27
11-2-76
rept to ITS
2

IS-3951

A SURVEY OF SEMANTIC
MODELING TECHNIQUES

Carol L. Smith

AMES LABORATORY, USERDA
IOWA STATE UNIVERSITY
AMES, IOWA

MASTER

Date Transmitted: July 1975

PREPARED FOR THE U. S. ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION
UNDER CONTRACT W-7405-eng-82

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

A SURVEY OF SEMANTIC MODELING TECHNIQUES

Carol L. Smith

Ames Laboratory, ERDA
Iowa State University
Ames, Iowa 50010

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Date Transmitted: July 1975

PREPARED FOR THE U.S. ENERGY RESEARCH AND
DEVELOPMENT ADMINISTRATION UNDER CONTRACT NO. W-7405-eng-82

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

fy

—NOTICE—

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Available from: National Technical Information Service
U. S. Department of Commerce
P.O. Box 1553
Springfield, VA 22161

Price: Microfiche	\$2.25
Paper Copy	\$4.00

TABLE OF CONTENTS

	PAGE
ABSTRACT	iv
0. INTRODUCTION	1
1. MOTIVATION FOR SEMANTIC MODELING	3
2. SEMANTICS OF COMPUTATIONAL PROCESSES	5
2.1 MATHEMATICAL MODELS	7
2.2 AXIOMATIC MODELS	12
2.3 OPERATIONAL MODELS	13
3. SEMANTICS OF DATA STRUCTURES	16
4. SEMANTICS OF NAME ACCESSING	19
5. APPLICABILITY OF FORMAL SEMANTIC MODELS	22
6. BIBLIOGRAPHY	25

ABSTRACT

The analysis of the semantics of programming languages has been attempted with numerous modeling techniques. By providing a brief survey of these techniques together with an analysis of their applicability for answering semantic issues, this report attempts to illuminate the state of the art in this area. The intent is to be illustrative rather than thorough in the coverage of semantic models. A bibliography is included for the reader who is interested in pursuing this area of research in more detail.

Distribution List

Chicago Patent Office	1
Oak Ridge	27
Ames Laboratory Library	5
Carol L. Smith	6
Charles T. Wright	11

Total	50
-------	----

0. INTRODUCTION

The main motivation for applying modeling techniques to the study of programming languages is to create a framework in which questions about programming languages can be clearly stated and hopefully answered. In classifying the various types of questions that one might ask about a programming language, three aspects of language study can be distinguished: syntax, semantics, and pragmatics.

Syntactic issues are defined as those which deal with the structure of the programming language. Questions such as 1) is "x13" a legal identifier in ALGOL 60? 2) what is the scope of a variable in a PL/I program? and 3) what constitutes a legal argument for the FORTRAN SQRT routine?, fall into this category. More specifically, a question involving the form of a program and independent of the execution of that program falls into the syntactic category.

Semantic issues, on the other hand, are totally dependent on program execution. These issues deal with the meaning of a language. Consider, for example, the FORTRAN assignment statement "A = B + C". Without knowing anything about the execution of this statement, it can be verified that this is a valid FORTRAN statement (a syntactic issue). However, in

order to define the meaning of this statement, for instance its effect on the variables "A", "B" and "C", we must know something about the environment in which it is executed.

The third classification of language study, pragmatics, is somewhat illusive and perhaps the most difficult of the three. Pragmatics covers all questions relating to programming languages that are neither syntactic or semantic. One reason that these questions are so difficult is that they often involve subjective analysis of phenomena influenced by many factors. The readability of a programming language and its ease of use are examples.

This report is concerned with a survey of the modeling techniques that have been used in addressing semantic issues. Section 1 discusses the motivation for using such techniques. Sections 2, 3 and 4 outline modeling techniques that have been used in analyzing computational processes, data structures and name accessing, respectively. The intent of these sections is to be illustrative rather than thorough in its coverage. Section 5 contains an analysis of the applicability of the various modeling techniques in answering semantic questions. A bibliography is included for the reader who is interested in pursuing this area of research in more detail.

1. MOTIVATION FOR SEMANTIC MODELING

The motivation for a formal semantic definition is best understood in terms of the needs of those groups who would use such a definition. Historically, it has been claimed that a formal semantic definition should be useful to language designers, language implementors and users (2). In this survey another group will be considered; namely the theoretical computer scientist who is interested in studying programming languages and programming language semantics as objects in their own right. The criteria which a formal semantic model should satisfy in order to be useful to these groups are outlined below.

I. Language designers

- A. The model should provide a theoretical framework for the design and comparison of various language constructs.
- B. The model should allow for the detection of ambiguous, incompatible or contradictory language constructs. Ideally, it should also act as a catalyst in the design of new language constructs.
- C. The model should allow small changes in the language constructs to be realized by small changes in the language description.
- D. The model should describe independent language constructs independently.

II. Language implementors

- A. The model should clearly indicate to the implementor those parts of the language definition that are

incompletely specified and therefore left to the implementor to define. The model must also indicate the restrictions that must be adhered to by the implementor in completing the definition.

