions between terms of the form $\sigma_1^{\eta,\theta} \subseteq \sigma_2^{\eta,\theta}$, the so-called *atomic formulae*.

Free occurrences of the variables $X_1,\ldots,X_n$ in a term σ are occurrences not contained in any subterm $\mu_i\ldots X\ldots[\ldots]$ of σ, and are indicated by writing $\sigma(X_1,\ldots,X_n)$. *Substitution* of terms τ_i for the free occurrences of X_i in $\sigma(X_1,\ldots,X_n)$, i=1,...,n, is denoted by $\sigma(\tau_1,\ldots,\tau_n)$ or $\sigma[\tau_i/X_i]_{i=1,\ldots,n}$; proper care has to be taken not to substitute terms containing free occurrences of $X_1,\ldots,X_n$ within $\mu_i X_1\ldots X_n[\sigma_1,\ldots,\sigma_n]$, a care reflected in the formal definition of substitution contained in De Roever [9].

The (mathematical) semantics *m* of *MU* is defined by:

(1) providing arbitrary (type-restricted) interpretations for the individual relation constants and relation variables, interpreting pairs $<p^{\eta,\eta},p'^{\eta,\eta}>$ of boolean relation constants as pairs $<m(p^{\eta,\eta}),m(p'^{\eta,\eta})>$ of disjoint subsets of the identity relation $m(E^{\eta,\eta})$, and interpreting the logical relation constants $\Omega^{\eta,\theta}$, $E^{\eta,\eta}$ and $U^{\eta,\theta}$, and $\pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$, i=1,...,n, as the empty relation $\emptyset$ ($\subseteq D_\eta \times D_\theta$), the identity relation over D_η, the universal relation $D_\eta \times D_\theta$, and the projection functions with graph $\{<<x_1,\ldots,x_n>,x_i> \mid x_j \in D_{\eta_j},\ j=1,\ldots,n\}$, i=1,...,n,

(2) interpreting ";", "∪", "∩", "$\smile$", "—" as usual,

(3) interpreting μ-terms $\mu_i X_1\ldots X_n[\sigma_1,\ldots,\sigma_n]$ as the i-th component of the minimal fixed point of the functional $<m(\sigma_1),\ldots,m(\sigma_n)>$ acting on n-tuples of relations.

An assertion $\Phi \vdash \Psi$ is *valid* provided for all *m* the following holds: if the inclusions contained in Φ are satisfied by *m*, then the inclusions contained in Ψ are satisfied by *m*.

The main result concerning *MU* is the *union* theorem,

$$m(\mu_i X_1\ldots X_n[\sigma_1,\ldots,\sigma_n]) = \bigcup_{j=0}^{\infty} m(\sigma_i^j), \quad i=1,\ldots,n,$$

with σ_i^j defined by $\sigma_i^0 = \Omega$, $\sigma_i^{j+1} = \sigma_i(\sigma_1^j,\ldots,\sigma_n^j)$, i=1,...,n. This theorem states that the (unique) minimal fixed point of a *continuous* transformation of n-tuples of relations can be obtained by a sequence of finite approximations, and is proved using the *monotonicity*, *continuity* and *substitutivity* properties, cf. De Roever [9]. One of its implications is the validity of *Scott's induction rule*, formulated in section 2.2.4.

2.2. *A calculus for recursive procedures, the parameters of which are called-by-value*

De Bakker and De Roever describe in [6] a calculus for recursive procedures which operate upon an *undivided* (monolithic) state vector. This calculus is generalized in the present section to recursive procedures, operating upon a state vector, the components of which can be accessed by using projection functions; conversely, the relational framework enables us to compose a new state vector from operated-upon components $R_1(\xi),\ldots,R_n(\xi)$ by the call-by-value product $R_1;\breve{\pi}_1 \cap \ldots \cap R_n;\breve{\pi}_n$, which is, as argued in section 1.2, a prerequisite for the relational description of the call-by-value parameter mechanism. We axiomatize projection functions (in section 2.2.3) by introducing the following axiom schemes:

$$C_1:\ \vdash\ \pi_1;\breve{\pi}_1 \cap \ldots \cap \pi_n;\breve{\pi}_n = E^{\eta_1\times\ldots\times\eta_n,\eta_1\times\ldots\times\eta_n}$$

$$C_2:\ \vdash\ X_1;Y_1 \cap \ldots \cap X_n;Y_n = (X_1;\breve{\pi}_1\cap\ldots\cap X_n;\breve{\pi}_n);(\pi_1;Y_1\cap\ldots\cap\pi_n;Y_n).$$

We want to point out that chapter 4 is devoted to a generalization of the results of this chapter to basepoint preserving relations over cartesian products of sets with unique basepoints, a generalization which is motivated by our wish to obtain a formal description of call-by-value and certain aspects of call-by-name.

The axiomatization of *MU* proceeds in four successive stages:

1. In section 2.2.1 we develop the axiomatization of typed binary relations.
2. This axiomatization is extended in section 2.2.2 to boolean constants.
3. The axiomatization of projection functions in section 2.2.3 then results in the axiomatization of binary relations over cartesian products.
4. The additional axiomatization of μ-terms in section 2.2.4 completes the axiomatization of *MU*.

2.2.1. *Axiomatization of typed binary relations*

Consider the following sublanguage of *MU*, called MU_0:
The *elementary* terms of MU_0 are restricted to the individual relation constants, relation variables and logical constants $\Omega^{\eta,\xi}$, $E^{\eta,\eta}$ and $U^{\eta,\xi}$ of *MU*, i.e., boolean constants and projection functions are excluded.
The *compound* terms of MU_0 are those terms of *MU* which are constructed using these basic terms and the ";", "∪", "∩", "˘" and "—" operators, i.e., the "μ_i" operators are excluded.
The *assertions* of MU_0 are those assertions of *MU* whose atomic formulae are inclusions between terms of MU_0. □
MU_0 is axiomatized by the following axioms and rules:

1. The typed versions of the axioms and rules of boolean algebra.
2. The typed version of Tarski's axioms for binary relations (cf. [30]):

$$T_1\ :\ \vdash (X^{\eta,\theta};Y^{\theta,\zeta});Z^{\zeta,\xi} = X^{\eta,\theta};(Y^{\theta,\zeta};Z^{\zeta,\xi})$$

$$T_2\ :\ \vdash \breve{\breve{X}}^{\eta,\xi} = X^{\eta,\xi}$$

$$T_3\ :\ \vdash (X^{\eta,\theta};Y^{\theta,\xi})^{\smile} = \breve{Y}^{\theta,\xi};\breve{X}^{\eta,\theta}$$

T_4 : $\vdash X^{\eta,\xi};E^{\xi,\xi} = X^{\eta,\xi}$

T_5 : $(X^{\eta,\theta};Y^{\theta,\xi}) \cap Z^{\eta,\xi} = \Omega^{\eta,\xi} \vdash (Y^{\theta,\xi};\breve{Z}^{\eta,\xi}) \cap \breve{X}^{\eta,\theta} = \Omega^{\theta,\eta}$

3. U : $\vdash U^{\eta,\xi} \subseteq U^{\eta,\theta};U^{\theta,\xi}$

In the sequel we omit parentheses in our formulae, based on the associativity of binary operators and on the convention that ";" has priority of "∩", which has in turn priority over "∪".

LEMMA 2.1.

a. $X^{\eta,\xi} \subseteq Y^{\eta,\xi} \vdash \breve{X}^{\eta,\xi} \subseteq \breve{Y}^{\eta,\xi}, X^{\eta,\xi};Z^{\xi,\theta} \subseteq Y^{\eta,\xi};Z^{\xi,\theta}, Z^{\theta,\eta};X^{\eta,\xi} \subseteq Z^{\theta,\eta};Y^{\eta,\xi}$

b. $\vdash \Omega^{\eta,\xi};X^{\xi,\theta} = \Omega^{\eta,\theta}, X^{\eta,\xi};\Omega^{\xi,\theta} = \Omega^{\eta,\theta}$

c. $\vdash E^{\eta,\eta};X^{\eta,\xi} = X^{\eta,\xi}$

d. $\vdash U^{\eta,\xi};U^{\xi,\theta} = U^{\eta,\theta}$

e. $\vdash \breve{\Omega}^{\eta,\xi} = \Omega^{\xi,\eta}, \breve{E}^{\eta,\eta} = E^{\eta,\eta}, \breve{U}^{\eta,\xi} = U^{\xi,\eta}$

f. $\vdash X^{\eta,\xi};(Y^{\xi,\theta} \cup Z^{\xi,\theta}) = X^{\eta,\xi};Y^{\xi,\theta} \cup X^{\eta,\xi};Z^{\xi,\theta}, (X^{\xi,\theta} \cup Y^{\xi,\theta});Z^{\theta,\eta} =$
$= X^{\xi,\theta};Z^{\theta,\eta} \cup Y^{\xi,\theta};Z^{\theta,\eta}$

g. $\vdash (X^{\eta,\xi} \cup Y^{\eta,\xi})^{\smile} = \breve{X}^{\eta,\xi} \cup \breve{Y}^{\eta,\xi}, (X^{\eta,\xi} \cap Y^{\eta,\xi})^{\smile} = \breve{X}^{\eta,\xi} \cap \breve{Y}^{\eta,\xi}, \breve{\breve{X}}^{\eta,\xi} = \bar{\breve{X}}^{\eta,\xi}$.

Except for the proof of part d, which is obtained using U and a law of boolean algebra, the proofs for the typed case are similar to the proofs for the untyped case as contained in Tarski [30].

Lemma 2.1.a expresses monotonicity of "˘" and ";". Together with the obvious monotonicity of "∪" and "∩", this will be used in lemma 2.9 to establish monotonicity of syntactically continuous terms in general.

Remarks. 1. Henceforward the laws of boolean algebra are used without explicit reference.

2. *Type indications are omitted provided no confusion arises.*

The proofs of the following two lemmas can be found in De Bakker and De Roever [6].

LEMMA 2.2. $\vdash X;Y \cap Z = X;(\breve{X};Z \cap Y) \cap Z$.

