stichting
mathematisch
centrum

MC

DEPARTMENT OF PURE MATHEMATICS ZW 78/76 JUNE

P. VAN EMDE BOAS

LEAST FIXED POINTS AND THE RECURSION THEOREM

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): 02F43, 68A10

Least fixed points and the recursion theorem

by

P. van Emde Boas

ABSTRACT

A constructive proof of the recursion theorem may be considered to yields a "recursion theorem" semantics to recursive procedures within the framework of abstract recursion theorem. An example indicates that this semantics without a furthergoing operational semantics can behave counter intuitively. As such it is no solution for the problems of denotational semantics.

KEY WORDS & PHRASES: *Recursive procedure, denotational semantics, recursion theorem, least fixed point.*

INTRODUCTION

The main result of contemporary denotational semantics has been the characterization of the behaviour of recursive procedures as the least fixed points of corresponding monotone and continuous transformations on some mathematically defined domain. In general these transformations depend essentially on the operational semantics formalized within the model; this fact lies at the root of the current day confusion concerning "correct computation rules" etc.

We consider this problem from the point of view of the abstract recursion theoretician. On the one hand a constructive proof of the recursion theorem yields a "recursion-theorem semantics" of a recursive procedure which conforms to the intuitive semantics for natural measures. On the other hand, by an easy counter example we show our formalism to allow counter-intuitive situations, where the identity function becomes the function represented by the procedure p suggested by

```
proc p = (int x) int: p(x).
```

§1. The problem

Consider the following two procedures p1 and p2:

```
proc p1 = (int x) int: if x = 0 then 1 else p(x-1)*x fi
proc p2 = (int x,y) int: if x = 0 then 1 else p(x-1,p(x,y))*x fi.
```

According to the ALGOL 68 semantics p1 represents the factorial function on the nonnegative integers (disregarding the finite integer capacity) and p2 represents the function which is also represented by p3 described by

```
proc p3 = (int x,y) int: if x = 0 then 1 else do skip od; 666 fi.
```

The fact that the non-termination of the computation of the dummy parameter y in p2 leads to divergent computations, although the value of y is

not used essentially to compute p2(x,y) is due to the fact that ALGOL 68 uses the call by value parameter mechanism. If one uses call by name instead one should have p2(x,y) = p1(x) instead.

The distinction between these parameter mechanisms and the resulting semantics of recursive procedures has been the object of intensive research and has caused a great deal of confusion; cf. e.g. MANNA & VUILLEMIN [5] MANNA [6], MANNA & SHAMIR [7], or DOWNEY & SETHI [3]. For clarifications see e.g. DE BAKKER [1] or DE ROEVER [10,11].

The confusion arises as follows. In the theory of semantics of recursive procedure is taken to be terminating provided it leads after a finite number of substitutions of the procedure body for the procedure name to a piece of program which can be executed without any inner calls of the procedure considered. This yields a characterization of the input-output behaviour of the procedure by an increasing union.

At the same time the "meaning" of the individual concepts of the programming language is expressed within some mathematical model, which "meaning" is extended to the composite constructs by homomorfisms selected according to the intuitive meaning of the program constructs (e.g. sequential composition in the language is mapped onto functional or relational composition in the model).

For the recursive procedure two possible interpretations are available: The infinite union due to the operational semantics can be translated directly into the model; on the other hand the transformation: procedure name $\rightarrow$ procedure body may be interpreted by a corresponding transformation on the model. If the semantics is well designed this transformation is monotone and continuous. Consequently this transformation possesses a least fixed point, which is taken to be the semantics of the procedure. Since the same fixed point can be approximated, using the Krasner-Kuratowski lemma by the infinite union mentioned before the two approaches yield the same result.

As stated above the description is incomplete as far that it does not explain in details the operationally determined body-replacement rules used to elaborate the procedure. Since these rules again determine the transformation in the model, and since the least fixed point again depends on

this transformation we observe that:

1) The least fixed point characterization is based upon the operational semantics modelled in the model
2) different operational inter pretations lead to different transformations and therefore to different least fixed points.

The confusion on "correct computation rules" etc. results if a certain transformation (i.e. a certain operational interpretation) is taken as absolute standard and if all other transformations are compared to the absolute one.

In this paper we investigate whether abstract recursion theory yields a justification for an absolute standard interpretation for recursive procedures. Although we succeed in defining a "recursion-theorem semantics" for recursive procedures we show that it allows counter intuitive results. Consequently abstract recursion theory gives no contribution to the solution of the confusion mentioned above.

