CS 101
A. I. 60

THE FORMAL THEORETIC ANALYSIS OF STRONG EQUIVALENCE FOR ELEMENTAL PROGRAMS

BY

DONALD M. KAPLAN

TECHNICAL REPORT NO. CS 101
JUNE 12, 1968

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

270

STANFORD ARTIFICIAL INTELLIGENCE REPORT
MEMO A.I. 60

June 12, 1968

CS 101

THE FORMAL THEORETIC ANALYSIS OF STRONG EQUIVALENCE FOR ELEMENTAL PROGRAMS

By

Donald M. Kaplan

ABSTRACT: The syntax and semantics is given for elemental programs, and the strong equivalence of these simple ALGOL-like flowcharts is shown to be undecidable. A formal theory is introduced for deriving statements of strong equivalence, and the completeness of this theory is obtained for various sub-cases. Several applications of the theory are discussed. Using a regular expression representation for elemental programs and an unorthodox semantics for these expressions, several strong equivalence detecting procedures are developed. This work was completed in essentially its present form in March, 1968.

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

ACKNOWLEDGEMENTS

I would like to express deepest thanks to my thesis advisor, Professor John McCarthy, for his assistance and good counsel, and for his being to me a constant source of inspiration. I am also grateful to Professors George Forsythe, David Gries and William Miller for their efforts in reading the thesis, and for their many helpful suggestions. In addition, I would like to thank Professor Dana Scott for his comments and advice, and Mr. Takayasu Ito for his participation in our many invigorating discussions.

Special thanks to Mrs. Judy Muller for her unfailing perserverance and excellence in the typing of this manuscript. And to Mrs. Dorothy McGrath and the staff of the Hansen Laboratories Drafting Room, my thanks for their highly professional work on the illustrations.

But above all, I am grateful for the encouragement and understanding of my wife Barbara, to whom this work is dedicated.

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense, (SD-183).

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

NOTATIONAL CONVENTIONS

The standard notational conventions of set theory as given by Halmos [14], for example, are assumed throughout the text. Aside from the notations introduced below, all others are introduced when they are used.

Sequences or n-tuples

A sequence of n items, i.e., an n-tuple will be represented as $\langle x_0, x_1, \ldots, x_{n-1} \rangle$ for $n < 1$, and as simply x_0 for $n = 1$.

Any list, set, sequence or whatever, containing the items $x_0, x_1, \ldots, x_{n-1}$, is the null list, set, sequence or whatever if $n = 0$.

The Natural Numbers

The set of natural numbers $\{0, 1, 2, \ldots\}$, as defined set theoretically by Halmos [14], is denoted by ω. Since, according to this definition, ω is well ordered by the "$\in$" or set membership relation, the notation $x < \omega$ will be used in lieu of $x \in \omega$.

The set $\omega - \{0\}$, i.e., $\{1, 2, \ldots\}$, is denoted by ω^-. Then, $x < \omega^-$ means $1 < x < \omega$.

CHAPTER 1

INTRODUCTION

The Need for a Theory

Computer programming is not yet a science, but rather still somewhat of an art. A great deal of ingenuity and heuristic methodology is required when we attempt to debug a program; convince ourselves that a computation will terminate; show two programs are equivalent; or certify that a compiler is correct. Although the "art" can never be fully removed from many of these endeavors, if more "science" could be employed, then at least our attack on these problems would benefit from the resultant organization and sophistication, and perhaps in some cases even be made susceptible to mechanized implementation. One way of injecting "science" into our approach would be to formulate a "theory of computation". Ideally, what we want is a theory of computation rich enough to admit interesting statements about programs, computations and compilers, and powerful enough to admit proof of the correctness of these statements. The theory we consider here falls short of this ideal in the sense that we treat only one small area of concern, namely, the equivalence of programs.

There is a parallel between computer programs and sentences in a formal theory of mathematical logic. Programs take on meaning only when the machine on which they are executed is specified; sentences in a formal theory take on meaning only when the mathematical system in which they are interpreted is specified. Thus, a formal theory of computation would seem to have intuitive appeal, and it is just such an approach that we take here.

A Theory of Strong Equivalence

The sort of theory that concerns us here is one whose well-formed formulas (wffs) express the strong equivalence (i.e., equivalence for all interpretations) of two programs from a certain restricted class of programs. The notion of such a formal theory has been explored by Ianov [16], but his results are abstract in nature and mirror only the coarser features of programs as we know them. The theory developed here differs from Ianov's in that the sort of programs we consider provide a far more detailed prescription for computations and in fact are ALGOL-like in structure and behavior (i.e., consist of assignment statements and conditional branches). A suggestive analogy is that our theory is to Ianov's, as the first order predicate calculus is to the propositional calculus. In fact, this endeavor constitutes a new and we believe necessary step in the formalization and detection of the strong equivalence of ALGOL-like programs.

Elemental Programs and Computing Structures

The literature abounds in different formal representations of ALGOL-like programs in the context of theoretical analyses. These vary from the complicated efforts of Ianov [16], Ershov [9] and Narasimhan [34] to the more succinct approaches of Luckham and Park [14], Paterson [36], Cooper [5] and Glushkov [13]. Nevertheless, each of these suffers some difficulty if we take as our objective a representation that is sufficiently ALGOL-like and yet amenable to formal treatment.

For our formal theory, we consider the class of elemental programs. These are multi-entrance, multi-exit flowcharts made up of (i) two-way conditional branches on the truth-value of quantifier-free formulas (qffs)

of the first order predicate calculus with equality, and (ii) operators called assignment schemata which assign the values of a set of terms to a set of distinct variables. This representation as explicated fully in Chapter 3 avoids the cumbersome complexity of definition given by Ershov [9] and Narasimhan [34] for their schemes, and at the same time alleviates the unnecessary deficiencies in expression found in the other representations mentioned above. In addition, by making use of the formal entities of the predicate calculus, we gain access to the abundance of results already known for this formalism.

Quite recently, and independently of this author, Engeler [8] and Manna [32] have introduced representations of programs which are very similar to the elemental programs considered here. However, both of these authors study the termination of program execution not the strong equivalence of programs.

The semantics of an elemental program is defined with respect to a mathematical system, called a computing structure, of the sort used to provide interpretation for formulas of the predicate calculus. In Chapter 2, we define such structures precisely and indicate how various bases of computation can be expressed as computing structures. In Chapter 3, we define just how computing structures are utilized to give the semantics of elemental programs.

Also introduced in Chapter 3 is the notion of subscripted variable. There we define a new data structure called a hierarchial state and show how such a structure can be accessed by a subscripted variable to produce a value.

The Strong Equivalence Decision Problem

In Chapter 4, we introduce the wffs of our theory and define the concepts of equivalence and strong equivalence in terms of the validity and general validity of these wffs. Luckham and Park [24], Kaluzhnin [19], and Paterson [36] define these notions similarly.

In Chapter 5, we examine in some detail the question of effective decidability of strong equivalence. Very recently, and independently of this author, Luckham, Park and Paterson [25,36] have considered this problem in some detail for a sub-class of the class of elemental programs. However, we obtain our basic undecidability result in Chapter 5 by utilizing a related result for partial recursive functions, whereas Luckham, Park and Paterson utilize certain results for Turing machines and two-headed automata. This appeal to recursive function theory makes our proof of undecidability brief and easy to follow.

As preface to these results, we prove the universality of elemental programs. Ershov [9] shows in a roughly sketched form how to compute all partial recursive functions in his formalism, but he fails to explicate the details. We give a new scheme which generates an elemental program for evaluating any partial recursive function at arbitrary arguments; the generating scheme utilizes the variables to simulate a first-in-last-out stack when the generated elemental program is executed.

In contrast to the pessimistic general undecidability results, certain sub-cases of the decision problem are found in Chapter 5 to yield a favorable solution. We first show that strong equivalence is decidable for the sub-class of elemental programs in which no function letters appear. The same result is obtained for the sub-class of elemental programs in which

no qffs appear. We also show that under conditions that yield decidability of the logical validity for qffs in the predicate calculus, we obtain decidability of strong equivalence for three further sub-classes of elemental programs: (i) the sub-class whose algorithms have no loops, (ii) the sub-class whose elemental programs contain no operators, and (iii) the sub-class whose elemental programs always terminate, i.e., terminate in all computing structures. As mentioned above, Paterson [36] has considered similar questions, but except for the case of always terminating elemental programs, our results were obtained independently.

Syntactic and Semantic Properties

In Chapter 6, we consider various syntactic and semantic preliminaries to the introduction of an inferential system of axioms and rules of inference for the formal theory of strong equivalence.

First, we define the notion of forward substitution of assignment schemata into other assignment schemata and into qffs. These simple syntactic operations, here examined in detail apparently for the first time, reveal the basic semantic interaction between operators and between operators and qffs.

To carry out derivations in the formal theory from hypotheses, we need the notion of instantiation of wffs. Thus, from a general statement of equivalence given by some wff, we want to produce when needed in a derivation, any relevant instance of that equivalence as given by some new wff. We give a powerful theorem which prescribes a sufficient condition for an instance of a wff to be valid when the wff itself is.

We then turn our attention to the standard matters of composition, decomposition and replacement of elemental programs. Here, we discuss these operations with respect to the graph theoretic properties of elemental programs, as do Ershov [9], Narasimhan [34] and Kaluzhnin [19].

The Inferential System

The wffs of our formal theory having been defined and studied, we introduce in Chapter 7 the inferential system of the formal theory. We discuss the usual notions of derivability, completeness and extended completeness and follow Feferman [10] and Mendelson [31] in these matters.

From the proof of undecidability, we obtain the further result that no axiomatic complete theory of strong equivalence exists. However, we proceed to specify an inferential system of fifteen axioms and five rules of inference. The first seven axioms characterize the properties of qffs; the next four, the properties of assignment schemata; and the last four, some of the graph theoretic properties of elemental programs. The first two rules characterize strong equivalence as an equivalence relation in the ordinary sense; the third rule permits instantiation of wffs; the fourth rule provides a bridge between strongly equivalent assignment schemata and qffs expressing the equality of terms; and the fifth rule permits reformulation of an elemental program in "iterative" form into "recursive" or "closed" form, but this rule is not effectively applicable, so that the theory is not axiomatic.

This inferential system is apparently the first such for deriving statements of strong equivalence between programs as rich in structure as the elemental programs considered here. Earlier efforts include McCarthy's axiomatization of the equivalence of conditional expressions [30]; Ianov's

already discussed results [16]; and this author's proof of completeness [21] of an axiomatization of the "assign" and "contents" functions first given by McCarthy [28]; and to some extent, these efforts relate to the current endeavor. The inferential system is shown to be sound in the sense that all derivable wffs are generally valid, i.e., express the strong equivalence of two elemental programs.

Completeness Results and Applications

The overall completeness properties of our non-axiomatic theory are unknown. It is nevertheless complete or even extended complete, as we show in Chapter 8, for those sets of wffs expressing the strong equivalence of (i) a two way branch and the always true branch; (ii) two sequences of assignment schemata, or further, any two elemental programs without qffs; (iii) two elemental programs without loops; and (iv) two elemental programs which always halt, i.e., in all computing structures.

We then consider an axiomatization of the properties of assignment schemata consisting of a single assignment of a term to a variable, and conjecture that this axiomatization is complete.

To illustrate the considerable derivational power of the formal theory, we consider in Chapter 8 several applications: (i) the reorganization of a simple loop from FORTRAN form, where the body of the loop is executed at least once, to ALGOL form, where the body of the loop may possibly not be executed at all; (ii) the detection of an elemental program that always fails to halt; (iii) loop reorganization to point up and isolate possible non-halting executions; (iv) the removal from a loop of a loop-independent operation; (v) the transfer of a loop-transparent operation from before the

loop to after it; (vi) the detection of strongly equivalent always halting elemental programs; (vii) the deduction of the strong equivalence of two elemental programs from certain hypotheses on the algebraic properties of functions appearing in them, e.g., commutativity or identity.

Initial conditions and K-events

The essential motivation for this work is the study, detection and derivation of the strong equivalence of elemental programs. Because this property is in general both undecidable and unaxiomatizable, we feel there should be a basic commitment to sharpening our analytic tools as much as possible. The aim, then, is to provide an organized comprehensive method for the detection of strong equivalence, to whatever extent such is obtainable.

To this end, we turn in Chapter 3 to the notions of regular expressions and regular events, as defined by Kleene [23], and as further studied by Harrison [15], Salomaa [38], McNaughton and Yamada [31], and many others. We show how to map any elemental program into a finite automaton, and thence into a characterizing regular expression. Independently of this author, Engeler [8] and Ito [18] use a similar regular expression representation.

We then develop through a series of theorems the notion of initial condition. Thus, given any word in the regular event associated with an elemental program, we define an initial condition that holds with respect to a given interpretation (i.e., computing structure) if and only if the elemental program, when executed in that computing structure, generates the given word. We then recast anew the definition of strong equivalence in terms of a possibly infinite propositional form involving the initial

conditions on the words in the regular events associated with the elemental programs involved. We give an interesting theorem which serves to verify this recasting of the definition of strong equivalence.

All of this leads to an operative tool in the detection of strong equivalence. We show that if two elemental programs have the same regular event associated with them, then they are strongly equivalent. Since the equality of regular events is decidable (cf. Salomaa [38]), this gives us an effective handle on strong equivalence.

To sharpen this technique somewhat, we introduce the notion of K-event. This reformulation of the semantics for the regular expression associated with an elemental program (now called a K-expression) reflects the previously ignored propositional structure of those letters in the alphabet for that elemental program that are qffs. We first prove that if two elemental programs have the same K-event associated with them, then they are strongly equivalent. Equality of K-events, i.e., the K-equivalence of K-expressions, is shown to be decidable concurrently with an examination of a formal theory of K-equivalence and a proof that this theory is complete. Since equality of regular events implies equality of K-events but not vice-versa, this result therefore gives us a stronger effective handle on strong equivalence.

This last result also gives us a fresh and pellucid reformulation of the equivalence problem for abstract program schemata as studied by Ianov [16] and Rutledge [37]. This follows since if we restrict our elemental programs by permitting but a single distinct variable, we have the abstract case. In this situation, K-equivalence and strong equivalence are identical

notions. As well, Ito [18] considers the equivalence problem for a class of nondeterministic abstract program schemata and his positive solution, obtained independently of this author, implies a positive solution to the deterministic case. However, he does not consider K-events and K-equivalence, as defined here, nor the relation of these to the strong equivalence of elemental programs.

To sharpen our strong equivalence detection tools even further, we introduce the notion of shift set. This concept was first introduced by Ianov [16] and subsequently extended by Rutledge [57]. For each operator occurring in an elemental program, we can effectively specify which atomic qffs occurring in the algorithm can be affected, i.e., with regard to their truth-value, by the execution of the given operator. This allows us to refine our notion of K-equivalence and so therefore strengthen our ability to detect strong equivalence.

Concluding Remarks

The contest between strong equivalence and the theoretician is not yet resolved. The opponent has gotten in some strong blows, viz., undecidability and unaxiomatizability. But, we have countered with a powerful formal theory and potent analytic tools. There are still a great many potentially productive attacks to be considered; this endeavor, it seems, has merely scratched the surface of the strong equivalence problem. In the concluding remarks at the end of this work, we consider what some of these as yet untried attacks might be.

CHAPTER

COMPUTING STRUCTURES

At first thought it may seem somewhat odd to be discussing a prime element of the semantics of a programming language before that language itself is defined. But not so. Given a problem to solve, one naturally comes in contact first with the primitives of the situation: domain of the problem space, transformations to aid in effecting a solution, and measures to define and evaluate progress. Once these fundamental entities, which we call a semantic basis, are established, some algorithmic process can be undertaken to generate the required solution, and only then will schemes for specifying such algorithms be relevant. To specify a semantic basis, we will use a computing structure.

A computing structure is a mathematical structure comprised of a non-empty set, called the domain, and finitely many relations, functions, and designated individuals in the domain. The relations and functions are to be total, i.e., defined for all arguments.

We classify computing structures according to their structural similarity. To specify this classification we use a signature which is of the form

$$s = \langle \langle n_0, \ldots, n_{k-1} \rangle, \langle m_0, \ldots, m_{\ell-1} \rangle, p \rangle$$

where $n_0, \ldots, n_{k-1}, m_0, \ldots, n_{\ell-1} \in \omega$; k, ℓ $p \in \omega$ and where if $k=0$ or $\ell=0$, the respective members of the triple s are simply 0.

By a computing structure of signature s we mean a sequence

$$\underset{\sim}{D} = < D, R_0, \ldots, R_{\kappa-1}, F_0, \ldots, F_{\ell-1}, a_0, \ldots, a_{p-1} >$$

such that

(i) D is a non-empty possibly infinite set, the domain

(ii) $R_i \subseteq D^{n_i}$ for $i < \kappa$, the relations

(iii) $F_i : D^{m_i} \to D$ for $i < \ell$, the functions

(iv) $a_i \in D$ for $i < p$, the designated individuals

Note that the first element of sequence $\underset{\sim}{D}$, i.e., $\underset{\sim}{D}_0$ is the domain D. In the sequel, when a computing structure is not explicitly defined, we will designate its domain in this fashion. Assumed present in every structure, regardless of signature, is the relation of equality over the domain of that structure.

Examples of Computing Structures

As examples of computing structures for which there is some interest in constructing programs, we can first mention some that are algebraic mathematical structures.

(i) The Boolean algebra $< \{T, F\}, \wedge, \vee, \neg >$ with signature $< 0, <2,2,1>, 0>$ serves as the semantic basis for the logical constructs of several programming languages.

(ii) The commutative ring of complex numbers $< C, +, \times, 1>$ with signature $<0, <2,2>, 1>$ might serve as the semantic basis for a complex arithmetic programming language.

As further examples, we can cite the following non-algebraic systems.

(i) The computing structure $<2^{36}, TZE, TMI, ADD, ALS>$ with signature $< <1,1>, <2,1>, 0>$ superficially mimics part of the order code in the IBM 7090 computer. Here 2^{36} denotes the set of all 36 bit words over $\{0,1\}$, and

TZE = {00...0}

TMI = $\{b_0 b_1 \ldots b_{35} \in 2^{36} : b_0 = 1\}$

ADD : $2^{36} \times 2^{36} \to 2^{36}$ according to some convenient rule of binary addition which ignores overflow, and

ALS : $2^{36} \to 2^{36}$ so that $ALS(b_0 b_1 \ldots b_{35}) = b_1 b_2 \ldots b_{35} 0$.

Here, the mnemonics TZE, TMI, ADD and ALS serve only to indicate the contexts in which these relations and operations might be used. Thus, the addition instruction on some computer might use the ADD operation together with various data transmissions, overflow tests and so on, to carry out its action.

(ii) The computing structure $\langle \omega, TZE, ADD1, SUB1, C \rangle$ with signature $\langle 1, \langle 1,1 \rangle, 1 \rangle$ is used for computing with the natural numbers. Here,

TZE = {0}

ADD1 : $\omega \to \omega$ so that $ADD1(n) = n + 1$

SUB1 : $\omega \to \omega$ so that $SUB1(n) = n - 1$ if $n > 0$

$= 0$ if $n = 0$

This system serves as semantic basis for several of the machines studied in recursive function theory, e.g., the URM of Shepherdson and Sturgis [40] or the register machines defined by Gandy [22].

(iii) The computing structure $\langle W, TEST, TRANS \rangle$ with signature $\langle 2, 2, C \rangle$ is the basis for computation in Post tag systems (cf. Davis [7]). Here, $W = A^*$ is the set of words over some finite alphabet A , and

$TEST = \{\langle x, y \rangle : x \in A$ and $y = xz$ for some $z \in W\}$

TRANS : $W \times W \to W$ so that $TRANS(x, y) = y'x$ where $y = uy'$ for some $u \in A$.

Many Sorted Computing Structures

We should remark at this point that there are certain mathematical systems which cannot be formulated in a natural way as computing structures in the sense used above (e.g., modules, of which vector spaces are instances; cf. Feferman [10]). Since it would certain be of interest to construct programs for such systems, there is some motivation for extending the concept and definition of both signature and computing structure to accomodate them. However, in the sequel, we concern ourselves only with the sort of signatures and computing structures already introduced. Therefore, the discussion of how these concepts can be extended to generalized signatures and many-sorted computing structures is relegated to Appendix I.

Remarks:

(i) As we shall see, a computing structure constitutes the bare bones of a class of partial functions computable via programs interpreted in that structure. This viewpoint seems to be in sympathy with Scott's feeling [ ] that functions computed by various machines are "more basic" than the sets accepted by them.

(ii) The notion of semantic basis is also employed by McCarthy [30], when he defines a class of functions $\mathcal{C}[\mathcal{F}]$ computable in terms of a base set $\mathcal{F}$ of functions, relations and constants.

(iii) It is conceivable that we could specify a computing structure, undoubtedly a many-sorted one, to mirror the true complexity of the operations and tests in, say, the IBM 7090 computer. However, our ability to carry out theoretical analyses would then be hampered by cumbersome

notations and involved formal procedures. The degree to which the formulation presented here falls short of reality reflects the degree of compromise required to achieve a tractable theoretical approach. Of course, a possible alternative for the future is to design computers with elegant and eminently blemish free operational characteristics so as to facilitate the theoretical analysis of their behavior. This is obviously the theoretician, not the engineer, talking.

BLANK PAGE

CHAPTER 2

ELEMENTARY PROGRAMS: SYNTAX AND SEMANTICS

The motivation for formulating a set of rules for the prescription of algorithms is that we want to have a convenient uniform method of specifying calculations in some mathematical structure of interest. Usually, our attention is focused on a specific structure, say S-expressions, real numbers, Turing machine tapes, the natural numbers or whatever. Of course, often we may be forced to do our calculations in a structure different from the one intended, either knowingly (e.g., we decide that it is better to compute with pointer linked machine words instead of symbolic S-expressions) or unknowingly (e.g., we may think of doing real arithmetic, but truncated floating point arithmetic is substituted instead).

As indicated in Chapter 1, we will define algorithms in terms of flowcharts labelled with assignment schemata and quantifier free formulas of a first order predicate calculus with equality (qffs). There are several reasons for pursuing a theoretical analysis of algorithms specified in this way.

(i) We can easily apply the flowchart method of prescribing algorithms to specify calculations in virtually all mathematical structures of interest. This is important if we are to study the strong equivalence problem which ranges over all structures.

(ii) Utilizing assignment schemata and qffs in a flowchart scheme in some sense provides us with maximal computing power. Thus, as we show later, in the structure $\langle \omega, +1, 0 \rangle$ we can compute all partial recursive functions.

(iii) The flowchart method for prescribing computational processes has proved itself to be both natural and intuitive. The hope then is that these properties will propogate into the theoretical analysis of these processes as well.

(iv) If the results obtained here are to be useful, then the programs whose properties are analyzed should be closely related in structure and intent to actual computer programs. And in spite of their linear string representation, modern ALGOL-like programs are indeed basically flowcharts of assignments and branches. In fact, variations in a program caused by squashing its flowchart into a linear string in different ways are not really of interest.

We will define for each signature s, a formal language, L_s, of <u>elemental programs</u> (or <u>E-programs</u> as we shall usually term them) for specifying algorithms that utilize computing structures of that signature. Strong equivalence, which we loosely said in Chapter 1 was "equivalence for all interpretations", will refer, for each signature s, to the equivalence of E-programs in L_s for all computing structures of that signature.

<u>The Syntax of E-programs</u>

For each signature s, we define the formal language L_s to be the set of all E-programs $\mathfrak{A} = \langle X, \Gamma, \mathcal{L} \rangle$ where X is a finite non-empty set of nodes; Γ is a partial map over X such that for each $x \in X$ where Γ is defined, Γx is either y or the ordered pair $\langle y, z \rangle$ for some $y, z \in X$; and $\mathcal{L} : X \to \mathcal{A} \cup \mathcal{Q} \cup \mathcal{B} \cup \mathcal{E}$ is a <u>consistent labelling</u> of the nodes in X with <u>operators</u> from $\mathcal{A}$, <u>discriminators</u> from $\mathcal{Q}$, <u>initiators</u> from $\mathcal{B} = \{b_0, b_1, \ldots\}$ and <u>terminators</u> from $\mathcal{E} = \{e_0, e_1, \ldots\}$. (Note: we write $[x]$ instead of $\mathcal{L}(x)$ for the label of node $x \in X$.) We define $X(\mathcal{A}) = \{x : x \in X \ \& \ [x] \in \mathcal{A}\}$, i.e., $X(\mathcal{A})$ is the sub-set of nodes labelled with an operator. Similarly for $X(\mathcal{Q})$, $X(\mathcal{B})$ and $X(\mathcal{E})$.

The labelling $\mathcal{L}$ of an E-program $\mathcal{M} = \langle X, \Gamma, \ldots \rangle$ being consistent means

(i) $[x] \in \mathcal{A} \Rightarrow \Gamma x = y$ for some $y \in X$.

(ii) $[x] \in \mathcal{Q} \Rightarrow \Gamma x = \langle y, z \rangle$ for some $y, z \in X$.

(iii) $[x] \in \mathcal{B} \Rightarrow \Gamma x = y$ for some $y \in X$, and for all $z \in X$, x is not reachable via Γ from z, i.e., is not in the transitive closure of Γ (cf. Berge [1] for a discussion of reachability).

(iv) $[x] \in \mathcal{E} \Rightarrow \Gamma$ is not defined at x.

(v) If $\mathcal{L}$ stipulates that $m < \omega$ nodes are labelled with initiators and $n < \omega$ nodes with terminators, then these must be $b_0, b_1, \ldots, b_{m-1}$ and $e_0, e_1, \ldots, e_{n-1}$ respectively. In this case, $\mathcal{M}$ is called a type $\langle m, n \rangle$ algorithm.

Remarks:

(i) In the set $\mathcal{B} = \{b_0, b_1, \ldots\}$, b_2, for example, stands for itself, i.e., for the letter "b" subscripted by a "2". Thus, $\mathcal{B}$, and $\mathcal{A}$, $\mathcal{Q}$ and $\mathcal{E}$ as well, are sets of formal constituents. However, we often make use of the fact that subscripts are well understood designations for natural numbers; so we may often say "the i-th variable" for v_i or "b_k where $k < \omega$". No confusion should result from this double usage.

(ii) Here the terminology and methodology are somewhat controversial. When we say L_s is a formal language, by "formal" we mean "purely syntactically defined", and this agrees with modern usage in most cases. One advantage of a formal system is that manipulation of formulas and expressions of the system can be expressed in a precise finitistic way involving only syntax; if the syntax is arranged properly, the effectiveness of various notions concerning the system becomes self-evident. Carnap [3] gave formal methods, as such, a big boost

much to the consternation of certain other mathematical logicians. Curry [6] vigorously remonstrates against Carnap's innovations and he chides the "syntax addicts" and others to "sign a declaration of independence" from purely syntactical methods. In fact, purely formal methodology (see, e.g., Karp [22]) can easily lead to intractable situations. For this author, Curry's advice is well-taken, and we adopt a somewhat middle course, making formal those parts of the endeavor that will profit from formalization (i.e., the sets $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{E}$) and leaving informal those parts that would suffer from it (i.e., the organization of an E-program as a graph defined in a set theoretic manner). In this light, our designation of L_s as a "formal" language is, in part, a misnomer (one, nevertheless, we shall continue to apply).

To define the sets $\mathcal{A}$ and $\mathcal{T}$ we first introduce a first order predicate calculus with equality, PC_s. Note the dependence of this calculus on the signature s. To see the connection between the definitions which follow and the computing structures for which the algorithms in L_s are defined, recall that a representative computing structure of signature

$$s = \langle \langle n_0, \ldots, n_{k-1} \rangle, \langle m_0, \ldots, m_{\ell-1} \rangle, p \rangle$$

is

$$\underset{\sim}{D} = \langle D, R_0, \ldots, R_{k-1}, F_0, \ldots, F_{\ell-1}, a_0, \ldots, a_{p-1} \rangle .$$

The countably-many symbols of PC_s are: the variables $v_0, v_1, \ldots$, the constants $k_0, \ldots, k_{p-1}$, the function letters $f_0, \ldots, f_{\ell-1}$, the relation letters $r_0, \ldots, r_{k-1}$, and the symbols "(", ")", "=", "$\supset$", "$\sim$" and "," .

We now define the <u>terms</u> of PC_s.

(i) The variables $v_0, v_1, \ldots$ are terms.

(ii) The constants $k_0, \ldots, k_{p-1}$ are terms.

(iii) For any $i < \ell$, if $\tau_0, \ldots, \tau_{m_i-1}$ are terms then $f_i(\tau_0, \ldots, \tau_{m_i-1})$ is a term.

(iv) An expression is a term only if it can be shown to be so through a finite number of applications of (i), (ii) and (iii) above. (Note: hereafter, this last proviso will be referred to only as the "extremal clause".)

Then the quantifier-free formulas (qffs) of PC_s are defined as follows. Of course, the set $\mathcal{Q}$ of discriminators is just the set of qffs of PC_s.

(i) For any $i < k$, if $\tau_0, \ldots, \tau_{n_i-1}$ are terms then $r_i(\tau_0, \ldots, \tau_{n_i-1})$ is a qff.

(ii) If τ and σ are terms, then $(\tau = \sigma)$ is a qff.

(iii) If p and q are qffs, then $(\sim p)$ and $(p \supset q)$ are qffs.

(iv) Extremal clause.

We can define other propositional connectives as follows.

(i) $(p \wedge q)$ will stand for $\sim(p \supset \sim q)$

(ii) $(p \vee q)$ will stand for $((\sim p) \supset q)$

(iii) $(p \equiv q)$ will stand for $((p \supset q) \wedge (q \supset p))$

The conjunction $(\tau_0 = \sigma_0) \wedge (\tau_1 = \sigma_1) \wedge \ldots \wedge (\tau_{n-1} = \sigma_{n-1})$, where τ_i and σ_i, $i < n < \omega^-$, are terms, will be abbreviated in the sequel as $(\tau_i = \sigma_i)_{i<n}$.

As the operators in $\mathcal{A}$, we take assignment schemata of the form

$$u_0 := \tau_0 \;\&\; u_1 := \tau_1 \;\&\ldots\&\; u_{n-1} := \tau_{n-1}$$

where $n < \omega^-$, and where if $n = 1$ we have simply $u_0 := \tau_0$. Here $\tau_0, \ldots, \tau_{n-1}$ are terms of PC_s and $u_0, \ldots, u_{n-1}$ are distinct variables of PC_s. (The intent here is that the terms $\tau_0, \ldots, \tau_{n-1}$, are computed before any assignments are done.) We will abbreviate expressions of the above form as $(u_i := \tau_i)_{i<n}$ and refer to the u_i, $i<n$, as the assigned variables.

In Figure 1 is an example of an E-program shown in flowchart form. We will call this form of an E-program its diagrammatic representation (dr). Because the dr of an E-program is such a convenient representation, we will in the sequel define E-programs in terms of their dr's rather than give the actual set-theoretic definition. We define the dr of an E-program $\mathcal{M} = \langle X, \Gamma, \mathcal{L} \rangle$ as follows.

(i) The dr of $x \in X(\mathcal{B}) \cup X(\mathcal{E})$ is a circle enclosing [x]; of $x \in X(\mathcal{A})$, a rectangle enclosing [x]; and of $x \in X(\mathcal{D})$, an oval enclosing [x].

(ii) For all $x \in X$ such that $\Gamma x = y$ for some $y \in X$, the dr of the partial map Γ acting on x consists of an arrow from the dr of x to the dr of y. For all $x \in X$ such that $\Gamma x = \langle y, z \rangle$ for some $y, z \in X$, the dr of the partial map Γ acting on x consists of two arrows from the dr of x, one to the dr of y labelled with the letter T and the other to the dr of z labelled with the letter F. When it is unambiguous, the T and F labels will be dropped, and the convention adopted that the arrows from the dr of x will point down and the leftmost one will be the T arrow.

(iii) Then the dr of $\mathcal{M} = \langle X, \Gamma, \mathcal{L} \rangle$ consists of the drs of the nodes in X, joined by the dr of the partial map Γ acting on the nodes in X.

To give some indication of the generality of L_s, and to further illustrate the idea of a diagrammatic representation for E-programs, we have constructed the rather artificial example of Figure 2.

Remarks:

(i) In spite of the fact that we will not have occasion to define an E-program in set-theoretic terms, but rather will always employ a dr of that E-program, we nevertheless will retain the set-theoretic definition and will

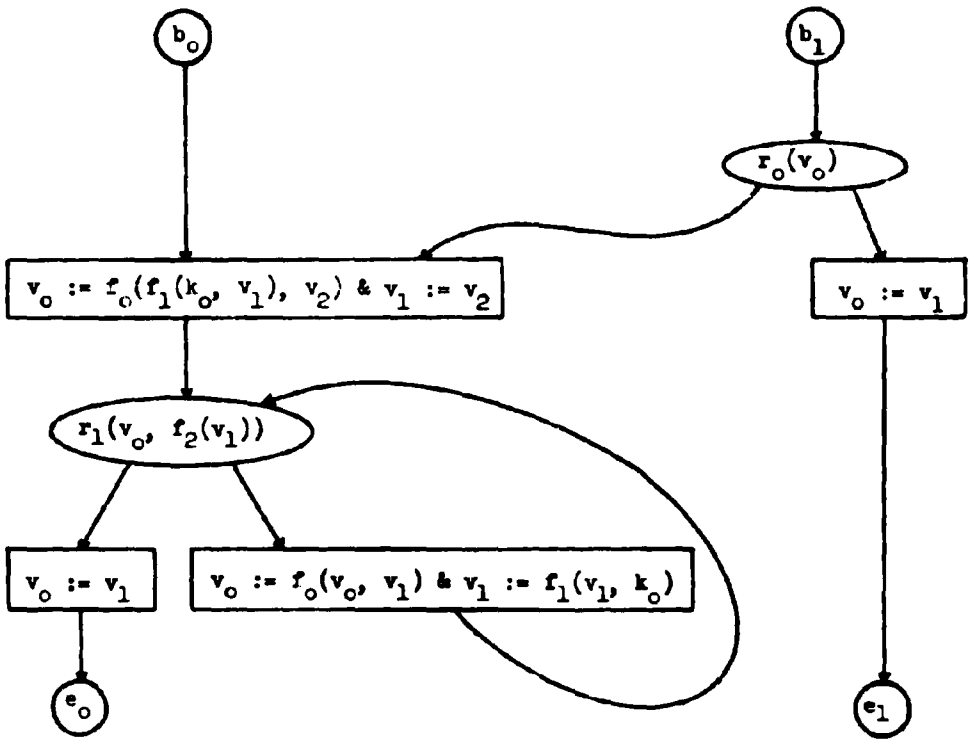


Figure 1

A type $\langle 2, 2\rangle$ E-program in L_s where $s = \langle\langle 1, 2, 1\rangle, \langle 2, 2, 1\rangle, 2\rangle$.
In this example f_2, r_2 and k_1, which are permissable, do not appear.

regard the dr merely as an aid to understanding. We do this because the syntactic manipulations required in applying a formal theory are much easier to describe and effectively carry out in set-theoretic terms rather than in terms of boxes, arrows, ovals, etc.

(ii) In Figure 2, we see that the definition of E-program allows totally isolated components and other components not reachable from any node in $X(\mathfrak{B})$. As well, certain loops once entered can never be left. Intuitively speaking, inclusion of these constructs would usually be classed as poor or improper programming. However, by admitting them here, we are facing up to the fact that such constructs do appear with unfortunate regularity in actual programs, and therefore should be subject to analysis in any theory of computation with pragmatic goals.

(iii) The E-programs of L_s can have many entrances and exits. Thus, if we want to study or transform not a whole E-program, but only some isolated fragment that may be entered and left in more than one way, we can do so by extracting that fragment as an E-program with many entrances and exits.

The Semantics of E-programs

To effect a computation, we need a type $\langle m, n\rangle$ E-program, $\mathfrak{A} = \langle X, \Gamma, \mathcal{L}\rangle$ in L_s, a natural number $i < m$, a computing structure $\underset{\sim}{D}$ of signature s, and a <u>state</u> $\xi : \omega \to \underset{\sim}{D}_0$. The E-program tells what to do; the number i tells where to start (i.e., at which initiator); the computing structure supplies the primitives for doing it; and the state acts first as input, then as "memory" during execution, and finally as output.

We first give the semantics for PC_s. It is assumed throughout this section that the signature s is $s = \langle\langle n_0, \ldots, n_{k-1}\rangle, \langle m_0, \ldots, m_{\ell-1}\rangle, p\rangle$

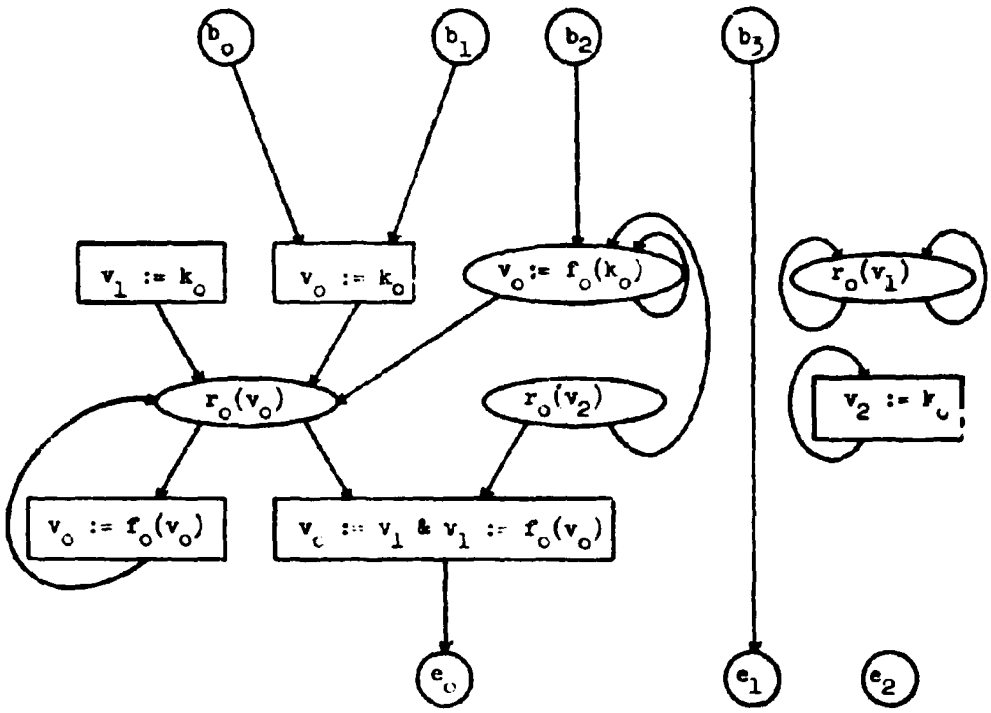


Figure 4

A type $\langle 4, 4 \rangle$ E-program in L_S where $S = \langle 1, 1, D \rangle$.

and that we are concerned with the fixed computing structure of signature σ

$$\underset{\sim}{D} = \langle D, R_0, \ldots, R_{k-1}, F_0, \ldots, F_{\ell-1}, a_0, \ldots, a_{p-1} \rangle ,$$

and the fixed state $\xi : \omega \to D$.

The <u>value</u> of a term τ with respect to $\underset{\sim}{D}$ and ξ, denoted $\tau[\underset{\sim}{D}, \xi]$, is an element of the domain and is defined recursively as follows.

(i) If τ is a variable v_i , then $\tau[\underset{\sim}{D}, \xi] = v_i[\underset{\sim}{D}, \xi] = c(i, \xi)$, where $c(i, \xi)$, read "the contents of location i in ξ", is the notation for ξ_i or $\xi(i)$ introduced by McCarthy [28].

(ii) If τ is a constant k_i, then $\tau[\underset{\sim}{D},\xi] = k_i[\underset{\sim}{D}, \xi] = a_i$.

(iii) If τ is $f_i(\tau_0, \ldots, \tau_{m_i-1})$, then $\tau[\underset{\sim}{D}, \xi] = f_i(\tau_0, \ldots, \tau_{m_i-1})[\underset{\sim}{D}, \xi] = F_i(\tau_0[\underset{\sim}{D}, \xi], \ldots, \tau_{m_i-1}[\underset{\sim}{D}, \xi])$.

We will say that qff p of PC_σ has a <u>truth-value</u> with respect to $\underset{\sim}{D}$ and ξ denoted by $p[\underset{\sim}{D},\xi]$, such that $p[\underset{\sim}{D},\xi]$ iff p is satisfied by ξ in $\underset{\sim}{D}$ in the usual sense of the predicate calculus. A recursive definition of $p[\underset{\sim}{D}, \xi]$ follows.

(i) If p is $r_i(\tau_0, \ldots, \tau_{n_i-1})$, then $p[\underset{\sim}{D},\xi] = r_i(\tau_0, \ldots, \tau_{n_i-1})[\underset{\sim}{D}, \xi] = R_i(\tau_0[\underset{\sim}{D},\xi], \ldots, \tau_{n_i-1}[\underset{\sim}{D},\xi])$.

(ii) If p is $(\tau = \sigma)$, then $p[\underset{\sim}{D},\xi] = (\tau = \sigma)[\underset{\sim}{D},\xi] = \tau[\underset{\sim}{D},\xi] = \sigma[\underset{\sim}{D}, \xi]$, i.e., $\tau[\underset{\sim}{D},\xi]$ and $\sigma[\underset{\sim}{D},\xi]$ are the same element of the domain D .

(iii) If p is $(q \supset r)$, then $p[\underset{\sim}{D},\xi] = (q \supset r)[\underset{\sim}{D},\xi] = \text{not } q[\underset{\sim}{D},\xi] \text{ or } r[\underset{\sim}{D},\xi]$.

(iv) If p is $(\sim q)$, then $p[\underset{\sim}{D},\xi] = (\sim q)[\underset{\sim}{D},\xi] = \text{not } q[\underset{\sim}{D},\xi]$.

An assignment schema $f = (u_i := \tau_i)_{i<n}$ applied to the state ξ produces a new state $f[\underset{\sim}{D}, \xi]$. Here, the values $\tau_i[\underset{\sim}{D}, \xi]$, $i<n$, are all first computed and then substituted in the state ξ at the places corresponding to the assigned variables u_i, $i<n$, so that we have for each variable v_j, $j<\omega$, of PC_s,

$$v_j[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]] = v_j[\underset{\sim}{D}, \xi] = c(i, \xi) \quad \text{if } u_i \neq v_j \text{ for all } i<n,$$
$$= \tau_i[\underset{\sim}{D}, \xi] \quad \text{if } u_i = v_j \text{ for some } i<n .$$

Alternatively, we have

$$(v_{i_k} := \tau_k)_{k<n}[\underset{\sim}{D}, \xi] = a(i_{n-1}, \tau_{n-1}[\underset{\sim}{D}, \xi], (v_{i_k} := \tau_k)_{k<n-1}[\underset{\sim}{D}, \xi])$$
$$\text{if } 1 < n < \omega$$
$$= a(i_o, \tau_o[\underset{\sim}{D}, \xi], \xi)$$
$$\text{if } n = 1 .$$

where $a(i, k, \xi)$, read "the assignment of quantity k to location i in ξ", is the notation introduced by McCarthy in [28] for the sequence obtained from ξ by replacing its i-th element by k. We may also write this as

$$(v_{i_k} := \tau_k)_{k<n}[\underset{\sim}{D}, \xi] = a(i_{n-1}, \tau_{n-1}[\underset{\sim}{D}, \xi], a(i_{n-2}, \tau_{n-2}[\underset{\sim}{D}, \xi], a(\ldots$$
$$a(i_1, \tau_1[\underset{\sim}{D}, \xi], a(i_o, \tau_o[\underset{\sim}{D}, \xi], \xi))\ldots))) .$$

Now we explain the semantics of E-programs themselves. The type $\langle m, n\rangle$ E-program $\mathfrak{M} = \langle \alpha, \Gamma, \chi\rangle$ applied to the state ξ starting at initiator b_i, $i<m$, produces

$$\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i\rangle] = E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$$

where $[x] = b_i$, and where the partial <u>execution function</u> E is defined as follows.

(i) If $[x] \in \mathcal{B}$, i.e., if $[x]$ is an initiator, then

$$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = E(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x) .$$

(ii) If $[x] \in \mathcal{A}$, i.e., if $[x]$ is an assignment schema f, then

$$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = E(\mathfrak{M}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma x).$$

(iii) If $[x] \in \mathcal{Q}$, i.e., if $[x]$ is a qff p, and if $\Gamma x = \langle y, z \rangle$ for some $y, z \in X$, then

$$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = E(\mathfrak{M}, \underset{\sim}{D}, \xi, y) \text{ if } p[\underset{\sim}{D}, \xi]$$
$$= E(\mathfrak{M}, \underset{\sim}{D}, \xi, z) \text{ otherwise.}$$

(iv) If $[x] \in \mathcal{E}$, i.e., if $[x]$ is a terminator e_j, then

$$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = \langle \xi, j \rangle .$$

Clearly, for certain $\mathfrak{M}$, $\underset{\sim}{D}$, ξ and i where $[x] = b_i$, $E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ does not terminate, and $\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is therefore indeterminate. If termination is obtained, so that $\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i \rangle] = \langle \xi', j \rangle$ for some $\xi' : \omega \to D$ and $j < n$, we say that when E-program $\mathfrak{M}$ is executed in computing structure $\underset{\sim}{D}$ with initial state ξ, starting at the i-th initiator, it halts at the j-th terminator producing the final state ξ'.

Remarks:

(i) The $c(i, \xi)$ and $a(i, k, \xi)$ notations, after first being introduced by McCarthy, have subsequently been used by him along with Painter [26, 35] and by this author, as well [20,21].

(ii) The execution function E, on reaching a node labelled with a qff p, will take the arrow in the dr labelled T (i.e., the left arrow) if p turns out true and the arrow labelled F (i.e., the right arrow) if p turns out false.

(iii) As an alternative form of assignment schema, we could take simple assignment schemata, i.e., those with only one assigned variable. This form would be somewhat more ALGOL-like, though not quite as general. In Chapter 8, we will examine briefly some of the implications of such a choice.

(iv) A modification in the definition of E-programs that would make them more ALGOL-like would be a provision for subscripted variables, i.e.,

arrays. In Appendix II, we give the details of a scheme for introducing subscripted variables. We redefine the syntax for terms, qffs and assignment schemata, and introduce a new data structure, the hierarchial state, which is used to store the arrays that are accessed by subscripted variables.

BLANK PAGE

CHAPTER 4

WELL-FORMED FORMULAS: SYNTAX AND SEMANTICS

Our principal interest is in the strong equivalence of E-programs, and so the well-formed formulas of the formal theory we develop in this and succeeding chapters will simply express for two E-programs of the same type that they are strongly equivalent.

The Syntax of Well-Formed Formulas

For each signature s, we define a formal theory $\mathcal{T}_s = \langle \mathcal{F}m_s, \mathcal{L}_s \rangle$ where $\mathcal{F}m_s$ is the set of well-formed formulas (wffs), and $\mathcal{L}_s$, as explained in Chapter 7, is an inferential system of axioms and rules of inference. The set $\mathcal{F}m_s$ is simply the set of all expressions of the form $\mathfrak{A} \simeq \mathfrak{B}$ where $\mathfrak{A}$ and $\mathfrak{B}$ are E-programs of the same type in L_s. Recall that by "of the same type", we mean with like numbers of initiators and like numbers of terminators.

The Semantics of Well-Formed Formulas

We say that a wff $\mathfrak{A} \simeq \mathfrak{B}$, where $\mathfrak{A}$ and $\mathfrak{B}$ are type $\langle m, n \rangle$ E-programs is valid in a computing structure $\underset{\sim}{D}$ and write $\vdash_{\underset{\sim}{D}} \mathfrak{A} \simeq \mathfrak{B}$ (i.e., $\mathfrak{A}$ is equivalent to $\mathfrak{B}$ in $\underset{\sim}{D}$) iff for all $i < m$, for all $\xi : \omega \to \underset{\sim}{D}_o$, we have that $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle] \cong \mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$. The notation $x \cong y$ means that either x and y are both indeterminate, or both are determinate and $x = y$.

Notice that if both $\mathfrak{A}$ and $\mathfrak{B}$ halt, producing $\langle \xi', i' \rangle$ and $\langle \xi'', i'' \rangle$ respectively, then for equivalence we require that $\langle \xi', i' \rangle = \langle \xi'', i'' \rangle$, i.e., $\xi' = \xi''$ and $i' = i''$. Thus, not only must two E-programs produce the same output state but they also must halt at the same terminator. This is a natural condition if we are to have substitution of equivalent subprograms.

We say that a wff $\mathfrak{A} \simeq \mathfrak{B}$ is generally valid and write $\models \mathfrak{A} \simeq \mathfrak{B}$ (i.e., $\mathfrak{A}$ is strongly equivalent to $\mathfrak{B}$) iff for all computing structures $\underset{\sim}{D}$, $\mathfrak{A} \simeq \mathfrak{B}$ is valid in $\underset{\sim}{D}$.

For any set of wffs $\Delta \subseteq \mathcal{F}_{\mathfrak{m}_\Sigma}$ (called either proper axioms or hypotheses), we write $\Delta \models \mathfrak{A} \simeq \mathfrak{B}$ iff for all computing structures $\underset{\sim}{D}$, if the wffs in Δ are all valid in $\underset{\sim}{D}$, then $\mathfrak{A} \simeq \mathfrak{B}$ is valid in $\underset{\sim}{D}$. In this case we say that $\mathfrak{A} \simeq \mathfrak{B}$ is a semantic consequence of Δ. Evidently, general validity is just a special case of this latter concept since $\emptyset \models \mathfrak{A} \simeq \mathfrak{B} \Leftrightarrow \models \mathfrak{A} \simeq \mathfrak{B}$, where $\emptyset$ is the empty set.

Remarks:

(i) We may use the notion of semantic consequence to aid in the study of equivalence for E-programs in particular computing structures. Thus, if the proper axioms in Δ can be shrewdly specified so that they are all valid only in the structure (or class of structures) of interest, then a wff $\mathfrak{A} \simeq \mathfrak{B}$ will be a semantic consequence of Δ just in case $\mathfrak{A} \simeq \mathfrak{B}$ is valid in that structure. When this is the case, we say we have axiomatized the properties of that structure.

(ii) It is not clear precisely what properties of structures can be axiomatized by a set of wffs of the form $\mathfrak{A} \simeq \mathfrak{B}$. It may be that more complicated statements about strong equivalence should be permitted so as to give us the axiomatizing power required to characterize certain structures, like the integers, for example. Thus, propositional statements, like $\mathfrak{A} \simeq \mathfrak{B} \wedge \mathfrak{C} \simeq \mathfrak{D} \supset \mathfrak{A} \simeq \mathfrak{R}$, or quantificational statements, like $(\exists x)(\mathfrak{A}(x) \simeq \mathfrak{B})$ may be desirable. We do not pursue this matter any further here.

CHAPTER 5

CONCERNING THE DECIDABILITY OF STRONG EQUIVALENCE

As one might expect, because of the complexity of the situation under study here, undecidability is lurking in every corner. There are two approaches both to the strong equivalence problem and to the axiomatizability problem which we discuss in Chapter 8. On the one hand we can examine these problems with respect to the whole of $\mathcal{Fm}_s$ for various signatures s ; or on the other hand we can consider various subsets of $\mathcal{Fm}_s$ for arbitrary fixed signatures s . One result obtains immediately.

Theorem 1: Strong equivalence is decidable for E-programs in which no function letters or constants occur.

Thus, $\vdash \mathfrak{A} \equiv \mathfrak{B}$? , where $\mathfrak{A} \equiv \mathfrak{B} \in \mathcal{Fm}_s$, is decidable for any signature $s = \langle \langle n_0, \ldots, n_{k-1} \rangle, 0, 0 \rangle$.

Proof: In this case, since there are no functions, the assignment schemata are relegated to merely transferring around the initial data from location to location. Thus, L_s is not too interesting or powerful a language.

Consider the type $\langle m, r \rangle$ algorithm $\mathfrak{C} \in L_s$ with $K < \omega$ nodes labelled with assignment schemata and qffs, and in which there occur $N < \omega$ distinct variables. Suppose we execute $\mathfrak{C}$ in some computing structure with some initial state ξ . Since elements in the state for variables that do not occur in $\mathfrak{C}$ are unchanged during execution, and since there can be at most N distinct values stored in the initial state ξ for the N distinct variables occurring in $\mathfrak{C}$, then there are at most N^N distinct states that can arise during the execution of $\mathfrak{C}$.

Thus, the at most N distinct initial values for the N distinct variables in $\mathcal{E}$ are shuffled around by the assignment schemata into at most N^N configurations.

Now, suppose that this execution of $\mathcal{E}$ we are considering fails to halt. Then there exists a node x of $\mathcal{E}$ such that execution passes through x more than N^N times. This is so because an infinite number of nodes are encountered during the non-halting execution, of $\mathcal{E}$, but since $\mathcal{E}$ itself has only a finite number of nodes, at least one node must be encountered infinitely often. Thus, after at most N^N+1 passes through node x, the current state at that point must repeat itself, since there are at most N^N distinct states. Of course, after a state repeats itself at a node, execution is thereafter periodic in nature with some fixed loop, which includes that node, executing repeatedly and generating a periodic sequence of states.

Since there are K nodes in $\mathcal{E}$ labelled with assignment schemata or qffs, an execution of $\mathcal{E}$ which has so far passed through at most $K \times N^N$ such nodes is guaranteed to have generated a repeated state at one of them, so that execution never halts. Since there are but a finite number of distinct paths through $\mathcal{E}$ consisting of less than $K \times N^N$ nodes, the number of distinct paths associated with halting executions, i.e., those beginning with an initiator and ending with a terminator, is therefore finite.

The E-program of Figure 3(a), for example, has but four paths through it that are associated with halting executions. Thus, we may execute the loop zero, one, two or three times and then halt, but if $r(u, w, y, x)$ is still false after three executions of the loop, then the E-program never

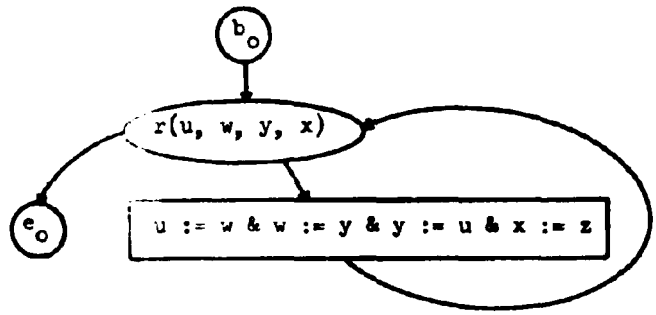


Figure 3(a)

Here u, w, x, y, z are variables and r is a relation letter.

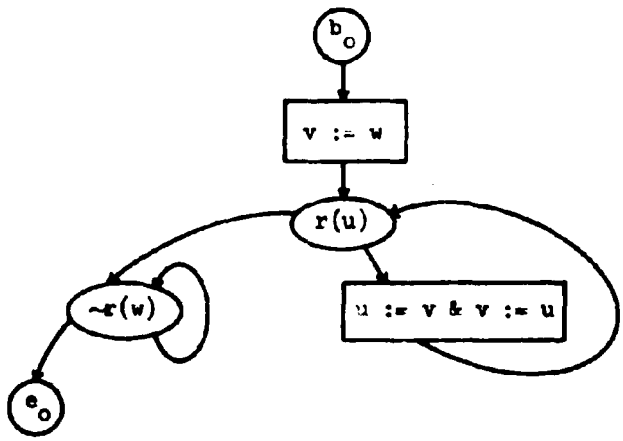


Figure 3(b)

Here u, v, w are variables and r is a relation letter.

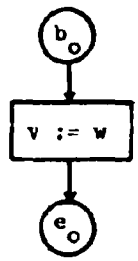


Figure 3(c)

Here v and w are variables.

halts. This is because after this point, the contents of u, w, y and x vary only periodically, and so $\neg(u, w, y, x)$ will be testing a state altready encountered.

In Chapter 9, we show how to associate with the set of all paths through $\mathfrak{C}$, that begin with an initiator and end with a terminator, a set $T_{\mathfrak{C}}$ of triples $\langle b_i, f, e_j \rangle$, where b_i and e_j are an initiator and terminator respectively, and where f is an assignment schema, called an operation in this context. We say two triples are similar iff they have the same initiators and terminators and their operations, f and g, say, are strongly equivalent (i.e., $f[\underset{\sim}{D}, \xi] = g[\underset{\sim}{D}, \xi]$ for all computing structures $\underset{\sim}{D}$ and states $\xi : \omega \to \underset{\sim}{D}_0$). No two triples in the set $T_{\mathfrak{C}}$ of triples for $\mathfrak{C}$ are similar, and furthermore, with each triple $\langle b_i, f, r_j \rangle$ we associate a qff p, called its joint initial condition, such that if $\mathfrak{C}$ is executed in $\underset{\sim}{D}$ with initial state ξ, starting at b_i, it will halt at e_j and produce the final state $f[\underset{\sim}{D}, \xi]$ iff $p[\underset{\sim}{D}, \xi]$. Then, we show (Theorem 25) that the two E-programs $\mathfrak{A}$ and $\mathfrak{B}$ are strongly equivalent iff first, for each pair of similar triples, one from $T_{\mathfrak{A}}$ and one from $T_{\mathfrak{B}}$, the corresponding joint initial conditions are logically equivalent; and second, for any triple in $T_{\mathfrak{A}}$ (or $T_{\mathfrak{B}}$) for which there is no similar triple in $T_{\mathfrak{B}}$ (or $T_{\mathfrak{A}}$), its joint initial condition must be identically false, i.e., a logical contradiction.

It is easy to show that the E-program of Figure 3(b) has but one path through it associated with a halting execution. The triple corresponding to this path is $\langle b_0, v := w, e_0 \rangle$ and the joint initial condition for this triple is $r(u) \wedge \sim r(w)$. This joint initial condition is just the

necessary and sufficient condition on the initial state for the E-program of Figure 3(b) to halt, having executed $v := w$. Notice that the E-program of Figure 3(c) also has a single path associated with halting execution. The triple here is also $\langle b_0, u := v, e_0 \rangle$, but its joint initial condition is identically true. Thus, since the joint initial conditions for the triple $\langle b_0, v := w, e_0 \rangle$ are not logically equivalent, i.e., $r(u) \wedge \sim r(w)$ is not identically true, the E-programs of Figures 3(a) and 3(b) are not strongly equivalent.

We have already shown that there are only a finite number of paths through $\mathfrak{A}$ associated with halting execution, each with less than $K \times N^N$ nodes; similarly for $\mathfrak{B}$. Thus, it suffices to consider only the set of triples and joint initial conditions for paths of length out to the respective maximums necessary for $\mathfrak{A}$ and $\mathfrak{B}$. Since the sets $T_{\mathfrak{A}}$ and $T_{\mathfrak{B}}$ of triples are then finite, we can decide the strong equivalence of $\mathfrak{A}$ and $\mathfrak{B}$ using the procedure outlined above, provided that we can decide the logical validity of qffs in PC_s and the strong equivalence of operations. But, as indicated by Church [4], since no function letters occur in the qffs of PC_s, their logical validity is decidable, and as indicated in Chapter 8 (Theorems 14, 15 and 16), the strong equivalence of operations, i.e., assignment schemata, is decidable. Thus, so is $?\vdash \mathfrak{A} = \mathfrak{B}?$. ∎

This decidability result can easily be extended to a far larger class of E-programs. Thus, strong equivalence is decidable whenever the number of paths associated with halting exscutions is finite, and the logical validity of qffs in PC_s is decidable. In Chapter 9, we return to these matters and indicate in detail the role of joint initial conditions in decision procedures for these cases.

There are two obvious applications of this extended decidability result. Thus, we have that strong equivalence is decidable for E-programs without loops and for E-programs that always halt (i.e., in all computing structures, with all initial states; cf. Figure 4.5 for an example of this case), provided, of course, the logical validity of qffs is decidable. In Chapter 6, we indicate how E-programs without loops can be put into a canonical form using the axioms and rules of our formal theory. The result for always halting E-programs is merely quoted here from the recent work of Paterson [36].

A specialization of the no-loops result, which does not depend on the decidability of logical validity for qffs, is the following

Theorem 2: Strong equivalence for E-programs consisting solely of assignment schemata, i.e., without any branching, is decidable.

If we let $L_s(\mathcal{A}) = \{\langle X, \Gamma, \Sigma\rangle \in L_S : X(\mathcal{Q}) = \emptyset\}$, and define $\mathcal{F}m_s(\mathcal{A}) = \{\mathfrak{A} = \mathfrak{B} \in \mathcal{F}m_S : \mathfrak{A}, \mathfrak{B} \in L_s(\mathcal{A})\}$, then Theorem 2 states that $\vdash \mathfrak{A} = \mathfrak{B}$, where $\mathfrak{A} = \mathfrak{B} \in \mathcal{F}m_s(\mathcal{A})$, is decidable for any arbitrary signature σ. (Incidentally, we define $L_s(\mathcal{Q})$ and $\mathcal{F}m_s(\mathcal{Q})$ in a similar fashion.) Figure 4 illustrates an E-program $\mathfrak{A} \in L_s(\mathcal{A})$.

Proof: In Chapter 8, we give a detailed proof that there is an effectively generable canonical form for E-programs in $L_s(\mathcal{A})$ (Theorems 14, 15 and 16), and this solves the decision problem for this case. We postpone this discussion, however, so that we can describe the generation of the canonical form in terms of the axioms and rules of our formal theory. ∎

The E-programs whose strong equivalence decision problem we have considered so far have been somewhat restrictive in the sense that the sorts

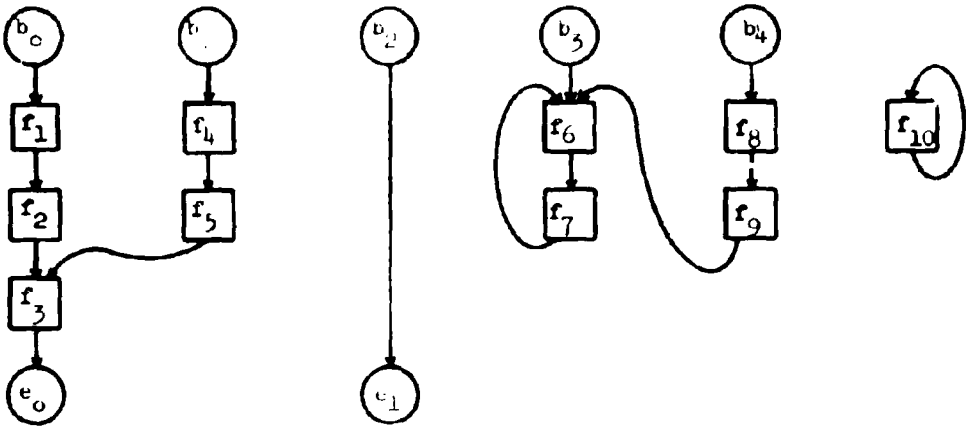


Figure 4

The E-program $\mathfrak{E} \in \mathcal{F}m_2(\mathcal{A})$. Here, $f_1, \ldots, f_{10}$ are assignment schemata.

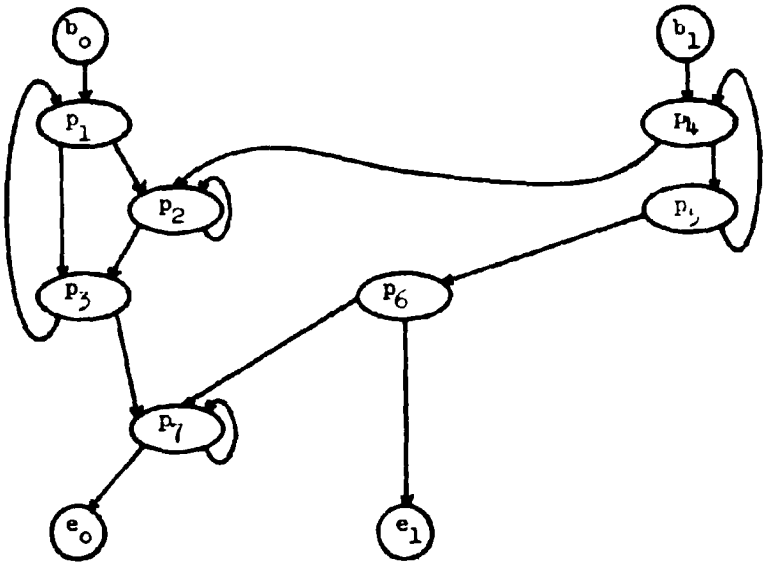


Figure 5

The E-program $\mathfrak{A} \in \mathcal{F}m_3(\mathcal{Q})$. Here $p_1, \ldots, p_7$ are qffs.

of computations they specify are not very complicated or interesting. Let us now consider the signature $s = \langle 1, 1, 1 \rangle$. Computing structures with this signature are of the form $\langle D, R_0, F_0, a_0 \rangle$ where the relation R_0 and function F_0 are monadic. If we further disregard the equality relation over the domain, then structures of this sort represent, in a sense, the barest bones with which we might want to carry out interesting meaningful computations. Now, let us consider the strong equivalence decision problem for E-programs in L_s that do not use equality. One immediate result is

Theorem 3: Strong equivalence is decidable for E-programs consisting solely of qffs build up from a single monadic relation letter, a single monadic function letter, a single constant, but without equality.

If we let $L_s^- = \{\mathfrak{A} \in L_s :$ no qff of the form $(\tau = \sigma)$ occurs in $\mathfrak{A}\}$, and define $\mathcal{F}_{m_s}^- = \{\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}_{m_s} : \mathfrak{A}, \mathfrak{B} \in L_s^-\}$, then Theorem 3. states that $?\vdash \mathfrak{A} \simeq \mathfrak{B}?$, where $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}_{m_s}^-(Q)$, is decidable for the signature $s = \langle 1, 1, 1 \rangle$.

Proof: Since E-programs here do not have any assignment schemata, we can argue, precisely as in the proof for Theorem 1, that there are but a finite number of paths through an E-program $\mathfrak{C}$ associated with halting executions of $\mathfrak{C}$. Furthermore, decidability of logical validity for qffs obtains here, so that the joint initial condition decision procedure used in Theorem 1 will suffice here as well.

To see how the joint initial conditions are obtained here, consider the E-program $\mathfrak{A}$ of Figure 5. Here in this example, it is obvious that

there are but five paths through $\mathfrak{A}$ associated with halting executions. These give rise to the triples $\langle b_0, v_0 := v_0, e_0 \rangle$, $\langle b_1, v_0 := v_0, e_0 \rangle$ and $\langle b_1, v_0 := v_0, e_1 \rangle$, where $v_0 := v_0$ is just a dummy identity operation. The joint initial conditions associated with these triples can be easily verified to be respectively

$$(p_1 \wedge \sim p_3 \wedge p_7) \vee (\sim p_1 \wedge p_2 \wedge \sim p_3 \wedge p_7) ,$$
$$(p_4 \wedge p_2 \wedge \sim p_3 \wedge p_7) \vee (\sim p_4 \wedge p_5 \wedge p_6 \wedge p_7) ,$$
$$(\sim p_4 \wedge p_5 \wedge \sim p_6) .$$

Thus, for $s = \langle 1, 1, 1 \rangle$, E-programs in L_s^- have a solvable strong equivalence decision problem when they consist solely of assignment schemata (Theorem 2) or solely of qffs (Theorem 3). Nevertheless, in general we still have the following unfortunate

Theorem 4: Strong equivalence is undecidable for E-programs built up from a single monadic function letter, a single monadic relation letter, a single constant, but without equality.

Thus, $? \vdash \mathfrak{A} \simeq \mathfrak{B} ?$, where $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}m_s^-$, is undecidable for the signature $s = \langle 1, 1, 1 \rangle$, and therefore, for any signature s' such that $\mathcal{F}m_s^- \subseteq \mathcal{F}m_{s'}^-$. So even without all the customary paraphernalia available for expressing algorithms, we are still saddled with the fact that in general the analysis of strong equivalence cannot be an effective process.

To prove Theorem 4, we will take a somewhat roundabout path and first show that in an appropriate computing structure we can compute all partial recursive functions. This result will then lead us to the proof we desire.

(i) The principal point of concern here is that for schemes that are of pragmatic interest, the strong equivalence of E-programs is undecidable. Of course, if a computing structure has a finite domain then equivalence in that structure is decidable for the same reasons the wffs of Theorem 1 are decidable. So, we might be tempted to say that, for example, equivalence of IBM 7090 programs is decidable since the domain 2^{36} is finite. There are two reasons why this sort of reasoning is not productive. First, the sort of exhaustive loop unwinding performed in the example of Figure 3(a) would take years in a domain the size of 2^{36}, thus making impractical the obvious decision procedure. Second, as suggested earlier, we may not even be aware of what computing structure our program is being executed in, thus rendering the concept of equivalence of programs somewhat impotent. The sort of statement we would be more interested in is: "No matter what computer these programs are executed on, they give the same result", i.e., a statement of strong equivalence. But Luckham, Park and Paterson [25] show that for a certain sub-class of programs, even equivalence in all computing structures with finite domains is undecidable.

(ii) Theorems 1, 2 and 3 discuss various decidability results, but actually, all of these results derive from the same set of facts. If the number of paths associated with halting executions is finite, and if we can effectively determine a bound on the length of such paths, then decidability is obtained in cases where the general validity of qffs is decidable. This is because in such situations, the sets of triples to be checked are finite and determinable, the strong equivalence of operations is decidable (this is always the case), and the logical equivalence of joint initial conditions is decidable.

(iii) The triples notation, used here merely for explanatory purposes, is not used in Chapter 9.

(iv) Within the undecidability limitations imposed by the structure of the problem being studied, we develop in succeeding chapters certain viable analytic tools for working on the strong equivalence problem for E-programs.

Partial Recursive Functions

We shall show that given a suitable computing structure, we can construct E-programs to compute all partial recursive functions. The reasons for this demonstration are twofold: first, to illustrate that E-programs as defined here are adequate in the sense that there is no function computable in a structure that we cannot specify with a suitable E-program; and second, to provide a convenient method of proof for Theorem 4 above.

Consider the computing structure $\underset{\sim}{N} = \langle \omega, F_o, a_o \rangle$ with signature $s = \langle 0, 1, 1 \rangle$, where F_o is the successor function, a_o is zero, and the equality relation over ω is included in $\underset{\sim}{N}$. (Note: we will write $x + 1$ for $F_o(x)$ and 0 for a_o.) For the purposes of defining partial recursive functions, we also consider the projection functions $U_i^n(x_o, \ldots, x_{n-1}) = x_i$ for all $n < \omega^-$, all $i < n$ and all $\langle x_o, \ldots, x_{n-1} \rangle \in \omega^n$ as initial or base functions. The partial recursive functions are obtained from zero, the successor and projection functions using three methods of combining functions. (This is all given by Mendelson in [33].)

(i) Composition: given the functions

$$g(x_o, \ldots, x_{n-1})$$
$$h_o(x_o, \ldots, x_{n-1})$$
$$h_1(x_o, \ldots, x_{n-1})$$
$$\vdots$$
$$h_{m-1}(x_o, \ldots, x_{n-1})$$

where $m < \omega$, $n < \omega$, we say that the function defined by

$$f(x_0, \ldots, x_{n-1}) = g(h_0(x_0, \ldots, x_{n-1}), \ldots, h_{m-1}(x_0, \ldots, x_{n-1}))$$

is obtained from the given functions by composition.

(ii) Primitive recursion: given the functions

$$g(x_0, \ldots, x_{n-2})$$

$$h(x_0, \ldots, x_{n-2}, x_{n-1}, x_n)$$

where $1 < n < \omega$, we say the function defined by

$$f(x_0, \ldots, x_{n-2}, 0) = g(x_0, \ldots, x_{n-2})$$

$$f(x_0, \ldots, x_{n-2}, x_{n-1}+1) = h(x_0, \ldots, x_{n-2}, x_{n-1}, f(x_0, \ldots, x_{n-2}, x_{n-1}))$$

is obtained from the given functions by primitive recursion.

(iii) The unrestricted μ-operator: given the function

$$g(x_0, \ldots, x_{n-1}, y)$$

where $n < \omega$, we say that the function defined by

$$f(x_0, \ldots, x_{n-1}) = \mu z(g(x_0, \ldots, x_{n-1}, z) = 0)$$

which we read as "the least z such that $g(x_0, \ldots, x_{n-1}, z) = 0$", is obtained from the given function by the unrestricted μ-operator. Here, $f(x_0, \ldots, x_{n-1})$ is defined for $\langle x_0, \ldots, x_{n-1} \rangle \in \omega^n$ iff for some $k < \omega$, $g(x_0 \ldots, x_{n-1}, z) = 0$ and for all $z < k$, $g(x_0, \ldots, x_{n-1}, z)$ exists and is not zero; and when such is the case, $f(x_0, \ldots, x_{n-1})$ then has the value k.

We will now describe a scheme such that for any partial recursive function and suitable arguments, we can construct an E-program in L_s which when executed in $\underset{\sim}{N}$ computes the value of that function at those arguments. Suppose that the function $f(x_0, \ldots, x_{n-1})$ is defined by a form $\mathcal{E}_f$ as given by the initial functions and (i), (ii) and (iii) above, and that $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$

where $\underset{\sim}{d} \in \omega^n$ is a set of arguments at which $f(x_o, \ldots, x_{n-1})$ is to be evaluated. Then the E-program $\mathfrak{R}(\xi_f, \underset{\sim}{d})$, produced by the generating function $\mathfrak{R}$ using the form ξ_f and arguments $\underset{\sim}{d}$, when executed in $\underset{\sim}{N}$ with any initial state, computes $f(\underset{\sim}{d})$. The E-program $\mathfrak{R}(\xi_f, \underset{\sim}{d})$ is illustrated in Figure 6. Here, $\mathfrak{T}$ is another E-program generating function; the composition of E-programs, indicated schematically in Figure 6, is discussed in Chapter 6.

The E-program $\mathfrak{T}(p, \xi_f)$ is generated in a recursive fashion according to the structure of ξ_f. The variables are utilized to simulate a stack as $\mathfrak{T}(p, \xi_f)$ proceeds, the first argument of $\mathfrak{T}$ acting as a stack pointer during construction of the E-program. When constructing $\mathfrak{T}(p, \xi_f)$, $p < \omega$, we assume that variables $v_{p+1}, \ldots, v_{p+n}$ will contain the arguments at which f is to be evaluated, and we arrange for the value of f at these arguments to be returned in v_p.

