

Reproduced by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information Springfield Va. 22151

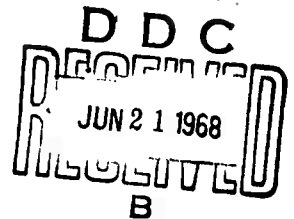
TERMINATION OF ALGORITHMS

by

Zohar Manna

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania
April 1968

Submitted to the Carnegie-Mellon University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy



This work was supported by the Advanced Research Projects
Agency of the Office of the Secretary of Defense (SD-146)
and is monitored by the Air Force Office of Scientific
Research. Distribution of this document is unlimited.

ABSTRACT

The thesis contains two parts which are self-contained units.

In Part I we present several results on the relation between

1. the problem of termination and equivalence of programs and abstract programs, and
2. the first order predicate calculus.

Part II is concerned with the relation between

1. the termination of interpreted graphs, and
2. properties of well-ordered sets and graph theory.

ACKNOWLEDGMENTS

I am indebted to Professors Robert Floyd and Alan Perlis for their guidance in this research.

I also wish to thank Professors David Cooper and Abraham Ginzburg for their help in the early stages of my work.

I received valuable comments from Professor John McCarthy, Professor Peter Andrews, Professor Donald Loveland, and Dr. David Shafer.

I am also grateful to the graduate students in the Department of Computer Science for their assistance, especially: Richard Waldinger, James King, Darius Irani, Frederick Haney and Robert Wagner.

Table of Contents

PART I

<u>Introduction</u>	1
<u>Chapter 1: Mathematical Background</u>	
1.1 The (First-Order) Predicate Calculus.	4
1.2 The Validity-Problem of the Predicate Calculus.	11
1.3 Directed Graphs	12
<u>Chapter 2: Definitions</u>	
2.1 Abstract Programs	14
2.2 Programs.	17
2.3 Interpreted Programs.	19
<u>Chapter 3: Termination of Programs and Abstract Programs</u>	
3.1 The Algorithm to Construct W_{AP} (Algorithm 1).	23
3.2 Termination of Programs (Theorem 1)	27
3.3 Termination of Abstract Programs (Theorem 2).	33
3.4 The Termination-Problem of Abstract Programs.	37
<u>Chapter 4: Equivalence of Programs and Abstract Programs</u>	
4.1 The Algorithm to Construct $W_{AP,AP}$ (Algorithm 2).	41
4.2 Equivalence of Programs (Theorem 3)	46
4.3 Equivalence of Abstract Programs (Theorem 4).	50
<u>Chapter 5: Termination of Non-Deterministic Programs and Non-Deterministic Abstract Programs</u>	
5.1 Definitions	54
5.2 Weak Termination (Theorems 5 and 6)	59

5.3 The Algorithm to Construct In_{Up} (Algorithm 3)	61
5.4 Strong Termination of Non-Deterministic Programs (Theorem 7)	64
5.5 Strong Termination of Non-Deterministic Abstract Programs (Theorem 8).	68
<u>References</u>	70

PART II

<u>Introduction</u>	72
<u>Chapter 1: Mathematical Background</u>	
1.1 Well-Ordered Sets	74
1.2 Directed Graphs	75
<u>Chapter 2: Definitions</u>	79
<u>Chapter 3: Termination of Interpreted Graphs</u>	
3.1 Theorem 1	83
3.2 Theorem 2	88
<u>Chapter 4: Applications</u>	
4.1 Example 1	90
4.2 Example 2	96
<u>References</u>	99

about the equivalence of abstract programs can be obtained just by applying well-known results in logic.

The corresponding result for programs suggests a new approach for proving the equivalence and correctness of 'real' programs.

Chapter 5 is concerned mainly with the strong termination of non-deterministic programs and non-deterministic abstract programs.

In a non-deterministic program an assignment of values to its input variables does not necessarily define a unique execution of the program. A non-deterministic program is said to terminate strongly if for each assignment of values to its input variables all possible executions terminate.