§2. The framework of abstract recursion theory

In this paper we use the concept of an Effective Enumeration of partial recursive functions as introduced by H. ROGERS [9]. Although the more extended framework of a BLUM complexity measure [2] is nowadays used for practising abstract recursion theory we do not use it since for the problem considered the run-times are irrelevant.

In the sequel all functions are partial functions from $\mathbb{N}$ (set of non-negative integers) into itself. The domain and range of a function f are denoted by $\mathcal{D}f$ and $\mathcal{R}f$ respectively. By $\langle x,y\rangle$ we denote a fixed recursive pairing function with coordinate projections π_1 and π_2, i.e. $\langle \pi_1 x.\pi_2 x\rangle = x$.

DEFINITION 1. An effective enumeration $(\phi_i)_i$ is a list of recursive functions, called programs, with the following properties:

0) Each function in the list is partial recursive and all partial recursive functions occur in the list
1) [Universal machine]: there exists an index u such that $\phi_u(\langle i,x\rangle) = \phi_i(x)$ for all i and x

2) [s-n-m axiom]: there exists a total recursive function s such that for all i, x and y one has $\phi_{s(i,x)}(y) = \phi_i(\langle x,y\rangle)$.

This definition requires knowledge which functions are recursive. The concept of recursiveness or (as is generally agreed) computability has been formalised by various methods which have yielded equivalent concepts. In the above definition no clue is given to an operational semantics, and also there are no tools enabeling the user to describe usefull programs, except by claiming programs (i.e. indices) for the corresponding functions to exist within the enumeration.

In order to amend these shortcomings we have proposed to fix a representation style for functions by means of a programming language. Description of this language is omitted at this place; the reader is refered to [4]. All functions contained in this report are assumed to be self-evident. The link to the abstract enumeration is created by allowing the use of expressions like <x,y> for the pairing function and $\phi_i(x)$ for the result of elaborating the i-th program at x. On the other hand we introduce a powerfull operator *index* whose behaviour is characterized by:

if (int x) int: E is some routine denotation

then index ((int x) int: E) is an index in the abstract

enumeration for a program computing the same function as does the routine posessed by the denotation in the current evironment. This way we simultaneously "axiomatise" Church's thesis and enable ourself to "define" the s-n-m function by:

proc snm = (int i,x) int: index ((int y) int: $\phi_i(\langle x,y\rangle)$).

An important role is played by the total functions which may be defined by declarations like

proc τ = (int $i_1,\ldots,i_k$) int: index ((int x) int: $E(i_1,\ldots,i_k,x)$)

where $E(i_1,\ldots,i_k,x)$ denotes an integral unit containing $i_1,\ldots,i_k,x$ as

global identifiers. We call these total functions transformations of programs, and allow their declarations to be written by:

$$\phi_{\tau(i_1,\ldots,i_k)}(x) \Leftarrow E(i_1,\ldots,i_k,x)$$

which denotation now defines τ!

Transformations of programs are related to procedure declarations of recursive procedures: If f is a recursive procedure declared by

$$\underline{\text{proc}}\ f = (\underline{\text{int}}\ x)\ \underline{\text{int}}\text{: } T(\ldots,x,\ldots,f,\ldots)$$

where $T(\ldots,x,\ldots,f,\ldots)$ denotes the procedure body we may consider the corresponding transformation of programs

$$\phi_{\tau(i)}(x) \Leftarrow T(\ldots,x,\ldots,\phi_i,\ldots).$$

Since f satisfies $f(x) = T(\ldots,x,\ldots,f,\ldots)$ an index j for f must satisfy

$$\phi_j(x) = T(\ldots,x,\ldots,\phi_j,\ldots) = \phi_{\tau(j)}(x)$$

which shows that j is a fixed point under the transformation τ. Existence of such a fixed point is guaranteed by the so called recursion theorem:

THEOREM 2. *There exists a transformation λ such that for all indices i such that ϕ_i is a total function $\phi_{\lambda(i)}$ is a fixed point of the transformation $\phi_j \rightarrow \phi_{\phi_i(j)}$.*

PROOF. Define the transformation ρ by

$$\phi_{\rho(j)}(x) \Leftarrow \phi_{\phi_j(j)}(x)$$

meaning:

$$\phi_{\rho(j)}(x) \Leftarrow (\underline{\text{int}}\ \text{ind} = \phi_j(j);\ \phi_{\text{ind}}(x)).$$

Let the transformation κ be defined by

$$\phi_{\kappa(i)}(x) \Leftarrow \phi_i(\rho(x))$$

and let λ be the total function $\lambda i[\rho(\kappa(i))]$.