Consider first $\mathfrak{T}(p, \xi_f)$ for the initial functions. These definitions are given in Figure 7. Figures 8, 9 and 10 show constructions for composition, primitive recursion and the unrestricted μ-operator. Also in the illustrations are representations of the run-time stack showing how the variables are assigned during E-program construction by $\mathfrak{T}(p, \xi_f)$. On the basis of these constructions, we have the following

Theorem 5: For all partial recursive functions $f(x_o, \ldots, x_{n-1})$ with form ξ_f and arguments $\underset{\sim}{d} = \langle d_o, \ldots, d_{n-1} \rangle$, and then for all initial states $\xi : \omega \to \omega$, $\mathfrak{R}(\xi_f, \underset{\sim}{d})[\underset{\sim}{N}, \langle \xi, \mathcal{O} \rangle]$ is determinate and has a value $\langle \xi', \mathcal{O} \rangle$ for some $\xi' : \omega \to \omega$ iff $f(d_o, \ldots, d_{n-1})$ exists. Furthermore, in case $f(d_o, \ldots, d_{n-1})$ exists, then $f(d_o, \ldots, d_{n-1}) = c(3, \xi')$.

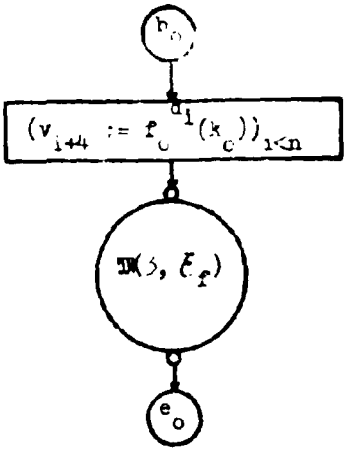
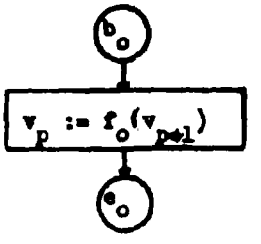


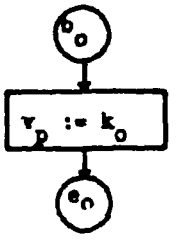
Figure 6

Here the variables $v_4, \ldots, v_{n+3}$ are loaded with the arguments $d_0, \ldots, d_{n-1}$ The variable v_3 is used to return the final value. The reason for not using v_0, v_1, or v_2 will become apparent when we consider the proof of Theorem 4.

$\mathfrak{M}(p, "x_0+1")$ is



$\mathfrak{M}(p, "0")$ is



$\mathfrak{M}(p, "U_i^n(x_0, \ldots, x_{n-1})")$ is

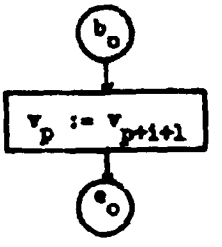


Figure 7

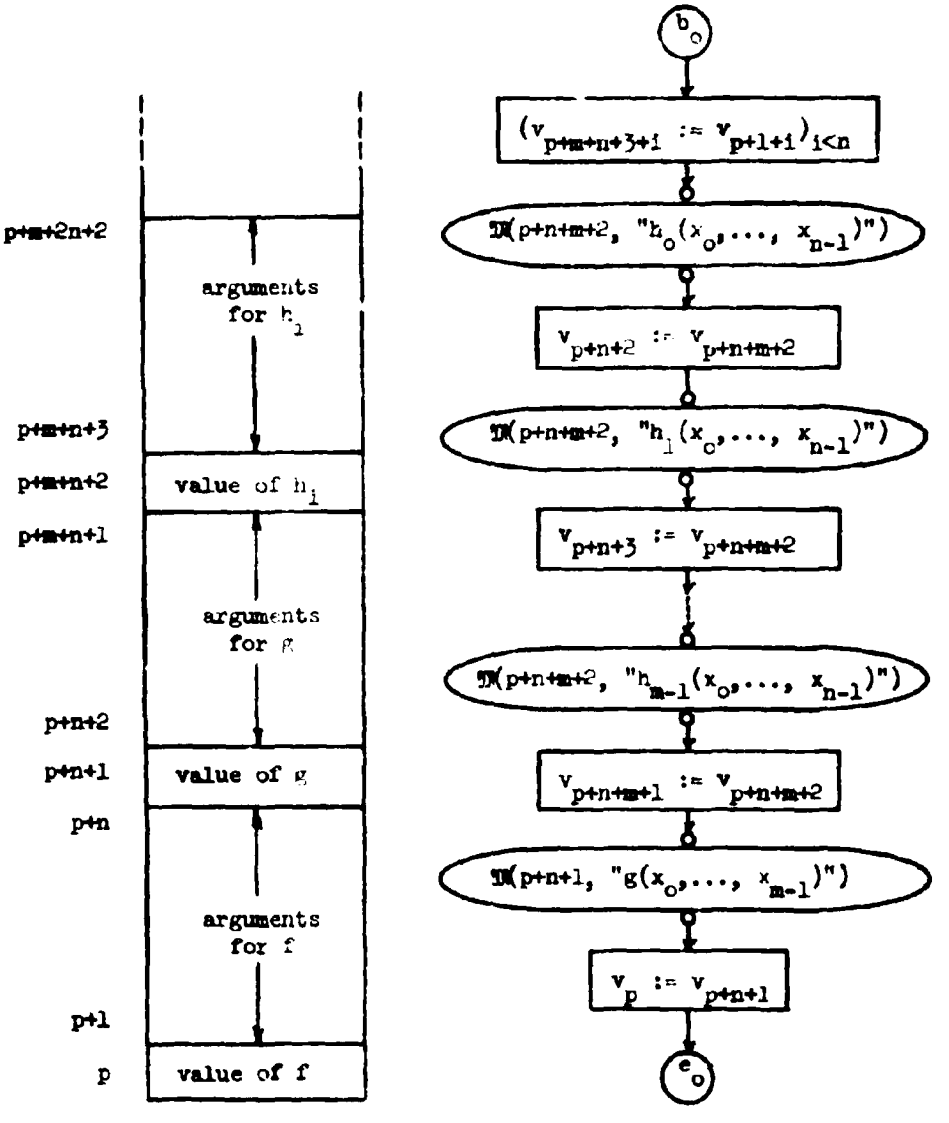


Figure 8

On the right is the definition of $\mathfrak{M}(p, "f(x_o,\ldots, x_{n-1})")$ where $f(x_o,\ldots, x_{n-1})$ is given by composition as $g(h_o(x_o,\ldots, x_{n-1}),\ldots, h_{m-1}(x_o,\ldots, x_{n-1}))$. On the left is the run-time stack showing the storage allocation given by $\mathfrak{M}$. The arguments for f, already present, are loaded as arguments for h_i, $i<m$. Then, $h_i(x_o,\ldots, x_{n-1})$, $i<m$, is computed and the result loaded as the i-th argument for g. Finally, g is computed with these arguments and the result returned as the value of f in v_p.

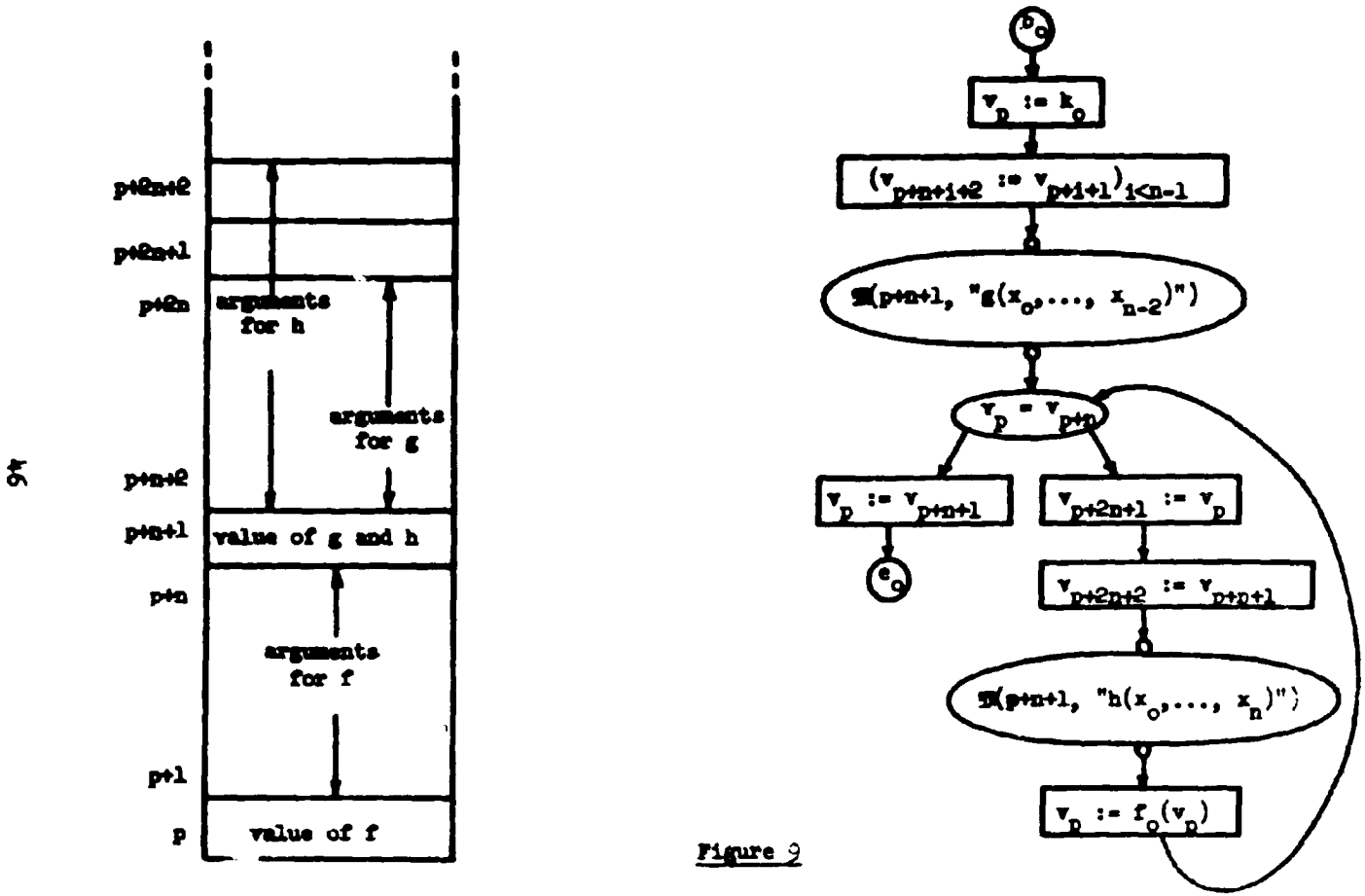


Figure 9

On the right is the definition of $\mathbb{N}(p, "f(x_o,\ldots, x_{n-1})")$ where $f(x_o,\ldots, x_{n-1})$ is defined by primitive recursion using $g(x_o,\ldots, x_{n-2})$ and $h(x_o,\ldots, x_{n-2}, x_{n-1}, f(x_o,\ldots, x_{n-2}, x_{n-1}))$. On the left is the run time stack. The value of $f(x_o,\ldots, x_{n-1})$ is computed "from the inside out": first, the arguments for g are loaded and g evaluated; then, using v_p as a temporary counter, h is repeatedly evaluated until the required depth of recursion is achieved. When evaluating h, the last argument, i.e., v_{p+2+2}, is loaded with the value of f at the previous level, i.e., v_{p+n+1}. Then v_p receives the final result.

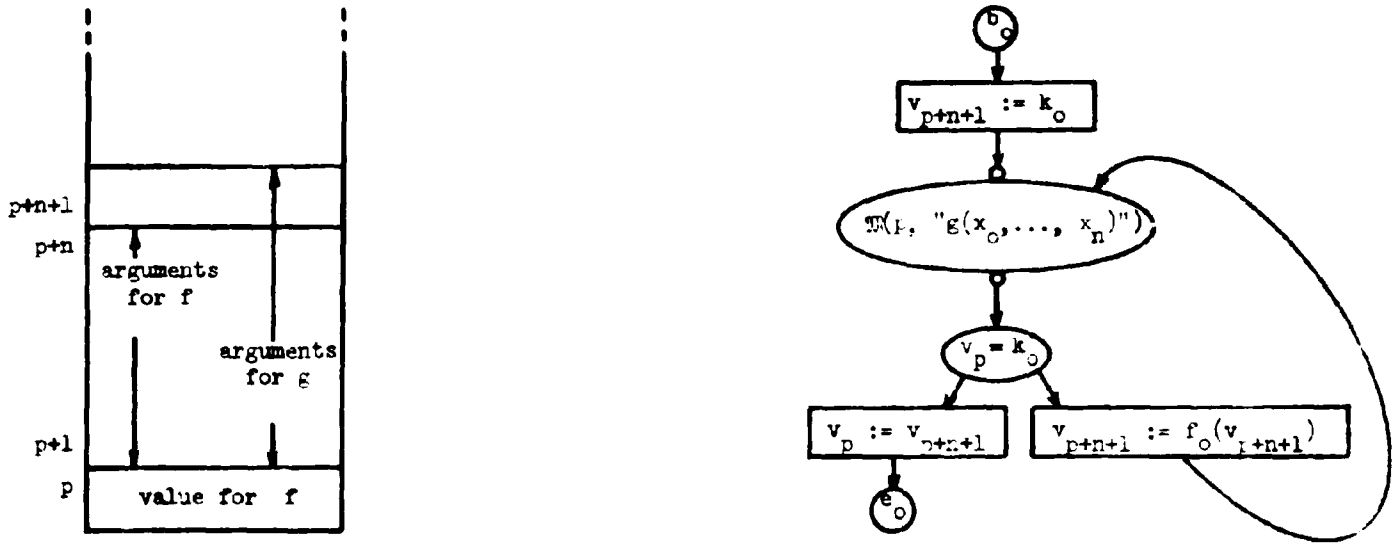


Figure 10

On the right is the definition of $\mathbb{T}(p, "f(x_o,\ldots, x_{n-1})")$ where $f(x_o,\ldots, x_{n-1})$ is defined by the unrestricted μ-operator using $g(x_o,\ldots, x_n)$. On the left is the run-time stack. The value of $f(x_o,\ldots, x_{n-1})$ is computed by simply computing $g(x_o,\ldots, x_{n-1}, y) = k$ for $y = 0, 1, \ldots$ until y is found such that $k = 0$. If no such y is found, the computation fails to halt, and the value of $f(x_o,\ldots, x_{n-1})$ is therefore undefined. The restriction on the unrestricted μ-operator assures us that for all z, if $g(x_o,\ldots, x_{n-1}, h) \neq 0$ for $y = 1, 2, \ldots, z-1$, then $g(x_o,\ldots, x_{n-1}, z)$ is defined.

Proof: An inductive proof on the structure of $\mathfrak{R}(\xi_f, \underset{\sim}{d})$ can be given in some detail. We will assume that the definition of $\mathfrak{R}(\xi_f, \underset{\sim}{d})$, the attendant comments, and the stack diagrams make the proof wholly obvious. ∎

Remarks:

(1) Ershov [9] indicates, for his scheme, how flowcharts can be given to compute partial recursive functions, but no details are given. There is really little or no ingenuity required in the construction of $\mathfrak{R}(\xi_f, \underset{\sim}{d})$ as we have seen above; this bespeaks of the naturalness of a flowchart representation of algorithms which utilizes assignment schemata and qffs.

Now let us return to Theorem 4. We are to show that $\mathfrak{T} \models \mathfrak{A} \simeq \mathfrak{B}$? , where $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}_{m_s}^{-}$ for signature $s = \langle 1, 1, 1 \rangle$, is undecidable. Recall that $\mathfrak{A}$ and $\mathfrak{B}$ contain no qff of the form $(\tau = \sigma)$.

Proof: Let us sketch a rough outline of the proof first before giving the details. We will show how to effectively construct, for any partial recursive function $f(x_0, \ldots, x_{n-1})$ and any arguments $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$, a type $\langle 1, 1 \rangle$ E-program $\mathfrak{R}^*(\xi_f, \underset{\sim}{d}) \in L_s^-$, where $s = \langle 1, 1, 1 \rangle$ such that if $\mathfrak{T} \in L_s^-$ is some type $\langle 1, 1 \rangle$ E-program that never halts (i.e., $\mathfrak{T}[\underset{\sim}{D}, \langle \xi, 0 \rangle]$ is indeterminate for all $\underset{\sim}{D}$ with signature $s = \langle 1, 1, 1 \rangle$ and all $\xi : \omega \to D_0$) then,

(i) There exists a computing structure $\underset{\sim}{X}$, with signature s , such that if $\mathfrak{R}^*(\xi_f, \underset{\sim}{d}) \simeq \mathfrak{T}$ is valid in $\underset{\sim}{X}$, then $f(d_0, \ldots, d_{n-1})$ does not exist;

(ii) for any computing structure $\underset{\sim}{D}$, with signature s , if $\mathfrak{R}^*(\xi_f, \underset{\sim}{d}) \simeq \mathfrak{T}$ is not valid in $\underset{\sim}{D}$, then $f(d_0, \ldots, d_{n-1})$ does exist.

Then, existence of a decision procedure for $? \models \mathfrak{A} \simeq \mathfrak{B}$? in general, where $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}_{\mathfrak{m}_s}^{-}$, would imply a decision procedure for $? \models \mathfrak{R}^*(\xi_f, \underset{\sim}{d}) \simeq \mathfrak{R}$? in particular, which, from (i) and (ii) above, would imply a decision procedure for the existence of $f(d_0, \ldots, d_{n-1})$ for an arbitrary partial recursive function f and arguments $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$. However, this last problem is trivially undecidable (cf. Mendelson [33, p. 255]), so that therefore a decision procedure for $? \models \mathfrak{A} \simeq \mathfrak{B}$?, where $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}_{\mathfrak{m}_s}^{-}$, does not exist.

From the foregoing discussion, we see that $\mathfrak{R}^*(\xi_f, \underset{\sim}{d})$ is going to have to behave as if it were attempting to compute $f(d_0, \ldots, d_{n-1})$, as it were, in all computing structures with signature $s = \langle 1, 1, 1 \rangle$. To accomplish this we introduce the concept of an image of a natural number. (cf. Luckham and Park [24] where a method of "representations" is used in a similar context.) Consider an arbitrary structure $\underset{\sim}{D} = \langle D, R_0, F_0, a_0 \rangle$ with signature s. We say that $x \in D$ is an <u>image</u> of $n \in \omega$ iff $R_0(F_0^k(x))$, $k < n$, and not $R_0(F_0^n(x))$. If we suppose $R' : D \to \{0, 1\}$ such that $R'(x) = 1$ if $R(x)$ and $R'(x) = 0$ otherwise, then x is an image of n iff the infinite sequence of 1's and 0's $\{R'(F^k(x))\}_{k<\omega}$ has an initial segment consisting of n 1's followed by a 0. Note that if $R'(x) = 0$ then x is an image of zero.

We will construct $\mathfrak{R}^*(\xi_f, \underset{\sim}{d}) \in L_s^{-}$, where $s = \langle 1, 1, 1 \rangle$, from $\mathfrak{R}(\xi_f, \underset{\sim}{d}) \in L_s$, where $s = \langle 0, 1, 1 \rangle$, by replacing individual constructs in $\mathfrak{R}(\xi_f, \underset{\sim}{d})$ with open subroutines in $\mathfrak{R}^*(\xi_f, \underset{\sim}{d})$ which have the required "in all computing structures" flavor. From the definition of $\mathfrak{R}(\xi_f, \underset{\sim}{d})$, we see that we will have to simulate, via open subroutines, the following constructs:

(i) loading of the arguments $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$

(ii) the test for equality of the values of two variables or of the value of a single variable and 0 .

(iii) the successor function +1 applied to the value of a variable

(iv) initializing the value of a variable to 0

(v) the assignment of the value of a variable to another performed by an assignment schema.

We will consider each of these in turn.

Loading of the arguments is accomplished with the aid of the macro shown in Figure 11(a). Here, starting from a_0, we search the domain D by repeated applications of F_0, at each step looking for an initial segment which indicates that the current value of the variable x is an image of $n \in \omega$. If it so happens that the computing structure $\langle D, R_0, F_0, a_0 \rangle$ is such that no image of n can be found, the routine will fail to halt. Figure 11(b) shows in detail how the argument loading section of $\mathfrak{R}(\mathcal{F}_f, \underset{\sim}{d})$ is simulated by a sequence of these macros in $\mathfrak{R}^*(\mathcal{F}_f, \underset{\sim}{d})$. If this section of $\mathfrak{R}^*(\mathcal{F}_f, \underset{\sim}{d})$ halts in some computing structure $\underset{\sim}{D}$, then the values of the variables $v_4, \ldots, v_{n+3}$ will be images of the arguments $d_0, \ldots, d_{n-1}$.

Testing for equality of the values of two variables is accomplished with the aid of the macro shown in Figure 12(a). Here, the working variables v_0 and v_1 are used to check that the values of the variables x and y are images of the same natural number by searching ahead using repeated applications of F_0 and checking for identity of the initial segments generated at each stage. By hypothesis, the values of x and y are both images of some natural numbers and so the computation halts in any computing structure, at terminator e_0 if the values of x and y are images of the same natural number and at e_1 if not.

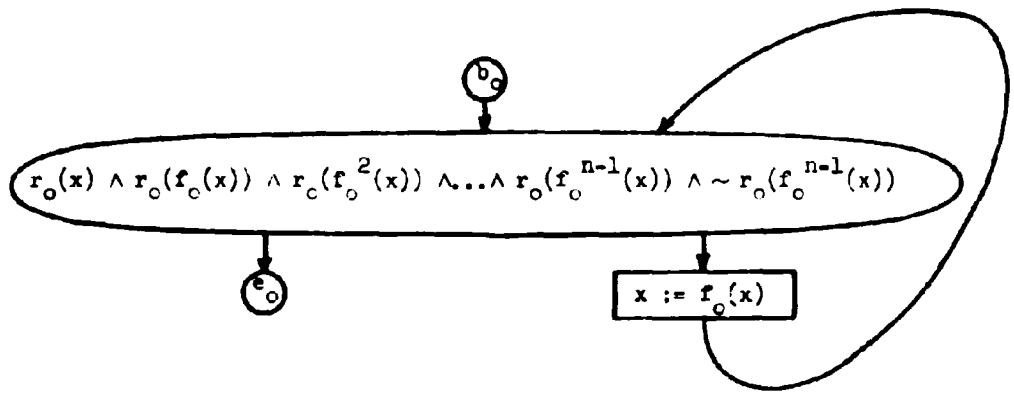


Figure 11(a)

Here, x is a variable. A search is made, starting from k_o, for an image of $n < \omega$.

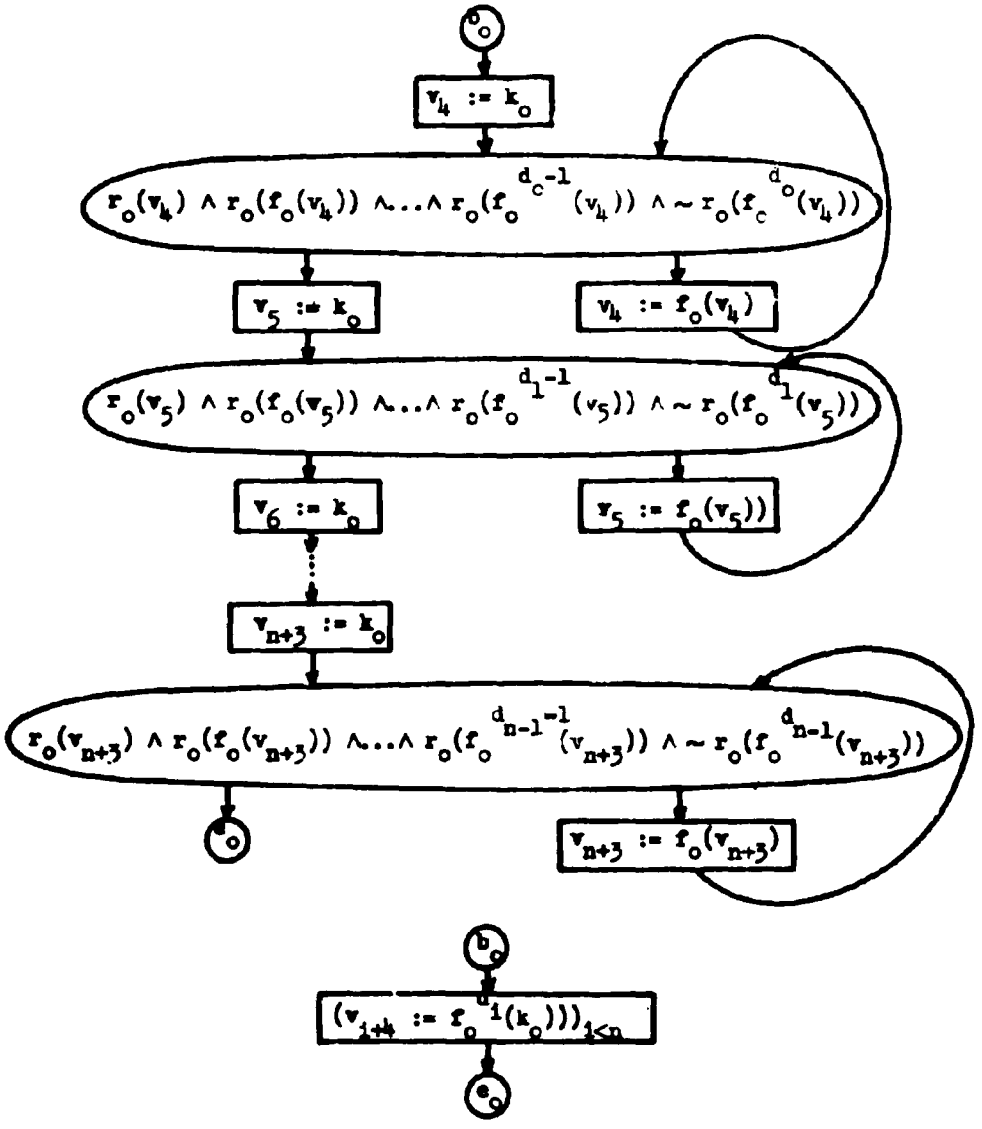


Figure 11(b)

The upper E-program replaces the lower in forming $\mathfrak{R}^*(\mathcal{E}_r, \underset{\sim}{d})$.

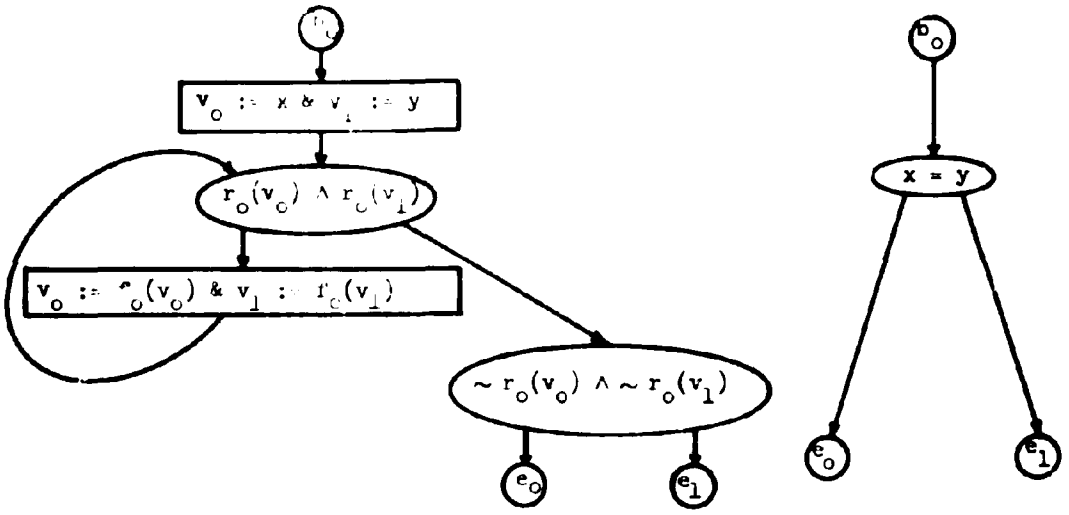


Figure 12(a)

Here, x and y are variables. The E-program on the left replaces the one on the right in $\mathfrak{R}^*(\mathcal{E}_f, \underset{\sim}{d})$.

Figure 12(b)

Here x is a variable. The E-program on the left replaces the one on the right in $\mathfrak{R}^*(\mathcal{E}_f, \underset{\sim}{d})$.

Testing for equality between the value of a variable x and 0 is accomplished with the aid of the macro shown in Figure 12(b). In $\mathfrak{N}(\mathcal{L}_f, \underset{\sim}{d})$, the qff $(x = k_0)$ is testing for 0 as the current value of variable x; in $\mathfrak{N}^*(\mathcal{L}_f, \underset{\sim}{d})$, the qff $\sim r_0(x)$ is testing for an image of 0 as the current value of variable x.

Applying the successor function is accomplished with the aid of the macro shown in Figure 13. Here, the value of the working variable v_2 is started at a_0 and stepped up by repeated applications of F_0. At each step, a check is made using the working variables v_0 and v_1 to see if the current value of v_2 is an image of a natural number greater by one than the number of which the value of variable x is an image. If the computation halts, the new value of variable x will be an image of $n+1$ where the old value of x was an image of n.

Initializing the value of a variable to 0 is accomplished with the macro of Figure 11(a) with n set to 0. Here the value of the variable x is initialized to k_0 and then F_0 is applied repeatedly until an image of 0 is found. If the computation halts, the value of x is an image of 0.

Assigning the value of one variable to another is done with precisely the same construct in $\mathfrak{N}^*(\mathcal{L}_f, \underset{\sim}{d})$ as in $\mathfrak{N}(\mathcal{L}_f, \underset{\sim}{d})$, i.e., an assignment schema.

This completes the definition of the process for constructing the E-program $\mathfrak{N}^*(\mathcal{L}_f, \underset{\sim}{d})$ in L_s^-, where $s = \langle 1, 1, 1 \rangle$, from the E-program $\mathfrak{N}(\mathcal{L}_f, \underset{\sim}{d})$ in L_s where $s = \langle 0, 1, 1 \rangle$. Evidently, an inductive proof on the construction of $\mathfrak{N}^*(\mathcal{L}_f, \underset{\sim}{d})$ gives us that if for any computing structure $\underset{\sim}{D}$, with signature

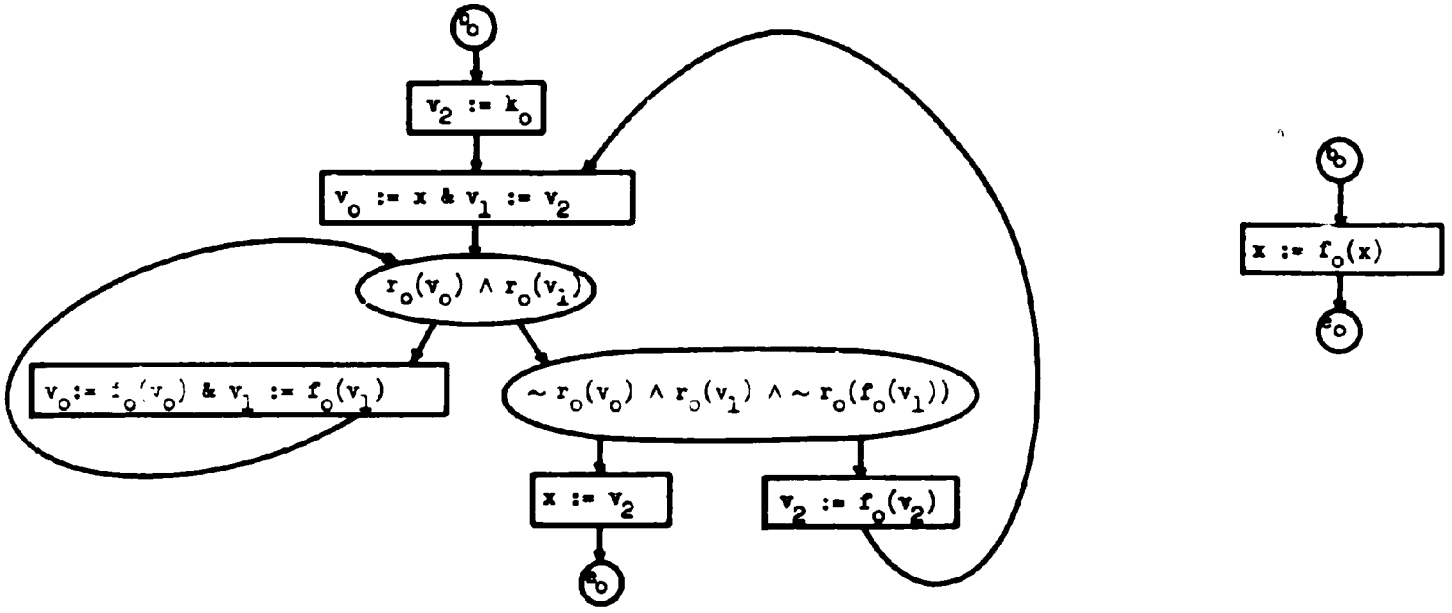


Figure 13

Here, x is a variable. The E-program on the left replaces the one on the right in $\mathfrak{R}^*(\mathcal{E}_f, \underset{\sim}{d})$.

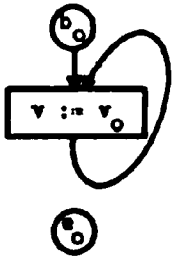


Figure 14

An always indeterminate type <1, 1> E-program.

$s = \langle 1, 1, 1 \rangle$ and input state $\xi : \omega \to \underset{\sim}{D}_o$, we have $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d})$ halts in $\underset{\sim}{D}$, then $f(d_o, \ldots, d_{n-1})$ must be defined. This is because determinacy implies that the required images were found at every stage, and this implies that the execution of $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d})$ therefore faithfully followed the execution of $\mathfrak{N}(\mathcal{E}_f, \underset{\sim}{d})$ which therefore must have halted, thus implying the existence of $f(d_o, \ldots, d_{n-1})$.

On the basis of this result, let us complete the proof of Theorem 4 by introducing the type $\langle 1, 1 \rangle$ E-program $\mathfrak{N} \in L_s^-$, where $s = \langle 1, 1, 1 \rangle$, in Figure 14. For topological reasons, $\mathfrak{N}[D, \langle \xi, \Phi \rangle]$ is indeterminate for all computing structures $\underset{\sim}{D}$ and input states $\xi : \omega \to \underset{\sim}{D}_o$. Let us consider the wff $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d}) \simeq \mathfrak{N} \in \mathcal{F}m_s^-$, and note that if $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d}) \simeq \mathfrak{N}$ is not valid in a certain computing structure $\underset{\sim}{D}$, then the reason must be that $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d})$ halts in $\underset{\sim}{D}$, i.e., in light of the discussion above, $f(d_o, \ldots, d_{n-1})$ exists. Thus, if we have a decision procedure for general validity, and we apply it to the wff $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d}) \simeq \mathfrak{N}$ and it says "not generally valid", then there must exist a computing structure $\underset{\sim}{D}$ in which the wff in question is not valid, which of course implies that $f(d_o, \ldots, d_{n-1})$ exists. In summary, there exists a wff $\theta_{f, \underset{\sim}{d}} \in \mathcal{F}m_s^-$ which if <u>not generally valid</u> then implies that $f(d_o, \ldots, d_{n-1})$ <u>exists</u>.

We now demonstrate the other alternative, namely, if $\theta_{f, \underset{\sim}{d}}$ <u>is generally valid</u> then $f(d_o, \ldots, d_{n-1})$ does <u>not exist</u>. To do this, we introduce the computing structure $\underset{\sim}{X} = \langle X, S, G, b \rangle$ with signature $\langle 1, 1, 1 \rangle$ defined so that the infinite sequence over $\{0, 1\}$ given by $\{S'(G^k(b))\}_{k<\omega}$ is just

0010010110010110111001011011101111 ...

which in a very obvious way contains images for the following sequence of natural numbers:

0010120123012 3 4 ...

Since the arguments $d_0, \ldots, d_{n-1}$ are loaded by searching from b using applications of G, and since all searches for images therefore stay in the sequence shown above, and since an image for every natural number occurs infinitely often, then all searches for images will succeed. This implies that if $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d}) = \mathfrak{N}$ is valid in $\underset{\sim}{X}$, and so $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d})$ does not halt in $\underset{\sim}{X}$, then it does so not because any image searches failed along the way, but because $\mathfrak{N}(\mathcal{E}_f, \underset{\sim}{d})$ does not halt in $\underset{\sim}{N}$, i.e., because $f(d_0, \ldots, d_{n-1})$ does not exist. Thus, if we have a decision procedure for general validity, and we apply it to the wff $\mathfrak{N}^*(\mathcal{E}_f, \underset{\sim}{d}) = \mathfrak{N}$ and it says "generally valid", then the wff in question must be valid in $\underset{\sim}{X}$ in particular, which of course implies that $f(d_0, \ldots, d_{n-1})$ does not exist. Thus, if $\theta_{f, \underset{\sim}{d}}$ is <u>generally valid</u> then $f(d_0, \ldots, d_{n-1})$ <u>does not exist</u>.

In summary, then, a decision procedure for general validity of wffs, $\theta_{f,\underset{\sim}{d}}$ in particular, would imply a decision procedure for the existence of $f(d_0, \ldots, d_{n-1})$ for arbitrary partial recursive functions and arguments $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$, which is impossible. Therefore $\vdash \mathfrak{M} = \mathfrak{N}$?, where $\mathfrak{M} = \mathfrak{N} \in \mathcal{F}_{M_s}$ for signature $s = \langle 1, 1, 1 \rangle$, is undecidable. ∎

Remarks:

(i) This undecidability result is essentially that given by Paterson [36] although our method of proof, obtained independantly, differs considerably. The forerunner of both these results is that given by Luckham and Park [24] for schemes that compute with the natural numbers.

BLANK PAGE

CHAPTER 6

FURTHER SYNTACTIC AND SEMANTIC PROPERTIES OF E-PROGRAMS

In Chapter 4, we introduced the formal theory $\mathcal{T}_S = \langle \mathcal{F}m_S, \mathcal{A}_S \rangle$, where $\mathcal{F}m_S$ is the set of wffs of $\mathcal{T}_S$ and $\mathcal{A}_S$ is the inferential system of $\mathcal{T}_S$. In Chapter 5, we examined the general validity decision problem for wffs in $\mathcal{F}m_S$, and in the present and succeeding chapters, we will complete our study of $\mathcal{T}_S$ by developing the inferential system $\mathcal{A}_S$. Before we get to specifying the actual axioms and rules of inference of $\mathcal{A}_S$ in Chapter 7, we will lay the necessary groundwork by examining further syntactic and semantic properties of E-programs in this chapter. We will assume a fixed signature $S = \langle \langle n_0, \ldots, n_{k-1} \rangle, \langle m_0, \ldots, m_{\ell-1} \rangle, p \rangle$ throughout.

Forward Substitution of Assignment Schemata

The first syntactic notions to be examined are illustrated in Figure 15(a). We want to know what assignment schema x and qff r will make the wffs pictured in Figure 15(a) generally valid.

To begin, let us consider the syntactic substitution of terms for variables. If t is a term or qff, then we write $(u_i ::= \sigma_i)_{i<n}\, t$, where $n < \omega$, to denote the term or qff obtained from t by the syntactic substitution of the terms σ_i, $i < n$, for all occurrences in t of the distinct variables u_i, $i < n$. We can conveniently read $(u ::= \sigma)t$ as "t with σ substituted for u"; the "::=" notation is, of course, borrowed from Backus-Naur Form where it also denotes substitution of strings. We define substitution rigorously as follows.

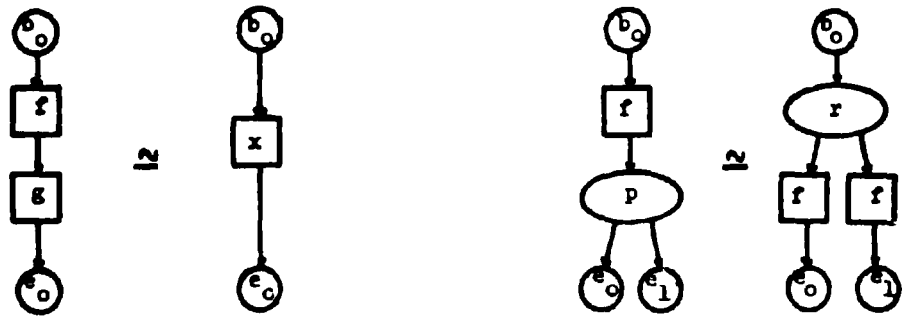


Figure 15(a)

Here, f and g are known assignment schema and p a known qff; x is an unknown assignment schema and r an unknown qff.

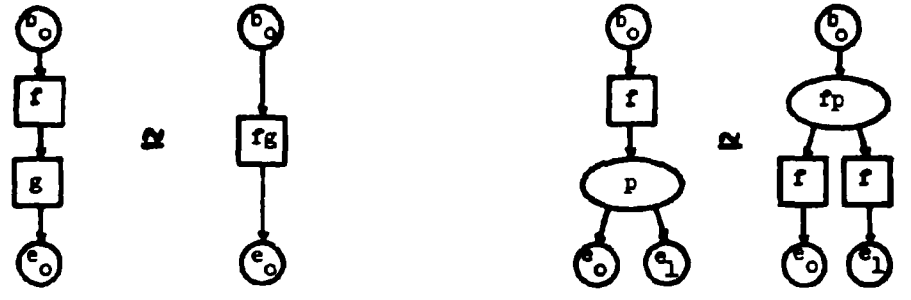


Figure 15(b)

Here, f and g are assignment schemata, p a qff, fg the forward substitution of f into g , and fp the forward substitution of f into p .

(i) If t is a variable v_j, then

$(u_i ::= \sigma_i)_{i<n} t = (u_i ::= \sigma_i)_{i<n} v_j = v_j$ if $u_i \neq v_j$, for all $i < n$

$= \sigma_i$ if $u_i = v_j$, for some $i < n$.

(ii) If t is a constant k_j, then $(u_i ::= \sigma_i)_{i<n} t$

$= (u_i ::= \sigma_i)_{i<n} k_j = k_j$.

(iii) If t is $f_j(\tau_o, \ldots, \tau_{m_j-1})$, then $(u_i ::= \sigma_i)_{i<n} t$

$= (u_i ::= \sigma_i)_{i<n} f_j(\tau_o, \ldots, \tau_{m_j-1}) = f_j((u_i ::= \sigma_i)_{i<n} \tau_o, \ldots, (u_i ::= \sigma_i)_{i<n} \tau_{m_j-1})$

(iv) If t is $r_j(\tau_o, \ldots, \tau_{n_j-1})$, then $(u_i ::= \sigma_i)_{i<n} t$

$= (u_i ::= \sigma_i)_{i<n} r_j(\tau_o, \ldots, \tau_{n_j-1}) = r_j((u_i ::= \sigma_i)_{i<n} \tau_o, \ldots, (u_i ::= \sigma_i)_{i<n} \tau_{n_j-1})$

(v) If t is $(\tau_1 = \tau_2)$, then $(u_i ::= \sigma_i)_{i<n} \tau$

$= (u_i ::= \sigma_i)_{i<n} (\tau_1 = \tau_2) = ((u_i ::= \sigma_i)_{i<n} \tau_1 = (u_i ::= \sigma_i)_{i<n} \tau_2)$.

(vi) If t is $(p \supset q)$, then $(u_i ::= \sigma_i)_{i<n} t$

$= (u_i ::= \sigma_i)_{i<n} (p \supset q) = ((u_i ::= \sigma_i)_{i<n} p \supset (u_i ::= \sigma_i)_{i<n} q)$.

(vii) If t is $(\sim p)$, then $(u_i ::= \sigma_i)_{i<n} t$

$= (u_i ::= \sigma_i)_{i<n} (\sim p) = (\sim (u_i ::= \sigma_i)_{i<n} p)$.

We will write $(u_i ::\approx \sigma_i)_{i<n} t$ if only <u>some</u> of the occurrences of the variables u_i are substituted for in t.

Next, let us define the syntactic operation on two assignment schemata denoted by their juxtaposition. Thus, if $f = (u_i := \tau_i)_{i<n}$ and $g = (w_j := \sigma_j)_{j<m}$, then their juxtaposition defines the assignment schema

$$fg = (w_j := (u_i ::= \tau_i)_{i<n} \sigma_j)_{j<m} \ \& \ (u_{i_k} := \tau_{i_k})_{k<\ell}$$

called the <u>forward substitution</u> of f into g, where $\{u_{i_k}\}_{k<\ell}$ is the largest subset of $\{u_i\}_{i<n}$ disjoint from $\{w_j\}_{j<m}$. Thus, we substitute forward all terms of f wherever the assigned variables of f occur in the terms of g, and in addition carry forward those assignments of f whose assigned variables do not conflict with those of g.

As well, let us define the syntactic operation on an assignment schema and a term or qff, denoted by their juxtaposition. Thus, if $f = (u_i := \tau_i)_{i<n}$ and t is a term or qff, then their juxtaposition defines the new term or qff

$$ft = (u_i ::= \tau_i)_{i<n} t$$

called the forward substitution of f into t.

The first step in the analysis of the forward substitution of assignment schemata is the following

Theorem 6: *For any assignment schema $f = (u_i := \tau_i)_{i<n}$, computing structure $\underset{\sim}{D}$ of appropriate signature, state $\xi : \omega \to \underset{\sim}{D}_o$ and either term or qff t,*

$$t[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]] = ft[\underset{\sim}{D}, \xi] .$$

Thus, executing an assignment schema f on the state ξ and then evaluating t yields the same value or truth value as the forward substitution of f into t, evaluated using ξ.

Proof: We use induction on the structure of t. First consider the case where t is a term.

(i) If t is the variable v_j, then

$t[\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$= v_j[\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$= v_j[D, \xi]$ if $v_j \neq u_i$ for all $i < n$, or $\tau_i[D, \xi]$ if $v_j = u_i$ for some $i < n$, by the definition of the semantics of assignment schemata and terms

$= (u_i ::= \tau_i)_{i<n} v_j[\underset{\sim}{D}, \xi]$ by the definition of substitution

$= (u_i ::= \tau_i)_{i<n} t[\underset{\sim}{D}, \xi]$ as required.

(ii) If t is the constant k_j, then

$t[\underset{\sim}{D}, (u_i := \tau_i)_{i<n} [\underset{\sim}{D}, \xi]]$

$= k_j[\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$= k_j[\underset{\sim}{D}, \xi]$ by the definition of the semantics for terms

$= (u_i ::= \tau_i)_{i<n} k_j[\underset{\sim}{D}, \xi]$ by the definition of substitution

$= (u_i ::= \tau_i)_{i<n} t[\underset{\sim}{D}, \xi]$ as required.

(iii) If t is $f_j(\tau_o, \ldots, \tau_{m_j-1})$, then

$t[\underset{\sim}{D}, (u_i := \tau_i)_{i<n} [\underset{\sim}{D}, \xi]]$

$= f_j(\tau_o, \ldots, \tau_{m_j-1}) [\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$= F_j(\tau_o[\underset{\sim}{D}, (u_i := \tau_i)_{i<n} [\underset{\sim}{D}, \xi]], \ldots, \tau_{m_j-1}[\underset{\sim}{D}, (u_i := \tau_i)_{i<n} [\underset{\sim}{D}, \xi]])$ by the definition of semantics for terms.

$= F_j((u_i ::= \tau_i)_{i<n} \tau_o[D, \xi], \ldots, (u_i ::= \tau_i)_{i<n} \tau_{m_j-1} [D, \xi])$ by induction hypothesis

$= f_j((u_i ::= \tau_i)_{i<n} \tau_o, \ldots, (u_i ::= \tau_i)_{i<n} \tau_{m_j-1})[\underset{\sim}{D}, \xi]$ by the definition of semantics for terms

$= (u_i ::= \tau_i)_{i<n} f_j(\tau_o, \ldots, \tau_{m_j-1})[\underset{\sim}{D}, \xi]$ by the definition of substitution

$= (u_i ::= \tau_i)_{i<n} t[\underset{\sim}{D}, \xi]$ as required.

Next, let us consider the case where t is a qff.

(i) If t is $r_j(\tau_o, \ldots, \tau_{n_j-1})$, then the proof parallels exactly the proof when t is $i_j(\tau_o, \ldots, \tau_{m_j-1})$.

(ii) If t is $(p \supset q)$, then

$t[\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$= (p \supset q)[\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$=$ not $p[\underset{\sim}{D}, (u_i := \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$ or $q[\underset{\sim}{D}, (u_i := \tau_i)_{i<n} [\underset{\sim}{D}, \xi]]$

by the definition of semantics for qffs

$\Leftrightarrow$ not $(u_i ::= \tau_i)_{i<n}\, p[\underset{\sim}{D}, \xi]$ or $(u_i ::= \tau_i)_{i<n}\, q[\underset{\sim}{D}, \xi]$ by induction hypothesis

$\Leftrightarrow$ $((u_i ::= \tau_i)_{i<n}\, p \supset (u_i ::= \tau_i)_{i<n}\, q)[\underset{\sim}{D}, \xi]$ by the definition of semantics for qffs

$\Leftrightarrow$ $(u_i ::= \tau_i)_{i<n}\, (p \supset q)[\underset{\sim}{D}, \xi]$ by the definition of substitution

$\Leftrightarrow$ $(u_i ::= \tau_i)_{i<n}\, t[D, \xi]$ as required.

(iii) If t is $(\sim p)$, then the proof parallels exactly the proof when t is $(p \supset q)$.

(iv) If t is $(\tau = \sigma)$, then

$t[\underset{\sim}{D}, (u_i ::= \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$\Leftrightarrow$ $(\tau = \sigma)[\underset{\sim}{D}, (u_i ::= \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$

$\Leftrightarrow$ $\tau[\underset{\sim}{D}, (u_i ::= \tau_i)_{i<n}[\underset{\sim}{D}, \xi]] = \sigma[\underset{\sim}{D}, (u_i ::= \tau_i)_{i<n}[\underset{\sim}{D}, \xi]]$ by the definition of semantics for qffs

$\Leftrightarrow$ $(u_i ::= \tau_i)_{i<n}\tau[\underset{\sim}{D}, \xi] = (u_i ::= \tau_i)_{i<n}\sigma[\underset{\sim}{D}, \xi]$ by induction hypothesis

$\Leftrightarrow$ $((u_i ::= \tau_i)_{i<n}\, \tau = (u_i ::= \tau_i)_{i<n}\sigma)[\underset{\sim}{D}, \xi]$ by the definition of the semantics of qffs

$\Leftrightarrow$ $(u_i ::= \tau_i)_{i<n}(\tau = \sigma)[\underset{\sim}{D}, \xi]$ by the definition of substitution

$\Leftrightarrow$ $(u_i ::= \tau_i)_{i<n}\, t[\underset{\sim}{D}, \xi]$ as required. ∎

The foregoing theorem immediately suggests the following.

Theorem 7: *For any assignment schemata* $f = (v_{m_i} ::= \tau_i)_{i<M}$ *and* $g = (v_{n_j} ::= \sigma_j)_{j<N}$, *computing structure* $\underset{\sim}{D}$ *of appropriate signature, and state* $\xi : \omega \to \underset{\sim}{D}_o$

$g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]] = fg[\underset{\sim}{D}, \xi]$.

Thus, executing two assignment schemata f and g in sequence on a state ξ produces the same state as the forward substitution of f into g executed on ξ.

Proof: It is possible to give an inductive proof over the number of assignments in either f or g, but a brute force method may, in fact, prove more merciful.

$g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$

$= (v_{n_j} := \sigma_j)_{j<N}[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$

$= (v_{n_j} := \sigma_j)_{j<N}[\underset{\sim}{D}, \eta]$, where $\eta = f[\underset{\sim}{D}, \xi]$

$= a(n_{N-1}, \sigma_{N-1}[\underset{\sim}{D}, \eta], a(\ldots a(n_1, \sigma_1[\underset{\sim}{D}, \eta], a(n_o, \sigma_o[\underset{\sim}{D}, \eta], \eta))\ldots))$ by the definition of semantics for assignment schemata.

Now, for all $j < n$

$\sigma_j[\underset{\sim}{D}, \eta]$

$= \sigma_j[\underset{\sim}{D}, (v_{m_i} := \tau_i)_{i<M}[\underset{\sim}{D}, \xi]]$ since $\eta = f[\underset{\sim}{D}, \xi]$

$= (v_{m_i} := \tau_i)_{i<M}\, \sigma_j[\underset{\sim}{D}, \xi]$ by Lemma 1,

$= f\sigma_j[\underset{\sim}{D}, \xi]$ by the definition of forward substitution of assignment schemata.

So that, continuing

$g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$

$= a(n_{N-1}, f\sigma_{N-1}[\underset{\sim}{D}, \xi], a(\ldots a(n_1, f\sigma_1[\underset{\sim}{D}, \xi], a(n_o, f\sigma_o[\underset{\sim}{D}, \xi], \eta))\ldots))$

$= a(n_{N-1}, f\sigma_{N-1}[\underset{\sim}{D}, \xi], a(\ldots a(n_1, f\sigma_1[\underset{\sim}{D}, \xi], a(n_o, f\sigma_o[\underset{\sim}{D}, \xi],$

$a(m_{M-1}, \tau_{M-1}[\underset{\sim}{D}, \xi], a(\ldots a(m_1, \tau_1[\underset{\sim}{D}, \xi], a(m_o, \tau_o[\underset{\sim}{D}, \xi], \xi))\ldots))))\ldots))$

The next step in the proof relies on certain axioms which characterize expressions involving a, the "assign" function, and c, the "contents" function. As mentioned above, these functions were introduced by McCarthy [28], and there he also gives the axioms

(i) $a(i, k, a(j, \ell, \xi)) = a(j, \ell, a(i, k, \xi))$ if $i \neq j$

$= a(i, k, \xi)$ if $i = j$

(ii) $a(i, c(i, \xi), \xi) = \xi$

(iii) $c(i, a(j, k, \xi)) = c(i, \xi)$ if $i \neq j$

$= k$ if $i = j$

which, this author [21] has shown to be both sound and adequate for deriving equality of states.

The relevance of these completeness results is that in the last expression for $g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$ above, if $n_j = m_i$ for some $j < N$, $i < M$, then we can prove that the assignment to m_i can be omitted since the one to n_j will be the only one to have effect. In fact the proof consists simply repeated applications of axiom (i) above. Let us suppose, without lack of generality, that $\{m_0, m_1 \ldots, m_{K-1}\}$ is the largest subset of $\{m_i\}_{i<M}$ disjoint from $\{n_j\}_{j<N}$, where $K \leq M$. Then, on the basis of the above discussion, $g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$

$= a(n_{N-1}, f\sigma_{N-1}[\underset{\sim}{D}, \xi], a(\ldots a(n_0, f\sigma_0[\underset{\sim}{D}, \xi], a(m_{K-1}, \tau_{K-1}[\underset{\sim}{D}, \xi], a(\ldots$

$a(m_1, \tau_1[\underset{\sim}{D}, \xi], a(m_0, \tau_0[\underset{\sim}{D}, \xi], \xi))\ldots))))\ldots))$

$= (v_{n_j} := f\sigma_j)_{j<N}$ & $(v_{m_i} := \tau_i)_{i<K}[\underset{\sim}{D}, \xi]$ by the definition of the semantics for assignment schemata

$= (v_{n_j} := (v_{m_i} := \ldots)_{i<M}\sigma_j)_{j<N}$ & $(v_{m_i} := \tau_i)_{i<K}[\underset{\sim}{D}, \xi]$ by the definition of forward substitution of assignment schemata

$= fg[\underset{\sim}{D}, \xi]$ as required. ∎

Remarks:

(i) In [21], we explain in detail how the axioms for the a and functions can be used to effect the simplification required in the foregoing theorem. It is felt that restatement of all the results in [21] is not warranted here.

(ii) Theorems 6 and 7 lead to a certain parsimony in notation for the diagrammatic representations of E-programs, and this will prove useful when we carry out deductions using the drs of various E-programs.

(iii) We now know through its forward substitution the entire effect of an operator, whether on another operator or on a discriminator. It is precisely the lack of this complete information that distinguishes the schemes of Ianov [16] and Glushkov [13] from that developed here.

Theorems 6 and 7 seem to tell us that in Figure 15(a) the assignment schema x should be fg and the qff r should be fp for the wffs pictured there to be generally valid. This, in fact, is the content of the following

Theorem 8: The wffs of Figure 15(b)(i) and 15(b)(ii) are generally valid.

Proof: (i) Consider executing the left-hand E-program $\mathfrak{M} = \langle X, \Gamma, \mathcal{X} \rangle$ say, in an arbitrary computing structure $\underset{\sim}{D}$, of the appropriate signature, with an arbitrary input state $\xi : \omega \to D_o$. Then,

$\mathfrak{M}[\underset{\sim}{D}, \langle \xi, \Lambda \rangle]$

$= E(\mathfrak{M}, \underset{\sim}{D}, \xi, w)$ where $w \in X$ and $[w] = b_o$

$= E(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma w)$ by the definition of the execution function E

$= E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ where $x \in X$ and $[x] = f$

$= E(\mathfrak{M}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma x)$ by the definition of E

$= E(\mathfrak{M}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], y)$ where $y \in X$ and $[y] = g$

$= E(\mathfrak{M}, \underset{\sim}{D}, g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]], \Gamma y)$ by the definition of E

$= E(\mathfrak{M}, \underset{\sim}{D}, g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]], z)$ where $z \in X$ and $[z] = e_o$

$= \langle g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]], \Lambda \rangle$ by the definition of E

$= \langle fg[\underset{\sim}{D}, \xi], \Lambda \rangle$ by Theorem 7.

Now, consider executing the right-hand E-program, $\mathfrak{B} = \langle X, \Gamma, \mathscr{L} \rangle$ say, in the same fashion. So

$\mathfrak{B}[\underset{\sim}{D}, \langle \xi, \omega \rangle]$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, x)$ where $x \in X$ and $[x] = b_0$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, \Gamma x)$ by the definition of E

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, y)$ where $y \in X$ and $[y] = fg$

$= E(\mathfrak{B}, \underset{\sim}{D}, fg[\underset{\sim}{D}, \xi], \Gamma y)$ by the definition of E

$= E(\mathfrak{B}, \underset{\sim}{D}, fg[\underset{\sim}{D}, \xi], z)$ where $z \in X$ and $[z] = e_0$

$= \langle fg[\underset{\sim}{D}, \xi], \omega \rangle$ by the definition of E.

Thus, for arbitrary $\underset{\sim}{D}$ and ξ, $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, \omega \rangle] = \mathfrak{A}[\underset{\sim}{D}, \langle \xi, \omega \rangle]$,

i.e., $\models \mathfrak{A} \simeq \mathfrak{B}$, as required.

(ii) Consider executing the left-hand E-program, $\mathfrak{A} = \langle X, \Gamma, \mathscr{L} \rangle$ say, in an arbitrary computing structure $\underset{\sim}{D}$, of the appropriate signature, with an arbitrary input state $\xi : \omega \to \underset{\sim}{D}_0$. Then,

$\mathfrak{A}[D, \langle \xi, \omega \rangle]$

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, v)$ where $v \in X$ and $[v] = b_0$

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, \Gamma v)$ by the definition of E

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, w)$ where $w \in X$ and $[w] = f$

$= E(\mathfrak{A}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma w)$ by the definition of E

$= E(\mathfrak{A}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], x)$ where $x \in X$ and $[x] = p$

$= E(\mathfrak{A}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], y)$ if $p[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$, or $E(\mathfrak{A}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], z)$

otherwise, where $y, z \in X$ and $[y] = e_0$, $[z] = e_1$ by the definition of E

$= E(\mathfrak{A}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], y)$ if $fp[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{A}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], z)$

otherwise, by Theorem 6.

$= \langle f[\underset{\sim}{D}, \xi], \mathcal{O}\rangle$ if $fp[\underset{\sim}{D}, \xi]$ or $\langle f[\underset{\sim}{D}, \xi], \mathcal{I}\rangle$ otherwise, definition of E .

Now, consider executing the right-hand E-program $\mathfrak{B} = \langle X, \Gamma, \mathcal{C}\rangle$ say, in the same fashion. So

$\mathfrak{B}[\underset{\sim}{D}, \langle \xi, \mathcal{O}\rangle]$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, u)$ where $u \in X$ and $[u] = b_c$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, \Gamma u)$ by the definition of F

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, v)$ where $v \in X$ and $[v] = fp$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, w)$ if $fp[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{B}, \underset{\sim}{D}, \xi, x)$ otherwise,
where $w, x \in X$ and $[w] = f$, $[x] = f$, by the definition of E

$= E(\mathfrak{B}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma w)$ if $fp[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{B}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma x)$ otherwise,
by the definition of E

$= E(\mathfrak{B}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], y)$ if $fp[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{B}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], z)$ otherwise,
where $y, z \in X$ and $[y] = e_o$, $[z] = e_1$

$= \langle f[\underset{\sim}{D}, \xi], \mathcal{O}\rangle$ if $fp[\underset{\sim}{D}, \xi]$, or $\langle f[\underset{\sim}{D}, \xi], \mathcal{I}\rangle$ otherwise, by
the definition of E .

Thus, for arbitrary $\underset{\sim}{D}$ and ξ , $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, \mathcal{O}\rangle] = \mathfrak{B}[\underset{\sim}{D}, \langle \xi, \mathcal{O}\rangle]$
i.e., $\models \mathfrak{A} \simeq \mathfrak{B}$, as required. ∎

Remarks:

(i) Though the proof of the foregoing theorem is somewhat tedious, it nevertheless points up the role of the execution function in semantically oriented proofs of general validity for wffs. This sort of verification of general validity will certainly be required of all the axioms in the inferential system introduced in the next chapter.

(ii) Of course, the wffs of Figure 15(b) will be key axioms in that inferential system

Instantiation of Well-Formed Formulas

We want to extend the ideas of the preceding section to allow substitution of terms for variables whenever they occur in the E-programs of a wff. The wff resulting from such a substitution will be called an instance of the original one. We are also interested in specifying the conditions under which an instance of a wff is also a semantic consequence of it. This sort of syntactic process must be available if we are to carry out derivations from a set of proper axioms or hypotheses.

First, let us extend the notion of substitution to E-programs, and write $(u_i ::= \tau_i)_{i<n}\,\mathfrak{M}$, where $n < \omega$, to denote the E-program obtained from $\mathfrak{M}$ by the simultaneous syntactic substitution of the terms τ_i, $i<n$, for all occurrences in $\mathfrak{M}$ of the distinct variables u_i, $i<n$. Note that if some u_i, $i<n$, occurs as an assigned variable in an assignment schema of $\mathfrak{M}$, and τ_i is not a variable, then $(u_i ::= \tau_i)_{i<n}\,\mathfrak{M}$ is not a permitted syntactic operation since the result, if the substitutions were performed as indicated, would not be an E-program.

To give a more precise definition, suppose that $\mathfrak{M} = \langle \mathfrak{A}, \Gamma, \mathcal{X} \rangle$. Then $(u_i ::= \tau_i)_{i<n}\,\mathfrak{M} = \langle \mathfrak{A}, \Gamma, \mathcal{X}' \rangle$ where $\mathcal{X}'$ is defined as follows. (Note: we write $[x]'$ for $\mathcal{X}'(x)$ here.)

(i) If $[x] \in \mathfrak{V} \cup \mathfrak{E}$, then $[x]' = [x]$.

(ii) If $[x] \in \mathfrak{A}$, then $[x]' = (u_i ::= \tau_i)_{i<n}[x]$.

(iii) If $[x]$ [illegible], say, $[x] = (w_j ::= \tau_j)_{j<m}$, then

$[x]' = ((u_i ::= \tau_i)_{i<n}\ w_j ::= (u_i ::= \tau_i)_{i<n}\ \sigma_j)_{j<m}$

Of course, as mentioned above, if $u_i = w_j$ for some $i<n$, $j<m$ and τ_i is not a variable, then $(u_i ::= \tau_i)_{i<n}\ \mathfrak{A}$ is not a defined operation.

Then, an instance $(u_i ::= \tau_i)_{i<n}\ \mathfrak{A} = \mathfrak{B}$ of the wff $\mathfrak{A} = \mathfrak{B}$ is simply $(u_i ::= \tau_i)_{i<n}\ \mathfrak{A} = (u_i ::= \tau_i)_{i<n}\ \mathfrak{B}$.

To discover when an instance of a wff is also a semantic consequence thereof, we need the concepts of scope and freedom. Consider the E-program $\mathfrak{M} = \langle X, \Gamma, \mathcal{L} \rangle$. If the variable u occurs as an assigned variable in an assignment schema $[x]$, where $x \in X$, then we define the scope of that occurrence of u to be the set of variables which have occurrences in the assignment schemata and qffs labelling nodes reachable via Γ from x. The example in Figure 16 indicates the scopes of the assigned variables in a simple E-program.

Then, if $\mathcal{V}(\tau) = \{w : w$ is a variable and w occurs in the term $\tau\}$, we say that the term τ is free for the variable u in $\mathfrak{M}$ iff (i) and (ii) below both hold.

(i) For each occurrence of a variable $w \in \mathcal{V}(\tau)$ as an assigned variable in $\mathfrak{M}$, u is not in the scope of that occurrence.

(ii) If u occurs anywhere in $\mathfrak{M}$ as an assigned variable, then τ is simply a variable w, and for each such occurrence of u as an assigned variable, w is not in the scope of that occurrence.

In Figure 16, notice that u is free for v, and that v is free for u, $f(u)$ and $g(u, v)$.

We bring together the notions of substitution, scope and freedom in the following

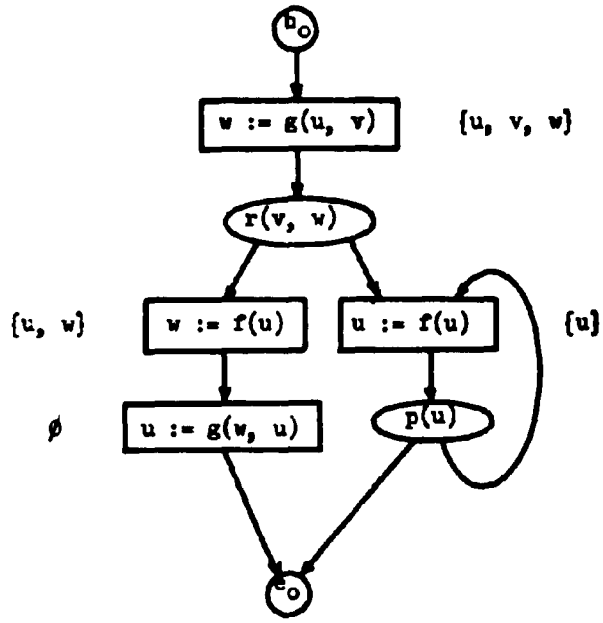


Figure 16

Here, u, v, w are distinct variables; f, g are function letters; and p, r are relation letters. The set of variables associated with each assignment schema is the scope of the assigned variable.

Theorem 9: For any signature s, any wff $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}m_s$, terms τ_i and distinct variables u_i, $i<n<\omega$, if for all $i<n$, τ_i is free for u_i in $\mathfrak{A}$ and $\mathfrak{B}$, then

(i) for all computing structures $\underset{\sim}{D}$ with signature s,

$\vdash_{\underset{\sim}{D}} \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \vdash_{\underset{\sim}{D}} (u_i ::= \tau_i)_{i<n}\, \mathfrak{A} \simeq \mathfrak{B}$,

or alternatively,

(ii) $\{\mathfrak{A} \simeq \mathfrak{B}\} \vdash (u_i ::= \tau_i)_{i<n}\, \mathfrak{A} \simeq \mathfrak{B}$.

In this case, we say that $(u_i ::= \tau_i)_{i<n}\, \mathfrak{A} \simeq \mathfrak{B}$ is a proper instantiation of $\mathfrak{A} \simeq \mathfrak{B}$. Evidently, then, any proper instantiation of a wff is a semantic consequence thereof.

Proof: The notion of freedom here is actually very simple, and the theorem follows easily once the nature of an E-program $\mathfrak{A}$ in which a term τ is free for a variable u is understood. We will employ an intuitive proof rather than a highly technical one since the latter would only obscure the simplicity of the situation. We will assume a fixed computing structure $\underset{\sim}{D}$.

Let us first consider the simple case of a term $f(w)$, say, where f is a function letter and w is a variable, being substituted for the variable v_K in the type $\langle m, n\rangle$ E-program $\mathfrak{A}$. In this case, only condition (i) for freedom is relevant, and we require that v_K is not in the scope of any occurrence of w as an assigned variable. That is, for any execution of $\mathfrak{A}$, v_K will not be evaluated once an assignment to w has been made. It follows, then, that any evaluation of a substituted occurrence of $f(w)$ during the execution of $(v_K ::= f(w))\mathfrak{A}$ will simply utilize the value of w it had in the input state. Thus, executing $(v_K ::= f(w))\mathfrak{A}$ with any input state η gives the

same result as executing $\mathfrak{A}$ with input state $a(K, f(w)[\underset{\sim}{D}, \eta], \eta)$, i.e., $(v_K ::= f(w))\mathfrak{A}[\underset{\sim}{D}, \langle\eta, i\rangle] \cong \mathfrak{A}[\underset{\sim}{D}, \langle a(K, f(w)[\underset{\sim}{D}, \eta], \eta), i\rangle]$, for all $i<m$.

Of course, the same sort of result holds for E-program $\mathfrak{B}$, i.e., $(v_K ::= f(w))\mathfrak{B}[\underset{\sim}{D}, \langle\eta, i\rangle] \cong \mathfrak{B}[\underset{\sim}{D}, \langle a(K, f(w)[\underset{\sim}{D}, \eta], \eta), i\rangle]$, for all $i<m$, if $f(w)$ is free for v_K in $\mathfrak{B}$. But, the hypotheses for Theorem 9 give us that $\models_{\underset{\sim}{D}} \mathfrak{A} \simeq \mathfrak{B}$, i.e., for all $i<m$, for all $\xi : \omega \to \underset{\sim}{D}_\omega$, we have $\mathfrak{A}[\underset{\sim}{D}, \langle\xi, i\rangle] \cong \mathfrak{B}[\underset{\sim}{D}, \langle\xi, i\rangle]$. But then the transitivity of $\cong$ gives that for all $i<m$,

$$\mathfrak{A}[\underset{\sim}{D}, \langle a(K, f(w)[\underset{\sim}{D}, \eta], \eta), i\rangle] \cong \mathfrak{B}[\underset{\sim}{D}, \langle a(K, f(w)[\underset{\sim}{D}, \eta], \eta), i\rangle]$$

since for any state $\eta: \omega \to \underset{\sim}{D}_\omega$, there is some $\xi : \omega \to \underset{\sim}{D}_\omega$ such that $\xi = a(K, f(w)[\underset{\sim}{D}, \eta], \eta)$. Then, combining the results above, we have $(v_K ::= f(w))\mathfrak{A}[\underset{\sim}{D}, \langle\eta, i\rangle] \cong (v_K ::= f(w))\mathfrak{B}[\underset{\sim}{D}, \langle\eta, i\rangle]$, for all $i<m$. But, this is true for all $\eta: \omega \to \underset{\sim}{D}_\omega$, so that $\models_{\underset{\sim}{D}} (v_K ::= f(w))\mathfrak{A} \simeq (v_K ::= f(w))\mathfrak{B}$, i.e., $\models_{\underset{\sim}{D}} (v_K ::= f(w))\mathfrak{A} \simeq \mathfrak{B}$, as required.

Extension of this result to terms with more than one distinct variable occurring in it is straightforward. But, now consider the more complicated case of a variable v_ℓ being substituted for another variable v_K in the type $\langle m, n\rangle$ E-program $\mathfrak{A}$, where v_K occurs in $\mathfrak{A}$ as an assigned variable. In this case, both conditions (i) and (ii) for freedom are relevant, so that we require that v_K is not in the scope of any occurrence of v_ℓ as an assigned variable and v_ℓ is not in the scope of any occurrence of v_K as an assigned variable. That is, for any execution of $\mathfrak{A}$, v_K will not be evaluated or be assigned a value once an assignment to v_ℓ has been made, and v_ℓ will not be evaluated or be assigned a new value once an assignment to v_K has been made. For both these conditions to be true simultaneously, either v_ℓ or v_K never occurs in

$\mathfrak{M}$ as an assigned variable. If it is v_K that never occurs in $\mathfrak{M}$ as an assigned variable, then we have precisely the case already analyzed above.

If we have the case where v_ℓ does not occur as an assigned variable in $\mathfrak{M}$, then it follows that any evaluation of a substituted occurrence of v_ℓ during the execution of $(v_K ::= v_\ell)\mathfrak{M}$ will simply utilize the value of v_ℓ it had in the input state. Thus, except for the values of v_K and v_ℓ in the output state (if such is determined), executing $(v_K ::= v_\ell)\mathfrak{M}$ with any input state η gives the same result as executing $\mathfrak{M}$ with input state $a(K, v_\ell[\underset{\sim}{D}, \eta], \eta)$, i.e.,

$(v_K ::= v_\ell)\mathfrak{M}[\underset{\sim}{D}, \langle\eta, i\rangle] \doteq_{\{v_K, v_\ell\}} \mathfrak{M}[\underset{\sim}{D}, \langle a(K, v_\ell[\underset{\sim}{D}, \eta], \eta), i\rangle]$ for all $i<m$.

The notation whereby the "$\doteq$" symbol is subscripted by a set of variables was first used by McCarthy [26], and has also been used by this author [20]. Here, for $\mathfrak{M}[\underset{\sim}{D}, \langle\xi, i\rangle] \doteq_{\{u_0,\ldots,u_{n-1}\}} \mathfrak{N}[\underset{\sim}{D}, \langle\xi, i\rangle]$ to hold, either both sides are indeterminate, or both are determinate, producing $\langle\xi', j'\rangle$ and $\langle\xi'', j''\rangle$ say, such that $j' = j''$ and $c(K, \xi') = c(K, \xi'')$ for all $K \in \{m : v_m \in \{u_0, \ldots, u_{n-1}\}\}$. Thus, ξ' and ξ'' may differ on in locations corresponding to the variables $u_0, \ldots, u_{n-1}$.

In this instance, we can actually say what the relationship is between the values of v_K and v_ℓ in the output state, if there is one. Suppose that

(i) $(v_K ::= v_\ell)\mathfrak{M}[\underset{\sim}{D}, \langle\eta, i\rangle] = \langle\alpha', j'\rangle$

(ii) $\mathfrak{M}[\underset{\sim}{D}, \langle a(K, v_\ell[\underset{\sim}{D}, \eta], \eta), i\rangle] = \langle\alpha'', j'\rangle$.

Then evidently, $c(\ell, \alpha') = c(K, \alpha'')$, $c(K, \alpha') = c(K, \eta)$, $c(\ell, \alpha'') = c(\ell, \eta)$ and $\langle\alpha', j'\rangle = \{v_K, v_\ell\}\langle\alpha'', j'\rangle$.

