SEMANTIC MODELS OF PARAMETER PASSING *

by

Richard E. Fairley

Department of Computer Science
University of Colorado
Boulder, Colorado

TR #CU-CS-016-73 March, 1973

* This work supported by NSF Grant #GJ-660.

ABSTRACT

This paper presents semantic models of four parameter passing mechanisms utilized in various algebraic programming languages: Call By Value, Copy Restore, Call By Reference, and Call By Name. The correspondence between actual parameter values and formal parameter names is established by use of an environment directory and a denotation component. The environment directory associates each identifier in the program with a unique name, and the denotation component associates unique names with information concerning the value of the identifier denoted by the unique name. The various parameter passing mechanisms are first described informally. A tree structured model of each parameter passing mechanism is then presented. The paper concludes with a discussion of formalizing the models.

I. <u>INTRODUCTION</u>

One of the more interesting problems in the semantic definition of an algebraic programming language is specification of the correspondence between formal parameter names in a procedure and actual parameter values associated with the formal parameters. Four parameter passing mechanisms utilized in the various programming languages are: (1) Call By Value, (2) Copy Restore, (3) Call By Reference, (4) Call By Name. Table I summarizes the parameter passing mechanisms utilized in six well-known programming languages.

This paper presents semantic models of the four parameter passing mechanisms. An informal description of the various mechanisms is presented first. This is followed by a discussion of representation dependent models of the mechanisms. The paper concludes with a discussion of formalization of the models.

II. <u>INFORMAL DESCRIPTION</u>

II.1 <u>Call By Value</u>

Call By Value parameter passing requires that the actual parameter be evaluated at the time of procedure call. The memory register associated with the formal parameter is then initialized to this value, and references to the formal parameter in the procedure body are treated as references to the local memory register in which the initial value of the actual parameter was stored. Due to the fact that a copy of the value associated with the actual parameter is copied into the local memory register, transformations on the parameter value within the procedure body are isolated from the actual parameter value. Because of this isolation of values, Call By Value cannot be used to communicate calculated values back to the calling program.

II.2 Copy Restore

Copy Restore is similiar to Call By Value in that a copy of the actual parameter value, at the time of procedure call, is copied into a register associated with the formal parameter. Copy Restore differs from Call By Value in that a copy of the final value associated with the formal parameter is copied back into the register associated with the actual parameter prior to return from the called procedure. Thus, Copy Restore can be used to communicate calculated values back to the calling procedure. However, intermediate values assigned to the formal parameter are isolated from the value of the actual parameter.

II.3 Call By Reference

In Call By Reference, the address (name) of the actual parameter at the time of procedure call is passed to the procedure as the value to be associated with the corresponding formal parameter. References to the formal parameter in the procedure body result in indirect addressing references through the formal parameter register to the memory register associated with the actual parameter in the calling procedure. Thus, transformations of formal parameter values are immediately transmitted to the calling procedure, because both the actual parameter and the formal parameter refer to the same register.

II.4 Call By Name

Call By Name parameter passing is similiar to Call By Reference, in that the address of the actual parameter is passed to the called procedure. Call By Name differs from Call By Reference in that the address of the actual parameter is re-evaluated each time the corresponding formal parameter is referenced in the procedure body. Thus, the actual parameter address associated with the formal parameter may change during procedure execution. In that case, the formal parameter will reference different actual parameter registers during procedure execution.

III. SEMANTIC MODELS

This section of the paper presents representation dependent models of the parameter passing mechanisms described in the previous section. The correct association of actual parameters with formal parameters is accomplished by the use of an environment directory and a denotation component. The environment directory associates each identifer in the program with a unique name, and the denotation component associates unique names with information concerning the value of the identifier denoted by the unique name.

The environment directory and the denotation component are represented as labeled trees. The branches of the environment tree are labeled by identifier names and the leaves are the unique names associated with the identifiers. In the denotation tree, the branches are labeled by unique names, and the leaves are either values or information that can be used to calculate values. The branch labels serve as selectors into the trees, and the leaves represent the values associated with the corresponding selectors.

In the following discussion, it is assumed that the language being modeled is a block structured ALGOL-like language. For simplicity, only simple identifiers (those directly associated with single memory registers containing numerical values) will be permitted as actual parameters in procedure calls. The next section discusses passing of expressions, and reference-valued variables as actual parameters.