- B. The model should clearly delimit the actions which can be performed at compile-time as opposed to those which must be done at run-time.
- C. The model should provide a framework within which the implementor can verify that the implementation methods he had chosen satisfy the language definition. Furthermore, the model should allow for the comparison of various meaning-preserving implementations.
- D. In general, a formal semantic model should provide the implementor with a complete, precise and unambiguous definition of the language he wishes to implement.

III. Language user

- A. The formal model should provide the user with a concise, readable and transparent description of the language.
- B. The model should provide a framework conducive to studying and proving properties about individual programs.

IV. Theoretical computer scientist

- A. The model should be applicable to a large class of programming languages. This would enable the computer

scientist to use a common framework in which
to analyze and compare various languages.

The criteria which have been listed are quite demanding. Hoare and Lauer (27) have recently suggested that more than one model may be necessary to satisfy the needs of these diverse groups. In any event, it seems appropriate to consider the extent to which existing models have met the above objectives.

2. SEMANTICS OF COMPUTATIONAL PROCESSES

To the best of the author's knowledge, no thorough attempt has been made to systematically classify and analyze the research that has been undertaken in specifying the semantics of programming languages. It is not intended that this task will be accomplished in this section, but steps in this direction will be presented.

It is difficult to state exactly when and why research in programming language semantics was initiated, but it seems quite likely that the publication of the Revised ALGOL 60 Report (47) stimulated much of this work. The contrast between the precise, clear BNF syntactic description of ALGOL 60 and the imprecise and often obscure natural language semantic description focused attention on the need for a formal semantic descriptive model. It was hoped that a semantic model could be developed that would contribute as much to semantic definition as BNF had contributed to syntactic description.

Prior to 1962, very little attention had been paid to semantic descriptions. About this time, Steel summarized the state of the art in this area as follows: "... the only existing method that adequately describes a programming language is the exhibition of a machine language version of a compiler for the programming language being described, together with a citation of the explicit machine on which the compiler is expected to function" (55, p. 25). Over the past 12 years, a great deal of research has been directed toward improving this situation. The following paragraphs in this section outline the work that has been undertaken in modeling the semantics of computational processes.

For purposes of comparison, three descriptive categories will be used to distinguish semantic approaches: mathematical approaches, axiomatic approaches and operational approaches. These categories are derived from Wegner's analysis of semantic modeling techniques (64). Mathematical semantic models treat the objects they represent as if they have an existence independent of any particular representation. In this approach, a mathematical formalism is used as a host for the programming language constructs which are being defined. The semantics of the language constructs are then defined in terms of the semantics of the mathematical host. The axiomatic approach is based on the foundations of mathematical logic to make assertions about the state of a computing system at various points of program execution. The semantics of a

programming language construct is defined in terms of the assertions that can be made prior to execution of the construct (input assertions) and the assertions that can be made after its execution (output assertions). The operational approach comes closest to modeling the actual execution of a computing device. If the mathematical and axiomatic approaches are characterized by the fact that they are concerned with what is computed, then it would be appropriate to state that in the operational approach emphasis is placed on how the computations are performed. In the operational approach, information structures are used to model the computer state. The semantics of a language construct is then defined in terms of the effect its execution has on the information structures.

Numerous semantic models for computational processes have been proposed using the semantic approaches described above. In the following paragraphs, some of these attempts will be outlined.

2.1 MATHEMATICAL MODELS

Several semantic models have been proposed which use the mathematical formalism of λ -calculus notation (10). This survey will consider two of these, the models proposed by Landin (34, 35) and Strachey (56).

The popularity of the λ -calculus is related to the fact that there is some degree of similarity between λ -calculus constructs and certain programming language constructs. The obvious correspondence between the notions of local and nonlocal variables in programming languages and the concepts of free and bound variables in the λ -calculus provides

a natural mechanism for modeling ALGOL-like block structure. In addition, the functional nature of the λ -calculus provides a natural model for function definition and function application. Other programming language constructs, such as assignments and jumps, are not easily modeled in the λ -calculus. The semantic models proposed by Landin and Strachey have used different approaches in overcoming these drawbacks.

Landin has proposed the definition of programming language constructs in terms of their "compilation" into extended λ -calculus expressions. The extensions, program points and assigners, are included to handle labels and assignments. Landin's approach can be outlined as follows: 1) translate the language constructs that are being defined into extended λ -expressions, 2) specify a mechanical evaluation procedure for the extended λ -expressions, and 3) identify the semantics of a language construct with its associated evaluation in extended λ -calculus form.

The pure λ -calculus already has the notion of evaluation specified in the formalism by means of reduction rules defined on λ -calculus expressions. Landin's extension of the λ -calculus necessitated the specification of an additional evaluation procedure since the reduction rules were no longer sufficient. In contrast to this approach, Strachey proposed to define programming language semantics in terms of pure λ -calculus expressions, thus eliminating the need for any additional evaluation mechanisms. Strachey handled assignments and jumps strictly

within the confines of pure λ -calculus notation. Jumps were accommodated, for the most part, by textual rearrangement and use of the λ -calculus conditional. Assignments were modeled via the postulation of the primitive notion of a store, where a store represents a mapping from L-values (generalized addresses) to R-values (the contents of an L-value). Programming language constructs were then defined in terms of λ -expressions which mapped one store into another.