Of course, the same sort of result holds for E-program $\mathfrak{B}$, i.e.,

$(v_K ::= v_\ell)\mathfrak{B}[\underset{\sim}{D}, \langle\eta, i\rangle] = {}_{\{v_K, v_\ell\}}\mathfrak{B}[\underset{\sim}{D}, \langle a(K, v[\underset{\sim}{D}, \eta], \eta), i\rangle]$

for all $i<m$. Let us suppose here that

(i) $(v_K ::= v_\ell)\mathfrak{B}[\underset{\sim}{D}, \langle\eta, i\rangle] = \langle\beta', j''\rangle$

(ii) $\mathfrak{B}[\underset{\sim}{D}, \langle a(K, v[\underset{\sim}{D}, \eta], \eta), i\rangle] = \langle\beta'', j''\rangle$

Here too, we have $c(\ell, \beta') = c(K, \beta'')$, $c(K, \beta') = c(K, \eta)$, $c(\ell, \beta'') = c(\ell, \eta)$ and $\langle\beta', j''\rangle = {}_{\{v_K, v_\ell\}}\langle\beta'', j''\rangle$.

Now from the hypotheses of Theorem 9, $\vdash_{\underset{\sim}{D}} \mathfrak{A} \simeq \mathfrak{B}$, so that in particular, $\langle\alpha', j'\rangle = \langle\beta'', j''\rangle$, i.e., $\alpha' = \beta''$ and $j' = j''$. What we are now is that $\langle\alpha', j'\rangle = \langle\beta', j''\rangle$. Well, we have $j' = j''$, so all that remains is $\alpha' = \beta'$. We proceed as follows: $\langle\alpha', j'\rangle = {}_{\{v_K, v_\ell\}}\langle\alpha', j'\rangle = \langle\beta'', j''\rangle = {}_{\{v_K, v_\ell\}}\langle\beta', j''\rangle$. Thus, we have only to show that $c(K, \alpha') = c(K, \beta')$ and $c(\ell, \alpha') = c(\ell, \beta')$. But $c(K, \alpha') = c(K, \eta) = c(K, \beta')$ and $c(\ell, \alpha') = c(K, \alpha'') = c(K, \beta'') = c(\ell, \beta')$. Thus, taken all together, these results give $\langle\alpha', j'\rangle = \langle\beta', j''\rangle$. Since this is true for all $i<m$ and all $\eta: \omega \to \underset{\sim}{D}_o$, we have $\vdash_{\underset{\sim}{D}}(v_K ::= v_\ell)\mathfrak{A} \simeq (v_K ::= v_\ell)\mathfrak{B}$, i.e., $\vdash_{\underset{\sim}{D}}(v_K ::= v_\ell)\mathfrak{A} \simeq \mathfrak{B}$, as required.

This covers all the cases. It is also easy to verify that the substitutions $(u_i ::= \tau_i)_{i<n}$ may be done simultaneously without damaging the results so far proven for the single substitution case. ∎

Figure 17 illustrates several examples of proper instantiation, and in the next chapter, this syntactic operation will be incorporated into our inferential system as a rule of inference.

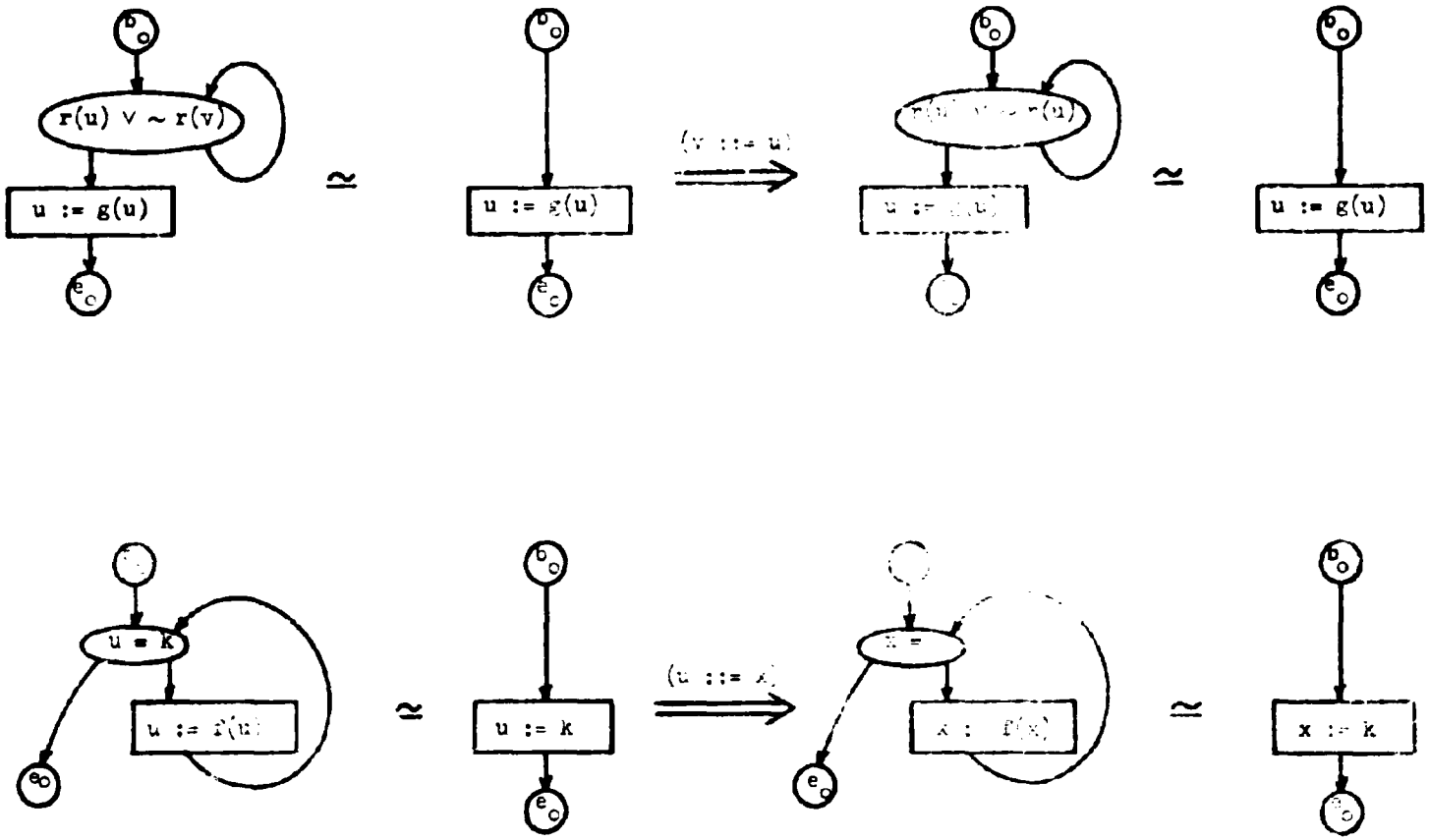


Figure 17

Here, u, v, x are variables; f, g, are function letters; r is a relation letter, and k is a constant. The right-hand E-programs result from the left-hand E-programs through the proper instantiation indicated. Continued next page.

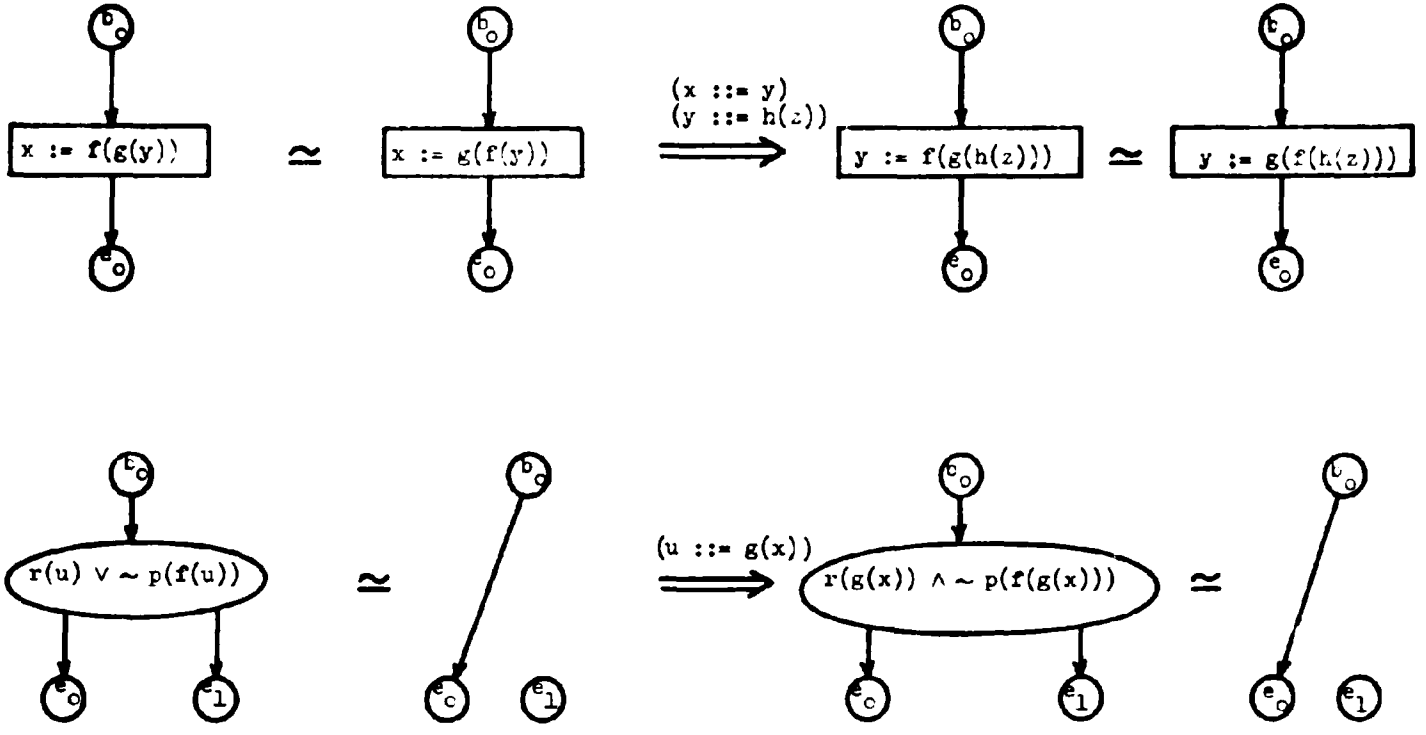


Figure 17 (contd.)

Here, x, y, z, u are variables; r, p are relation letters; and f, g, h are function letters. The right-hand E-programs result from the left-hand E-programs through the proper instantiations indicated.

Remarks:

(i) The notions of "occurrence" and of a variable or term "occurring" in some formula has been left informal in this discussion. This is simply because intuition alone is an excellent guide in these matters, not because the formal definition is intractable. Feferman [10] treats a similar matter for the predicate calculus.

(ii) The roles of substitution, scope and freedom in this work seem curiously similar to related matters in the predicate calculus. For example, see Mendelson [33, pp. 48 and 53]. There is likely more in this than at first meets the eye.

Composition, Decomposition and Replacement of E-programs

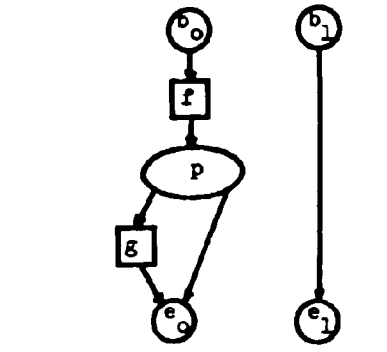
Before considering transformations on E-programs which alter their topological structure, we must consider the syntactic operations of composition and decomposition of E-programs. In Chapter 5 we leaned heavily on an intuitive understanding of how the graphs of E-programs could be combined to form the graph of the composition of these, but a few further details are in order to make these ideas more formal.

Roughly speaking, a composition of two E-programs $\mathcal{M}$ and $\mathcal{N}$ to form a new E-program $\mathcal{C}$ is accomplished simply by pairing in a 1-1 fashion some terminators of $\mathcal{N}$ with initiators of $\mathcal{M}$, and then joining $\mathcal{M}$ and $\mathcal{N}$ together at these points and simultaneously eliminating these terminator-initiator pairs. To assure that the result of this composition is in fact an E-program we require there be $m < \omega$ initiators and $n < \omega$ terminators remaining, and that these be relabelled $b_0, b_1, \ldots, b_{m-1}$ and $e_0, e_1, \ldots, e_{n-1}$ respectively. This rather

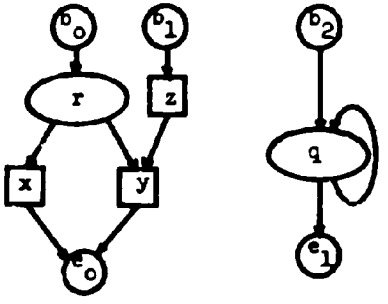
loose description can be made more precise in terms of the set theoretic definition of E-programs, but no real benefit is to be gained by such an endeavor. Instead, we use the example of composition in Figure 18 to illustrate the details.

Roughly speaking, a <u>decomposition</u> of an E-program $\mathfrak{C}$ into two E-programs $\mathfrak{A}$ and $\mathfrak{B}$ is accomplished simply by interposing a number of new terminator-initiator pairs between nodes of $\mathfrak{C}$, and breaking $\mathfrak{C}$ apart of these places. Of course, proper attention must be paid to the labelling of the new initiators and terminators to assure well-formedness of $\mathfrak{A}$ and $\mathfrak{B}$. This rather loose description can also be made more precise in terms of the set-theoretic definition of E-programs, but rather than introducing such opacity, we will simply say that $\mathfrak{C}$ can be decomposed into $\mathfrak{A}$ and $\mathfrak{B}$ if there exists a composition of $\mathfrak{A}$ and $\mathfrak{B}$ to form $\mathfrak{C}$. Figure 19 illustrates a simple example of decomposition.

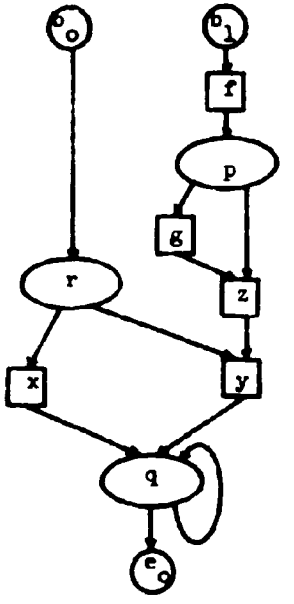
If $\mathfrak{C}$ can be decomposed into two E-programs, one of which is $\mathfrak{A}$, we may say that $\mathfrak{A}$ is a <u>sub-program</u> of $\mathfrak{C}$ and write $\mathfrak{C}(\mathfrak{A})$ instead of $\mathfrak{C}$ to indicate this. If $\mathfrak{A}$ and $\mathfrak{B}$ are E-programs of the same type, then we say that the E-program $\mathfrak{C}(\mathfrak{B})$ arises from $\mathfrak{C}(\mathfrak{A})$ through <u>replacement</u> of sub-program $\mathfrak{A}$ by $\mathfrak{B}$, provided that the composition that forms $\mathfrak{C}(\mathfrak{A})$, and the composition that forms $\mathfrak{C}(\mathfrak{B})$, are identical. Figure 20 illustrates a simple example of replacement.



E-program 𝔄



E-program 𝔅



E-program ℭ

Figure 18

A composition of E-programs 𝔄 and 𝔅 to form E-program ℭ . Here, f, g, x, y, z are assignment schemata; and p, q, r are qffs.

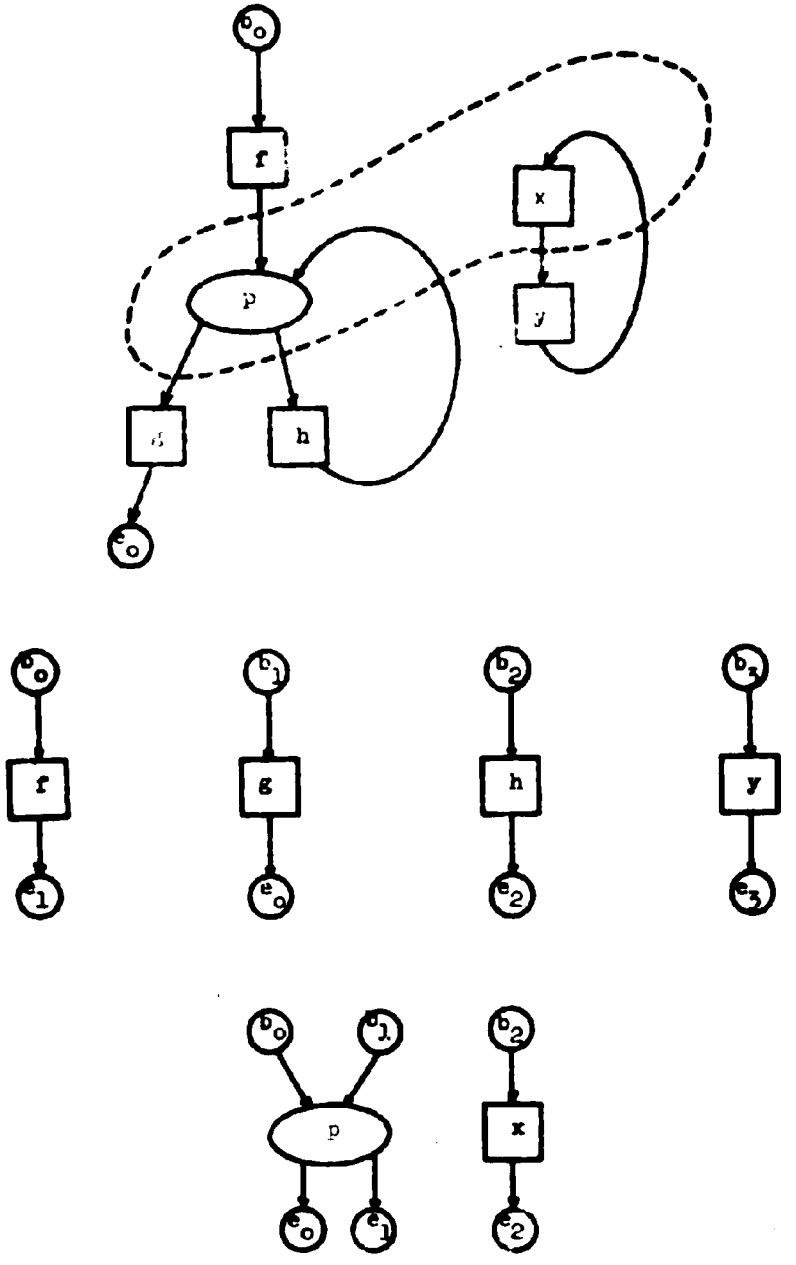
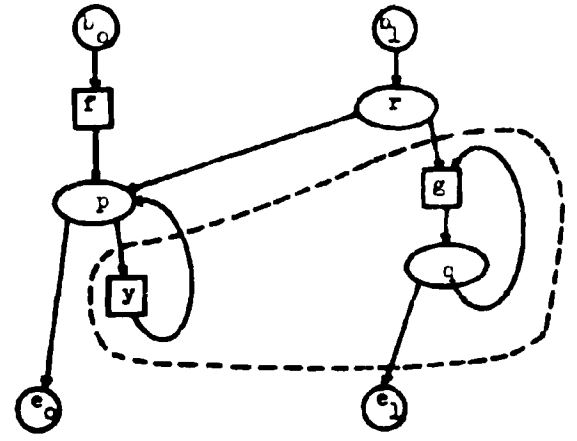
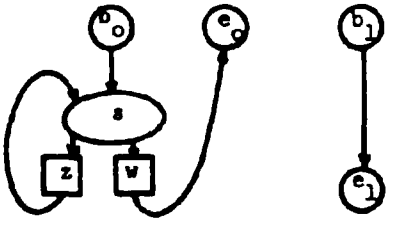


Figure 1b

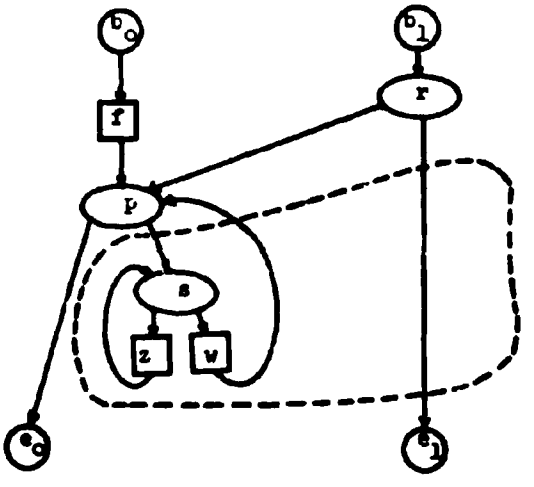
Here, f, g, h, x, y are assignment schemata, and p is a qff. The lower two E-programs result from the indicated decomposition of the uppermost E-program.



E-program $\mathfrak{A}(\mathfrak{M})$



E-program $\mathfrak{B}$



E-program $\mathfrak{C}(\mathfrak{B})$

Figure 20

Here, f, g, w, z, y are assignment schemata, and p, q, r, s are qffs. E-program $\mathfrak{C}(\mathfrak{B})$ arises from $\mathfrak{C}(\mathfrak{B})$ through replacement of sub-program $\mathfrak{M}$ by $\mathfrak{B}$.

We bring together the notions of composition, decomposition and replacement in the following

Theorem 10: For any signature s, E-programs $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}(\mathfrak{A})$ and $\mathfrak{C}(\mathfrak{B})$ in L_s, where $\mathfrak{A}$ and $\mathfrak{B}$ are of the same type and $\mathfrak{C}(\mathfrak{B})$ arises from $\mathfrak{C}(\mathfrak{A})$ by replacement of sub-program $\mathfrak{B}$ by $\mathfrak{A}$,

(i) for all computing structures $\underset{\sim}{D}$ with signature s,

$\models_{\underset{\sim}{D}} \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \models_{\underset{\sim}{D}} \mathfrak{C}(\mathfrak{A}) \simeq \mathfrak{C}(\mathfrak{B})$, or alternatively

(ii) $\{\mathfrak{A} \simeq \mathfrak{B}\} \models \mathfrak{C}(\mathfrak{A}) \simeq \mathfrak{C}(\mathfrak{B})$.

Proof: Since $\mathfrak{C}(\mathfrak{A})$ and $\mathfrak{C}(\mathfrak{B})$ are formed using the same composition, if during the execution of $\mathfrak{C}(\mathfrak{A})$, $\mathfrak{A}$ is entered where b_i was and left where e_j was, thereby producing a certain result, then since $\models_{\underset{\sim}{D}} \mathfrak{A} \simeq \mathfrak{B}$ by hypothesis, during the execution of $\mathfrak{C}(\mathfrak{B})$, $\mathfrak{B}$ will be entered where b_i was and left where e_j was, and will thereby produce the same result. Thus, $\mathfrak{C}(\mathfrak{A})$ and $\mathfrak{C}(\mathfrak{B})$ are equivalent in $\underset{\sim}{D}$ whenever $\mathfrak{A}$ and $\mathfrak{B}$ are.

Remarks:

(i) Theorem 10 will serve as the basis for a replacement rule of inference in the inferential system defined in the next chapter.

(ii) The substitutivity properties of equivalence and strong equivalence, along with their obvious symmetry, characterize them as equivalence relations in the ordinary sense.

CHAPTER 7

THE INFERENTIAL SYSTEM

In Chapter 1, we introduced several basic ideas regarding a formal theory of strong equivalence and we will continue here to develop along those lines. In Chapter 4, we introduced the formal theory $\mathcal{T}_s = \langle \mathcal{F}m_s, \mathcal{A}_s \rangle$, and in Chapters 4, 5 and 6 the set $\mathcal{F}m_s$ of wffs has been studied. In this chapter, the inferential system $\mathcal{I}_s = \langle \mathcal{A}x_s, \mathcal{K} \rangle$ is presented and its soundness demonstrated.

The inferential system $\mathcal{I}_s = \langle \mathcal{A}x_s, \mathcal{K} \rangle$ consists of an effectively decidable set $\mathcal{A}x_s \subseteq \mathcal{F}m_s$ of wffs called the <u>axioms</u> and a finite set $\mathcal{K}$ of rules of <u>inference</u>. For any set $\Delta \subseteq \mathcal{F}m_s$, we write $\Delta \vdash \mathfrak{A} \simeq \mathfrak{B}$ iff the wff $\mathfrak{A} \simeq \mathfrak{B}$ is <u>finitely derivable</u> from $\Delta \cup \mathcal{A}x_s$ using $\mathcal{K}$. By this we mean that there exists a finite sequence of wffs $\theta_0, \theta_1, \ldots, \theta_{n-1}$ such that for all $i < n$, either $\theta_i \in \Delta \cup \mathcal{A}x_s$ or θ_i can be inferred from $\theta_0, \ldots, \theta_{i-1}$ by some rule in $\mathcal{K}$, and θ_{n-1} is $\mathfrak{A} \simeq \mathfrak{B}$.

The rules of inference in $\mathcal{K}$ will be prescribed independently of the signature s. In the same spirit, we will define $\mathcal{A}x_s$ by first defining a finite set $\mathcal{A}x$ of <u>axiom schemata</u>, also independent of s, and then $\mathcal{A}x_s = \{\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}m_s : \mathfrak{A} \simeq \mathfrak{B}$ is an instance of some axiom schema in $\mathcal{A}x\}$.

If $\mathcal{O} \subseteq \mathcal{F}m_s$ is a set of wffs such that for any wff $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{O}$ and any $\Delta \subseteq \mathcal{O}$ we have $\Delta \models \mathfrak{A} \simeq \mathfrak{B} \Leftrightarrow \Delta \vdash \mathfrak{A} \simeq \mathfrak{B}$, then we say that the inferential system $\mathcal{I}_s$ is <u>extended complete</u> for $\mathcal{O}$; if we have only $\models \mathfrak{A} \simeq \mathfrak{B} \Leftrightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}$, then we say $\mathcal{I}_s$ is <u>complete</u> for $\mathcal{O}$. Notice that

extended completeness implies completeness. As to be expected, whether or not $\mathcal{A}_S$ is complete or extended complete for some $\mathcal{J} \subseteq \mathcal{M}_S$ depends on the signature s, the set $\mathcal{J}$ and, of course, the sets $\mathcal{M}$ and $\mathcal{R}$. In addition, we will say that the theory $\mathcal{T}_S = \langle \mathcal{M}_S, \mathcal{A}_S \rangle$ is complete (or extended complete) if $\mathcal{A}_S$ is complete (or extended complete) for $\mathcal{F}m_S$.

Before we formulate $\mathcal{A}_S$ and study its properties, we should note that there are definite limitations on the axiomatizability of strong equivalence, i.e., on the existence of an inferential system complete for $\mathcal{F}m_S$. We say that the formal theory $\mathcal{T}_S = \langle \mathcal{M}_S, \mathcal{A}_S \rangle$ is <u>axiomatic</u> if its inferential system $\mathcal{A}_S = \langle \mathcal{A}x_S, \mathcal{R} \rangle$ is <u>effective</u>, i.e., $\mathcal{A}x_S \subseteq \mathcal{F}m_S$ is effectively decidable and the rules in $\mathcal{R}$ are effectively applicable. This means that we can always recognize when a wff is an instance of an axiom schema, and we can always determine if the proposed application of a rule of inference is legitmate.

In Chapter 5, we showed that the strong equivalence problem for wffs in $\mathcal{F}m_S^-$, where $s = \langle 1, 1, 1 \rangle$, is unsolvable (Theorem 4). But the minimal language L_S^-, where recall qffs of the form $(\tau = \sigma)$ have been suppressed, suffers from another unfortunate malaise which must serve as a basic restriction in our attempts at formulating an axiomatic complete theory of strong equivalence. Consider the following

<u>Theorem 10: For the signature $s = \langle 1, 1, 1 \rangle$, there exists no effective inferential system complete for $\mathcal{F}m_S^-$.</u>

Thus, for any signature $s = \langle \langle n_0, \ldots, n_{k-1} \rangle, \langle m_0, \ldots, m_{\ell-1} \rangle, p \rangle$, if $k > 0$, $\ell > 0$ and $p > 0$, any effective inferential system is of necessity incomplete. This precludes, then, any axiomatic complete formal theory of strong equivalence for $\mathcal{E}$-programs in systems with any appreciable computing power.

Proof: In the proof of Theorem 4, we established an isomorphism between the set $A \subseteq \mathcal{F}m_s^-$ of generally valid wffs of the form $\mathfrak{M}^*(\xi_f, \underset{\sim}{d}) = \mathfrak{N}$, where $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$ and the set B of pairs $\langle \xi_f, \underset{\sim}{\Phi} \rangle$ such that $f(d_0, \ldots, d_{n-1})$ does not exist. Now, consider the set C of all pairs $\langle \xi_f, \underset{\sim}{\Phi} \rangle$ and the set $D \subseteq C$ of all pairs $\langle \xi_f, \underset{\sim}{\Phi} \rangle$ such that $f(d_0, \ldots, d_{n-1})$ does exist. Now D is certainly effectively enumerable, for example, by computing each $f(d_0, \ldots, d_{n-1})$ a little bit, infinitely often, and noting when one produces a final value. But, recall that D is not effectively decidable (i.e., the set membership decision problem for D is unsolvable). Then, certainly $B = C - D$ is not effectively enumerable, since the effective enumerability of both B and D where $B \cup D = C$, would imply an existance decision procedure for any $f(d_0, \ldots, d_{n-1})$, namely, perform the enumerations of B and D until $\langle \xi_f, \underset{\sim}{\Phi} \rangle$ comes up; if $\langle \xi_f, \underset{\sim}{\Phi} \rangle \in D$, $f(d_0, \ldots, d_{n-1})$ does not exist and if $\langle \xi_f, \underset{\sim}{\Phi} \rangle \in D$ then $f(d_0, \ldots, d_{n-1})$ does exist. This implies D is effectively decidable, a contradiction. So we have that B is not effectively enumerable.

But then neither is $A \subseteq \mathcal{F}m_s^-$ effectively enumerable, since A and B are isomorphic. Suppose that there exists an effective inferential system complete for $\mathcal{F}m_s^-$, i.e., we have $\vdash \mathfrak{M} = \mathfrak{N} \Leftrightarrow \vdash \mathfrak{M} = \mathfrak{N}$ for all $\mathfrak{M} = \mathfrak{N} \in \mathcal{F}m_s^-$. Then, since the inferential system is effective, we can

enumerate all the theorems, which by completeness yields an enumeration of all the generally valid wffs in $A \subseteq \mathcal{F}m_s^- \subseteq \mathcal{F}m_s$, a contradiction. Thus, no effective inferential system complete for $\mathcal{F}m_s^-$ exists. ∎

Remarks:

(i) In spite of the pessimism that this incompleteness result is likely to engender, we can nevertheless take heart in the several areas for which $\mathcal{A}_s$ is both effective and complete and even extended complete. We will take these matters up in Chapter 8 after introducing the sets $\mathcal{A}x$ and $\mathcal{R}$.

(ii) Of course, if we accept Church's thesis, we can replace "effective" with "recursive" in the above discussion. Then, $\langle \xi_f, \underset{\sim}{d} \rangle$ would be a "sequence number" or "gödel number" generated in an appropriate manner from the form ξ_f and sequence $\underset{\sim}{d} = \langle d_0, \ldots, d_{n-1} \rangle$.

The Axioms and Rules of Inference

There are fifteen axiom schemata: $\mathcal{A}x = [\underline{A1}, \underline{A2}, \ldots, \underline{A15}]$, and five rules of inference: $\mathcal{R} = [\underline{R1}, \underline{R2}, \ldots, \underline{R5}]$.

First, we give the rules which characterize "$=$" as an equivalence relation in the ordinary sense.

$\underline{R1}$: $\mathfrak{A} = \mathfrak{B} \Rightarrow \mathfrak{B} = \mathfrak{A}$

$\underline{R2}$: $\mathfrak{A} = \mathfrak{B} \Rightarrow \mathfrak{C}(\mathfrak{A}) = \mathfrak{C}(\mathfrak{B})$ where $\mathfrak{C}(\mathfrak{B})$ arises from $\mathfrak{C}(\mathfrak{A})$ through the replacement of $\mathfrak{A}$ by $\mathfrak{B}$.

Then, to permit derivations from hypotheses, we have

$\underline{R3}$: $\mathfrak{A} = \mathfrak{B} \Rightarrow (u_i ::= \tau_i)_{i<n}\, \mathfrak{A} = \mathfrak{B}$.

The axiom schemata $\underline{A1}, \underline{A2}, \ldots, \underline{A7}$ characterize the properties of qffs, and are illustrated in Figure 21. Also in Figure 21 is a rule $\underline{R3}'$ which is just a particularization of $\underline{R3}$, we mention it here because it reflects the instantiation properties of qffs.

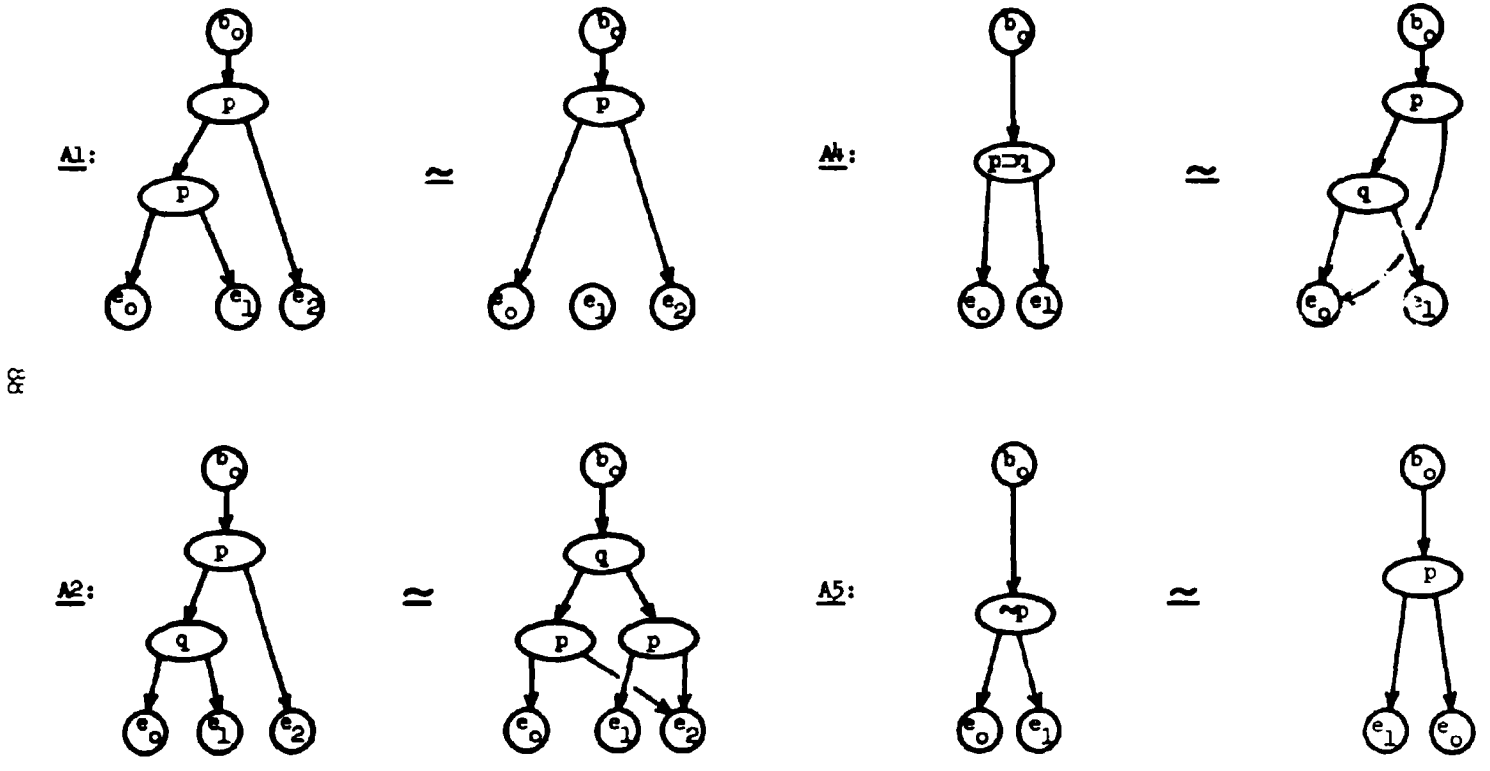


Figure 21

Axiom schemata that characterizes qffs. Here p, q are qffs. Continued next page.

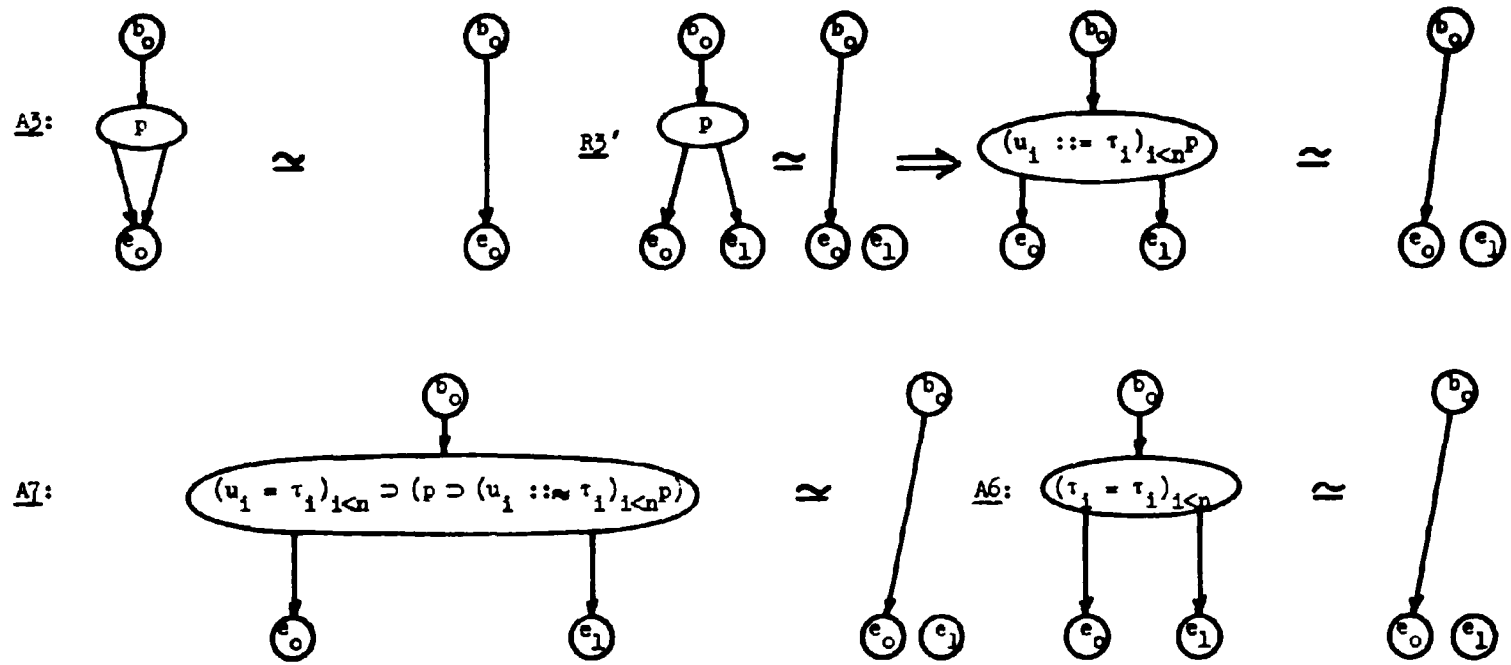


Figure 21 contd.

Axiom schemata and rule that characterize qffs. Here, p is a qff; u_i, $i<n$, are distinct variables; and τ_i, $i<n$, are terms.

The axiom schemata A8, A9, A10 and the rule R4 characterize the properties of assignment schemata, and are illustrated in Figure 22.

Axiom schema A11 characterizes the effect of operators on discriminators, and is illustrated in Figure 23(a). When this axiom schema is applied, we will sometimes say that the operator f is "pushed through" the discriminator p.

The three axiom schemata, A12, A13 and A14, provide a characterization of the graph-theoretic properties of E-programs. In what follows, $\mathfrak{M} = \langle X, \Gamma, \Sigma \rangle$ is assumed to be a type $\langle m, n \rangle$ E-program. First, we have

A12 : $\mathfrak{M} \simeq \Phi_i(\mathfrak{M})$

where $i < m$ and $\Phi_i(\mathfrak{M})$ is called the i-th separation of $\mathfrak{M}$. Roughly speaking, $\Phi_i(\mathfrak{M})$ is formed as follows

(i) A copy $\mathfrak{M}_i$ is made of the sub-program of $\mathfrak{M}$ whose nodes are reachable via Γ from the node labelled with b_i.

(ii) Then $\mathfrak{M}_i$ is composed with $\mathfrak{M}$ so that the node labelled with b_i now leads into $\mathfrak{M}_i$.

An instance of A12 is shown in Figure 24(a), and this should make matters intuitively clear. However, a demonstration is given in Figure 24(b), which shows, albeit in schematic form, the decomposition and composition undertaken to obtain the instance of A12 given in Figure 24(a). When this axiom is applied, we will sometimes say $\Phi_i(\mathfrak{M})$ is obtained by "separation".

Notice that separation gives rise to extraneous nodes in $\Phi_i(\mathfrak{M})$ which are not reachable from any initiator node. Axiom schema A13 does away with such nodes. We have,

A13 : $\mathfrak{M} \simeq \Theta(\mathfrak{M})$,

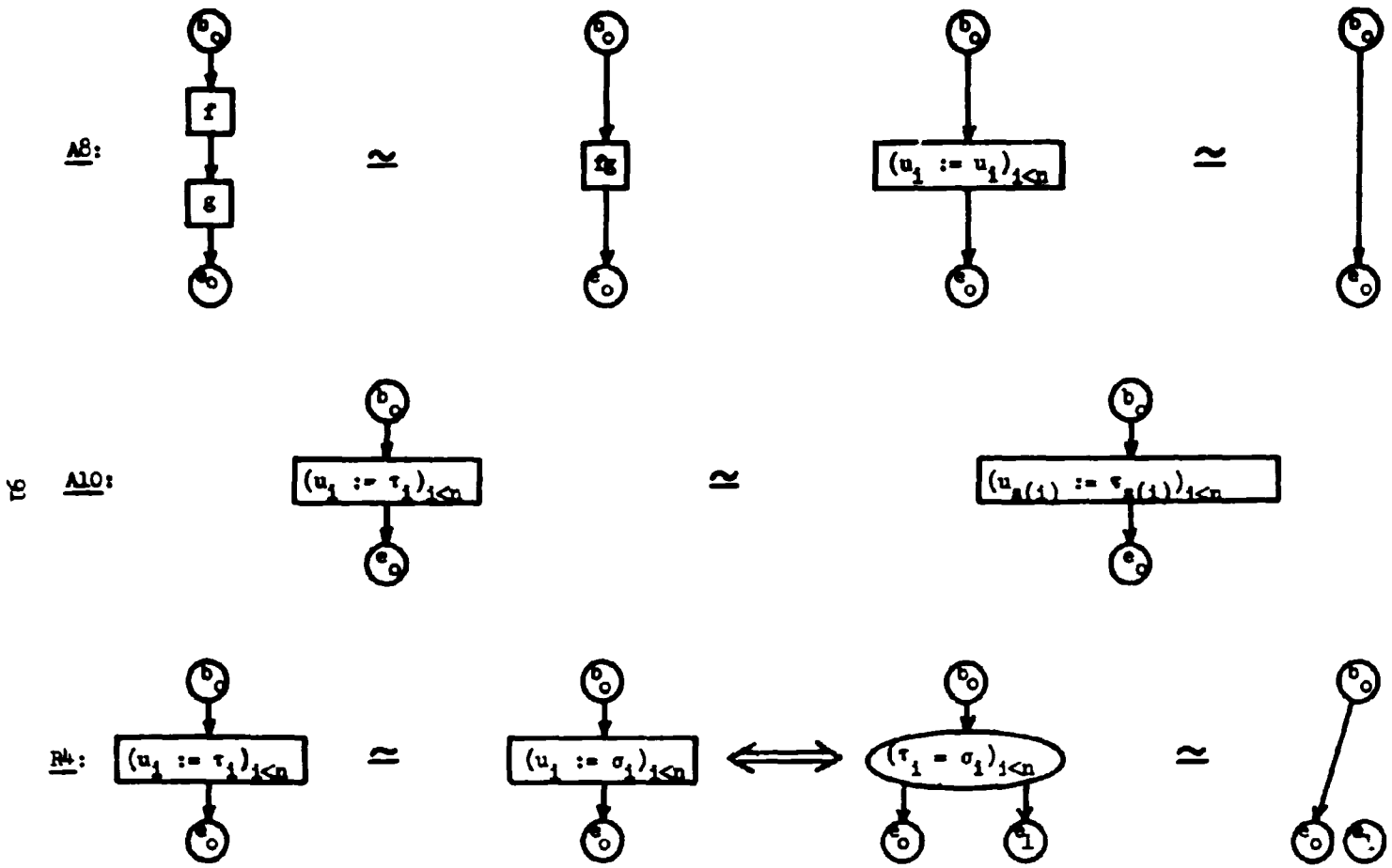


Figure 22

The axiom schemata and rule that characterize assignment schemata. Here, f and g are assignment schemata; u_i, $i<n$, are distinct variables; τ_i, σ_i, $i<n$, are terms; and $s : n \to n$ is any 1-1 onto permutation.

A11:

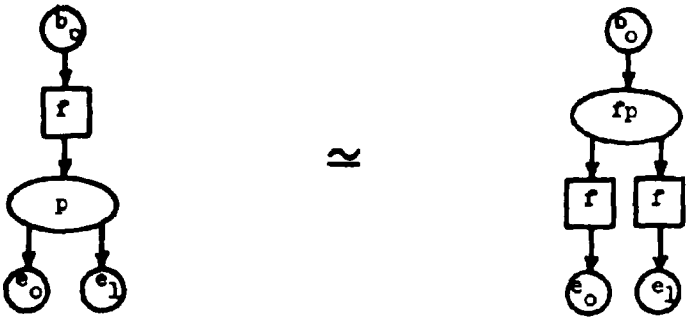


Figure 23(a)

The push through axiom schema. Here f is an assignment schema and p a qff.

A15:

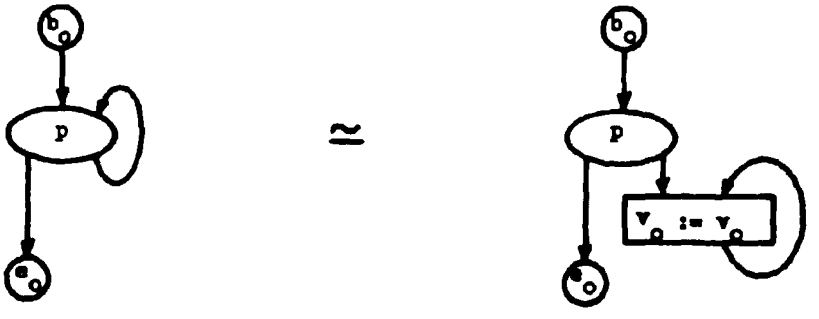


Figure 23(b)

Here p is a qff.

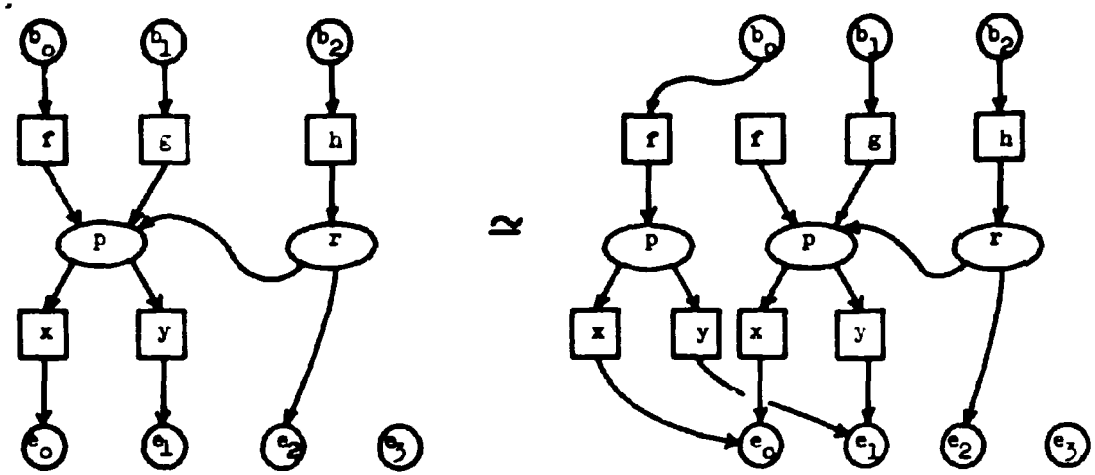
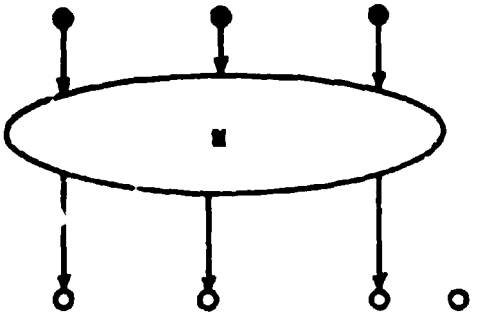
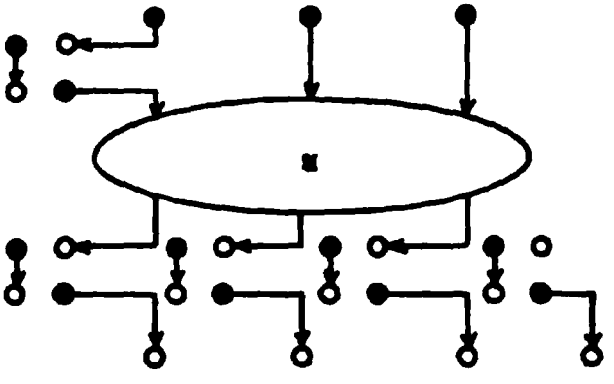


Figure 24(a)

An instance of the separation axiom schema, A12: $\mathfrak{A} = \Phi_0(\mathfrak{A})$. Here f, g, h, x, y are assignment schemata and p, r. are qffs.



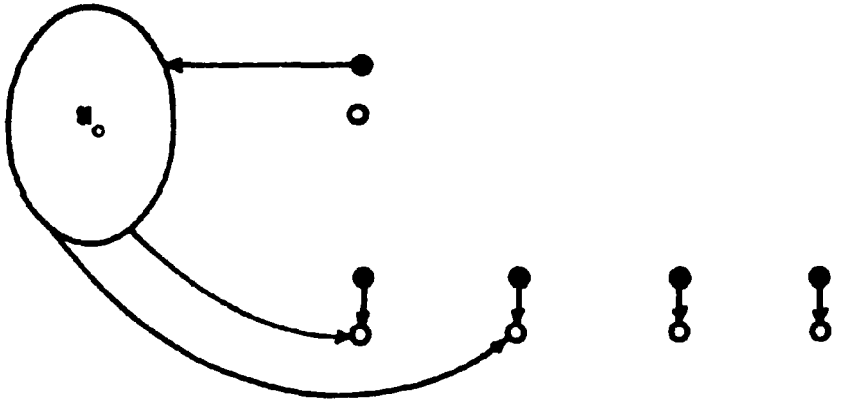
Schematic form of E-program M . ● denotes an initiator, and O a terminator.



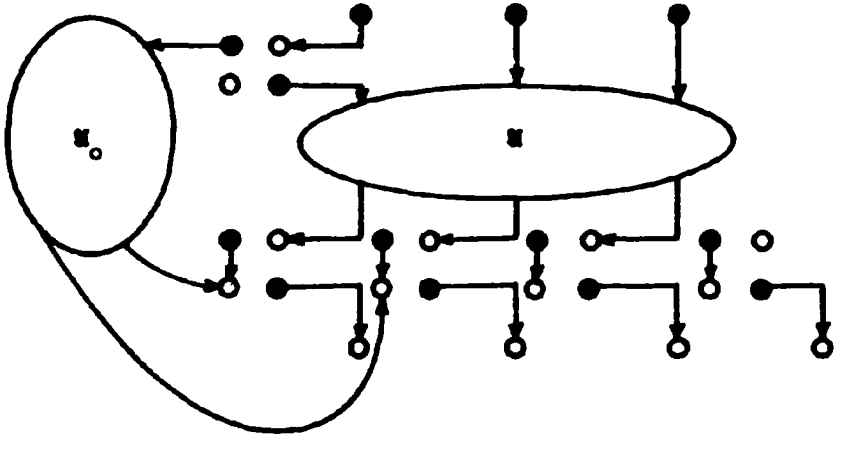
M is decomposed into a type <8, 9> E-program and a type <5, 5> "null" E-program.

Figure 24(b)

Continued next page.



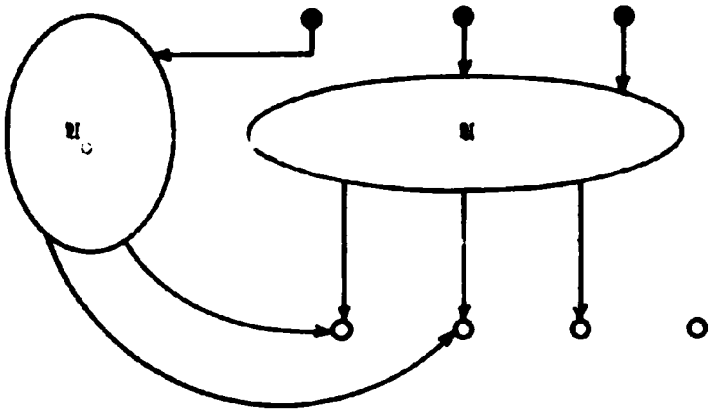
Schematic of the sub-program $\mathfrak{A}_o$.



The "null" E-program is then replaced with sub-program $\mathfrak{A}_o$.

Figure 24(b) contd.

Continued next page.



The final composition produces the 0-th separation of $\mathfrak{A}$, $\Phi_0(\mathfrak{A})$.

Figure 2.4(b) contd.

where $\Theta(\mathfrak{M}) = \langle X', \Gamma', \mathcal{L}' \rangle$, $X' = X(\mathfrak{B}) \cup X(\mathfrak{E}) \cup \{x \in X: x$ is reachable via Γ from some $y \in X(\mathfrak{I})\}$, Γ' is Γ restricted to X', and $\mathcal{L}'$ is $\mathcal{L}$ restricted to X'. An instance of A13 is shown in Figure 24(c).

If a certain section of an E-program is a cul-de-sac, i.e., has no exit to a terminator, then we shall want to detect this at a graph-theoretic level. We have,

A14 : $\mathfrak{M} \simeq \Omega_i(\mathfrak{M})$

where $i < m$. Roughly speaking, $\Omega_i(\mathfrak{M})$ is formed as follows.

(i) A copy $\mathfrak{M}_i$ is made of the sub-program of $\mathfrak{M}$ whose nodes are reachable via Γ from the node labelled with b_i, as in the case of A12 when forming $\Theta_i(\mathfrak{M})$.

(ii) If at least one of the terminator nodes of $\mathfrak{M}_i$ is reachable from the initiator node of $\mathfrak{M}_i$ corresponding to that of $\mathfrak{M}$ labelled with b_i, then $\Omega_i(\mathfrak{M}) = \mathfrak{M}$, i.e., we make no changes.

(iii) If none of the terminator nodes of $\mathfrak{M}_i$ is reachable from the initiator node of $\mathfrak{M}_i$ corresponding to that of $\mathfrak{M}$ labelled b_i, then a special always indeterminate E-program $\mathfrak{N}$ of the same type as $\mathfrak{M}_i$ is composed with $\mathfrak{M}$ so that the node labelled with b_i now leads into $\mathfrak{N}$.

An instance of A14 is shown in Figure 24(d), and this should make matters intuitively clear; the special always indeterminate E-program $\mathfrak{N}$ is also illustrated there. However, a step by step demonstration is given in Figure 24(e) which shows, again in schematic form, the decomposition and composition undertaken to obtain the instance of A14 given in Figure 24(d).

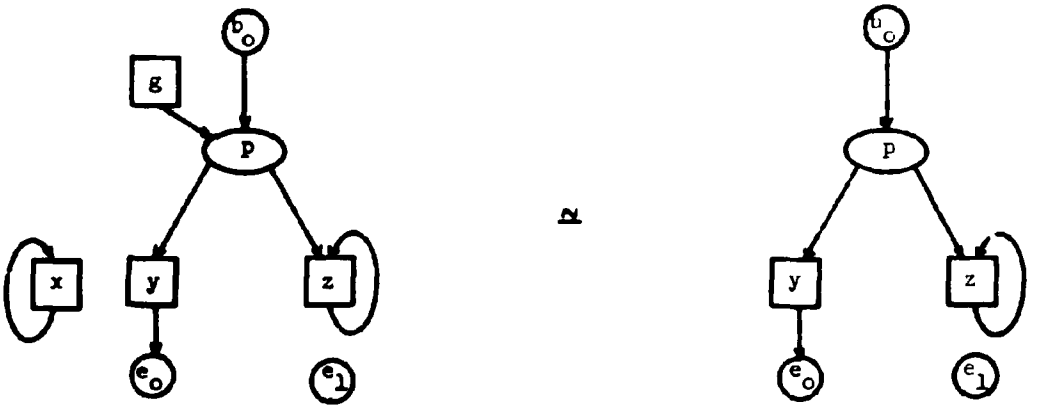


Figure 24(c)

An instance of A13. Here, g, x, y, z are assignment schemata, and p is a qff.

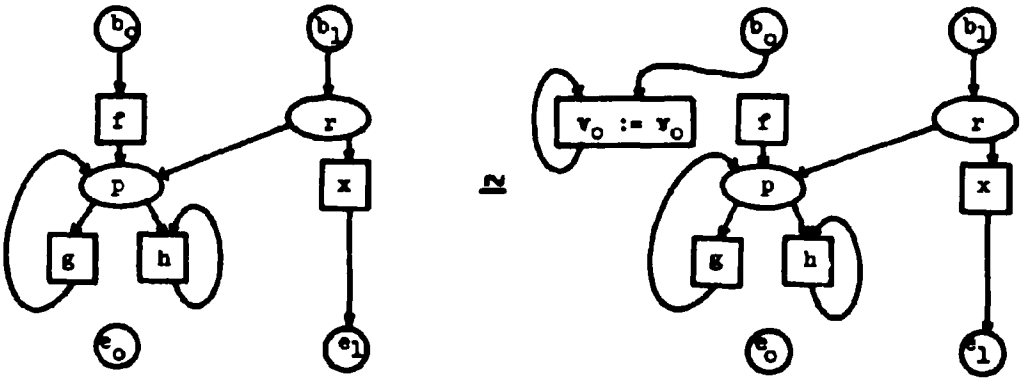
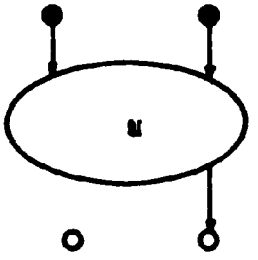
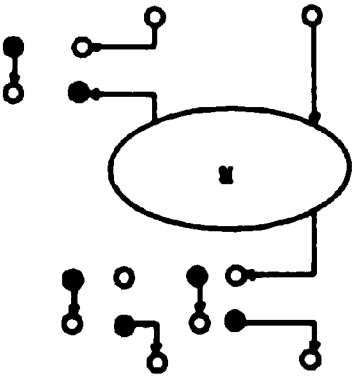


Figure 24(d)

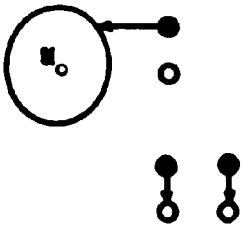
An instance of axiom schema A14: $\mathfrak{A} \simeq \Omega_0(\mathfrak{A})$. Here, f, g, h, x are assignment schemata, and p, r are qffs.



Schematic form of E-program 𝔐 .



𝔐 is decomposed into a type <5, 5> E-program and a type <3, 3> "null" E-program.



Sub-program $\mathfrak{M}_o$ is a cul-de-sac as is that at right.

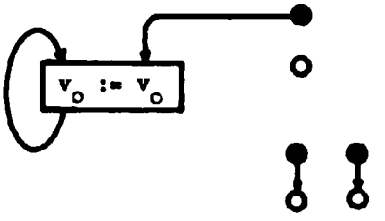
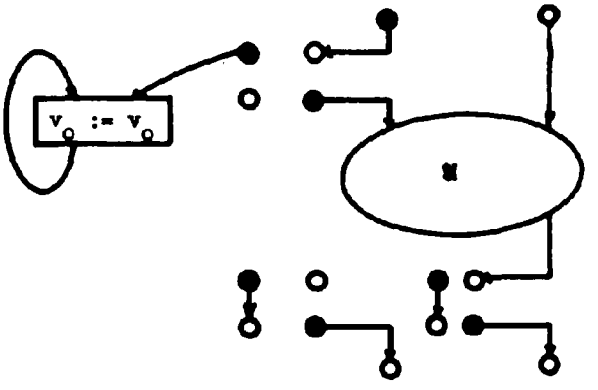
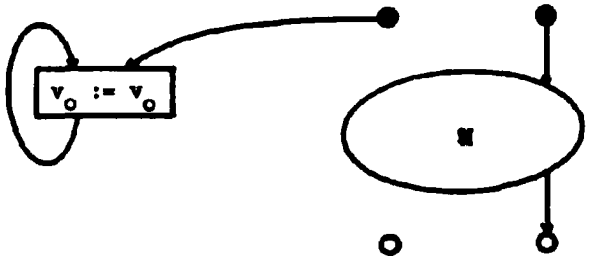


Figure 24(e) contd.

Continued next page.



The "null" E-program is replaced with the special always indeterminate E-program.



The result is $\Omega_0(M)$.

Figure 24(e) contd.

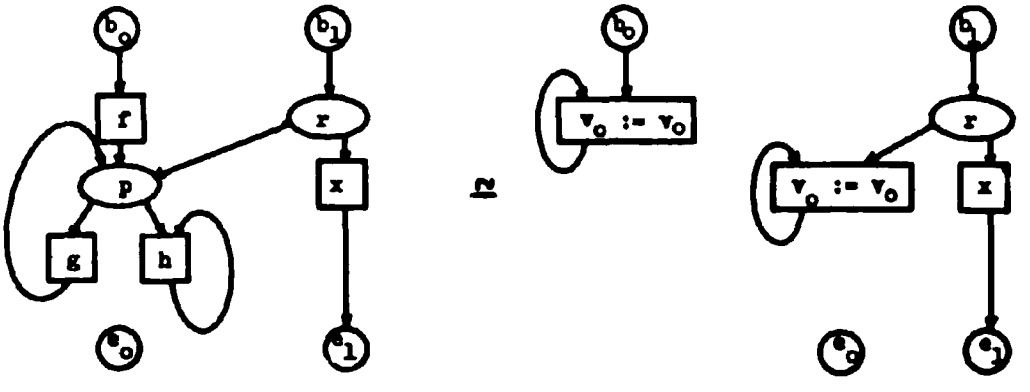


Figure 24(f)

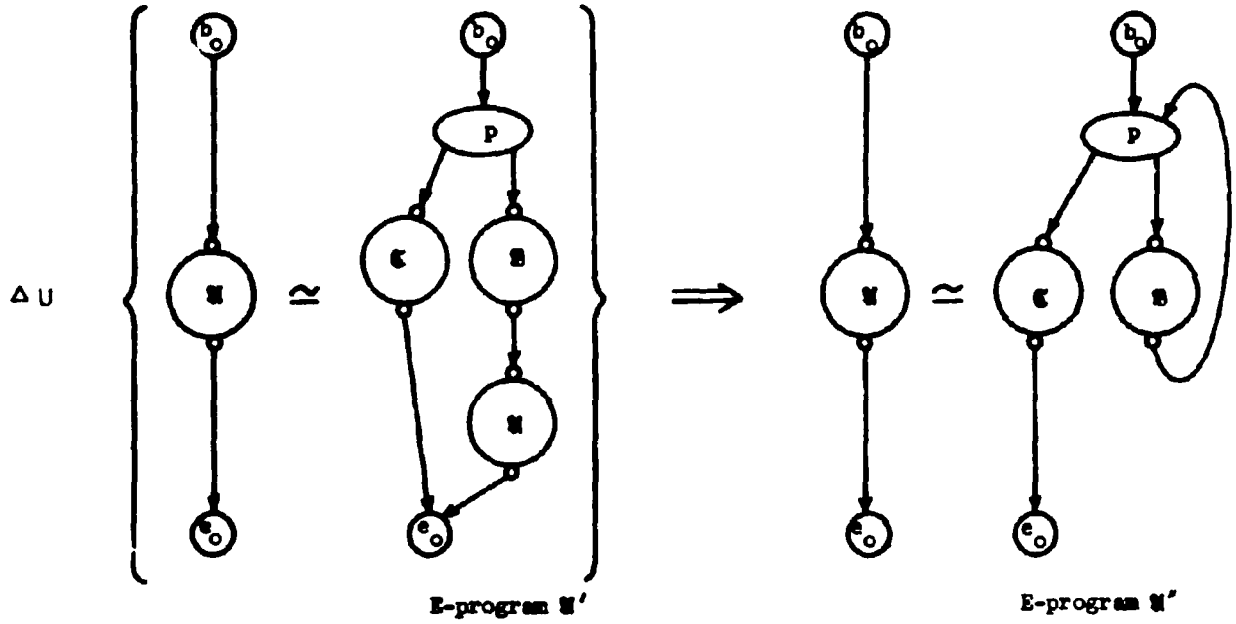
Simplification of the left-hand E-program of Figure 24(d) using A13 and A4.

Of course, further simplification of the right-hand member of the wff in Figure 24(e) can be made by applying A14 to the appropriate subprogram, and then applying A13 to remove the isolated component. Figure 24(f) shows the final result.

Another sort of cul-de-sac is detected by axiom schema A15. As illustrated in Figure 23(b), we are dealing here with logical, as opposed to topological indeterminacy. This sort of construct can be used to define pseudo-qffs with "undefined" as an additional truth-value.

The last rule of inference is R5 and is called the recursion rule. Before giving the general statement of the recursion rule, we consider a restricted instance thereof which illustrates, in a simple manner, most of the salient points. This restricted instance, which we refer to as R5′, and which is illustrated in Figure 25(a), allows us to infer a recursive closed form for an E-program, which is initially defined iteratively, i.e., not in closed form. As we shall see in Chapter 8, even in this restricted form, the recursion rule is a powerful derivational tool and is essential if certain derivations concerning E-programs with loops are to be made.

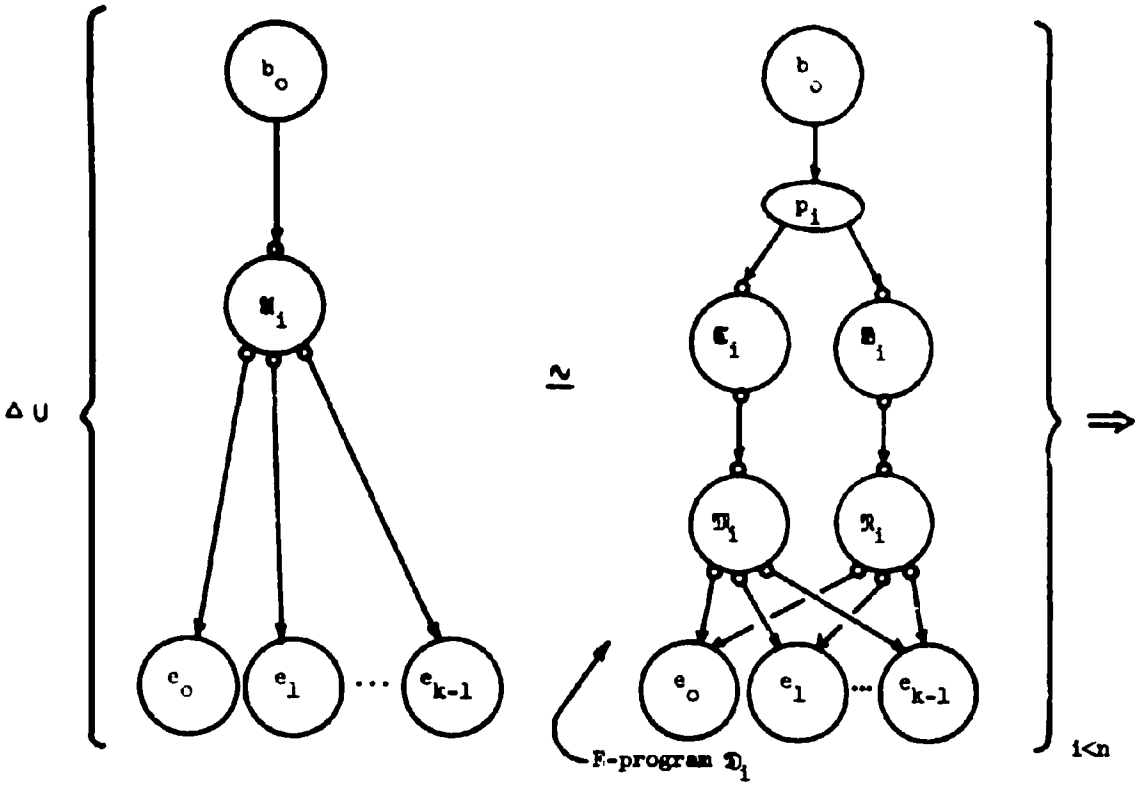
The restricted rule R5′ is actually that used by McCarthy [28] when discussing "recursion induction" for flowcharts. Thus, if two E-programs $\mathfrak{M}_1$ and $\mathfrak{M}_2$ satisfy the "equation" in the premise of R5′, then $\mathfrak{M}_1 \simeq \mathfrak{M}_2$, since by R5′ both $\mathfrak{M}_1$ and $\mathfrak{M}_2$ are strongly equivalent to the recursive, or closed form, E-program inferred using R5′.

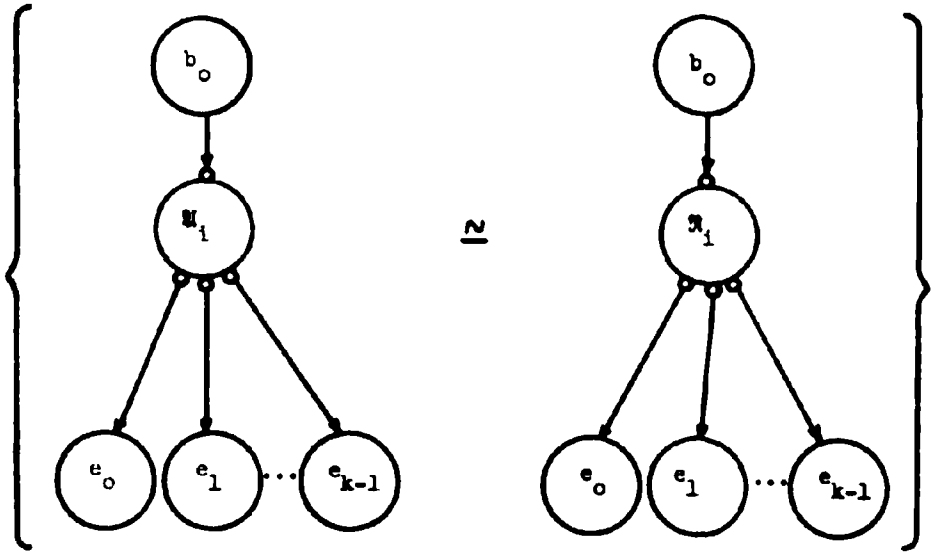


provided that in any computing $\underset{\sim}{D}$ in which the wffs of Δ and $\mathfrak{M} = \mathfrak{M}'$ are valid, then for all $\xi : \omega \to \underset{\sim}{D}_o$, if $\mathfrak{M}'[\underset{\sim}{D}, \langle \xi, \mathcal{O} \rangle]$ is indeterminate, then so is $\mathfrak{M}[\underset{\sim}{D} \langle \xi, \mathcal{O} \rangle]$.

Figure 25(a)

The recursion rule, R5′. Here p is any qff and $\mathfrak{M}$, $\mathfrak{B}$, $\mathfrak{C}$ and type $\langle 1, 1 \rangle$ E-programs.

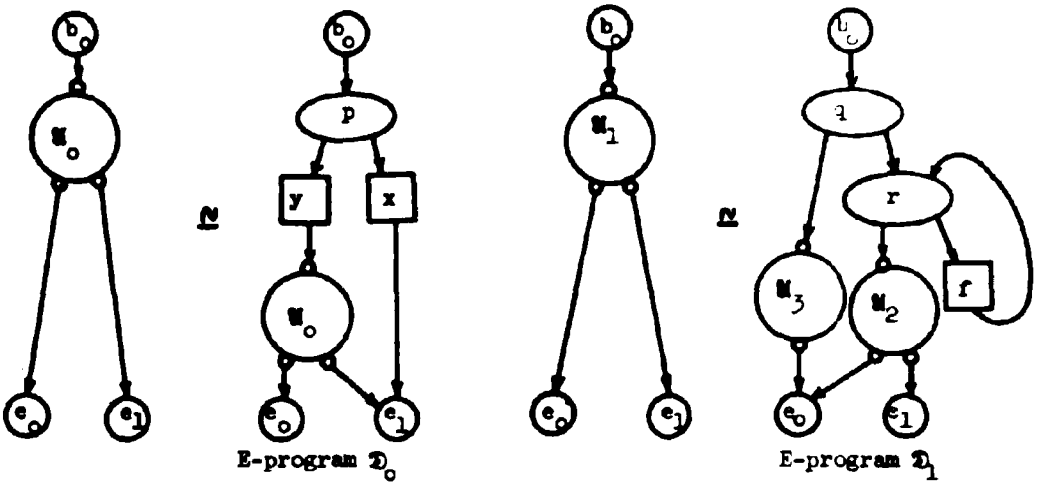




where, for each $i<n$, $\mathfrak{M}_i$, $\mathfrak{N}_i \in \{\mathfrak{M}_i\}_{i<n} \cup$ {the k distinct type $\langle 1, k\rangle$ null E-programs}, $\mathfrak{C}_i$, $\mathfrak{B}_i$ are any type $\langle 1, n\rangle$ E-programs, and $\mathfrak{N}_i$ is the i-th closed form composition (defined in the text) of the E-programs in $\{\mathfrak{D}_i\}_{i<n}$; provided that in any computing structure $\mathfrak{D}$ in which the proper axioms in Δ, and the wffs in $\{\mathfrak{M}_i \simeq \mathfrak{D}_i\}_{i<n}$ are valid, then for all $\xi : \omega \to D$, and $i<n$, if $\mathfrak{N}_i[\mathfrak{D}, \langle \xi, O\rangle]$ is indeterminate, so is $\mathfrak{M}_i[\mathfrak{D}, \langle \xi, O\rangle]$, or alternatively, if $\mathfrak{M}_i[\mathfrak{D}, \langle \xi, O\rangle]$ is determinate, so is $\mathfrak{N}_i[\mathfrak{D}, \langle \xi, O\rangle]$.

Figure 25(b)

The recursion rule R5.