Consider the following program skeleton in which "spec" denotes the parameter passing mechanism being utilized:

```
begin int A, B, C;
      proc P(X,Y);  "spec"  X,Y;
           :        :
      end
      :       :
      P(A,B);
      :       :
```

This program will be referred to in the following discussion.

The environment tree, E, and the denotation tree, De, for the main program block are presented in Figure 1. The values associated with identifiers A, B, and C are denoted as v(A), v(B), and v(C) in the denotation tree. The denotation of the procedure P, De(P), includes the formal parameters, the "spec", and the text of the procedure body.

The following subsections describe representation models of Call By Reference, Call By Value, and Call By Name. Copy Restore is described as an extension of Call By Value.

III.1 Call By Reference

Figure 2 illustrates the environment and denotation trees during procedure activation assuming "spec" denotes the Call By Reference parameter passing mode. Formal parameters X and Y are assigned the same unique names as the corresponding actual parameters, A and B. Thus, the formal parameters and the actual parameters refer to the same memory registers in the denotation component. Identifier C is accessible as a global variable in the procedure body. A copy of the calling procedure environment has been saved in a "dump" component, Dp. Upon procedure exit, the original environment is restored, thus rendering the formal parameters inaccessible to the main program.

III.2 Call By Value

Figure 3 depicts the situation during procedure activation when "spec" denotes the Call By Value parameter passing mode. Formal parameters X and Y are assigned individual unique names in the environment tree, and the denotation components corresponding to those unique names are initialized to the values of the actual parameters at the time of procedure call.

Thus, the actual parameter memory cells are isolated from the formal parameter memory cells. Upon procedure exit, the calling program environment is restored, making the formal parameter values inaccessible to the calling program following procedure exit. If the procedure is called again, the formal parameters will be associated with new unique names, and new entries will be created in the denotation tree.

Copy Restore is similiar to Call By Value, except that the final values of the formal parameters are copied into the actual parameter registers by the appropriate assignment statements.

III.3 Call By Name

Figure 4 illustrates the semantic model of Call By Name procedure activation. In Call By Name, the denotation of the formal parameter is the text of the actual parameter. References to the formal parameter within the procedure body result in evaluation of the actual parameter name in the calling procedure environment (which was saved in the dump). The name of the actual parameter is then used as a selector to retrieve the corresponding unique name in the environment tree. The unique name of the actual parameter is then used as a selector in the denotation tree to associate the correct register with the formal parameter.

III.4 Extensions to the Model

The semantic model of parameter passing is easily extended to handle passing of expressions (including constants) and reference variables. A reference variable is an identifier whose denotation is a unique name. Thus, pointers, labels, and procedure names can be passed as reference variables.

Expressions passed by Value are evaluated in the calling environment and the value thus obtained becomes the initial denotation of the formal parameter. Expressions passed by Reference can be modeled as Call By Value. Reference variables passed by Reference are handled by making the unique name which is the value of the actual parameter be the denotation of the formal parameter. Call By Name for expressions can be modeled as Call By Value. Call By Name for reference variables is treated as Call By Reference, with the usual re-evaluation of the actual-formal parameter correspondence in the procedure body.

In the previous section, it was assumed that a nested block structure language was being modeled. There are several ramifications to parameter passing in such a language:

1. In accordance with the Renaming Rule, global variables in a procedure body are those known at the point of procedure declaration. This requires storing of the environment tree at the point of procedure declaration as a component of the procedure denotation. The environment for procedure activation is then established by appropriate modification of the saved environment. Modification of the saved environment also serves to resolve naming conflicts in the procedure body.

2. Call By Name formal parameter references require reevaluation of the actual parameter name in the calling environment. This requires saving of the environment at the point of procedure call in the denotation of the formal parameter, along with the text of the actual parameter.

3. Nested procedure calls from within procedures (including recursive calls) require that the dump component be structured as a stack of successive calling environments which are restored in LIFO order upon successive procedure exits.

IV. FORMALIZATION OF THE MODEL

Semantic models of programming langu ges are valuable to language users, language implementors, language designers, and language theorists. Users and implementors are primarily interested in representation dependent models that provide insight into the meaning of various language constructs, and which clarify subtle points in the language. On the other hand, designers and theorists are concerned with formal models of a language which permit logical, deductive statements about the properties of the language such as the completeness of semantics, inconsistency of semantics and equivocacy (semantic ambiguity).