A number of useful properties of relations and functions are collected in lemma 2.3 below. Remember that $X \circ E$ has been defined as $X;\breve{X} \cap E$ (cf. example 1.2). *By convention the "∘" operator has a higher priority then the ";" operator.*

LEMMA 2.3.

a. $\breve{X};X \subseteq E \vdash X;(Y \cap Z) = X;Y \cap X;Z$

b. $X \subseteq E \vdash X = \breve{X}$

c. $\vdash X = X \circ E\ ; X,\ X = X; \breve{X} \circ E,\ X \circ E = X; \breve{X} \cap E,\ X;U = X \circ E\ ;U$

d. $X \subseteq Y,\ \breve{Y};Y \subseteq E \vdash X \circ E;\ Y = X$

e. $\vdash \bigcap_{i=1}^{n} X_i;Y_i = X_1 \circ E; \ldots ; X_n \circ E;\ (\bigcap_{i=1}^{n} X_i;Y_i);\ \breve{Y}_1 \circ E; \ldots ; \breve{Y}_n \circ E.$

2.2.2. *Axiomatization of boolean relation constants*

Partial predicates are represented within *MU* by pairs $<p^{n,n}, p'^{n,n}>$ whose interpretation is restricted to pairs of disjoint subsets of the identity relation corresponding to inverse images of <u>true</u> and <u>false</u>. MU_0 is extended to MU_1 by adding the boolean relation constants of *MU* to the basic terms of MU_0. MU_1 is axiomatized by adding the following two axioms to those of MU_0:

$$P_1 : \vdash p^{n,n} \subseteq E^{n,n},\ p'^{n,n} \subseteq E^{n,n}$$

$$P_2 : \vdash p^{n,n} \cap p'^{n,n} \subseteq \Omega^{n,n}.$$

The axiomatization of MU_1 leads to a theory of conditionals (cf. ex. 1.1), as demonstrated by corollary 2.1, cf. McCarthy [22]. Again, proofs can be found in De Bakker and De Roever [6] or De Roever [9].

LEMMA 2.4. $\vdash \breve{p} = p,\ p;q = p \cap q.$

COROLLARY 2.1. *Using the notation* $(p \to X,Y) = p;X \cup p';Y$, *we have* $\vdash (p \to (p \to X,Y),Z) =$
$= (p \to X,Z),(p \to X,(p \to Y,Z)) = (p \to X,Z),(p \to (q \to X_1,X_2),(q \to Y_1,Y_2)) =$
$= (q \to (p \to X_1,Y_1),(p \to X_2,Y_2)).$

COROLLARY 2.2. $\vdash p;X \cap Y = p;(X \cap Y).$

In example 1.2 we defined the "$\circ$" operator by $X \circ p\ =\ X;p;\breve{X} \cap E$. Its basic properties are collected in lemmas 2.5, 3.2, 3.3, and theorem 3.2. This operator is crucial to a theory of programs since it enables a description of the interaction between programs and predicates. This is demonstrated by the axiomatization both of ordered data structures such as ordered linear lists (cf. De Roever [9]), and of the call-by-value parameter mechanism contained in the following section. For other examples of its use we refer to De Bakker and De Roever [6].

LEMMA 2.5.

a. $\vdash (X;Y) \circ p = X \circ (Y \circ p)$

b. $\vdash (X \cup Y) \circ p = X \circ p \cup Y \circ p$

c. $\vdash (X \cap Y) \circ p = X;p;\breve{Y} \cap E$

d. $\vdash X;p \subseteq X \circ p\ ;X$

e. $\breve{X};X \subseteq E \vdash X;p = X \circ p\ ;X$

f. $X;p \subseteq q;X \vdash X \circ p \subseteq q.$

Observe that from parts d and f of this lemma, we obtain $X \circ p = \bigcap\{q \mid X;p \subseteq q;X\}$.

2.2.3. *Axiomatization of binary relations over cartesian products*

The language MU_2 for binary relations over cartesian products is obtained from MU_1 by adding, for i=1,...,n, projection function symbols $\pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$ to the basic terms of MU_1, for all types concerned. MU_2 is axiomatized by adding the following two axiom schemes to the axioms and rules of MU_1:

$$C_1 : \vdash \pi_1;\breve{\pi}_1 \cap \ldots \cap \pi_n;\breve{\pi}_n = E$$

$$C_2 : \vdash X_1;Y_1 \cap \ldots \cap X_n;Y_n = (X_1;\breve{\pi}_1 \cap \ldots \cap X_n;\breve{\pi}_n);(\pi_1;Y_1 \cap \ldots \cap \pi_n;Y_n),$$

where π_i is of type $<\eta_1\times\ldots\times\eta_n,\eta_i>$, E stands for $E^{\eta_1\times\ldots\times\eta_n,\eta_1\times\ldots\times\eta_n}$, and X_i and Y_i are of types $<\theta,\eta_i>$ and $<\eta_i,\xi>$, respectively, i=1,...,n.

As remarked in example 1.1, an assignment V of the form $x_i := f(x_1,\ldots,x_n)$ is described by a term T of the form $\pi_1;\breve{\pi}_1 \cap \ldots \cap \pi_{i-1};\breve{\pi}_{i-1} \cap R;\breve{\pi}_i \cap \pi_{i+1};\breve{\pi}_{i+1} \cap \ldots \cap \pi_n;\breve{\pi}_n$. Hence Hoare's *axiom* for the assignment (cf. [16]) $\vdash \{p(x_1,\ldots,x_{i-1},f(x_1,\ldots,x_n),x_{i+1},\ldots,x_n)\}x_i := f(x_1,\ldots,x_n)\{p(x_1,\ldots,x_n)\}$ corresponds with the assertion $\vdash T\circ p ;T \subseteq T;p$, as by example 1.2 $\{q_1\}$ V $\{q_2\}$ is expressed by $q_1;R \subseteq R;q_2$ and $p(x_1,\ldots,x_{i-1},f(x_1,\ldots,x_n),x_{i+1},\ldots,x_n) =$ = <u>true</u> iff $<<x_1,\ldots,x_n>,<x_1,\ldots,x_n>> \in T\circ p$. As functionality of f implies $\breve{T};T \subseteq E$ by lemma 2.11 below, this assertion follows from lemma 2.5.e. Thus leads the axiomatization of MU_2 to a theory of assignments.

LEMMA 2.6. *For* i=1,...,n:

a. $\vdash \pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i}\circ E^{\eta_i,\eta_i} = E^{\eta_1\times\ldots\times\eta_n,\eta_1\times\ldots\times\eta_n}$

b. $\vdash \pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i};U^{\eta_i,\xi} = U^{\eta_1\times\ldots\times\eta_n,\xi}$

c. $\vdash \breve{\pi}_i^{\eta_i,\eta_1\times\ldots\times\eta_n};\pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i} = E^{\eta_i,\eta_i}$

d. $\vdash \breve{\pi}_i^{\eta_i,\eta_1\times\ldots\times\eta_n};\pi_j^{\eta_1\times\ldots\times\eta_n,\eta_j} = U^{\eta_i,\eta_j}$, *for* $i\neq j$, j=1,...,n.

Proof. a. Let E_n denote $E^{\eta_1\times\ldots\times\eta_n,\eta_1\times\ldots\times\eta_n}$, then $E_n = (C_1)\ \pi_i;\breve{\pi}_i \cap E_n =$ = (lemma 2.3.c) $\pi_i\circ E^{\eta_i,\eta_i} \subseteq E_n$.

b. $\pi_i;U^{\eta_i,\xi}$ = (lemma 2.3.c) $\pi_i\circ E^{\eta_i,\eta_i};U^{\eta_1\times\ldots\times\eta_n,\xi}$ = (part a above) $U^{\eta_1\times\ldots\times\eta_n,\xi}$.

c. Consider, e.g., n=2,i=1:

E^{η_1,η_1} = (lemma 2.1.d) $E^{\eta_1,\eta_1};E^{\eta_1,\eta_1} \cap U^{\eta_1,\eta_1};U^{\eta_1,\eta_1}$

... = (C_2) $(E^{\eta_1,\eta_1};\breve{\pi}_1 \cap U^{\eta_1,\eta_1};\breve{\pi}_2);(\pi_1;E^{\eta_1,\eta_1} \cap \pi_2;U^{\eta_1,\eta_1})$ =

= (lemma 2.1 and part b above) $\breve{\pi}_1;\pi_1$.

d. Consider, e.g., n=2, i=1 and j=2:

$U^{\eta_1,\eta_2} = E^{\eta_1,\eta_1};U^{\eta_1,\eta_2} \cap U^{\eta_1,\eta_2};E^{\eta_2,\eta_2}$

$$\ldots = (C_2)\ (E^{\eta_1,\eta_1};\breve{\pi}_1 \cap U^{\eta_1,\eta_2};\breve{\pi}_2);(\pi_1;U^{\eta_1,\eta_2} \cap \pi_2;E^{\eta_2,\eta_2}) = \text{(part b above)}\ \breve{\pi}_1;\pi_2.\ \square$$

Already in example 1.1 we signalled tha analogy between $\bigcap_{i=n}^{n} X_i;\breve{\pi}_i$ and a list of parameters called-by-value. From this point of view properties such as $(\bigcap_{i=1}^{n} X_i;\breve{\pi}_i)\circ E^{\eta_1\times\ldots\times\eta_n,\eta_1\times\ldots\times\eta_n} = \bigcap_{i=1}^{n} X_i\circ E^{\eta_i,\eta_i}$ -the computation of such a list terminates iff the computations of its individual members terminates- and $(\bigcap_{i=1}^{n} X_i;\breve{\pi}_i);\pi_j = (\bigcap_{i=1}^{n} X_i\circ E^{\eta_i,\eta_i});X_j$ -the request for the value of a parameter contained in such a list amounts to computation of the individual value of this parameter plus termination of the computation of the other parameters- are intuitively evident. These and similar properties follow from the following lemma and its corollary.

LEMMA 2.7. *For* k,l ≤ n,

$$\vdash X_{i_1}\circ E;\ldots;X_{i_k}\circ E;(\bigcap_{\substack{i_j=s_t,j=1,\ldots,k\\ t=1,\ldots,l}} X_{i_j};Y_{s_t});\breve{Y}_{s_1}\circ E;\ldots;\breve{Y}_{s_l}\circ E =$$

$= (\bigcap_{j=1}^{k} X_{i_j};\breve{\pi}_{i_j});(\bigcap_{t=1}^{l} \pi_{s_t};Y_{s_t})$, *with* π_i *of type* $\langle\eta_1\times\ldots\times\eta_n,\eta_i\rangle$, *and* X_{i_j} *and* Y_{s_t} *of types* $\langle\theta,\eta_{i_j}\rangle$ *and* $\langle\eta_{s_t},\xi\rangle$, *respectively*, i=1,...,n, j=1,...,k, t=1,...,l.