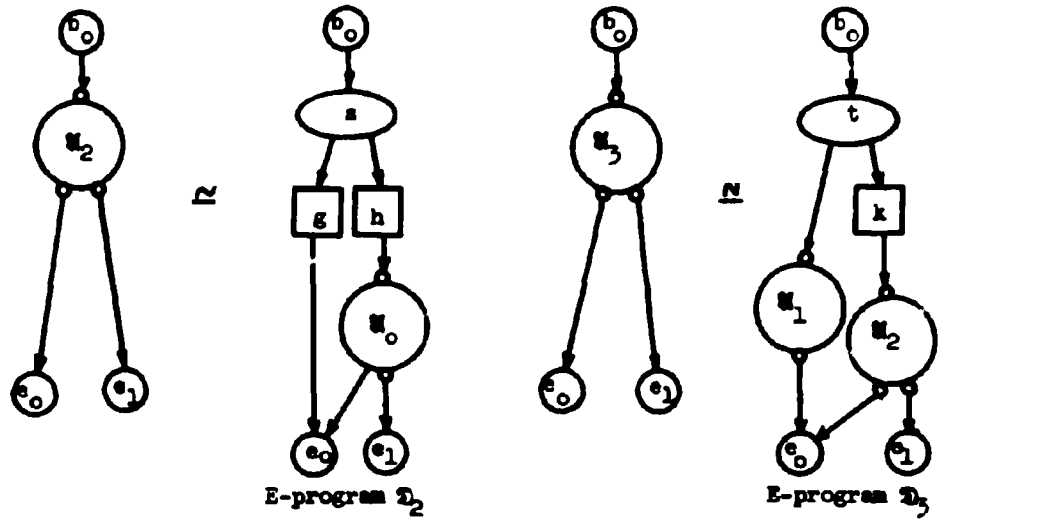


Figure 25(c)

Here $n = 3$, and the wffs $\{\mathfrak{A}_i \simeq \mathfrak{D}_i\}_{i<3}$ are shown. As well, p, q, r, s, t are qffs; and f, g, h, k, x, y are assignment schemata. This figure is continued on the next page.

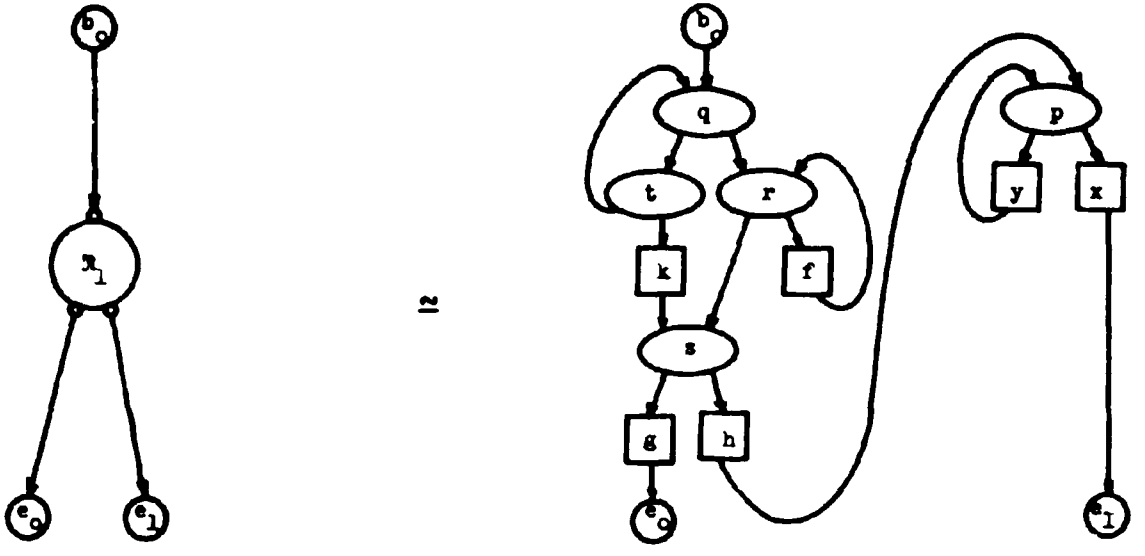


Figure 25(c) contd.

An example showing the 2nd closed form composition of $\mathfrak{D}_0$, $\mathfrak{D}_1$, $\mathfrak{D}_2$ and $\mathfrak{D}_3$.

The general form of the recursion rule R5 is illustrated in Figure 25(b). The ith closed form composition referred to in Figure 25(b) is formed from the E-programs $\{\mathfrak{D}_i\}_{i<n}$ in a simple way. For each $\mathfrak{D}_i$, $i<n$, suppose that $\mathfrak{R}_i$ is $\mathfrak{A}_{m_i}$ and $\mathfrak{T}_i$ is $\mathfrak{A}_{n_i}$ for some $m_i, n_i < n$, since recall $\mathfrak{R}_i$, $\mathfrak{T}_i$ may be members of $\{\mathfrak{A}_i\}_{i<n}$. Then, we delete, in $\mathfrak{D}_i$, the entries into $\mathfrak{A}_{m_i}$ and $\mathfrak{A}_{n_i}$, and instead set up branches from $\mathfrak{S}_i$ and $\mathfrak{B}_i$ directly to the discriminators p_{m_i} and p_{n_i} respectively in the E-programs $\mathfrak{D}_{m_i}$ and $\mathfrak{D}_{n_i}$. We retain one set of k terminators, and for any $\mathfrak{R}_i$ (or $\mathfrak{T}_i$) that is a null type $\langle 1, k\rangle$ E-program, i.e., a single branch from the initiator to one of the terminators, we set up a branch from $\mathfrak{S}_i$ (or $\mathfrak{B}_i$) to the appropriate terminator. Of the k initiators, we retain only b_i, but relabel it b_o. The result of all these operations is the ith closed form composition of the E-programs $\{\mathfrak{D}_i\}_{i<n}$. An example of this construction is given in Figure 25(c).

The recursion rule R5 can be used to implement recursion induction, in a generalized sense, for flowcharts. Suppose that $\vdash \mathfrak{A}_i \simeq \mathfrak{D}_i$, $i < n$, and $\vdash \mathfrak{A}_i' \simeq \mathfrak{D}_i$, $i < n$, and that the side conditions for application of R5 hold in each case. Then, $\vdash \mathfrak{A}_i \simeq \mathfrak{R}_i$ and $\vdash \mathfrak{A}_i' \simeq \mathfrak{R}_i$, $i < n$, by R5, and then $\vdash \mathfrak{A}_i \simeq \mathfrak{A}_i'$, $i < n$, by R1 and R2. Without referring to R5, we could use the terminology of recursion induction and say that since $\{\mathfrak{A}_i\}_{i<n}$ and $\{\mathfrak{A}_i'\}_{i<n}$ both "satisfy" the "equations" $X_i \simeq \mathfrak{D}_i$, $i < n$, and since both $\mathfrak{A}_i$ and $\mathfrak{A}_i'$ are "defined (i.e., halt) for the same arguments", then, by recursion induction $\vdash \mathfrak{A}_i \simeq \mathfrak{A}_i'$, $i < n$. The side condition "defined for all arguments" is, of course, equivalent to the side condition for R5.

It is straightforward to show that using separation (i.e., A12) and push through (i.e., A11), we can put any type <1, n> E-program into a form directly expressable as the 0-th closed form composition of a set $\{\mathfrak{D}_i\}_{i<N}$ of E-programs, where $\mathfrak{U}_i \simeq \mathfrak{D}_i$, $i < N$, is a set wffs like that required for R5, where $N < \omega$ is the number of qffs in the original E-program. Thus, we can prove the strong equivalence of any two type <1, n> E-programs using R5. But, proving the strong equivalence of any two type <m, n> E-programs can be achieved by reducing the problem, via separation, to proving the strong equivalence of m sets of pairs of type <1, n> E-programs, which we can do using R5. Thus, proving any two E-programs strongly equivalent can be carried out using recursion induction, i.e., using the recursion rule R5.

Of course, for certain signatures s, the question of whether the side condition for the application of R5 holds may, in general, be undecidable. However, the possible ineffectiveness of R5 usually presents no problem when carrying out relatively simple derivations as the side condition usually can be resolved. In fact, $\mathscr{L}_s$ being ineffective in general leaves open the possibility that $\mathscr{V}_s$ is complete for $\mathscr{FM}_s$; we will not pursue this possibility here, however.

Remarks:

(i) The axiom schemata A8, A9 and A10, as we shall see, correspond directly to the axioms mentioned in the foregoing chapter for the "assign" function a and "contents" function c.

(ii) The axiom schemata A1, A2 and A3 are similar to three axioms from a set of ten given by McCarthy [30] for conditional expressions. The rest

are obviated by the flowchart representation (as opposed to linear strings), and by the presence of a rule of replacement in our inferential system.

(iii) The inferential system $\mathcal{L}_s$ implies that given by Ianov [16], i.e., we can prove all of his axioms (and more, of course) in our system. More on this will appear in Chapter 9.

(iv) It is, of course, mildly unpleasant to have to use the assignment schema $v_o := v_o$ in the special always-indeterminate E-program $\mathfrak{A}$. This is akin to not having logical constants in the propositional calculus, and instead employing $p \vee \sim p$ and $p \wedge \sim p$ for truth and falsity. Ianov, faced with the same unpleasantness [16], used an identically false discriminator whose false branch returned to itself, and whose true branch proceeded to the exit of the program scheme.

Soundness of the Theory $\mathcal{T}_s$

If the theory $\mathcal{T}_s$ is to be useful at all, we should require that the theorems of $\mathcal{T}_s$ be generally valid, or at least semantic consequences of any hypotheses used in their derivation.

Theorem 11: The theory $\mathcal{T}_s$ is sound, i.e., for all $\Delta \subseteq \mathcal{F}_{\mathcal{L}_s}$ and all $\mathfrak{M} \simeq \mathfrak{N} \in \mathcal{F}_{\mathcal{L}_s}$, $\Delta \vdash \mathfrak{M} \simeq \mathfrak{N} \Rightarrow \Delta \models \mathfrak{M} \simeq \mathfrak{N}$.

Proof: It is sufficient to show that the axiom schemata in $\mathcal{A}x$ generate axioms in $\mathcal{A}x_s$ that are all generally valid, and that the rules of inference in $\mathcal{R}$ all preserve validity, i.e., if a rule is $\mathfrak{M} \simeq \mathfrak{N} \Rightarrow \mathfrak{M}' \simeq \mathfrak{N}'$ then we require $\{\mathfrak{M} \simeq \mathfrak{N}\} \models \mathfrak{M}' \simeq \mathfrak{N}'$.

Rule R1 : partial equality, "$\simeq$", in the definition of validity is symmetric.

Rule R2 : Theorem 10.

Rule R3 : Theorem 9.

Axiom schema A1: first consider an instance of the left hand member of A1 (cf. Figure 21(a)), $\mathfrak{A} = \langle X, \Gamma, \mathcal{L} \rangle$ say. Assume that $\underset{\sim}{D}$ is an arbitrary computing structure of the appropriate signature, and that $\xi : \omega \to \underset{\sim}{D}_o$ is an arbitrary input state. Then,

$\mathfrak{A}[\underset{\sim}{D}, \langle \xi, \Omega \rangle]$

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, u)$ where $u \in X$ and $[u] = b_o$

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, \Gamma u)$ by the definition of E

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, v)$ where $v \in X$, $[v] = p$ and $\Gamma v = \langle w, z \rangle$, $w, z \in X$

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, w)$ if $p[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{A}, \underset{\sim}{D}, \xi, z)$ if not $p[\underset{\sim}{D}, \xi]$, where $[w] = p$ and $[z] = e_2$, $\Gamma w = \langle x, y \rangle$, $x, y \in X$, by the definition of E

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, x)$ if $p[\underset{\sim}{D}, \xi]$ and $p[\underset{\sim}{D}, \xi]$, or
$E(\mathfrak{A}, \underset{\sim}{D}, \xi, y)$ if $p[\underset{\sim}{D}, \xi]$ and not $p[\underset{\sim}{D}, \xi]$, or
$E(\mathfrak{A}, \underset{\sim}{D}, \xi, z)$ if not $p[\underset{\sim}{D}, \xi]$, where $[x] = e_o$, $[y] = e_1$, by the definition of E

$= E(\mathfrak{A}, \underset{\sim}{D}, \xi, x)$ if $p[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{A}, \underset{\sim}{D}, \xi, z)$ otherwise

$= \langle \xi, \Omega \rangle$ if $p[\underset{\sim}{D}, \xi]$, or $\langle \xi, 2 \rangle$ otherwise, by the definition of E

Now, consider executing an instance of the right hand member of A1, $\mathfrak{B} = \langle X, \Gamma, \mathcal{L} \rangle$ say.

$\mathfrak{B}[\underset{\sim}{D}, \langle \xi, \Omega \rangle]$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, v)$ where $v \in X$ and $[v] = b_o$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, \Gamma v)$ by the definition of E

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, w)$ where $w \in X$, $[w] = p$ and $\Gamma w = \langle x, z \rangle$, $x, z \in X$

$= E(\mathfrak{B}, \underset{\sim}{D}, \xi, x)$ if $p[\underset{\sim}{D}, \xi]$, or $E(\mathfrak{B}, \underset{\sim}{D}, \xi, z)$ otherwise, where

$[x] = e_0$ and $[z] = e_2$, by the definition of E

$= \langle \xi, 0 \rangle$ if $p[\underset{\sim}{D}, \xi]$, or $\langle \xi, 2 \rangle$ otherwise, by the definition of E

Thus, for arbitrary $\underset{\sim}{D}$ and $\xi : \omega \to \underset{\sim}{D}_0$, we have

$\mathfrak{A}[\underset{\sim}{D}, \langle \xi, 0 \rangle] = \mathfrak{B}[\underset{\sim}{D}, \langle \xi, 0 \rangle]$, i.e., $\models \mathfrak{A} \simeq \mathfrak{B}$ for any instance $\mathfrak{A} \simeq \mathfrak{B}$ of A1.

Axiom schemata A2, A3, A4 and A5: that all instances of these axiom schemata are generally valid can be shown by arguments just as simple (and as tedious too) as that given for A1.

Axiom schema A6 and A7: these are the axiom schemata which characterize "=" as the equality relation (cf. Mendelson [33, p. 75]). Since $(\tau = \tau)[\underset{\sim}{D}, \xi]$ for any term τ, computing structure $\underset{\sim}{D}$ and state $\xi : \omega \to \underset{\sim}{D}_0$, then clearly A6 generates only generally valid wffs. Schema A7, however, requires somewhat more comment.

Consider the assignment schema $f = (v_{i_k} := \tau_k)_{k<n}$, computing structure $\underset{\sim}{D}$ and any state $\xi : \omega \to \underset{\sim}{D}_0$ such that $(v_{i_k} = \tau_k)_{k<n}[\underset{\sim}{D}, \xi]$. Then, from the semantics for qffs and terms, we have that $c(i_k, \xi) = \tau_k[\underset{\sim}{D}, \xi]$, $k<n$. Then, for all $j < \omega$,

$$v_j[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]] = v_j[\underset{\sim}{D}, \xi] = c(j, \xi)$$
$$\text{if } j \neq i_k \text{ for all } k<n$$
$$= \tau_k[\underset{\sim}{D}, \xi] = c(i_k, \xi) = c(j, \xi)$$
$$\text{if } j = i_k \text{ for some } k<n,$$

i.e., $f[\underset{\sim}{D}, \xi] = \xi$. Now, from Theorem 6, $fp[\underset{\sim}{D}, \xi] = p[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$, which from the above result gives $fp[\underset{\sim}{D}, \xi] = p[\underset{\sim}{D}, \xi]$, i.e.,

$$(v_{i_k} ::= \tau_k)p[\underset{\sim}{D}, \xi] = p[\underset{\sim}{D}, \xi].$$

Then from the semantics for qffs, $(p \supset (v_{i_k} ::= \tau_k)p)[\underset{\sim}{D}, \xi]$. But, from the

hypothesis on ξ, $(v_{i_k} = \tau_k)_{k<n}[D, \xi]$, so that, $((v_{i_k} = \tau_k)_{k<n} \supset (p \supset (v_{i_k} ::= \tau_k)p))[D, \xi]$. If we relax the substitution to only <u>some</u> of the occurrences of the v_{i_k}, this result still holds, and the general validity of the wffs generated by <u>A7</u> is obtained

Axiom schema <u>A8</u>: Theorem 8. Recall that the proof of Theorem 8 depended on the axioms for the "assign" and "contents" function, especially,

$a(i, k, a(i, \ell, \xi)) = a(i, k, \xi)$.

Axiom schema <u>A9</u>: follows immediately from the same axioms, especially

$a(i, c(i, \xi), \xi) = \xi$,

applied to the semantics for assignment schemata

Axiom schema <u>A10</u>: follows immediately from the same axioms, especially

$a(i, k, a(j, \ell, \xi)) = a(j, \ell, a(i, k, \xi))$, where $i \neq j$,

applied to the semantics for assignment schemata.

Rule <u>R4</u>: if the hypothesis is valid, then the values of the terms assigned to each variable must be equal. Hence, the rule preserves validity.

Axiom schema <u>A11</u>: Theorem 8.

Axiom schema <u>A12</u>: The separation $\Phi_i(\mathfrak{M})$ merely routes execution through a copy of the sub-graph of $\mathfrak{M}$ reachable from the node labelled b_i. Since no other nodes are reachable, and since $\mathfrak{M}_i$ is a copy of this sub-graph, the execution of $\mathfrak{M}$ and $\Phi_i(\mathfrak{M})$, starting at b_i, will be identical.

Axiom schema <u>A13</u>: Since none of the nodes deleted from $\mathfrak{M}$ to form $\Theta(\mathfrak{M})$ can be reached during execution of $\mathfrak{M}$, their absence in $\Theta(\mathfrak{M})$ will not cause $\Theta(\mathfrak{M})$ to execute any differently than $\mathfrak{M}$.

Axiom schema A14: Clearly replacing one cul-de-sac with another will not cause $\Omega_i(\mathfrak{M})$ to execute any differently than $\mathfrak{M}$.

Axiom schema A15: If the qff is initially false it will remain so, thus giving an indeterminate execution.

Rule R5′: Consider an arbitrary computing structure $\underset{\sim}{D}$ in which the wffs of Δ and $\mathfrak{M} \simeq \mathfrak{M}'$ are valid, and state $\xi : \omega \to \underset{\sim}{D}_o$. Now, consider $A = \mathfrak{M}'[\underset{\sim}{D}, \langle \xi, \mathcal{O} \rangle]$ and $B = \mathfrak{M}''[\underset{\sim}{D}, \langle \xi, \mathcal{O} \rangle]$ (cf. Figure 25(a) for $\mathfrak{M}'$ and $\mathfrak{M}''$).

Suppose that B halts; there are two possibilities.

(i) If $p[\underset{\sim}{D}, \xi]$, and $\mathfrak{C}[\underset{\sim}{D}, \langle \xi, \mathcal{O} \rangle]$ halts in $\mathfrak{M}''$, then in $\mathfrak{M}'$ the same will occur, and A will halt producing the same output state as B , i.e., $A = B$.

(ii) If not $p[\underset{\sim}{D}, \xi]$, and the loop is executed n times before $\mathfrak{C}$ is executed, then since $\vdash_{\underset{\sim}{D}} \mathfrak{M} \simeq \mathfrak{M}'(\mathfrak{M})$, we can perform n replacements of $\mathfrak{M}$ by $\mathfrak{M}''$ to give $\mathfrak{M}_n = \mathfrak{M}''(\mathfrak{M}''(\ldots(\mathfrak{M})\ldots))$. Then, $A_n = \mathfrak{M}_n[\underset{\sim}{D}, \langle \xi, \mathcal{O} \rangle]$ will behave just as A does, and in light of the n executions of the loop in $\mathfrak{M}''$, $A_n = B$, whence $A = B$.

Suppose that A halts. If B halts too, then we reason as above and obtain $A = B$. But, B cannot fail to halt, since by the side condition on R5, A fails to halt, a contradiction.

Thus A halts iff B halts, and in case they do halt, $A = B$. Since the state ξ is arbitrary, $\vdash_{\underset{\sim}{D}} \mathfrak{M}' \simeq \mathfrak{M}''$, and then $\vdash_{\underset{\sim}{D}} \mathfrak{M} \simeq \mathfrak{M}'$ gives $\vdash_{\underset{\sim}{D}} \mathfrak{M} \simeq \mathfrak{M}''$, as required.

Rule R5: By an argument similar to that for R5′. ∎

BLANK PAGE

CHAPTER 8

SOME COMPLETENESS RESULTS AND APPLICATIONS

So far, we have defined the formal theory $\mathcal{T}_s = \langle \mathcal{F}m_s, \mathcal{I}_s \rangle$, and demonstrated its soundness. In this chapter we isolate some of the sub-cases for which the inferential system $\mathcal{I}_s$ is complete or extended complete. In addition, various illustrative examples are given which demonstrate the utility and derivational power of the theory $\mathcal{T}_s$.

Extended Completeness for Single qffs

Let $\mathcal{F}m_s'(\mathfrak{A}) \subseteq \mathcal{F}m_s$ be the set of all wffs of the sort depicted in Figure 26, where p is any qff. (Note: we denote the rather trivial algorithm that serves as the right hand member for all wffs in $\mathcal{F}m_s'(\mathfrak{A})$ as $\dagger$.)

Theorem 12: *For any signature s, the inferential system $\mathcal{I}_s = \langle \mathcal{A}x_s, \mathcal{R} \rangle$ is extended complete for $\mathcal{F}m_s'(\mathfrak{A})$.*

Proof: We have $\Delta \vdash \mathfrak{M} := \dagger \Rightarrow \Delta \models \mathfrak{M} := \dagger$, for any $\Delta \subseteq \mathcal{F}m_s'(\mathfrak{A})$ and $\mathfrak{M} := \dagger \in \mathcal{F}m_s'(\mathfrak{A})$, from Theorem 11.

Therefore, we have only to show that $\Delta \models \mathfrak{M} := \dagger \Rightarrow \Delta \vdash \mathfrak{M} := \dagger$, and we do this by making use of the obvious analogy between the general validity of wffs in $\mathcal{F}m_s'(\mathfrak{A})$ and the logical validity of the qffs that occur in them. (Note: we denote validity in $\underset{\sim}{\mathfrak{D}}$, in the sense of PC_s, by $\models^*_{\underset{\sim}{\mathfrak{D}}}$ and logical validity by $\vdash^*$.) Thus, the wff $\mathfrak{M} := \dagger$ in Figure 26 is valid in $\underset{\sim}{\mathfrak{D}}$ iff $p[D, \xi]$ for all $\xi : \omega \to \underset{\sim}{D}_0$, which is equivalent to p being valid in $\underset{\sim}{\mathfrak{D}}$ in the sense of PC_s, i.e., $\models_{\underset{\sim}{\mathfrak{D}}} \mathfrak{M} := \dagger \Leftrightarrow \models^*_{\underset{\sim}{\mathfrak{D}}} p$. By extending this analogy with the predicate calculus, we have that $\Delta \models \mathfrak{M} := \dagger \Rightarrow \Delta' \vdash p$, where Δ' is the set of qffs occurring in the wffs of Δ.

Now, consider the qffs

(i) $p \supset (q \supset p)$

(ii) $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$

(iii) $(\sim q \supset \sim p) \supset ((\sim q \supset p) \supset q)$,

which occur in the wffs of Figure 27, and the qffs

(iv) $(\tau_i = \tau_i)_{i<n}$

(v) $(u_i = \tau_i)_{i<n} \supset (p \supset (u_i ::= \tau_i)_{i<n} p)$,

which occurs in axiom schemata A6 and A7 , together with the rules

(vi) modus ponens: $p,\ p \supset q \Rightarrow q$

(vii) particularization: $p \Rightarrow (u_i ::= \tau_i)_{i<n} p$,

where (vii) is obtained from R3′. The inferential system (i) through (vii) (an adaptation of that given by Mendelson [33], is known to be extended complete for the qffs of PC_s . Thus, if we mean by $\Delta \vdash^* r$ that the qff r finitely derivable from Δ using (i) through (vii) above, then $\Delta \vdash^* r \Leftrightarrow \Delta \vdash^* r$ for all such r and sets Δ of qffs.

In particular, we have that $\Delta \vdash \mathfrak{A} = \mathfrak{t} \Rightarrow \Delta' \vdash^* p \Rightarrow \Delta' \vdash^* p$. Now, the derivation of p from Δ' using the inferential system (i) through (vii) can be mimicked in a one-to-one fashion by a derivation of $\mathfrak{A} = \mathfrak{t}$ from Δ using the wffs of Figure 27, the axioms given by A6 and A7, the rule R3′ and the rule corresponding to modus ponens shown in Figure 28. Thus, if the wffs of Figure 27 and the rule in Figure 28 can be derived using ϑ_s , then $\Delta' \vdash^* p \Rightarrow \Delta \vdash \mathfrak{A} = \mathfrak{t}$, so that finally, $\Delta \vdash \mathfrak{A} = \mathfrak{t} \Leftrightarrow \Delta' \vdash^* p \Rightarrow \Delta' \vdash^* p \Rightarrow \Delta \vdash \mathfrak{A} = \mathfrak{t}$. A derivation of the wffs of Figure 27 using ϑ_s is given in Figure 29, and a derivation of the rule in Figure 28 is given in Figure 30. ∎

Figure 26

A member of $\mathcal{F}m'_s(\mathfrak{A})$. Here p is any qff of PC_S .

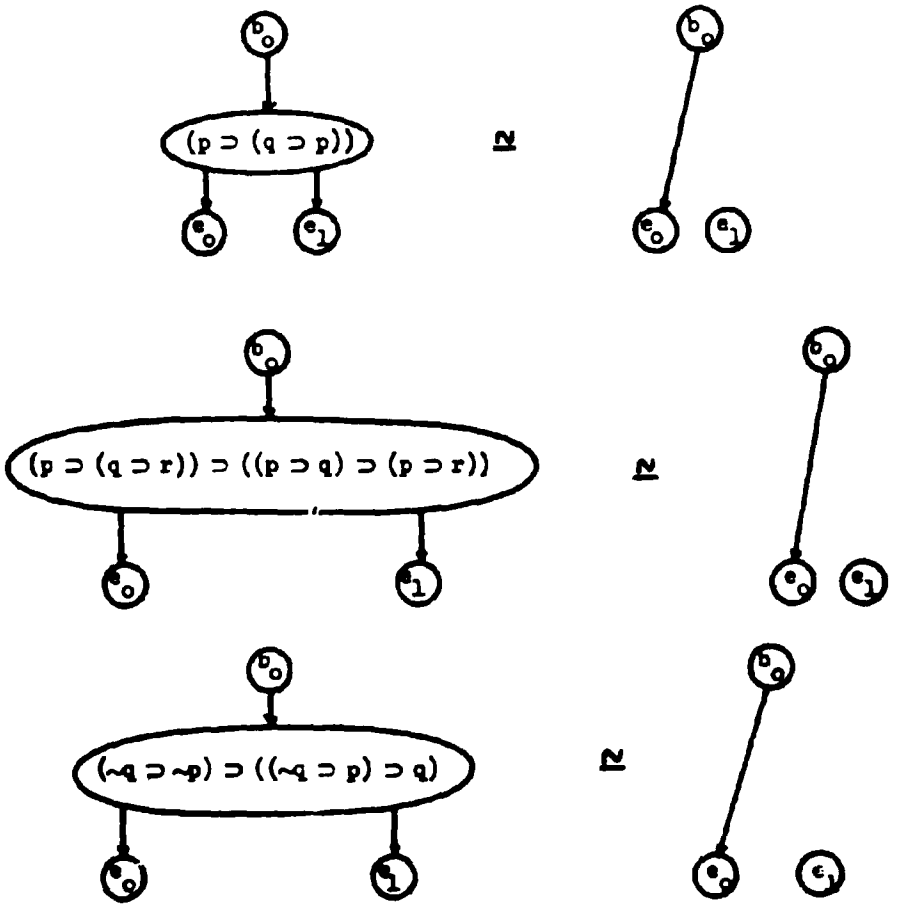


Figure 27

Three members of $\mathcal{F}m_s'(\mathfrak{A})$ corresponding to axioms for the propositional calculus. Here, p, q r are any qffs.

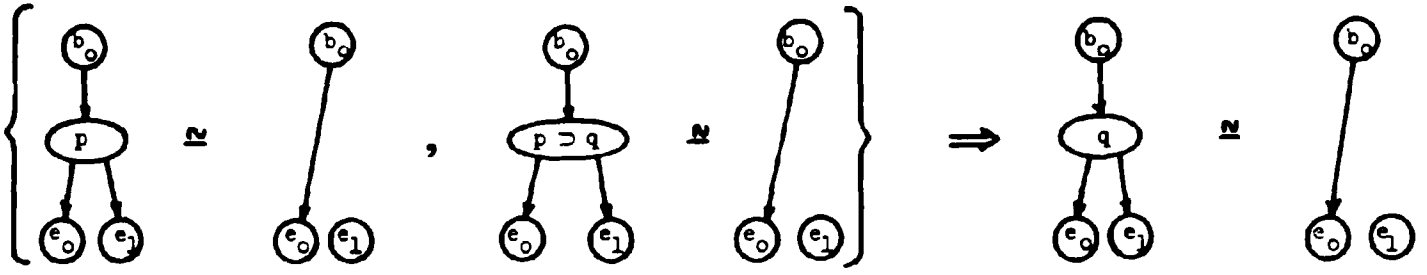


Figure 2b

A derived rule corresponding to modus ponens. Here, p and q are any qffs.

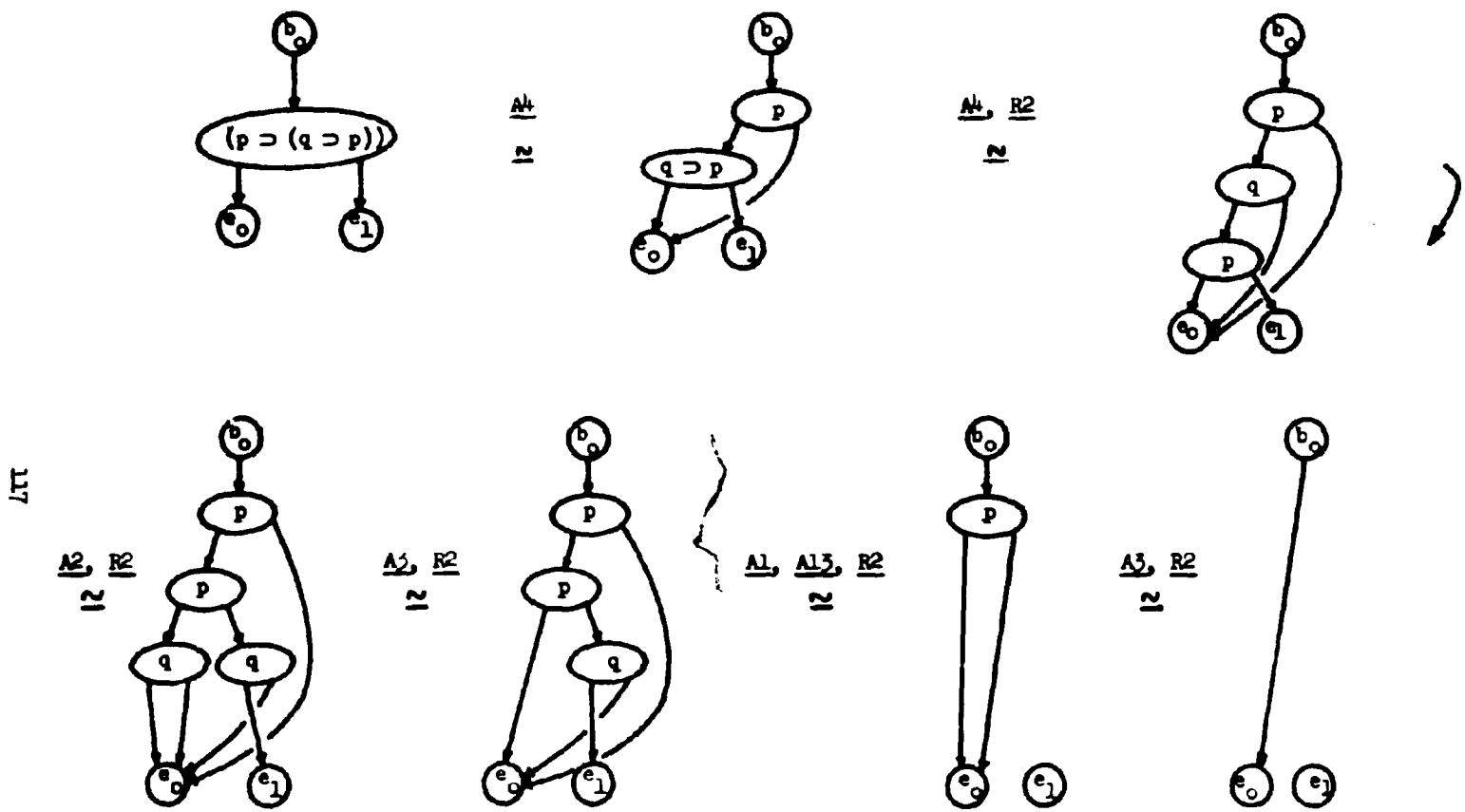


Figure 29(a)

Derivation of the first wff of Figure 27.

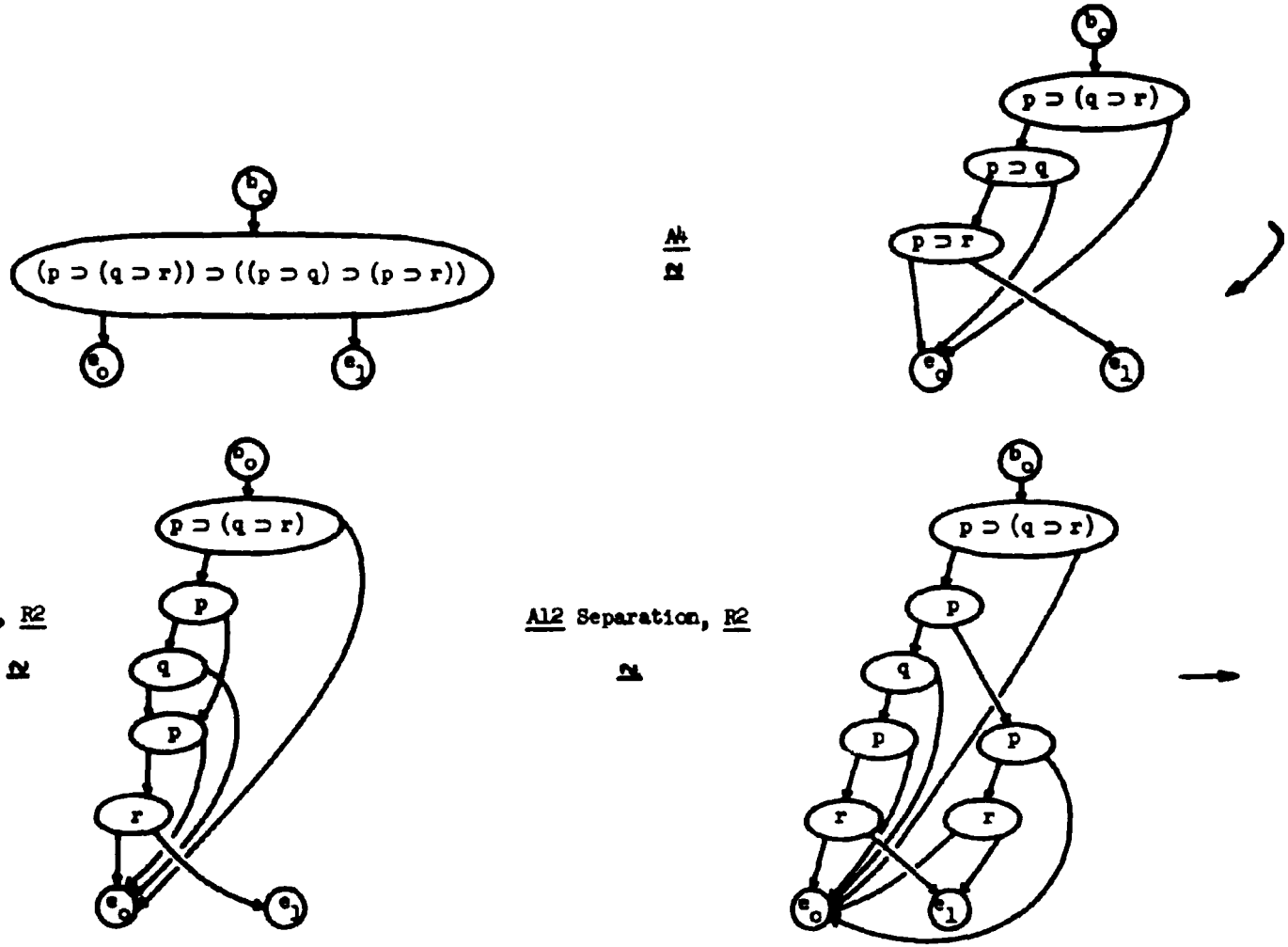


Figure 29(b)

First part of the derivation of the second wff of Figure 27. Continued next page.

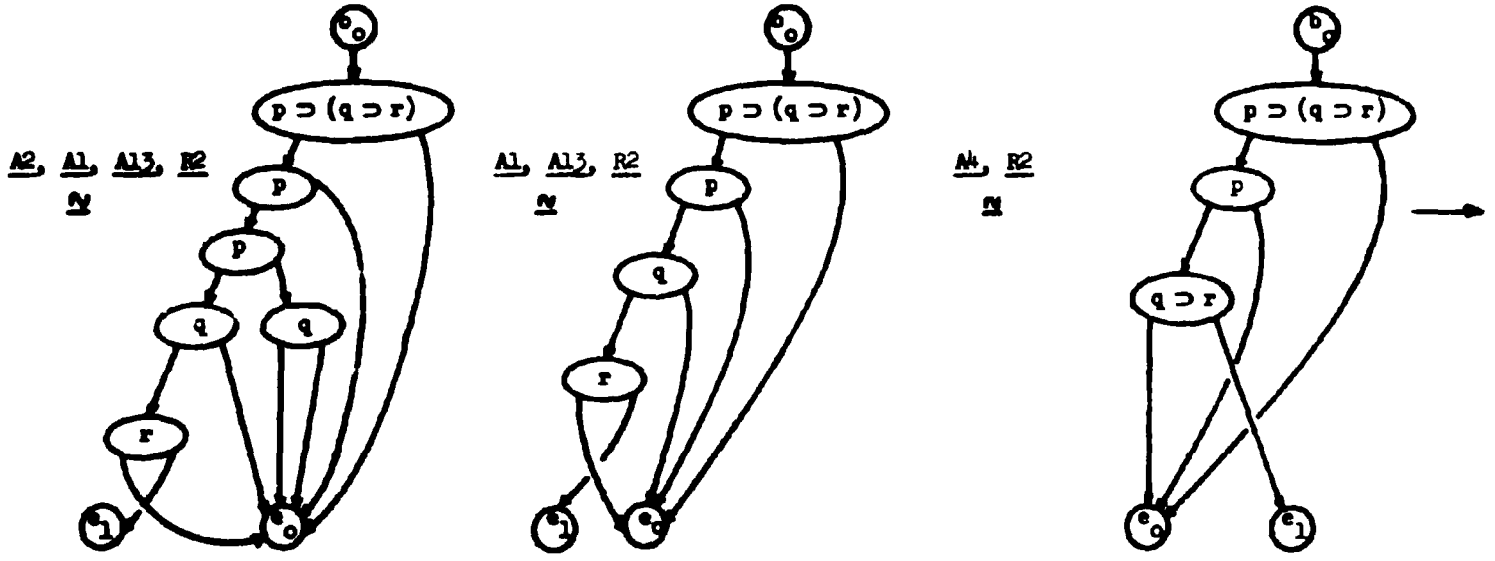


Figure 29(b) contd.

Second part of the derivation of the second wff of Figure 27. Continued next page.

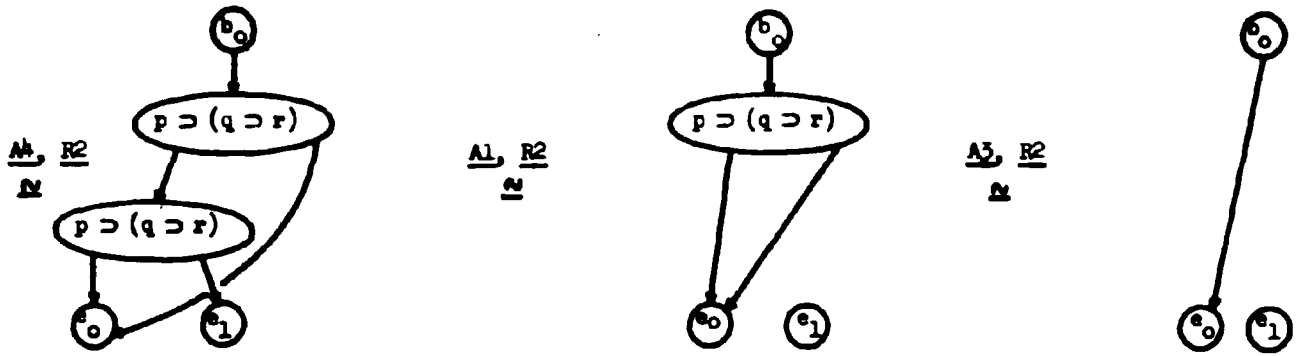


Figure 29(b) contd.

Completion of the derivation of the second wff of Figure 27.

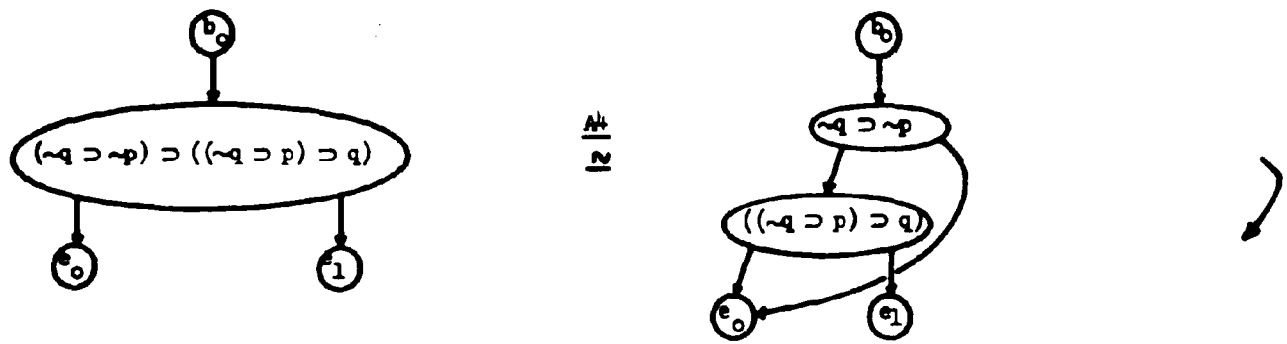


Figure 29(c)

First part of the derivation of the third wff of Figure 27. Continued next page.

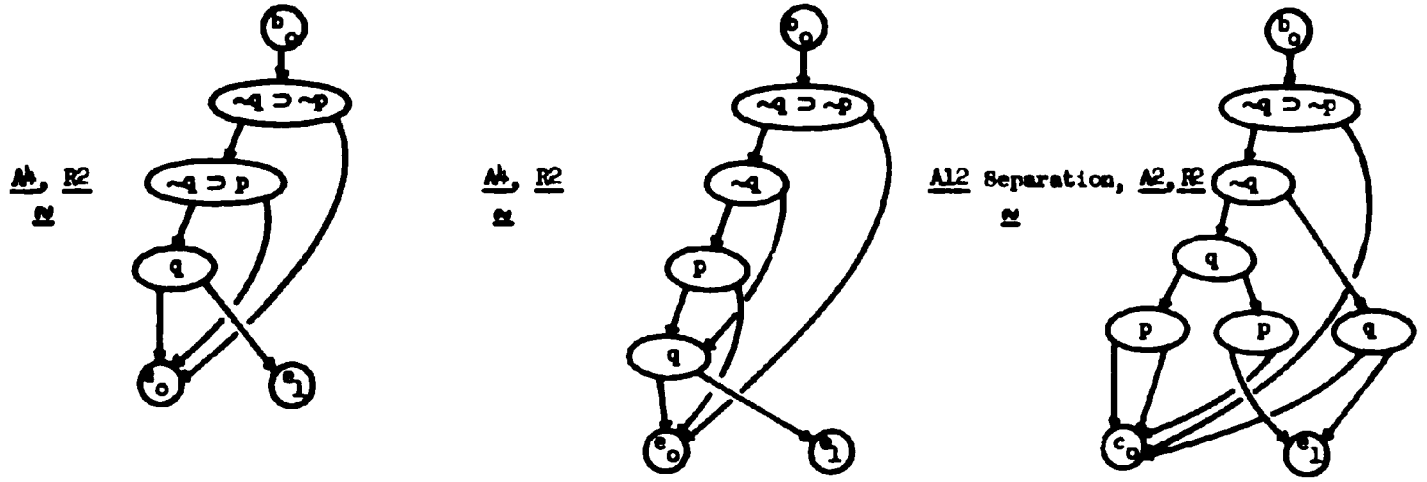


Figure 29(c) contd.

Second part of the derivation of the third wff of Figure 27. Continued next page.

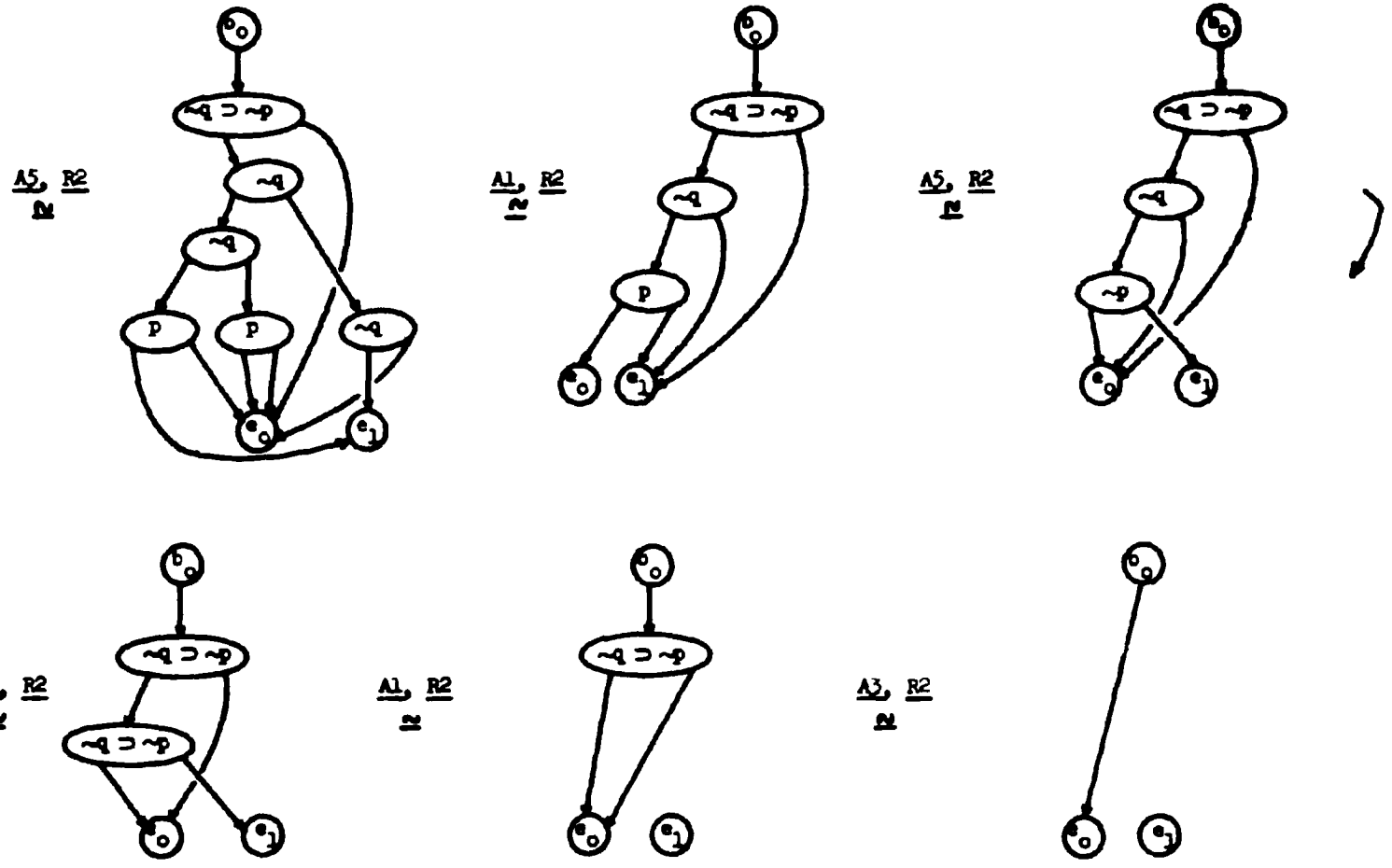


Figure 29(c) contd.

Completion of the derivation of the third wff of Figure 27.

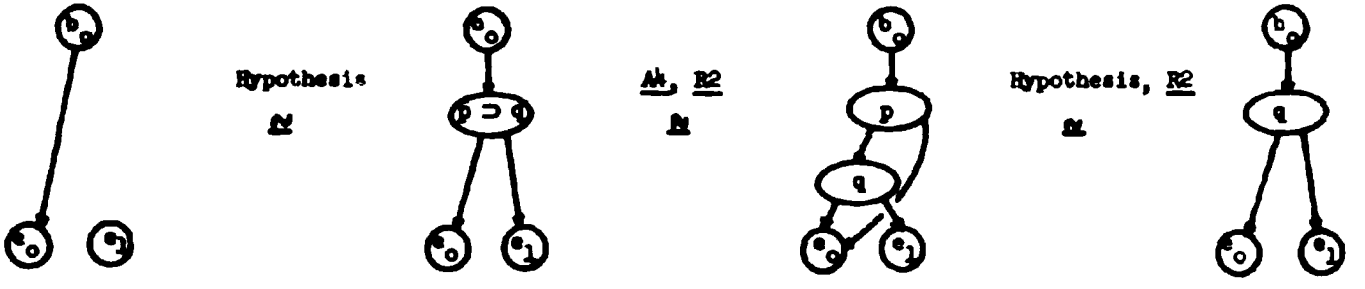


Figure 30

Derivation of the rule shown in Figure 28.

Remarks:

(i) Notice that $\mathcal{I}_s$ is still complete for $\mathcal{F}m_s'(Q)$ (but not extended complete) even if rule R3 is omitted.

(ii) Also, with reference to the argument above, if the signature s is such that $?\Delta' \vdash^* p?$ for an arbitrary Δ' and p is decidable, then so is $?\Delta \vdash \mathfrak{M} = \mathfrak{k}?$ for an arbitrary $\Delta \subseteq \mathcal{F}m_s'(Q)$ and $\mathfrak{M} = \mathfrak{k} \in \mathcal{F}m_s'(Q)$.

(iii) It is interesting that the usual axioms for propositional calculus need not be given here, i.e., we can depend solely on the more basic characterization given by axiom schemata A1 through A5 to provide an adequate axiomatization.

Extended Completeness for Sequences of Assignment Schemata

Let $\mathcal{F}m_s'(\mathcal{A}) \subseteq \mathcal{F}m_s$ be the set of all wffs of the sort depicted in Figure 31, where f_i, $i < \kappa < \omega^-$, and g_j, $j < \lambda < \omega^-$, are any assignment schemata.

Theorem 13: For any signature s, the inferential system $\mathcal{I}_s = \langle \mathcal{F}r_s, \mathcal{R} \rangle$ is extended complete for $\mathcal{F}m_s'(\mathcal{A})$.

Proof: We have $\Delta \vdash \mathfrak{M} = \mathfrak{B} \Rightarrow \Delta \vdash \mathfrak{M} = \mathfrak{B}$, for any $\Delta \subseteq \mathcal{F}m_s'(\mathcal{A})$ and $\mathfrak{M} = \mathfrak{B} \in \mathcal{F}m_s'(\mathcal{A})$, from Theorem 11.

Therefore, we have only to show that $\Delta \vdash \mathfrak{M} = \mathfrak{B} \Rightarrow \Delta \vdash \mathfrak{M} = \mathfrak{B}$, and we do this by developing the notion of normal form wffs in $\mathcal{F}m_s'(\mathcal{A})$. If a wff $\mathfrak{C}' = \mathfrak{D}' \in \mathcal{F}m_s'(\mathcal{A})$ has the same form as the left-hand hypothesis for R4 (cf. Figure 22), and if $\vdash \mathfrak{C} = \mathfrak{C}'$ and $\vdash \mathfrak{D} = \mathfrak{D}'$, then $\mathfrak{C}' = \mathfrak{D}'$ is said to be a normal form of the wff $\mathfrak{C} = \mathfrak{D}$. Evidently, for any computing structure $\underset{\sim}{D}$, we have $\vdash_{\underset{\sim}{D}} \mathfrak{C} = \mathfrak{D} \Rightarrow \vdash_{\underset{\sim}{D}} \mathfrak{C}' = \mathfrak{D}'$. By applying this to our problem we get that $\Delta \vdash \mathfrak{M} = \mathfrak{B} \Rightarrow \Delta' \vdash \mathfrak{M}' = \mathfrak{B}'$ where Δ' is a set of normal forms for the wffs in Δ and $\mathfrak{M}' = \mathfrak{B}'$ is a normal form of $\mathfrak{M} = \mathfrak{B}$.

Since R4 preserves validity, we have that $\Delta' \vdash \mathfrak{A} = \mathfrak{B}' \Rightarrow \Delta'' \vdash \mathfrak{D} = \mathfrak{t}$ where $\mathfrak{D} = \mathfrak{t} \in \mathcal{F}m_s'(\mathcal{V})$ and is obtained from $\mathfrak{A}' = \mathfrak{B}'$ as indicated by R4, just as the wffs in $\Delta'' \subseteq \mathcal{F}m_s'(\mathcal{V})$ are obtained from the wffs in Δ'. Since Theorem 12 gives that $\mathcal{J}_s$ is extended complete for $\mathcal{F}m_s'(\mathcal{V})$, we have that $\Delta'' \vdash \mathfrak{D} = \mathfrak{t} \Delta \Rightarrow \Delta'' \vdash \mathfrak{D} = \mathfrak{t}$, and by applying R4 "in reverse", we obtain $\Delta'' \vdash \mathfrak{D} = \mathfrak{t} \Rightarrow \Delta' \vdash \mathfrak{A}' = \mathfrak{B}'$. This last step is accomplished as follows: starting from Δ', use R4 repeatedly until Δ'' is obtained; then, since $\Delta'' \vdash \mathfrak{D} = \mathfrak{t}$, we can derive $\mathfrak{D} = \mathfrak{t}$; next, we apply R4 and so obtain $\mathfrak{A}' = \mathfrak{B}'$, as required. We so far have reasoned that $\Delta \vdash \mathfrak{A} = \mathfrak{B} \Rightarrow \Delta \vdash \mathfrak{A} = \mathfrak{B} \Rightarrow \Delta' \vdash \mathfrak{A}' = \mathfrak{B}' \Rightarrow \Delta'' \vdash \mathfrak{D} = \mathfrak{t} \Rightarrow \Delta'' \vdash \mathfrak{D} = \mathfrak{t} \Rightarrow \Delta' \vdash \mathfrak{A}' = \mathfrak{B}'$; to furnish the last step, i.e., $\Delta' \vdash \mathfrak{A}' = \mathfrak{B}' \Rightarrow \Delta \vdash \mathfrak{A} = \mathfrak{B}$, we have only to demonstrate how to use $\mathcal{J}_s$ to derive normal forms for wffs in $\mathcal{F}m_s'(\mathcal{A})$. For, then, we start from Δ and derive Δ' and so therefore $\mathfrak{A}' = \mathfrak{B}'$; if a derivation of $\mathfrak{A}' = \mathfrak{B}'$ from $\mathfrak{A} = \mathfrak{B}$ can be given, then it will, in reverse order, serve as a derivation of $\mathfrak{A} = \mathfrak{B}$ from $\mathfrak{A}' = \mathfrak{B}'$.

Thus, it remains for us to show how any desired normal form of a wff $\mathfrak{A} = \mathfrak{B} \in \mathcal{F}m_s'(\mathcal{A})$ can be derived using $\mathcal{J}_s$. We use R1 and R2 as required and proceed as follows.

(i) Apply A8 repeatedly to each of $\mathfrak{A}$ and $\mathfrak{B}$ until only a single assignment schema remains in each.

(ii) Apply A9 followed by A8 or vice versa as needed, in order to make the sets of assigned variables in the two assignment schemata identical to the desired set.

(iii) Apply A10 to arrange the assignments in both assignment schemata in the desired order. ∎

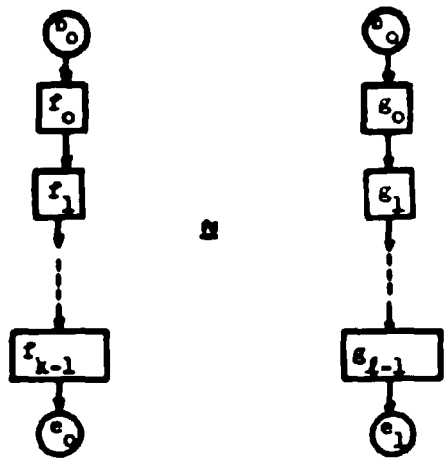


Figure 31

A member of $\mathfrak{F}_{m'g}(\mathfrak{A})$. Here f_i, $i < k < \omega^-$, and g_j, $j < \ell < \omega^-$, are assignment schemata.

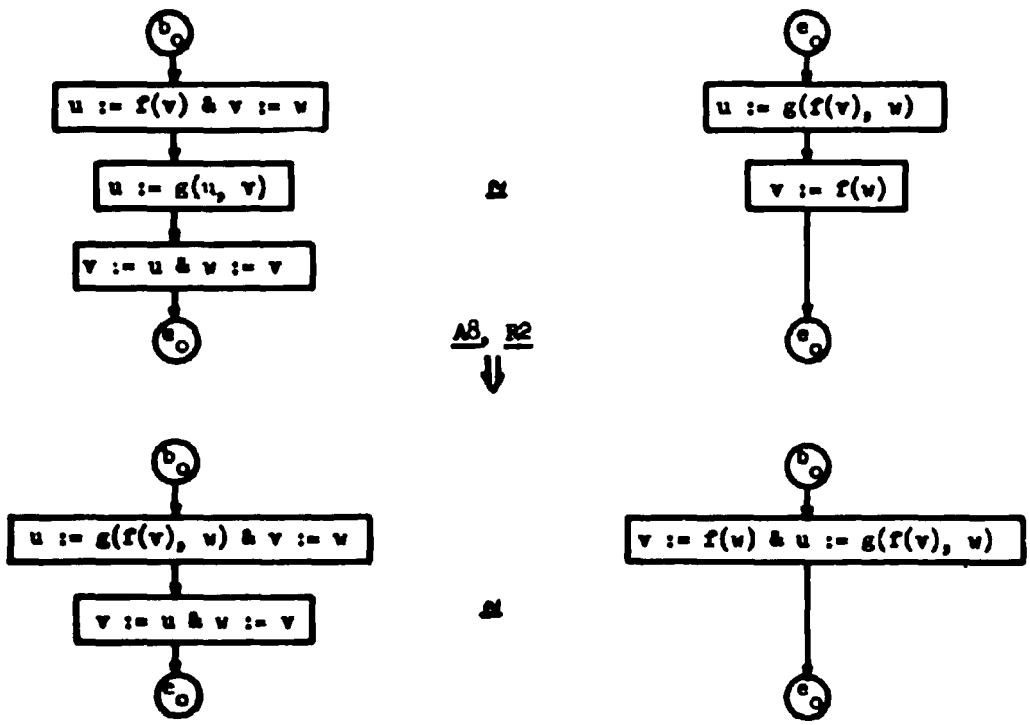


Figure 32(a)

First step in the derivation of a normal form for the upper wff. Here u, v, w are distinct variables and f, g are function letters.
Continued next page.

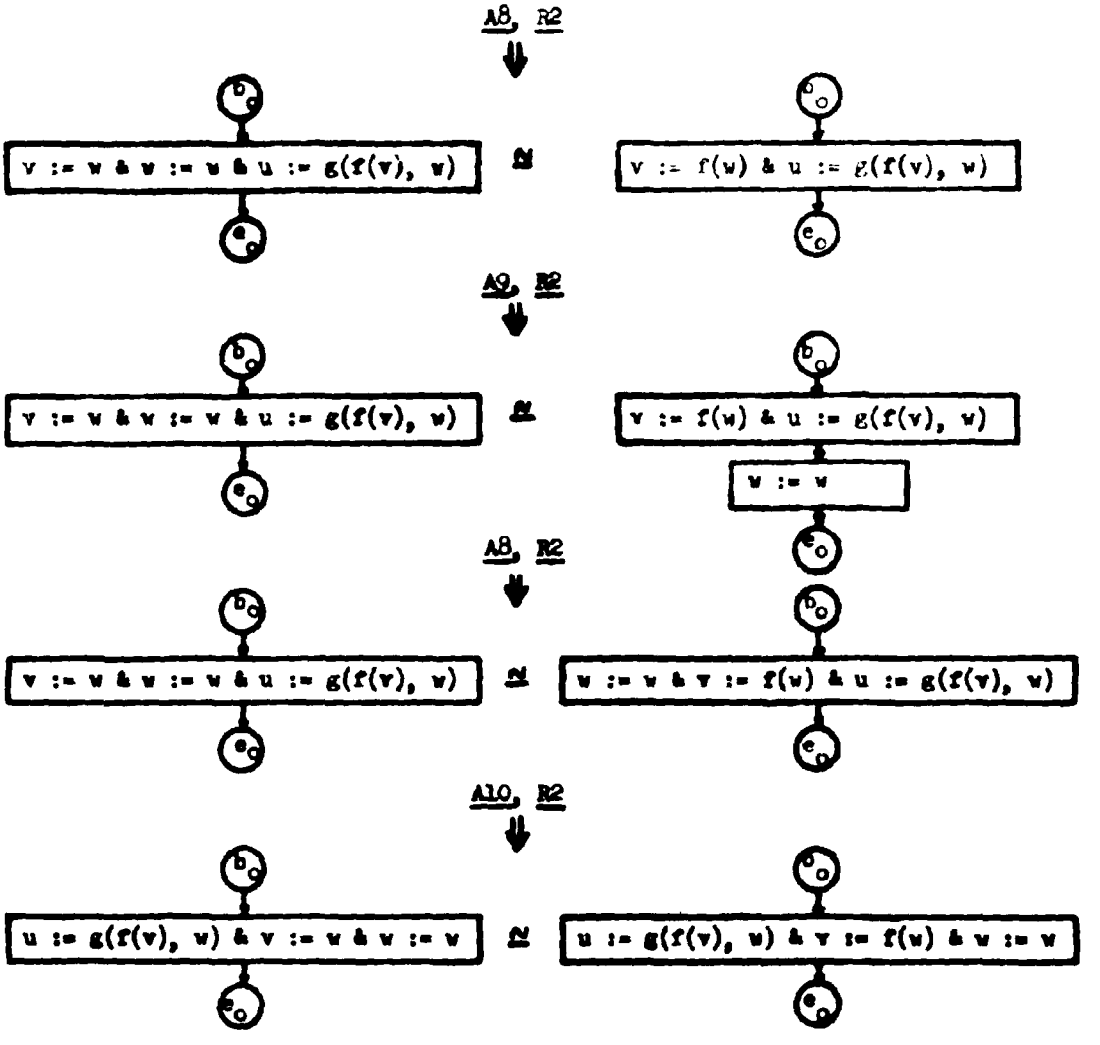


Figure 32(a) contd.

Completion of the derivation of a normal form of the first wff of this figure.

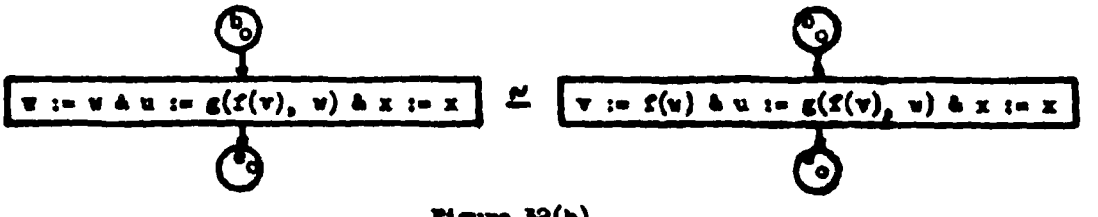


Figure 32(b)

A second normal form for the first wff of Figure 32(a). Here x is a variable distinct from u and v.

Notice that two normal forms for a wff $\mathfrak{A} \simeq \mathfrak{B} \in \mathfrak{F}_{\mathfrak{m}_s}'(\mathfrak{f})$ may differ only in the order of the assignments in $\mathfrak{A}$ and $\mathfrak{B}$ and in the number and sort of null assignments of the form $u := u$ that occur in both $\mathfrak{A}$ and $\mathfrak{B}$. Figure 32(a) illustrates the derivation of a normal form for a wff in $\mathfrak{F}_{\mathfrak{m}_s}'(\mathfrak{f})$, and Figure 32(b) illustrates how two normal forms may differ by giving a second normal form for the same wff.

Let $L_s'(\mathfrak{f}) \subseteq L_s$ be the set of all E-programs that occur in the wffs of $\mathfrak{F}_{\mathfrak{m}_s}'(\mathfrak{f}) \subseteq \mathfrak{F}_{\mathfrak{m}_s}$, i.e., the set of all E-programs consisting of a single sequence of assignment schemata. The idea of normal forms for wffs in $\mathfrak{F}_{\mathfrak{m}_s}'(\mathfrak{f})$ can be extended to yield a canonical form for algorithms in $L_s'(\mathfrak{f})$. We use R1 and R2 and proceed as follows.

(i) Apply A8 repeatedly until a single assignment schema remains.

(ii) Apply A8 followed by A9 to delete all vacuous assignments of the form $u := u$.

(iii) Apply A10 to arrange the assigned variables in order by their subscripts.

If this process applied to $\mathfrak{A} \in L_s'(\mathfrak{f})$ yields $\mathfrak{A}' \in L_s'(\mathfrak{f})$, then we say that $\mathfrak{A}'$ is the canonical form for $\mathfrak{A}$. Of course, we should say with respect to what property this form is canonical, and that is the content of the following

Theorem 14: For any signature s and $\mathfrak{A}, \mathfrak{B} \in L_s'(\mathfrak{f})$ with canonical forms $\mathfrak{A}'$ and $\mathfrak{B}'$ respectively, $\vdash \mathfrak{A} \simeq \mathfrak{B} \circ \mathfrak{A}'$ is identical to $\mathfrak{B}'$.

Proof: First, we prove that $\mathfrak{A}'$ identical to $\mathfrak{B}' \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}$. Now, $\vdash \mathfrak{A} \simeq \mathfrak{A}' \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{A}'$ and $\vdash \mathfrak{B} \simeq \mathfrak{B}' \Rightarrow \vdash \mathfrak{B} \simeq \mathfrak{B}'$ by Theorem 11. Furthermore, $\mathfrak{A}'$ identical to $\mathfrak{B}' \Rightarrow \vdash \mathfrak{A}' \simeq \mathfrak{B}'$, and since strong equivalence is transitive and symmetric, $\vdash \mathfrak{A}' \simeq \mathfrak{B}' \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}$.

To prove that $\vdash \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \mathfrak{A}'$ and $\mathfrak{B}'$ are identical, assume the contrary, i.e., that $\vdash \mathfrak{A} \simeq \mathfrak{B}$, but $\mathfrak{A}'$ and $\mathfrak{B}'$ differ. Since $\mathfrak{A}'$ and $\mathfrak{B}'$ each consist of a single assignment schema, they can differ in only two ways.

(i) A variable u occurs an assigned variable in $\mathfrak{A}'$ but not in $\mathfrak{B}'$, say, or

(ii) Assignments $u := \tau$ and $u := \sigma$, where τ and σ are distinct terms, occur in $\mathfrak{A}'$ and $\mathfrak{B}'$ respectively.

Consider case (i). An assignment $u := \tau$ occurs in $\mathfrak{A}'$ and τ is not u since $\mathfrak{A}'$ is canonical. Thus, a computing structure $\underset{\sim}{D}$ and state $\xi : \omega \to \underset{\sim}{D}_{\circ}$ can be found such that $\tau[\underset{\sim}{D}, \xi] \neq u[\underset{\sim}{D}, \xi]$, and so therefore $\mathfrak{A}' \simeq \mathfrak{B}'$ is not valid in $\underset{\sim}{D}$, i.e., not $\vdash \mathfrak{A}' \simeq \mathfrak{B}'$. But, $\vdash \mathfrak{A} \simeq \mathfrak{A}'$, $\vdash \mathfrak{B} \simeq \mathfrak{B}'$ and $\vdash \mathfrak{A} \simeq \mathfrak{B}$, so that $\vdash \mathfrak{A}' \simeq \mathfrak{B}'$, a contradiction.

Consider case (ii). Here a computing structure $\underset{\sim}{D}$ and state $\xi : \omega \to \underset{\sim}{D}_{\circ}$ can be found so that $\tau[\underset{\sim}{D}, \xi] \neq \sigma[\underset{\sim}{D}, \xi]$, thus giving the same result as (i) above. Hence, $\vdash \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \mathfrak{A}'$ and $\mathfrak{B}'$ are identical. ∎

We have, therefore, an effective test for the strong equivalence of E-programs in $L_s'(\mathcal{A})$, for any signature s. Of course, semantic consequence remains an unvolvable problem here because of its direct connection with the logical validity of qffs in PC_s.

Using Theorem 14, we can easily obtain a further result. In Chapter 5, we considered the set $\mathcal{F}_{\mathcal{M}_s}(\mathcal{A}) \subseteq \mathcal{F}_{\mathcal{M}_s}$ for arbitrary signatures s, and in Theorem 2, we put off proving until now that $?\vdash \mathfrak{A} \simeq \mathfrak{B}?$, for $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}_{\mathcal{M}_s}(\mathcal{A})$, is decidable. Recall that the wffs of $\mathcal{F}_{\mathcal{M}_s}(\mathcal{A})$ contain only E-programs from

$L_s(\mathcal{A})$, and that no qffs occur in these E-programs (cf. Figure 4(a) for an example). We prove the decidability result of Theorem 2 during the proof of the following

Theorem 15: For any signature s , the inferential system $\mathcal{I}_s = \langle \mathcal{A}x_s, \mathcal{R} \rangle$ is complete for $\mathcal{F}m_s(\mathcal{A})$.

Proof: We have $\vdash \mathfrak{A} = \mathfrak{B} \Rightarrow \models \mathfrak{A} = \mathfrak{B}$, for any $\mathfrak{A} = \mathfrak{B} \in \mathcal{F}m_s(\mathcal{A})$ from Theorem 11.

Therefore, we have only to show that $\models \mathfrak{A} = \mathfrak{B} \Rightarrow \vdash \mathfrak{A} = \mathfrak{B}$, and we do this using methods similar to those used in the proof of Theorem 14. First, we develop the notion of a canonical form for E-programs in $L_s(\mathcal{A})$. We use R1 and R2 and proceed as follows.

(i) Apply A12 at each initiator to separate out the various sequences of assignment schemata. (Figure 33(a) illustrates this step for the E-program of Figure 4(a).)

(ii) Apply A14 at each initiator to detect which sequences never terminate, and use A13 to clear away all unreachable nodes. (Figure 33(b) illustrates this step.)

(iii) Apply A8, A9 and A10 to the remaining sub-programs, each consisting of a sequence of assignment schemata, to put them into canonical form, as we have already described. (Figure 33(c) illustrates this step.)

If this process applied to $\mathfrak{A} \in L_s(\mathcal{A})$ yields $\mathfrak{A}' \in L_s(\mathcal{A})$, then we say $\mathfrak{A}'$ is the canonical form for $\mathfrak{A}$. The decision procedure required for Theorem 2 is then given by

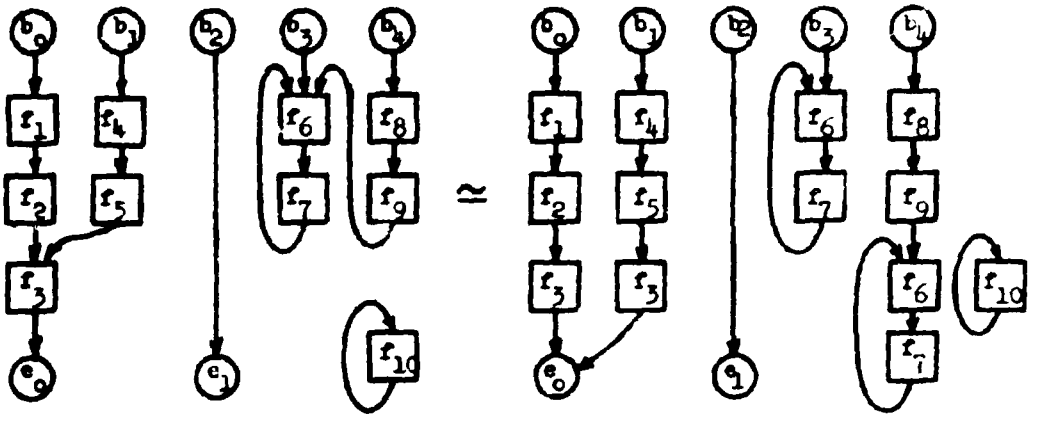


Figure 33(a)

E-program $\mathfrak{E}$ of Figure 4(a) is processed by A12, the separation axiom schema.

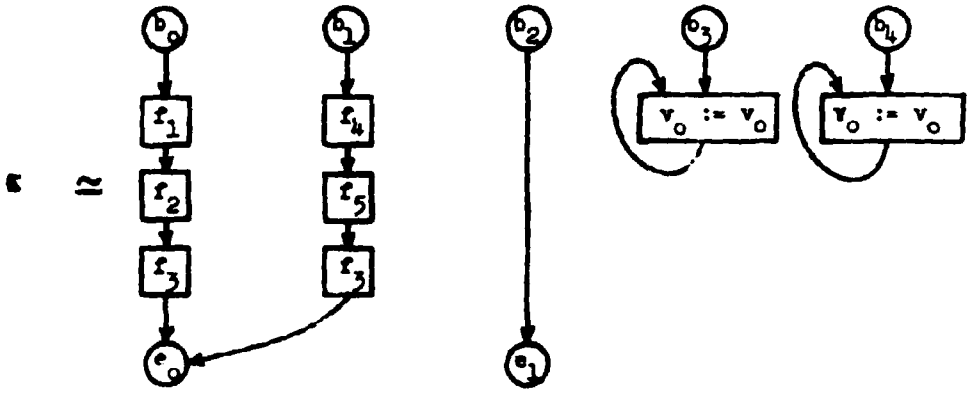


Figure 33(b)

Further simplification of $\mathfrak{E}$ arises from the application of A13 and A14.