In order to facilitate communication between the various interest groups, it is desirable that a semantic model have both a representation and a formalization. The semantic models of parameter passing presented in this paper have been formalized by the author (2) within the framework of the Vienna Definition Language (3). The Vienna Definition Language model of parameter passing comprises an abstract machine which has internal states and a transition function. The internal states incorporate highly structured counterparts of the tree representations of the unique name, environment, denotation, and dump components of the model. In addition, a control component contains the abstract machine instructions.

The transition function maps machine states into successor states, and the sequence of machine states generated during program execution defines the semantics of the program. Formalization of the abstract machine is achieved by describing the machine state and transition function in terms of primitive functions. The primitive functions are in turn axiomatized. Thus, it is possible to transform the tree representations into primitive function notation and to describe the transition function in terms of mappings of primitive functions into primitive functions. It is then possible to make rigorous statements about a program and to generalize from properties of particular programs to properties of the programming language. An example of this technique will be presented in a later paper. Due to space limitations, it is not possible to present the formal model of parameter passing in this paper. The formalization is contained in Reference 2.

V. SUMMARY

This paper has discussed the semantics of parameter passing in algebraic programming languages. The parameter passing mechanisms of Call By Value, Copy Restore, Call By Reference, and Call By Name were described informally and then modeled by tree structured representations. The paper concluded with a discussion of formalizing the models within the framework of the Vienna Definition Language.

REF:cah

REFERENCES

1. Gries, David, "Compiler Construction for Digital Computers," John Wiley and Sons, Inc., 1971.

2. Fairley, Richard, "The Formal Definition of A Parameter Passing Language," University of Colorado Computer Science Report #CU-CS-010-72, December 1972.

3. Lucas, P., "Method and Notation for the Formal Definition of Programming Languages," IBM Laboratory Vienna TR-25.087, June 1968.

REF:cah

TABLE I

Parameter Passing Mechanisms in Six Algebraic Programming Languages

LANGUAGE	MECHANISM
FORTRAN	Reference, Copy Restore
ALGOL 60	Value, Name
PL/1	Reference, Value
PASCAL	Reference, Value
EULER	Reference, Value, Name
ALGOL 68	Reference, Value, Name