Another mathematical approach used in defining programming language semantics is based on Markov Algorithms. Markov Algorithms define a computation on a given input string in terms of a set of transformation rules which consist of left-hand and right-hand parts. If the input string contains a subsequence of symbols which corresponds to the left-hand side of a transformation rule, then the symbols on the right-hand side of this rule replace the subsequence of symbols in the input string. This process is repeated with the modified input string until no more replacements are applicable. One of the earliest proponents of this approach was van Wijngaarden (60), who proposed the use of extended Markov Algorithms for defining language semantics. His extensions included the use of metalinguistic variables in the transformation rules together with the notion of a dynamically growing set of rules. An application of this approach can be found in de Bakker's formal definition of ALGOL 60 (1). Related work in this area can be found

in Caracciolo (8), Caracciolo and Wolkenstein (9), van Wijngaarden, et al. (61), and Ledgard (36, 37).

The formalism of algebraic theories has provided a rich source for semantic modeling. In a recent paper, Goguen and Thatcher (21) have shown that many apparently diverse approaches to semantics can be discussed within the framework of "initial algebra" semantics. They define an initial algebra as follows: Given a class of algebras $\underline{C}$, $S \in \underline{C}$ is initial if for each $A \in \underline{C}$ there exists a unique homomorphism $h_A: S \rightarrow A$. Within this definition, a formalized syntactic definition, S , can be considered an initial algebra for the class $\underline{C}$ of semantic algebras. A semantic definition is provided with the specification of the unique homomorphism $h_A: S \rightarrow A$, where A is in $\underline{C}$.

Knuth has outlined a pseudo algebraic approach for defining the semantics of context-free languages (33). His approach to semantic specification is tied directly to the syntactic representation provided by context-free grammars. He associates meaning or semantics with the terminals and nonterminals of the grammar. The semantics consist of both synthesized and inherited attributes, where synthesized attributes depend only on the descendants of nonterminals in the derivation tree and inherited attributes depend on the ancestors which appear in the derivation tree. The semantics of a string in the language is determined by the composition of the semantics of the terminals and nonterminals used in the derivation of the string. Subsequent work using Knuth's approach can be found in Wilner (66) and Fang (16).

A second algebraic approach, based on a lattice-theoretic formalism, has been proposed by Scott and Strachey (53, 54, 57). To a large extent, this approach is closely related to the Strachey λ -calculus approach discussed earlier. One of the objections to the λ -calculus semantic models was that there existed no set theoretic models for the λ -calculus. The difficulty in finding a set-theoretic basis for the λ -calculus is related to the paradoxical notion of the set of all sets. Scott demonstrated how a set-theoretic model for the λ -calculus could be constructed by restricting ones attention to only continuous functions (54). In a semantic model proposed by Scott and Strachey, computable functions are modeled by continuous functions defined on complete lattices. A system state (S) is modeled as a complete lattice and language commands are modeled as producing state transformations $[S \rightarrow S]$. These transformations are defined to be continuous functions on complete lattices. This class of functions is guaranteed to have minimal fixed points (Tarski, 58). The existence of these minimal fixed points guarantees the unique definition of semantics in this model. The Scott and Strachey approach has been used in specifying the semantics of QUEST (59), ALGOL 60 (46), and PL/I (3).

Quite recently, there has been a great deal of work undertaken in attempting to relate the notions of language semantics to the concepts of category theory. The intent here is to represent programming language models and the relationship between them as functors between

various categories. The interested reader should consult Goguen, et al. (22) and Goguen and Thatcher (21) for extensive bibliographies of research in this area.

2.2 AXIOMATIC MODELS

In the axiomatic approach to semantic definition, memory states are no longer explicitly modeled. In this approach, emphasis is placed on what properties are satisfied by a memory state at a particular point in the execution of a program. The formalism of predicate calculus is used in stating these properties. In general, the semantics of a programming language command Q is expressed by an assertion of the form

$$P\{Q\}R$$

where P and R are propositional formulae representing properties of memory states. The interpretation of this statement asserts that if P is true prior to execution of Q , and if Q terminates, then R is true after the execution of Q . Initial work in applying axiomatic semantics is attributable to Floyd (18). More recently, research in this area has been undertaken by Hoare (26), Hoare and Wirth (28), Manna (44, 45), and Cadiou and Manna (7).

Axiomatic semantics is particularly well suited to accommodate proof of correctness issues. The user can discuss his program in terms of assertions on memory states and hopefully can derive a proof of the program's correctness in terms of assertions on the final memory state.