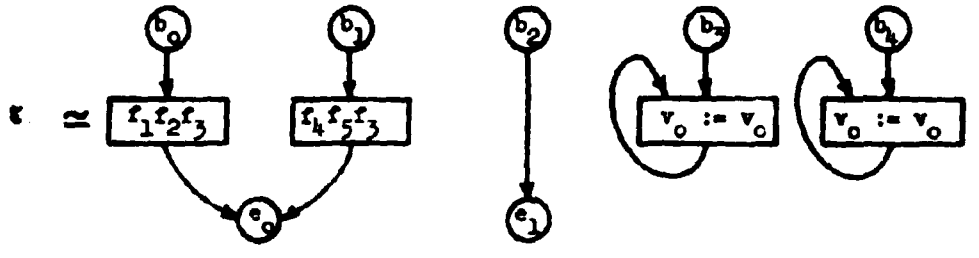


Figure 33(c)

The canonical form of E-program $\mathfrak{E}$, where $f_1f_2f_3$ and $f_4f_5f_3$ are in their respective canonical forms.

Theorem 16: For any signature s and any $\mathfrak{A}$, $\mathfrak{B} \in L_s(\mathcal{A})$ with canonical forms $\mathfrak{A}'$ and $\mathfrak{B}'$ respectively, $\vdash \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \mathfrak{A}'$ and $\mathfrak{B}'$ are identical.

Proof: If e_j is reachable from b_i in $\mathfrak{A}'$ but not in $\mathfrak{B}'$, then clearly $\mathfrak{A}$ and $\mathfrak{B}$ cannot be strongly equivalent, i.e., not $\vdash \mathfrak{A} \simeq \mathfrak{B}$, which contradicts the hypothesis for Theorem 16. Therefore, $\mathfrak{A}'$ and $\mathfrak{B}'$ can be analyzed by comparing the sub-program of $\mathfrak{A}'$ between b_i and e_j and that of $\mathfrak{B}'$ between b_i and e_j, using Theorem 14. Evidently, then $\vdash \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \mathfrak{A}'$ and $\mathfrak{B}'$ are identical; the converse is, as in Theorem 14, trivially true. ∎

To return to the proof of Theorem 15, we now consider the problem of showing that $\vdash \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}$. From Theorem 16, $\vdash \mathfrak{A} \simeq \mathfrak{B} \Rightarrow \mathfrak{A}'$ and $\mathfrak{B}'$ are identical, so that $\vdash \mathfrak{A}' \simeq \mathfrak{B}'$ by R1 and R2. Furthermore $\vdash \mathfrak{A} \simeq \mathfrak{A}'$ and $\vdash \mathfrak{B} \simeq \mathfrak{B}'$ so that, once again by R1 and R2, $\vdash \mathfrak{A} \simeq \mathfrak{B}$. ∎

Results for Other Possible Assignment Schemata

In Chapter 3, we introduced simple assignment schemata, i.e., those each consisting of a single assignment. Let $\mathcal{A}^\circ$ be the set of all simple assignment schemata (relative to a given signature s, of course), and let L_s° be the formal language obtained when $\mathcal{A}^\circ$ is the class of operators. If $\mathcal{F}m_s^\circ$ is the set of wffs built up from L_s°, then we want to consider the new theory $\mathcal{T}_s^\circ = \langle \mathcal{F}m_s^\circ, \mathcal{A}_s^\circ \rangle$, where $\mathcal{A}_s^\circ$ is like $\mathcal{A}_s$ except that axiom schemata B1, B2, B3, B4 and rule S1, illustrated in Figure 34, replace A8, A9, A10 and R4.

Certainly $?\vdash \mathfrak{A} \simeq \mathfrak{B}?$ for $\mathfrak{A}$, $\mathfrak{B} \in L_s^\circ(\mathcal{P})$, i.e., $\mathfrak{A} \simeq \mathfrak{B} \in \mathcal{F}m_s^{\circ\prime}(\mathcal{P})$, is decidable by applying the methods of the foregoing section. (Here the

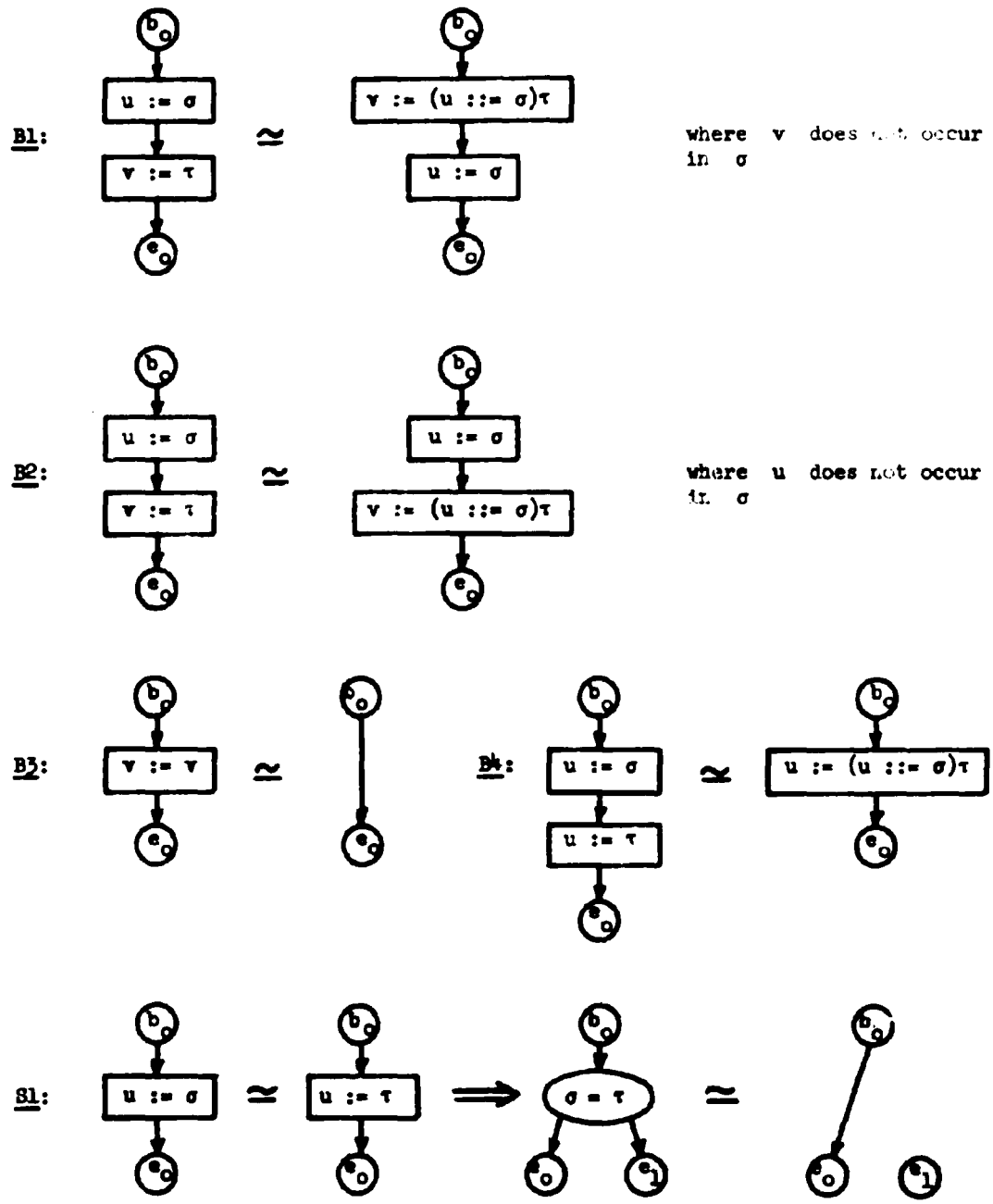


Figure 34

An inferential sub-system for simple assignment schemata. Here, u, v are distinct variables, and σ, τ are terms.

E-programs of $L_s^{\circ\prime}(\mathcal{A}^\circ)$ are all type $\langle 1, 1\rangle$, and consist of a single sequence of simple assignment schemata.) Moreover, we conjecture the following

Theorem 17: For any signature s, the inferential system $\mathcal{I}_s^\circ$ is complete for $\mathcal{F}m_s^{\circ\prime}(\mathcal{A}^\circ)$.

Proof: In support of this conjecture, notice that essentially all we have to show is that B1, B2, B3 and B4 do not miss any of the derivational power afforded by A8, A9 and A10 for theorems $\mathfrak{M} \simeq \mathfrak{N} \in \mathcal{F}m_s^{\circ\prime}(\mathcal{A}^\circ)$.

Consider $\mathfrak{M} \in L_s^{\circ\prime}(\mathcal{A}^\circ)$, illustrated in Figure 35(a). We want to find all E-programs $\mathfrak{N} \in L_s^{\circ\prime}(\mathcal{A}^\circ)$ such that $\vdash \mathfrak{M} \simeq \mathfrak{N}$, i.e., $\mathfrak{M} \simeq \mathfrak{N}$ is derivable with A8, A9 and A10.

(i) Under the hypotheses that u and v are distinct, σ is not u, τ is not v, and u does not occur in σ, the derivation in Figure 35(b) can be carried out.

(ii) Under the hypotheses that u and v are distinct, σ is not u, τ is not v, and v does not occur in σ, the derivation in Figure 35(c) can be carried out.

(iii) Under the hypothesis that neither σ nor τ is u, the derivation in Figure 35(d) can be carried out.

This clearly exhausts all the possibilities for E-programs $\mathfrak{N} \in L_s^{\circ\prime}(\mathcal{A}^\circ)$, such that $\mathfrak{M} \simeq \mathfrak{N}$ is derivable in $\mathcal{I}_s$. But we have simply derived B1, B2 and B3 here; as well, B4 is the equivalent in $\mathcal{I}_s^\circ$ of A9. Thus, $\mathcal{I}_s^\circ$ reflects all of the derivational power of $\mathcal{I}_s$ with respect to theorems in $\mathcal{F}m_s^{\circ\prime}(\mathcal{A}^\circ)$. ∎

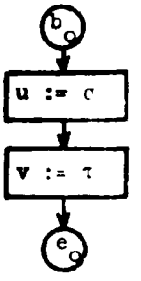


Figure 35(a)

Here, u, v are variables, and σ, τ are terms

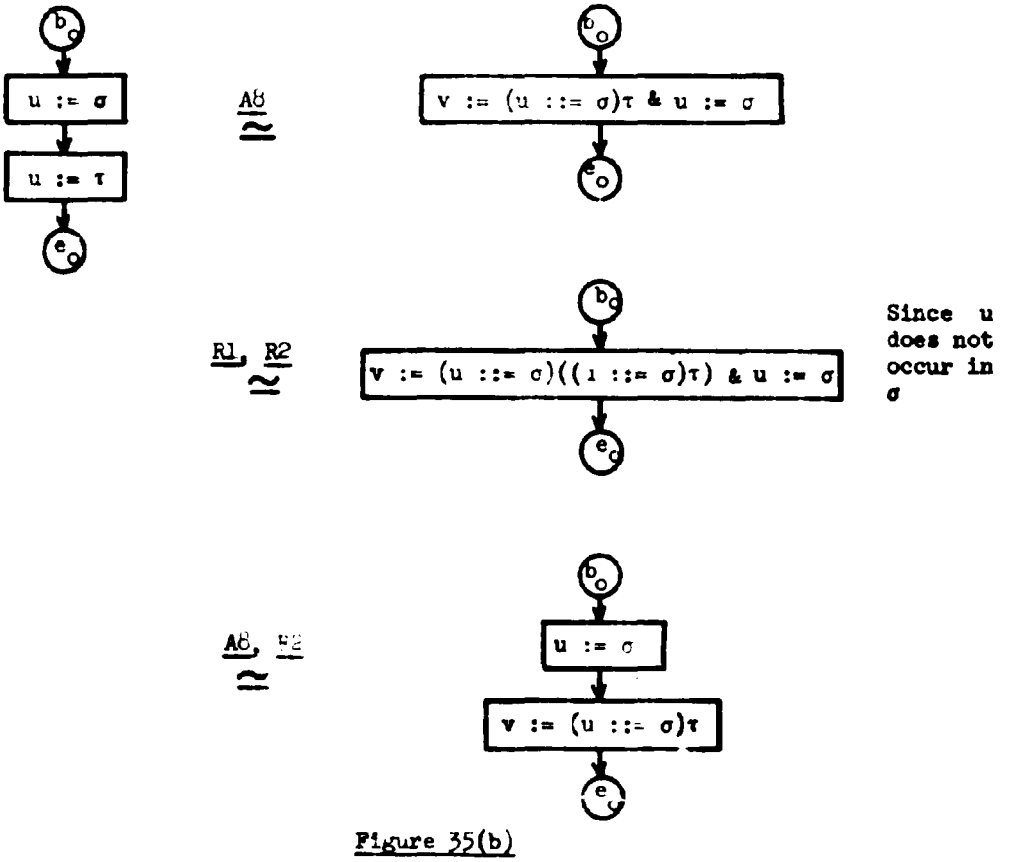
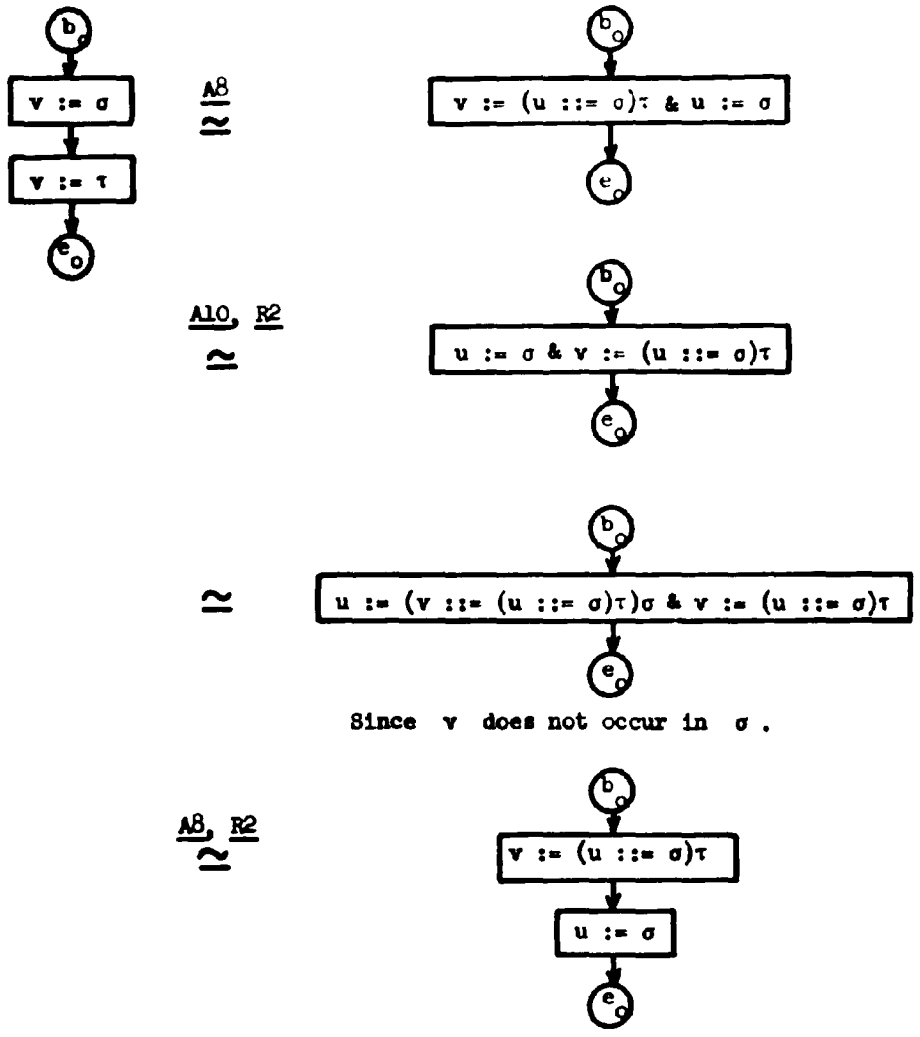


Figure 35(b)

Here, u, v are distinct variables, σ is a term not u, τ is a term not v, and u does not occur in σ.



Since v does not occur in σ .

Figure 35(c)

Here, u, v are distinct variables; σ is a term not u ; τ is a term not v ; and v does not occur in σ .

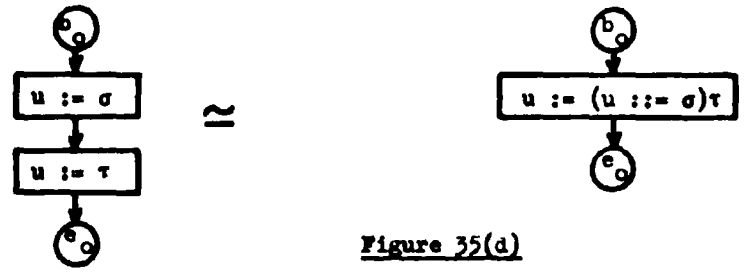


Figure 35(d)

Here, u is a variable and σ τ are terms not u .

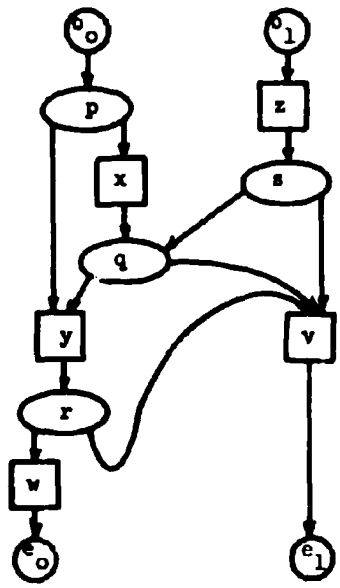
In Chapter 3, we introduced subscripted variables as a possible extension of our formal language L_s. Let $\mathcal{A}^+$ be the set of all assignment schemata with subscripted variables permitted (relative to a given signature s, of course), and let $L_s^{+\prime}(\mathcal{A}^+)$ be the set of all type $\langle 1, 1\rangle$ E-programs each consisting of a single sequence of assignment schemata from $\mathcal{A}^+$.

We conjecture that for any signature s, $?\vdash_{\mathfrak{M}} \Gamma \equiv \Gamma'$ for $\mathfrak{M}, \mathfrak{N} \in L_s^{+\prime}(\mathcal{A}^+)$ is decidable, but here do not consider the matter any further.

E-programs with no Loops

It is straightforward to prove that $\mathcal{A}_s$ is complete for the subset of $\mathfrak{Im}_s$ involving only E-programs with no loops. An E-program $\mathfrak{M} = \langle X, \Gamma, x\rangle$ is said to have "no loops" iff for all $x \in X$, x is not reachable from x via Γ, i.e., not in the transitive closure of Γ. E-program $\mathfrak{M}_0$ of Figure 36(a) is an example of an E-program with no loops.

Here, we only indicate, using an example, the derivational steps required to put an E-program with no loops into a canonical form. Figure 36(b) shows E-program $\mathfrak{M}_1$, the result after the separation axiom schema A12 has been applied throughout to $\mathfrak{M}_0$. Figure 36(c) shows $\mathfrak{M}_2$, the result after the push through axiom schema A11 and the forward substitution axiom schema A8 have been applied throughout to $\mathfrak{M}_1$. Each assignment schema is then put into canonical form, and axiom schemata A1 through A9 applied to compress the network of qffs. The result is



E-program M

Figure 36(a)

An E-program with no loops. Here p, q, r, s are qffs; v, w, x, y, z are assignment schemata.

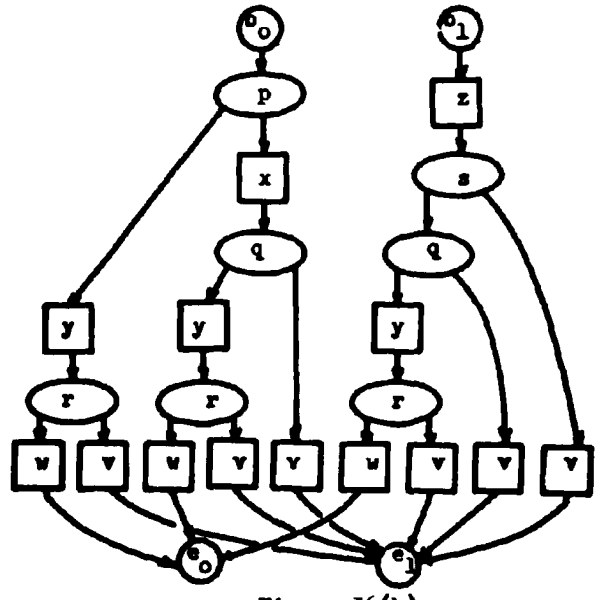


Figure 36(b)

E-program M after separation using A12.

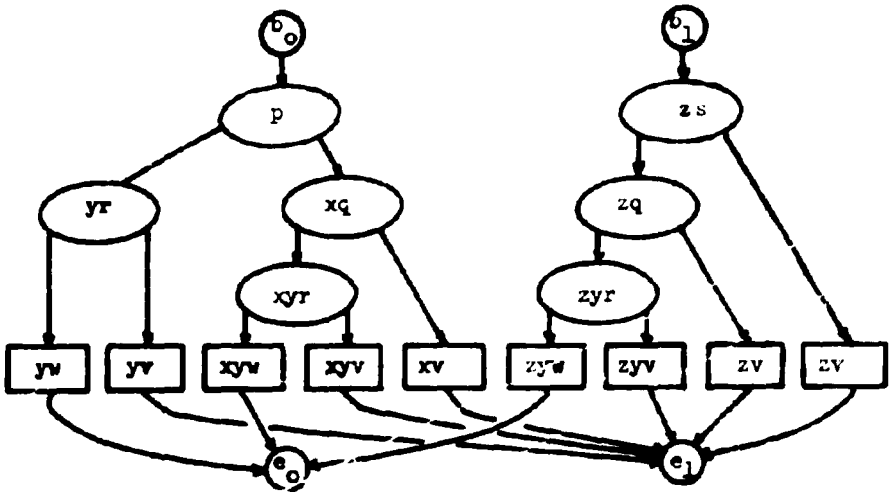


Figure 36(c)

E-program $\mathfrak{E}$ after push through using All.

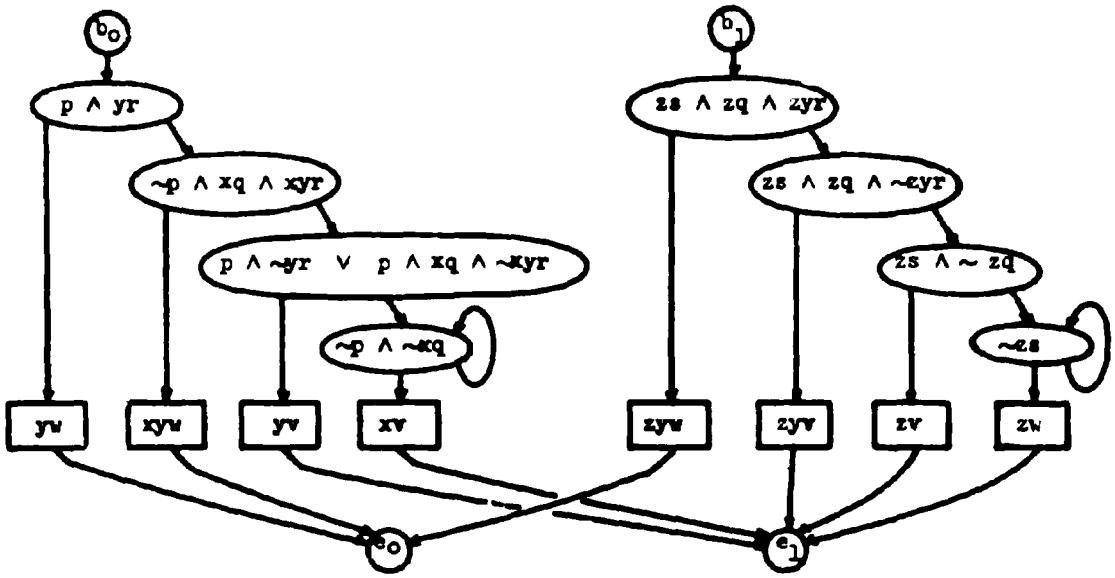


Figure 36(d)

E-program $\mathfrak{E}$ after processing by A1,..., A5. Here, we have assumed that $yv[D, \xi] = xyv[D, \xi]$ for all D, all $\xi : \omega \to D_o$, so that the condition for yv is a disjunction of two conjunctions. Here, the result just displays the initial conditions for each of the operations.

$\mathfrak{M}_3$, shown in Figure 36(d). Note that since two of the assignment schemata have the same canonical form, one occurrence is deleted and the relevant qff is expressed as a disjunction. By specifying an ordering and a canonical form for the qffs of $\mathfrak{M}_3$, a canonical form for E-programs with no loops is obtained.

We see that a canonical form for these E-programs merely displays each of the finite number of operations (as discussed in the proof of Theorem 1) and its attendant initial condition.

Remarks:

(i) The canonical form concept for E-programs without loops, can be extended to any E-program in L_s provided that infinite canonical forms are permitted. Thus, all the loops are unwound, and the E-program takes on the loop-free property in infinitary form.

(ii) The completeness result for qffs as given in Theorem 12 can be used to show $\mathcal{J}_s$ is complete for the sub-set of $\mathfrak{F}_{\mathfrak{M}_s}$ involving E-programs which correspond to the "conditional expressions" discussed by McCarthy [30]. Type $\langle 1, 2\rangle$ E-programs containing qffs only, but possibly having loops, can be used to simulate predicates which are undefined for certain arguments. It is conjectured that $\mathcal{J}_s$ is complete for this case as well.

(iii) In Theorem 1, $?\vdash \mathfrak{A} = \mathfrak{B}?$, where $\mathfrak{A} = \mathfrak{B} \in \mathfrak{F}_{\mathfrak{M}_s}$ for signatures $s = \langle \langle n_0, \ldots, n_{k-1}\rangle, 0, 0\rangle$, was shown to be decidable. We conjecture that $\mathcal{J}_s$ is complete for $\mathfrak{F}_{\mathfrak{M}_s}$.

Some Applications of the Formal Theory $\mathcal{T}_s$

To give some indication of the derivational power of the inferential system $\mathcal{J}_s$, we consider a few applications of the formal theory $\mathcal{T}_s$.

In Figure 37, we show how a simple loop, organized as in FORTRAN where the body is executed at least once, can be transformed into a loop organized as in ALGOL where the body is possibly not executed at all. The notations "gp" and "gh" indicate the forward substitution of assignment schema g into the qff p and assignment schema h, respectively.

In Figure 38(a), we use two hypotheses to deduce that the simple loop considered is always indeterminate. Of course, the hypotheses may in fact be generally valid, i.e., the syntactic properties of f, g and p, may permit proof of the wffs taken here as hypotheses. Figure 38(b) illustrates just this possibility for the always indeterminate loop problem considered here. In all of the examples considered here, where certain hypotheses are assumed, there is the parallel case where the hypotheses themselves are derivable.

In Figure 39, we show a simple loop can be reorganized to display the cases where execution is determinate and indeterminate. Simply put, we have here the case of the body of a loop having no new effect after one execution so that if no exit is made after one circuit, no exit will be made at all.

In Figure 40, we illustrate the classic removal from a FORTRAN-like loop of an operation which is loop-independent. The conditions expressed by the hypotheses are sufficient to permit this reorganization, but not necessary. For example, if the qff p were $r_o(v_o)$ and the assignment schema k were $v_1 := f_o(v_1)$, and if g and h commuted with $v_1 := f_o(v_1)$ then the same removal of k, i.e., $v_1 := f_o(v_1)$, from the loop is warranted and is derivable. Figure 41 illustrates this sort of situation in a simple loop.

The problem illustrated in Figure 42 is also classic. Here we derive from a sufficient set of hypotheses that the assignment schema f may be executed before or after the loop in question, i.e., we prove that the loop is "transparent" to f .

Figure 43 illustrates a derivable wff expressing the strong equivalence of two always determinate E-programs.

Figure 44 illustrates the sort of deduction concerning assignment schemata that can be carried out using hypotheses which express algebraic properties of the functions involved, e.g., commutativity or identity.

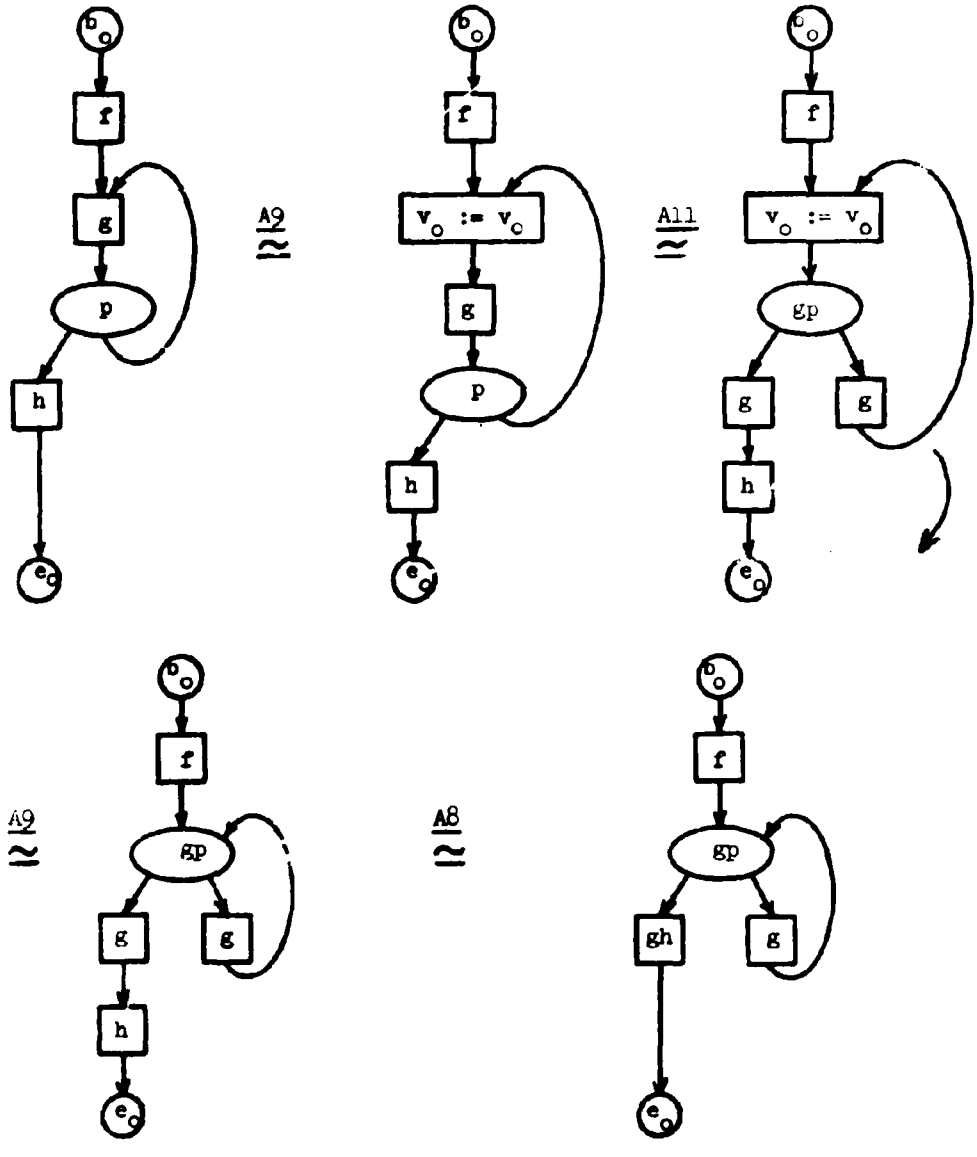


Figure 37

Conversion of a FORTRAN-like loop to ALGOL-like form. Here, f, g, h are assignment schemata and p is a qff.

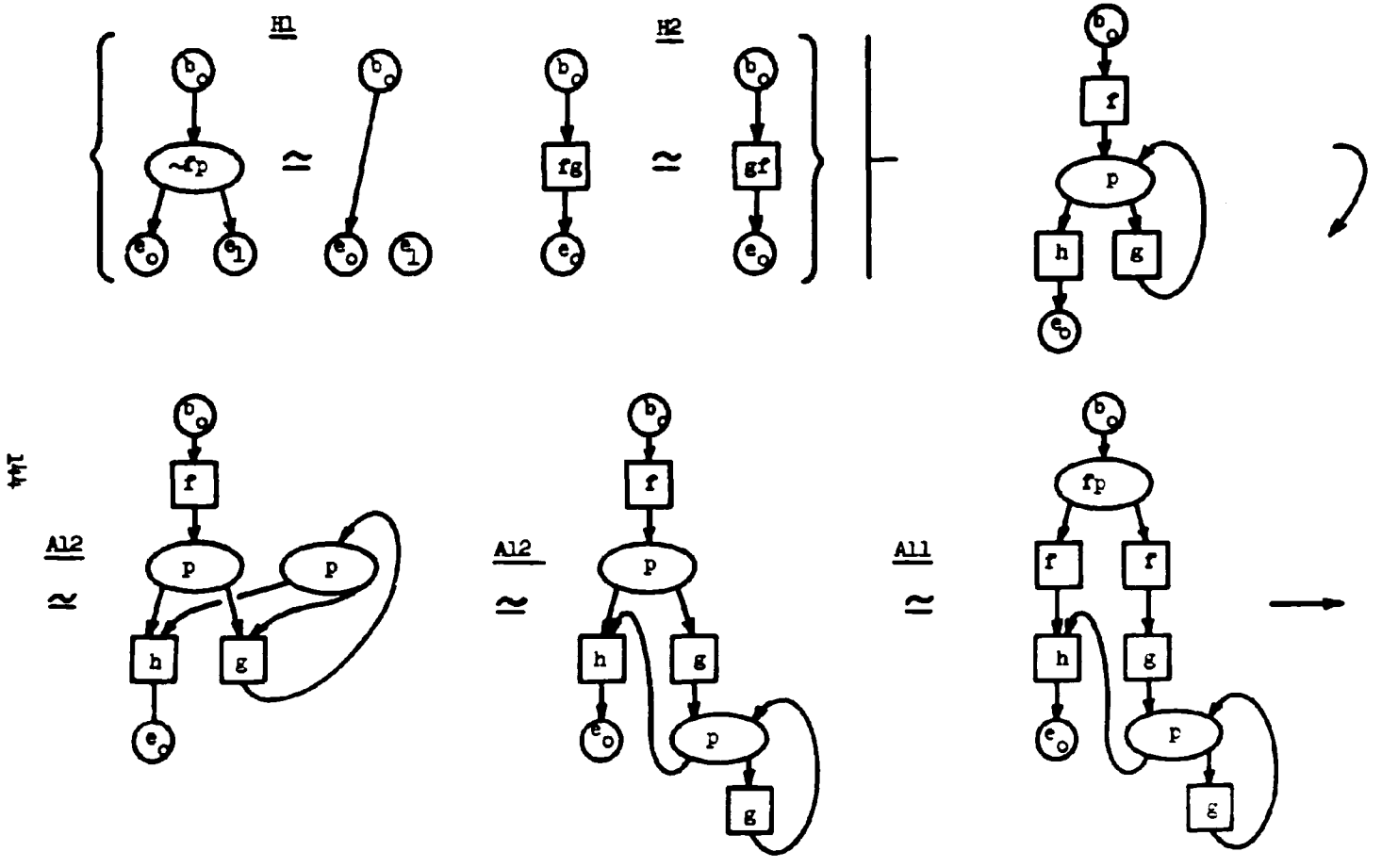


Figure 38(a)

Detection of an always indeterminate E-program under certain hypotheses. Continued next page.

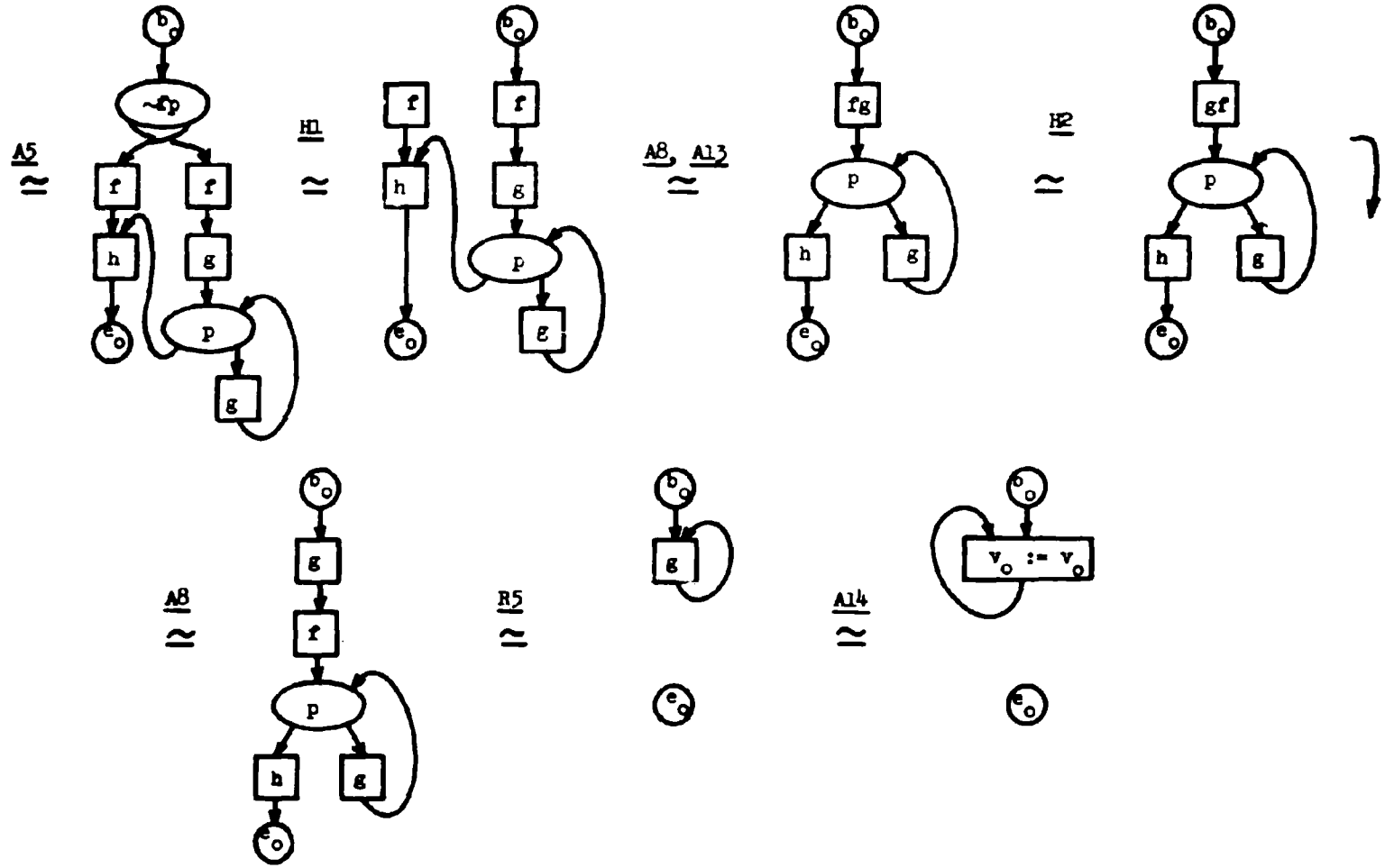


Figure 38(a) contd.

Detection of an always indeterminate E-program under certain hypotheses.

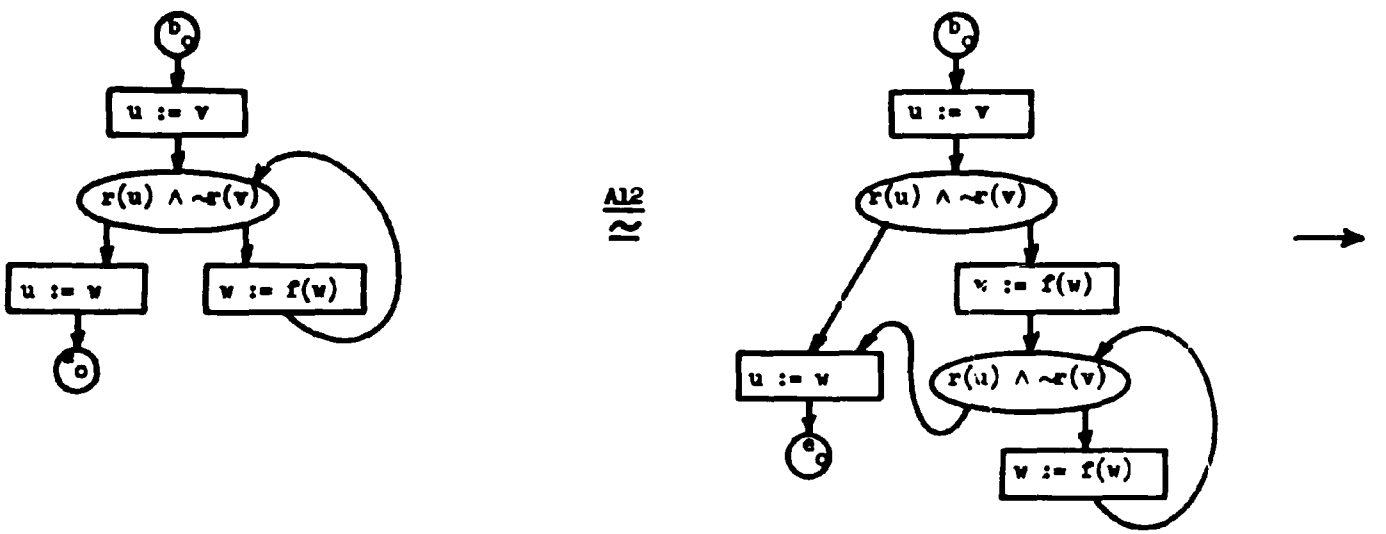


Figure 38(b)

Detection of an always indeterminate E-program. Here, u, v, w are variables; f is a function letter; and r is a relation letter. Continued next page.

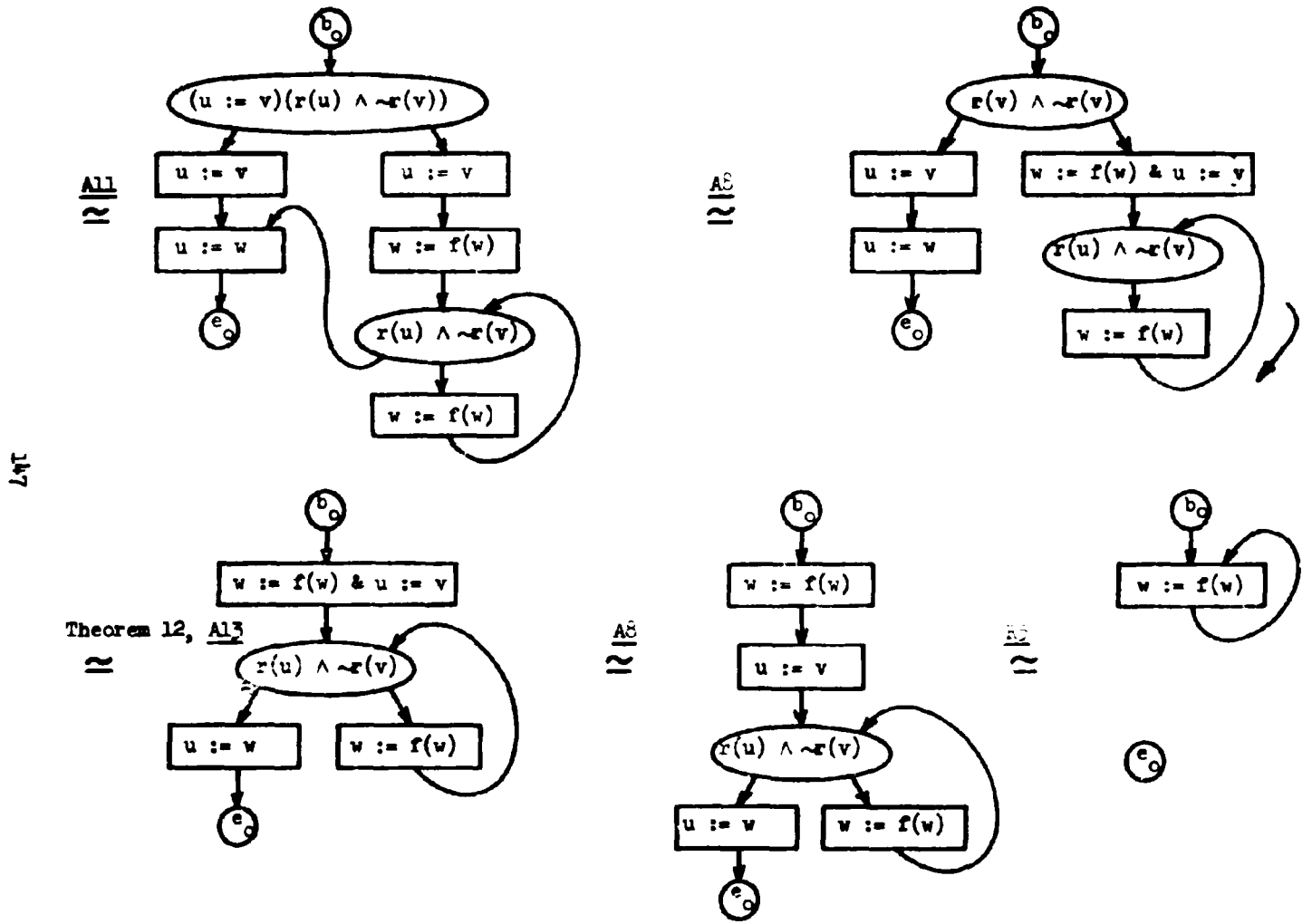
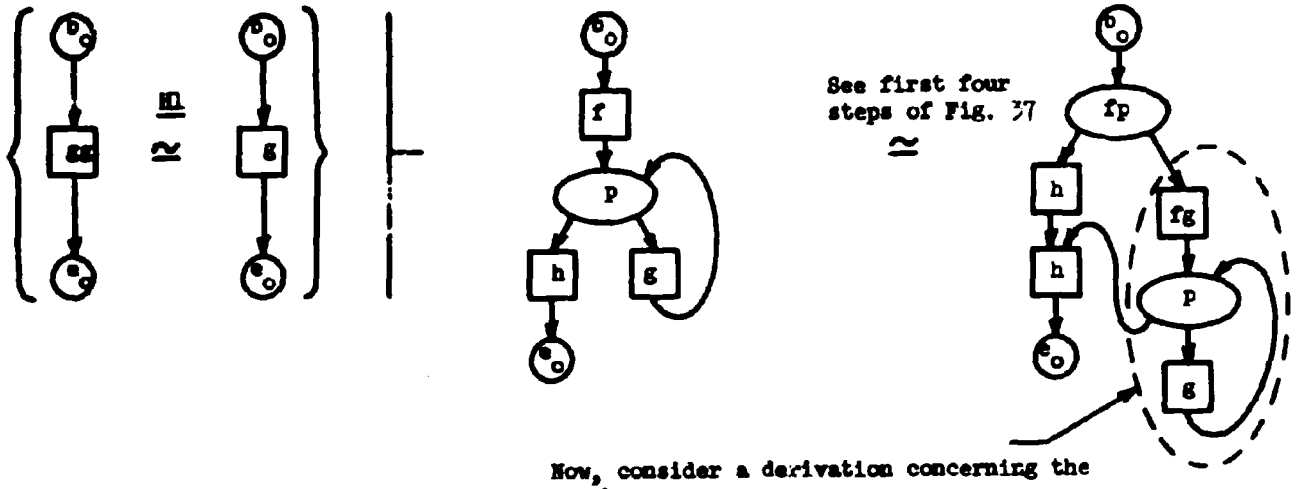


Figure 38(b) contd.

Detection of an always indeterminate E-program.



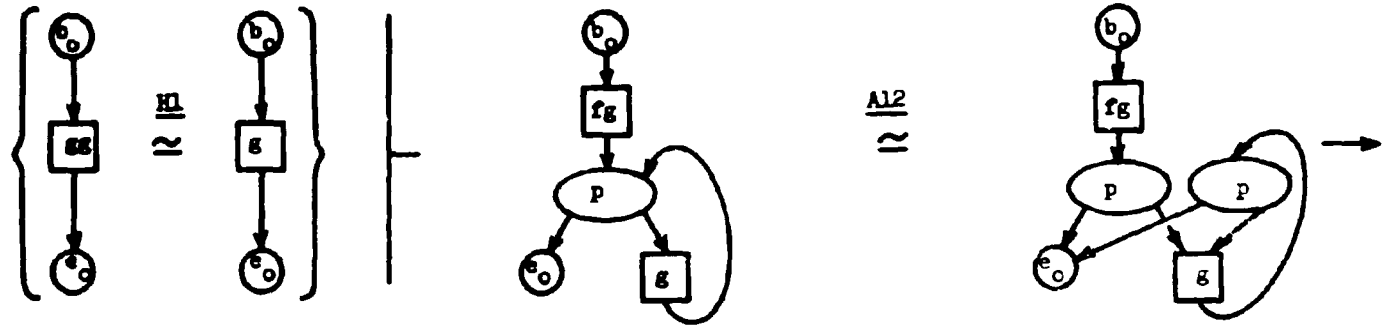
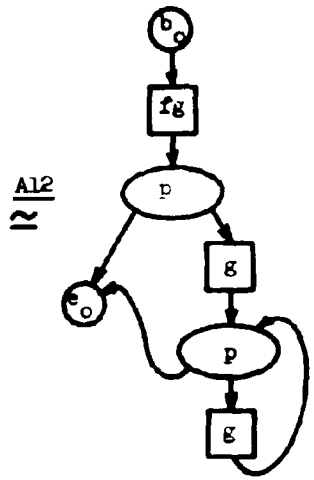
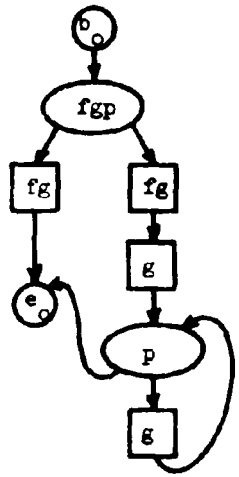


Figure 39

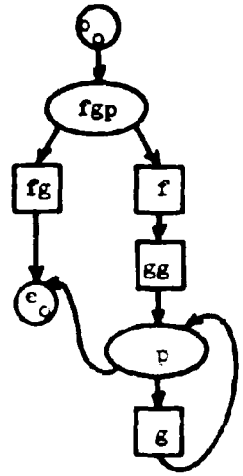
Detecting the halting cases of an E-program. Here, f, g, h are assignment schemata; and p is a qff. Continued next page.



A12 ≃

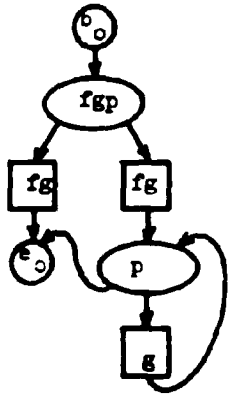


A11 ≃

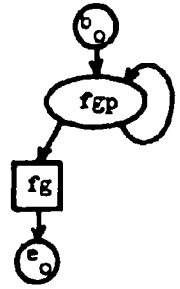


A8 ≃

This last E-program can replace the indicated sub-program above.



H1, A8 ≃



R5 ≃

Figure 39 contd.

Detecting the halting cases of an E-program. Continued next page.

So, we have

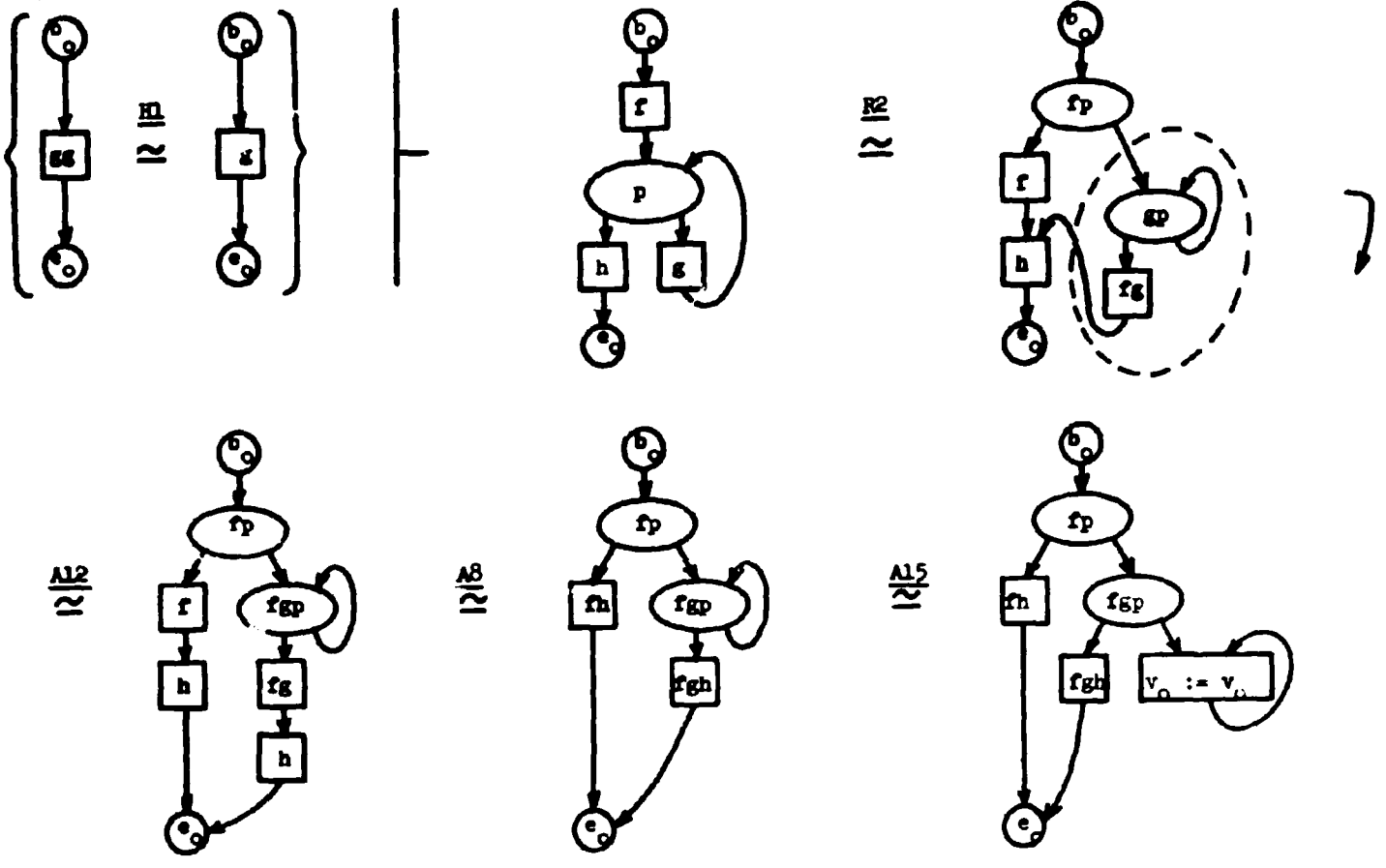


Figure 39 contd.

Detecting the halting cases of an E-program.

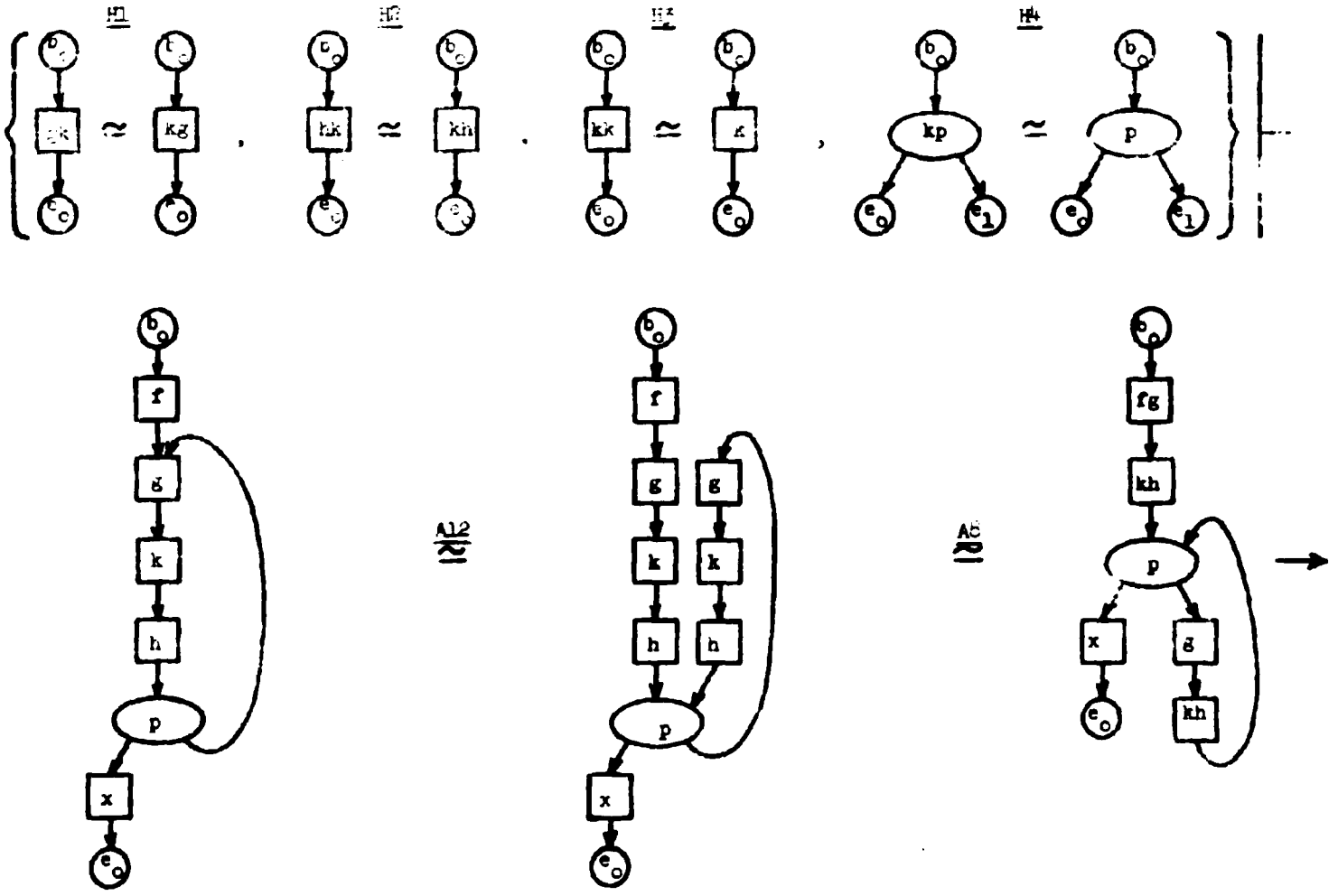


Figure 40

Removal from a loop of a loop-independent operation. Here, f, g, h, k, x are assignment schemata and q is a qff. Continued next page.

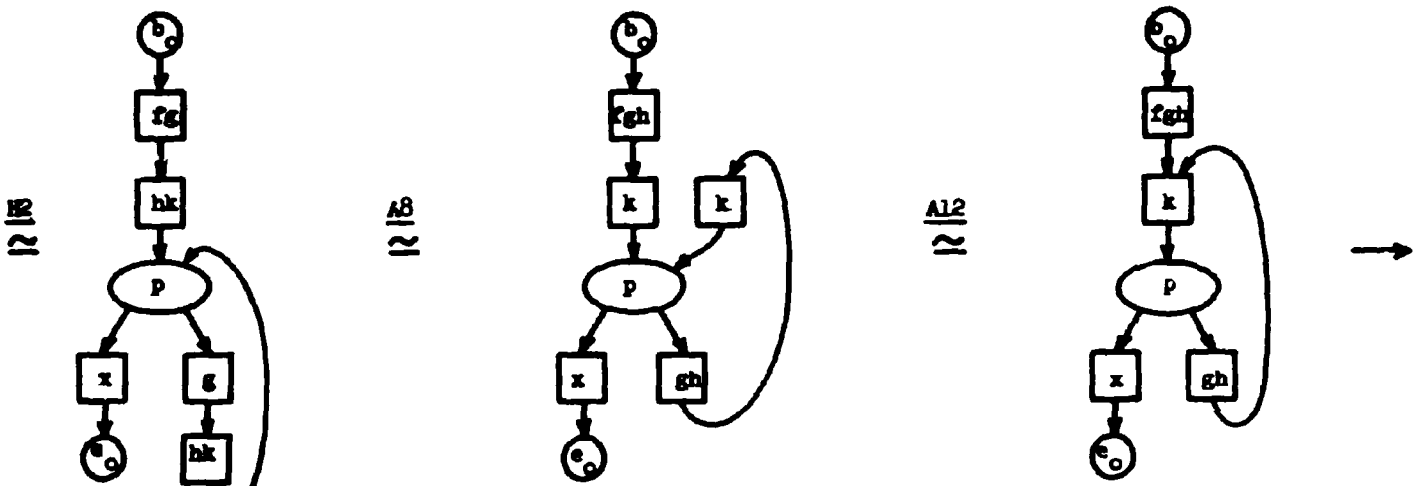


Figure 40 contd.

Removal from a loop of a loop-independent operation. Continued next page.

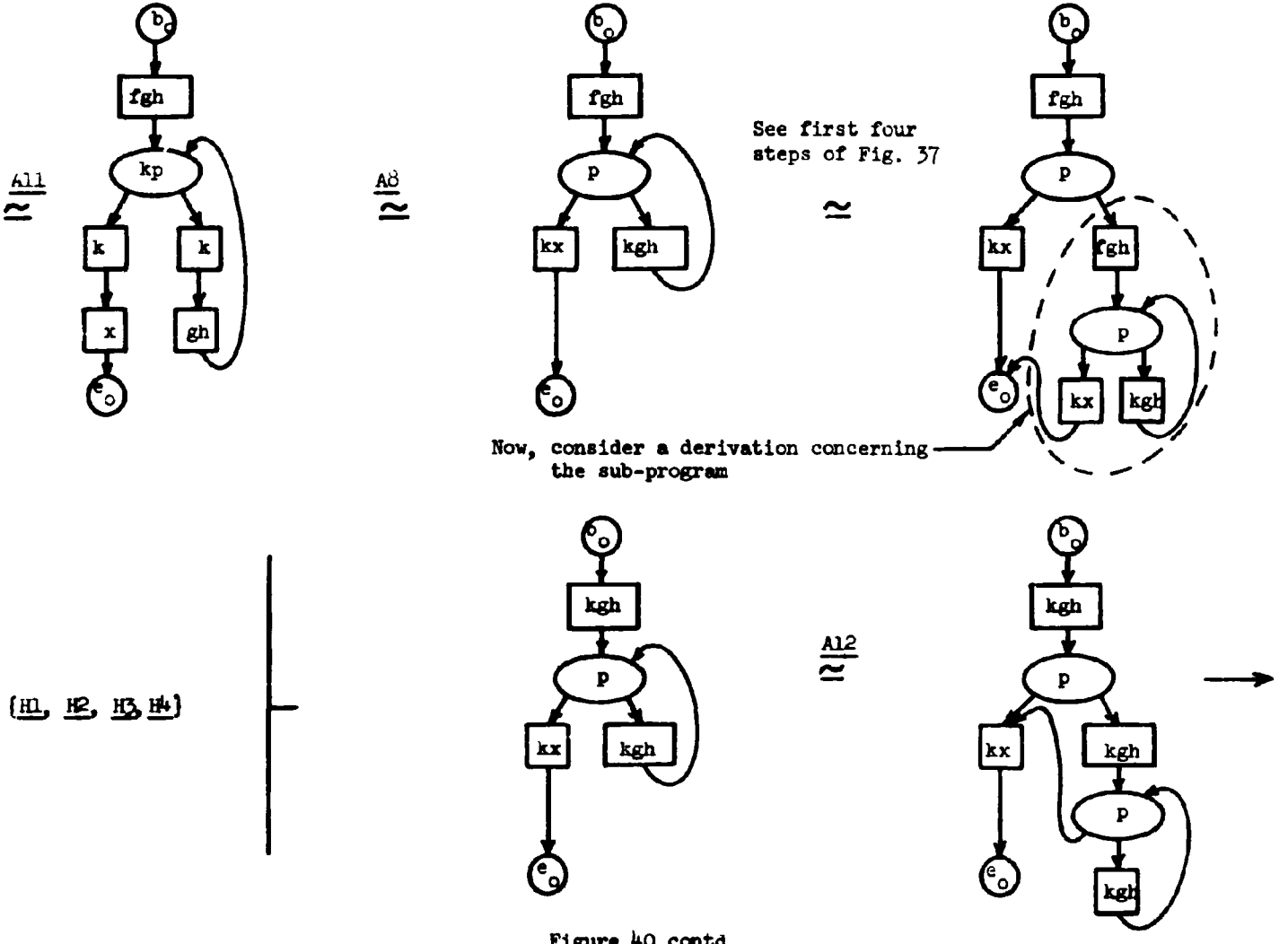


Figure 40 contd.

Removal from a loop of a loop-independent operation. Continued next page.

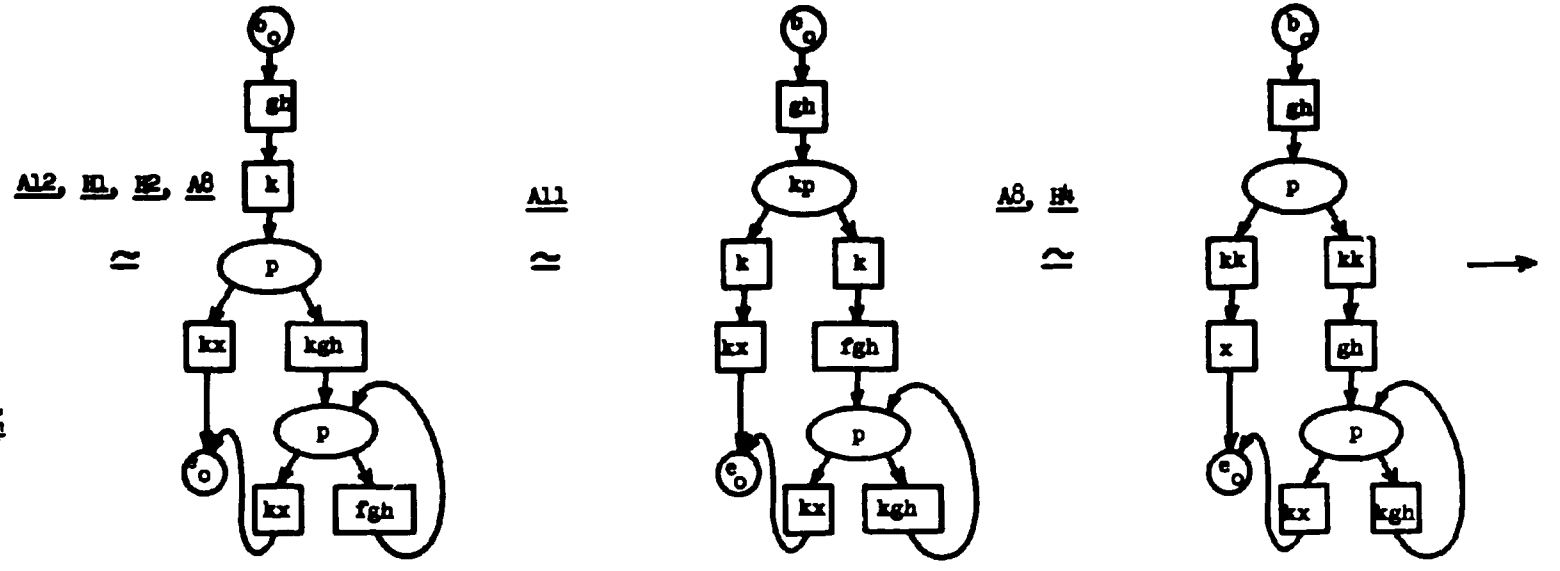


Figure 40 contd.

Removal from a loop of a loop-independent operation. Continued next page.

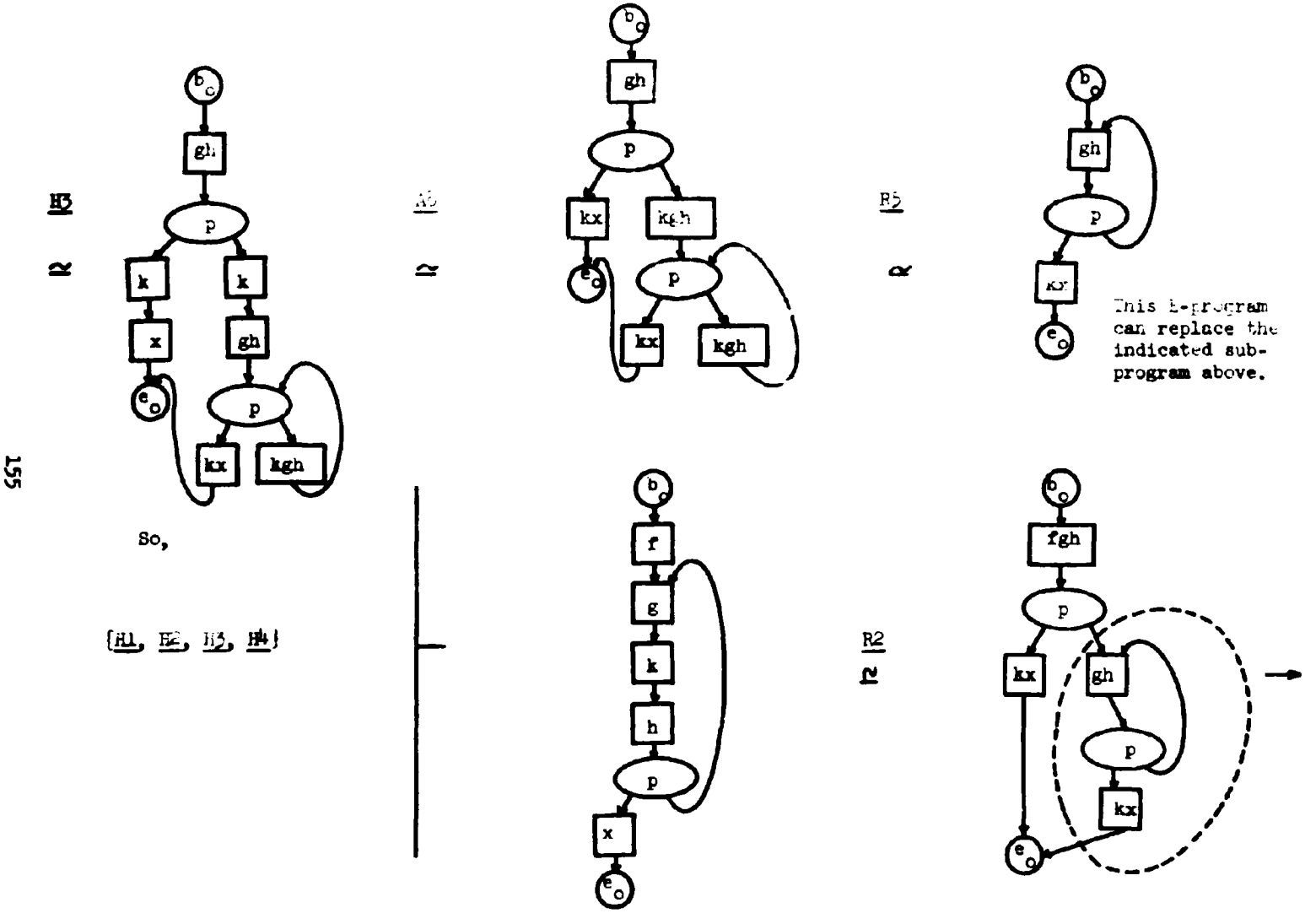


Figure 40 contd.

Removal from a loop of a loop-independent operation. Continued next page.

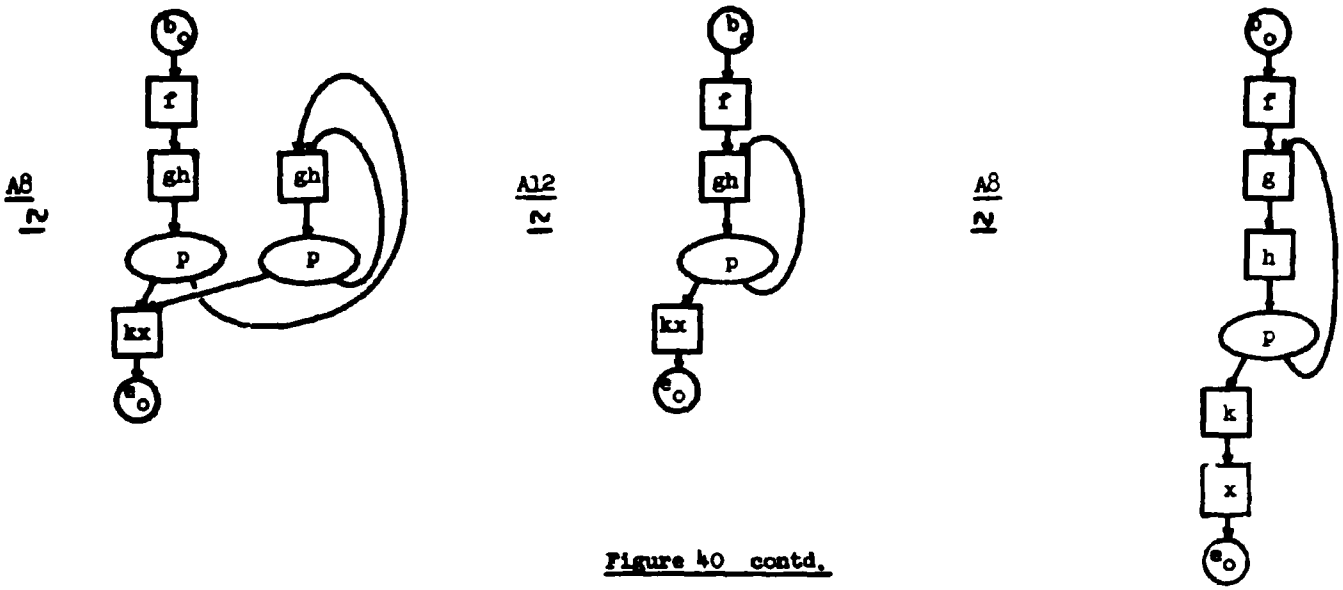
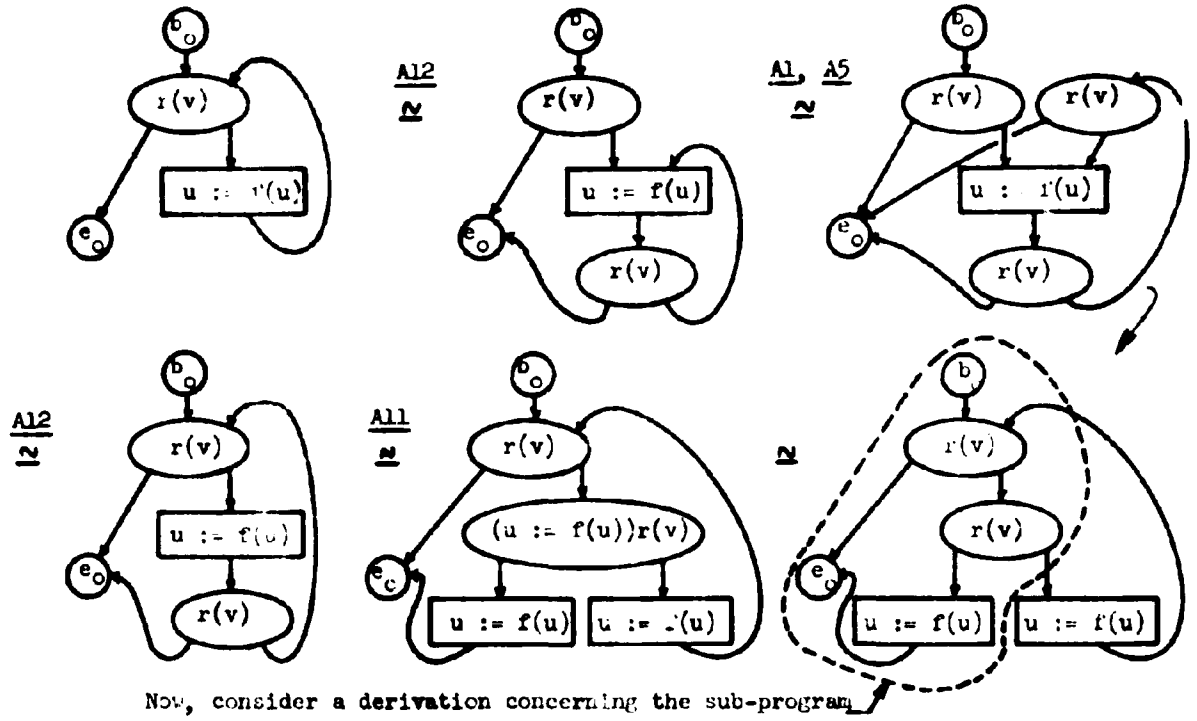


Figure 40 contd.