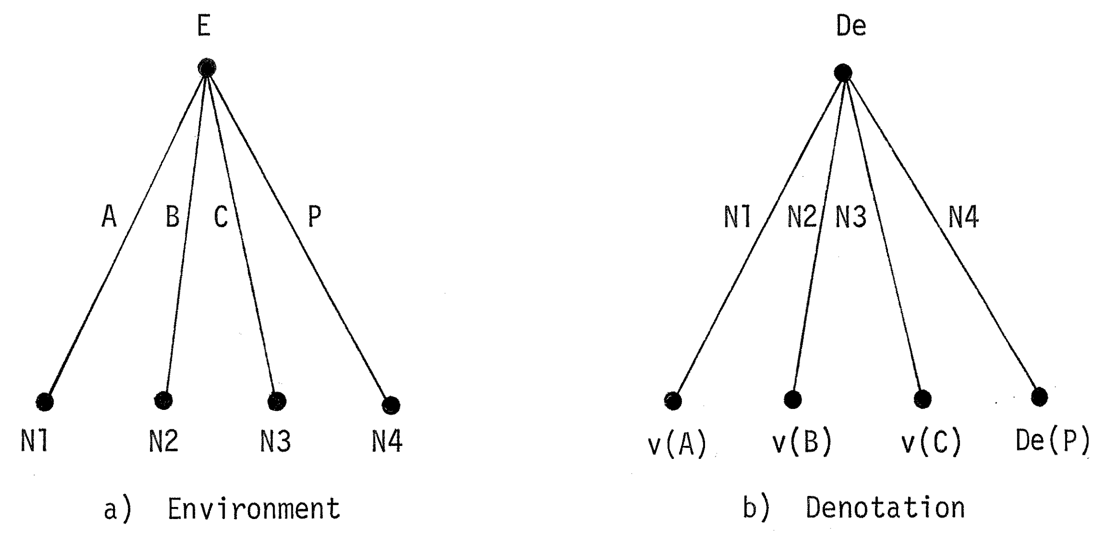


Figure 1

Main Program Environment and Denotation Trees

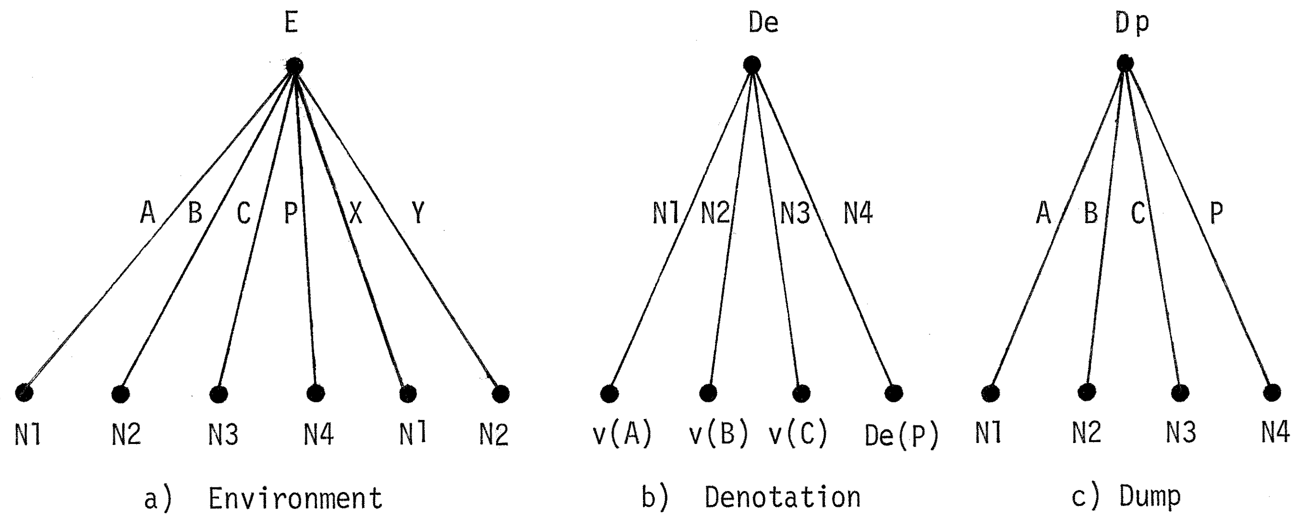
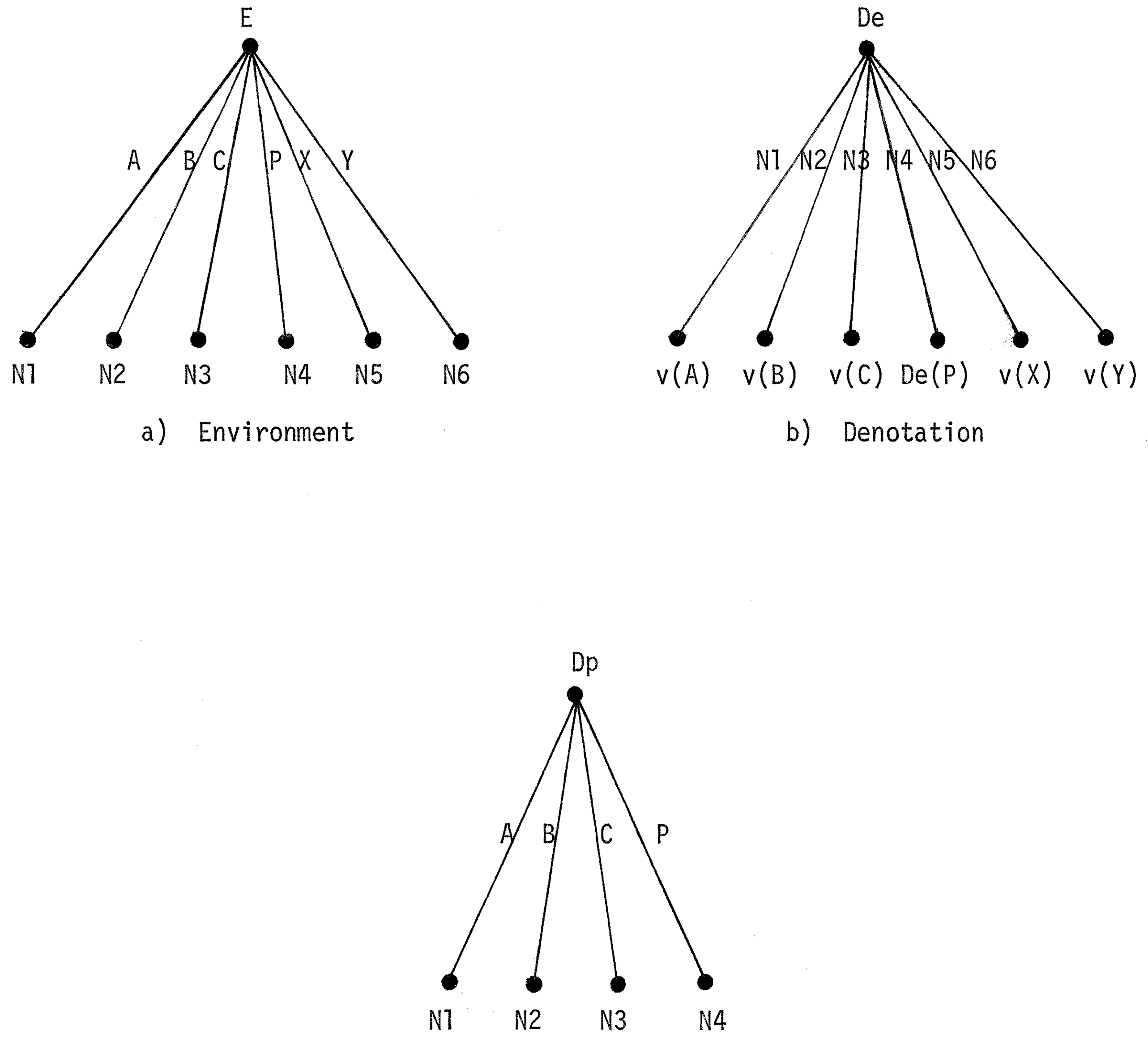