2.3 OPERATIONAL MODELS

There are two basic approaches for specifying the operational semantics of a programming language: interpreter-based models and compiler-based models. Both of these approaches are normally syntax driven in the sense that actions are dictated by the recognition of syntactic entities. In an interpretive approach, the recognition of a syntactic unit in the source program is directly associated with a corresponding memory state transformation. In a compiler-based model, the recognition of a syntactic unit in a source program written in language L is associated with a language construct from some target language L'. The "execution" of the L' construct, subsequent to the complete translation of the L source program, produces the desired memory state transformation. In general, there are two methods of defining how the execution of the L' program will proceed. This execution can be specified by an interpreter-based model for L' or L' can be defined to consist of primitive constructs which require no further definition. In either event, compiler-based approaches are justifiably criticized on the grounds that they provide only a restatement of the semantic definition problem, rather than a solution (65). In this survey the compiler-based models of Feldman and Wirth and Weber and the interpretive models proposed by McCarthy, the IBM Vienna group and Johnston will be discussed.

As previously mentioned, the unique characteristic of the operational approach is the use of information structures to

explicitly model memory states. In this sense, operational models are representation-dependent definitional frameworks. The type of information structure used in an operational model provides one criterion for classifying an operational scheme.

McCarthy (41, 42), the founder of operational semantics, used the notion of a state vector as the information structure in his approach. A state vector is defined to be a list of ordered pairs of the form (identifier, value). At any given point t during execution, the state vector contains the identifiers known at time t together with the values currently associated with those identifiers. McCarthy uses two primitive functions to access and update state vectors. The semantics of a language construct is expressed via the state vector transformation that it induces. McCarthy has used the state vector approach to define the semantics of a subset of ALGOL 60 (43). This subset did not include block structure, thus insuring that the identifiers in the state vectors were unique.

The development of the programming language PL/I was accompanied by the development of a definition language for the language. This definition language, the Vienna Definition Language (VDL), uses labeled trees as its basic information structure. In the VDL model, an interpreter state is usually modeled as a tree with three labeled components: a text component, a memory component and a control component. The first step in driving a VDL semantic specification is to convert programs written in the source language (concrete programs) into abstract

representations or "abstract syntax". The concept of an abstract syntax was introduced by McCarthy (43). Programs represented in "abstract syntax" form have a tree structure in which the labels on the edges are used to access particular instructions. The abstract syntax representation of the program is held in the text component of the interpreter state. The memory component of the interpreter state is used to hold tree representations of the data values used in a computation. The control component contains information which specifies how instructions in the abstract program map one interpreter state to another. The semantics of a language construct is defined in terms of its effect on the interpreter state. The defining document on the VDL approach is Lucas, et al. (39). Other papers demonstrating the applicability of VDL include Henhapl and Jones (24), Lucas (38), Lucas and Walk (40), Walk, et al. (63) and Wegner (65).

The Contour Model proposed by Johnston (29), has a two dimensional tree-like information structure base. It has been claimed that this model is at least as powerful as the VDL approach and is more amenable to a discussion of implementational issues (5). The Contour Model is based on the concepts of nested block structure, accessing environments, labels and cell retention. In this model, computations are represented by sequences of snapshots, where each snapshot is a data structure (contour configuration) consisting of cells and their contents. The Contour Model is most effective in modeling computations expressed in

block structured languages. A contour is allocated upon block or procedure entry and deallocated (unless retention is indicated) upon exit. The nesting of the contours reflects the block structured nature of the computation. The relation between source programs and their semantics is specified by defining an interpreter which transforms source language commands into operations on contours. The use of the Contour Model in defining the semantics of the programming language Oregano is presented in Berry (5). Recent theoretical development of the model is presented in Johnston (30, 31) and Johnston et al. (32).

The notion of defining the semantics of a language in terms of a standard compiler for the language was originally proposed by Garwick (19). In attempting to realize this objective, Feldman (17) proposed a Formal Semantic Language (FSL) which defines a high-level programming language in terms of code generation routines defined on an abstract computer. The FSL primitives, e.g., JUMP, MULTIPLY, ASSIGN and PLUS, are not defined. As a result FSL can be criticized on the basis that the defining language is more complex than necessary and very much in need of semantic definition itself. This objection was satisfied in the Wirth and Weber approach to compiler-based semantic definition (67, 68). Wirth and Weber defined the semantics of Euler in terms of elementary operations on a pushdown stack machine.

3. SEMANTICS OF DATA STRUCTURES

The semantic models discussed up to this point have been concerned with the definition of the semantics of computational processes. A great deal of work has also been undertaken in developing models which

define the semantics of data structures. Since it is the author's contention that there is no clear cut separation between research in programming languages and research in data structures, it is appropriate to include a discussion of research in data structure semantics in this survey.

Rosenberg (48, 49) has proposed a formalism called the Data Graph Model in which the semantics of data structures can be expressed. A data graph is obtained from a data structure by ignoring the specific data items which appear at the nodes of the structure and concentrating only on the linkages in the structure. Rosenberg has attempted to relate the logical concept of a data structure to its physical representation, i.e., its implementation in a computer. In particular, he uses the data graph representation of structures to detect structural uniformities which can be exploited in implementing the accessing of nodes within a computer. Rosenberg has characterized two classes of data graphs which are realizable by two distinct implementation mechanisms. Rooted or addressable data graphs are those which are implementable via relative addressing while free-rooted or uniformly translatable data graphs admit of relocatable implementations. An application of this model in analyzing the properties of data graphs which correspond to extendible arrays can be found in Rosenberg (50-52).