Removal from a loop of a loop-independent operation.



Now, consider a derivation concerning the sub-program

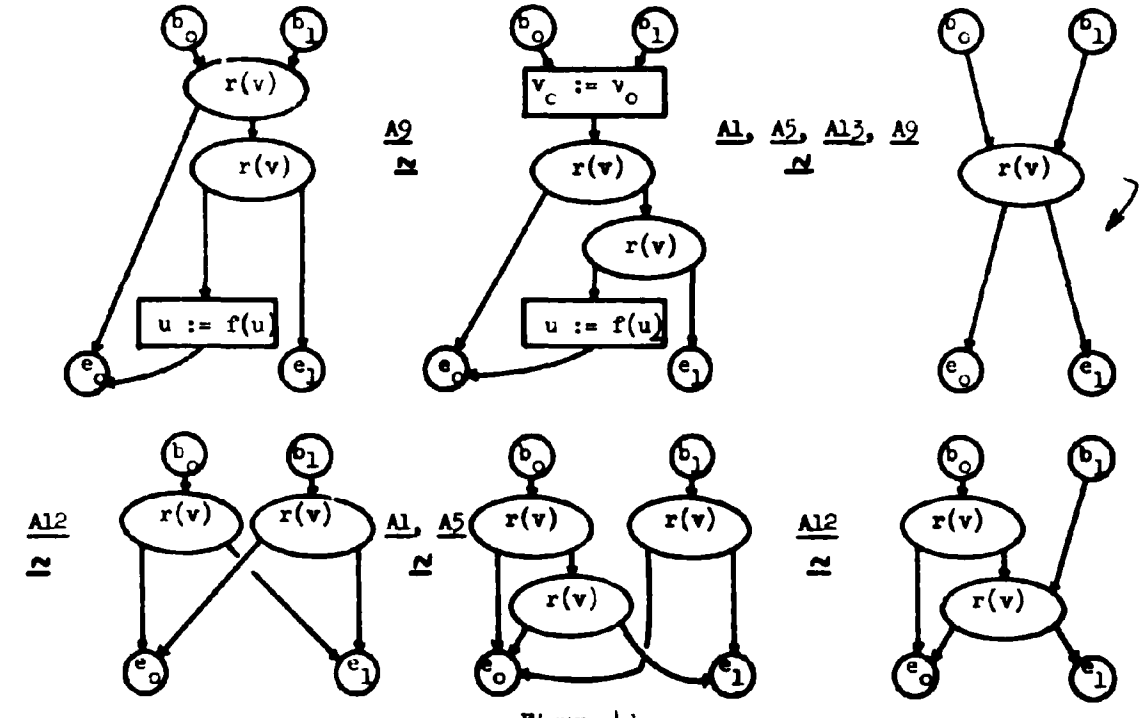


Figure 41

Detection of an indeterminate loop. (Continued next page.) Here, u, v are variables, f is a function letter, and r is a relation letter.

The result of the derivation concerning the indicated sub-program then replaces that sub-program.

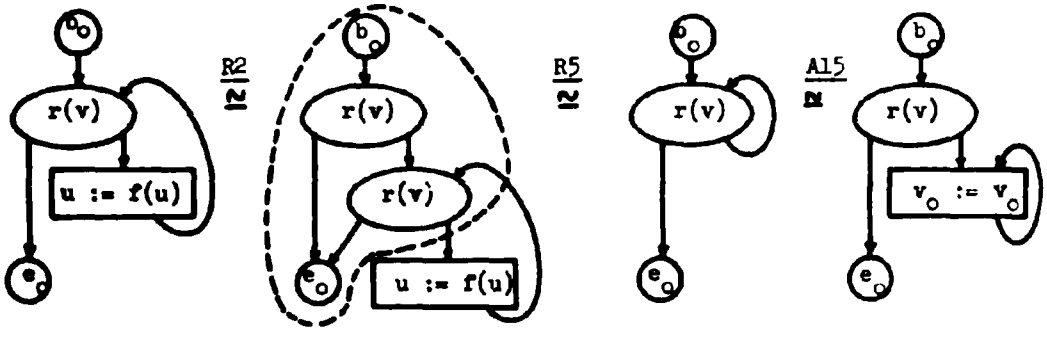
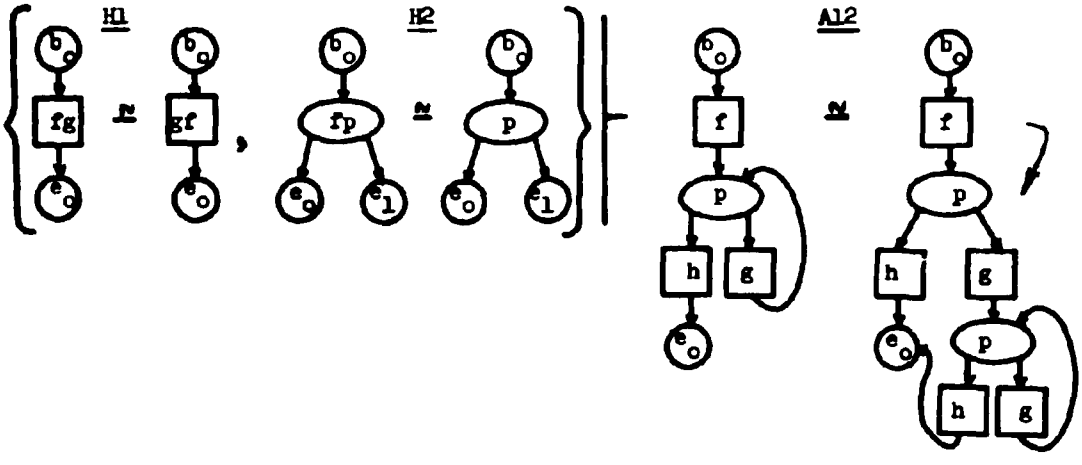


Figure 41 contd.

Detection of an indeterminate loop.



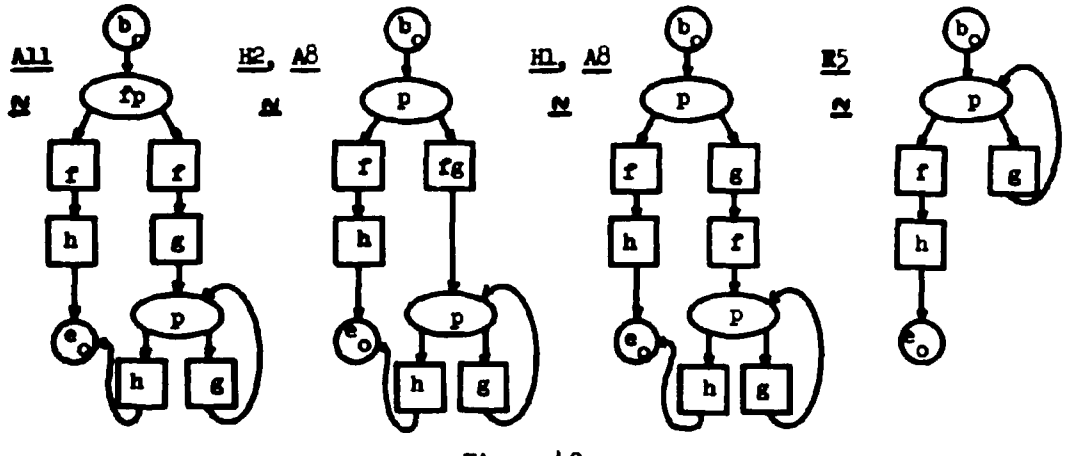


Figure 42

An example of loop-transparency.

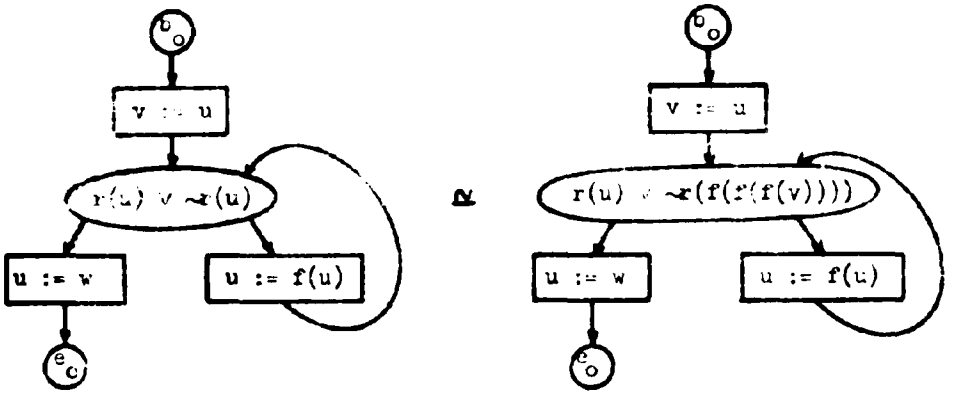
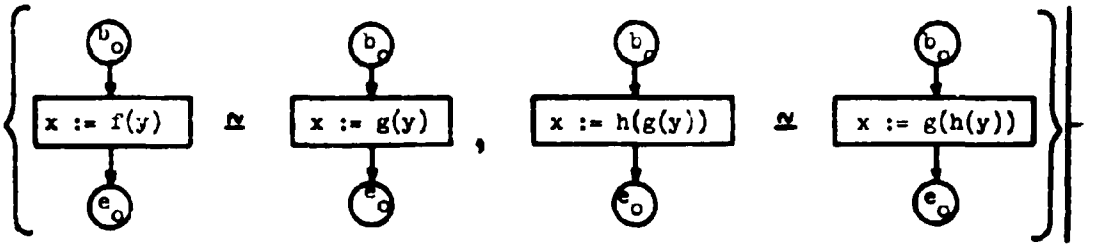


Figure 43

Two strongly equivalent always halting E-programs. The left-hand E-program executes zero circuits of the loop, and the right-hand E-program executes up to three circuits. Here, u, v, w are variables; f is a function letter; and r is a relation letter.



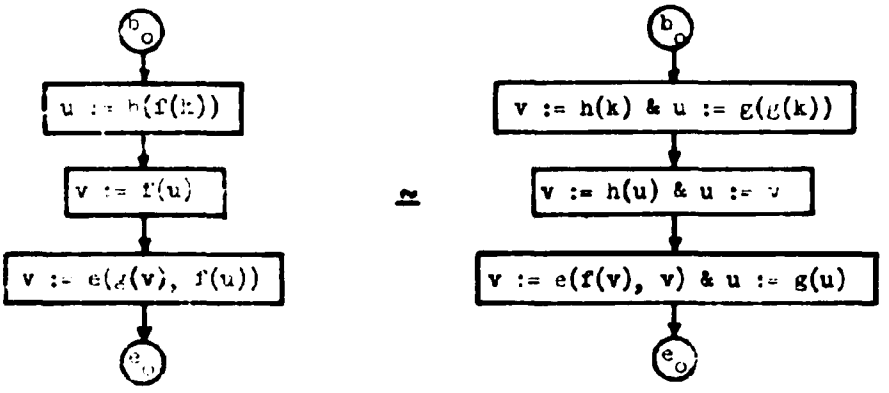


Figure 44

Two assignment schemata, equivalent provided f and g are the same function and g and h commute. Here, x, y, u, v are variables; e, f, g, h are function letters; and k is a constant.

BLANK PAGE

CHAPTER 9

INITIAL CONDITIONS AND K-EVENTS

Representing E-programs as flowcharts is advantageous in that it avoids a plethora of syntactic structure which might impede intuitive understanding and confound meta-level analysis. However, as always, there are two sides to this coin. In fact, the analysis of some properties of E-programs would be made more tractable if a neater, more orderly, syntactic representation were available (cf. the efforts of Bohm and Jacopini [2] in this direction). Many researchers in switching and automata theory, have discovered that certain of their problems yield solutions more readily when studied in terms of regular expressions instead of state transition diagrams (cf. Harrison [15, p. 321]). In this chapter, we will examine how regular expressions and K-expressions, used as an alternative representation for E-programs, also lead to a more productive analysis.

Regular Expressions and Regular Events

Before proceeding, we will repeat here the basic definitions associated with regular expressions and regular events. This material is also given by Salomaa [38], Harrison [15] and by many others; we include it here only to avoid notational misunderstandings. First, we discuss the syntax of regular expressions.

Let $\Sigma = \{x_0, x_1, \ldots, x_{k-1}\}$, $k < \omega$, be an alphabet; here, each letter x_i is assumed to be some formal expression, i.e., perhaps a sequence of symbols on some other lower level alphabet. This possibility will not concern us just now, however.

We build up regular expressions from Σ using the additional symbols: "(", ")", "*", ".", "$\vee$", "0", and "1".

(i) $x_0, x_1, \ldots, x_{n-1}$, 0, 1 are all regular expressions

(ii) If α and β are regular expressions, then $(\alpha \vee \beta)$, $(\alpha \cdot \beta)$ and α^* are regular expressions

(iii) Extremal clause.

In practice, we omit "." from regular expressions of the form $(\alpha \cdot \beta)$. In addition, parentheses are often omitted with the understanding that "." is performed before "$\vee$", and "*" before either "." or "$\vee$". Thus, $\alpha \vee \beta\gamma^*$ is to be read as $(\alpha \vee (\beta \cdot \gamma^*))$.

The semantics of a regular expression over the alphabet Σ yields a certain sub-set of Σ^*, which is the set of all finite words (or simply words if there is no confusion) over Σ, i.e., the free semi-group with identity generated by Σ where the operation is juxtaposition. The subset of Σ^* associated with the regular expression α is denoted by $|\alpha|$ and is called a regular event.

(i) $|x_i| = \{x_i\}$

(ii) $|1| = \{\Lambda$, the empty word, i.e., the identity of $\Sigma^*\}$

(iii) $|0| = \emptyset$, the empty set

(iv) $|\alpha \vee \beta| = |\alpha| \cup |\beta|$

(v) $|\alpha\beta| = \{ab : a \in |\alpha| \ \& \ b \in |\beta|\}$

(vi) $|\alpha^*|$ = the smallest set that contains the empty word, and for any $a \in |\alpha|$ and $b \in |\alpha^*|$ contains the word ab, i.e.,

$$|\alpha^*| = |1| \cup |\alpha| \cup |\alpha\alpha| \cup |\alpha\alpha\alpha| \cup \ldots$$

E-programs as Regular Expressions

Throughout these discussions, we assume some fixed arbitrary signature $\mathcal{S}$.

Each E-program $\mathcal{M} = \langle X, \Gamma, \mathcal{L} \rangle$ has associated with it the alphabet $\Sigma_{\mathcal{M}} = \{u : \text{for some } x \in X, [x] = u\} \cup \{\bar{u} : \text{for some } x \in X(\mathcal{Q}), [x] = u\}$ where here, and in the sequel, we may write $\bar{u}$ instead of $\sim u$ for $u \in \mathcal{Q}$. Thus, $\Sigma_{\mathcal{M}}$ is a finite subset of $\mathcal{O} \cup \mathcal{A} \cup \mathcal{Q} \cup \mathcal{E}$.

We will define $\alpha_{\mathcal{M}}$, a regular expression over $\Sigma_{\mathcal{M}}$ that corresponds to an E-program $\mathcal{M}$, by utilizing a finite automaton $M_{\mathcal{M}}$ that accepts the set $|\alpha_{\mathcal{M}}|$, i.e., given a word x as input, $M_{\mathcal{M}}$ reaches the final state iff $x \in |\alpha_{\mathcal{M}}|$. Let us define these ideas in more detail.

Given the E-program $\mathcal{M} = \langle X, \Gamma, \mathcal{L} \rangle$, we define the finite automaton $M_{\mathcal{M}} = \langle S, T, \Sigma_{\mathcal{M}} \rangle$, where S is a finite set of <u>automaton states</u> (or simply <u>states</u> if no confusion with states as sequences over a domain results), and $T : S \times \Sigma_{\mathcal{M}} \to S$ is a <u>transition function</u>. The set S of states is

$S = (X - X(\mathcal{O})) \cup \{b, e, d\}$

where $\{b, e, d\} \cap X = \emptyset$, b is the <u>start</u> state, e is the <u>final</u> state and d is the <u>dead</u> state.

The transition function is defined as follows.

(i) If $x \in X(\mathcal{A})$ and $\Gamma x = y$, then $T(x, [x]) = y$ and $T(x, u) = d$ for all $u \in \Sigma_{\mathcal{M}} - \{[x]\}$.

(ii) If $x \in X(\mathcal{Q})$ and $\Gamma x = \langle y, z \rangle$, then $T(x, [x]) = y$, $T(x, \sim[x]) = z$ and $T(x, u) = d$ for all $u \in \Sigma_{\mathcal{M}} - \{[x], \sim[x]\}$.

(iii) If $x \in X(\mathcal{E})$, then $T(x, [x]) = e$ and $T(x, u) = d$ for all $u \in \Sigma_{\mathcal{M}} - \{[x]\}$.

(iv) $T(b, [x]) = y$ for all $z \in X(\mathfrak{B})$ where $\Gamma x = y$, and $T(b, u) = d$ for all $u \in \Sigma_{\mathfrak{M}} - \{b_0, \ldots, b_{m-1}\}$ where $\mathfrak{M}$ is type $\langle m, n \rangle$, i.e., has m initiators.

(v) $T(e, u) = d$ for all $u \in \Sigma_{\mathfrak{M}}$.

(vi) $T(d, u) = d$ for all $u \in \Sigma_{\mathfrak{M}}$.

These not altogether pellucid definitions can be rendered more informative by referring to Figure 45. Here the diagrammatic representation of an E-program $\mathfrak{M}$ and a partial transition diagram (pdt) for $M_{\mathfrak{M}}$ are illustrated. The pdt is partial in that the dead state has been omitted as have all transitions to it. As we now see, the formation of $M_{\mathfrak{M}}$ from $\mathfrak{M}$ is really a trivial operation. Nevertheless, this characterization of $\mathfrak{M}$ as a finite automaton $M_{\mathfrak{M}}$ enables us to apply many well understood powerful techniques to the analysis of the strong equivalence problem for E-programs.

The behavior of the automaton $M_{\mathfrak{M}}$ is simply the set of words in $\Sigma_{\mathfrak{M}}^*$ that $M_{\mathfrak{M}}$ accepts, i.e., that cause $M_{\mathfrak{M}}$ to go from the start state to the final state via the transition function. Let us define the acceptor function $T^* : S \times \Sigma_{\mathfrak{M}}^* \to S$ as follows

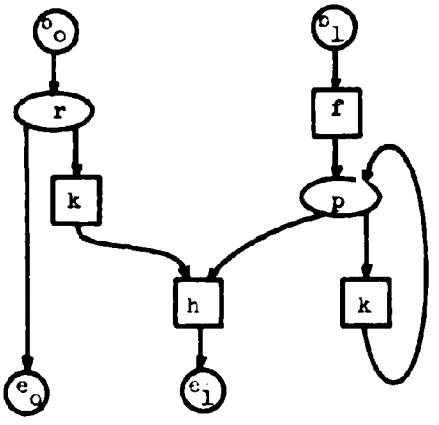
(i) $T^*(x, \Lambda) = x$

(ii) $T^*(x, \sigma w) = T^*(T(x, \sigma), w)$ for $\sigma \in \Sigma_{\mathfrak{M}}$, $w \in \Sigma_{\mathfrak{M}}^*$.

Then the behavior of $M_{\mathfrak{M}}$ is $B_{\mathfrak{M}} = \{w \in \Sigma_{\mathfrak{M}}^* : T^*(b, w) = e\}$. From the construction of $M_{\mathfrak{M}}$, $w \in B_{\mathfrak{M}}$ will begin with an initiator and end with a terminator. (Note: the semi-group operation of juxtaposition does not further imply forward substitution as discussed in Chapter 6.)

Theorem 18: There exists an effective procedure, which for any finite automaton M constructs a regular expression α such that $|\alpha|$ is the behavior of M.

Thus, the regular expression $\alpha_{\mathfrak{M}}$, where $|\alpha_{\mathfrak{M}}| = B_{\mathfrak{M}}$, is effectively constructable.



E-program $\mathfrak{A}$

Partial Transition Diagram for $M_{\mathfrak{A}}$

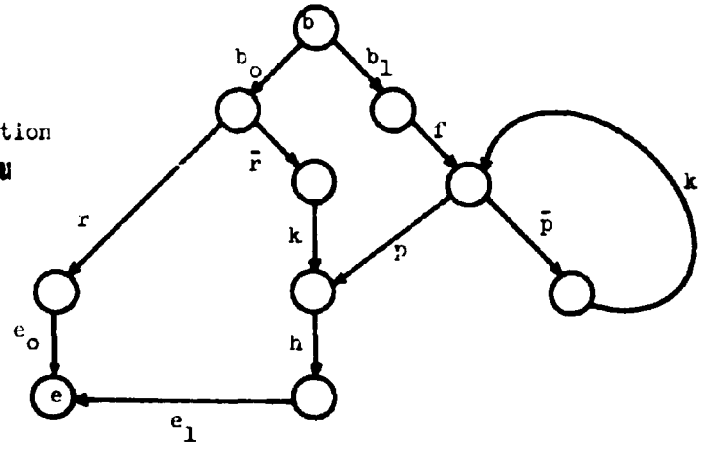


Figure 45

An E-program $\mathfrak{A}$ and the partial transition diagram for $M_{\mathfrak{A}}$. Here, r, p are qffs; and f, h, k are assignment schemata.

Proof: This result is due to Kleene [23]. See also Harrison [15]. ∎

Remarks:

(i) The procedure of Theorem 18 is involved and complicated, and the regular expression produced is usually inordinately large and liable to drastic simplification. In the sequel, we will use heuristic methods for writing down regular expressions in simple form. For example, the E-program $\mathfrak{M}$ of Figure 45 yields

$$\alpha_{\mathfrak{M}} = b_0 re_0 \vee b_0 \bar{r} khe_1 \vee b_1 f(\bar{p}k)^* phe_1$$

(ii) The inherent utility of the regular expression notation is now evident. For, no matter how convoluted and entangled the graph of an E-program may be, its regular expression has a hierarchial structure where, so to speak, all the loops are nested. Thus, while the dr of $\mathfrak{M}$ is easy to understand (and easy to encode or program up), the regular expression $\alpha_{\mathfrak{M}}$ is easier to analyze.

(iii) It would seem that there is really something very primitive and pervasive about the ideas involved with regular expressions and regular events. They play a key role in many areas, have a potential role in several others, e.g., pure graph theory or artificial languages (cf. Tixier [41]), and their application to the analysis of E-programs is quite natural and productive.

We now want to consider the relationship between executions of $\mathfrak{M}$ and words in $|\alpha_{\mathfrak{M}}|$. Clearly, we can associate a word in $|\alpha_{\mathfrak{M}}|$ with every halting execution of $\mathfrak{M}$, but the converse is not always true. That is, certain paths through $\mathfrak{M}$ (i.e., certain words in $|\alpha_{\mathfrak{M}}|$) starting at some initiator b_i and ending with some terminator e_j may not be executable. In formulating the initial conditions for Figure 5, we appealed to an intuitive notion of unexecutable paths, and now we will consider these matters in more detail.

We define $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ to be the (possibly infinite) word over $\Sigma_{\mathfrak{M}}$ associated with the execution $E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ of E-program $\mathfrak{M}$ starting at node x. Eventually we shall want to show that $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \in |\alpha_{\mathfrak{M}}|$ when $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite and $x \in X(\mathcal{B})$.

(i) If $x \in X(\mathcal{B})$, then

$W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x)$

(ii) If $x \in X(\mathcal{A})$, then

$W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = [x]\, W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x)$

(iii) If $x \in X(\mathcal{Q})$ and $\Gamma x = \langle y, z \rangle$, then

$W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, y)$ if $[x][\underset{\sim}{D}, \xi]$

$= \sim [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, z)$ otherwise

(iv) If $x \in X(\mathcal{E})$, then

$W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = [x]$.

So the function W follows through $\mathfrak{M}$, just as the execution function E would, except that here a word over $\Sigma_{\mathfrak{M}}$ is built up as we proceed through the E-program. Of course, W produces a finite word iff the execution given by E halts. To study such finite words, we need the following.

Theorem 19: For any E-program $\mathfrak{M} = \langle X, \Gamma, \mathcal{X} \rangle$, computing structure $\underset{\sim}{D}$, state $\xi : \omega \to \underset{\sim}{D}_0$ and $x \in X(\mathcal{B})$, if $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite, then $T^*(b, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)) = e$.

Proof: First we do an induction on the word $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ to show that for any $x \in X - X(\mathcal{B})$, $T^*(x, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)) = e$.

(i) The primitive basis of induction is the case $x \in X(\mathcal{E})$, where we have

$T^*(x, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= T^*(x, [x])$ by the definition of W

$= T^*(T(x, [x]), \Lambda)$ by the definition of T^*

$= T^*(e, \Lambda)$ by the definition of T

$= e$ by the definition of T^* .

(ii) If $x \in X(\mathcal{A})$, then

$T^*(x, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= T^*(x, [x]\, W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x))$ by the definition of W

$= T^*(T(x, [x]), W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x))$ by the definition of T^*

$= T^*(\Gamma x, W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x))$ by the definition of T

$= e$ by induction hypothesis .

(iii) If $x \in X(\mathcal{C})$ and $\Gamma x = \langle y, z \rangle$, then

$T^*(x, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= T^*(x, [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))$ if $[x][\underset{\sim}{D}, \xi]$ or

$T^*(x, \sim[x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))$ otherwise, by the definition of W

$= T^*(T(x, [x]), W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))$ if $[x][\underset{\sim}{D}, \xi]$ or

$T^*(T(x, \sim[x]), W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))$ otherwise, by the definition of T^*

$= T^*(y, W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))$ if $[x][\underset{\sim}{D}, \xi]$ or

$T^*(z, W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))$ otherwise, by the definition of T

$= e$ if $[x][\underset{\sim}{D}, \xi]$ or e otherwise, by induction hypothesis

$= e$

and this completes the induction.

Now consider $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ where $x \in X(\mathcal{B})$.

$T^*(b, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= T^*(b, [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))$ by the definition of W

$= T^*(T(b, [x]), W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))$ by the definition of T^*

$= T^*(\Gamma x, W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))$ by the definition of T

$= e$ from the result obtained above, since $\Gamma x \notin X(\mathfrak{H})$

by the definition of E-programs. ∎

Since $T^*(b, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)) = e \Rightarrow W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \in B_{\mathfrak{M}}$, and since $B_{\mathfrak{M}} = |\alpha_{\mathfrak{M}}|$ by Theorem 18, then an immediate corollary to Theorem 19 is that if $x \in X(\mathfrak{H})$ and $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite, then $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \in |\alpha_{\mathfrak{M}}|$.

To illustrate the connection between $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ and $E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$, we introduce a function E^* which "executes" the word $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$.

(i) If $\sigma \in \mathcal{Q} \cup \mathcal{B}$ and $w \in \Sigma_{\mathfrak{M}}^*$, then

$E^*(\underset{\sim}{D}, \xi, \sigma w) = E^*(\underset{\sim}{D}, \xi, w)$

(ii) If $\sigma \in \mathcal{A}$ and $w \in \Sigma_{\mathfrak{M}}^*$, then

$E^*(\underset{\sim}{D}, \xi, \sigma w) = E^*(\underset{\sim}{D}, \sigma[\underset{\sim}{D}, \xi], w)$

(iii) If σ is $e_j \in \mathcal{E}$, then

$E^*(\underset{\sim}{D}, \xi, \sigma) = E^*(\underset{\sim}{D}, \xi, e_j) = \langle \xi, \mathcal{D} \rangle$

The function E^* simply applies each assignment schema encountered to the current state and bypasses the initiator and all qffs. The relationship between the execution functions E and E^*, and the function W is then given by the following

Theorem 20: For any E-program $\mathfrak{M} = \langle \mathcal{X}, \Gamma, \mathcal{E} \rangle$, computing structure $\underset{\sim}{D}$, state $\xi : \omega \to \underset{\sim}{D}_0$ and $x \in X$,

$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \simeq E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$.

Proof: Notice first that $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite, i.e., $E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$ is determinate, iff $E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is determinate. We give an inductive proof for the case of halting execution.

(i) The primitive basis of induction is the case $x \in X(\mathcal{E})$ where $[x] = e_j$. Then, the left-hand side is

$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) = \langle \xi, j \rangle$ by the definition of E,

and the right-hand side is

$E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= E^*(\underset{\sim}{D}, \xi, e_j)$ by the definition of W

$= \langle \xi, j \rangle$ by the definition of E^*, which is identical to the left-hand side.

(ii) If $x \in X(\mathcal{A})$, then the left-hand side is

$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$

$= E(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x)$ by the definition of E

$= E^*(\underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x))$ by induction hypothesis,

and the right-hand side is

$E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= E^*(\underset{\sim}{D}, \xi, [x]\, W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x))$ by the definition of W

$= E^*(\underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], W(\mathfrak{M}, \underset{\sim}{D}, [x][\underset{\sim}{D}, \xi], \Gamma x))$ by the definition of E^*,

which is identical to the left-hand side.

(iii) If $x \in X(\mathcal{B})$, where $\Gamma x = \langle y, z \rangle$, then the left-hand side is

$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$

$= E(\mathfrak{M}, \underset{\sim}{D}, \xi, y)$ if $[x][\underset{\sim}{D}, \xi]$ or

$E(\mathfrak{M}, \underset{\sim}{D}, \xi, z)$ otherwise, by the definition of E

$= E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))$ if $[x][\underset{\sim}{D}, \xi]$ or

$E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))$ otherwise, by induction hypothesis,

and the right-hand side is

$E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= E^*(\underset{\sim}{D}, \xi, [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))$ if $[x][\underset{\sim}{D}, \xi]$ or

$E^*(\underset{\sim}{D}, \xi, \sim[x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))$ otherwise, by the definition of W

$= E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))$ if $[x][\underset{\sim}{D}, \xi]$ or

$E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))$ otherwise, by the definition of E^*,

which is identical to the left-hand side.

(iv) If $x \in X(\overline{\mathfrak{B}})$, then the left-hand side is

$E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$

$= E(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x)$ by the definition of E

$= E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))$ by induction hypothesis,

and the left-hand side is

$E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$

$= E^*(\underset{\sim}{D}, \xi, [x]\, W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))$ by definition of W

$= E^*(D, \xi, W(\mathfrak{M}, D, \xi, \Gamma x))$ by definition of E^*, which is identical to the left-hand side, and this completes the induction. ∎

We see then that executions of an E-program $\mathfrak{M}$ give rise to words in $|\alpha_{\mathfrak{M}}|$ that can themselves be "executed" producing the same result. Intuitively, the qffs encountered during the execution of an E-program $\mathfrak{M}$ specify the condition on the input state for that execution; the qffs in the corresponding word in $|\alpha_{\mathfrak{M}}|$ play the same role with regard to the execution of that word. In Chapter 5, we called this the <u>initial condition</u>; we now develop this notion in detail.

We first define a function $\pi : \mathcal{W}_{\mathfrak{M}} \to \Sigma_{\mathfrak{M}}^*$, where $\mathcal{W}_{\mathfrak{M}} = \{u \in \Sigma_{\mathfrak{M}}^* - \{\Lambda\} :$ there exists $w \in \Sigma_{\mathfrak{M}}^*$ such that $wu \in |\alpha_{\mathfrak{M}}|\}$. Notice that $|\alpha_{\mathfrak{M}}| \subseteq \mathcal{W}_{\mathfrak{M}}$. Applying π to a word, is termed applying "push through";

we will see how this ties in with the notion of push through associated with axiom schema All of $\mathcal{A}x$. In Chapter 6, we denoted the forward substitution of an assignment schema into either another assignment schema or a qff by juxtaposition. To avoid conflict with the juxtaposition of letters in $\Sigma_{\mathcal{G}}$ to form words in $\Sigma_{\mathcal{G}}^*$, we will denote the forward substitution of an assignment schema f into t by $(f \to t)$, where $t \in \mathcal{A} \cup \mathcal{Q}$. In addition, we adopt that the convention that $\to$ associates to the left, so that $(f \to g \to h \to p)$ denotes $(((f \to g) \to h) \to p)$. Now, we are ready to define the function π . (Note: for convenience, the parentheses around the argument of π are dropped.)

$$\pi b_i w = b_i \pi w$$
$$\pi f g w = \pi(f \to g) w$$
$$\pi f p w = (f \to p) \pi f w$$
$$\pi p w = p \pi w$$
$$\pi e_j = e_j$$
$$\pi f e_j = f e_j$$

where $f, g \in \mathcal{A}$, $p \in \mathcal{Q}$ and $w \in \Sigma_{\mathcal{G}}^*$. This definition clearly covers all the cases of πu for $u \in \mathcal{W}_{\mathcal{G}}$.

To illustrate the effect of the push through function π , we consider a few examples.

(i) Suppose $b_i x p y z q r f e_j \in |\alpha_{\mathcal{G}}|$, where $x, y, z, f \in \mathcal{A}$ and $p, q, r \in \mathcal{Q}$. Then,

$\pi b_i x p y z q r f e_j$
$= b_i \pi x p y z q r f e_j$
$= b_i (x \to p) \pi x y z q r f e_j$

$= b_i(x \to p)\pi(x \to y)zqrfe_j$

$= b_i(x \to p)\pi(x \to y \to z)qrfe_j$

$= b_i(x \to p)(x \to y \to z \to q)\pi(x \to y \to z)rfe_j$

$= b_i(x \to p)(x \to y \to z \to q)(x \to y \to z \to r)\pi(x \to y \to z)fe_j$

$= b_i(x \to p)(x \to y \to z \to q)(x \to y \to z \to r)\pi(x \to y \to z \to f)e_j$

$= b_i(x \to p)(x \to y \to z \to q)(x \to y \to z \to r)(x \to y \to z \to f)e_j$

(ii) Consider once again the E-program $\mathfrak{M}$ of Figure 3(b), that was used in the proof of Theorem 1 (Chapter 5). Here,

$$\alpha_{\mathfrak{M}} = b_o v := w(\sim r(u)u := v \;\&\; v := u)^*r(u)(\sim\sim r(w))^*\sim r(w)e_o ,$$

where u, v, w are variables and r is a relation letter. Consider the path through $\mathfrak{M}$ that executes each loop zero times. The word in $|\alpha_{\mathfrak{M}}|$ corresponding to that path is

$$x_1 = b_o v := wr(u)\sim r(w)e_o$$

and applying the push through function, we have

πx_1

$= b_o \pi v := wr(u)\sim r(w)e_o$

$= b_o(v := w \to r(u))\pi v := w\sim r(w)e_o$

$= b_o r(u)(v := w \to \sim r(w))\pi v := we_o$

$= b_o r(u)\sim r(w)v := we_o$

Now consider the path that executes the upper loop once and the lower loop zero times. The word in $|\alpha_{\mathfrak{M}}|$ corresponding to that path is

$$x_2 = b_o v := w\sim r(u)u := v \;\&\; v := ur(u)\sim r(w)e_o$$

and applying the push through function, we have

πx_2

$= b_o \pi v := w\sim r(u)u := v \;\&\; v := ur(u)\sim r(w)e_o$

$= b_o(v := w \to \sim r(u))\pi v := wu := v \;\&\; v := ur(u)\sim r(w)e_o$

$= b_o r(u)\pi(v := w \to u := v \;\&\; v := u)r(u)\sim r(w)e_o$

$= b_o r(u)\pi u := w \;\&\; v := ur(u)\sim r(w)e_o$

$= b_o r(u)(u := w \;\&\; v := u \to r(u))\pi u := w \;\&\; v := u\sim r(w)e_o$

$= b_o r(u)r(w)(u := w \;\&\; v := u \to \sim r(w))\pi u := w \;\&\; v := ue_o$

$= b_o r(u)r(w)\sim r(w)u := w \;\&\; v := ue_o$

From πx, we can abstract the qff $r(u) \wedge r(w) \wedge \sim r(w)$. In the proof of Theorem 1, these qffs were called the initial conditions for the paths (i.e., words) in question. Specifically, we define the functions $I : W_{\Sigma} \to \mathcal{Q}$, $A : W_{\Sigma} \to \mathcal{A}$, $Y_b : |\alpha_{\Sigma}| \to \omega$ and $Y_e : W_{\Sigma} \to \omega$, as follows. First notice that for any $w \in W_{\Sigma}$, πw is of the form $\beta p_o p_1, \ldots, p_{n-1} \varphi e_j$ where $\beta \in \mathcal{B} \cup \{\Lambda\}$, $p_k \in \mathcal{Q}$, $k < n < \omega$, and $\varphi \in \mathcal{A} \cup \{\Lambda\}$. Then, $I(w) = (v_o = v_o)$ i.e., $I(w)$ is identically true, if $n = 0$, or $I(w) = p_o \wedge p_1 \wedge \ldots \wedge p_{n-1}$ otherwise. Also, $A(w) = v_o := v_o$, i.e., $A(w)$ is the identity operator, if $\varphi = \Lambda$, or $A(w) = \varphi$ otherwise. Finally, $Y_e(w) = j$, and for any $w \in |\alpha_{\Sigma}|$ where $w = b_i u$ for some $u \in W_{\Sigma}$, $Y_b(w) = i$. We say that $A(w)$ is the operation of w, and that $I(w)$ is the initial condition of that operation or of the word w itself.

Intuitively, the soundness of the push through axiom schema All (given by Theorem 8) means that the push through function π gives us the right qff for the initial condition of a word. Furthermore, the soundness of the forward substitution axiom schema A8 (given by Theorem 8) means that π gives us the right assignment schema for the operation of a word. This latter notion is explicated in the following

Theorem 21: *For any E-program $\mathfrak{P} = \langle X, \Gamma, \mathcal{X} \rangle$, computing structure $\underset{\sim}{D}$, state $\underset{\sim}{s} : \omega \to \underset{\sim}{D}_o$ and word $w \in |\alpha_{\Sigma}|$,*

$\underline{E}^{\pi}(\underset{\sim}{D}, \underset{\sim}{s}, w) = \langle A(w)[\underset{\sim}{D}, \underset{\sim}{s}], Y_e(w) \rangle$.

Proof: We given an inductive proof of an even stronger result. In fact, we prove the statement of the theorem for all $w \in \mathcal{W}_{\Sigma}$, and since $|\alpha_{\Sigma}| \subseteq \mathcal{W}_{\Sigma}$, this includes the case $w \in |\alpha_{\Sigma}|$.

(i) The first case in the primitive basis of induction is $w = e_j$. Then, the left-hand side is

$E^*(\underset{\sim}{D}, \xi, e_j) = \langle \xi, \perp \rangle$ by the definition of E^*,

and the right-hand side is

$\langle A(e_j)[\underset{\sim}{D}, \xi], Y_e(e_j) \rangle$

$= \langle v_o := v_o[\underset{\sim}{D}, \xi], \perp \rangle$ by the definitions of A and Y_e

$= \langle \xi, \perp \rangle$ by the definition of semantics for assignment schemata, which is identical to the left-hand side.

(ii) The second case in the primitive basis of induction is $w = fe_j$, where $f \in \mathcal{A}$. Then the left-hand side is

$E^*(\underset{\sim}{D}, \xi, fe_j)$

$= E^*(\underset{\sim}{D}, f[\underset{\sim}{D}, \xi], e_j)$ by the definition of E^*

$= \langle f[\underset{\sim}{D}, \xi], \perp \rangle$ by the definition of E^*,

and the right-hand side is

$\langle A(fe_j)[\underset{\sim}{D}, \xi], Y_e(fe_j) \rangle$

$= \langle f[\underset{\sim}{D}, \xi], \perp \rangle$ by the definitions of A and Y_e, which is identical to the left-hand side.

(iii) If $w = pu$, where $p \in \mathcal{P}$, then the left-hand side is

$E^*(\underset{\sim}{D}, \xi, pu)$

$= E^*(\underset{\sim}{D}, \xi, u)$ by the definition of E^*

$= \langle A(u)[\underset{\sim}{D}, \xi], Y_e(u) \rangle$ by induction hypothesis,

and the right-hand side is

$\langle A(pu)[\underset{\sim}{D}, \xi], Y_e(pu)\rangle$

$= \langle A(u)[\underset{\sim}{D}, \xi], Y_e(u)\rangle$ by the definitions of A and Y_e,

which is identical to the left-hand side.

(iv) If $w = fpu$, where $f \in \mathcal{A}$ and $p \in \mathcal{Q}$, then the left-hand side is

$E^*(\underset{\sim}{D}, \xi, fpu)$

$= E^*(\underset{\sim}{D}, f[\underset{\sim}{D}, \xi], pu)$ by the definition of E^*

$= E^*(\underset{\sim}{D}, f[\underset{\sim}{D}, \xi], u)$ by the definition of E^*

$= \langle A(u)[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]], Y_e(u)\rangle$ by induction hypothesis

Before proceeding with the right-hand side, we need the result that forward substitution of assignment schemata is associative. Thus, if $f, g, h \in \mathcal{A}$, we have for any computing structure $\underset{\sim}{D}$ and state $\xi : \omega \to \underset{\sim}{D}_o$,

$((f \to g) \to h)[\underset{\sim}{D}, \xi]$

$= h[\underset{\sim}{D}, (f \to g)[\underset{\sim}{D}, \xi]]$ by Theorem 7

$= h[\underset{\sim}{D}, g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]]$ by Theorem 7

$= (g \to h)[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$ by Theorem 7

$= (f \to (g \to h))$ by Theorem 7

Then, the right-hand side is

$\langle A(fpu)[\underset{\sim}{D}, \xi], Y_e(fpu)\rangle$

$= \langle (f \to A(pu))[\underset{\sim}{D}, \xi], Y_e(u)\rangle$ by the associativity of forward substitution and the definition of Y_e

$= \langle (f \to A(u))[\underset{\sim}{D}, \xi], Y_e(u)\rangle$ by the definition of A

$= \langle A(u)[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]], Y_e(u)\rangle$ by Theorem 7, which is identical to the left-hand side.

(v) If $w = fgu$, where $f, g \in \mathcal{A}$, then the left-hand side is

$E^*(\underset{\sim}{D}, \xi, fgu)$

$= E^*(\underset{\sim}{D}, f[\underset{\sim}{D}, \xi], gu)$ by the definition of E^*

$= E^*(\underset{\sim}{D}, g[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]], u)$ by the definition of E^*

$= E^*(\underset{\sim}{D}, (f \rightarrow g)[\underset{\sim}{D}, \xi], u)$ by Theorem 7

$= \langle A(u)[\underset{\sim}{D}, (f \rightarrow g)[\underset{\sim}{D}, \xi]], Y_e(u)\rangle$ by induction hypothesis,

and the right-hand side is

$\langle A(fgu)[\underset{\sim}{D}, \xi], Y_e(fgu)\rangle$

$= \langle((f \rightarrow g) \rightarrow A(u))[\underset{\sim}{D}, \xi], Y_e(u)\rangle$ by the associativity of forward substitution and the definition of Y_e .

$= \langle A(u)[D, (f \rightarrow g)[\underset{\sim}{D}, \xi]], Y_e(u)\rangle$ by Theorem 7, which is identical to the left-hand side.

(vi) If $w = b_i u$, then the left-hand side is

$E^*(\underset{\sim}{D}, \xi, b_i u)$

$= E^*(\underset{\sim}{D}, \xi, u)$ by the definition of E^*

$= \langle A(u)[\underset{\sim}{D}, \xi], Y_e(u)\rangle$ by induction hypothesis,

and the right-hand side is

$\langle A(b_i u)[\underset{\sim}{D}, \xi], Y_e(b_i u)\rangle$

$= \langle A(u)[\underset{\sim}{D}, \xi], Y_e(u)\rangle$ by the definitions of A and Y_e , which is identical to the left-hand side, and this completes the induction. ∎

From Theorem 19 we found that if $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$, where $x \in X(\mathfrak{B})$ and $[x] = b_i$, is finite, then $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \in |\alpha_{\mathfrak{M}}|$. Also, from the semantics of E-programs and Theorem 20, we have that $\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i\rangle] \simeq E(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \simeq E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$. These results, together with Theorem 21, give us that if $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite, then $\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i\rangle] = \langle A(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi], Y_e(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))\rangle$. Thus, for halting executions, the set $|\alpha_{\mathfrak{M}}|$ completely characterizes the output of $\mathfrak{M}$. We continue along these lines by examining the role of the initial condition $I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$ when $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite.

Theorem 22: For any E-program $\mathfrak{M} = \langle X, \Gamma, \mathcal{L} \rangle$, computing structure $\underset{\sim}{D}$, state $\xi : \omega \to \underset{\sim}{D}_o$ and $x \in X$, if $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite, then

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$.

So the initial condition of a word, produced by a halting execution, must hold.

Proof: We give an inductive proof.

(i) If $x \in X(\xi)$ and $[x] = e_j$, then

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$

$= I(e_j)[\underset{\sim}{D}, \xi]$ by the definition of W

$= (v_o = v_o)[\underset{\sim}{D}, \xi]$ by the definition of x and I, and $(v_o = v_o)[\underset{\sim}{D}, \xi]$ by the definition of semantics for qffs.

(ii) If $x \in (\mathcal{A})$ and $[x] = f$, then

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$

$= I(fW(\mathfrak{M}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma x))[\underset{\sim}{D}, \xi]$ by the definition of W

Notice that for any $w \in \mathcal{W}_{\mathfrak{M}}$, $I(w) = I(xw)$, and suppose that $xW(\mathfrak{M}, \underset{\sim}{D}, \epsilon, x)$ is of the form

$$p_o p_1 \cdots p_{k-1} ((\ldots((f \to g_o) \to g_1) \ldots \to g_{\ell-1}))e_j$$

where $p_i \in \mathcal{Q}$, $i < k < \omega$, and $g_i \in \mathcal{A}$, $i < \ell < \omega$. Then,

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$

$= I(p_o p_1 \cdots p_{k-1} ((\ldots((f \to g_o) \to g_1) \ldots \to g_{\ell-1}))e_j)[\underset{\sim}{D}, \xi]$

$= I(p_o p_1 \cdots p_{k-1}(f \to ((\ldots((g_o \to g_1) \to g_2)\ldots)))e_j)[\underset{\sim}{D}, \xi]$ by the associativity of forward substitution

$= I((p_o \wedge p_1 \wedge \ldots \wedge p_{k-1})(f \to (\ldots))e_j)[\underset{\sim}{D}, \xi]$ by the definition of I

$= I((f \to (q_o \wedge q_1 \wedge \ldots \wedge q_{k-1}))(f \to (\ldots))e_j)[\underset{\sim}{D}, \xi]$ by the definition of x, where $p_i = (f \to q_i)$, $i < k$

$= I(f(q_o \wedge q_1 \wedge \ldots \wedge q_{k-1})(\ldots)e_j)[\underset{\sim}{D}, \xi]$ by the definition of x.

So $(q_0 \wedge q_1 \wedge \ldots \wedge q_{k-1})$ is just $I(W(\mathfrak{M}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma x))$. If we take $I(W(\mathfrak{M}, \underset{\sim}{D}, f[\underset{\sim}{D}, \xi], \Gamma x))[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$ as the induction hypothesis, then we have simply $(q_0 \wedge q_1 \wedge \ldots \wedge q_{k-1})[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$. But,

$(q_0 \wedge q_1 \wedge \ldots \wedge q_{k-1})[\underset{\sim}{D}, f[\underset{\sim}{D}, \xi]]$

$\Leftrightarrow f \rightarrow (q_0 \wedge q_1 \wedge \ldots \wedge q_{k-1})[\underset{\sim}{D}, \xi]$ by Theorem 6

$\Leftrightarrow (p_0 \wedge p_1 \wedge \ldots \wedge p_{k-1})[\underset{\sim}{D}, \xi]$ by the definition of q_i, $i < k$

But, $(p_0 \wedge p_1 \wedge \ldots \wedge p_{k-1})$ is just $I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$, so that finally, $I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$, as required,

(iii) If $x \in X(\mathfrak{B})$, $\Gamma x = \langle y, z \rangle$ and $[x] = p$, then

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$

$\Leftrightarrow I(pW(\mathfrak{M}, \underset{\sim}{D}, \xi, y))[\underset{\sim}{D}, \xi]$ if $p[\underset{\sim}{D}, \xi]$ or

$I(\sim pW(\mathfrak{M}, \underset{\sim}{D}, \xi, z))[\underset{\sim}{D}, \xi]$ if $\sim p[\underset{\sim}{D}, \xi]$, by the definition of W

$\Leftrightarrow (p \wedge I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, y)))[\underset{\sim}{D}, \xi]$ if $p[\underset{\sim}{D}, \xi]$ or

$(\sim p \wedge I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, z)))[\underset{\sim}{D}, \xi]$ if $\sim p[\underset{\sim}{D}, \xi]$, by the definitions of π and I.

$\Leftrightarrow p[\underset{\sim}{D}, \xi] \wedge I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))[\underset{\sim}{D}, \xi]$ if $p[\underset{\sim}{D}, \xi]$ or

$\sim p[\underset{\sim}{D}, \xi] \wedge I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))[\underset{\sim}{D}, \xi]$ if $\sim p[\underset{\sim}{D}, \xi]$, by the definition of semantics for qffs

$\Leftrightarrow I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, y))[\underset{\sim}{D}, \xi]$ if $p[\underset{\sim}{D}, \xi]$ or

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, z))[\underset{\sim}{D}, \xi]$ otherwise, by the definition of semantics for qffs, and this holds by induction hypothesis.

(iv) If $x \in X(\mathfrak{B})$, then

$I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$

$\Leftrightarrow I([x]W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))[\underset{\sim}{D}, \xi]$ by the definition of W

$\Leftrightarrow I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x))[\underset{\sim}{D}, \xi]$ by the definitions of π and I, and this holds by induction hypothesis, and the induction is therefore complete. ∎

So, we have shown that if we execute $\mathfrak{A}$ and so generate a word $w \in |\alpha_{\mathfrak{A}}|$, then the initial condition of w must hold. But, what of the converse? Suppose we are given $\underset{\sim}{D}$, ξ and $w \in |\alpha_{\mathfrak{A}}|$, and we find that $I(w)[\underset{\sim}{D}, \xi]$. Can we infer that $w = W(\mathfrak{A}, \underset{\sim}{D}, \xi, x)$, where $[x] = b_i$ and $Y_b(w) = i$? The answer is yes, and this is the content of the following

Theorem 23: *For any E-program $\mathfrak{A} = \langle X, \Gamma, \chi \rangle$, computing structure $\underset{\sim}{D}$, state $\xi : \omega \to \underset{\sim}{D}_0$ and word $w \in |\alpha_{\mathfrak{A}}|$, if $I(w)[\underset{\sim}{D}, \xi]$, then $w = W(\mathfrak{A}, \underset{\sim}{D}, \xi, x)$ where $[x] = b_i$, $i = Y_b(w)$.*

Proof: Roughly speaking, we show that $I(w)[D, \xi]$ implies that all the qffs in w, or alternatively in the path defined by w, have truth-values such that execution, and hence W too, will follow that path. In the proof, we imagine that the acceptor function T^* is used to step letter by letter through w, causing the automaton $M_{\mathfrak{A}}$ to undergo transitions from state to state. At the same time, W is used to step through the E-program $\mathfrak{A}$, from node to node, generating a word letter by letter. At each step in the process, we verify that the node of $\mathfrak{A}$ corresponds to the state of $M_{\mathfrak{A}}$, and that the letters of W and w are identical. We use an inductive proof, but of a strange variety. Induction hypotheses are made about the situation on either side of the current position of the word in question, so that a primitive basis case occurs at each end of the word, one for each induction.

(i) Consider the initial case. Suppose $w = b_i u$. Then,

$T^*(b, w)$

$= T^*(T(b, b_i), u)$ by the definition of T^*

$= T^*(\Gamma x, u)$ since $[x] = b_i$ and by the definition of T.

and,

$W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$

$= [x]W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x)$ by the definition of W

$= b_i W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x)$

Thus, the next state and node coincide, i.e., are Γx .

Furthermore, note that $I(w)[\underset{\sim}{D}, \xi] \Rightarrow I(u)[\underset{\sim}{D}, \xi]$, by the definition of I . Then by induction hypothesis, $I(u)[\underset{\sim}{D}, \xi] \Rightarrow u = W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x)$, so that $w = b_i u = b_i W(\mathfrak{M}, \underset{\sim}{D}, \xi, \Gamma x) = W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$, as required.

(ii) Consider an intermediate case. By induction hypothesis assume that $M_{\mathfrak{M}}$ is in state $s \in X(\mathfrak{A})$, and that W is at node s with state η . Also assume that $v = \sigma u$, $\sigma \in \Sigma_{\mathfrak{M}}$, is the word remaining, and that $I(v)[\underset{\sim}{D}, \eta]$. Then,

$T^*(s, v)$

$= T^*(T(s, \sigma), u)$ by the definition of T^*

If $w \in |\alpha_{\mathfrak{M}}|$, then $T^*(s, v) = e$, and therefore it must be the case that $\sigma = [s]$, otherwise transition to the dead state d would occur, and e would not be reached. So,

$T^*(s, v)$

$= T^*(T(s,[s]), u)$

$= T^*(\Gamma s, u)$ by the definition of T^* .

As well,

$W(\mathfrak{M}, \underset{\sim}{D}, \eta, s)$

$= [s]W(\mathfrak{M}, \underset{\sim}{D}, [s][\underset{\sim}{D}, \eta], \Gamma s)$ by the definition of W .

Thus, the next state and node coincide, i.e., are Γs .

Furthermore, note that $I(v)[\underline{D}, \eta] \Rightarrow I(u)[\underline{D}, [s][\underline{D}, \eta]]$, as indicated in the proof of Theorem 22. Then, by induction hypothesis, $I(u)[\underline{D}, [s][\underline{D}, \eta]] \Rightarrow u = W(\mathfrak{M}, \underline{D}, [s][\underline{D}, \eta], \Gamma s)$, so, that $v = [s]u = [s]W(\mathfrak{M}, \underline{D}, [s][\underline{D}, \eta], \Gamma s) = W(\mathfrak{M}, \underline{D}, \eta, s)$, as required.

(iii) Consider the other intermediate case. By induction hypothesis, assume that $M_{\mathfrak{M}}$ is in state $s \in X(\underline{n})$, and that W is at node s with state η. Also assume that $v = \sigma u$, $\sigma \in \Sigma_{\mathfrak{M}}$, is the word remaining, and that $I(v)[D, \eta]$. Then,

$T^*(s, v)$

$= T^*(T(s, \sigma), u)$ by the definition of T^*.

If $w \in |\alpha_{\mathfrak{M}}|$, then $T^*(s, v) = e$, and therefore it must be the case that $\sigma \in \{[s], \sim[s]\}$, otherwise transition to the dead state d would occur, and e would not be reached. So,

$T^*(s, v)$

$= T^*(T(s, \sigma), u)$ where $\sigma \in \{[s], \sim[s]\}$

$= T^*(t, u)$ where $\Gamma s = \langle y, z \rangle$ and $t \in \{y, z\}$.

As well,

$W(\mathfrak{M}, \underline{D}, \eta, s)$

$= [s]W(\mathfrak{M}, \underline{D}, \eta, y)$ if $[s][\underline{D}, \eta]$ or

$\sim[s]W(\mathfrak{M}, \underline{D}, \eta, z)$ if $\sim[s][\underline{D}, \eta]$, by the definition of W.

Now, suppose that, in fact, $\sigma = [s]$. Then

$I(v)[\underline{D}, \eta]$

$\Rightarrow I(\sigma u)[\underline{D}, \eta]$

$\Rightarrow I([s]u)[\underline{D}, \eta]$

$\Rightarrow [s][\underline{D}, \eta] \wedge I(u)[\underline{D}, \eta]$ as indicated in the proof of Theorem 22.

Then, $W(\mathfrak{M}, \underset{\sim}{D}, \eta, s) = [s]W(\mathfrak{M}, \underset{\sim}{D}, \eta, y)$. If, in fact, $\tau = \sim[s]$, then we get that $W(\mathfrak{M}, \underset{\sim}{D}, \eta, s) = \sim[s]W(\mathfrak{M}, \underset{\sim}{D}, \eta, z)$. Thus, the next state and node coincide, i.e., are y or z , as the case may be, where $\Gamma s = \langle y, z \rangle$.

Furthermore, note that $I(v)[\underset{\sim}{D}, \eta] \Rightarrow I(u)[\underset{\sim}{D}, \eta]$, as indicated above. Then, by induction hypothesis, $I(u)[\underset{\sim}{D}, \eta] \Rightarrow u = W(\mathfrak{M}, \underset{\sim}{D}, \eta, t)$, $t \in \{y, z\}$, so that $v = \sigma u = \sigma W(\mathfrak{M}, \underset{\sim}{D}, \sigma[\underset{\sim}{D}, \eta], t) = W(\mathfrak{M}, \underset{\sim}{D}, \eta, s)$, as required.

(iv) Consider the final case. By induction hypothesis, assume that $M_{\mathfrak{M}}$ is in state $s \in X(\mathfrak{E})$, and that W is at node s with state η . Also assume that $v = \sigma u$, $\sigma \in \Sigma_{\mathfrak{M}}$, is the word remaining, and that $I(v)[\underset{\sim}{D}, \eta]$. Then,

$T^*(s, v)$

$= T^*(T(s, \sigma), u)$ by the definition of T^*

If $v \in |\alpha_{\mathfrak{M}}|$, then $T^*(s, v) = e$, and therefore it must be the case that $\sigma = [s]$, otherwise transition to the dead state d would occur, and e would not be reached. Of course, then $u = \Lambda$. So,

$T^*(s, v)$

$= T^*(T(s, \sigma), \Lambda)$ by the definition of T^*

$= T^*(e, \Lambda)$ by the definition of T

$= e$ by the definition of T^*

As well, $W(\mathfrak{M}, \underset{\sim}{D}, \eta, s) = [s]$, by the definition of W . Thus, $v = \sigma u = [s] = W(\mathfrak{M}, \underset{\sim}{D}, \eta, s)$, as required. This completes the induction. ∎

Together, Theorems 19, 20, 21 and 23 give us the following useful

<u>Theorem 24:</u> <u>For any E-program $\mathfrak{M} = \langle X, \Gamma, \mathcal{X} \rangle$, computing structure $\underset{\sim}{D}$, state $\xi : \omega \to D_\circ$ and word $w \in |\alpha_{\mathfrak{M}}|$,</u>

<u>$I(w)[\underset{\sim}{D}, \xi] = \mathfrak{M}[\underset{\sim}{D}, \langle \xi, Y_b(w\triangleright) \rangle] = \langle A(w)[\underset{\sim}{D}, \xi], Y_e(w\triangleright) \rangle$</u>

<u>Proof:</u> Consider the $\Rightarrow$ case first.

$\mathfrak{M}[\underset{\sim}{D}, \langle \xi, Y_b(w)\rangle]$

$\simeq E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ by the definition of semantics for E-programs where

$[x] = b_i$ and $Y_b(w) = i$.

$\simeq E^*(\underset{\sim}{D}, \xi, W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))$ by Theorem 20

$= \langle A(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[D, \xi], Y_e(w)\rangle$ since by Theorem 23,

$I(w)[\underset{\sim}{D}, \xi] \Rightarrow w = W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$, i,e, $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite. Then, from Theorem 19 we get that $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \in |\alpha_{\mathfrak{M}}|$, and this allows application of Theorem 21.

$= \langle A(w)[\underset{\sim}{D}, \xi], Y_e(w)\rangle$ once again, by Theorem 23.

Now, consider the $\Leftarrow$ case. We know

$\mathfrak{M}[\underset{\sim}{D}, \langle \xi, Y_b(w)\rangle]$

$= \langle A(w)[\underset{\sim}{D}, \xi], Y_e(w)\rangle$ from the hypotheses of the theorem.

Thus, $E(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ must be determinate, and so therefore is $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$. In this case, from Theorem 19 we get again that $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x) \in |\alpha_{\mathfrak{M}}|$. So applying Theorems 20 and 21 as in the first case,

$\mathfrak{M}[\underset{\sim}{D}, \langle \xi, Y_b(w)\rangle]$

$= \langle A(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi], Y_e(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))\rangle$

whence, $w = W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$. But since $W(\mathfrak{M}, \underset{\sim}{D}, \xi, x)$ is finite, then $I(W(\mathfrak{M}, \underset{\sim}{D}, \xi, x))[\underset{\sim}{D}, \xi]$, by Theorem 22, i.e., $I(w)[\underset{\sim}{D}, \xi]$. ∎

Using Theorem 24 as a starting point, we can recast the definition of strong equivalence in a form which naturally sheds light on the decidable sub-cases.

To start, let us define the binary relation $\leftrightarrow$ between words of $|\alpha_{\mathfrak{M}}|$, such that for any $u, v \in |\alpha_{\mathfrak{M}}|$, we say u is <u>similar</u> to v and write $u \leftrightarrow v$, iff

(i) $Y_b(u) = Y_b(v)$

(ii) $Y_e(u) = Y_e(v)$

(iii) for all $\underset{\sim}{D}$, for all $\xi : \omega \to \underset{\sim}{D}_o$, $A(u)[\underset{\sim}{D}, \xi] = A(v)[\underset{\sim}{D}, \xi]$, i.e., A(u) and A(v) are <u>strongly equivalent</u>. This we write as $A(u) \simeq A(v)$.

Now, consider a partition $\mathcal{C}_\alpha$ of $|\alpha_\alpha|$ into similarity equivalence classes such that

(i) $\cup_{Z \in \mathcal{C}_\alpha} Z = |\alpha_\alpha|$

(ii) $\cap_{Z \in \mathcal{C}_\alpha} Z = \emptyset$,

and such that for any $u \in |\alpha_\alpha|$, the equivalence class of u with respect to $\rightharpoonup$ is $\{v \in |\alpha_\alpha| : u \rightharpoonup v\}$.

Let us extend the notion of similarity to similarity equivalence classes themselves. If $U \in \mathcal{C}_\alpha$ and $V \in \mathcal{C}_\beta$ are similarity equivalence classes, then we say U is similar to V, and write $U \rightharpoonup V$, iff for some $u \in U$ and $v \in V$, $u \rightharpoonup v$.

With each $U \in \mathcal{C}_\alpha$, we associate a <u>joint initial condition</u> J(U) (no confusion will result from the duplicate "initial condition" nomenclature), where

$J(U) = \vee\{p \in \mathcal{Q} : p = I(u)$ for some $u \in U\}$.

Here we write $\vee S$, where S is a set of formal objects, for the disjunction formed with "$\vee$" of the objects in S in any order. If S is infinite, then $\vee S$ is an infinite disjunction. Intuitively, J(U) is the initial condition for any of the words in U, all of which have strongly equivalent operations

Three further notational matters: we use $\wedge S$ to denote the conjunction formed with "$\wedge$" of the objects in S; we denote $(p \supset q) \wedge (q \supset p)$ as $p \equiv q$; and we extend logical validity, denoted $\models^*$, to infinitely long qffs in the natural way (cf. Karp [22] for a detailed treatment).

With the notions of similarity and joint initial condition defined, we can give the following

Theorem 25: For any two E-programs of the same type,

$$\vdash_{\mathfrak{A}} \cong \mathfrak{B} \Leftrightarrow \vdash^{*} \wedge \{J(U) \equiv J(V) : V \in \mathcal{C}_{\mathfrak{A}} \ \&\ V \in \mathcal{C}_{\mathfrak{B}} \ \&\ U \leftrightarrow V\} \wedge$$
$$\wedge \{\sim J(U) : U \in \mathcal{C}_{\mathfrak{A}} \ \&\ \text{for all } V \in \mathcal{C}_{\mathfrak{B}},\ U \not\leftrightarrow V\} \wedge$$
$$\wedge \{\sim J(V) : V \in \mathcal{C}_{\mathfrak{B}} \ \&\ \text{for all } U \in \mathcal{C}_{\mathfrak{A}},\ U \not\leftrightarrow V\}$$

In very rough terms, Theorem 25 states that $\mathfrak{A}$ and $\mathfrak{B}$ are strongly equivalent iff they have the same class of potential outputs, and the conditions for $\mathfrak{A}$ and $\mathfrak{B}$ to produce each such output are logically equivalent.

Proof: Consider the $\Leftarrow$ case first. Consider any computing structure $\underset{\sim}{D}$, state $\xi : \omega \to \underset{\sim}{D}_0$, and $i < m$, where $\mathfrak{A}$ and $\mathfrak{B}$ are both of type $\langle m, n \rangle$, say. We write A for $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ and B for $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ in what follows. We will show that the logical validity of the qff, Q say, in the statement of this theorem implies that $A \cong B$. There are five cases to consider.

(i) A and B are both indeterminate. Thus, $A \cong B$.

(ii) A and B are both determinate and $A = B$. Thus, $A \cong B$.

(iii) A is determinate and produces a word u in $U \in \mathcal{C}_{\mathfrak{A}}$, but B is indeterminate. From Theorem 24 we have $I(u)[\underset{\sim}{D}, \xi]$, so that by the definition of joint initial condition, $J(U)[\underset{\sim}{D}, \xi]$. But then, notice that for no $V \in \mathcal{C}_{\mathfrak{B}}$, such that $U \leftrightarrow V$, do we have $J(V)[\underset{\sim}{D}, \xi]$. This is because if there were such a V, we would have $J(U) \equiv J(V)[\underset{\sim}{D}, \xi]$, and further, since $J(U)[\underset{\sim}{D}, \xi]$, therefore $J(V)[\underset{\sim}{D}, \xi]$. But then by Theorem 24, B must be determinate and produce

some $v \in V$. Since this contradicts the hypothesis of this case, there is, therefore, no such $V \in \mathcal{C}_{\mathfrak{B}}$, such that $U \leftrightarrow V$. But, then the second conjunct of Q gives that $\sim J(U)[\underset{\sim}{D}, \xi]$, which also contradicts the hypothesis of this case. Therefore, this case cannot arise.

(iv) B is determinate and produces a word v in $V \in \mathcal{C}_{\mathfrak{B}}$, but A is indeterminate. The argument proceeds here as in case (ii) above, except that we make use of the third conjunct of Q to show that this case cannot arise.

(v) A and B are both determinate and produce words u and v in $U \in \mathcal{C}_{\mathfrak{A}}$ and $V \in \mathcal{C}_{\mathfrak{B}}$, respectively, but $A \neq B$. From Theorem 24 we have $I(u)[\underset{\sim}{D}, \xi]$ and $I(v)[\underset{\sim}{D}, \xi]$, so that by the definition of joint initial condition, $J(U)[\underset{\sim}{D}, \xi]$ and $J(V)[\underset{\sim}{D}, \xi]$. However, since $A \neq B$, therefore $U \not\leftrightarrow V$. Suppose that for some $W \in \mathcal{C}_{\mathfrak{B}}$, $U \leftrightarrow W$. Then, the first conjunct of Q gives that $J(U) \equiv J(W)[\underset{\sim}{D}, \xi]$, and since $J(U)[\underset{\sim}{D}, \xi]$, therefore $J(W)[\underset{\sim}{D}, \xi]$. Now, since W and V are distinct (they must be since $U \not\leftrightarrow V$ and $U \leftrightarrow W$), and since not both $J(W)[\underset{\sim}{D}, \xi]$ and $J(V)[\underset{\sim}{D}, \xi]$ (otherwise, by Theorem 24, one execution would give two different outputs), and since $J(W)[\underset{\sim}{D}, \xi]$ therefore not $J(V)[\underset{\sim}{D}, \xi]$. But, this is a contradiction so that for no $W \in \mathcal{C}_{\mathfrak{B}}$ do we have $U \leftrightarrow W$. But, then, the second conjunct of Q gives that $\sim J(U)[\underset{\sim}{D}, \xi]$ which is also a contradiction, so that therefore this case does not arise either.

Since only cases (i) and (ii) can arise, and since they give $A \equiv B$ for any such $\underset{\sim}{D}$, ξ and i, we therefore have $\vdash \mathfrak{A} \simeq \mathfrak{B}$.

Now, consider the $\Rightarrow$ case. We will assume the conjunction Q is not logically valid, and then show that $\mathfrak{A}$ and $\mathfrak{B}$ are therefore not

strongly equivalent. Since the conjunction Q is not logically valid, one (at least) of the conjuncts is not logically valid. Suppose that $\sim J(U)$, where $U \in \mathcal{C}_{\mathfrak{A}}$ and for all $V \in \mathcal{C}_{\mathfrak{B}}$, $U \not\leftrightarrow V$, is not logically valid. Then, for some computing structure $\underset{\sim}{D}$, state ξ, not $\sim J(U)[\underset{\sim}{D}, \xi]$, i.e., $J(U)[\underset{\sim}{D}, \xi]$. Then by the definition of joint initial condition, for some $u \in U$, $I(u)[\underset{\sim}{D}, \xi]$, so that by Theorem 24, $\mathfrak{A}[D, \langle \xi, Y_b(u) \rangle] = \langle A(u)[\underset{\sim}{D}, \xi], Y_e(u) \rangle$. Then, since for all $V \in \mathcal{C}_{\mathfrak{B}}$, $U \not\leftrightarrow V$, therefore $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, Y_b(u) \rangle]$ either is indeterminate, is determinate with execution not halting at e_j where $j = Y_e(u)$, or is determinate with execution halting at e_j, but having executed a word in some other similarity equivalence class different from U. If either of the first two cases occur, then we are done, since then clearly not $\vdash_{\mathfrak{A}} \simeq \mathfrak{B}$. The last case must be considered at length.

However, let us first consider the second alternative for making Q not logically valid. Suppose that $J(U) \equiv J(V)$, where $U \in \mathcal{C}_{\mathfrak{A}}$, $V \in \mathcal{C}_{\mathfrak{B}}$ and $U \not\leftrightarrow V$, is not logically valid. Thus, for some computing structure $\underset{\sim}{D}$ and state ξ, not $J(U) \equiv J(V)[\underset{\sim}{D}, \xi]$, i.e., for example, $J(U)[\underset{\sim}{D}, \xi]$ but not $J(V)[\underset{\sim}{D}, \xi]$. But, since for all $W \in \mathcal{C}_{\mathfrak{B}}$ different from V we have $V \not\leftrightarrow W$, this case is therefore precisely like that considered above.

So, we are concerned with the case where $I(u)[\underset{\sim}{D}, \xi]$, $u \in U$, $U \in \mathcal{C}_{\mathfrak{A}}$ and $I(w)[\underset{\sim}{D}, \xi]$, $w \in W$, $W \in \mathcal{C}_{\mathfrak{B}}$. As well, $U \not\leftrightarrow W$, but $Y_b(u) = Y_b(w)$ and $Y_e(u) = Y_e(w)$. It may be, for this $\underset{\sim}{D}$ and ξ, that $A(u)[\underset{\sim}{D}, \xi] \neq A(w)[\underset{\sim}{D}, \xi]$, so that not $\vdash_{\mathfrak{A}} \simeq \mathfrak{B}$, in which case we are done. But, suppose $A(u)[\underset{\sim}{D}, \xi] = A(w)[\underset{\sim}{D}, \xi]$. We want now to show the existence of a special computing structure $\underset{\sim}{D}^*$ and special initial state $\xi^* : \omega \to \underset{\sim}{D}_o^*$, such that $I(u)[\underset{\sim}{D}^*, \xi^*)$ and $I(w)[\underset{\sim}{D}^*, \xi^*]$, but where $A(u)[\underset{\sim}{D}^*, \xi^*] \neq A(w)[\underset{\sim}{D}^*, \xi^*]$, so that not $\vdash_{\mathfrak{A}} \simeq \mathfrak{B}$.

Consider the computing structure $\underset{\sim}{D}^*$, with signature

$s = \langle \langle n_0, \ldots, n_{k-1} \rangle, \langle m_0, \ldots, m_{\ell-1} \rangle, p \rangle$, with domain

$D^* = (\{v_0, v_1, \ldots, v_{t-1}\} \cup \{k_0, \ldots, k_{p-1}\} \cup \{f_0, \ldots, f_{\ell-1}\} \cup \{"(", ")", ","\})^*$

where $\mathfrak{A}, \mathfrak{B} \in L_s$; $\{v_0, v_1, \ldots, v_{t-1}\}$ contains all the variables occurring in $\mathfrak{A}$ and $\mathfrak{B}$; and $(\ldots)^*$ once again denotes the free semi-group with identity generated by $(\ldots)$. Here, then, the domain consists of finite strings made up of some of the symbols that appear in $\mathfrak{A}$ and $\mathfrak{B}$.