Proof. The case of n=3, k=l=2, $i_1=1$, $i_2=2$, $s_1=2$, $s_2=3$ is representative. Hence we prove $X_1\circ E;X_2\circ E;X_2;Y_2;\breve{Y}_2\circ E;\breve{Y}_3\circ E = (X_1;\breve{\pi}_1 \cap X_2;\breve{\pi}_2);(\pi_2;Y_2 \cap \pi_3;Y_3)$. By lemma 2.6, $X_1;\breve{\pi}_1 \cap X_2;\breve{\pi}_2 = X_1;\breve{\pi}_1 \cap X_2;\breve{\pi}_2 \cap U^{\theta,\eta_3};\breve{\pi}_3$ and $\pi_2;Y_2 \cap \pi_3;Y_3 = \pi_1;U^{\eta_1,\xi} \cap \pi_2;Y_2 \cap \pi_3;Y_3$, whence $(X_1;\breve{\pi}_1 \cap X_2;\breve{\pi}_2);(\pi_2;Y_2 \cap \pi_3;Y_3) = (C_2)\ X_1;U^{\eta_1,\xi} \cap X_2;Y_2 \cap U^{\theta,\eta_3};Y_3$

$\ldots =$ (lemma 2.3.c) $X_1\circ E;U^{\theta,\xi} \cap X_2;Y_2 \cap U^{\theta,\xi};\breve{Y}_3\circ E$

$\ldots =$ (lemma 2.3.e) $X_1\circ E;X_2\circ E;(X_1\circ E;U^{\theta,\xi} \cap X_2;Y_2 \cap U^{\theta,\xi};\breve{Y}_3\circ E);\breve{Y}_2\circ E;\breve{Y}_3\circ E$.

By corollary 2.2, $X_1\circ E;U^{\theta,\xi} \cap X_2;Y_2 \cap U^{\theta,\xi};\breve{Y}_3\circ E = X_1\circ E;X_2;Y_2;\breve{Y}_3\circ E$, whence the result follows by lemma 2.4. □

COROLLARY 2.3. $\vdash (\bigcap_{i=1}^{n} X_i;\breve{\pi}_i)\circ(\bigcap_{i=1}^{n} \pi_i;p_i;\breve{\pi}_i) = X_1\circ p_1;\ldots;X_n\circ p_n$, *with* X_i *of type* $\langle\theta,\eta_i\rangle$ *and* p_i *of type* $\langle\eta_i,\eta_i\rangle$.

Proof. $(\bigcap_{i=1}^{n} X_i;\breve{\pi}_i)\circ(\bigcap_{i=1}^{n} \pi_i;p_i;\breve{\pi}_i) = (C_2)\ (\bigcap_{i=1}^{n} X_i;p_i;\breve{\pi}_i);U^{\eta_1\times\ldots\times\eta_n,\theta} \cap E^{\theta,\theta}$

$\ldots =$ (lemma 2.6.b) $(\bigcap_{i=1}^{n} X_i;p_i;\breve{\pi}_i);\pi_1;U_1^{\eta_1,\theta} \cap E^{\theta,\theta} =$

$=$ (lemma 2.7) $(X_1;p_1)\circ E;\ldots;(X_n;p_n)\circ E;X_1;p_1;U^{\eta_1,\theta} \cap E^{\theta,\theta}$

$\ldots =$ (corollary 2.2 and lemma 2.5.a) $X_1\circ p_1;\ldots;X_n\circ p_n$. □

One of the consequences of lemma 2.7 is

$$\vdash (\bigcap_{i=1}^{n-1} X_i;\breve{\pi}_i);(\bigcap_{i=1}^{n-1} \pi_i;Y_i) = \bigcap_{i=1}^{n-1} X_i;Y_i,$$

with π_i, X_i and Y_i of types $<\eta_1\times\ldots\times\eta_n,\eta_i>$, $<\theta,\eta_i>$ and $<\eta_i,\xi>$, respectively. Assume $\eta_1 = \eta_2 = \ldots = \eta_n$ for simplicity, then, apart from the intended interpretation of π_i as special subset of $D^n \times D$,

"axiom C_2 for n-1, in which $\pi_1,\ldots,\pi_{n-1}$ are interpreted as subsets of $D^{n-1} \times D$ "follows from" axiom C_2 for n, n > 2".

This line of thought may be pursued as follows: Change the definition of type in that only compounds $(\eta_1\times\eta_2)$ are considered, and introduce projection function symbols $\pi_1^{(\eta\times\xi),\eta}$ and $\pi_2^{(\eta\times\xi),\xi}$ only. For n > 2 define $(\eta_1\times\ldots\times\eta_n)$ as $(\ldots((\eta_1\times\eta_2)\times\eta_3)\times\ldots\times\eta_n)$ and $\pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$ as, e.g., for n=3 and i=1,2,3, $\pi_1^{((\eta_1\times\eta_2)\times\eta_3),(\eta_1\times\eta_2)};\pi_1^{(\eta_1\times\eta_2),\eta_1}$, $\pi_1^{((\eta_1\times\eta_2)\times\eta_3),(\eta_1\times\eta_2)};\pi_2^{(\eta_1\times\eta_2),\eta_2}$ and $\pi_2^{((\eta_1\times\eta_2)\times\eta_3),\eta_3}$. Then it is a simple exercise to *deduce* C_1 and C_2 for n=3 from *axioms* C_1 and C_2 for n=2. This indicates that our original approach may be conceived of as a "sugared" version of the more fundamental set-up suggested above. These considerations are related to the work of Hotz on X-categories (cf. Hotz [17]). □

Arbitrary applications of the "$\smile$" operator can be restricted to projection functions, as demonstrated below; this result will be used in section 3.2 to prove Wright's result on the regularization of linear procedures.

LEMMA 2.8. $\vdash \breve{X} = \breve{\pi}_2;(E \cap \pi_1;X;\breve{\pi}_2);\pi_1$.

Proof. We prove $X = \breve{\pi}_1;(E \cap \pi_1;X;\breve{\pi}_2);\pi_2$. The result then follows by lemma 2.3.b.

$\pi_1;X;\breve{\pi}_2 \cap E = (C_1)\ \pi_1;X;\breve{\pi}_2 \cap \pi_1;\breve{\pi}_1 \cap \pi_2;\breve{\pi}_2 =$ (lemmas 2.6.c and 2.3.a) $\pi_1;(X;\breve{\pi}_2 \cap \breve{\pi}_1) \cap \pi_2;\breve{\pi}_2$.

Hence, $\breve{\pi}_1;(\pi_1;X;\breve{\pi}_2 \cap E);\pi_2 =$ (lemma 2.7) $(X;\breve{\pi}_2 \cap \breve{\pi}_1);\pi_2 =$ (lemma 2.7 again) X. □

2.2.4. *Axiomatization of the minimal fixed point operators*

MU is obtained from MU_2 by introducing the "μ_i" operators, and is axiomatized by adding Scott's induction rule *I* and axiom scheme *M*, which are both formulated below, to the axioms and rules of MU_2:

$$I:\quad \frac{\Phi \vdash \Psi[\Omega^{\eta_k,\xi_k}/X_k^{\eta_k,\xi_k}]_{k=1,\ldots,n} \qquad \Phi,\Psi \vdash \Psi[\sigma^{\eta_k,\xi_k}/X_k^{\eta_k,\xi_k}]_{k=1,\ldots,n}}{\Phi \vdash \Psi[\mu_k X_1\ldots X_n[\sigma_1,\ldots,\sigma_n]/X_k^{\eta_k,\xi_k}]_{k=1,\ldots,n}},$$

with Φ *only containing occurrences of* X_i *which are bound (i.e., not free) and* Ψ *only containing occurrences of* X_i *which are not contained in any complemented subterm,* $i=1,\ldots,n$.

$$M : \vdash \{\sigma_j[\mu_i X_1 \ldots X_n[\sigma_1,\ldots,\sigma_n]/X_i]_{i=1,\ldots,n} \subseteq \mu_j X_1 \ldots X_n[\sigma_1,\ldots,\sigma_n]\}_{j=1,\ldots,n}.$$

The basic results about minimal fixed point operators are collected in lemma 2.9, proved in De Bakker and De Roever [6], and lemma 2.10, which asserts that *simultaneous* minimalization by μ_i-terms is equivalent to *successive singular* minimalization by μ-terms, and is proved in Hitchcock and Park [15]. The *modularity* property (corollary 2.4), which is new, is proved in De Roever [9].

LEMMA 2.9.

a. *If* $\tau_1(X_1,\ldots,X_n,Y),\ldots,\tau_n(X_1,\ldots,X_n,Y)$ *are monotonic in* $X_1,\ldots,X_n$ *and* Y, *i.e.* $A_1 \subseteq B_1,\ldots,A_{n+1} \subseteq B_{n+1} \vdash \tau_i(A_1,\ldots,A_{n+1}) \subseteq \tau_i(B_1,\ldots,B_{n+1})$, $i=1,\ldots,n$, *then* $Y_1 \subseteq Y_2 \vdash \{\mu_j X_1 \ldots X_n[\tau_1(X_1,\ldots,X_n,Y_1)\ldots\tau_n(X_1,\ldots,X_n,Y_1)] \subseteq$
$\subseteq \mu_j X_1 \ldots X_n[\tau_1(X_1,\ldots,X_n,Y_2)\ldots\tau_n(X_1,\ldots,X_n,Y_2)]\}_{j=1,\ldots,n}$.

b. (Monotonicity). *If* $\tau(X_1,\ldots,X_n)$ *is syntactically continuous in* $X_1,\ldots,X_n$ *then* τ *is monotonic in* $X_1,\ldots,X_n$, *i.e.*, $X_1 \subseteq Y_1,\ldots,X_n \subseteq Y_n \vdash \tau(X_1,\ldots,X_n) \subseteq \tau(Y_1,\ldots,Y_n)$.

c. (Fixed point property). $\vdash \{\tau_j[\mu_i X_1 \ldots X_n[\tau_1,\ldots,\tau_n]/X_i]_{i=1,\ldots,n} =$
$= \mu_j X_1 \ldots X_n[\tau_1,\ldots,\tau_n]\}_{j=1,\ldots,n}$.

d. (Minimal fixed point property, Park [25]).
$\{\tau_j(Y_1,\ldots,Y_n) \subseteq Y_j\}_{j=1,\ldots,n} \vdash \{\mu_j X_1 \ldots X_n[\tau_1,\ldots,\tau_n] \subseteq Y_j\}_{j=1,\ldots,n}$.

LEMMA 2.10. (Iteration, Scott and De Bakker [29]).

$\vdash \mu_j X_1 \ldots X_{j-1} X_j X_{j+1} \ldots X_n[\sigma_1,\ldots,\sigma_{j-1},\sigma_j,\sigma_{j+1},\ldots,\sigma_n] =$
$= \mu X_j[\sigma_j[\mu_i X_1 \ldots X_{j-1} X_{j+1} \ldots X_n[\sigma_1,\ldots,\sigma_{j-1},\sigma_{j+1},\ldots,\sigma_n]/X_i]_{i\in I}]$, with
$I = \{1,\ldots,j-1,j+1,\ldots,n\}$.