Now λ has the desired property, since assuming ϕ_i to be total we have

$$\phi_{\lambda(i)}(x) = \phi_{\rho(\kappa(i))}(x) = \phi_{\phi_{\kappa(i)}(\kappa(i))}(x) = \phi_{\phi_i(\rho(\kappa(i)))}(x) = \phi_{\phi_i(\lambda(i))}(x)$$

(1) (2) (3) (4)

showing $\phi_{\lambda(i)}$ to be a fixed point under the transformation $\phi_j \to \phi_{\phi_i(j)}$. □

Note that equations 1) and 4) only involve the definition of λ as a composition whereas 2) and 3) are based on the meaning of the transformations ρ and κ. In case the equations 2 and 3 may be understood as operational definitions the whole sequence of equations yield the conclusion:

"$\phi_{\lambda(i)}(x)$ is operationally defined by $\phi_{\phi_i(\lambda(i))}(x)$" .

This formulation gives strong evidence that $\phi_{\lambda(i)}$ indeed represents the least fixed point determined according to the underlying operational semantics. In particular if ϕ_i denotes the identity function so that $\phi_j \to \phi_{\phi_i(j)} = \phi_j$ denotes the identity transformation it is reasonably to expect that the function $\phi_{\lambda(i)}$ whose meaning is to be derived from

"$\phi_{\lambda(i)}(x)$ is operationally defined by $\phi_{\lambda(i)}(x)$"

represents the everywhere undefined function.

In abstract recursion theory however the proof uses nothing but the two transformations ρ and κ satisfying

$$\phi_{\rho(j)}(x) = \phi_{\phi_j(j)}(x) \quad \text{and} \quad \phi_{\kappa(j)}(x) = \phi_i(\rho(x)).$$

If these two equations are to be read extensionally we have in the above example nothing but the triviality:

$$"\phi_{\lambda(i)}(x) \quad \text{equals} \quad \phi_{\lambda(i)}(x)"$$

from which nothing can be derived on the convergence of the computations of $\phi_{\lambda(i)}$.

The above argumentation can be used as justification to call the transformation λ or equivalently the pair of transformations ρ and κ the "recursion theorem" semantics (based on ρ and κ) of recursive procedures. It remains questionable whether this semantics has any of the properties we may expect from a good semantics of recursive procedures. The example of the next section indicates that this is not the case. In particular it shows that the recursion theorem semantics does not need to conform to a "least fixed point" of some model of the transformation within some (hidden) operational semantics.

§3. The example

In the sequel we consider a fixed enumeration $(\phi_i)_i$. From recursion theory we recall the so called padding lemma [9], which tells us that for each function a sequence of new programs, computing this function can be generated effectively and uniformly:

LEMMA 3. [Padding lemma]. *There exists a transformation* α *such that*

1) α *is* 1 - 1, *monotonous in both arguments and satisfying* $\alpha(i,0) > i$

2) $\forall_i \forall_n [\phi_{\alpha(i,n)} = \phi_i]$.

PROOF. cf. [9] or [4].

Using the padding lemma each transformation of programs can be modified such that it becomes increasing in all arguments; moreover its range can be made disjoint from the range of another transformation (which may have to be modified analogously).

We may assume therefore without loss of generality that we are given

three transformations τ, ρ and κ and an index j_0 such that:

1) τ, ρ and κ are 1 - 1 increasing
2) R_τ, R_ρ and R_κ are mutually disjoint (recursive) sets.
3) $j_0 \notin R_\tau \cup R_\rho \cup R_\kappa$
4) $\phi_{j_0}(x) = x$ for every x
5) $\phi_{\tau(j)} = \phi_j$, $\phi_{\rho(j)}(x) = \phi_{\phi_j(j)}(x)$ and $\phi_{\kappa(j)}(x) = \phi_j(\rho(x))$.

Taking $\lambda = \rho \circ \kappa$ we may take λ to be the recursion theorem semantics for the enumeration $(\phi_i)_i$. In particular we may expect that $\phi_{\lambda(j_0)}$, the semantics of the procedure proc p = (int x) int: p(x), represents the everywhere undefined function.

Next we destroy the above enumeration, preserving however the meaning of the transformations ρ and κ. Since ρ and κ are 1 - 1 and increasing the sets R_ρ and R_κ are recursive and if $x \in R_\rho$ or $x \in R_\kappa$ the argument y such that $\rho(y) = x$ or $\kappa(y) = x$ can be computed. This justifies the following definition of a sequence of functions $(\phi^*_i)_i$:

$$\begin{aligned}\phi^*_i(x) = \ & \underline{\text{if}}\ \ i = \lambda(j_0)\ \ \underline{\text{then}}\ \ x \\ & \underline{\text{elif}}\ \ i = \rho(k)\ \ \underline{\text{then}}\ \ \phi^*_{\phi^*_k(k)}(x) \\ & \underline{\text{elif}}\ \ i = \kappa(k)\ \ \underline{\text{then}}\ \ \phi^*_k(\rho(x)) \\ & \underline{\text{else}}\ \ \phi_i(x)\ \ \underline{\text{fi}}\ .\end{aligned}$$