Now, we define the other constituents of

$\underset{\sim}{D}^* = \langle D^*, R_0, \ldots, R_{k-1}, F_0, \ldots, F_{\ell-1}, a_0, \ldots, a_{p-1} \rangle$

The designated individuals $a_0, \ldots, a_{p-1}$ are just the symbols $k_0, \ldots, k_{p-1} \in D^*$. The functions F_i, $i < \ell$, are defined by $F_i(x_0, \ldots, x_{m_i-1}) = f_i(x_0, \ldots, x_{m_i-1})$, where $x_0, \ldots, x_{m_i-1} \in D^*$. Here, "f_i", "(", ")", "," , and the $x_0, \ldots, x_{m_i-1}$ are juxtaposed as shown to give $f_i(x_0, \ldots, x_{m_i-1}) \in D^*$. We define R_i, $i < k$, by referring back to $\underset{\sim}{D}$ and ξ , namely, $R_i(x_0, \ldots, x_{n_i-1}) \Leftrightarrow r(x_0, \ldots, x_{n_i-1})[\underset{\sim}{D}, \xi]$.

Then let the input state $\xi^* : \omega \to D^*$ be defined so that $c(i, \xi^*) = v_i$, where of course $v_i \in D^*$.

This exotic computing structure and initial state permit us to show that not $\vdash \mathfrak{A} \supset \mathfrak{B}$. First, notice that $I(u)[\underset{\sim}{D}, \xi] \Rightarrow I(u)[\underset{\sim}{D}^*, \xi^*]$ and $I(w)[\underset{\sim}{D}, \xi] \Rightarrow I(w)[\underset{\sim}{D}^*, \xi^*]$, since this relationship holds for qffs in general. (For example, suppose $r_2(v_6)[\underset{\sim}{D}, \xi]$; then R_2 is defined so that $R_2(v_6)$. But then $r_2(v_6)[\underset{\sim}{D}^*, \xi^*] \Leftrightarrow R_2(c(6, \xi^*)) \Leftrightarrow R_2(v_6)$, which holds. Thus, $r_2(v_6)[\underset{\sim}{D}, \xi] \Rightarrow r_2(v_6)[\underset{\sim}{D}^*, \xi^*]$.) Therefore, we can examine $A(u)[\underset{\sim}{D}^*, \xi^*]$ and $A(w)[\underset{\sim}{D}^*, \xi^*]$.

Because of the way the F_i and R_i are defined, these executions follow exactly the computations given by $A(u)[\underset{\sim}{D}, \xi]$ and $A(w)[\underset{\sim}{D}, \xi]$, except now the computation is being carried out in symbolic form, the current state reflecting all past computations. Since $U \not\equiv W$, then $u \not\equiv w$, and since $Y_b(u) = Y_b(w)$ and $Y_e(u) = Y_e(w)$, we must have $A(u) \neq A(w)$. Now if $A(u)[\underset{\sim}{D}^*, \xi^*] = A(w)[\underset{\sim}{D}^*, \xi^*]$ then certainly $A(u) \simeq A(w)$, so that $A(u)[\underset{\sim}{D}^*, \xi^*] \neq A(w)[\underset{\sim}{D}^*, \xi^*]$ must be the case. Thus, $\mathfrak{M}[\underset{\sim}{D}^*, \langle \xi^*, Y_b(u) \rangle] \neq \mathfrak{M}[\underset{\sim}{D}^*, \langle \xi^*, Y_b(w) \rangle]$, and hence not $\mathfrak{M} \simeq \mathfrak{B}$. ∎

Remarks:

(i) The ideas of initial condition, $I(w)$, and operation, $A(w)$, are closely related to McCarthy's method [29] for prescribing conditions α_{ij} for entering $\mathfrak{M}$ at b_i and leaving at e_j, and operators s_{ij} telling what function is computed entering at b_i and leaving at e_j. The difference lies in the fact that McCarthy concerns himself with schemes of recursive equations for defining the α_{ij}, the solution to which are not considered. Essentially, the outcome here is that for simple flowcharts, as characterize algorithms here, regular expressions provide a convenient way of expressing the solution of such schemes of equations. Ito [17], considers these matters in some detail.

(ii) If finite sub-sets of $|\alpha_{\mathfrak{M}}|$ and $|\alpha_{\mathfrak{B}}|$ can be isolated that we know contain all words produced by halting executions of $\mathfrak{M}$ and $\mathfrak{B}$ respectively, then the partitions $\zeta_{\mathfrak{M}}$ and $\zeta_{\mathfrak{B}}$ are finite, as is, therefore, the qff in the statement of Theorem 25. Then, since the strong equivalence of operations is decidable (Theorem 16), the strong equivalence of E-programs like $\mathfrak{M}$ and $\mathfrak{B}$ is therefore decidable under those circumstance for which the logical validity of qffs is decidable.

K-events and the Decidability of K-equivalence

At a pragmatic level, the decidability of certain exotic and computationally uninteresting sub-cases of the strong equivalence problem does little to help us analyze the more complex, and hence more useful, general case. Our interest, then, is in finding a sufficiently rich subset of $\mathcal{F}m_s$, for arbitrary signatures s, for which $\mathcal{A}_s$ is complete. In this chapter, we attack this problem indirectly by defining the notion of K-equivalence between E-programs, showing that K-equivalence is decidable and that $\mathcal{A}_s$ implies a second inferential system $\mathcal{J}$ adequate for deriving K-equivalence. The further result that the K-equivalence of two E-programs implies their strong equivalence, means that a handle on strong equivalence for a large and interesting subset of $\mathcal{F}m_s$ is therefore available.

Before we proceed with the definitions of K-equivalence and K-events, and to foreshadow what is to come, let us use the results of the preceding section concerning initial conditions to derive the following useful

Theorem 26: For any E-programs $\mathfrak{A}$ and $\mathfrak{B}$ of the same type,

$$|\alpha_{\mathfrak{A}}| = |\alpha_{\mathfrak{B}}| \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}$$

Proof: Since $|\alpha_{\mathfrak{A}}| = |\alpha_{\mathfrak{B}}|$, then the partitions $\mathcal{C}_{\mathfrak{A}}$ and $\mathcal{C}_{\mathfrak{B}}$ are identical, i.e., $\mathcal{C}_{\mathfrak{A}} = \mathcal{C}_{\mathfrak{B}}$. Thus in

$$\wedge\{J(U) \equiv J(V) : U \in \mathcal{C}_{\mathfrak{A}} \ \& \ V \in \mathcal{C}_{\mathfrak{B}} \ \& \ U \leftrightsquigarrow V\}$$

$U \leftrightsquigarrow V$ and $\mathcal{C}_{\mathfrak{A}} = \mathcal{C}_{\mathfrak{B}}$ imply $U = V$. Thus, $\vdash J(U) \equiv J(V)$ for all such conjuncts, making the entire conjunction logically valid. Hence, by Theorem 25, $\vdash \mathfrak{A} \simeq \mathfrak{B}$. ∎

Now, the equality of regular sets is decidable (cf. Salomaa [58]) so that Theorem 26 gives us a tool, however meager, for investigating strong equivalence.

Thus, given $\mathfrak{A}$, $\mathfrak{B}$, and the question whether $\mathfrak{A}$ and $\mathfrak{B}$ are strongly equivalent, we can generate $\alpha_{\mathfrak{A}}$ and $\alpha_{\mathfrak{B}}$, by effective process, and then test for $|\alpha_{\mathfrak{A}}| = |\alpha_{\mathfrak{B}}|$, also an effective process. If $|\alpha_{\mathfrak{A}}| = |\alpha_{\mathfrak{B}}|$ then $\models \mathfrak{A} \simeq \mathfrak{B}$, but if $|\alpha_{\mathfrak{A}}| \neq |\alpha_{\mathfrak{B}}|$, we can draw no conclusions one way or the other.

Consider the wff $\mathfrak{A} \simeq \mathfrak{B}$ of Figure 46, which is generally valid since $|\alpha_{\mathfrak{A}}| = |\alpha_{\mathfrak{B}}|$. We see that the test for strong equivalence can detect this property in a fairly large subset of $\mathcal{F}_S$. However, this method fails on even so simple a wff as $A\supset$ (cf. Figure 21), where for the method to work, we would have to show $|p \vee \bar{p}| = |1|$ which is clearly not true. It is just this sort of problem that the introduction of τ-events and K-equivalence will alleviate.

Syntax and Semantics of K-expressions

K-expressions are defined over an alphabet $\Sigma = \mathcal{A}_m \cup \{0, 1\} \cup \mathcal{G}$ where $\mathcal{A}_m$ and $\mathcal{G}$ are disjoint finite non-empty sets of atomic formulas and operators respectively. For this discussion, we will write $\mathcal{A}_m = \{p_0, p_1, \ldots, p_{m-1}\}$, $m < \omega^-$, and $\mathcal{G} = \{g_0, g_1, \ldots, g_{n-1}\}$, $n < \omega^-$, where the p_i, $i < m$, and g_i, $i < n$, merely stand for the actual letters of $\mathcal{A}_m$ and $\mathcal{G}$.

First, we define the set $\mathcal{P}$ of propositions

(i) $\mathcal{A}_m \cup \{0, 1\} \subseteq \mathcal{P}$

(ii) If $p, q \in \mathcal{P}$, then $(\sim p) \in \mathcal{P}$ and $(p \supset q) \in \mathcal{P}$.

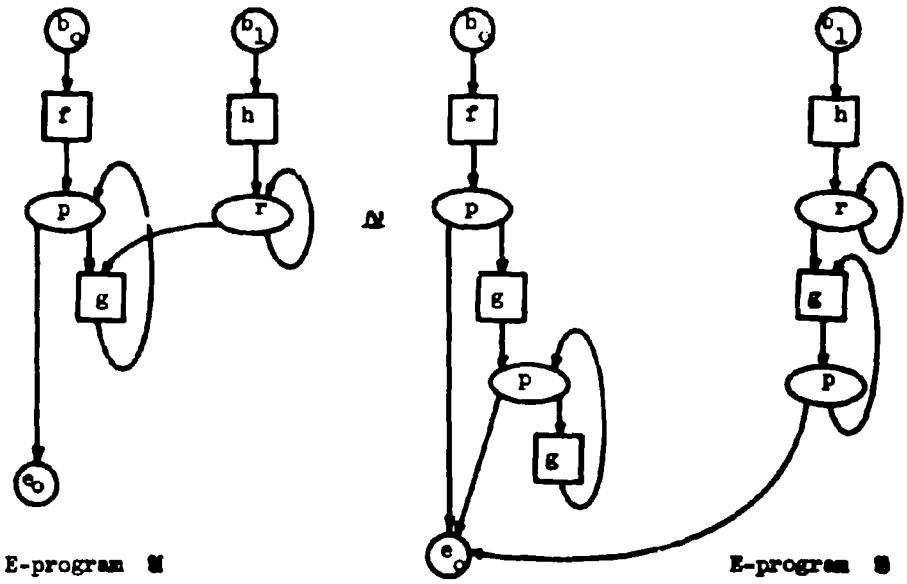
(iii) Extremal clause.

Then, we define the set $\mathcal{K}$ of K-expressions.

(i) $\mathcal{P} \cup \mathcal{G} \subseteq \mathcal{K}$

(ii) If $\alpha, \beta \in \mathcal{K}$, then $(\alpha \cdot \beta)$, $(\alpha \vee \beta)$, $\alpha^* \in \mathcal{K}$

(iii) Extremal clause.



$\alpha_{\mathfrak{M}} = b_0 f(\bar{p}q)^* p e_0 \vee b_1 h\bar{r}^* r g(\bar{p}q)^* p e_0$

$\alpha_{\mathfrak{B}} = b_0 f(p \vee \bar{p}q(\bar{p}q)^* p) e_0 \vee b_1 h\bar{r}^* r(g\bar{p})^* g p e_0$

Figure 46

An example of $|\alpha_{\mathfrak{M}}| = |\alpha_{\mathfrak{B}}| \Rightarrow \vdash \mathfrak{M} \simeq \mathfrak{B}$. Of course, the $\alpha_{\mathfrak{M}}$ and $\alpha_{\mathfrak{B}}$ given above are not the only regular expressions generable from $\mathfrak{M}$ and $\mathfrak{B}$. Here, p, r are qffs, and f, g, h are assignment schemata.

We use the same conventions as for regular expressions when dropping parentheses and the "." from $(\alpha \cdot \beta)$.

The semantics of a K-expression α is a set called a K-event denoted by $\|\alpha\|$. In the definition of $\|\alpha\|$, we make use of the set $\mathcal{T}$ called truth, defined as

$$\mathcal{T} = \{s_0 s_1 \ldots s_{m-1} : s_i = p_i \text{ or } s_i = (\sim p_i) , i < m\}$$

which is just the set of disjuncts of the full disjunctive normal form for a tautology in $\mathcal{P}$. Then,

$$\|p_i\| = \{s_0 s_1 \ldots s_{m-1} \in \mathcal{T} : s_i = p_i\} \qquad p_i \in \mathcal{A}tm$$

$$\|g_i\| = \{pg_i q : p,q \in \mathcal{T}\} \qquad g_i \in \mathcal{G}$$

$$\|1\| = \mathcal{T}$$

$$\|0\| = \emptyset$$

$$\|\sim p\| = \mathcal{T} - \|p\| \qquad p \in \mathcal{P}$$

$$\|p \supset q\| = \|\sim p\| \cup \|q\| \qquad p,q \in \mathcal{P}$$

$$\left.\begin{aligned} \|\alpha \vee \beta\| &= \|\alpha\| \cup \|\beta\| \\ \|\alpha^*\| &= \|1\| \cup \|\alpha\| \cup \|\alpha\alpha\| \cup \|\alpha\alpha\alpha\| \cup \ldots \\ \|\alpha\beta\| &= \{xpy : xp \in \|\alpha\| \ \&\ py \in \|\beta\| \ \&\ p \in \mathcal{T}\} \end{aligned}\right\} \quad \alpha,\beta \in \mathcal{K}$$

Notice that for any E-program $\mathfrak{A}$, we can take $\mathcal{A}tm_{\mathfrak{A}}$ to be the set of all distinct qffs of the form $r_j(\tau_0, \ldots, \tau_{n_j-1})$ or $(\tau = \sigma)$ occurring in $\mathfrak{A}$ (or simply the set $\{(v_0 = v_0)\}$ if there are no qffs occurring in $\mathfrak{A}$) , and $\mathcal{G}_{\mathfrak{A}}$ to be the set of all initiators, terminators and distinct assignment schemata occurring in $\mathfrak{A}$. Then evidently, $\alpha_{\mathfrak{A}}$, the regular expression over $\Sigma_{\mathfrak{A}}$ corresponding to $\mathfrak{A}$, is also a K-expression over $\mathcal{A}tm_{\mathfrak{A}}$ and $\mathcal{G}_{\mathfrak{A}}$. In general, we concentrate our attention on K-expressions that come from E-programs even though some of the theorems we prove in the sequel hold for all K-expressions in general.

Let us consider some examples of E-programs and their respective K-expressions and corresponding K-events. For E-program $\mathfrak{A}$ of Figure 47, we can write

$$\alpha_{\mathfrak{A}} = b_o v_6 := k_o(\sim r_2(v_6)v_6 := k_1(v_6))^* r_2(v_6)(r_9(v_6)e_o \vee \sim r_9(v_6)v_6 := k_o e_1)$$

or, in a more compact abbreviated form

$$\alpha_{\mathfrak{A}} = b_o k(\bar{r}\bar{p})^* r(pe_o \vee \bar{p}ke_1)$$

where k stands for $v_6 := k_o$, and so on. Then,

$$|\alpha_{\mathfrak{A}}| = \{b_o krpe_o,\ b_o kr\bar{p}ke_1,\ b_o k\bar{r}frpe_o,\ b_o k\bar{r}fr\bar{p}ke_1, \ldots\}$$

and since $Atm_{\mathfrak{A}} = \{r_2(v_6),\ r_9(v_6)\} = \{r,\ p\}$, then

$$\mathcal{T} = \{rp,\ \bar{r}p,\ r\bar{p},\ \bar{r}\bar{p}\}$$

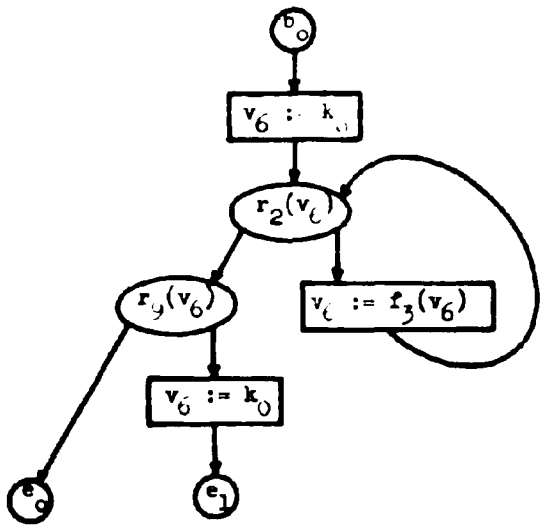
and

$$\|\alpha_{\mathfrak{A}}\| = \{rpb_o rpkrpe_o rp,\ rpb_o rpkrpe_o \bar{r}p,\ rpb_o rpkrpe_o r\bar{p},$$
$$rpb_o rpkrpe_o \bar{r}\bar{p},\ rpb_o \bar{r}pkrpe_o rp,\ rpb_o \bar{r}pkrpe_o \bar{r}p, \ldots,$$
$$\bar{r}\bar{p}b_o \bar{r}\bar{p}krpe_o \bar{r}\bar{p}, \ldots\}\ .$$

Each word of $\|\alpha_{\mathfrak{A}}\|$ is an alternating sequence of words of truth and operators, the first and last operators being an initiator and a terminator respectively. Each word of truth, in some sense, depicts the state of affairs at that point in the execution of $\mathfrak{A}$. Thus, as $\mathfrak{A}$ is executed, before and after each operator (i.e., assignment schema) is encountered, the atomic formulas in $Atm_{\mathfrak{A}}$ have certain truth values with respect to the current state, and these are mirrored in the words of truth preceding and following the operator in $w \in \|\alpha_{\mathfrak{A}}\|$. Consider, for example,

$$\bar{r}pb_o \bar{r}pkrpe_o rp \in \|\alpha_{\mathfrak{A}}\|$$

This word corresponds to an execution of $\mathfrak{A}$, which starts at initiator b_o and halts at terminator e_o; and if $\mathfrak{A}$ is executed in $\underset{\sim}{D}$ with input state ξ, then this word tells us that $\sim r \wedge p\,[\underset{\sim}{D},\ \xi]$ and $r \wedge p[\underset{\sim}{D},\ k[\underset{\sim}{D},\ \xi]]$.



E-program M

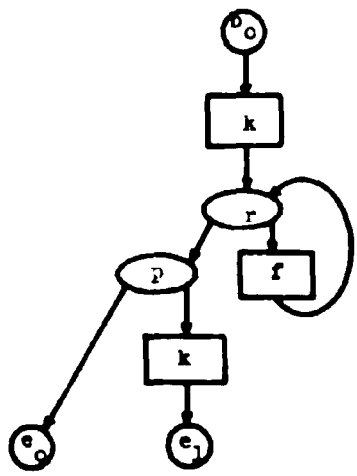


Figure 47

The upper figure shows E-program M in full detail. The lower figure is an abbreviated form where k stands for the assignment schema $v_6 := k_0$, and so on.

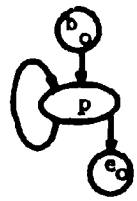


Figure 48

Here, p is a qff. The K-expression for this E-program is $b_0 p^* \bar{p} e_0$.

In general, as with certain words of $|\alpha_{\mathfrak{A}}|$, many words of $\|\alpha_{\mathfrak{A}}\|$ do not correspond to any execution of $\mathfrak{A}$ simply because the proposed sequence of words of truth and operators is not possible. An obvious example is

$$\bar{r}pb_o rpkrpe_o rp \in \|\alpha_{\mathfrak{A}}\| ,$$

since b_o certainly does not effect the input state and so could not reverse the truth-value of r. As another example, consider a word containing the sequence $p(v)r(u)v := f(v)p(v)\sim r(u)$. This suggests that for some $\underset{\sim}{D}$, and ξ, $p(v) \wedge r(u)[\underset{\sim}{D}, \xi]$ and $p(v) \wedge \sim r(u)[\underset{\sim}{D}, w := f(v)[\underset{\sim}{D}, \xi]]$, i.e., $(v := f(v) \rightarrow p(v) \wedge \sim r(u))[\underset{\sim}{D}, \xi]$, by Theorem 6, i.e., $p(f(v)) \wedge \sim r(u)[\underset{\sim}{D}, \xi]$. Taken together, these would give $p(v) \wedge r(u) \wedge p(f(v)) \wedge \sim r(u)[\underset{\sim}{D}, \xi]$, which is clearly impossible since $r(u) \wedge \sim r(u)$ is a logical contradiction.

To see how the concept of K-event will be more powerful in detecting strong equivalence, consider once again the wff of A_3^3 (cf. Figure 21). Earlier, we observed that $|p \vee \bar{p}| = [p, \bar{p}] \neq [1] = |1|$, so that the regular set approach to deciding strong equivalence is no help. But observe that

$$\begin{aligned} &\|p \vee \bar{p}\| \\ &= \|p\| \cup \|\bar{p}\| \\ &= \|p\| \cup (\mathcal{T} - \|p\|) \\ &= \mathcal{T} \\ &= \|1\| \end{aligned}$$

We we shall see shortly, equality of K-events does in general imply strong equivalence.

To see some of the implications of the semantics of K-expressions, consider the following examples.

(i) $\|p^*\| = \|1\| \cup \|p\| \cup \|pp\| \cup \ldots$ for $p \in \mathcal{P}$

Now, $\|pp\| = \{xs \in \|p\| \ \& \ sy \in \|p\| \ \& \ s \in \mathcal{T}\}$

$= \{s : s \in \|p\| \ \& \ s \in \|p\| \ \& \ s \in \mathcal{T}\}$

$= \|p\|$

so, $\|p^*\| = \|1\| \cup \|p\| \cup \|p\| \cup \|p\| \cup \ldots$

$= \mathcal{T} \cup \|p\|$

$= \mathcal{T}$

Thus, for the algorithm of Figure 48 with K-expression $b_o p^* \bar{p} e_o$, we have

$\|p^*\bar{p}\| = \{s : s \in \|p^*\| \ \& \ s \in \|\bar{p}\| \ \& \ s \in \mathcal{T}\}$

$= \{s : s \in \mathcal{T} \ \& \ s \in \|\bar{p}\| \ \& \ s \in \mathcal{T}\}$

$= \mathcal{T} \cap \|\bar{p}\|$

$= \|\bar{p}\|$

so, $\|b_o p^* \bar{p} e_o\| = \|b_o \bar{p} e_o\|$, since "." is associative.

(ii) $\|1\alpha\| = \{py : p \in \|1\| \ \& \ py \in \|\alpha\| \ \& \ p \in \mathcal{T}\}$, $\alpha \in \mathcal{K}$

$= \{py : p \in \mathcal{T} \ \& \ py \in \|\alpha\| \ \& \ p \in \mathcal{T}\}$

$= \|\alpha\|$

(iii) $\|\bar{1}\| = \mathcal{T} - \|1\| = \mathcal{T} - \mathcal{T} = \emptyset = \|0\|$

$\|\bar{0}\| = \mathcal{T} - \|0\| = \mathcal{T} - \emptyset = \mathcal{T} = \|1\|$

From these examples, we see that K-events have many useful and interesting properties: the "1" is seen to behave both as a symbol for truth and as an "identity" operator; the fact that the loop in the E-program of Figure 48 once entered is never left is reflected in the semantics for $p^*\bar{p}$; "0" and "1" are seen to behave consistently as symbols for truth and falsity.

Remarks:

(i) As pointed out earlier, the representation of an E-program (i.e., flowchart) as a regular expression is a natural adaptation of the state diagram methods of automata theory. Both Ito [18] and Engeler [8] have independently used syntactic representations closely related to that used by this author. However, the form of K-event and the description of K-expression semantics, as given here, has not appeared elsewhere.

(ii) This author feels that the notion of K-event and the scheme for attaching semantics to K-expressions represent a significant step forward in the description and representation of algorithms. As we indicate in the sequel, these ideas link in with, and bring cohesion to the work of Ianov [16] and Rutledge [57], and in addition, provide a starting point for even more productive investigations.

K-equivalence

Two K-expressions α and β are said to be K-equivalent iff $|\alpha| = |\beta|$. We introduce $\alpha = \beta$ as a well-formed formula expressing the K-equivalence of α and β, and write $\vdash \alpha = \beta$ just in case $|\alpha| = |\beta|$. We write $\mathfrak{Thm}$ for the set of all such wffs. K-equivalence can be a useful tool in the analysis of strong equivalence, as we see in the following

Theorem 27: For any E-programs $\mathfrak{A}$ and $\mathfrak{B}$ of the same type,

$$\vdash \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}} \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}.$$

This theorem is the counterpart of Theorem 26, but provides a much more powerful test for strong equivalence. If $\mathfrak{A}$ and $\mathfrak{B}$ are strongly equivalent because $\vdash \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$, then we say $\mathfrak{A}$ and $\mathfrak{B}$ are K-equivalent as well.

Proof: Actually what is properly called for here is a development similar to that given in the first part of this chapter. Then a proof for this theorem would follow as naturally as did the one for Theorem 26. However, here we give a less sophisticated proof, and this suffices for our present purposes.

With every halting execution of $\mathfrak{A}$, we can associate a word from $\|\alpha_{\mathfrak{A}}\|$, and similarly for $\mathfrak{B}$ and $\|\alpha_{\mathfrak{B}}\|$. Suppose, in fact, that for the computing structure $\underset{\sim}{D}$, state $\xi : \omega \to \underset{\sim}{D}_o$ and initiator b_i, $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate and produces the word

$$w = q_o x_o q_1 x_1 q_2 \cdots q_n x_n q_{n+1}$$

where $w \in \|\alpha_{\mathfrak{A}}\|$, $n < \omega$, $q_k \in \mathcal{T}$, $k < n+2$, $x_o = b_i$, $x_n = e_j$, $x_k \in \mathcal{A}$, $0 < k < n$ and, of course, $q_o = q_1$ and $q_n = q_{n+1}$. Since the hypothesis of the theorem gives that $\|\alpha_{\mathfrak{A}}\| = \|\alpha_{\mathfrak{B}}\|$, then $w \in \|\alpha_{\mathfrak{B}}\|$ as well; we will show that $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is also determinate and, in fact, produces this very word w. This, of course, gives $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle] = \mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$, for any $\underset{\sim}{D}$, ξ and i such that $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate. A similar result obtains when we assume that $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate, and both together give $\models \mathfrak{A} \simeq \mathfrak{B}$.

So we must show that $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ produces the word w assuming that $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ does. We do not assume that $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate, but show that the first $2n+3$ letters of u, the (possibly infinite) word produced by $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$, are those of w, and since $x_n = e_j$, this implies that $\mathfrak{B}$ does in fact halt, and that it produces just w. We proceed by discussing the various ways that u can differ from w.

Suppose that u and w differ first at some q_k. This cannot be q_o however, since $\mathfrak{A}$ and $\mathfrak{B}$ are each started with ξ and the atomic formulas

will have like truth-values in each case. Since $q_c = q_i$, i.e., b_i has no effect, then the first place u and w could differ is at q_2. Let us assume they differ first at q_r, $2 < r < n + 1$, so that

$$u = q_0 x_0 q_1 x_1 q_2 \cdots x_{r-1} p \cdots$$

where $p \neq q_r$; also recall that

$$w = q_0 x_0 q_1 x_1 q_2 \cdots x_{r-1} q_r \cdots q_n x_n q_{n+1} .$$

Since $\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i \rangle]$ produced u, we have that

$$q_r[\underset{\sim}{D}, (\ldots((x_1 \rightarrow x_2) \rightarrow x_3) \rightarrow \ldots \rightarrow x_{r-1})[\underset{\sim}{D}, \xi]]$$

But p and q_k are both words from $\mathcal{T}$, and from the definition of $\mathcal{T}$, $s[\underset{\sim}{D}, \xi] \neq t[\underset{\sim}{D}, \xi]$, for $s, t \in \mathcal{T} \Rightarrow s$ is t, i.e., only one of the disjuncts in the full disjunctive normal form of a tautology can be true for any given truth assignment. Thus $p = q_r$.

Suppose that u and w differ first at some x_k. This cannot be x_0 however, since $\mathfrak{M}$ and $\mathfrak{B}$ are each started at initiator b_i. Let us assume they differ first at x_r, $0 < r < n+1$, so that

$$u = q_0 x_0 q_1 x_1 q_2 \cdots x_{r-1} q_r y \cdots$$

where $y \in \mathcal{A} \cup \mathcal{E}$, $y \neq x_r$; also recall that

$$w = q_0 x_0 q_1 x_1 q_2 \cdots x_{r-1} q_r x_r \cdots q_n x_n q_{n+1} .$$

Since $w \in \|\alpha_{\mathfrak{M}}\|$ and $\|\alpha_{\mathfrak{M}}\| = \|\alpha_{\mathfrak{B}}\|$, then $w \in \|\alpha_{\mathfrak{B}}\|$. So there is a path through $\mathfrak{B}$, starting at the node labelled b_i and proceeding to one labelled x_{r-1}. Since $\mathfrak{M}[\underset{\sim}{D}, \langle \xi, i \rangle]$ produces u, we can consider another path through $\mathfrak{B}$ starting at the node labelled b_i and proceeding to one labelled x_{r-1}. We want to show that these two paths must, in fact, be the same path. The only way the paths could differ is if at some discriminator p, say, one path takes the true branch and the other the false branch. But this would imply

that w and u must differ at some q_k, $1 < k < r+1$, and this is not so since we are assuming u and w are identical up to q_r. Thus the assignment schemata y and x_r must be reachable from the same node, i.e., one labelled x_{r-1}.

If $\mathfrak{B} = \langle X, \Gamma, \mathcal{L} \rangle$, and $x \in X$ is the node such that $[x]$ is the assignment schema x_{r-1}, and if $\Gamma x \in X(\mathcal{A}) \cup X(\mathcal{E})$, then both u and w must have $[\Gamma x]$ as the next letter after q_r, i.e., u and w do not differ at x_r. So the case left to consider is where there are discriminators intervening between x_{r-1} and y in the path corresponding to u, and between x_{r-1} and x_r in the path corresponding to w. This situation is illustrated in Figure 49. Since $w \in \|\alpha_{\mathfrak{B}}\|$, the definition of the semantics for "." tells us that $q_r \in \|t\|$ where t is the qff specifying the condition for reaching x_r; in Figure 49, we have $q_r \in \|\bar{t}_1 \wedge t_3\|$. But one of the conjuncts in t must be negated in the condition t' for reaching y; in Figure 49, we have $t' = t_1 \wedge t_2$. Therefore $\|t'\| \cap \|t\| = \emptyset$, and so $q_r \notin \|t'\|$, which means the path to y cannot be executed. Simply put, the topology of the situation establishes a certain qff which if true implies we get y next, and if false implies we get x_r next. Then since $w \in \|\alpha_{\mathfrak{B}}\|$, this eliminates the former possibility, and so we must get x_r as the next letter.

From the preceding arguments,

$$u = q_0 x_0 q_1 x_1 q_2 \cdots q_n x_n \cdots,$$

but since $x_n = e_j$, this means $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ terminates at this point, and so $q_{n+1} = q_n$ is the last letter of u.

So, we conclude that $u = w$, i.e., if $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate and produces w, then $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate and produces w. Since the sequences of assignment schemata encountered are therefore identical, then $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle] = \mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$. An argument similar to the foregoing yields that

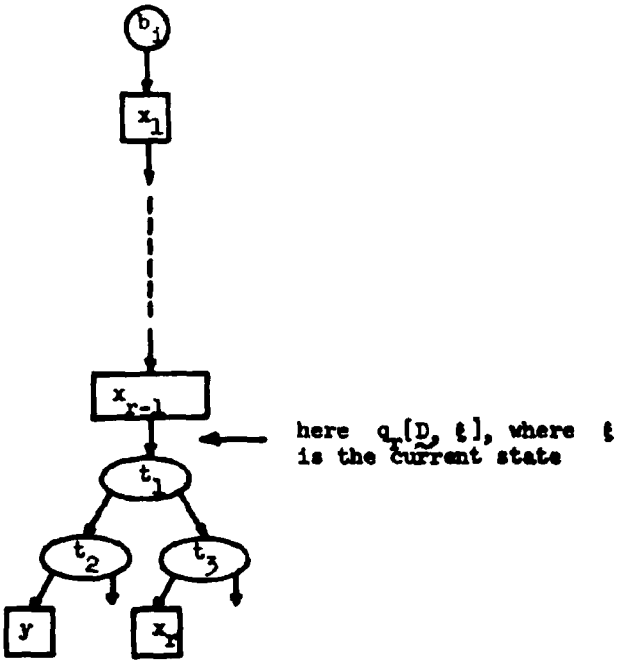


Figure 49

Here p, q are qffs; $x_1, \ldots, x_{r-1}$ are assignment schemata and each of y and x_r may be either an assignment schema or the terminator e_j.

if $\mathfrak{M}[D, \langle \xi, i \rangle]$ is determinate and produces w, then $\mathfrak{N}[D, \langle \xi, i \rangle]$ is determinate and produces w. Again, we obtain $\mathfrak{M}[D, \langle \xi, i \rangle] = \mathfrak{N}[D, \langle \xi, i \rangle]$. Together, these results give $\mathfrak{M}[D, \langle \xi, i \rangle] \simeq \mathfrak{N}[D, \langle \xi, i \rangle]$ for any D, ξ and i. Thus $\models \mathfrak{M} \simeq \mathfrak{N}$.

Remarks:

(i) Certainly, the proof of Theorem 27 can be given in a far more precise form, for example, in the context of a supporting set of theorems like that given for Theorem 26.

(ii) Theorem 27 is useful only if $?\models \alpha = \beta?$ is decidable for arbitrary K-expressions α and β. It is to this problem that we now turn our attention.

An Inferential System for K-equivalence

We investigate the decidability of K-equivalence by studying the properties of $\mathcal{I}$, an inferential system for deriving wffs in $\mathcal{Fm}$. Together, $\mathcal{Fm}$ and $\mathcal{I}$ constitute the formal theory of K-equivalence, $\mathcal{T}_K = \langle \mathcal{Fm}, \mathcal{I} \rangle$. Later, we show that $\mathcal{I}$ is complete for $\mathcal{Fm}$, and that $?\models \alpha = \beta?$ is decidable for arbitrary $\alpha = \beta \in \mathcal{Fm}$.

Here, and in subsequent discussions concerning the completeness of $\mathcal{I}$, we deal with some fixed set $\mathcal{K}$ of K-expressions defined over some fixed alphabets $\mathcal{Atm}$ and $\mathcal{b}$. For these discussions, it will not concern us what the actual constituents of these alphabets are, or whence they came. The point is simply that one application of K-expressions can involve the study of strong equivalence for E-programs. In that case, the alphabets are defined as $\mathcal{Atm} = \mathcal{Atm}_{\mathfrak{M}} \cup \mathcal{Atm}_{\mathfrak{N}}$ and $\mathcal{b} = \mathcal{b}_{\mathfrak{M}} \cup \mathcal{b}_{\mathfrak{N}}$, where the strong equivalence of E-programs $\mathfrak{M}$ and $\mathfrak{N}$ is the point in question. This is why the signature $\mathcal{s}$ plays no direct role here, i.e., the set $\mathcal{K}$ of K-expressions is determined solely by the sets $\mathcal{Atm}$ and $\mathcal{b}$.

The inferential system $\mathcal{I} = \langle \mathcal{A}x_K, \mathcal{R}_K \rangle$ specifies a finite set $\mathcal{A}x_K$ of axiom schemata, the instances of which constitute a subset of $\mathcal{F}_{\mathcal{M}}$, and a finite set $\mathcal{R}_K$ of rules of inference. If $\alpha = \beta \in \mathcal{K}$ is finitely derivable using $\mathcal{I}$, then we say $\alpha = \beta$ is a theorem and write $\vdash \alpha = \beta$.

The axiom schemata and rules of $\mathcal{I}$ come from two sources:

(i) an analysis of the axiom schemata and rules of $\mathcal{I}_S$ to determine what properties of K-expressions they imply, and

(ii) an analysis of the inferential system given by Salomaa [38] for deriving wffs of the form $\alpha = \beta$ where α and β are regular expressions.

We start by analyzing axiom schemata A1 through A7 of $\mathcal{A}x$. From axiom schema A1 in Figure 21, we obtain the following wff of $\mathcal{F}_{\mathcal{M}}$:

$$b_o p p e_c \vee b_o p \bar{p} e_1 \vee b_o \bar{p} e_2 = b_o p e_o \vee b_o \bar{p} e_2$$

which suggests the following axiom schemata for $\mathcal{A}x_K$

$$\left.\begin{array}{ll} \underline{C1}: & pp = p \\ \underline{C2}: & p\bar{p} = 0 \end{array}\right\} \quad p \text{ is any proposition in } \mathcal{P}$$

From axiom schema A2, we obtain

$$b_o pqe_c \vee b_o p\bar{q}e_1 \vee b_o \bar{p} e_2 = b_o qpe_o \vee b_o \bar{q}pe_1 \vee b_o(q\bar{p} \vee \bar{q}\bar{p})e_2$$

which suggests the following axiom schema for $\mathcal{A}x_K$

$\underline{C3}$: $pq = qp$ $\quad p, q \in \mathcal{P}$.

From axiom schema A3, we obtain

$$b_o(p \vee \bar{p})e_o = 1$$

which suggests the following axiom schema for $\mathcal{A}x_K$

$\underline{C4}$: $p \vee \bar{p} = 1$ $\quad p \in \mathcal{P}$

From axiom schema A4, we obtain

$$b_o(p \supset q)e_o \vee b_o \sim(p \supset q)e_1 = b_o(\bar{p} \vee pq)e_c \vee b_o p\bar{q}e_1$$

which suggests the following axiom schemata for $\mathcal{A}_K$

$$\left.\begin{array}{l} \underline{C5}: p \supset q = \bar{p} \vee q \\ \underline{C6}: \sim(p \supset q) = p\bar{q} \end{array}\right\} \quad p, q \in \mathcal{P}$$

From axiom schema A5, we obtain

$$b_o\bar{p}e_o \vee b_o\bar{p}e_1 = b_o\bar{p}e_o \vee b_o pe_1$$

which suggests the following axiom schema for $\mathcal{A}_K$

$$\underline{C7}: \bar{\bar{p}} = p \qquad p \in \mathcal{P}$$

The remainder of the axiom schemata in $\mathcal{A}_K$ are taken directly from [38] where Salomaa uses them to characterize the equality of regular expressions.

$\underline{S1}$: $\alpha \vee (\beta \vee \gamma) = (\alpha \vee \beta) \vee \gamma$

$\underline{S2}$: $\alpha(\beta\gamma) = (\alpha\beta)\gamma$

$\underline{S3}$: $\alpha \vee \beta = \beta \vee \alpha$

$\underline{S4}$: $\alpha(\beta \vee \gamma) = \alpha\beta \vee \alpha\gamma$

$\underline{S5}$: $(\alpha \vee \beta)\gamma = \alpha\gamma \vee \beta\gamma$

$\underline{S6}$: $\alpha \vee \alpha = \alpha$

$\underline{S7}$: $\alpha\, 1 = \alpha$

$\underline{S8}$: $\alpha\, 0 = 0$

$\underline{S9}$: $\alpha \vee 0 = \alpha$

$\underline{S10}$: $\alpha^* = 1 \vee \alpha\alpha^*$

$\underline{S11}$: $\alpha^* = (1 \vee \alpha)^*$

This completes the definition of $\mathcal{A}_K = \{\underline{C1}, \ldots, \underline{C7}, \underline{S1}, \ldots, \underline{S11}\}$. The set $\mathcal{R}_K = \{\underline{T1}, \underline{T2}, \underline{T3}\}$ is defined as follows.

$\underline{T1}$: $\alpha = \beta \Rightarrow \beta = \alpha$

$\underline{T2}$: $\alpha = \beta \Rightarrow \gamma(\alpha) = \gamma(\beta)$

In $\underline{T2}$, one or more occurrences of the K-expression α in $\gamma(\alpha)$ is replaced by β to give $\gamma(\beta)$. The rules $\underline{T1}$ and $\underline{T2}$ correspond to $\underline{R1}$ and $\underline{R2}$ of $\mathcal{R}$ and serve to characterize "=" as an equality relation.

$\underline{\text{I2}}$: $\alpha = \gamma \vee (\alpha \Rightarrow \alpha = \beta * \gamma$ provided $\mathcal{T} \not\subseteq \|\beta\|$.

Soundness of the Theory $\mathcal{T}_K$

Once again, if the theory $\mathcal{T}_K$ is to be useful in the derivation of K-equivalence, then we should require that the theorems $\alpha = \beta$ of $\mathcal{T}_K$ be valid, i.e., $\models \alpha = \beta$.

Theorem 28: The theory $\mathcal{T}_K$ is sound, i.e., for all $\alpha = \beta \in \mathcal{K}$, $\vdash \alpha = \beta \Rightarrow \models \alpha = \beta$.

Proof: It is sufficient to show that the axiom schemata in the set $\mathcal{A}_K$ generate valid wffs, and that the rules of inference in $\mathcal{R}_K$ all preserve validity.

Axiom schema C1:

$$\begin{aligned}\|pp\| &= \{xsy : xs \in \|p\| \ \&\ sy \in \|p\| \ \&\ s \in \mathcal{T}\} \\ &= \{s : s \in \|p\| \ \&\ s \in \|p\| \ \&\ s \in \mathcal{T}\} \\ &= \|p\|\end{aligned}$$

This follows from the obvious result that for all $p \in \mathcal{P}$, $\|p\| \subseteq \mathcal{T}$.

Axiom schema C2:

$$\begin{aligned}\|p\bar{p}\| &= \{xsy : xs \in \|p\| \ \&\ sy \in \|\bar{p}\| \ \&\ s \in \mathcal{T}\} \\ &= \{s : s \in \|p\| \ \&\ s \in \mathcal{T} - \|p\| \ \&\ s \in \mathcal{T}\} \\ &= \emptyset \\ &= \|0\|\end{aligned}$$

Axiom schema C3:

$$\begin{aligned}\|pq\| &= \{xsy : xs \in \|p\| \ \&\ sy \in \|q\| \ \&\ s \in \mathcal{T}\} \\ &= \{s : s \in \|p\| \ \&\ s \in \|q\| \ \&\ s \in \mathcal{T}\}\end{aligned}$$

$= \|p\| \cap \|q\|$

$= \|q\| \cap \|p\|$

$= \{s : s \in \|q\| \ \& \ s \in \|p\| \ \& \ s \subset \mathcal{T}\}$

$= \|qp\|$

Axiom schema C4**:**

$\|p \vee \bar{p}\| = \|p\| \cup \|\bar{p}\|$

$= \|p\| \cup (\mathcal{T} - \|p\|)$

$= \mathcal{T}$

$= \|1\|$

Axiom schema C5**:**

$\|p \supset q\| = \|\bar{p}\| \cup \|q\|$

$= \|\bar{p} \vee q\|$

Axiom schema C6**:**

$\|\vdash(p \supset q)\| = \mathcal{T} - \|p \supset q\|$

$= \mathcal{T} - (\|\bar{p}\| \cup \|q\|)$

$= \mathcal{T} - ((\mathcal{T} - \|p\|) \cup \|q\|)$

$= \mathcal{T} - ((\mathcal{T} - \|p\|) \cup (\mathcal{T} - (\mathcal{T} - \|q\|)))$

$= \mathcal{T} - (\mathcal{T} - (\|p\| \cap (\mathcal{T} - \|q\|)))$

$\|p\| \cap (\mathcal{T} - \|q\|)$

$= \|p\| \cap \|\bar{q}\|$

$= \{s : s \in \|p\| \ \& \ s \in \|\bar{q}\| \ \& \ s \in \mathcal{T}\}$

$= \|p\bar{q}\|$

Axiom schema C7**:**

$\|\bar{\bar{p}}\| = \mathcal{T} - \|\bar{p}\|$

$= \mathcal{T} - (\mathcal{T} - \|p\|)$

$= \|p\|$

Axiom schemata S1,... S6: by the properties of set union and set intersection. The "∨" operation directly corresponds to set union, and the "·" operation corresponds to set intersection directly for K-expressions which are propositions in $\mathcal{P}$ and indirectly for K-expressions in general. To see this latter point, notice that if

$$S_\alpha = \{s \in \mathcal{T}: \text{ for some } x\text{, } xs \in \|\alpha\|\}$$

$$S_\beta = \{s \in \mathcal{T}: \text{ for some } y\text{, } sy \in \|\beta\|\}$$

$$X_\alpha = \{x : \text{ for some } s \in \mathcal{T}, \ xs \in \|\alpha\|\}$$

$$X_\beta = \{y : \text{ for some } s \in \mathcal{T}, \ sy \in \|\beta\|\}$$

then

$$\|\alpha \cdot \beta\| = \{xsy : x \in X_\alpha \ \& \ s \in S_\alpha \cap S_\beta \ \& \ y \in X_\beta\} .$$

Axiom schema S7:

$$\|\alpha 1\| = \{xsy : xs \in \|\alpha\| \ \& \ sy \in \|1\| \ \& \ s \in \mathcal{T}\}$$
$$= \{xs : xs \in \|\alpha\| \ \& \ s \in \mathcal{T}\}$$
$$= \|\alpha\|$$

Axiom schema S8:

$$\|\alpha 0\| = \{xsy : xs \in \|\alpha\| \ \& \ sy \in \|0\| \ \& \ s \in \mathcal{T}\}$$
$$= \{xsy : xs \in \|\alpha\| \ \& \ sy \in \emptyset \ \& \ s \in \mathcal{T}\}$$
$$= \emptyset$$
$$= \|0\|$$

Axiom schema S9:

$$\|\alpha \vee 0\| = \|\alpha\| \cup \|0\|$$
$$= \|\alpha\| \cup \emptyset$$
$$= \|\alpha\|$$

Axiom schema S10:

$$
\begin{aligned}
&\|1 \vee \alpha\alpha^*\| \\
&= \|1\| \cup \|\alpha\alpha^*\| \\
&= \|1\| \cup \{xsy : xs \in \|\alpha\| \ \&\ sy \in \|\alpha^*\| \ \&\ s \in \mathcal{T}\} \\
&= \|1\| \cup \{xsy : xs \in \|\alpha\| \ \&\ sy \in \|1\| \cup \|\alpha\| \cup \|\alpha\alpha\| \cup \ldots \ \&\ s \in \mathcal{T}\} \\
&= \|1\| \cup \{xsy : xs \in \|\alpha\| \ \&\ sy \in \|1\|\} \\
&\qquad \cup \{xsy : xs \in \|x\| \ \&\ sy \in \|\alpha\|\} \\
&\qquad \cup \{xsy : xs \in \|\alpha\| \ \&\ sy \in \|\alpha\alpha\|\} \\
&\qquad \vdots \\
&= \|1\| \cup \|\alpha\| \cup \|\alpha\alpha\| \cup \|\alpha\alpha\alpha\| \cup \ldots \\
&= \|\alpha\|
\end{aligned}
$$

Axiom schema S11:

$$
\begin{aligned}
&\|(1 \vee \alpha)^*\| \\
&= \|1\| \cup \|1 \vee \alpha\| \cup \|(1 \vee \alpha)(1 \vee \alpha)\| \cup \| (1 \vee \alpha)(1 \vee \alpha)(1 \vee \alpha)\| \cup \ldots \\
&= \|1\| \cup (\|1\| \cup \|\alpha\|) \\
&\qquad \cup (\|1\| \cup \|\alpha\| \cup \|\alpha\alpha'\|) \\
&\qquad \cup (\|1\| \cup \|\alpha\| \cup \|\alpha\alpha\| \cup \|\alpha\alpha\alpha\|) \\
&\qquad \vdots \\
&= \|1\| \cup \|\alpha\| \cup \|\alpha\alpha\| \cup \|\alpha\alpha\alpha\| \cup \ldots \\
&= \|\alpha^*\|
\end{aligned}
$$

Rules T1, T2: these rules reflect the symmetry and substitutivity properties of set equality.

Rule T3: First we show that $\alpha = \beta^*\gamma$ satisfies the equation $\alpha = \gamma \vee \beta\alpha$. Substituting for α, we have

$\|\gamma \vee \beta(\beta^*\gamma)\|$

$= \|(1 \vee \beta\beta^*)\gamma\|$ S5, and since $\|1\gamma\| = \|\gamma\|$

$= \|\beta^*\gamma\|$ S10

Now we show that the solution $\alpha = \beta^*\gamma$ is unique iff $\mathcal{T} \notin \|\beta\|$.

Suppose that $\mathcal{T} \notin \|\beta\|$., but that there exists another solution Y, i.e., $\|Y\| = \|\gamma \vee \beta Y\|$. Then from

$\|\beta\gamma^*\| = \|\gamma \vee \beta\beta^*\gamma\|$, and

$\|Y\| = \|\gamma \vee \beta Y\|$

we obtain

$\|\beta^*\gamma\| - \|Y\|$

$= \|\gamma \vee \beta\beta^*\gamma\| - \|\gamma \vee \beta Y\|$

$= (\|\gamma\| \cup \|\beta\beta^*\gamma\|) - (\|\gamma\| \cup \|\beta Y\|)$

$= \|\beta\beta^*\gamma\| - \|\beta Y\|$

$= \{xpy : xp \in \|\beta\| \ \& \ py \in \|\beta^*\gamma\|\} - \{uqv : uq \in \|\beta\| \ \& \ qv \in \|Y\|\}$

$= \{xpy : xp \in \|\beta\| \ \& \ py \in \|\beta^*\gamma\| - \|Y\|\}$

so that

$(\|\beta^*\gamma\| - \|Y\|) - \{xpy : xp \in \|\beta\| \ \& \ py \in (\|\beta^*\gamma\| - \|Y\|)\} = \emptyset$

i.e, $\{xpy : xp \in (\mathcal{T} - \|\beta\|) \ \& \ py \in (\|\beta^*\gamma\| - \|Y\|)\} = \emptyset$.

But this means that either $\mathcal{T} - \|\beta\| = \emptyset$ or $\|\beta^*\gamma\| - \|Y\| = \emptyset$. Since $\mathcal{T} \notin \|\beta\|$, we cannot have $\mathcal{T} - \|\beta\| = \emptyset$, so therefore we must have $\|\beta^*\gamma\| - \|Y\| = \emptyset$, i.e., $\|\beta^*\gamma\| = \|Y\|$. Thus, $\mathcal{T} \notin \|\beta\| \Rightarrow$ the solution $\alpha = \beta^*\gamma$ is unique. This argument could be simplified somewhat if "-" were introduced as part of the formal definition of K-expressions, i.e., if we defined $\|\alpha - \beta\| = \|\alpha\| - \|\beta\|$.

To see that the solution $\alpha = \beta^*\gamma$ is no longer unique if $\mathcal{T} \subseteq \|\beta\|$, first observe that $\mathcal{T} \subseteq \|\beta\|$ implies that for some K-expressions μ, ν either (i) β is 1, or (ii) β is $\mu \vee \nu$ and either $\mathcal{T} \subseteq \|\mu\|$ or $\mathcal{T} \subseteq \|\nu\|$, or (iii) β is $\mu\nu$ and $\mathcal{T} \subseteq \|\mu\|$ and $\mathcal{T} \subseteq \|\nu\|$, or (iv) β is μ^*. Then a simple inductive argument on the structure of β gives us that $\mathcal{T} \subseteq \|\beta\| \Rightarrow \|\beta\| = \|1 \vee \delta\|$ for some K-expression δ.

Then, assuming $\mathcal{T} \subseteq \|\beta\|$ and so $\|\beta\| = \|1 \vee \delta\|$, we can show that $\alpha = \beta^*\gamma \vee \beta^*X$, for any K-expression X, is also a solution of $\alpha = \gamma \vee \beta\alpha$. Substituting for α, we have

$\|\alpha \vee \beta(\beta^*\gamma \vee \beta^*X)\|$

$= \|\gamma \vee (1 \vee \delta)((1 \vee \delta)^*\gamma \vee (1 \vee \delta)^*X)\|$

$= \|\gamma \vee (1 \vee \delta)(\delta^*\gamma \vee \delta^*X)\|$ S11

$= \|\gamma \vee \delta^*\gamma \vee \delta^*X \vee \delta\delta^*\gamma \vee \delta\delta^*X\|$ S4

$= \|(1 \vee \delta\delta^*)\gamma \vee \delta^*\gamma \vee (\delta^* \vee \delta\delta^*)X\|$ S5

$= \|\delta^*\gamma \vee \delta^*\gamma \vee (1 \vee \delta\delta^* \vee \delta\delta^*)X\|$ S10

$= \| \delta^*\gamma \vee (1 \vee \delta\delta^*)X\|$ S6

$= \| \delta^*\gamma \vee \delta^*X\|$ S10

$= \| (1 \vee \delta)^*\gamma \vee (1 \vee \delta)^*X\|$ S11

$= \| \beta^*\gamma \vee \beta^*X\|$ as required.

This completes the proof of soundness of $\mathcal{T}_K$. ∎

One immediate and very useful result of this soundness of $\mathcal{T}_K$ arises from the identity of S1,..., S11, T1, T2, T3 with Salomma's axioms and rules [S8]. Evidently, the set $\mathcal{K}$ of K-expressions over the alphabets $\mathcal{b}$ and $\mathcal{M}$ is identical to the set of regular expressions over the alphabet $\Sigma = \mathcal{b} \cup (\mathcal{P} - \{0, 1\})$. Now, if α and β are two such regular expressions and $|\alpha| = |\beta|$, then $\alpha = \beta$ is derivable from S1 through S8 using T1, T2

and T3. Thus, any of the standard derivability results concerning regular expressions can be applied to K-expressions if we take the alphabet as Σ. In fact, this is true for any sub-set of $\mathcal{K}$ that can be regarded as regular over some alphabet. Among the results we often use are

P1: $(\alpha \vee \beta)^* = \alpha^*(\beta\alpha^*)^*$

P2: $\alpha(\beta\alpha)^* = (\alpha\beta)^*\alpha$

P3: $0\alpha = 0$

P4: $1\alpha = \alpha$

In derivations, we will use P1,..., P4 to denote these standard derivations.

Remarks:

(i) The K-expressions derived from E-programs have certain characteristic feactures. For example, we can write any such K-expression in a form where "∨" and "*" appear only in the constructs $(pu \vee \bar{p}v)$ and $(pu)^*\bar{p}$ respectively. Here u and v are any K-expressions and p is a qff serving as a discriminator in the E-program in question. Apparently, Engeler [8] has pinned the form of such K-expressions down precisely. Ito [18] discusses what sort of objects could give rise to K-expressions not derived from schemes like E-programs, namely certain non-deterministic programs.

(ii) By making use of this knowledge of the form of K-expressions derived from E-programs, one can show that for such K-expressions, the condition on rule T3, namely that "$\mathcal{T} \not\subseteq \|\beta\|$" can be reduced to $\mathcal{T} \not\models \|\beta\|$.

(iii) Also, it is possible to give interpretation of Salomaa's axioms S1,..., S5 in terms of the structural properties of E-programs. The

point is that properties that can remain unformalized in the graph representation of E-programs, must submit to axiomatization when the linear representation of K-expressions is adopted.

(iv) As well, the rule T3 in $\mathcal{H}_K$ is a direct counterpart of R5', the recursion rule, in $\mathcal{S}_s$. The connection between these two warrants further investigation, especially the relationship of the side conditions for the application of each.

(v) With a complete knowledge of the relationship between E-programs and the structure of K-expressions derived from them, we can see how to extend the result of Theorem 27. Thus, if $\vdash \alpha_{\mathfrak{M}} = \alpha_{\mathfrak{N}}$ then $\vdash \mathfrak{M} \simeq \mathfrak{N}$, but in addition, we can derive $\alpha_{\mathfrak{M}} = \alpha_{\mathfrak{N}}$ using the axioms and rules of $\mathcal{S}$. Then this derivation could be used to indicate a derivation using $\mathcal{S}_s$ of the wff $\mathfrak{M} \simeq \mathfrak{N}$, so that not only would strong equivalence be detected, it would be derived as well. We leave these matters in their present incomplete state.

Adequacy of the Theory $\mathcal{T}_K$

The theory $\mathcal{T}_K$ being adequate means that if $\vdash \alpha = \beta$ where $\alpha, \beta \in \mathcal{K}$, then $\vdash \alpha = \beta$; thus, we can derive all instances of K-equivalence. The first step in obtaining this result is to prove propositional adequacy, and this is the content of the following

Theorem 29: $\vdash p = q \Rightarrow \vdash p = q$ for arbitrary propositions $p, q \in \mathcal{P}$.

This theorem in fact assures us that the axioms C1,..., C7 are complete in the sense of the propositional calculus.

Proof: The first [illegible] to [illegible] validity of wffs of the form $p = 1$ (where $p \in \mathcal{P}$) corresponds to [illegible], regarded as a statement of the propositional calculus. For any $p \in \mathcal{P}$ we develop the notions of truth assignment and truth-value under such a truth assignment. It will be convenient to regard a word of truth, i.e., $v \in \mathcal{T}$, as a truth assignment to the atomic formulas in $Atm = \{p_0, \ldots, p_{m-1}\}$, m [illegible], truth assignment $v = v_0 v_1 \ldots v_{m-1}$ acts as a mapping $v : \mathcal{P} \to \{0,1\}$ which for any proposition in $\mathcal{P}$ generates [illegible]:

$$\left.\begin{array}{lll} v(p) = 1 & \text{if} & v(p) = 1 \\ = 0 & \text{otherwise} & \\ v(\sim p) = 1 & \text{if} & v(p) = 0 \\ = 0 & \text{otherwise} & \\ v(p_i) = 1 & \text{if} & v_i = 1 \\ = 0 & \text{if} & v_i = \sim p_i \end{array}\right\} p, q \in \mathcal{P}$$

$$v(0) = 0$$

$$v(1) = 1$$

Since we are following the notions of propositional calculus, we say $p \in \mathcal{P}$ is tautology, and write $\models^+ p$, iff for all $v \in \mathcal{T}$, $v(p) = 1$. We tie together the notions of K-events of propositions being truth, and those propositions being tautologies, in the following

Theorem 30: For any proposition $p \in \mathcal{P}$, $\models p = 1 \Leftrightarrow \models^+ p$, i.e., the K-event for p is truth just in case p is a tautology.

Proof: Evidently, we must show $u \in \|p\|$ for all $u \in \mathcal{T} \Leftrightarrow u(p) = 1$ for all

$u \in \mathcal{T}$. It is sufficient then to show that $v \in \|p\| \Leftrightarrow v(p) = 1$ for an arbitrary $v \in \mathcal{T}$. We use an inductive proof on the structure of p.

(i) If p is 1, then $\|p\| = \|1\| = \mathcal{T}$ and so $u \in \|p\|$ for all $u \in \mathcal{T}$. Moreover, $u(p) = u(1) = 1$ for all $u \in \mathcal{T}$. Thus $v \in \|p\| \Leftrightarrow v(p) = 1$, since $v \in \mathcal{T}$.

(ii) If p is 0, then $\|p\| = \|0\| = \emptyset$ and so $u \notin \|p\|$ for all $u \in \mathcal{T}$. Moreover, $u(p) = u(0) = 0$, i.e., $u(p) \neq 1$, for all $u \in \mathcal{T}$. Thus $v \in \|p\| \Leftrightarrow v(p) = 1$, since $v \in \mathcal{T}$.

(iii) If p is $p_i \in Atm$, then $\|p\| = \|p_i\| = \{s_0\ s_1 \ldots s_{m-1} \in \mathcal{T}: s_i = p_i\}$. Thus, $v \in \|p_i\| \Leftrightarrow v_i = p_i$. Moreover, $v(p) = v(p_i) = 1$ iff $v_i = p_i$, so that together we have $v \in \|p\| \Leftrightarrow v(p) = 1$.

(iv) If p is $(q \supset r)$, $q, r \in \mathcal{P}$, then $\|p\| = \|q \supset r\| = \|\sim q\| \cup \|r\|$. Thus, $v \in \|p\| \Leftrightarrow v \in \|\sim q\| \cup \|r\|$, i.e., $v \notin \|q\|$ or $v \in \|r\|$. Moreover, $v(p) = v(q \supset r) = 1$ iff $v(\sim q) = 1$ or $v(r) = 1$, i.e., $v(q) \neq 1$ or $v(r) = 1$. By induction hypothesis, $v \in \|r\| \Leftrightarrow v(r) = 1$ and $v \in \|q\| \Leftrightarrow v(q) = 1$, i.e., $v \notin \|q\| \Leftrightarrow v(q) \neq 1$. Then, taken together these results give $v \in \|p\| \Leftrightarrow v(p) = 1$.

(v) If p is $(\sim q)$, $q \in \mathcal{P}$, then $\|p\| = \|\sim q\| = \mathcal{T} - \|q\|$. Thus, $v \in \|p\| \Leftrightarrow v \in \mathcal{T} - \|q\|$, i.e., $v \notin \|q\|$. Moreover, $v(p) = v(\sim q) = 1$ iff $v(q) = 0$, i.e., $v(q) \neq 1$. By induction hypothesis, $v \in \|q\| \Leftrightarrow v(q) = 1$, i.e., $v \notin \|q\| \Leftrightarrow v(q) \neq 1$. Then taken together these results give $v \in \|p\| \Leftrightarrow v(p) = 1$. This completes the induction and proof. ∎

Because the classical propositional calculus does not allow the constants 0 and 1, we will have cause to utilize the mapping $R : \mathcal{P} \to \mathcal{P}^-$ defined as follows

$$R(0) = \sim(p_0 \supset p_0)$$
$$R(1) = (p_0 \supset p_0)$$
$$R(p \supset q) = R(p) \supset R(q)$$
$$R(\sim p) = \sim R(p)$$
$$R(p_i) = p_i \qquad \Big\} \quad p, q \in \mathcal{P}$$

Thus, for any $p \in \mathcal{P}$, $R(p)$ contains no 0 or 1 but for all $v \in \mathcal{T}$, $v(p) = v(R(p))$. This result follows from a simple inductive proof on the structure of p utilizing the fact that $v(p_0 \supset p_0) = 1$, since either $v_0 = p_0$ or $v_0 = \sim p_0$.

Consider the following three axiom schemata and rule of inference.

(i) $p \supset (q \supset p)$

(ii) $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$

(iii) $(\sim p \supset q) \supset ((\sim p \supset \sim q) \supset p)$

(iv) $p,\ p \supset q \Rightarrow q$ modus ponens

$\Big\}$ $p, q \in \mathcal{P}^{\sigma}$

This inferential system is given by Mendelson [33] for the propositional calculus. If a proposition $p \in \mathcal{P}^{\sigma}$ is derivable in this system, then we write $\vdash^{+} p$ to denote this. Since this system is known to be both sound and adequate, we have that for all $p \in \mathcal{P}^{\sigma}$, $\models^{+} p \Leftrightarrow \vdash^{+} p$.

Let us now outline the steps in proving Theorem 29. First, for any $p \in \mathcal{P}$, we have

$\models p = 1 \Rightarrow \models^{+} p$	by Theorem 30
$\Rightarrow \models^{+} R(p)$	property of the mapping R
$\Rightarrow \vdash^{+} R(p)$	completeness of the inferential system for propositional calculus
$\Rightarrow \vdash R(p) = 1$	this we must show
$\Rightarrow \vdash p = 1$	this we must show

Then this will allow us to finally show that $\vdash p = 1 \Rightarrow \vdash p = q$, for any propositions $p, q \in \mathcal{P}$. So the first step is to show that $\vdash^{+} R(p) \Rightarrow \vdash R(p) = 1$, i.e., we have to show how to mimic a derivation of $R(p)$ using the inferential system given above for the propositional calculus, so that a derivation of $R(p) = 1$ using the inferential system $\mathcal{I}$ is produced.

In Appendix III, we show how to construct the following derivations:

(i) $\vdash (p \supset (q \supset p)) = 1$

(ii) $\vdash ((p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))) = 1$

(iii) $\vdash ((\sim p \supset q) \supset ((\sim p \supset \sim q) \supset p)) = 1$

(iv) $\vdash p \supset q = 1$, $\vdash p = 1 \Rightarrow \vdash q = 1$.

Then, given any derivation $\vdash^{+} p$, we can construct the required derivation $\vdash p = 1$ by mimicking each step of the former with the appropriate derivation given in Appendix III.

The next step is to show that $\vdash R(p) = 1 \Rightarrow \vdash p = 1$. The appropriate occurrences of $\sim(p_o \supset p_o)$ and $(p_o \supset p_o)$ in $R(p)$ can be replaced by 0 and 1 using the following derivations:

(i) $\sim(p_o \supset p_o) = p_o \sim p_o$ C6

$= 0$ C2

(ii) $(p_o \supset p_o) = \sim p_o \vee p_o$ C5

$= p_o \vee \sim p_o$ S3

$= 1$ C4

Thus, so far we have shown that for any $p \in \mathcal{P}$, $\vdash p = 1 \Rightarrow \vdash p = 1$. Now, to show $\vdash p = q \Rightarrow \vdash p = q$, note that $\vdash p = q \Rightarrow \vdash \sim p \sim q \vee pq = 1$. We can easily verify this as follows.

$$\begin{aligned}
\| \sim p \sim q \vee pq\| &= \|\sim p \sim q\| \cup \|pq\| \\
&= (\|\sim p\| \cap \|\sim q\|) \cup (\|p\| \cap \|q\|) \\
&= ((\mathcal{T} - \|p\|) \cap (\mathcal{T} - \|q\|)) \cup (\|p\| \cup \|q\|) \\
&= ((\mathcal{T} - \|q\|) \cap (\mathcal{T} - \|q\|)) \cup (\|q\| \cup \|q\|) \text{ since } \|p\| = \|q\| \\
&= (\mathcal{T} - \|q\|) \cup \|q\| \\
&= \mathcal{T} \\
&= \|1\|
\end{aligned}$$

Then, $\vdash p = q \Rightarrow \models \sim p \sim q \vee pq = 1 \Rightarrow \vdash \sim p \sim q \vee pq = 1$ using the derivability results already obtained. Now consider the following two derivations

$pq = pq \vee 0$	S9
$= 0 \vee pq$	S3
$= \sim p0 \vee pq$	S8
$= \sim p \sim qq \vee pqq$	C1 and C2
$= (\sim p \sim q \vee pq)q$	S5
$= 1\, q$	since $\vdash \sim p \sim q \vee pq = 1$
$= q$	P4

$pq = pq \vee 0$	S9
$= 0 \vee pq$	S3
$= \sim q\, 0 \vee pq$	S8
$= \sim q \sim pp \vee pqp$	C1, C2 and C3
$= (\sim p \sim q \vee pq)p$	S5 and C3
$= 1p$	since $\vdash \sim p \sim q \vee pq = 1$
$= p$	P4

Then, since $\vdash pq = pq$ by T1 and T2, we use T2 again to substitute $\vdash pq = p$ and $\vdash pq = q$ and so obtain $\vdash p = q$. Thus, $\vdash p = q \Rightarrow \vdash p = q$. ∎

Before we discuss the adequacy of $\mathcal{T}_K$ in general, we need two further results (Theorems 31 and 32). Recall that $\vee S$ denotes the conjunction formed with "$\vee$" of the formal objects in the set S, and $\vee\emptyset = 0$.

Theorem 31: For any proposition $r \in \mathcal{P}$, $\vdash r = \vee\|r\|$.

This corresponds to an existence theorem for the full disjunctive normal form of the propositional calculus.

Proof: First let us extend the result of Theorem 29 and show that $\vdash p = q \Leftrightarrow \vdash p = q$ for arbitrary K-expressions $p, q \in \mathcal{P}^+$ where

(i) $\mathcal{P} \subseteq \mathcal{P}^+$

(ii) if $\alpha, \beta \in \mathcal{P}^+$ then $(\alpha \vee \beta) \in \mathcal{P}^+$ and $(\alpha\beta) \in \mathcal{P}^+$

(iii) extremal clause.

Thus, we want to extend the adequacy statement of Theorem 29 to the closure of $\mathcal{P}$ over "$\vee$" and "." . The entities of $\mathcal{P}^+$ are still propositional in nature since "$\vee$" behaves as disjunction and "." as conjunction.

Consider the following derivations:

$$\left.\begin{array}{lll} rs = r \sim\sim s & & \\ \quad = \sim(r \supset \sim s) & C6 & \\ & & \\ r \vee s = \sim\sim r \vee s & C7 & \\ \quad = (\sim r \supset s) & C5 & \end{array}\right\} \quad r, s \in \mathcal{P}$$

Using these derivations, we can convert any $t \in \mathcal{P}^+$ into a proposition $t' \in \mathcal{P}$, such that $\vdash t = t'$. This follows from a simple inductive argument over the structure of $t \in \mathcal{P}^+$. Then for $p, q \in \mathcal{P}^+$,

$\vdash p = q \Rightarrow \vdash p' = q'$ by the above argument

$\Rightarrow \vdash p' = q'$ Theorem 29

$\Rightarrow \vdash p = q$ by the above derivations.

Then, to show $\vdash r = \vee\|r\|$ as required in the statement of Theorem 31, we first show $\vdash r = \vee\|r\|$ and then apply the extended adequacy result just obtained. Notice that for a word of truth $v = v_0\, v_1 \cdots v_{m-1}$,

$$\begin{aligned}\|v\| &= \|v_0\, v_1 \cdots v_{m-1}\| \\ &= \cap_{i<m} \|v_i\| \\ &= \cap_{i<m} \{s_0\, s_1 \cdots s_{m-1} \in \mathcal{T}: s_i = v_i\} \\ &= \{v_0\, v_1 \cdots v_{m-1}\} \\ &= \{v\}\ .\end{aligned}$$

Then to show $\vdash r = \vee \|r\|$, we have

$$\begin{aligned}\|\vee\|r\|\ \| &= \cup_{v \in \|r\|} \|v\| \\ &= \cup_{v \in \|r\|} \{v\} \\ &= \|r\| \qquad \text{as required.}\end{aligned}$$

Finally, $\vdash r = \vee \|r\| \Rightarrow \vdash r = \vee \|r\|$ from the extended adequacy result obtained above. ∎

In the proof of adequacy for $\mathcal{T}_K$, we make use of the notion of <u>standard</u> K-expressions. A K-expression $\alpha \in \mathcal{K}$ is said to be standard iff it is of the form

$$p_0 \vee \ldots \vee p_{k-1} \vee s_0 x_0 t_0 \vee \ldots \vee s_{\ell-1} x_{\ell-1} t_{\ell-1}$$

where $p_0, \ldots, p_{k-1}, s_0, t_0, \ldots, s_{\ell-1}, t_{\ell-1} \in \mathcal{T}$, $x_0, \ldots, x_{\ell-1} \in \mathcal{K}$, and $k, \ell < \omega$, $k + \ell < \omega^-$. We denote by $\mathcal{K}^-$ the set of all standard K-expressions in $\mathcal{K}$.

We shall make use of the syntactic operation $\otimes : \mathcal{K}^- \times \mathcal{K}^- \to \mathcal{K}^-$ which allows us to "multiply out" two standard K-expressions. If $\alpha = \vee A$, $\beta = \vee B$ and $\alpha, \beta \in \mathcal{K}^-$, then

$$\alpha \otimes \beta = \vee\{xpy : xp \in A \ \& \ py \in B \ \& \ p \in \mathcal{T}\} .$$

As an illustrative example we have

$$(s_1 \vee s_2 \vee s_1 x s_2) \otimes (s_1 \vee s_2 y s_3 \vee s_1 z s_1) = (s_1 \vee s_1 z s_1 \vee s_2 y s_3 \vee s_1 x s_2 y s_3)$$

where $s_1, s_2, s_3 \in \mathcal{T}$ and $x, y, z \in \mathcal{H}$. Concerning the idea of standard K-expression and the $\otimes$ operator, we have the following

Theorem 32: For any standard K-expressions α and β, $\vdash \alpha\beta = \alpha \otimes \beta$.

Proof: First notice that for any distinct $u, v \in \mathcal{T}$,

$$\begin{aligned} \|uv\| &= \{s : s \in \|u\| \ \& \ s \in \|v\| \ \& \ s \in \mathcal{T}\} \\ &= \{s : s \in \{u\} \ \& \ s \in \{v\} \ \& \ s \in \mathcal{T}\} \\ &= \emptyset \end{aligned}$$

and

$$\begin{aligned} \|vv\| &= \{s : s \in \|v\| \ \& \ s \in \|v\| \ \& \ s \in \mathcal{T}\} \\ &= \|v\| \end{aligned}$$

Thus, $\vdash uv = 0$ and $\vdash vv = v$. This result together with repeated application of S4, by which "." is seen to distribute over "$\vee$", and S9, by which any extraneous 0 is dropped, yields the desired derivation of $\alpha \otimes \beta$. ∎

Also used in the proof of adequacy for $\mathcal{T}_K$ is the notion of normal form K-expressions. For any $\gamma \in \mathcal{K}$, the K-expression $N(\gamma)$ is said to be a normal form of γ iff $N(\gamma)$ is a regular expression over the alphabet $\mathcal{H} \cup \mathcal{T}$ and $\|\gamma\| = \|N(\gamma)\| = |N(\gamma)|$.

Since the structure of normal form K-expressions plays a key role in the sequel, let us consider some examples which make this structure clear. Suppose that $\mathcal{Atm} = \{p, q\}$ and $\mathcal{V} = \{x\}$; then $\mathcal{T} = \{pq, p\bar{q}, \bar{p}q, \bar{p}\bar{q}\}$. Below is a list of K-expressions and a possible normal form for each.

K-expression α	A Normal Form $N(\alpha)$
x	$(pq \vee p\bar{q} \vee \bar{p}q \vee \bar{p}\bar{q})\ x\ (pq \vee p\bar{q} \vee \bar{p}q \vee \bar{p}\bar{q})$
$(p \supset q)$	$\bar{p}q \vee \bar{p}\bar{q} \vee pq$
$\sim p$	$\bar{p}q \vee \bar{p}\bar{q}$
$\sim(p \supset \sim q)x$	$pqx(pq \vee p\bar{q} \vee \bar{p}q \vee \bar{p}\bar{q})$
$(\sim(p \supset \sim q)x)^*$	$(pqx)^*(pq \vee p\bar{q} \vee \bar{p}q \vee \bar{p}\bar{q})$

We are now ready to discuss the overall adequacy of $\mathcal{T}_K$, and we state the main result in the following

Theorem 33: For any K-expressions $\alpha, \beta \in \mathcal{K}$, $\models \alpha = \beta \Rightarrow \vdash \alpha = \beta$.

Proof: Let us first give a sketch of the proof.

We first prove a Normal Form Theorem (Theorem 34) which states that $\vdash \gamma = N(\gamma)$ for any K-expression γ and some normal form $N(\gamma)$.