Earley (12, 13) has suggested that there are three hierarchical levels at which data structures should be discussed: the relational level, the access path level and the machine level. The relational

level represents the most abstract level in this hierarchy. At this level a mathematical description of the structure is given along with a specification of how the structure may be manipulated. This level does not describe the access paths that will be used nor how the structure may change. The access path level makes explicit the access relationships that exist between the individual data items, but it does not specify how these access paths should be implemented. Earley uses a V-Graph Model to describe structures at the access path level. In a V-graph, the nodes represent parts of the structure and the links represent access paths. Finally, at the machine level, the actual implementation of a structure on a machine is discussed. This multilevel approach to data structure analysis is consistent with the stepwise refinement philosophy of structured programming. Earley has implemented some of his ideas in the VERS language (14).

A purely operational approach to specifying the semantics of data structures has been proposed by Ellis (15). Ellis' work is based on the common base language interpreter proposed by Dennis (11). The common base language model is an interpreter-based operational semantic model for programming languages. Ellis uses this framework to discuss the ways in which programming languages deal with data structures.

An area of research related to the semantics of data structures is the analysis of storage models for programming languages. A

programming language "XYZ" defines its view of storage in terms of the language definition for XYZ. In turn, an XYZ-machine would support storage management routines consistent with the use of storage for the execution of XYZ programs. The definition of formal storage models for programming languages is a relatively new research endeavor. An analysis of the formal storage properties of PL/I has been attempted by Henhapl (23) and Bekic and Walk (4). A comparative study involving models for PL/I, ALGOL 60, ALGOL 68 and LISP may be found in Walk (62).

4. SEMANTICS OF NAME ACCESSING

A very small number of semantic models have emphasized name accessing descriptions. Two such models are considered in this survey: namely, those proposed by Gilmore (20) and Boyle and Grau (6).

The Gilmore model is based on an abstract computer with three primitive operations: an initial load operation, a lambda removal or function application operation and a conditional operation. The initial load operation has the form "a;b", where "a" and "b" are strings. The effect of executing such an operation is the placement of the string "b" into a memory location corresponding to the string "a". The initial load operation thus has the effect of establishing naming relationships between a location string name and the contents of the location. A subsequent reference to the string "a" will denote the value contained in location a.

The lambda removal operation is quite standard. If the string "g" denotes a location containing the string "(lambda, y, (lambda, x, y(y(x))))", then "g(f)" will result in an application of the lambda removal operation, i.e., the bound variable of the contents of g will be replaced by "f" giving "(lambda, x, f(f(x)))".

The conditional operation has four arguments and is defined as follows:

$$\text{cond}(a_1, a_2, a_3, a_4) = \begin{array}{ll} a_3 & \text{if } a_1 = a_2 \\ a_4 & \text{otherwise} \end{array}$$

Higman (25) has extended some of Gilmore's original ideas and defined a Gilmore system to be one in which the following hold:

- 1) A name is a string of characters. A name may be either valid, invalid or intrinsic.
- 2) A name may possess (potentially) an intermediate value and a final value.
- 3) A result is defined to be the final value of a program, where a program is a name (character string).
- 4) A string may denote a literal and there are rules for deciding when this is the case. Literals denote themselves and therefore have immediate final values.
- 5) A definition is a subsequence of a program. The semantic effect of a definition is that a certain literal is the final value of a certain name.

- 6) If a name cannot be assigned a final value under (4) or (5), there are rules for transforming a string into its intermediate value. The result of this transformation is a sequence of names consisting of one operator and one or more operands.
- 7) There are rules for evaluating an (operator, operand(s)) grouping and obtaining a result.
- 8) If the final value of a name is not immediate, the final value is recursively defined to be the result obtained by applying the operator to the final values of its operands.
- 9) A name is valid if a final value can be derived from it. An intrinsic name has no final value but it is permitted for its use as an operator. All other names are invalid. The set of all valid names under a given set of rules constitutes a language.

To the author's knowledge no additional work has been done with Gilmore's system. His original model accommodated only functional languages, but Higman (25) did include an extension to handle imperative assignment. A more serious objection to the Gilmore system is that naming clashes were avoided by simply prohibiting their occurrence in legal programs. This is a somewhat restrictive assumption that obscures the complexity of the name accessing problem. The accommodation of naming clashes is handled directly in the Boyle and Grau model.

The Boyle and Grau algorithmic semantic model for ALGOL 60 identifiers (6) provides, via a set of algorithms, a mapping from pure ALGOL 60 programs to $ALGOL_1$ programs. In $ALGOL_1$ no identifier is ever

redeclared and therefore each identifier is unambiguously associated with its proper declaration.