Figure 2

Environment, Denotation, and Dump Trees
during Procedure Activation for Call By Reference



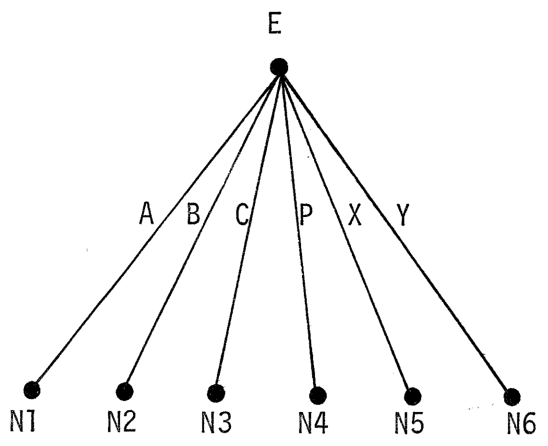
a) Environment

b) Denotation

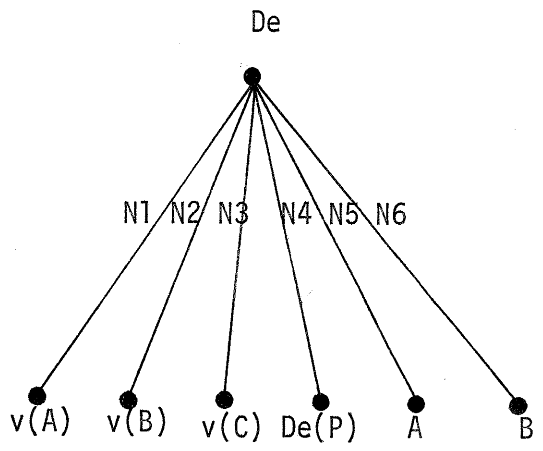
c) Dump

Figure 3

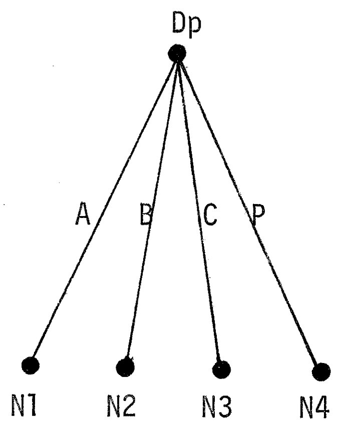
Environment, Denotation, and Dump Trees during Procedure Activation for Call By Value



a) Environment



b) Denotation



c) Dump

Figure 4

Environment, Denotation, and Dump Trees during Procedure Activation for Call By Name