The results of this chapter are a generalization of the results obtained in Chapter 3. These results have an application in proving the convergence of recursively defined functions.

INTRODUCTION

In this part of the thesis we shall present several results on the relation between:

1. the problem of termination and equivalence of programs and abstract programs, and
2. the first order predicate calculus.

An abstract program (program schema) is a program, but with function, predicate and constant symbols, instead of specified functions, predicates and constants. Thus, an abstract program AP may be thought of as representing a family of (real) programs. By specifying an interpretation $\mathfrak{I}$ for the symbols of AP, a program $(AP, \mathfrak{I})$ of this family is obtained. The program contains a set of input variables. Each assignment of values to the input variables defines a (unique) execution of the program.

Chapter 1 (Mathematical Background) and Chapter 2 (Definitions) are introductory chapters.

Chapter 3 is concerned with the termination problem of programs and abstract programs. A program $(AP, \mathfrak{I})$ is said to terminate if all possible executions of the program terminate. An abstract program AP is said to terminate if for every interpretation $\mathfrak{I}$, the program $(AP, \mathfrak{I})$ terminates.

Given an abstract program AP , an algorithm is described to construct a well-formed formula W_{AP} of the first order predicate calculus, such that AP terminates if and only if W_{AP} is unsatisfiable, i.e., $\sim W_{AP}$ is valid. This implies that conclusions about the termination of abstract programs can be obtained just by applying well-known results in logic.

A corresponding result for programs is presented.

Chapter 4 is concerned with the equivalence problem of programs and abstract programs.

Two programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are said to be equivalent if their 'corresponding' execution sequences always terminate and give the same final value. Two abstract programs AP and AP' are said to be equivalent if for every interpretation $\mathfrak{I}$, the corresponding programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are equivalent.

Given two abstract programs AP and AP' , an algorithm is described to construct well-formed formula $W_{AP, AP'}$ of the first-order predicate calculus, such that AP and AP' are equivalent if and only if $W_{AP, AP'}$ is unsatisfiable, i.e., $\sim W_{AP, AP'}$ is valid. Consequently, conclusions

CHAPTER 1: MATHEMATICAL BACKGROUND

1.1 The (First-Order) Predicate Calculus

In this section we shall partially follow the exposition of Davis and Putnam [1960].

The symbols of which our formulas are constructed are:

(a) Improper symbols

punctuation marks

, ()

logical symbols

$\sim \supset \wedge \vee = \exists$

primitive constants

T and F.

(b) Constants

n-adic function constants

f_i^n ($i \geq 1, n \geq 0$)

$[f_i^0$ are called individual constants],

n-adic predicate constants

p_i^n ($i \geq 1, n \geq 0$)

$[p_i^0$ are called propositional constants].

(c) Variables

individual variables

x_i ($i \geq 1$)

n-adic predicate variables

q_i^n ($i \geq 1, n \geq 0$)

$[q_i^0$ are called propositional variables].⁽¹⁾

¹In the following, we shall use also y_i as individual variables and a_i as individual constants.

The subscripts and the superscripts will be omitted whenever their omission can cause no confusion.

Among all the expressions which can be formed using these symbols, we distinguish three classes which are defined recursively as follows:

(a) Terms

1. Each individual variable x_i and each individual constant f_i^0 is a term;
2. If $t_1, t_2, \dots, t_n$ ($n \geq 1$) are terms, then so is $f_i^n(t_1, t_2, \dots, t_n)$;
3. The terms consist exactly of the expressions generated by 1 and 2.

(b) Atomic formulas

1. T, F, p_i^0 and q_i^0 are atomic formulas.
2. If $t_1, t_2, \dots, t_n$ ($n \geq 1$) are terms, then the expressions $p_i^n(t_1, t_2, \dots, t_n)$ and $q_i^n(t_1, t_2, \dots, t_n)$ are atomic formulas.
3. The atomic formulas consist exactly of the expressions generated by 1 and 2.