A mapping, Φ , from ALGOL 60 to ALGOL_1 is defined as a composite mapping consisting of $\Phi_1 \circ \Phi_2 \circ \Phi_3 \circ \Phi_4$ where

$$\Phi_4: \text{ALGOL } 60 \rightarrow \text{ALGOL}_4,$$

$$\Phi_3: \text{ALGOL}_4 \rightarrow \text{ALGOL}_3,$$

$$\Phi_2: \text{ALGOL}_3 \rightarrow \text{ALGOL}_2,$$

$$\text{and } \Phi_1: \text{ALGOL}_2 \rightarrow \text{ALGOL}_1.$$

Mappings φ_1 , φ_2 , φ_3 and φ_4 are defined respectively to be the projections of Φ_1 , Φ_2 , Φ_3 and Φ_4 which handle the mapping of identifiers. φ_1 and φ_2 are concerned with the static (compile-time) resolution of identifiers while φ_3 and φ_4 involve the dynamic (run-time) resolution of names. A pure copy rule semantic form is used in the resolution of the locals of procedures for recursive calls.

The Boyle and Grau model is quite exact and very detailed in its handling of identifier resolution. Perhaps the main drawback of this approach is that it is too detailed and therefore complicates rather than illustrates the semantics of name accessing.

5. APPLICABILITY OF FORMAL SEMANTIC MODELS

At the beginning of this report several objectives for formal semantics approaches were listed. This section is devoted to a brief evaluation of the various semantic approaches that have been surveyed in terms of whether or not they meet those objectives.

It should be apparent that, to date, no "super" semantic model has been developed. The term "super" is used here in the sense that it satisfies all objectives for designers, implementors, users and theoretical computer scientists. It may very well be that "super" is a myth, an unobtainable goal, and our efforts would be more fruitfully rewarded if we attempted to develop models for individual rather than conglomerate groups. In any event the following critique should be made before proceeding.

The mathematical models have been quite popular with the computer scientist. This is true despite the fact that, in general, the mathematical models have not met the primary objective of the computer scientist, i.e., applicability to a large class of programming languages. The popularity of this approach with the computer scientist is probably due to the fact that the rigor of the mathematical approach has been used to support the legitimacy of theoretical computer science research. The author is of the opinion that the legitimacy of this type of research is independent of mathematical parentage and advances the proposal that mathematical rigor should be employed only as a tool and not for its name dropping effect.

Since mathematical models tend to abstract out implementation details, these models tend to support the objectives of the language designer. This approach seems well-suited to the detection of ambiguous, incompatible or contradictory language constructs.

In emphasizing implementation-dependent models, the operational approach has been quite appealing to the language implementor. The use of information structures is quite appropriate for the implementor. Although this approach has not been used widely by theoretical computer scientists, it seems that operational models do offer a tremendous advantage in that they highlight rather than abstract out the objects that the computer scientist wants to study.

No approach has yet offered the user a concise, readable and transparent description of a complicated programming language. The user has been able to derive some benefit from the axiomatic models in formalizing and proving properties about programs, but even this benefit is useful to only a small number of users.

The data structure and naming models discussed in this chapter have not been adequately tested as to their applicability to a wide class of problems. Both Rosenberg and Earley have achieved limited success in dealing with implementational questions however.

6. BIBLIOGRAPHY

1. Bakker, J. W. de "Formal Definition of Programming Languages." Mathematical Center Tracts 16. Amsterdam: Mathematisch Centrum, 1967.
2. Bakker, J. W. de "Semantics of Programming Languages." Advances In Information Systems Science 2. Edited by J. T. Tou. New York: Plenum Press, 1969, 173-228.
3. Bekic, H.; Bjorner, D.; Henhapl, W.; Jones, C. B.; and Lucas, P. "A Formal Definition of a PL/I Subset." Technical Report TR25.139, IBM Laboratory Vienna, 1974.
4. Bekic, H. and Walk, K. "Formalization of Storage Properties," Symposium on Semantics of Algorithmic Languages. Edited by E. Engeler. Berlin: Springer-Verlag, 1961, 28-61.
5. Berry, D. M. "Introduction to Oregan." Proceedings of a Symposium on Data Structures in Programming Languages, SIGPLAN Notices, 6 (February, 1961), 171-190.
6. Boyle, J. M. and Grau, A. A. "An Algorithmic Semantics for ALGOL 60 Identifier Denotation." Journal of the ACM, 17 (April, 1970), 361-382.
7. Cadiou, J. M. and Manna, Z. "Recursive Definitions of Partial Functions and their Computations." Proceedings of an ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, 7 (January, 1972, 58-65.
8. Caracciolo, A. "On the Concept of Formal Linguistic Systems." Formal Language Description Languages for Computer Programming. Edited by T. B. Steel, Jr., Amsterdam: North-Holland Publishing Company, 1966, 37-51.
9. Caracciolo, A. and Wolkenstein, N. "On a Class of Programming Languages for Symbol Manipulation Based on Extended Markov Algorithms." Report No. 21, Pisa: Centro Studi Calcolatrici Elettroniche, 1963.
10. Church, A. "The Calculi of Lambda-Conversion." Annals of Mathematical Studies 6. Princeton, New Jersey: Princeton University Press, 1951.
11. Dennis, J. B. "On the Design and Specification of a Common Base Language." Proceedings of the Symposium on Computers and Automation. Brooklyn, New York: Polytechnic Press of the Polytechnic Institute of Brooklyn, 1971, 47-74.
12. Earley, J. "Toward an Understanding of Data Structures." Communications of the ACM, 14 (October, 1971), 617-627.