COROLLARY 2.4. (Modularity). *For* $i=1,\ldots,n$,

$\vdash \mu_i X_1 \ldots X_n[\sigma_1(\tau_{11}(X_1,\ldots,X_n),\ldots,\tau_{1m}(X_1,\ldots,X_n)),\ldots,$
$\sigma_n(\tau_{n1}(X_1,\ldots,X_n),\ldots,\tau_{nm}(X_1,\ldots,X_n))] =$
$= \sigma_i(\mu_{i1} X_{11} \ldots X_{nm}[\tau_{11}(\sigma_1(X_{11},\ldots,X_{1m}),\ldots,\sigma_n(X_{n1},\ldots,X_{nm})),\ldots,\tau_{nm}(\ldots)],\ldots,\mu_{im}\ldots)$.

Modularity has some interesting applications, e.g., it reduces the two-page proof of the "tree-traversal" result of De Bakker and De Roever [6] to a two-line proof, as demonstrated below. Let p*A be defined by p*A = $\mu X[p;A;X \cup p']$. This construct describes the while statement while p do A. We quote: "Suppose one wishes to perform a certain action A in all nodes of all trees of a forest (in the sense of Knuth [19], pp.305-307). Let, for x any node, s(x) be interpreted as "has x a son?", and b(x) as "has x a brother?". Let S(x) be: "Visit the first son of x", B(x) be: "Visit the first brother of x", and F(x): "Visit the father of x". The problem posed to us can then be formulated as: Let $T_1 = \mu X[A;(s \to S;X;F,E);(b \to B;X,E)]$, and $T_2 = \mu X[A;(s \to S;X;\ b*(B;X)\ ;F,E)]$. Show that $T_1 = T_2;\ b*(B;T_2)$".

Proof. Apply first corollary 2.4, taking n=1, m=2, $\sigma_1(X,Y) = X;Y$, $\tau_{11}(X) = A;(s \to S;X;F,E)$, and $\tau_{12}(X) = (b \to B;X,E)$, and apply then lemma 2.10. □

The last lemma of this chapter states some sufficient conditions for provability of $\Phi \vdash \breve{\sigma};\sigma \subseteq E$, i.e. *functionality* of σ.

LEMMA 2.11. (Functionality). *The assertion* $\Phi \vdash \breve{\sigma};\sigma \subseteq E$ *is provable if one of the following assertions is provable:*

a. *If* $\sigma = \bigcup_{i=1}^{n} \sigma_i$ *then* $\Phi \vdash \{\sigma_i \circ E;\sigma_j = \sigma_j \circ E;\sigma_i\}_{1 \le i \le j \le n} \cup \{\breve{\sigma}_i;\sigma_i \subseteq E\}_{i=1,\dots,n}$.

b. *If* $\sigma = \sigma_1;\breve{\pi}_1 \cap \dots \cap \sigma_n;\breve{\pi}_n$ *then* $\Phi \vdash \{\breve{\sigma}_i;\sigma_i \subseteq E\}_{i=1,\dots,n}$.

c. *If* $\sigma = \sigma_1;\sigma_2$ *then* $\Phi \vdash \breve{\sigma}_1;\sigma_1 \subseteq E$, $\breve{\sigma}_2;\sigma_2 \subseteq E$.

d. *If* $\sigma = \sigma_1 \cap \sigma_2$ *then* $\Phi \vdash \breve{\sigma}_1;\sigma_1 \subseteq E$ *or* $\Phi \vdash \breve{\sigma}_2;\sigma_2 \subseteq E$ *or* $\Phi \vdash \breve{\sigma}_1;\sigma_2 \subseteq E$ *or* $\Phi \vdash \breve{\sigma}_2;\sigma_1 \subseteq E$.

e. *If* $\sigma = \mu_i X_1 \dots X_n[\sigma_1,\dots,\sigma_n]$ *then* $\Phi,\{\breve{X}_i;X_i \subseteq E\}_{i=1,\dots,n} \vdash \{\breve{\sigma}_i;\sigma_i \subseteq E\}_{i=1,\dots,n}$, *provided* X_i *does not occur free in* Φ, i=1,...,n.

In the following chapter we shall use the following notations:

1. $[\sigma_1,\dots,\sigma_n]$ for $\sigma_1;\breve{\pi}_1 \cap \dots \cap \sigma_n;\breve{\pi}_n$.
2. $[\sigma_1|\dots|\sigma_n]$ for $\pi_1;\sigma_1;\breve{\pi}_1 \cap \dots \cap \pi_n;\sigma_n;\breve{\pi}_n$.

3. APPLICATIONS

3.1. *An example due to Morris*

In [24] Morris proves equivalence of f(x,y) and g(x,y) given by:

f(x,y) ⇐ <u>if</u> p(x) <u>then</u> y <u>else</u> h(f(k(x),y)),

g(x,y) ⇐ <u>if</u> p(x) <u>then</u> y <u>else</u> g(k(x),h(y)).

We present a proof in our framework. The following equivalence is stated without proof:

LEMMA 3.1. $\vdash [A_1|\dots|A_{i-1}|A_i|A_{i+1}|\dots|A_n];\pi_i = [A_1|\dots|A_{i-1}|E|A_{i+1}|\dots|A_n];\pi_i;A_i$.

THEOREM 3.1. (Morris) *Let* $F = \mu X[[p|E];\pi_2 \cup [p'|E];[K|E];X;H]$ *and* $G = \mu Y[[p|E];\pi_2 \cup [p'|E];[K|H];Y]$. *Then* $\vdash F = G$, $[E|H];G = G;H$.

Proof. Let Φ be empty, $\Psi(X,Y) = \{X = Y, [E|H];Y = Y;H\}$, $\sigma(X) = [p|E];\pi_2 \cup [p'|E];[K|E];X;H$ and $\tau(Y) = [p|E];\pi_2 \cup [p'|E];[K|H];Y$. Hence, we must prove

$$\vdash \Psi(\mu X[\sigma(X)], \mu Y[\tau(Y)]) \tag{3.1.1}$$

We *intend* to use Scott's induction rule. Unfortunately, this rule (as formulated in section 2.2.4) does *not* apply to (3.1.1), as, *in case of a simultaneous induction argument*, it only yields results about *components of one simultaneous μ-term*. However, the observation that $\vdash \mu_1 XY[\sigma(X),\tau(Y)] = \mu X[\sigma(X)]$ and $\vdash \mu_2 XY[\sigma(X),\tau(Y)] = \mu Y[\tau(Y)]$ are straightforward applications of iteration (lemma 2.10), gives us the equivalent assertion $\vdash \Psi(\mu_1 XY[\sigma(X),\tau(Y)], \mu_2 XY[\sigma(X),\tau(Y)])$ to which Scott's induction rule *does* apply. Thus, we have to prove:

1. $\vdash \Psi(\Omega,\Omega)$. Obvious.
2. $X = Y, [E|H];Y = Y;H \vdash \sigma(X) = \tau(Y), [E|H];\tau(Y) = \tau(Y);H$.

a. $\sigma(X) = \tau(Y)$: $[p|E];\pi_2 \cup [p'|E];[K|E];X;H =$ (hyp.) $[p|E];\pi_2 \cup [p'|E];[K|E];Y;H =$
$=$ (hyp.) $[p|E];\pi_2 \cup [p'|E];[K|E];[E|H];Y =$ (C_2) $[p|E];\pi_2 \cup [p'|E];[K|H];Y$.

b. $[E|H];\tau(Y) = \tau(Y);H$: $[E|H];([p|E];\pi_2 \cup [p'|E];[K|H];Y) =$
$= [E|H];[p|E];\pi_2 \cup [E|H];[p'|E];[K|H];Y =$ (C_2) $[p|H];\pi_2 \cup [p';K|H;H];Y =$
$=$ (lemma 3.1) $[p|E];\pi_2;H \cup [p';K|H];[E|H];Y =$
$=$ (hyp.) $[p|E];\pi_2;H \cup [p'|E];[K|H];Y;H = ([p|E];\pi_2 \cup [p'|E];[K|H];Y);H$. □

3.2. *Wright's regularization of linear procedures*

In [33] Wright obtains the following results:

a. The class of recursively enumerable subsets of N^2 is the smallest class of sets with the successor relation S as member and closed under the operations "$\smile$", ";" and "$\mu X[Q \cup P;X;R]$", where Q, P and R are subsets of N^2 which are contained in this class.

b. In the proof of part a the main auxiliary result can be generalized to a setting in which N is replaced by any abstract domain $\mathcal{D}$. This generalization is:

$$\vdash \mu X[Q \cup P;X;R] = \breve{\pi}_1;\mu Y[E \cup [P|\breve{R}];Y]\circ(E \cap \pi_1;Q;\breve{\pi}_2);\pi_2 \quad \ldots \quad (3.2.1)$$

In the present calculus (3.2.1) can be proved axiomatically. The following two auxiliary lemmas are needed:

LEMMA 3.2. $\vdash [A|B]\circ p = E \cap \pi_1;A;\breve{\pi}_1;p;\pi_2;\breve{B};\breve{\pi}_2$.

Proof. Straightforward from lemma 2.5.c. □

LEMMA 3.3. $\vdash \mu X[A;X \cup B]\circ p = \mu X[A\circ X \cup B\circ p]$.

Proof. Amounts to a straightforward application of Scott's induction rule. □

Now Wright's result (3.2.1) follows by application of lemma 3.3 from

THEOREM 3.2. (Wright) $\vdash \underbrace{\mu X[Q \cup P;X;R]}_{L} = \breve{\pi}_1;\underbrace{\mu X[(E \cap \pi_1;Q;\breve{\pi}_2) \cup [P|\breve{R}];X]\circ E}_{R};\pi_2$

Proof. $\subseteq$: Follows by the minimal fixed point property from: $\breve{\pi}_1;\ R\circ E\ ;\ \pi_2 =$
$=$ (fpp) $\breve{\pi}_1;\{(E \cap \pi_1;Q;\breve{\pi}_2) \cup [P|\breve{R}];R\}\circ E\ ;\pi_2 =$ (lemma 2.5.a)

$\breve{\pi}_1;(E \cap \pi_1;Q;\breve{\pi}_2);\pi_2 \cup \breve{\pi}_1;[P|\breve{R}]\circ(R\circ E)\ ;\pi_2 =$ (lemma 2.8) $Q \cup \breve{\pi}_1;\ [P|\breve{R}]\circ(R\circ E)\ ;\pi_2 =$
$=$ (lemma 3.2) $Q \cup \breve{\pi}_1;(E \cap \pi_1;P;\breve{\pi}_1;\ R\circ E\ ;\pi_2;R;\breve{\pi}_2);\pi_2 =$ (lemma 2.8) $Q \cup P;\breve{\pi}_1;\ R\circ E\ ;\pi_2;R$.