(c) Well-formed formulas (wff's)

1. An atomic formula is a wff.
2. If R is a wff, then so are $\sim R$, $(x_i)R$ [x_i is said to be universally quantified], and $(\exists x_i)R$ [x_i is said to be existentially quantified].
3. If R and S are wffs, then so are $(R \supset S)$, $(R \wedge S)$, $(R \vee S)$, and $(R \equiv S)$.

4. The wff's consist exactly of the expressions generated by 1, 2, and 3.

Parentheses will be omitted whenever their omission can cause no confusion.

An occurrence of x_i in a wff R is a bound occurrence if it is in a wf-part of R of the form $(x_i)S$ or $(\exists x_i)S$. An occurrence of x_i which is not bound is called a free occurrence. x_i is free in R if it has at least one free occurrence in R . R is closed if it has no free individual variables.

Our next step is to single out from the class of wff's those which are logically valid. This can be done either by specifying axioms and rules of inference or by referring to "interpretations" of the wff's of the system, and by a basic result due to Gödel (Gödel Completeness Theorem) both of these procedures will lead to the same class of formulas. For our present purposes it is most convenient to use the latter formulation employing "interpretation".

An interpretation $\mathfrak{I}$ for a wff W consists of a non-empty set of elements $D_{\mathfrak{I}}$ (called the domain of the interpretation) and assignments to the constants of W :

1. To each function constant f_i^n which occurs in W , we assign a total function of n variables ranging over $D_{\mathfrak{g}}$, whose values are in $D_{\mathfrak{g}}$. [If $n = 0$, the individual constant f_i^0 is assigned some fixed element of $D_{\mathfrak{g}}$.]
2. To each predicate constant p_i^n which occurs in W , we assign a total function of n variables ranging over $D_{\mathfrak{g}}$, whose values are T or F . [If $n = 0$, the propositional constant p_i^0 is assigned the value T or F .]

Given a wff W and an interpretation $\mathfrak{g}$ for W [notation: $(W, \mathfrak{g})$].

An assignment Γ for $(W, \mathfrak{g})$ consists of assignments to the variables of W :

1. To each free individual variable x_i in W , we assign some fixed element of $D_{\mathfrak{g}}$.
2. To each predicate variable q_i^n which occurs in W , we assign a total function of n variables ranging over $D_{\mathfrak{g}}$, whose values are T or F . [If $n = 0$, the propositional variable q_i^0 is assigned the value T or F .]

Let W be a wff. Then given an interpretation $\mathfrak{g}$ for W and an assignment Γ for $(W, \mathfrak{g})$ [notation: $(W, \mathfrak{g}, \Gamma)$], a value T or F will be assigned to $(W, \mathfrak{g}, \Gamma)$. This value is obtained simply by using the assignments of $\mathfrak{g}$ and Γ , interpreting F as falsehood and T as truth,

using the usual truth tables of $\sim$, $\wedge$, $\vee$, $\supset$, and $=$, and interpreting the universally and existentially quantified variables in the standard way.

$(W, \mathfrak{I})$ is said to be:

1. valid, if for every assignment Γ , $(W, \mathfrak{I}, \Gamma)$ has the value T.
2. satisfiable (or consistent), if $(W, \mathfrak{I}, \Gamma)$ has the value T for some assignment Γ .
3. unsatisfiable, if it is not satisfiable.

Clearly, $(W, \mathfrak{I})$ is valid if and only if $(\sim W, \mathfrak{I})$ is unsatisfiable.

A wff W is said to be:

1. valid, if for every interpretation $\mathfrak{I}$, $(W, \mathfrak{I})$ is valid.
2. satisfiable (or consistent), if $(W, \mathfrak{I})$ is satisfiable for some interpretation $\mathfrak{I}$.
3. unsatisfiable, if it is not satisfiable.