13. Earley, J. "Relational Level Data Structures in Programming Languages." Acta Informatica, 2 (1973), 293-309.
14. Earley, J. and Caizerques, P. "VERS Manual." Computer Science Department, University of California, Berkeley, 1971.
15. Ellis, D. J. "Semantics of Data Structures and References." Computation Structures Group Memo 106, Project MAC, Massachusetts Institute of Technology, 1974.
16. Fang, I. "FOLDS, A Declarative Formal Language Definition System." Computer Science Report CS-72-329, Computer Science Department, Stanford University, 1972.
17. Feldman, J. A. "A Formal Semantics for Computer Languages and its Application in a Compiler-Compiler." Communications of the ACM, 9 (January, 1966), 3-9.
18. Floyd, R. W. "Assigning Meanings to Programs." Mathematical Aspects of Computer Science 19. Edited by J. T. Schwartz. Providence, Rhode Island: American Mathematical Society, 1967, 19-32.
19. Garwick, J. V. "The Definition of Programming Languages by their Compilers." Formal Language Description Languages for Computer Programming. Edited by T. B. Steel, Jr. Amsterdam: North Holland Publishing Company, 1966, 139-147.
20. Gilmore, P. C. "An Abstract Computer with a List-Like Machine Language without a Label Operator." Computer Programming and Formal Systems. Edited by P. Braffort and D. Hirschberg. Amsterdam: North Holland Publishing Company, 1967, 71-86.
21. Goguen, J. A. and Thatcher, J. W. "Initial Algebra Semantics." Proceedings of the 15th IEEE Conference on Switching and Automata Theory, 15 (1974), 63-77.
22. Goguen, J. A.; Thatcher, J. W.; Wagner, E. G.; and Wright, J. B. "A Junction between Computer Science and Category Theory, I: Basic Concepts and Examples (Part 1)." IBM Technical Report RC4526, IBM Thomas J. Watson Research Center, September, 1973.
23. Henhapl, W. "A Storage Model Derived from Axioms." Technical Report TR25.100, IBM Laboratory, Vienna, 1969.
24. Henhapl, W. and Jones, C. B. "The Block Structure Concept and Some Possible Implementations with Proofs of Equivalence." Technical Report TR25.104, IBM Laboratory Vienna, 1970.
25. Higman, B. A Comparative Study of Programming Languages. New York: American Elsevier Publishing Company, Inc., 1967.
26. Hoare, C. A. R. "An Axiomatic Basis for Computer Programming." Communications of the ACM, 12 (October, 1969), 576-580, 583.

27. Hoare, C. A. R. "An Axiomatic Basis for Computer Programming." Communications of the ACM, 12 (October, 1969), 576-580, 583.
28. Hoare, C. A. R. and Lauer, P. E. "Consistent and Complimentary Formal Theories of the Semantics of Programming Languages." Acta Informatica, 3 (1974), 135-153.
29. Johnston, J. B. "The Contour Model of Block Structured Processes." Proceedings of a Symposium on Data Structures in Programming Languages, SIGPLAN Notices, 6 (February, 1971), 55-82.
30. Johnston, J. B. "Identifier Binding and Access in Nested Declaration Computations." Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems, 7 (1973), 306-312.
31. Johnston, J. B. "Binding and Flow of Control in Nested Declaration Computations." Unpublished lecture notes, Computer Science Department, New Mexico State University, 1973.
32. Johnston, J. B.; Berry, D. M.; and Murphy, D. P. "Expression Stack Management in Nested Declaration Computations." Unpublished paper, Computer Science Department, New Mexico State University, 1974.
33. Knuth, D. E. "Semantics of Context-Free Languages." Mathematical Systems Theory, 2 (June, 1968), 127-145.
34. Landin, P. J. "The Mechanical Evaluation of Expressions." The Computer Journal, 6 (January, 1964), 308-320.
35. Landin, P. J. "The Next 700 Programming Languages." Communications of the ACM, 9 (March, 1966), 157-166.
36. Ledgard, H. F. "A Formal System for Defining the Syntax and Semantics of Computer Languages." Technical Report MAC-TR60, Project MAC, Massachusetts Institute of Technology, 1969.
37. Ledgard, H. F. "Production Systems: Or Can We Do Better Than BNF?" Communications of the ACM, 17 (February, 1974), 94-102.
38. Lucas, P. "Two Constructive Realizations of the Block Concept and Their Equivalence." Technical Report TR25.085, IBM Laboratory Vienna, 1968.
39. Lucas, P.; Lauer, P.; and Stigleitner, H. "Method and Notation for the Formal Definition of Programming Languages." Technical Report TR25.087, IBM Laboratory Vienna, 1968.
40. Lucas, P. and Walk, K. "On the Formal Description of PL/I." Annual Review in Automatic Programming 6, Part 3 (1969), 105-182.
41. McCarthy, J. "Towards a Mathematical Science of Computation." Information Processing 1962. Edited by C. M. Popplewell. Amsterdam: North-Holland Publishing Company, 1963, 21-28.