$\supseteq$: One derives $\breve{\pi}_1;((E \cap \pi_1;Q;\breve{\pi}_2) \cup [P|\breve{R}]\circ(E \cap \pi_1;L;\breve{\pi}_2))\ ;\pi_2 = L$ by similar techniques, whence by lemmas 2.8 and 3.2 $(E \cap \pi_1;Q;\breve{\pi}_2) \cup [P|\breve{R}]\circ(E \cap \pi_1;L;\breve{\pi}_2) \subseteq$ $\subseteq E \cap \pi_1;L;\breve{\pi}_2$, and by the minimal fixed point property $R\circ E \subseteq E \cap \pi_1;L;\breve{\pi}_2 \subseteq \pi_1;L;\breve{\pi}_2$. By lemma 2.6.c one therefore obtains $\breve{\pi}_1;\ R\circ E\ ;\pi_2 \subseteq L$. □

The reader might notice that $\breve{\pi}_1;\mu X[(\pi_1;Q;\breve{\pi}_2 \cap E) \cup [P|\breve{R}];X]\circ E\ ;\pi_2$ does not correspond with any program scheme. Using work of Luckham and Garland [12] this has been remedied in Guessarian [13] by replacing this term by an equivalent one which does correspond with a program scheme.

3.3. *Axiomatization of lists*

In general, programs manipulate data *of a special structure*, such as natural numbers, lists and trees. Consequently, proofs about the input-output relationships of these program often make use of the specific structural properties of these data. In order to axiomatize such proofs, we have to axiomatize relations over *special* domains. This is effected by adding certain axioms, *characterizing the structural properties of these data as properties of certain relation constants*, to the general system of chapter 2.

Symbolstrings have been axiomatized in De Bakker [5], finite domains in Hitchcock [14], and the natural numbers, linear lists and ordered linear lists in De Roever [9]. Lists are axiomatized below; in the following section this axiomatization is applied to derive both an informal and a formal correctness proof for the Schorr-Waite marking algorithm.

For our present purpose it is sufficient to characterize a domain of *lists* as a collection of binary trees which is closed w.r.t. the following operations:

(1) *taking a binary tree* t *apart* by applying the *car* and *cdr* functions, resulting in its constituent subtrees car(t) and cdr(t), if possible; otherwise, t is an *atom* and satisfies the predicate *at*, whence at(t) = t.

(2) *constructing a new binary tree from two old ones* by application of the function *cons*,

where car, cdr and cons are related by car $= \breve{\text{cons}};\pi_1$ and cdr $= \breve{\text{cons}};\pi_2$.
Thus we introduce one individual constant $\text{cons}^{\eta\times\eta,\eta}$ and one boolean constant $\text{at}^{\eta,\eta}$, and postulate

$$L_1 : \vdash \text{cons};\breve{\text{cons}} = E^{\eta\times\eta,\eta\times\eta}$$
$$L_2 : \vdash \breve{\text{cons}};\text{cons} \subseteq E^{\eta,\eta}$$
$$L_3 : \vdash \text{at} \cap \breve{\text{cons}};\text{cons} = \Omega^{\eta,\eta}$$
$$L_4 : \vdash E^{\eta,\eta} \subseteq \mu X[\text{at} \cup [\breve{\text{cons}};\pi_1;X,\breve{\text{cons}};\pi_2;X];\text{cons}].$$

Remark. L_1 implies that cons is total and $\breve{\text{cons}}$, whence $\breve{\text{cons}};\pi_1$ and $\breve{\text{cons}};\pi_2$ (by lemma 2.11), are functions, L_2 that cons is a function, L_3 that an atom can never by taken apart, and L_4 that any list is either an atom or can be first taken apart and then fitted together again.

LEMMA 3.4. *Let* at' *denote* $\breve{\text{cons}}$;cons. *Then lists satisfy the following properties:*
$\vdash$ E = μX[at ∪ [car;X,cdr;X];cons], at ∪ at' = E, cons;at' = cons, cons;at = Ω.

Proof. E = μX[at ∪ [car;X,cdr;X];cons]: ⊆. Axiom L_4.
⊇. Use I with Φ empty, taking {X ⊆ E} for Ψ, and (at ∪ [car;X,cdr;X];cons) for σ.
at ∪ at' = E: E = μX[at ∪ [car;X,cdr;X];cons] = (fpp) at ∪ [car,cdr];cons =
= (lemma 2.3.a axioms C_1,L_1) at ∪ $\breve{\text{cons}}$;cons = at ∪ at'.
cons;at' = cons: cons;at' = cons;$\breve{\text{cons}}$;cons = (L_1) cons.
cons;at = Ω: cons;at = cons;$\breve{\text{cons}}$∘E ;at = (L_2) cons;($\breve{\text{cons}}$;cons ∩ at) = (L_3) Ω. □

3.4. *Correctness proofs for the Schorr-Waite marking algorithm*

3.4.1. *Informal proof*

The correctness will be proved of a certain version of the Schorr-Waite marking algorithm (cf. Knuth [19], pp.417-418) for binary trees with one bitfield in each non-atomic node, the so-called *marked binary trees*.

Assume that the bitfields of a given marked binary tree have been initialized to zero. The Schorr-Waite algorithm traverses this tree in pre-order and in such a way, that, once a subtree has been traversed, the bitfields of this subtree all are set to one, whence upon termination of the algorithm all nodes are marked by ones. The interesting property of the algorithm is that it does not use an external auxiliary stack, but *codes during this process of traversal the stack of its return links into the tree itself.* This is realized by

(1) temporarily destroying the branching structure of the tree in order to store the return links, and
(2) using the bitfields both in order to distinguish which field of a node refers to a tree with a return link and for the actual process of marking.

In informal notation our version of this algorithm looks as follows:

```
SCHORR-WAITE(l) ⇐ LEFT(l,NIL),
LEFT(l,r)       ⇐ if at(l) ∨ nil(l) then BACK(l,r)
                  else LEFT(car(l),cons(cdr(l),r,1)),
BACK(l,r)       ⇐ if nil(r) then <l,NIL> else if                    (3.4.1)
                  bitfield(r)=1 then LEFT(car(r),cons(cdr(r),l,0))
                  else BACK(cons(cdr(r),l,1),car(r)),
```

with NIL denoting the empty marked binary tree, and bitfield(r) isolating the bitfield of r.

This program may be understood as follows:

(1) LEFT is called with l still to be traversed and marked,

(2) BACK is called with l traversed and marked already,

(3) if r is not NIL or no atom, bitfield(r) = 1 implies that car(r) must still be traversed,

(4) if r is not NIL or no atom, bitfield(r) = 0 implies that cdr(r) has been traversed and marked already.

Consider both NIL and any atom to be marked. Then it follows from (3.4.1) by a simple induction argument on the number of nodes of l, that the four assertions above are invariants of LEFT and BACK. Hence, provided l is unmarked and LEFT(l,NIL) terminates, LEFT(l,NIL) results in the marking of l, i.e., in this way we convince ourselves of the *partial* correctness of SCHORR-WAITE only.

An informal proof of total correctness of SCHORR-WAITE by a different argument, using Burstall's structural induction (cf. [2]) is given below. In the next section this proof will be formalized.

Let M(l) and Notmarked(l) be declared by

$$M(l) \Leftarrow \underline{if}\ at(l) \vee nil(l)\ \underline{then}\ l\ \underline{else}\ cons(M(car(l)),M(cdr(l)),1), \qquad \ldots \quad (3.4.2)$$

$$Notmarked(l) \Leftarrow \underline{if}\ at(l) \vee nil(l)\ \underline{then}\ \underline{true}\ \underline{else}\ bitfield(l)=0 \wedge \\ \wedge\ Notmarked(car(l)) \wedge Notmarked(cdr(l)). \qquad \ldots \quad (3.4.3)$$

Then total correctness of SCHORR-WAITE follows as special case from the validity of

$$(Notmarked(l) \supset LEFT(l,r) = BACK(M(l),r)) \qquad \ldots \quad (3.4.4)$$

by taking r = NIL, since BACK(M(l),NIL) = <M(l),NIL> follows from (3.4.1).

Proof of (3.4.4). (1) If at(l) ∨ nil(l) holds, (3.4.4) follows directly from (3.4.1).

(2) Let $l = cons(l_1,l_2,0)$ and let Notmarked(l) = $\underline{true}$. ... (3.4.5)

Assume by hypothesis, $(Notmarked(l_i) \supset LEFT(l_i,r) = BACK(M(l_i),r))$, i=1,2.

$LEFT(cons(l_1,l_2,0),r) = LEFT(l_1,cons(l_2,r,1))$.

From (3.4.5) and (3.4.3) we have $Notmarked(l_i) = \underline{true}$, i=1,2, hence

$LEFT(l_1,cons(l_2,r,1)) =$ (hypothesis) $BACK(M(l_1),cons(l_2,r,1)) =$

$= LEFT(l_2,cons(r,M(l_1),0)) =$ (hypothesis) $BACK(M(l_2),cons(r,M(l_1),0)) =$

$= BACK(cons(M(l_1),M(l_2),1),r) \overset{!}{=} BACK(M(cons(l_1,l_2,0)),r)$. □

3.4.2. *Formal proof*

Prior to formalizing the informal correctness proof of SCHORR-WAITE of the previous section, the axiomatization of lists (or binary trees) of section 3.3 must be extended in order to incorporate (1) the presence of a bitfield in each non-atomic node, and (2) the empty list.

First we formalize 2-elements sets by introducing two boolean constants $\underline{0}$ and $\underline{1}$, and postulating

$$TWO_1 : \vdash \underline{0};U \cap U;\underline{0} \subseteq \underline{0},\ \ \underline{1};U \cap U;\underline{1} \subseteq \underline{1}$$

TWO_2 : $\vdash$ $U \subseteq U;\underline{0};U$, $U \subseteq U;\underline{1};U$

TWO_3 : $\vdash$ $\underline{0} \cup \underline{1} = E$, $\underline{0} \cap \underline{1} = \Omega$.

TWO_1 confines any interpretation of both $\underline{0}$ and $\underline{1}$ to *at most* one pair of (identical) elements, TWO_2 expresses that any interpretation of both $\underline{0}$ and $\underline{1}$ must contain *at least* one pair of elements, and TWO_3 speaks for itself.
Satisfaction of these axioms establishes $<D_\eta,\underline{0}^{\eta,\eta},\underline{1}^{\eta,\eta}>$ as a structure for a 2-element set. This leads us to introduce $\underline{2}$ as type reserved for 2-element sets.