Clearly, W is valid if and only if $\sim W$ is unsatisfiable.

A wff is called quantifier free if it contains no occurrence of (x_i) or $(\exists x_i)$.

A wff is in prenex normal form, if it begins with a sequence of quantifiers (x_i) and $(\exists x_i)$ in which no variable occurs more than once

(called the prefix), and if the sequence is followed by a quantifier free wff (called the matrix).

The disjunction of the wff's $R_1, R_2, \dots$, and R_n , $n \geq 1$, is the wff $R_1 \vee R_2 \vee \dots \vee R_n$; their conjunction is the wff $R_1 \wedge R_2 \wedge \dots \wedge R_n$.

A literal is a wff which is either an atomic formula or of the form $\sim R$, where R is atomic.

A clause is a disjunction $R_1 \vee R_2 \vee \dots \vee R_n$ in which each R_i is a literal and in which no atomic formula occurs twice.

A conjunction of clauses is said to be a wff in conjunctive normal form.

Let W be a wff in prenex normal form. Then the functional form of W is defined as follows:

Let the variables in the prefix of W (in order of occurrence) be $x_1, x_2, \dots, x_N$. Let the existentially quantified variables in the prefix be $x_{i_1}, x_{i_2}, \dots, x_{i_M}$. Then for every j , $1 \leq j \leq M$:

1. the quantifier $(\exists x_{i_j})$ is to be deleted from the prefix, and
2. each occurrence of x_{i_j} in the matrix of W is to be replaced by an occurrence of the term $f_{i_j}^q(x_{k_1}, x_{k_2}, \dots, x_{k_q})$, where $(x_{k_1}), (x_{k_2}), \dots, (x_{k_q})$, $q \geq 0$, are all the universal quantifiers that precede $(\exists x_{i_j})$ in the prefix of W and $f_{i_j}^q$

is the first q -adic function constant which does not occur in W and has not been used previously in this process.

We shall use the following known result:

W is satisfiable if and only if its functional form is satisfiable.

1.2 The Validity-Problem of the Predicate-Calculus

The validity problem of the predicate-calculus is undecidable.

That is, there can be no algorithm which takes as input any wff and in all cases terminates with a decision as to whether the wff is valid or not.

But, the validity-problem of the predicate-calculus is semi-decidable. That is, there are algorithms, called semi-decision procedures, which take as input any wff and: (1) If the wff is valid the algorithm will stop and say so; (2) If the wff is not valid the algorithm will never stop.

The algorithms have undergone successive reductions so that by now they have a simple structure. In this work, we shall use one recent algorithm based on the resolution principle (Robinson [1965]).

Though the validity-problem of the predicate-calculus is undecidable, there nevertheless exist classes of wff's for which the problem is decidable. For example, the validity-problem is decidable for the following three classes:⁽¹⁾

1. $W_1 = \{W \mid W \text{ is a wff in prenex-normal form, without function constants, and with prefix of the form } \forall \dots \forall \exists \dots \exists\}$,
2. $W_2 = \{W \mid W \text{ is a wff in prenex-normal form, without function constants, and with prefix of the form } \forall \dots \forall \exists \dots \forall\}$,
3. $W_3 = \{W \mid W \text{ is a wff in prenex-normal form, without function constants, and with prefix of the form } \forall \dots \forall \exists \dots \exists \forall \dots \forall\}$.

¹ See Ackermann [1954] or Church [1956] Section 46.

1.3 Directed Graphs

A directed graph G is an ordered triple $\langle V, L, A \rangle$ where:

1. V is a non-empty set of elements called the vertices of G ;
2. L is a non-empty set of elements called the labels of G ; and
3. A is a set of ordered triples (v, l, v') , where $v \in V$, $v' \in V$,
and $l \in L$. These triples are called the arcs of G .

If V and L are finite sets, G is called a finite directed graph.