More formally a universal program for $(\phi^*_i)_i$ is described by

```
proc  fraud = (int i,x) int:
(int case := 0, prog;
for j from 0 to i do
    if i = λ(j0) then case := 1; goto found
  elif ρ(j) = i then case := 2; prog := j; goto found
  elif κ(j) = i then case := 3; prog := j; goto found
    fi od;
```

found:

```
case case in x,
                fraud(fraud(prog,prog),x),
                fraud(prog,ρ(x))
           out  φ_i(x)  esac
)# fraud #
```

For programs ϕ^*_i with $i \notin R_\rho \cup R_\kappa$ we have $\phi^*_i = \phi_i$; in particular $\phi^*_{\tau(j)} = \phi_{\tau(j)} = \phi_j$. This shows that $(\phi_j)_j$ can be 1 - 1 embedded into $(\phi^*_i)_i$. Since on the other hand fraud is recursive $(\phi^*_i)_i$ can be embedded into $(\phi_j)_j$.

Application of the padding lemma and the Myhill isomorfism theorem ([9] or [4]) proves that $(\phi^*_i)_i$ is an effective enumeration.

Next we must show that ρ and κ satisfy the conditions:

$$\phi^*_{\rho(i)}(x) = \phi^*_{\phi^*_i}(x) \quad \text{and} \quad \phi^*_{\kappa(i)}(x) = \phi^*_i(\rho(x)).$$

These relations follow straightforward by the definition of $(\phi^*_i)_i$ except for the case $\phi^*_{\rho(\kappa(j_0))}(x)$. For this special case we have

$$\phi^*_{\rho(\kappa(j_0))}(x) = x \quad \text{and}$$

$$\phi^*_{\phi^*_{\kappa(j_0)}(\kappa(j_0))}(x) = \phi^*_{\phi^*_{j_0}(\rho(\kappa(j_0)))}(x) = (\text{since } j_0 \notin R_\rho \cup R_\kappa)$$

$$= \phi^*_{\phi_{j_0}(\rho(\kappa(j_0)))}(x) = \phi^*_{\rho(\kappa(j_0))}(x) = x$$

which proves the desired equality.

Finally consider the transformation $\phi^*_{j_0} = \sigma$. Since $j_0 \notin R_\rho \cup R_\kappa$ we have $\phi^*_{j_0} = \phi_{j_0} = \text{id}$. Consequently σ denotes the identity transformation. Its fixed point, according to the recursion theorem semantics based on ρ and κ, equals $\phi^*_{\lambda(j_0)} = \text{id}$. Thus the identity function has become the meaning of the procedure p declared by

```
proc p = (int x) int: p(x).
```

This completes the description of the example.

REFERENCES

1. BAKKER DE, J.W., *Least fixed points revisited*, to appear in Theoretical Computer Science.

2. BLUM, M., *A Machine Independent theory of the complexity of recursive functions*, J. Assoc. Comput. Mach. 14 (1967) 322-336.

3. DOWNEY, P.J. & R. SETHI, *Correct computation rules for recursive languages*, Proc. FOCS 16 Symp. Berkeley (1975) 48-56.

4. EMDE BOAS VAN, P., *Abstract Resource-Bound Classes*, Ph.d. Thesis Univ. of Amsterdam. (1974). ed. Math. Centre, Amsterdam.

5. MANNA Z. & J. VUILLEMIN, *Fixpoint approach to the theory of computation comm*, ACM 15 (1972) 528-536.

6. MANNA, Z., *Mathematical Theory of Computation*, McGraw-Hill (1974).

7. MANNA, Z. & A. SHAMIR, *The optimal fixed point of recursive programs*, Proc. 7 ACM Symp. Theory of Computing. Alberquerque (1975) 194-206.

8. ROGERS, H., *Gödel numberings of partial recursive functions*, J. Symb. Logic. 23 (1958) 331-341.

9. ROGERS, H., *The theory of recursive functions and effective computability*, McGraw Hill N.Y. (1967).

10. ROEVER DE, W.P., *Recursive Program Schemes, semantics and proof theory*, Math. Centre Tracts 70 (1976).

11. ROEVER DE, W.P., *First order reduction of call-by-name to call-by-value*, Proc. Math. Found. Comp. Sci. Symp. Marianske Lazne (1975). Springer LCS. 32. 377-398.