Next marked binary trees with an empty element are axiomatized by introducing $\text{cons}^{\eta\times\eta\times\underline{2},\eta}$ as relation constant, and $\text{nil}^{\eta,\eta}$ and $\text{at}^{\eta,\eta}$ as boolean constants, and postulating

ML_1 : $\vdash$ $\text{cons};\breve{\text{cons}} = E$

ML_2 : $\vdash$ $\breve{\text{cons}};\text{cons} \subseteq E$

ML_3 : $\vdash$ $\breve{\text{cons}};\text{cons} \cap (\text{at} \cup \text{nil}) = \Omega$

ML_4 : $\vdash$ $\text{at} \cap \text{nil} = \Omega$

ML_5 : $\vdash$ $E \subseteq \mu X[\tau_{ML}(X)]$,

where $\tau_{ML}(X)$ is defined by $\tau_{ML}(X) = (\text{at} \cup \text{nil} \cup [\text{car};X,\text{cdr};X,\text{bitfield}];\text{cons})$, and car, cdr and bitfield are defined by $\text{car} = \breve{\text{cons}};\pi_1$, $\text{cdr} = \breve{\text{cons}};\pi_2$, $\text{bitfield} = \breve{\text{cons}};\pi_3$.
Satisfaction of these axioms establishes $<D_\eta,\text{cons},\text{at},\text{nil}>$ as a structure of marked binary trees with an empty element, of type $\underline{ML}$.

LEMMA 3.5. *Let* $a' = \breve{\text{cons}};\text{cons}$. *Then marked binary trees satisfy*
$\vdash$ $E = \mu X[\tau_{ML}(X)]$, $\text{at} \cup \text{nil} \cup \text{at}' = E$, $\text{cons};\text{at}' = \text{cons}$, $\text{cons};\text{at} = \Omega$, $\text{cons};\text{nil} = \Omega$.

Proof. Similar to the proof of lemma 3.4. □

Finally, we give a formal definition of LEFT, BACK, M and Notmarked, which were informally declared in the previous section.
Let $\tau_{LEFT}(X,Y) = (\pi_1\circ(\text{at} \cup \text{nil})\ ;Y \cup [\pi_1;\text{car},[\pi_1;\text{cdr},\pi_2,U;\underline{1}];\text{cons}];X)$, and
$\tau_{BACK}(X,Y) = (\pi_2\circ\text{nil} \cup (\pi_2;\text{bitfield})\circ\underline{1}\ ;[\pi_2;\text{car},[\pi_2;\text{cdr},\pi_1,U;\underline{0}];\text{cons}];X \cup$
$\cup\ (\pi_2;\text{bitfield})\circ\underline{0}\ ;\ [[\pi_2;\text{cdr},\pi_1,U;\underline{1}];\text{cons},\pi_2;\text{car}];Y$, where U is of type $(\underline{ML}\times\underline{ML},\underline{2})$.
Then LEFT, BACK, M and N(otmarked) are defined by

$$\text{LEFT} = \mu_1 XY[\tau_{LEFT}(X,Y),\tau_{BACK}(X,Y)],$$
$$\text{BACK} = \mu_2 XY[\tau_{LEFT}(X,Y),\tau_{BACK}(X,Y)],$$
$$M = \mu X[\text{at} \cup \text{nil} \cup [\text{car};X,\text{cdr};X,U^{\underline{ML},\underline{2}};\underline{1}];\text{cons}], \text{ and}$$
$$N = \mu X[\text{at} \cup \text{nil} \cup \text{car}\circ X\ ;\text{cdr}\circ X\ ;\text{bitfield}\circ\underline{0}]. \quad \square$$

THEOREM 3.3. $\vdash$ $\pi_1\circ N\ ;\text{LEFT} = \pi_1\circ N\ ;[\pi_1;M,\pi_2];\text{BACK}$.

Proof. By lemma 3.5, $E^{\underline{ML},\underline{ML}} = \mu X[\tau_{ML}(X)]$. Hence we prove

$$\vdash [\pi_1;\mu X[\tau_{ML}(X)],\pi_2];\pi_1\circ N\ ;\text{LEFT} = [\pi_1;\mu X[\tau_{ML}(X)],\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];\text{BACK}$$

using Scott's induction rule. It is sufficient to prove the induction step:

$[\pi_1;X,\pi_2];\pi_1\circ N\ ;LEFT = [\pi_1;X,\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];BACK \vdash$
$\quad [\pi_1;\tau_{ML}(X),\pi_2];\pi_1\circ N\ ;LEFT = [\pi_1;\tau_{ML}(X),\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];BACK.$

Part a

$[\pi_1;(at \cup nil),\pi_2];\pi_1\circ N\ ;LEFT =$ (lemma 2.4) $\pi_1\circ N\ ;[\pi_1;(at \cup nil),\pi_2];LEFT =$ (fpp) $\pi_1\circ N\ ;[\pi_1;(at \cup nil),\pi_2];BACK =$ (fpp and C_2) $\pi_1\circ N\ ;[\pi_1;(at \cup nil),\pi_2];[\pi_1;M,\pi_2];BACK \overset{!}{\doteq}$
$\doteq$ (lemma 2.4) $[\pi_1;(at \cup nil),\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];BACK.$

Part b. Assume the hypothesis.

$[\pi_1;[car;X,cdr;X,bitfield];cons,\pi_2];\pi_1\circ N\ ;LEFT =$ (lemmas 2.3.a and 2.6.c)
$[[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons,\pi_2];\pi_1\circ N\ ;LEFT =$ (lemma 2.5.e, since functionality of $[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons$ follows in standard fashion from lemma 2.11, by adding $\breve{X};X \subseteq E$ to the hypotheses, and proving $\breve{\tau}_{ML};\tau_{ML} \subseteq E$ using lemma 2.11 again)

$\underbrace{[[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons,\pi_2]\circ(\pi_1\circ N)}_{\mathcal{E}}\ ;$
$\qquad [[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons,\pi_2];LEFT =$ (fpp)

$\mathcal{E};[\pi_1;car;X,[\pi_1;cdr;X,\pi_2,U;\underline{1}];cons];LEFT =$ (part c below)
$\mathcal{E};(\pi_1;car;X)\circ N\ ;[\pi_1;car;X,[\pi_1;cdr;X,\pi_2,U;\underline{1}];cons];LEFT =$ (lemmas 2.5.e and 2.7)
$\mathcal{E};[\pi_1;car;X,[\pi_1;cdr;X,\pi_2,U;\underline{1}];cons];\pi_1\circ N\ ;LEFT =$ (C_2)
$\mathcal{E};[\pi_1;car,[\pi_1;cdr;X,\pi_2,U;\underline{1}];cons];[\pi_1;X,\pi_2];\pi_1\circ N\ ;LEFT =$ (hypothesis)
$\mathcal{E};[\pi_1;car,[\pi_1;cdr;X,\pi_2,U;\underline{1}];cons];[\pi_1;X,\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];BACK =$ (similar to above)
$\mathcal{E};[\pi_1;car;X;M,[\pi_1;cdr;X,\pi_2,U;\underline{1}];cons];BACK =$ (fpp)
$\mathcal{E};[\pi_1;cdr;X,[\pi_2,\pi_1;car;X;M,U;\underline{0}];cons];LEFT =$ (part c below)
$\mathcal{E};(\pi_1;cdr;X)\circ N\ ;[\pi_1;cdr;X,[\pi_2,\pi_1;car;X;M,U;\underline{0}];cons];LEFT =$ (lemmas 2.5.e and 2.7, and C_2)
$\mathcal{E};[\pi_1;cdr\ ,[\pi_2,\pi_1;car;X;M,U;\underline{0}];cons];[\pi_1;X,\pi_2];\pi_1\circ N\ ;LEFT =$ (hypothesis)
$\mathcal{E};[\pi_1;cdr\ ,[\pi_2,\pi_1;car;X;M,U;\underline{0}];cons];[\pi_1;X,\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];BACK =$ (similar to above)
$\mathcal{E};[\pi_1;cdr;X;M,[\pi_2,\pi_1;car;X;M,U;\underline{0}];cons];BACK =$ (fpp)
$\mathcal{E};[[\pi_1;car;X;M,\pi_1;cdr;X;M,U;\underline{1}];cons,\pi_2];BACK =$ (lemmas 2.3.c, 2.6.c, and ML_1, ML_2)
$\mathcal{E};[[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons;[car;M,cdr;M,U^{\underline{ML},2};\underline{1}];cons,\pi_2];BACK =$ (fpp)
$\mathcal{E};[[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons,\pi_2];[\pi_1;M,\pi_2];BACK \overset{!}{\doteq}$ (lemmas 2.5.e and 2.11, cf. above, and lemma 2.3.a)
$[\pi_1;[car;X,cdr;X,bitfield];cons,\pi_2];\pi_1\circ N\ ;[\pi_1;M,\pi_2];BACK.$

Part c

We prove $\mathcal{E} = \mathcal{E};(\pi_1;car;X)\circ N\ ;(\pi_1;cdr;X)\circ N\ ;(\pi_1;bitfield)\circ\underline{0}$, with $\mathcal{E}$ as defined above.
$[[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons,\pi_2]\circ(\pi_1\circ N) =$ (lemma 2.5.a, since $N\circ E = N$ follows from lemma 2.4 and P_1)
$([[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons,\pi_2];\pi_1;N)\circ E =$ (lemmas 2.7 and 2.6.a)
$([\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons;N)\circ E =$ (fpp)
$([\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons;car\circ N\ ;cdr\circ N\ ;bitfield\circ\underline{0})\circ E =$ (lemma 2.5.e and ML_1)

$([\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];\pi_1\circ N\ ;\pi_2\circ N\ ;\pi_3\circ \underline{0}\ ;cons)\circ E =$ (lemma 2.5.e, cf. part b)
$((\pi_1;car;X)\circ N\ ;(\pi_1;cdr;X)\circ N\ ;(\pi_1;bitfield)\circ \underline{0}\ ;$
$[\pi_1;car;X,\pi_1;cdr;X,\pi_1;bitfield];cons)\circ E =$ (corollary 2.2)
$E;(\pi_1;car;X)\circ N\ ;(\pi_1;cdr;X)\circ N\ ;(\pi_1;bitfield)\circ \underline{0}$. □