Let $\alpha = (v, l, v')$ be an arc of a directed graph. Then, we define:

1. v - the initial vertex of the arc,
2. l - the label of the arc,
3. v' - the terminal vertex of the arc.

And we shall say that the arc α leads from the vertex v to the vertex v' .

Let v be a vertex of a directed graph. Then,

1. The number (finite or infinite) of arcs α , $\alpha \in A$, s.t. v is the initial vertex of α is called the out-degree of v .
2. The number (finite or infinite) of arcs α , $\alpha \in A$, s.t. v is the terminal vertex of α is called the in-degree of v .

A finite path of a graph G (path, for short) is a finite sequence of n arcs of G , $n \geq 1$,

$$(v_{i_1}, l_{i_1}, v_{i_2}), (v_{i_2}, l_{i_2}, v_{i_3}), \dots, (v_{i_n}, l_{i_n}, v_{i_{n+1}}),$$

s.t. the terminal vertex of each arc coincides with the initial vertex of the succeeding arcs.

We say that the vertices $v_{i_1}, v_{i_2}, \dots, v_{i_{n+1}}$ are on the path, and that the path joins the vertices v_{i_1} and $v_{i_{n+1}}$.

CHAPTER 2: DEFINITIONS

2.1 Abstract Programs

An abstract program (or program schema) AP consists of:

1. A finite directed graph $\langle V, L, A \rangle$, with
 - (a) exactly one vertex $S \in V$ with in-degree 0 (i.e., no arcs leading to S), called the start vertex;
 - (b) exactly one vertex $H \in V$ with out-degree 0 (i.e., no arcs leading from H), called the halt vertex; and
 - (c) every vertex $v \in V$ is on some path that joins S and H .
2. (a) a set of m , $m \geq 0$, distinct individual variables $\bar{y} = (y_1, y_2, \dots, y_m)$, called input variables; and
 (b) a set of n , $n \geq 1$, distinct individual variables $\bar{x} = (x_1, x_2, \dots, x_n)$, called program variables.
3. With each arc $\alpha = (v, l, v') \in A$ there is associated:
 - (a) a quantifier free wff φ_α called the test predicate of α ; and
 - (b) an n -tuple $\bar{t}_\alpha = (t_1^{(\alpha)}, t_2^{(\alpha)}, \dots, t_n^{(\alpha)})$ of terms called the assignment function of α .⁽¹⁾

The wff φ_α does not contain any predicate variables.

¹The Intended interpretation is

v : if φ_α then [replace simultaneously each variable x_i by $t_i^{(\alpha)}$ and go to v'].

The wff φ_α and the terms $t_i^{(\alpha)}$ do not contain individual variables other than $\bar{y}$ and $\bar{x}$.⁽¹⁾ If $v = S$ (i.e., α is an arc leading from the start vertex) the wff φ_α and the terms t_i^α do not contain the program variables $\bar{x}$.