42. McCarthy, J. "A Basis for A Mathematical Theory of Computation." Computer Programming and Formal Systems. Edited by P. Braffort and D. Hirschberg. Amsterdam: North-Holland Publishing Company, 1963. 33-69.
43. McCarthy, J. "A Formal Description of a Subset of ALGOL." Formal Language Description Languages for Computer Programming. Edited by T. B. Steel, Jr. Amsterdam: North-Holland Publishing Company, 1966, 1-12.
44. Manna, Z. "Properties of Program and the First-Order Predicate Calculus." Journal of the ACM, 16 (April, 1969), 244-255.
45. Manna, Z.; Ness, S.; and Viullemine, J. "Inductive Methods for Proving Properties of Programs." Proceedings of an ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, 7 (January, 1972), 27-50.
46. Mosses, P. D. "Mathematical Semantics of ALGOL 60." Technical Monograph PRG-12, Oxford University Computing Laboratory, 1974.
47. Naur, P., et al. "Revised Report on the Algorithmic Languages ALGOL 60." Communications of the ACM, 6 (January, 1963), 1-23.
48. Rosenberg, A. L. Data Graphs and Addressing Schemes." Journal of Computer and System Sciences, 5 (June, 1971), 193-238.
49. Rosenberg, A. L. "Addressable Data Graphs." Journal of the ACM, 19 (April 1972, 309-340.
50. Rosenberg, A. L. "Transitions in Extendible Arrays." ACM Symposium on Principles of Programming Languages, 1973, 218-225.
51. Rosenberg, A. L. "Managing Storage for Extendible Arrays." Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, 6 (October, 1974), 297-301.
52. Rosenberg, A. L. "Allocating Storage for Extendible Arrays." Journal of the ACM, 21 (october, 1974), 652-670.
53. Scott, D. "An Outline of a Mathematical Theory of Computation." Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems, 4 (1970), 169-176.
54. Scott, D. and Strachey, C. "Toward a Mathematical Semantics for Computer Languages." Proceedings of the Symposium on Computers and Automata. Brooklyn, New York: Polytechnic Institute of Brooklyn, 1971, 19-46.
55. Steel, T. B., Jr. "A Formalization of Semantics of Programming Language Description." Formal Language Description Languages for Computer Programming. Edited by T. B. Steel, Jr. Amsterdam: North-Holland Publishing Company, 1966, 25-36.

56. Strachey, C. "Towards a Formal Semantics." Formal Language Description Languages for Computer Programming. Edited by T. B. Steel, Jr. Amsterdam: North-Holland Publishing Company, 1966, 198-220.
57. Strachey, C. "The Varieties of Programming Languages." Technical Monograph PRG-10, Oxford University Computing Laboratory, 1973.
58. Tarski, A. "A Lattice-Theoretical Fixpoint Theorem and Its Applications." Pacific Journal of Mathematics, 15 (May 1955), 285-309.
59. Tennent, R. D. "Mathematical Semantics and Design of Programming Languages." Technical Report No. CSRG-39, Computer Systems Research Group, University of Toronto, 1973.
60. van Wijngaarden, A. "Recursive Definition of Syntax and Semantics." Formal Language Description Languages for Computer Programming. Edited by T. B. Steel, Jr. Amsterdam: North-Holland Publishing Company, 1966, 13-24.
61. van Wijngaarden, A.; Maulloux, B. J.; Peck, J. E. L.; and Koster, C. H. A. "Report on the Algorithmic Language ALGOL 68." Numerische Mathematik 14. New York: Springer-Verlag, 1969, 79-218.
62. Walk, K. "Modeling of Storage Properties of Higher-Level Languages." Proceedings of a Symposium on Data Structures in Programming Languages, SIGPLAN Notices, 6 (February, 1971), 146-170.
63. Walk, K.; Alber, K.; Bandat, K.; Bekic, H.; Chroust, G.; Kudielka, V.; Oliva, P.; and Zeisel, G. "Abstract Syntax and Interpretation of PL/I." Technical Report TR25.082, IBM Laboratory Vienna, 1968.
64. Wegner, P. "Programming Language Semantics." Formal Semantics of Programming Languages. Edited by R. Rustin. Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1970, 149-248.
65. Wegner, P. "The Vienna Definition Language." ACM Computing Surveys, 4 (March, 1972), 5-63.
66. Wilner, W. T. "Formal Semantic Definition Using Synthesized and Inherited Attributes." Formal Semantics of Programming Languages. Edited by R. Rustin. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1970, 25-40.
67. Wirth, N. and Weber, H. "EULER: A Generalization of ALGOL, and its Formal Definition: Part I." Communications of the ACM, 9 (January, 1966), 13-23, 25.
68. Wirth, N. and Weber, H. "EULER: A Generalization of ALGOL, and its Formal Definition: Part II." Communications of the ACM, 9 (January 1966), 89-99.