4. A CALCULUS FOR RECURSIVE PROCEDURES WITH VARIOUS PARAMETER MECHANISMS

4.1. *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 section 2.2.3, 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 4.2.2 contains an axiomatization of *all* these products. By replacing in the axiom system of chapter 2 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 4.2.2, 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 [27,28] 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 $D_1,\ldots,D_n$ with basepoints $\underline{pt}_1,\ldots,\underline{pt}_n$, which are contained in $D_1,\ldots,D_n$, 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.* ... (4.1.1)

(2) *Only a non-basepoint can be transformed into a non-basepoint.* ... (4.1.2)

EXAMPLE 4.1. Let $D_1,\dots,D_n$ be domains with basepoints, $\underline{pt}_1,\dots,\underline{pt}_n$, respectively, then the projection function $\pi_i\colon D_1\times\dots\times D_n \to D_i$ is defined as follows:

$$\pi_i(<x_1,\dots,x_n>) = \begin{cases} x_i, \text{ provided } x_i \neq \underline{pt}_i, \\ \underline{pt}_i, \text{ in case } x_j = \underline{pt}_j,\ j=1,\dots,n, \\ \text{undefined, otherwise,} \end{cases} \qquad \dots \quad (4.1.3)$$

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 (4.1.1) and (4.1.2). Then $[R_1\times\dots\times R_n]$ is interpreted as follows:

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

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 4.1, that $[R_1\times\dots\times R_n];\pi_i = R_i$, $i=1,\dots,n$. Notice also that $[R_1\times\dots\times R_n];(\pi_{j_1};\breve{\pi}_1 \cap\dots\cap \pi_{j_k};\breve{\pi}_k) = (R_{j_1};\breve{\pi}_1 \cap \dots \cap R_{j_k};\breve{\pi}_k)$, for $1\le j_1<\dots<j_k\le n$, 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 $D,D_1,\dots,D_n$ be as above, then the relation constants $\star_1,\dots,\star_n$ are defined by

$$<<x_1,\dots,x_n>,x> \in \star_i \text{ iff } \begin{cases} x = \underline{pt}, \text{ in case } x_j = \underline{pt}_j,\ j=1,\dots,n, \\ x \in D\text{-pt}, \text{ provided } x_i = \underline{pt}_i, \text{ and} \\ \quad x_j \neq pt_j \text{ for at least one } j,\ j\neq i, \\ \text{undefined, otherwise,} \end{cases} \qquad \dots \quad (4.1.4)$$

for $i=1,\dots,n$. □

The introduction of these constants is motivated by the following property: $\check{*}_i$ 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

$$[R_1 \times \ldots \times R_n] = \bigcup_{\substack{I \subseteq \{1,\ldots,n\},\\ \text{and } I \neq \emptyset}} \{(\bigcap_{i\in I} R_i;\breve{\pi}_i) \cap (\bigcap_{i\in\{1,\ldots,n\}-I} \check{*}_i)\}.$$

For example, $[R_1 \times R_2] = (R_1;\breve{\pi}_1 \cap R_2;\breve{\pi}_2) \cup (\check{*}_1 \cap R_2;\breve{\pi}_2) \cup (R_1;\breve{\pi}_1 \cap \check{*}_2)$.

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 $(f_1(\xi),\ldots,f_n(\xi))$ from the call $f(f_1(\xi),\ldots,f_n(\xi))$ results in an expression of the form $[f_1(\xi) \times \ldots \times f_n(\xi)]^{\underline{\text{value}}\{1,\ldots,k\}};P$, where the value of $[f_1(\xi) \times \ldots \times f_n(\xi)]^{\underline{\text{value}}\{1,\ldots,k\}}$ is only defined in case the evaluation of the first k parameters, the call-by-value parameters $f_1(\xi),\ldots,f_k(\xi)$, terminates. Therefore a relational description of this parameter list is obtained by introducing a product of relations $[R_1 \times \ldots \times R_n]^{\underline{\text{value}}\{1,\ldots,k\}}$, which satisfies

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

for i=1,...,n.

In general, such products are interpreted as follows:

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

for $J \subseteq \{1,\ldots,n\}$ s.t. $J \neq \emptyset$:

$$[R_1 \times \ldots \times R_n]^{\underline{\text{value}}\ J} = \bigcup_{K \subseteq I}\{(\bigcap_{j\in J} R_j;\breve{\pi}_j) \cap (\bigcap_{k\in K} R_k;\breve{\pi}_k) \cap (\bigcap_{k\in I-K} \check{*}_k)\},$$

for $J = \emptyset$: $\qquad \ldots \quad (4.1.5)$

$$[R_1 \times \ldots \times R_n]^{\underline{\text{value}}\ \emptyset} = \bigcup_{K\subseteq I, K\neq\emptyset}\{(\bigcap_{k\in K} R_k;\breve{\pi}_k) \cap (\bigcap_{k\in I-K} \check{*}_k)\}. \quad \square$$

For example, $[R_1 \times R_2 \times R_3]^{\underline{\text{value}}\{1\}} = (R_1;\breve{\pi}_1 \cap \check{*}_2 \cap \check{*}_3) \cup (R_1;\breve{\pi}_1 \cap R_2;\breve{\pi}_2 \cap \check{*}_3) \cup$ $\cup (R_1;\breve{\pi}_1 \cap \check{*}_2 \cap R_3;\breve{\pi}_3) \cup (R_1;\breve{\pi}_1 \cap R_2;\breve{\pi}_2 \cap R_3;\breve{\pi}_3)$. Hence, $[R_1 \times R_2 \times R_3]^{\text{value}\{1\}};\pi_i = R_1 \circ E\,;R_i$, i=1,2,3.

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 $J = \{1,\ldots,n\}$ and $J = \emptyset$, respectively.

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

4.2.1. *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 section 2.1.

The syntax of MU^* is obtained from the syntax of MU by adding for n≥2 the logical relation constants $*_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$, for i=1,...,n, and all $\eta_1,\ldots,\eta_n$, to the elementary terms of MU.

The semantics of MU^* is determined by considering binary relations over domains with unique basepoints only, observing restrictions (4.1.1) and (4.1.2), and interpreting $\pi_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$ and $*_i^{\eta_1\times\ldots\times\eta_n,\eta_i}$ as in (4.1.3) and (4.1.4), for i=1,...,n, and all $\eta_1,\ldots,\eta_n$. Hence,

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

(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 $\langle p^{\eta,\eta},p'^{\eta,\eta}\rangle$ 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 $m(\Omega^{\eta,\theta}) \subseteq m(X^{\eta,\theta}) \subseteq m(U^{\eta,\theta})$,

(5) the operators "∪", "∩", ";", "⌣" 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) $\mu_i X_1\ldots X_n[\sigma_1,\ldots,\sigma_n]$ is interpreted as the i-th component of the (unique) minimal fixed point of the transformation $\langle m(\sigma_1),\ldots,m(\sigma_n)\rangle$ acting on n-tuples of relations satisfying (4.1.1) and (4.1.2), i=1,...,n. Observe that it follows from the definitions that any fixed point of $\langle m(\sigma_1),\ldots,m(\sigma_n)\rangle$ acting on these relations satisfies (4.1.1) and 4.1.2); hence the minimal fixed point of this transformation, being the intersection of all these fixed points, satisfies (4.1.1) and (4.1.2) also.

4.2.2. *Axiomatization*

MU^* is axiomatized by replacing in the axiom system for MU, as contained in chapter 2, axioms C_1 and C_2 by BP_1, BP_2, BP_3, BP_4 and BP_5 below: For n≥2,

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

BP_2 : $\vdash *_i = *_i;U^{\eta_i,\eta_i}$, i=1,...,n,

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

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

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

$$\vdash \bigcap_{i\in I} X_i;Y_i = \{(\bigcap_{i\in I} X_i;\breve{\pi}_i) \cap (\bigcap_{i\in\{1,\ldots,n\}-I} \breve{*}_i)\};\{(\bigcap_{i\in I} \pi_i;Y_i) \cap (\bigcap_{i\in\{1,\ldots,n\}-I} *_i)\},$$

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

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

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

LEMMA 4.1. *Let* $n\geq 2$, $i=1,\ldots,n$, *and* $j=1,\ldots,n$, *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* $\vdash \breve{\pi}_i;\pi_i = E$.

Proof. We prove parts a and b only.

a. $\breve{*}_i;\pi_j = U$, $i\neq j$: The case $n=2$, $i=1$, $j=2$ is representative.
$*_1 = *_1 \cap (\pi_2 \cup *_2);U$ and $\pi_2 = (\pi_1 \cup *_1);U \cap \pi_2$ follow by BP_4 from lemma 2.3.c. Hence, $\breve{*}_1;\pi_2 = (\breve{*}_1 \cap U;(\breve{\pi}_2 \cup \breve{*}_2));((\pi_1 \cup *_1);U \cap \pi_2) \supseteq$ (lemma 2.1.f, BP_2) $(\breve{*}_1 \cap U;\breve{\pi}_2);(*_1 \cap \pi_2;E) =$ (BP_5) U. $\breve{*}_i;\pi_i = \Omega$: $\breve{*}_i;\pi_i = \breve{*}_i;*_i\circ E\ ;\pi_2\circ E\ ;\pi_2 = \Omega$, since $*_i\circ E\ ;\pi_i\circ E \subseteq *_i;\breve{*}_i \cap \pi_i;\breve{\pi}_i =$ (BP_3) Ω.

b. $\breve{*}_i;*_j = \Omega$, $i\neq j$, $n=2$: $\breve{*}_1;*_2 = (\breve{*}_1 \cap U;(\breve{\pi}_2 \cup \breve{*}_2));((\pi_1 \cup *_1);U \cap *_2) =$ (BP_2) $((\breve{*}_1 \cap U;\breve{\pi}_2) \cup (\breve{*}_1 \cap \breve{*}_2));((\pi_1;U \cap *_2) \cup (*_1 \cap *_2)) = (\breve{*}_1 \cap U;\breve{\pi}_2);(\pi_1;U \cap *_2)$, since $*_1 \cap *_2 = \Omega$ follows from BP_1; moreover, $(\breve{*}_1 \cap U;\breve{\pi}_2);(\pi_1;U \cap *_2) \subseteq \breve{*}_1;\pi_1;U =$ (part a) Ω. $\breve{*}_i;*_i = U$, for $n=2$, and $\breve{*}_i;*_j = U$, $i\neq j$, for $n\geq 3$: proved using similar techniques. □

Let $[X_1\times\ldots\times X_n]^{\underline{\text{value}}\ J}$ be defined as in (4.1.5). Then the proofs of corollaries 4.1 and 4.2 follow from lemma 4.1 and the definitions.