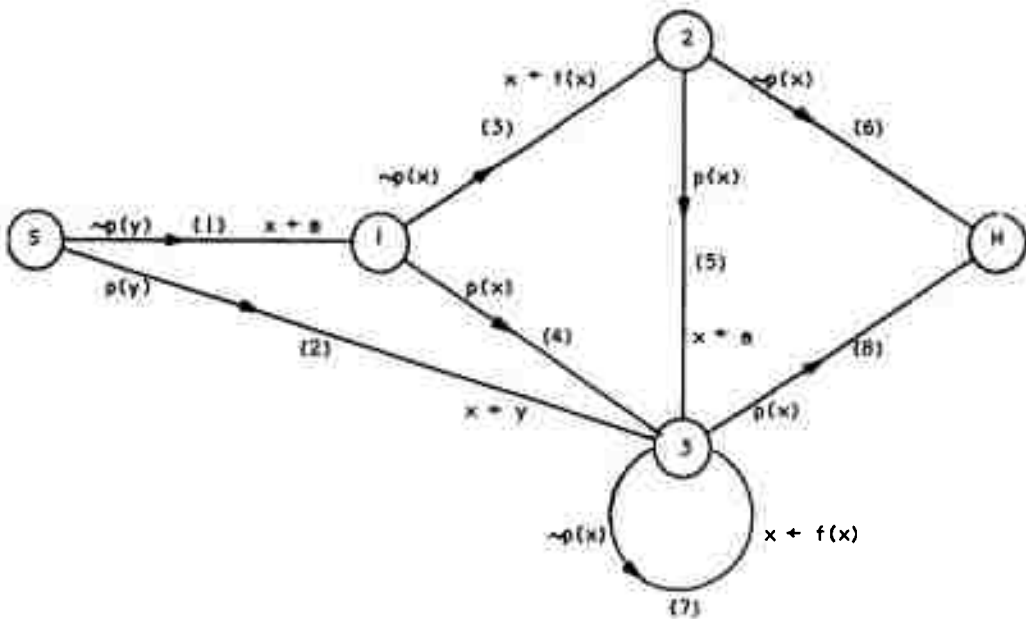
In addition, an abstract program should satisfy the following restriction:

4. For every vertex $v (v \neq H)$, if $\alpha_1, \alpha_2, \dots, \alpha_N$ is the set of all arcs leading from v , the set of the test predicates $\varphi_{\alpha_1}, \varphi_{\alpha_2}, \dots, \varphi_{\alpha_N}$ is
- (a) complete, i.e., $(\bar{x})(\bar{y}) [\varphi_{\alpha_1} \vee \varphi_{\alpha_2} \vee \dots \vee \varphi_{\alpha_N}]$ is valid, and
 - (b) mutually exclusive, i.e., $(\exists \bar{x})(\exists \bar{y}) [\varphi_{\alpha_i} \wedge \varphi_{\alpha_j}]$ is unsatisfiable for every pair (i, j) , $1 \leq i \neq j \leq N$.

¹We have restricted φ_α to be a quantifier free wff. However, all the theorems presented in this work are true also in the case when φ_α is any wff that does not contain free individual variables other than $\bar{y}$ and $\bar{x}$.

Example

The following diagram represents an abstract program. We shall refer later to this abstract program as AP*.



where

- a - individual constant,
- f - monadic function constant,
- p - monadic predicate constant,
- y - input variable,
- x - program variable.

2.2 Programs

An interpretation $\mathfrak{I}$ of an abstract program AP consists of a non-empty set of elements $D_{\mathfrak{I}}$ (called the domain of the interpretation) and assignments to the constants of AP:

1. To each function constant f_i^n which occurs in AP, we assign a total function of n variables ranging over $D_{\mathfrak{I}}$, whose values are in $D_{\mathfrak{I}}$. [If $n = 0$, the individual constant f_i^n is assigned some fixed element of $D_{\mathfrak{I}}$.]
2. To each predicate constant p_i^n which occurs in AP, we assign a total function of n variables ranging over $D_{\mathfrak{I}}$, whose values are T or F. [If $n = 0$, the propositional constant p_i^0 is assigned the value T or F.]

Let AP be an abstract program and $\mathfrak{I}$ an interpretation of AP. The pair $(AP, \mathfrak{I})$ is called a program.

Example

Consider the abstract program AP* of sec. 2.1. Let $\mathfrak{I}^*$ be the following interpretation of AP*:

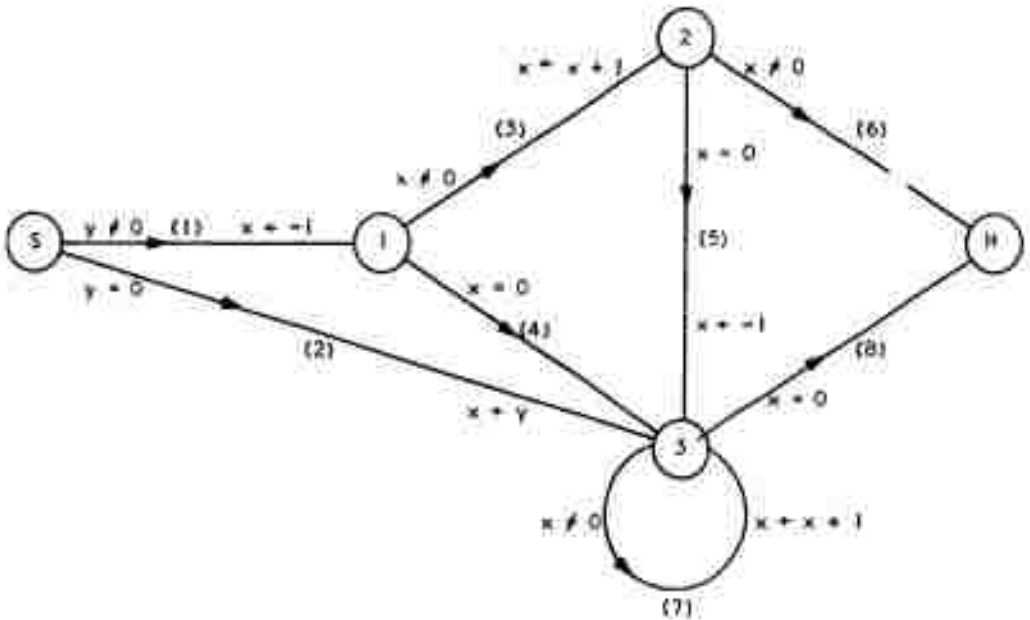
D is I (the domain of the integers),

$f(x)$ is $x + 1$,

$p(x)$ is $x = 0$, and

a is -1 .

Then the program (AP^*, S^*) can be represented by the diagram:



In order to give a rough idea of what will follow in the next section, let us only mention that the Algol meaning of this diagram is:

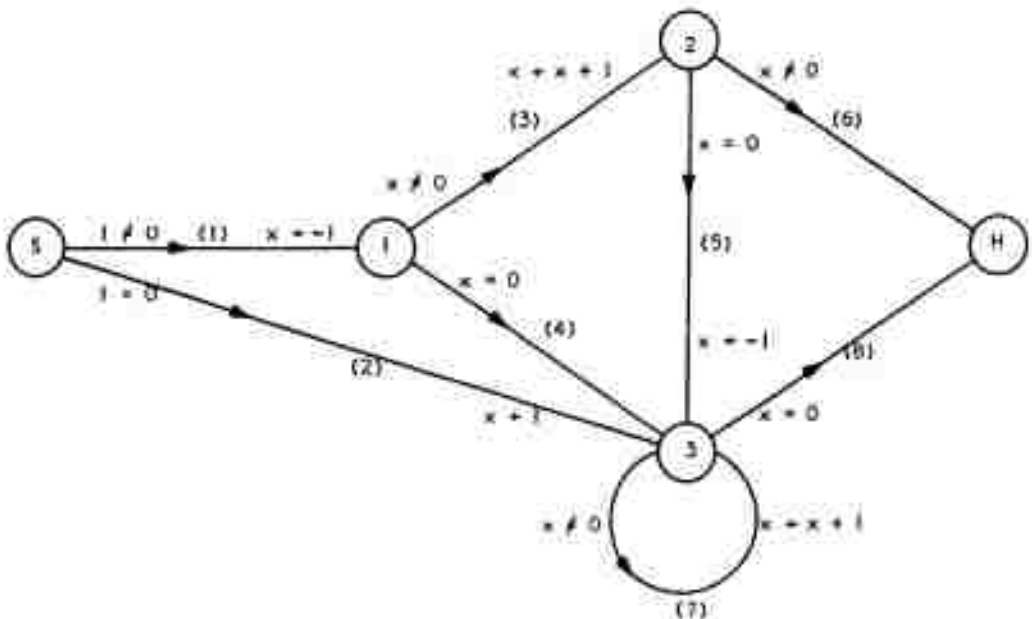
START: if $y=0$ then $[x \leftarrow y; \text{go to } 3]$ else $[x \leftarrow -1; \text{go to } 1]$;
 1: if $x=0$ then $[x \leftarrow x; \text{go to } 3]$ else $[x \leftarrow x + 1; \text{go to } 2]$;
 2: if $x=0$ then $[x \leftarrow -1; \text{go to } 3]$ else $[x \leftarrow x; \text{HALT}]$;
 3: if $x=0$ then $[x \leftarrow x; \text{HALT}]$ else $[x \leftarrow x + 1; \text{go to } 3]$.