Adequacy follows immediately: $\models \alpha = \beta \Rightarrow \|\alpha\| = \|\beta\| \Rightarrow$ $\|N(\alpha)\| = \|N(\beta)\| \Rightarrow |N(\alpha)| = |N(\beta)|$, and since Salomaa's system is adequate, $|N(\alpha)| = |N(\beta)| \Rightarrow \vdash N(\alpha) = N(\beta)$. The normal form theorem then gives $\vdash N(\alpha) = N(\beta) \Rightarrow \vdash \alpha = \beta$, so that altogether we have $\models \alpha = \beta \Rightarrow \vdash \alpha = \beta$ as required.

The first step, then, is the following

Theorem 34: (Normal Form Theorem): For any K-expression α , $\vdash \alpha = N(\alpha)$, where $N(\alpha)$ is a normal form of α .

Proof: The proof is inductive over the structure of α. We define a **simple** K-expression as one in which no "*" occurs, and let $\mathcal{K}^+ \subseteq \mathcal{K}$ be the set of all simple K-expressions. Thus $\mathcal{K}^+$ is the closure of $\mathcal{P} \cup \mathcal{G}$ over "$\vee$" and "$\cdot$". We prove the result first for $\alpha \in \mathcal{K}^+$ (again by induction on the structure of α) and then for $\alpha = \beta^*$, where $\vdash \beta = N(\beta)$ is our induction hypothesis.

Suppose α is simple; there are four cases to consider.

(i) α is an operator $g_i \in \mathcal{G}$. Then we take

$$N(\alpha) = N(g_i) = \vee\{pg_iq : p, q \in \mathcal{T}\} = \vee\|g_i\| .$$

We have $\vdash g_i = N(g_i)$ as follows:

$g_i = g_i$	T1, T2
$= 1g_i1$	S7, P4
$= \vee\mathcal{T} \cdot g_i \cdot \vee\mathcal{T}$	Theorem 31
$= \vee\{pg_iq : p, q \in \mathcal{T}\}$	S4, S5
$= N(g_i)$	

(ii) α is a proposition $p \in \mathcal{P}$. Then we take

$$N(\alpha) = N(p) = \vee\|p\|$$

and $\vdash p = N(p)$ by Theorem 31.

(iii) α is $\beta \vee \gamma$. Then we take

$$N(\alpha) = N(\beta \vee \gamma) = N(\beta) \vee N(\gamma) .$$

By induction hypothesis, $\vdash \beta = N(\beta)$ and $\vdash \gamma = N(\gamma)$, so that $\vdash \beta \vee \gamma = N(\beta \vee \gamma)$ as required.

(iv) α is $\beta\gamma$. Then we take

$$N(\alpha) = N(\beta\gamma) = N(\beta) \otimes N(\gamma) .$$

The "⊛" is permissible since evidently the normal forms we generate for simple K-expressions are themselves standard K-expressions. By induction hypothesis, $\vdash \beta = N(\beta)$ and $\vdash \gamma = N(\gamma)$, so that $\vdash \beta\gamma = N(\beta)\,N(\gamma)$ by R2. Then Theorem 32 gives $\vdash N(\beta)\,N(\gamma) = N(\beta) \circledast N(\beta)$, so that finally, $\vdash \beta\gamma = N(\beta\gamma)$. This completes the induction for the case of simple α.

So far we have shown that $\vdash \alpha = N(\alpha)$ where α is simple, but to verify that $N(\alpha)$ is indeed a normal form of α, we must also show that $\|N(\alpha)\| = |N(\alpha)|$, where the regular event $|N(\alpha)|$ is evaluated considering $N(\alpha)$ as a regular expression over the alphabet $\mathcal{B} \cup \mathcal{T}$. It is straightforward to show from the definitions in (i) through (iv) above that $N(\alpha)$ is in fact a regular expression over $\mathcal{B} \cup \mathcal{T}$. We now prove, again by induction over the structure of α, that $\|N(\alpha)\| = |N(\alpha)|$. There are four cases to consider.

(i) α is an operator $g_i \in \mathcal{G}$ and $N(\alpha) = \vee\{pg_iq : p, q \in \mathcal{T}\}$. Then,

$= \| \vee\{pg_iq : p, q \in \mathcal{T}\}\|$

$= \cup_{p,\ q \in \mathcal{T}} \|pg_iq\|$

$= \cup_{p,\ q \in \mathcal{T}} \{rx : r \in \|p\| \ \&\ rx \in \|g_iq\| \ \&\ r \in \mathcal{T}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{rx : r \in \{p\} \ \&\ rx \in \{ut : ut \in \|g_i\| \ \&\ t \in \|q\| \ \&\ t \in \mathcal{T}\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{rx : r = p \ \&\ rx \in \{ut : ut \in \{mg_in : m, n \in \mathcal{T}\} \ \&\ t \in \{q\}\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{px : px \in \{ut : ut \in \{mg_in : m, n \in \mathcal{T}\} \ \&\ t = q\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{px : px \in \{uq : uq \in \{mg_in : m, n \in \mathcal{T}\}\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{px : px \in \{mg_iq : m, q \in \mathcal{T}\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{pg_i q\}$

$= \{pg_i q : p,\ q \in \mathcal{T}\}$,

and,

$|N(\alpha)|$

$= |\vee\{pg_i q : p,\ q \in \mathcal{T}\}|$

$= \cup_{p,\ q \in \mathcal{T}} |pg_i q|$

$= \cup_{p,\ q \in \mathcal{T}} \{uv : u \in |p| \ \&\ v \subset |g_i q|\}$

$= \cup_{p,\ q \in \mathcal{T}} \{uv : u \in \{p\} \ \&\ v \in \{xy : x \in |g_i| \ \&\ y \in |q|\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{uv : u = p \ \&\ v \subset \{xy : x \in \{g_i\} \ \&\ y \in \{q\}\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{pv : v \in \{xy : x = g_i \ \&\ y = q\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{pv : v \in \{g_i q\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{pv : v = g_i q\}\}$

$= \cup_{p,\ q \in \mathcal{T}} \{pg_i q\}$

$= \{pg_i q : p,\ q \in \mathcal{T}\}$,

so that $\|N(\alpha)\| = |N(\alpha)|$ in case α is $g_i \in \mathcal{G}$.

(ii) α is a proposition $p \in \mathcal{P}$ and $N(\alpha) = \vee\|p\|$. Then,

$\|N(\alpha)\| = \|\vee\|p\|\ \|$

$= \|p\|$ by Theorem 31,

and,

$|N(\alpha)| = |\vee\|p\|\ |$

$= \cup_{r \in \|p\|} |r|$

$= \cup_{r \in \|p\|} \{r\}$ since $r \in \mathcal{T}$

$= \|p\|$,

so that $\|N(\alpha)\| = |N(\alpha)|$ in case α is $p \in \mathcal{P}$.

(iii) α is $\beta \vee \gamma$ and $N(\alpha) = N(\beta) \vee N(\gamma)$. Then,

$\|N(\alpha)\| = \|N(\beta) \vee N(\gamma)\|$

$= \|N(\beta)\| \cup \|N(\gamma)\|$

$= |N(\beta)| \cup |N(\gamma)|$ by induction hypothesis

$= |N(\beta) \vee N(\gamma)|$

$= |N(\alpha)|$ as required.

(iv) α is $\beta\gamma$ and $N(\alpha) = N(\beta) \circledast N(\gamma)$. Then,

$$\|N(\alpha)\| = \|N(\beta) \circledast N(\gamma)\|$$

$$= \|\vee\{xpy : xp \in B \ \& \ py \in C \ \& \ p \in \mathcal{T}\}\| ,$$

where $N(\beta) = \vee B$ and $N(\gamma) = \vee C$. By induction hypothesis, $\|N(\beta)\| = |N(\beta)|$, i.e., $\|\vee B\| = |\vee B|$, and from this we can easily show that $\|xp\| = |xp|$ for all $xp \in B$. Similarly, from the induction hypothesis $\|N(\gamma)\| = |N(\gamma)|$, we obtain $\|py\| = |py|$ for all $py \in C$. Using the semantics of "." , this last result can be restated as $\{pz : pz \in \|y\|\} = \{pz : z \in |y|\}$, so that for any v , $pv \in \|y\| \Leftrightarrow v \in |y|$. To complete the evaluation of $N(\alpha)$ started above, notice that

$$\|xp.y\| = \{upv : up \in \|xp\| \ \& \ pv \in \|y\|\}$$

$$= \{upv : up \in |xp| \ \& \ v \in |y|\}, \text{ above results}$$

$$= |xp.y| .$$

Then,

$$\|N(\alpha)\| = \|\vee\{xpy : xp \in B \ \& \ py \in C \ \& \ p \in \mathcal{T}\}\|$$

$$= \cup_{xpy : xp \in B \ \& \ py \in C \ \& \ p \in \mathcal{T}} \|xpy\|$$

$$= \cup_{xpy : xp \in B \ \& \ py \in C \ \& \ p \in \mathcal{T}} |xpy|, \text{ above result}$$

$$= |\vee\{xpy : xp \in B \ \& \ py \in C \ \& \ p \in \mathcal{T}\}|$$

$$= |N(\alpha)| ,$$

as required. This completes the induction.

Thus, so far, we have shown that for any simple K-expression $\alpha \in \mathcal{K}^*$, $\vdash \alpha = N(\alpha)$ where $N(\alpha)$ is a normal (and standard) form

of α. Now, let us complete the proof of Theorem 34, by showing that $\vdash \beta^* = N(\beta^*)$ where by induction hypothesis, $\vdash \beta = N(\beta)$.

Since $\vdash \beta = N(\beta)$, then $\vdash \beta^* = N(\beta)^*$, and since $N(\beta)$ is a standard K-expression of the form $a_0 \vee a_1 \vee \ldots \vee a_{m-1}$, $m < \omega$, then $N(\beta)^*$ is of the form $(a_0 \vee a_1 \vee \ldots \vee a_{m-1})^*$. Now, if θ_0 is a_0^* and θ_n is $(a_n\theta_0\theta_1 \ldots \theta_{n-1})^*$, $0 < n < m$, then

$\vdash N(\beta)^* = \theta_0\theta_1 \ldots \theta_{m-1}$.

This follows simply if we recall that θ is regular over $\mathcal{U} \cup \mathcal{T}$ and then apply P1: $(\alpha \vee \beta)^* = \alpha^*(\beta\alpha^*)^*$ repeatedly. To see the form of $\theta_0\theta_1 \ldots \theta_{m-1}$, consider the following example of the above result:

$\vdash (a \vee b \vee c \vee d)^* = a^* (ba^*)^*(ca^*(ba^*)^*)^*(da^*(ba^*)^*(ca^*(ba^*)^*)^*)^*$.

Here, $M = 4$ and,

$N(\beta)^* = (a \vee b \vee c \vee d)^*$

θ_0 is a^*

θ_1 is $(ba^*)^*$

θ_2 is $(ca^*(ba^*)^*)^*$

θ_3 is $(da^*(ba^*)^*(ca^*(ba^*)^*)^*)^*$.

Since $\vdash \beta^* = N(\beta)^*$ and $\vdash N(\beta)^* = \theta_0\theta_1 \ldots \theta_{m-1}$, we then have $\vdash \beta^* = \theta_0\theta_1 \ldots \theta_{m-1}$ using T2. Now, for the moment, let us assume that we can show

$\vdash \theta_0 = N(\theta_0)$, $\vdash \theta_1 = N(\theta_1)$, ..., $\vdash \theta_{m-1} = N(\theta_{m-1})$.

Then, using T2 again, we have $\vdash \beta^* = N(\theta_0)N(\theta_1) \ldots N(\theta_{m-1})$. Theorem 31 then yields $\vdash \beta^* = N(\theta_0) \otimes N(\theta_1) \otimes \ldots \otimes N(\theta_{m-1})$. By an argument precisely like that given for the case of simple α of the form $\beta\gamma$, we have that

$\|N(\theta_0) \otimes N(\theta_1) \otimes \ldots \otimes N(\theta_{m-1})\| = |N(\theta_0) \otimes N(\theta_1) \otimes \ldots \otimes N(\theta_{m-1})|$

and furthermore, since $\mathcal{T}_K$ is sound, we have

$$\|\beta^*\| = \|N(\theta_0) \oplus N(\theta_1) \oplus \ldots \oplus N(\theta_{m-1})\|$$

Thus, by the definition of normal form, we can take $N(\beta^*)$ to be $N(\theta_0) \oplus N(\theta_1) \oplus \ldots \oplus N(\theta_{m-1})$. What remains for us to show is that

$$\vdash \theta_0 = N(\theta_0),\ \vdash \theta_1 = N(\theta_1), \ldots,\ \vdash \theta_{m-1} = N(\theta_{m-1}) .$$

We will show inductively how to carry out these derivations. First we show $\vdash \theta_0 = N(\theta_0)$, and then assuming that we have $\vdash \theta_0 = N(\theta_0), \ldots, \vdash \theta_{n-1} = N(\theta_{n-1})$, we show $\vdash \theta_n = N(\theta_n)$, $n < m$.

Since θ_0 is a_0 where $N(\beta) = (a_0 \vee a_1 \vee \ldots \vee a_{m-1})$, there are three possibilities to consider in light of the fact that $(a_0 \vee a_1 \vee \ldots \vee a_{m-1})$ is a standard K-expression.

(i) a_0 is sxs
(ii) a_0 is sxt
(iii) a_0 is s

} for some $s, t \in \mathcal{T}$

We consider derivations P5, P6 and P7 for each case in turn

P5: $(sxs)^* = 1 \vee sxs(sxs)^*$ S10
$= 1 \vee sx(sxs)^*s$ P2
$= 1 \vee sx(sx)^*s$ C1
$= \vee\mathcal{T} \vee sx(sx)^*s$ Theorem 31

P6: $(sxt)^* = 1 \vee sxt(sxt)^*$ S10
$= 1 \vee sxt(1 \vee sxt(sxt)^*)$ S10
$= 1 \vee sxt\ 1 \vee sxtsxt(sxt)^*$ S4
$= 1 \vee sxt \vee sx0xt(sxt)^*$ Theorem 32
$= 1 \vee sxt$ S8, S9
$= \vee\mathcal{T} \vee sxt$ Theorem 31

F1: First we show $\vdash 1 = 1^*$.

$1 = 1 \vee 0$	S9
$= 1 \vee 01$	S7
$= 0^*1$	T3
$= 0^*$	S7
$= (1 \vee 0)^*$	S10
$= (0 \vee 1)^*$	S2
$= 0^*(10^*)^*$	P1
$= 0^*(0^*)^*$	P4
$= 1\,1^*$	Since $\vdash 1 = 0^*$
$= 1^*$	P4

Then we have,

$s^* = (1 \vee s)^*$	S11
$= (\vee\mathcal{T} \vee s)^*$	Theorem 31
$= (\vee\mathcal{T})^*$	S6
$= 1^*$	Theorem 31
$= 1$	Since $\vdash 1 = 1^*$
$= \vee\mathcal{T}$	Theorem 31

It is a straightforward matter to show that because $(a_0 \vee a_1 \vee \ldots \vee a_{m-1})$ is a normal form then so is a_0, i.e., $\|a_0\| = |a_0|$, where a_0 is regular over $\mathcal{U} \cup \mathcal{T}$. From this we can easily obtain that $\vee\mathcal{T} \vee sx(sx)^*s$, $\vee\mathcal{T} \vee sxt$, or $\vee\mathcal{T}$, as the case may be, is a normal form for a_0^* , i.e., for θ_0 . Furthermore, this normal form, which we may now write as $N(\theta_0)$ is also a standard form, as is required of all normal forms used in the proof of this theorem.

To complete the construction, we show how from $\vdash \theta_0 = N(\theta_0)$, $\vdash \theta_1 = N(\theta_1), \ldots,$ $\vdash \theta_{n-1} = N(\theta_{n-1})$, $n < m$, we have

$\vdash \theta_n = N(\theta_n)$. Recall that θ_n is $(a_n\theta_o\theta_1 \ldots \theta_{n-1})^*$, so that

$$\vdash \theta_n = (a_n\theta_o\theta_1 \ldots \theta_{n-1})^*$$

by T1 and T2 . Since the normal forms $N(\theta_o),\ldots, N(\theta_{n-1})$ have been assumed derivable, we have

$$\vdash \theta_n = (a_n N(\theta_o)N(\theta_1) \ldots N(\theta_{n-1}))^* .$$

Then, applying Theorem 32,

$$\vdash \theta_n = (a_n \otimes N(\theta_o) \otimes N(\theta_1) \otimes \ldots \otimes N(\theta_{n-1}))^* .$$

Because a_n is a single disjunct of the form sxs , sxt or s , for some $s, t \in \mathcal{T}$, then $a_n \otimes N(\theta_o) \otimes N(\theta_1) \otimes \ldots \otimes N(\theta_{n-1})$ must be of the form

$$s \vee sx_o s \vee \ldots \vee sx_{k-1}s \vee sy_o t_o \vee \ldots \vee sy_{\ell-1}t_{\ell-1}$$

where the leading $s \in \mathcal{T}$ may be absent, and where $x_o,\ldots, x_{k-1}, y_o,\ldots, y_{\ell-1} \in \mathcal{K}$, $k, \ell < \omega$, $t_o,\ldots, t_{\ell-1} \in \mathcal{T}$, $\ell < \omega$, and are distinct from s . If all the disjuncts are absent, we have simply $\vdash \theta_n = 0^*$, and we can take $N(\theta_n)$ as $\vee\mathcal{T}$. In any case, we can easily generate a normal form here by making use of the fact that the leftmost words of truth in the disjuncts of the expression are the same, namely s .

First, let us show that $\vdash (s \vee \alpha)^* = \alpha^*$ for any $s \in \mathcal{T}$ and $\alpha \in \mathcal{K}$.

$(s \vee \alpha)^* = (1 \vee s \vee \alpha)^*$	S11
$= (\vee\mathcal{T} \vee s \vee \alpha)^*$	Theorem 31
$= (\vee\mathcal{T} \vee \alpha)^*$	S6
$= (1 \vee \alpha)^*$	Theorem 31
$= \alpha^*$	S11

Thus, it suffices to consider finding a normal form for $sXs \vee sy_0t_0 \vee \ldots \vee sy_{\ell-1}t_{\ell-1}$, where $X = \vee\{x_i\}_{i<k}$, i.e., we may ignore the leading s even if it is present. To add further brevity, we rewrite this as $sXs \vee \vee\{sy_it_i\}_{i<\ell}$. Now, consider the following derivation.

$(sXs \vee \vee\{sy_it_i\}_{i<\ell})^*$

$= (sXs)^*(\vee\{sy_it_i\}_{i<\ell}(sXs)^*)^*$ P1

$= (sXs)^*(\vee\{sy_it_i\}_{i<\ell}(1 \vee sX(sX)^*s))^*$ P5

$= (sXs)^*(\vee\{sy_it_i\}_{i<\ell} \vee \vee\{sy_it_i\}_{i<\ell}sX(sX)^*s)^*$ S4

$= (sXs)^*(\vee\{sy_it_i\}_{i<\ell} \vee 0)^*$ Theorem 32

$= (sXs)^*(\vee\{sy_it_i\}_{i<\ell})^*$ S9

$= (sXs)^*(sy_0t_0)^*(\vee\{sy_it_i\}_{i<\ell-1})^*$ P1, P5, S4, Theorem 32, S9

$= (sXs)^*(sy_0t_0)^*(sy_1t_1)^* \ldots (sy_{\ell-1}t_{\ell-1})^*$ P1, P5, S4, Theorem 32, S9

$= (1 \vee sX(sX)^*s)(1 \vee sy_0t_0) \ldots (1 \vee sy_{\ell-1}t_{\ell-1})$ P5, P6

$= \vee\mathcal{T} \vee \vee\{sy_it_i\}_{i<\ell} \vee sX(sX)^*s \vee \vee\{sX(sx)^*sy_it_i\}_{i<\ell}$ Theorems 31, 32

This last K-expression is both normal and standard and will serve as the required $N(\theta_n)$.

We conclude, therefore, that $\vdash\theta_0 = N(\theta_0)$, $\vdash\theta_1 = N(\theta_1), \ldots,$ $\vdash\theta_{m-1} = N(\theta_{m-1})$. Then, as we have already shown, $N(\beta^*)$ can be taken as $N(\theta_0) \otimes \ldots \otimes N(\theta_{m-1})$, and furthermore, $\vdash\beta^* = N(\beta^*)$. ∎

The normal form theorem (Theorem 34) we have just proven, immediately leads to a proof of Theorem 33, which expresses the adequacy of $\mathcal{T}_K$.

The hypothesis of Theorem 33 is that $\vdash\alpha = \beta$ for arbitrary K-expressions $\alpha,\beta \in \mathcal{K}$. Then,

$\vdash\alpha = \beta \Rightarrow \|\alpha\| = \|\beta\|$ by definition.

$\Rightarrow \|N(\alpha)\| = \|N(\beta)\|$ where $N(\alpha)$ and $N(\beta)$ are the normal forms provided by Theorem 33,

$\Rightarrow |N(\alpha)| = |N(\beta)|$ property of normal forms,

$\Rightarrow \vdash N(\alpha) = N(\beta)$ using only Salomaa's system which is adequate.

Then, since $\vdash\alpha = N(\alpha)$ and $\vdash\beta = N(\beta)$ by the normal form theorem, we have finally, $\vdash\alpha = \beta$. Thus, $\vdash\alpha = \beta \Rightarrow \vdash\alpha = \beta$, as required. ∎

Remarks:

(i) The imbedding of $X = \vee\{x_i\}_{i<k}$ in the construction of $N(\theta_n)$ implies that an arbitrary K-expression α cannot in general be expressed in a purely disjunctive normal form, i.e., one where no "$\vee$" appears inside of "$(\ )^*$".

(ii) Since the proof of Theorem 33 is constructive, $?\vdash\alpha = \beta?$ for an arbitrary wff $\alpha = \beta \in \mathcal{F}_K$ is therefore effectively decidable. Thus, for any two K-expressions α and β, we can say whether or not they are K-equivalent, and if they are, then we can produce a proof in $\mathcal{T}_K$ of this fact.

(iii) Theorem 27 tells us that for any two E-programs $\mathfrak{A}$ and $\mathfrak{B}$ of the same type, $\vdash\alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}} \Rightarrow \vdash\mathfrak{A} \simeq \mathfrak{B}$. Since $?\vdash\alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}?$ is decidable, this means we have a test for strong equivalence which in part answers the question $?\vdash\mathfrak{A} \simeq \mathfrak{B}?$. Thus, by testing $\alpha_{\mathfrak{A}}$ and $\alpha_{\mathfrak{B}}$ for K-equivalence, we either obtain "yes" or "maybe" to the question: is $\mathfrak{A}$ strongly equivalent to $\mathfrak{B}$.

(iv) Even though the K-event formulation and proof of completeness given here (Theorems 28 and 33) are new and independently obtained, the completeness result is not new. In fact, despite the K-event formulation, this result actually includes that of Ianov [16] and Rutledge [37], and is apparently equivalent to that of Ito [18]. The completeness results of Ianov, Rutledge and Ito, and the work by Ito in the recasting of regular expression semantics, are among the factors that motivated this author's quest for the here presented elegant formalism and concise completeness proof, which together constitute a measurable improvement over those earlier works.

K-events and Ianov's Results

We shall not embroil ourselves here in a dissection of Ianov's work. Rather, we shall define for any signature $s = \langle\langle n_0, \ldots, n_{k-1}\rangle, \langle m_0, \ldots, m_{\ell-1}\rangle, p\rangle$ sub-class $A_s \subseteq L_s$ of abstract E-programs such that the properties of strong equivalence and K-equivalence are identical. This in the sense that for any abstract E-programs $\mathfrak{M}, \mathfrak{N} \in A_s$, $\vdash_{\mathfrak{M}} \simeq \mathfrak{N} \Leftrightarrow \vdash_{\alpha_{\mathfrak{M}}} = \alpha_{\mathfrak{N}}$. It will then be evident that abstract E-programs are like Ianov's program schemata, except that we allow more than one entrance and exit, and repeated occurrences of operators, i.e., assignment schemata.

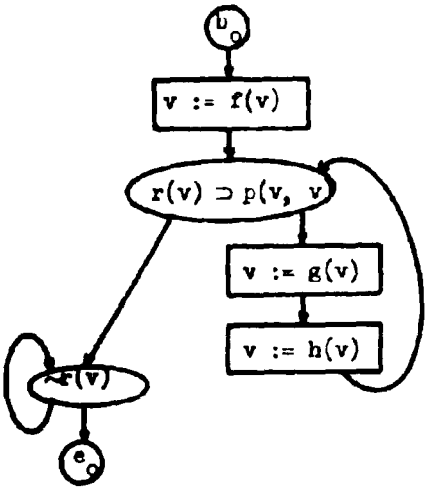
An E-program $\mathfrak{M}$ is said to be abstract iff

(i) v_0 is the only variable occurring in $\mathfrak{M}$ (we will write simply "v"),

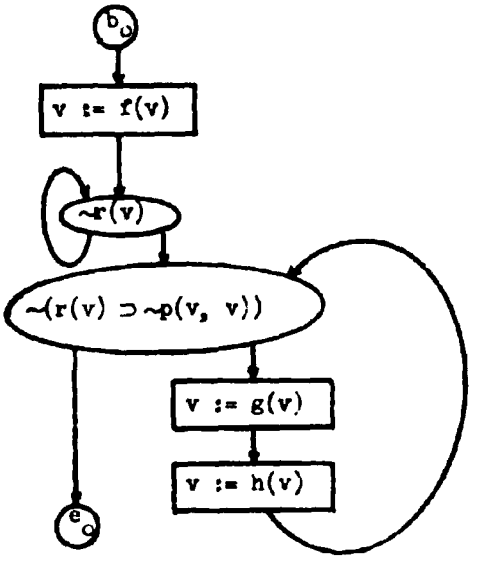
(ii) no constants occur in $\mathfrak{M}$.

(iii) no function letters occur in any qff occurring in $\mathfrak{M}$,

(iv) all assignment schemata occurring in $\mathfrak{M}$ are of the form $v := f_j(\tau_0, \ldots, \tau_{m_j-1})$ where $j < \ell$, and each one of $\tau_0, \ldots, \tau_{m_j-1}$ is just the variable v.



Abstract E-program $\mathfrak{A}$



Abstract E-program $\mathfrak{B}$

Figure 50

Two strongly equivalent abstract E-programs. Here, f, g, h are functional letters; p, r are relation letters; v is the variable v_0.

Figure 50 illustrates [illegible] $\mathfrak{A}$ [illegible] $\mathfrak{B}$, and evidently $\models \mathfrak{A} = \mathfrak{B}$.

Theorem 35: For any two [illegible], $\mathfrak{A}$, $\mathfrak{B}$ [illegible],

$$\models \mathfrak{A} = \mathfrak{B} \Leftrightarrow \models \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$$

Proof: The case $\models \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}} \Rightarrow \models \mathfrak{A} = \mathfrak{B}$ [illegible] immediately from Theorem 27. For the other case, we need the [illegible]

(i) For any computing structure $\underset{\sim}{X}$ of signature s, state ξ and $i < m$ (where $\mathfrak{A}$ and $\mathfrak{B}$ are [illegible]), if $\mathfrak{A}[\underset{\sim}{X}, \langle \xi, i \rangle]$ [illegible] and so also $\mathfrak{B}[\underset{\sim}{X}, \langle \xi, i \rangle]$ [illegible] in other words $u \in \|\alpha_{\mathfrak{A}}\|$ and $w \in \|\alpha_{\mathfrak{B}}\|$ respectively, then [illegible].

(ii) For any abstract E-[illegible] Σ of type $\langle m, n \rangle$, if $W_{\Sigma} = \{sb_i s' x t e_j t' \in \|\alpha_{\Sigma}\| : [illegible] \ \& \ t = t'\}$, then for any $u \in W_{\Sigma}$, there exists a computing structure $\underset{\sim}{X}$ of signature s, state η and $K < m$ such that $\mathfrak{A}[\underset{\sim}{X}, \langle \eta, K \rangle]$ is determinate and produces the word u.

(iii) $W_{\mathfrak{A}} = W_{\mathfrak{B}} \Rightarrow \|\alpha_{\mathfrak{A}}\| = \|\alpha_{\mathfrak{B}}\|$

From these results (which we [illegible] below), the desired result follows immediately. Consider any $u \in W_{\mathfrak{A}}$. By (iii), $\mathfrak{A}[\underset{\sim}{X}, \langle \eta, K \rangle]$ generates u; and by (i) $\mathfrak{B}[\underset{\sim}{X}, \langle \eta, K \rangle]$ also generates u, so that $u \in W_{\mathfrak{B}}$ as well. Thus, $W_{\mathfrak{A}} \subseteq W_{\mathfrak{B}}$. A similar argument gives $W_{\mathfrak{B}} \subseteq W_{\mathfrak{A}}$, so that finally $W_{\mathfrak{A}} = W_{\mathfrak{B}}$. Then, (iii) gives $\|\alpha_{\mathfrak{A}}\| = \|\alpha_{\mathfrak{B}}\|$, i.e., $\models \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$, as required.

To obtain result (i) above, we need not give a detailed proof like that given for Theorem 27. Let us simply note that two sequences of assignment schemata of the sort found in abstract E-programs are not strongly equivalent

unless they are syntactically identical. Thus, in the words $u \in \|\alpha_{\mathfrak{A}}\|$ and $w \in \|\alpha_{\mathfrak{B}}\|$, the operator letters that are assignment schemata in each are equinumerous, identical, and appear in the same order. Since the atomic formulas in two words of truth cannot all be true if evaluated in a state, the words of truth following each operator letter in u and w must therefore be identical. This, because the states at corresponding points in the execution of $\mathfrak{A}$ and $\mathfrak{B}$ are identical as a result of identical sequences of assignment schemata having been executed on identical initial states. Furthermore $\mathfrak{A}$ and $\mathfrak{B}$ start at the same initiator and halt at the same terminator (since they are strongly equivalent), so that the initial and final operator letters in u and w are also identical. Thus, $u = w$.

To obtain result (ii) above, we actually show for any word $u \in W_{\mathfrak{S}}$ how to specify the required computing structure $\underset{\sim}{X} = \langle X, R_0, \ldots, R_{k-1}, F_0, \ldots, F_{\ell-1}, a_0, \ldots, a_{p-1} \rangle$, state η and $K < m$. Let the domain X be the set of all terms in which no constants and only the variable v occur. For the functions, we let $F_j(c_0, \ldots, c_{m_j-1}) = f_j(c_0, \ldots, c_{m_j-1})$, $j < \ell$, where $c_0, \ldots, c_{m_j-1} \in X$, i.e., are terms. The constants $a_0, \ldots, a_{p-1}$ can be chosen arbitrarily, of course. The relations will be specified below by examining the word $u \in W_{\mathfrak{S}}$ to determine what the branching through $\mathfrak{S}$ must be during execution so that u is generated.

For the initial state η, we simply specify that $c(0, \eta) = v$ so that $v[\underset{\sim}{X}, \eta] = v$ initially. Suppose that $u \in W_{\mathfrak{S}}$ is of the form

$$sb_i s x_0 p_0 x_1 \cdots x_{N-2} p_{N-2} x_{N-1} t e_j t$$

where $s, t, p_0, \ldots, p_{N-2} \in \mathcal{T}$, $b_i \in \mathcal{B}$, $e_j \in \mathcal{E}$ and $x_0, \ldots, x_{N-1} \in \mathcal{A}$. Choose $K = J$ so that we start at the correct initiator. In specifying the

relations, we need only concern ourselves with the terms stored in v at each stage of the computation. We use the word of truth at each stage to set the truth-values of the relations for the current value of v. Thus, for example, if after executing assignment schemata $x_0, x_1, \ldots, x_{r-1}$, $r < N$, the term σ is stored in v, then the word of truth $p_{r-1} = q_0 q_1 \ldots q_{M-1}$ (where $Atm_{\mathfrak{C}}$ has M members) tells us how to specify the relations. If q_j, $j < M$ is $r_i(\tau_0, \ldots, \tau_{n_i-1})$, where each of $\tau_0, \ldots, \tau_{n_i-1}$ is the variable v, then $R_i(c_0, \ldots, c_{n_i-1})$, where each one of $c_0, \ldots, c_{n_i-1}$ is σ, the current contents of v. If q_j, $j < M$, is $\neg r_i(\tau_0, \ldots, \tau_{n_i-1})$, then not $R_i(c_0, \ldots, c_{n_i-1})$. We are guaranteed that the specification of the relations can be achieved without conflict since after each assignment schema x_i, $i < N$, is executed, we know that v will contain a new term that has not previously arisen earlier in the execution of $\mathfrak{C}$.

Thus, when $\mathfrak{C}$ is executed in $\underset{\sim}{X}$ with the initial state η starting at initiator b_K, it halts and generates the word $u \in W_{\mathfrak{C}}$.

To obtain result (iii), simply notice that for any E-program $\mathfrak{C}$,

$$\|\alpha_{\mathfrak{C}}\| = \bigcup_{sb_i sxte_j t \in W_{\mathfrak{C}}} [sb_i s'xte_j t' : s', t' \in \mathcal{T}]$$

Thus, the words in $W_{\mathfrak{A}}$ and $W_{\mathfrak{B}}$ represent the generable words, and associated with each such word is a set of words which are not generable because operator letters that are initiators or terminators cannot affect the truth values of the atomic formulas. Since all of $\|\alpha_{\mathfrak{A}}\|$ and all of $\|\alpha_{\mathfrak{B}}\|$ are obtained this way, and since like words from $W_{\mathfrak{A}}$ and $W_{\mathfrak{B}}$ give rise to like sets of non-generable words, $W_{\mathfrak{A}} = W_{\mathfrak{B}} \Rightarrow \|\alpha_{\mathfrak{A}}\| = \|\alpha_{\mathfrak{B}}\|$. ∎

Remarks:

(i) Since $\vdash \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$, for arbitrary abstract E-programs $\mathfrak{A}, \mathfrak{B} \in A_s$,

is decidable (Theorem 33', the strong equivalence of abstract E-programs is therefore decidable by Theorem 35.

(ii) Abstract E-programs correspond to Ianov's program schemes because, as we have seen, only syntactically identical sequences of operators, i.e., assignment schemata, are strongly equivalent, and the truth value of each atomic formula may be affected by each of the operators. These are the properties that characterize Ianov's schemes.

(iii) Since the notions of strong equivalence for abstract E-programs and K-equivalence for K-expressions are identical, the notions of K-expression and K-event therefore constitute a reformulation of Ianov's results. As well, to say any two E-programs are K-equivalent is to say they are equivalent in the sense of Ianov's definition of equivalence.

(iv) One notion we have not yet explored is that of "shift distribution" as defined by Ianov [16] and extended by Rutledge [37]. This we do in the next section.

Shift Sets and Shift K-events

Ianov [16] uses his "shift distribution" to indicate for each operator in a program schema what atomic formulas could be affected by the execution of that operator. Rutledge [37] extends this by indicating for each operator and each possible set of truth-values for the atomic formulas (i.e., for each word of truth in our scheme) the possible sets of truth-values after execution of the operator. Rutledge's method for specifying relationships between the operators and atomic formulas is more extensive than Ianov's shift distribution, and in fact includes it as a sub-concept.

For our scheme of K-expressions and K-events, the implementation of these notions is straightforward and natural. We will first define the notions of shift set and shift K-event with respect to the set $\mathcal{K}$ of K-expressions defined over the alphabets $\mathcal{b} = \{g_0, \ldots, g_{n-1}\}$ and $\mathcal{Am} = \{p_0, \ldots, p_{m-1}\}$, and then subsequently indicate how these concepts strengthen our ability to detect strong equivalence of E-programs.

Shift K-events are simply a generalization of K-events. In fact, the shift K-event $\|\alpha\|_S$ associated with the K-expression α is evaluated in precisely the same manner as the K-event $\|\alpha\|$, except for the case of operator letters in $\mathcal{b}$. Shift K-events are defined with respect to a shift set $S = \{S_0, S_1, \ldots, S_{n-1}\}$, where $S_i \subseteq \|g_i\|$, $i < n$. (Intuitively, a word $sg_it \in S_i$, where $s, t \in \mathcal{T}$, indicates a "permissable event", i.e., the words of truth s and t give truth-values for the atomic formulas that are compatible with the properties of the operator letter g_i.) The definition of shift K-event with respect to a shift set S is then,

$$\|p\|_S = \|p\| \qquad p \in \mathcal{p}$$

$$\|g_i\|_S = S_i \qquad g_i \in \mathcal{b}$$

$$\|\alpha \vee \beta\|_S = \|\alpha\|_S \cup \|\beta\|_S$$

$$\|\alpha^*\|_S = \|1\|_S \cup \|\alpha\|_S \cup \|\alpha\alpha\|_S \cup \|\alpha\alpha\alpha\|_S \cup \ldots$$

$$\|\alpha\beta\|_S = \{xpy : xp \in \|\alpha\|_S \ \& \ py \in \|\beta\|_S \ \& \ p \in \mathcal{T}\} .$$

Two K-expressions $\alpha, \beta \in \mathcal{K}$ are said to be K-equivalent with respect to a shift set S iff $\|\alpha\|_S = \|\beta\|_S$, i.e., iff $\vdash_S \alpha = \beta$.

For any shift set S, let $\mathcal{T}_K(S)$ be the formal theory obtained from $\mathcal{T}_K$ by adjoining to $\mathcal{Ax}_K$ the axiom schema

C8: $g_i = \vee S_i$, $i = 0, 1, \ldots, n-1$.

If $\alpha = \beta$ is derivable in $\mathcal{T}_K(S)$, we write $\vdash_S \alpha = \beta$.

Theorem 36: For any K-expressions $\alpha, \beta \in \mathcal{K}$ and any shift set S , $\vdash_S \alpha = \beta \Leftrightarrow \vdash_S \alpha = \beta$.

Thus, the theory $\mathcal{T}_K(S)$ is complete, i.e., both sound and adequate, for K-equivalence with respect to the shift set S .

Proof: First, let us show soundness, i.e., $\vdash_S \alpha = \beta \Rightarrow \vdash_S \alpha = \beta$. It is obvious from the definition of shift K-event, that $\vdash \alpha = \beta \Rightarrow \vdash_S \alpha = \beta$. Since $\vdash \alpha = \beta \Rightarrow \models \alpha = \beta$ by Theorem 28, then $\vdash \alpha = \beta \Rightarrow \models_S \alpha = \beta$. In addition, for axiom schema C8, $\|g_i\|_S = S_i = \|\vee S_i\|$, so that $\models_S g_i = \vee S_i$. Thus, $\vdash_S \alpha = \beta \Rightarrow \models_S \alpha = \beta$, as required.

Now, let us consider adequacy, i.e., $\models_S \alpha = \beta \Rightarrow \vdash_S \alpha = \beta$. We proceed precisely as in Theorems 33 and 34, except that now the normal form for $g_i \in \mathcal{L}$ is obtained directly using C8. Thus $N(g_i) = \vee\|g_i\|_S = \vee S_i$ (instead of $N(g_i) = \vee\|g_i\|$, as before), and C8 gives $\vdash g_i = N(g_i)$, as required. ∎

Our goal is to detect the strong equivalence of two E-programs $\mathfrak{A}$ and $\mathfrak{B}$ by testing for the K-equivalence, with respect to a shift set, of the K-expressions $\alpha_{\mathfrak{A}}$ and $\alpha_{\mathfrak{B}}$ derived from those E-programs. To facilitate this testing, we develop the notion of consistent shift set. Intuitively, a shift set is consistent iff for any E-program $\mathfrak{A}$, no word in $\|\alpha_{\mathfrak{A}}\| - \|\alpha_{\mathfrak{A}}\|_S$ is generable by an execution of $\mathfrak{A}$ in some computing structure. Thus, by cutting down a K-event for an E-program to some shift K-event, we have not deleted any words which could be produced by some execution of that E-program.

There are several possibilities for the construction of consistent shift sets, one of which we now examine. Consider the word

$$u = s_0 s_1 \cdots s_{m-1}\, g_i\, t_0 t_1 \cdots t_{m-1}$$

where $s_0 s_1 \cdots s_{m-1}$, $t_0 t_1 \cdots t_{m-1} \in \mathcal{T}$, i.e., are words of truth, and g_i is an assignment schema. If for some $j < m$, $s_j \neq t_j$, and yet none of the assigned variables in g_i occur in s_j, then the word u could never arise during the execution of an E-program. That is, words cannot arise during execution that indicate changes in truth-values of atomic formulas when those atomic formulas do not contain an occurrence of one of the assigned variables of the intervening assignment schema. Thus, S_i is defined to be the set of all words $u \in \|g_i\|$ such that for all $j < m$, if $s_j \neq t_j$ then an assigned variable in g_i occurs in S_j.

Consider the following example. Suppose

$$\mathcal{Atm} = \{r(u),\ p(u,\ f(v))\}$$

$$\mathcal{Y} = \{u := f(u),\ v := g(v),\ w := h(w)\}\ .$$

Then, abbreviating $\mathcal{Atm}$ as $\{r,\ p\}$, we have

$$\mathcal{T} = \{rp,\ r\bar{p},\ \bar{r}p,\ \bar{r}\bar{p}\}$$

Using the criterion discussed above for forming a consistent shift set, and abbreviating $\mathcal{Y}$ as $\{g_0,\ g_1,\ g_2\}$, we have,

$$S_0 = \|g_0\|$$

$$S_1 = \{rpg_1rp,\ rpg_1r\bar{p},\ r\bar{p}g_1r\bar{p},\ r\bar{p}g_1rp$$
$$\bar{r}pg_1\bar{r}p,\ \bar{r}pg_1\bar{r}\bar{p},\ \bar{r}\bar{p}g_1\bar{r}\bar{p},\ \bar{r}\bar{p}g_1\bar{r}p\}$$

$$S_2 = \{rpg_2rp,\ r\bar{p}g_2r\bar{p},\ \bar{r}pg_2\bar{r}p,\ \bar{r}\bar{p}g_2\bar{r}\bar{p}\}\ .$$

Thus, for example, $rpg_1\bar{r}p \notin S_1$ since $v := g(v)$ cannot affect the truth-value of $r(u)$.

Theorem 37: For any E-programs $\mathfrak{A}$ and $\mathfrak{B}$, of the same type, with associated K-expressions $\alpha_{\mathfrak{A}}$, $\alpha_{\mathfrak{B}} \in \mathcal{K}$, and then for any consistent shift set S defined with respect to $\mathcal{K}$,

$$\vdash_S \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}} \Rightarrow \vdash \mathfrak{A} \simeq \mathfrak{B}.$$

Proof: This theorem is the counterpart of Theorem 27, and is proved in a similar manner, although with far less detail.

With every halting execution of $\mathfrak{A}$, we can associate a word in $\|\alpha_{\mathfrak{A}}\|_S$. This is so because S being consistent assures us that only "impossible" words have been deleted from $\|\alpha_{\mathfrak{A}}\|$ to yield $\|\alpha_{\mathfrak{A}}\|_S$. Suppose $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate and produces the word $w \in \|\alpha_{\mathfrak{A}}\|_S$. Since the hypothesis of the theorem gives that $\|\alpha_{\mathfrak{A}}\|_S = \|\alpha_{\mathfrak{B}}\|_S$, then $w \in \|\alpha_{\mathfrak{B}}\|_S$ as well. Using precisely the same argument given in the proof of Theorem 27, we obtain that $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ also is determinate, and in fact produces this same word w.

This, of course, gives $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle] = \mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$, for any $\underset{\sim}{D}$ ξ and i such that $\mathfrak{A}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate. A similar result obtains when we assume that $\mathfrak{B}[\underset{\sim}{D}, \langle \xi, i \rangle]$ is determinate, and both together give $\vdash \mathfrak{A} \simeq \mathfrak{B}$. ∎

Once again, we have been able to strengthen our ability to detect strong equivalence of E-programs. Given two E-programs we would construct their associated K-expressions, and the shift set S, based on the method described above. Since the proof of Theorem 36 is constructive, then $?\vdash_S \alpha = \beta?$, for arbitrary $\alpha, \beta \in \mathcal{K}$ and shift set S, is decidable.

In fact, if $\vdash_S \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$, then a proof of this in $\mathcal{J}_K(S)$ is produced, and we conclude $\vdash \mathfrak{A} \simeq \mathfrak{B}$.

To see that this technique constitutes an improvement in our ability to detect strong equivalence, consider the example in Figure 51. For the E-program $\mathfrak{A}$, we have

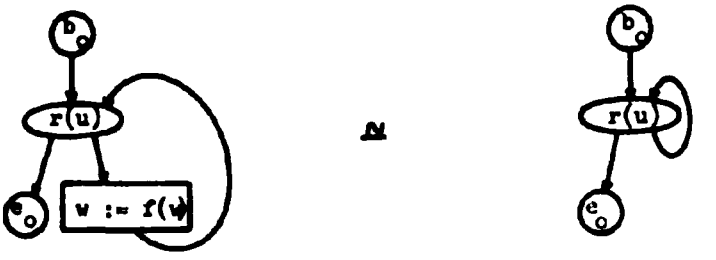
$$\alpha_{\mathfrak{A}} = b_o(\sim r(u)w := f(w))^* r(u)e_o$$
$$= b(\bar{r}f)^* re \text{, if we abbreviate.}$$

For E-program $\mathfrak{B}$, we have

$$\alpha_{\mathfrak{B}} = b_o(\sim r(u))^* r(u)e_o$$
$$= b(\bar{r})^* r\, e \text{, if we abbreviate.}$$

Notice that $\|\alpha_{\mathfrak{A}}\| \neq \|\alpha_{\mathfrak{B}}\|$. If we construct a consistent shift set S, where $\{rfr, \bar{r}f\bar{r}\} \in S$ is the member for the operator letter f (reflecting the fact that $w := f(w)$ cannot affect the truth value of $r(u)$), then $\mathcal{J}_K(S)$ yields the following derivation.

$b(\bar{r}f)^* re = b(\bar{r}(rfr \vee \bar{r}f\bar{r}))^* re$	C8, R2
$= b(\bar{r}rfr \vee \bar{r}\bar{r}f\bar{r})^* re$	S4
$= b(0fr \vee \bar{r}f\bar{r})^* re$	C1, C2, C3
$= b(0 \vee \bar{r}f\bar{r})^* re$	P3
$= b(\bar{r}f\bar{r})^* re$	S3, S9
$= b(1 \vee \bar{r}f\bar{r}(\bar{r}f\bar{r})^*)re$	S10
$= b(1 \vee \bar{r}f(\bar{r}\bar{r}f)^*\bar{r})re$	P2
$= bre \vee b\bar{r}f(\bar{r}\bar{r}f)^*\bar{r}re$	S4, S5
$= bre \vee b\bar{r}f(\bar{r}\bar{r}f)^*0e$	C2, C3
$= bre \vee 0$	S8, P3
$= blre$	S9, S7
$= b(r)^* re$	Theorem 31, F7



E-program $\mathfrak{M}$ $\not\simeq$ E-program $\mathfrak{N}$

Figure 51

Two strongly equivalent E-programs for which ordinary K-event analysis will not suffice. Here, r is a relation letter; f is a function letter; and u, w are variables.

E-program $\mathfrak{M}$ $\not\simeq$ E-program $\mathfrak{N}$

Figure 52

Two strongly equivalent E-programs for which ordinary shift K-event analysis fails. Here, r is a relation letter; f is a function letter; and u, v, w are variables.

Thus, $\vdash_S b(\bar{r}f)^*re = b(\bar{r})^*re$, i.e., $\vdash_S \alpha_{\mathfrak{M}} = \alpha_{\mathfrak{N}}$, so that Theorem 36 gives $\vdash_S \alpha_{\mathfrak{M}} = \alpha_{\mathfrak{N}}$. Then, finally, Theorem 37 gives $\vdash \mathfrak{M} = \mathfrak{N}$. So, using shift K-events, we can detect strong equivalence in cases where ordinary K-events fail.

There are many other possibilities for the construction of consistent shift sets. In Figure 52 we see two E-programs $\mathfrak{M}$ and $\mathfrak{N}$ which are strongly equivalent, but comparison of shift K-events, for a consistent shift set constructed as above, fails to detect this fact. Here we would want the shift set to reflect the fact that after certain operators, certain atomic formulas are constrained to have identical truth values.

For E-program $\mathfrak{M}$, we have

$\alpha_{\mathfrak{M}} = b_o u := v(\sim(r(u) \supset r(v)))^*(r(u) \supset r(v))e_o$

$= bf(\sim(p \supset r))^*(p \supset r)e$, if we abbreviate.

For E-program $\mathfrak{N}$, we have

$\alpha_{\mathfrak{N}} = b_o u := ve_o$

$= b\bar{e}$, if we abbreviate.

Notice that if we take the member of the shift set S for operator f to be

$\{prfpr, pr\bar{f}\bar{p}r, p\bar{r}fp\bar{r}, p\bar{r}f\bar{\bar{p}}\bar{r}, \bar{p}rf\bar{p}r, \bar{p}rfpr, \bar{\bar{p}}r\bar{f}\bar{\bar{p}}r, \bar{\bar{p}}\bar{r}fp\bar{r}\}$

as we would using the earlier method for constructing consistent shift sets, then $\|\alpha_{\mathfrak{M}}\|_S \neq \|\alpha_{\mathfrak{N}}\|$. However, let us take the member of S for f to be

$\{prfpr, p\bar{r}f\bar{p}\bar{r}, \bar{p}rfpr, \bar{p}\bar{r}f\bar{p}\bar{r}\}$,

which reflects in addition the fact that after executing $u := v$, $r(u)$ and $r(v)$ must have the same truth-value. Then $\mathcal{T}_K(S)$ yields the following derivation.

$bf(\sim(p \supset r))^*(p \supset r)e$

$= bf(p\bar{r})^*(p \supset r)e$ Theorem 31

$= bf1(p \supset r)e$ Theorem 31, P7

$= bf(p \supset r)e$ S7

$= bf(pr \vee \bar{p}r \vee \bar{\bar{p}}\bar{r})e$ Theorem 31

$= b(prfpr \vee p\bar{r}fp\bar{\bar{r}} \vee \bar{p}rfpr \vee \bar{\bar{p}}\bar{r}fp\bar{\bar{r}})(pr \vee \bar{p}r \vee \bar{\bar{p}}\bar{r})e$ C8

$= b(prfpr \vee p\bar{r}fp\bar{\bar{r}} \vee \bar{p}rfpr \vee \bar{\bar{p}}\bar{r}fp\bar{\bar{r}})e$ Theorem 32

$= bfe$ C8

Thus, $\vdash_S \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$, and so $\vdash \alpha_{\mathfrak{A}} = \alpha_{\mathfrak{B}}$. Then, Theorem 37 gives $\vdash \mathfrak{A} = \mathfrak{B}$.

Remarks:

(i) From the simple nature of the proof for Theorem 36, we see that our ability to detect strong equivalence will be improved by any device, technique or heuristic that serves to delete words from $\|\alpha_{\mathfrak{A}}\|$ that $\mathfrak{A}$ could never produce in execution. Among several possibilities, is the identification of identity operators. Thus $u := u$ occurring in $\mathfrak{A}$ would not be included in $\mathcal{L}$ but converted to 1 directly when $\alpha_{\mathfrak{A}}$ was formed.

(ii) The K-expression representation for algorithms finds application in another area besides the detection of strong equivalence. This author has devised another formulation of K-expression semantics that permits us to write down for any node in the graph of an E-program a possibly infinite qff which tells what is true at that node. At present, this work, which bears closely on the problems discussed by Floyd [11], is incomplete and so will not be discussed here any further.

BLANK PAGE

CHAPTER 10

CONCLUDING REMARKS

In this work, the principal goal has been to investigate the strong equivalence of elemental programs (i.e., E-programs). We have sought to provide a formal theoretic framework within which proofs of strong equivalence can be generated. Thus, any sequence of transformations performed on an elemental program using the axioms generates a proof that the final program is strongly equivalent to the initial one. Furthermore, availability of a formal theory of equivalence is essential should we want to mechanize proof generation or proof checking. This because such automated systems would treat these matters from the point of view of syntax not semantics.

The elemental programs and computing structures considered here together only barely meet the criterion of being ALGOL-like. Thus, while many-entrance, many-exit flowcharts of assignment schemata (possibly with subscripted variables included) and conditional branches can be termed ALGOL-like, there is still a wealth of structure in ALGOL not programmable in or reducable to this sort of formalism. Clearly FOR-loops, conditional arithmetic expressions, etc., are reducable to elemental programs, but block structure, recursive procedures and the like escape such reduction. In addition, we have concentrated on single-sorted computing structures. This even though, as we indicate in Appendix I, we likely would have to resort to many-sorted computing structures to bring into the scope of the theory those bases of computation of topical interest.

The theory of strong equivalence we introduce, while possibly incomplete, is nevertheless powerful enough to serve in many applications. In addition, the theory is complete, and even extended complete, for certain sub-cases of interest. However, there are many unanswered questions in this area. Precisely what are the limits on the derivational power of the theory? Can this power be increased? Is the theory complete for certain decidable strong equivalence sub-problems studied recently by Paterson [36]? These questions beg to be answered, but are beyond the scope of this work.

Taking another approach to the strong equivalence problem, we have introduced a hierarchy of analytic tools for discovering strong equivalence. These tools or methods rely on the notions of K-expression representation for elemental programs and on K-event interpretation of these expressions. In this area too, there are many paths of investigation that seem promising. For example, can we further refine our ability to detect strong equivalence by finding even more unexecutable words that can be cast out of the K-event corresponding to an elemental program? If so, what are the limits on this capability to detect unexecutable paths in an elemental program?

There are other aspects of the strong equivalence problem we have barely touched on in this work. One is our ability to characterize the properties of a computing structure by providing a set of proper axioms. In fact, what sort of properties of computing structures can be expressed by a set of equivalences given by proper axioms? In general, what formal techniques are required to at least partially characterize such domains as the integers? Should the theory be extended to allow propositional or quantificational statements about strong equivalence so that useful domain characterizations can be made?

In some sense, strong equivalence is too strong a property. That is, two strongly equivalent elemental programs share this property for reasons that are not very complicated. This of course must be so since all information regarding any computing structure is suppressed when making this statement of strong equivalence. That being the case, we are still a long way from a theory of equivalence which is widely applicable to strictly ALGOL-like programs. This is so because a great many of the transformations we would like to make will depend on various properties of a specific computing structure and so will not be included in our theory of strong equivalence. This makes the characterization of computing structures through proper axioms a very relevant issue, since this is the route we would take to arrive at a theory which could derive statements of equivalence of the sort we are interested in.

Another potentially useful extension of the theory presented here would be to make it "bilingual". Thus, we would define two languages for specifying algorithms, one a high level source language and the other a machine-like object language. For each language, an inferential system for deriving strong equivalence of its programs would be specified. We also would specify an additional axiom embodying a compiling transformation from the source language into the object language. To prove such a theory sound, we would have to verify that the compiler axiom was sound, i.e., that the compiling transformation was "correct". McCarthy, Painter [26,28,35] and this author [20] have all studied the problem of proving compiler transformations "correct". Such a bilingual theory would find application in systems where both pre- and post-compilation optimizations and transformations

are performed on a program. The soundness of the theory would guarantee that the final object program was in some sense strongly equivalent to the initial source program. The basic inadequacy of a mechanized version of such a theory, however, precludes a system which will always fully optimize or simplify a given source program.

We see that the road to a viable useful theory of equivalence for ALGOL-like programs is strewn with many obstacles. Our hope is that this work, to some extent, has removed some of those obstacles and so moved us further along that road.

APPENDIX I

MANY-SORTED COMPUTING STRUCTURES

Single-sorted computing structures are often inadequate for characterizing semantic bases of topical interest. The principal difficulty lies in the restriction to a single domain, since in many programming languages and computers we have more than one "data type". Here we extend the notions of signature and computing structure to indicate how this deficiency might be remedied.

A generalized signature is a 7-tuple of the form

$s = \langle I, J_o, J_1, J_2, n, m, p \rangle$

where,

(i) I is a non-empty possibly infinite index set telling how many domains there are.

(ii) J_o is a possibly infinite index set for the relations and n is a function on J_o such that for $j \in J_o$,

$n(j) = \langle i_o, \ldots, i_{s_j} \rangle$, $i_r \in I$, $r \leq s_j < \omega$.

(iii) J_1 is a possibly infinite index set for the functions and m is a function on J_1 such that for $j \in J_1$,

$m(j) = \langle i_o, \ldots, i_{t_j}, i_{t_j+1} \rangle$, $i_r \in I$, $r \leq t_j+1 <$.

(iv) J_2 is a possibly infinite index set for the designated individuals and p is a function on J_2 such that for $j \in J_2$, $p(j) \in I$.

By a many-sorted computing structure of generalized signature s, we mean a 4-tuple, $\underset{\sim}{D} = \langle \langle D_i \rangle_{i \in I}, \langle R_j \rangle_{j \in J_o}, \langle F_j \rangle_{j \in J_1}, \langle a_j \rangle_{j \in J_2} \rangle$

where,

(i) D_i is a non-empty possibly infinite set for $i \in I$

(ii) $R_j \subseteq D_{i_o} \times \ldots \times D_{i_{s_j}}$ for $j \in J_o$

(iii) $F_j : D_{i_o} \times \cdots \times D_{i_{t_j}} \to D_{i_{t_j+1}}$ for $j \in J_1$

(iv) $a_j \in D_{p(j)}$ for $j \in J_2$.

We can easily find systems that are either naturally or of necessity defined as many sorted.

(i) Consider the system

$$\langle \langle A, D, S \rangle, \langle atom, eq \rangle, \langle car, cdr, cons \rangle, \langle NIL \rangle \rangle$$

with generalized signature

$$\langle \langle 0,1,2 \rangle, \langle 0,1 \rangle, \langle 0,1,2 \rangle, 0, n, m, p \rangle$$

where,

$n(0) = 2$ $\quad$ $m(0) = \langle 1,2 \rangle$ $\quad$ $p(0) = 0$

$n(1) = \langle 0,0 \rangle$ $\quad$ $m(1) = \langle 1,2 \rangle$

$m(2) = \langle 2,2,1 \rangle$

This system is like that given by McCarthy [27] where he defines the LISP programming language. The many-sorted computing structure defined above provides a basis for computation with either s-expressions or with lists. Definitions for the various constituents of the system follow below.

A the set of atoms

D the set of dotted pairs

$S = A \cup D$ the set of s-expressions

atom = A

$eq = \{\langle x,y \rangle : x,y \in A \text{ and } x = y\}$

car : $D \to S$ so that car((x.y)) = x

cdr : $D \to S$ so that cdr((x.y)) = y

cons : $S^2 \to S$ so that cons(x,y) = (x.y)

$NIL \in A$

(ii) Consider the many-sorted computing structure

$$\langle \langle 2^{36}, 2^{15} \rangle, \langle TZE, TMI, TIX \rangle, \langle ADD, ALS, SXA \rangle, \Omega \rangle$$

with generalized signature

$$\langle \langle 0, 1 \rangle, \langle 0,1,2 \rangle, \langle 0,1,2 \rangle, 0, m, n, \Omega \rangle$$

where,

$n(0) = 0$ $\quad$ $m(0) = \langle 0,0,\Omega \rangle$

$n(1) = 0$ $\quad$ $m(1) = \langle 0,\Omega \rangle$

$n(2) = \langle 1,1 \rangle$ $\quad$ $m(2) = \langle 0,1,\Omega \rangle$

This system is an extension of the IBM 7090 example used in the text to demonstrate the single-sorted case. Here, 2^{15} denotes the set of all 15 bit words over $\{0,1\}$, which is just the range of possible values for index registers in the IBM 7090. Then, in addition to the definitions given in the previous example, we have

TIX = $\{\langle x,y \rangle : x,y \in 2^{15}$ and taken as binary representations of natural numbers, $x > y\}$

SXA : $2^{36} \times 2^{15} \rightarrow 2^{36}$ so that

$$SXA(b_0 \ldots b_{20} b_{21} \ldots b_{35}, c_0 \ldots c_{14}) = b_0 \ldots b_{20} c_0 \ldots c_{14} .$$

Most of the results obtained in this work for single-sorted computing structures would seem to have straightforward extension to the many-sorted case. It is because of the somewhat cumbersome notational framework required for many-sorted computing structures that we concern ourselves mainly with the simpler single-sorted case.

BLANK PAGE

APPENDIX II

SUBSCRIPTED VARIABLES

To introduce subscripted variables, we first must modify the definition for terms by amending clause (i) to read

(i) If $\tau_0, \ldots, \tau_{k-1}$, $k < \omega$, are terms and v_i is a variable, then $v_i(\tau_0, \ldots, \tau_{k-1})$ is a term, called a subscripted variable, and $\tau_0, \ldots, \tau_{k-1}$ are called subscripts. If $k = 0$, we have simply v_i.

The concept of the value of a term must also be revised. Now, the value of a term is defined with respect to a computing structure $\underset{\sim}{D}$, an indexing function $\underset{\sim}{i} : \underset{\sim}{D}_0 \to \omega$, which constitutes a partition of $\underset{\sim}{D}_0$ into countably-many equivalence classes called indices and a hierarchial state θ of $\underset{\sim}{D}$. A hierarchial state, or simply h-state, of $\underset{\sim}{D}$ is an ordered pair $\langle \xi, \theta' \rangle$ where $\xi : \omega \to \underset{\sim}{D}_0$ is a state in the ordinary sense and $\theta' : \omega \to \Theta$, where Θ is the set of all h-states of $\underset{\sim}{D}$. Thus, an h-state of $\underset{\sim}{D}$ is a state together with a sequence of further h-states.

Roughly speaking, the values of the subscripts of a subscripted variable give rise to indices which are used to filter down through the hierarchial structure of an h-state to finally produce a value. To accomplish this, we must first define two auxilliary functions dealing with h-states; here $\langle i_0, \ldots, i_{k-1} \rangle \in \omega^k$, $\langle \xi, \theta' \rangle \in \Theta$ and $a \in \underset{\sim}{D}_0$

$$\hat{c}(\langle i_0, \ldots, i_{k-1}, \langle \xi, \theta' \rangle)$$

$$= \hat{c}(\langle i_0, \ldots, i_{k-2} \rangle, c(i_{k-1}, \theta')) \qquad \text{if } 1 < k < \omega$$

$$= c(i_0, \xi) \qquad \text{if } k = 1$$

$\hat{a}(\langle i_0, \ldots, i_{k-1}\rangle, d, \langle \xi, \theta'\rangle)$

$= \langle \xi, c(i_{k-1}, \hat{a}(\langle i_0, \ldots, i_{k-2}\rangle, d, c(i_{k-1}, \theta')), \theta'\rangle$ if $1 < k < \omega$

$= \langle a(i_0, d, \xi), \theta\rangle$ if $k = 1$

Then, the value of a term τ with respect to $\underset{\sim}{D}$, $\underset{\sim}{I}$ and θ is denoted by $\tau[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$ and is defined as follows.

(i) If τ is a constant k_i, then $\tau[\underset{\sim}{D}, \underset{\sim}{I}, \theta] = k_i[D, I, \theta] = a_i$.

(ii) If τ is $f_i(\tau_0, \ldots, \tau_{m_i-1})$, then

$\tau[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$

$= f_i(\tau_0, \ldots, \tau_{m_i-1})[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$

$= F_i(\tau_0[\underset{\sim}{D}, \underset{\sim}{I}, \theta], \ldots, \tau_{m_i-1}[\underset{\sim}{D}, \underset{\sim}{I}, \theta])$.

(iii) If τ is $v_i(\tau_0, \ldots, \tau_{k-1})$, $k < \omega$, then

$\tau[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$

$= v_i(\tau_0, \ldots, \tau_{k-1})[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$

$= \hat{c}(\langle i, \underset{\sim}{I}(\tau_0[\underset{\sim}{D}, \underset{\sim}{I}, \theta]), \ldots, \underset{\sim}{I}(\tau_{k-1}[\underset{\sim}{D}, \underset{\sim}{I}, \theta])\rangle, \theta)$.

We amend the definition of qffs to permit terms as redefined above.

Now let us amend the definition of assignment schemata so that in $(u_j := \sigma_j)_{j<n}$, $\sigma_0, \ldots, \sigma_{n-1}$ are terms as redefined above and the $u_0, \ldots, u_{n-1}$ are subscripted variables. The well-formedness condition here is that for all $i < \omega$, for all $k < \omega$, v_i can occur but once as an assigned variable with k subscripts in $(u_i := \tau_i)_{i<n}$. Thus, we allow two different "arrays" to have the same "name" provided they have a different number of subscripts, and the well-formedness condition for assignment schemata simply says that the $u_0, \ldots, u_{n-1}$ must access or refer to distinct "arrays". This is a necessary condition, since for some $\underset{\sim}{D}$, $\underset{\sim}{I}$ and θ we might try to make simultaneous assignments to the same element of some "array".

An assignment schema $f = (u_j := \sigma_j)_{j<n}$ applied to an h-state θ produces a new state $f[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$ defined as follows.

$(u_j := \sigma_j)_{j<n}[\underset{\sim}{D}, \underset{\sim}{I}, \theta]$

$= \hat{a}(\langle i, \underset{\sim}{I}(\tau_0[\underset{\sim}{D}, \underset{\sim}{I}, \theta]), \ldots, \underset{\sim}{I}(\tau_{k-1}[\underset{\sim}{D}, \underset{\sim}{I}, \theta])\rangle ,$

$\sigma_{n-1}[\underset{\sim}{D}, \underset{\sim}{I}, \theta], (u_j := \sigma_j)_{j<n-1}[\underset{\sim}{D}, \underset{\sim}{I}, \theta])$

if $1 < n < \omega$ and $u_{n-1} = v_i(\tau_0, \ldots, \tau_{k-1})$, $k < \omega$

$= \hat{a}(\langle i, \underset{\sim}{I}(\tau_0[\underset{\sim}{D}, \underset{\sim}{I}, \theta]), \ldots, \underset{\sim}{I}(\tau_{k-1}[\underset{\sim}{D}, \underset{\sim}{I}, \theta])\rangle . \sigma_0[\underset{\sim}{D}, \underset{\sim}{I}, \theta], \theta)$

if $n = 1$ and $u_{n-1} = v_i(\tau_0, \ldots, \tau_{k-1})$, $k < \omega$

Notice that the case of no subscripted variables is just a special case of the extension introduced above. In that situation, only the "first" element of each "array" is accessed.

Notice also that the functions $\hat{c}$ and $\hat{a}$ act together like a "storage mapping function" in the sense this term is usually used when subscripted variables in a source language are implemented in some object language, usually machine code.

BLANK PAGE

APPENDIX I.1

DERIVATIONS FOR THE PROPOSITIONAL AXIOMS

(i) $\vdash (p \supset (q \supset p)) = 1$

$p \supset (q \supset p) = \sim p \vee (q \supset p)$	C5
$= \sim p \vee (\sim q \vee p)$	C5
$= \sim p \vee (p \vee \sim q)$	S2
$= (\sim p \vee p) \vee \sim q$	S1
$= (p \vee \sim p) \vee \sim q$	S3
$= 1 \vee \sim q$	C4
$= (q \vee \sim q) \vee \sim q$	C4
$= q \vee (\sim q \vee \sim q)$	S1
$= q \vee \sim q$	C1
$= 1$	C4

(ii) $\vdash ((p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))) = 1$

$(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$	
$= \sim (p \supset (q \supset r)) \vee ((p \supset q) \supset (p \supset r))$	C5
$= p \sim (q \supset r) \vee \sim (p \supset q) \vee (p \supset r)$	C5, C6
$= pq\bar{r} \vee p\bar{q} \vee \bar{p} \vee r$	C5, C6
$= pq\bar{r} \vee p\bar{q}1 \vee \bar{p}11 \vee 11r$	S7, P4
$= pq\bar{r} \vee p\bar{q}(r \vee \bar{r}) \vee$ $\bar{p}(q \vee \bar{q})(r \vee \bar{r}) \vee (p \vee \bar{p})(q \vee \bar{q})r$	C4
$= pq\bar{r} \vee (p\bar{q}r \vee p\bar{q}\bar{r}) \vee$ $(\bar{p}qr \vee \bar{p}q\bar{r} \vee \bar{p}\bar{q}r \vee \bar{p}\bar{q}\bar{r}) \vee$ $(pqr \vee p\bar{q}r \vee \bar{p}qr\ \bar{p}\bar{q}r)$	S4, S2
$= \vee \mathcal{T}$	S6
$= 1$	Theorem 31

(iii) $\vdash((\sim p \supset q) \supset ((\sim p \supset \sim q) \supset p)) = 1$

$(\sim p \supset q) \supset ((\sim p \supset \sim q) \supset p)$

$= \sim(\sim p \supset q) \vee ((\sim p \supset \sim q) \supset p)$ C5

$= \bar{p}\bar{\bar{q}} \vee \sim(\sim p \supset \sim q) \vee p$ C5, C6

$= \bar{p}\bar{\bar{q}} \vee \bar{p}\bar{\bar{\bar{q}}} \vee p$ C6

$= \bar{p}\bar{\bar{q}} \vee \bar{p}q \vee p$ C7

$= \bar{p}(\bar{q} \vee q) \vee p$ S4

$= \bar{p}\,1 \vee p$ C4, S3

$= \bar{p} \vee p$ S7

$= p \vee \bar{p}$ S3

$= 1$ C4

(vi) $\vdash(p \supset q) = 1,\ \vdash p = 1 \Rightarrow \vdash q = 1$

$q = 0 \vee q$ S9, S3

$= 1\bar{1} \vee q$ C2

$= \bar{1} \vee q$ P4

$= \bar{p} \vee q$ hypothesis

$= (p \supset q)$ C5

$= 1$ hypothesis

BIBLIOGRAPHY

[1] Berge, C., The Theory of Graphs and its Applications, Wiley, New York (1962).

[2] Bohm, C. and Jacopini, G., Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules, Comm. ACM, Vol. 9, No. 5 (1964) pp. 366-371.

[3] Carnap, R., The Logical Syntax of Language, Harcourt, Brace & Co., New York (1937).

[4] Church, A., Introduction to Mathematical Logic, Princeton University Press, Princeton (1956).

[5] Cooper, D. C., Some Transformations and Standard Forms of Graphs, with Applications to Computer Programs, in Machine Intelligence 2, Edited by D. Michie, Oliver and Boyd, Edinburgh (1968).

[6] Curry, H. B., Outlines of a Formalist Philosophy of Mathematics, North-Holland, Amsterdam (1961).

[7] Davis, M., Computability and Unsolvability, McGraw Hill, New York (1958).

[8] Engeler, E., Algorithmic Properties of Structures, Mathematical Systems Theory, Vol. 1, No. 3 (1967).

[9] Ershov, A. P., Operator Algorithms I, Problems of Cybernetics, Vol. III (1962) pp. 697-763.

[10] Feferman, S., Notes for a Course in Metamathematics, Stanford Libraries, Lib. of Congr. call no. QA9/F4.

[11] Floyd, R. W., Assigning Meaning to Programs, Preliminary draft, (May 1966).

[12] Gandy, R., Recursive Function Theory, Class notes for Phil. 293a, Stanford University (Sept. 1966).

[13] Glushkov, V. M., Automata Theory and Formal Microprogram Transformations, Kibernetika, Vol. 1, No. 5 (1965) pp. 1-9.

[14] Halmos, P. R., Naive Set Theory, Van Nostrand, Princeton (1960).

[15] Harrison, M. A., Introduction to Switching and Automata Theory, McGraw Hill, New York (1965).

[16] Ianov, Iu I., The Logical Schemes of Algorithms, Problems of Cybernetics, Vol. I (1960) pp. 82-140.

[17] Ito, T., On Certain Representation of Algorithms - Part I, Unpublished draft, Stanford University (1967).

[18] Ito, T., Personal communication concerning an uncirculated preliminary draft of On Certain Representation of Algorithms - Part II, Stanford University (1967).

[19] Kaluzhnin, L. A., Algorithmization of Mathematical Problems, Problems of Cybernetics, Vol. II (1961) pp. 371-391.

[20] Kaplan, D. M., Correctness of a Compiler for ALGOL-like Programs, Stanford Artificial Intelligence Project Memo No. 48, Stanford University (July 1967).

[21] Kaplan, D. M., Some Completeness Results in the Mathematical Theory of Computation, J. ACM, Vol. 15, No. 1 (1968) pp. 124-134.

[22] Karp, C. R., Languages with Expressions of Infinite Length, North-Holland, Amsterdam (1964).

[23] Kleene, S. C., Representation of Events in Nerve Nets and Finite Automata, in Automata Studies, edited by C. E. Shannon and J. McCarthy, Princeton University Press, Princeton (1956) pp. 3-42.

[24] Luckham, D. and Park, D., The Undecidability of the Equivalence Problem for Program Schemata, Bolt, Beranek and Newman Inc. Report No. 1164 (1964).

[25] Luckham, D. C., Park, D. M. R. and Paterson, M. S., On Formalised Computer Programs, Preliminary Draft, Programming Research Group, Oxford University, (August 1967).

[26] McCarthy, J. and Painter, J., Correctness of a Compiler for Arithmetic Expressions, Stanford Artificial Intelligence Project Memo No. 40, Stanford University (April 1966).

[27] McCarthy, J., Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I, Comm. ACM, Vol. 3 (196) pp. 184-195.

[28] McCarthy, J., Towards a Mathematical Science of Computation, Proc. IFIP Congress 62, North-Holland, Amsterdam (1962) pp. 21-26.

[29] McCarthy, J., Lectures for CS245, Mathematical Theory of Computation, Stanford University (Jan. 1967).

[30] McCarthy, J., A Basis for a Mathematical Theory of Computation, in Computer Programming and Formal Systems, Edited by P. Braffort and D. Hirshberg, North-Holland, Amsterdam (1963).

[31] McNaughton, R. and Yamada, H., Regular Expressions and State Graphs for Automata, Trans. IRE, EC-9 (1960) pp. 39-47.

[32] Manna, Z., Termination of Algorithms, Thesis, Carnegie-Mellon University (1968).

[33] Mendelson, E., Introduction to Mathematical Logic, Van Nostrand, Princeton (1964).

[34] Narasimhan, R., Programming Languages and Computers: a Unified Metatheory, Advances in Computers, Vol. 8 (1967) pp. 189-224.

[35] Painter, J., Semantic Correctness of a Compiler for an ALGOL-like Language, Stanford Artificial Intelligence Project Memo No. 44, Stanford University (March 1967).

[36] Paterson, M. S., Equivalence Problems in a Model of Computation, Thesis, University of Cambridge (August 1967).

[37] Rutledge, J. D., On Ianov's Program Schemata, J. ACM, Vol. 11, No. 1 (1964) pp. 1-9.

[38] Salomaa, A., Two Complete Axiom Systems for the Algebra of Regular Events, J. ACM, Vol. 13, No. 1 (1966) pp. 158-169.

[39] Scott, D., Some Definitional Suggestions for Automata Theory, Journal of Computer and System Sciences, Vol. 1, No. 2 (1967).

[40] Shepherdson, J. C. and Sturgis, H. E., Computability of Recursive Functions, J. ACM, Vol. 10, No. 2 (1963) pp. 217-256.

[41] Tixier, V., Recursive Functions of Regular Expressions in Language Analysis, Computer Science Department Technical Report No. 58, Stanford University (March 1967).