COROLLARY 4.1. $\vdash [X_1\times\ldots\times X_n]^{\underline{\text{value}}\{j_1,\ldots,j_k\}};\pi_i = X_{j_1}\circ E\ ;\ldots;X_{j_k}\circ E\ ;X_i$, $i=1,\ldots,n$.

COROLLARY 4.2. $\vdash [X_1\times\ldots\times X_n]^{\underline{\text{value}}\{j_1,\ldots,j_m\}};(\pi_{k_1};\breve{\pi}_1 \cap \ldots \cap \pi_{k_p};\breve{\pi}_p) =$
$= X_{j_1}\circ E\ ;\ldots;X_{j_m}\circ E\ ;(X_{k_1};\breve{\pi}_1 \cap \ldots \cap X_{k_p};\breve{\pi}_p)$.

5. CONCLUSION AND RELATED WORK

5.1. *Conclusion*

This investigation shows that

1. The relational approach allows a unified axiomatization of both call-by-value and certain aspects of call-by-name (chapter 1 and 4).

2. A theory of correctness of programs requires an operator describing the interaction between programs and predicates; in the present theory this is the "∘"

operator (theory: section 2.2.2, applications: sections 3.2 and 3.4).

3. The "∘" operator is crucial to an expedient formalization of the call-by-value parameter mechanism (theory: section 2.2.3, application: section 3.4).

4. The axiomatization of correctness proofs for recursive programs can be applied to recursive data structures (sections 3.3 and 3.4, the main reference being Hitchcock and Park [15]).

5. Informal use of structural induction may lead to understandable and conceptually attractive correctness proofs (section 3.4.1, the main reference being Burstall [2]; cf. also section 6.3.a of De Roever [9] which contains an informal correctness proof for the recursive solution of the *Towers of Hanoi* problem).

Notably, we have not discussed the topic of providing any operationally, interpreter-defined, semantics for the various programming concepts whose mathematical semantics were axiomatized. Here the main issue is that one must actually *prove* that the interpreter-defined input-output behaviour of the programs of one's particular programming language coincides with the mathematically defined semantics of the corresponding (relational) terms.
An interpreter for a simple recursive programming language with call-by-value as parameter mechanism has been defined in De Roever [8 , 9]. The input-output behaviour of the programs of this language has been proved to coincide with the mathematical semantics of the corresponding relational terms in De Roever [9].
Using the techniques of introducing parameters called-by-name by procedures which have these parameters as their bodies (suggested in this context by J.W. de Bakker), and of describing an invokation of such a parameter by a call of the corresponding procedure, we defined an interpreter for a recursive programming language with both call-by-value and call-by-name as parameter mechanism, with the use of the latter being restricted as in section 1.2. A proof that the input-output behaviour again coincides with the mathematical semantics is presently being investigated.

5.2. *Related work*

This discussion of related work confines itself mainly to the *relational* approach to correctness of recursive programs. Dominant in this approach is the minimal fixed point characterization, which is initiated by Scott and De Bakker in [29], elaborated by De Bakker in [4], and crossbred with Tarski's algebra of relations [30] in De Bakker and De Roever [6] to yield an axiomatic framework for proving equivalence, correctness and termination of first-order recursive programs with *one* variable. The present paper amplifies on the latter in that that the restriction to one variable is removed by considering arbitrary subdivisions of the state; these are incorporated within the relational framework by considering binary relations over cartesian products of domains, introduced in unpublished work of Milner [23] and Park [26]. In De Roever [9] we (1) clarify the distinction on the one hand and the connection on the other between operational and mathematical semantics, (2) axiomatize the natural numbers, lists, linear lists and ordered linear lists within the relational framework, and (3) give numerous axiomatic correctness proofs for programs which manipulate values from these domains, with special emphasis on axiomatic list manipulation and correctness of the recursive solution of the Towers of Hanoi problem.

The connection between *induction rules* and *termination proofs* is described in Hitchcock and Park [15] and elaborated in Hitchcock's dissertation [14], which also contains a correctness proof of a translator of arbitrary recursive programs into regular recursive procedures with stacks, and an axiomatization of finite domains.

Maximal fixed points, introduced by Park in [25], are applied in Mazurkiewicz [21] to obtain a mathematical characterization of *divergent* computations, and may lead to the axiomatization of Hitchcock and Park's results within an extension of our framework.

In a different setting Blikle and Mazurkiewicz [1] also use an algebra of relations to investigate programs.

The *completeness* of the method of *inductive assertions* for general recursive procedures is proved in De Bakker and Meertens [7].

The relation between the minimal fixed point characterization and various rules of computation is studied by Manna, Cadiou, Vuillemin and their colleagues in, e.g., Manna and Vuillemin [20], Cadiou [3] and Vuillemin [31].

The works of Dijkstra [10,11], Hoare [16] and Wirth [32] relate to the present paper in that we provide a possible axiomatic basis for some techniques of structured programming; e.g., our correctness operator "∘" is independently described in Dijkstra [11].

ACKNOWLEDGEMENTS

The original incentive which lead to this work arose out of the lectures of E.W. Dijkstra, C.A.R. Hoare, and N. Wirth at the International Summer School on Program Structures and Fundamental Concepts of Programming, organized by F.L. Bauer, H.J. Helms and M. Paul in 1971.

I am deeply indebted to my thesis adviser J.W. de Bakker for his guidance during my studies and his scrutiny of my writings, which lead to numerous improvements.

I am grateful to M. Nivat for his remarks about the first version of this paper, for his invitation to lecture at the University of Paris and at IRIA, and for publishing its second version in the Séminaires IRIA series.

This particular paper would not have been written were it not for the constant support of Peter van Emde Boas, and his interest in my work.

BIBLIOGRAPHY

[1] Blikle, A., and A. Mazurkiewicz, *An algebraic approach to the theory of programs,*

algorithms, languages and recursiveness, in Proc. of an International Symposium and Summer School on the Mathematical Foundations of Computer Science, Warsaw-Jablonna, 1972.

[2] Burstall, R.M., *Proving properties of programs by structural induction*, Comput. J., 12 (1969), 41-48.

[3] Cadiou, J.M., *Recursive definitions of partial functions and their computations*, Thesis, Stanford University, 1972.

[4] De Bakker, J.W., *Recursive procedures*, Mathematical Centre Tracts 24, Amsterdam, 1971.

[5] De Bakker, J.W., *Recursion, induction and symbol manipulation*, in Proc. MC-25 Informatica Symposium, Mathematical Centre Tracts 37, Amsterdam, 1971.

[6] De Bakker, J.W., and W.P. de Roever, *A calculus for recursive program schemes*, in Proc. IRIA Symposium on Automata, Formal languages and Programming, M. Nivat (ed.), North-Holland, Amsterdam, 1972.

[7] De Bakker, J.W., and L.G.L.Th. Meertens, *On the completeness of the inductive assertion method*, Prepublication, Mathematical Centre Report IW 12/73, Amsterdam, 1973.

[8] De Roever, W.P., *Operational and mathematical semantics for recursive polyadic program schemata (Extended abstract)*, in Proceedings of Symposium and Summer School "Mathematical Foundations of Computer Science", 3-8 September 1973, High Tatras, Czechoslovakia, pp.293-298.

[9] De Roever, W.P., *Operational, mathematical and axiomatized semantics for recursive procedures and data structures*, Mathematical Centre Report ID/1, Amsterdam.

[10] Dijkstra, E.W., *Notes on structured programming*, in Hoare, C.A.R., Dijkstra, E.W., and O.J. Dahl, *Structured Programming*, Academic Press, New York, 1972.

[11] Dijkstra, E.W., *A simple axiomatic basis for programming language constructs*, Indagationes Mathematicae, 36 (1974) 1-15.

[12] Garland, S.J., and D.C. Luckham, *Translating recursion schemes into program schemes*, in Proc. of an ACM Conference on Proving Assertions about Programs, Las Cruces, New Mexico, January 6-7, 1972.

[13] Guessarian, I., *Sur une réduction des schémas de programmes polyadiques à des schémas monadiques et ses applications*, Memo GRIT no. 73. 05, Université de Paris, 1973.

[14] Hitchcock, P., *An approach to formal reasoning about programs*, Thesis, University of Warwick, Coventry, England, 1974.

[15] Hitchcock, P., and D. Park, *Induction rules and proofs of termination*, in Proc. IRIA Symposium on Automata, Formal Languages and Programming, M. Nivat (ed.), North-Holland, Amsterdam, 1972.

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

[17] Hotz, G., *Eindeutigkeit und Mehrdeutigkeit formaler Sprachen*, Electron. Informationsverarbeit. Kybernetik, 2 (1966), 235-246.

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

[19] Knuth, D.E., *The Art of Computer Programming*, Vol. 1, *Fundamental Algorithms*, Addison Wesley, Reading (Mass.), 1968.

[20] Manna, Z., and J. Vuillemin, *Fixpoint approach to the theory of computation*, Comm. ACM, 15 (1972) 528-536.

[21] Mazurkiewicz, A., *Proving properties of processes*, PRACE CO PAN-CC, PAS Reports 134, Warsaw, 1973.

[22] McCarthy, J., *A basis for a mathematical theory of computation*, in *Computer Programming and Formal Systems*, pp.33-70, P. Braffort and D. Hirschberg (eds.), North-Holland, Amsterdam, 1963.

[23] Milner, R., *Algebraic theory of computable polyadic functions*, Computer Science Memorandum 12, University College of Swansea, 1970.

[24] Morris Jr., J.H., *Another recursion induction principle*, Comm. ACM, 14 (1971) 351-354.

[25] Park, D., *Fixpoint induction and proof of program semantics*, in *Machine Intelligence*, Vol. 5, pp.59-78, B. Meltzer and D. Michie (eds.), Edinburgh University Press, Edinburgh, 1970.

[26] Park, D., *Notes on a formalism for reasoning about schemes*, Unpublished notes, University of Warwick, 1970.

[27] Scott, D., *Outline of a mathematical theory of computation*, in Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems, pp.169-176, Princeton. 1970.

[28] Scott, D., *Lattice theory, data types, and semantics*, in NYU Symposium on formal semantics, pp.64-106, Prentice Hall, 1972.

[29] Scott, D., and J.W. de Bakker, *A theory of programs*, Unpublished notes, IBM Seminar, Vienna, 1969.

[30] Tarski, A., *On the calculus of relations*, J. Symbolic Logic, 6 (1941) 73-89.

[31] Vuillemin, J., *Proof techniques for recursive programs*, Thesis, Stanford University, 1972.

[32] Wirth, N., *Program development by stepwise refinement*, Comm. ACM, 14 (1971) 221-227.

[33] Wright, J.B., *Characterization of recursively enumerable sets*, J. Symbolic Logic, 37 (1972) 507-511.