2.3 Interpreted Programs

Let (AP, S) be a program. Then the result obtained by assigning values $\bar{y}, \bar{y} \in (D_S)^m$, for the input variables $\bar{y}$ of the program - is called the interpreted program $(AP, S, \bar{y})$.⁽¹⁾

Example

By assigning the value 1 to the input variable y of the program (AP^*, S^*) of sec. 2.2, we obtain the interpreted program $(AP^*, S^*, 1)$:



¹ Programs with no input variables (i.e., $m = 0$) will be considered as interpreted programs.

The interpreted program $(AP, \mathfrak{J}, \bar{\gamma})$ defines an execution sequence $\langle AP, \mathfrak{J}, \bar{\gamma} \rangle$ which is a (finite or infinite) sequence of triples

$$(l^{(1)}, v^{(1)}, \bar{x}^{(1)}), (l^{(2)}, v^{(2)}, \bar{x}^{(2)}), (l^{(3)}, v^{(3)}, \bar{x}^{(3)}), \dots$$

where,

1. $(l^{(j)}, v^{(j)}, \bar{x}^{(j)}) \in L \times V \times (D_{\mathfrak{J}})^n$ for every $j, j \geq 1$.
2. $(l^{(1)}, v^{(1)}, \bar{x}^{(1)})$ is the first triple in the sequence if and only if there exists an arc $\alpha = (S, l^{(1)}, v^{(1)}) \in A$ s.t.

$$\varphi_{\alpha}(\bar{\gamma}) = T \quad \text{and} \quad \bar{x}^{(1)} = \bar{t}_{\alpha}(\bar{\gamma})^{(1)}$$

3. $(l^{(j)}, v^{(j)}, \bar{x}^{(j)})$ and $(l^{(j+1)}, v^{(j+1)}, \bar{x}^{(j+1)})$ are two successive triples in the sequence if and only if there exists an arc $\alpha = (v^{(j)}, l^{(j+1)}, v^{(j+1)}) \in A$ s.t.

$$\varphi_{\alpha}(\bar{x}^{(j)}, \bar{\gamma}) = T \quad \text{and} \quad \bar{x}^{(j+1)} = \bar{t}_{\alpha}(\bar{x}^{(j)}, \bar{\gamma})^{(2)}$$

4. The sequence is finite and $(l^{(q)}, v^{(q)}, \bar{x}^{(q)})$, $q \geq 1$, is the last triple of the sequence if and only if $v^{(q)} = H$. In

¹ $\varphi_{\alpha}(\bar{\gamma})$ and $\bar{t}_{\alpha}(\bar{\gamma})$ stand for the result of substituting $\bar{\gamma}$ for $\bar{y}$ in φ_{α} and $\bar{t}_{\alpha}$.

² $\varphi_{\alpha}(\bar{x}^{(j)}, \bar{\gamma})$ and $\bar{t}_{\alpha}(\bar{x}^{(j)}, \bar{\gamma})$ stand for the result of substituting $\bar{x}^{(j)}$ for $\bar{x}$ and $\bar{\gamma}$ for $\bar{y}$ in φ_{α} and $\bar{t}_{\alpha}$.

this case $\bar{x}^{(q)}$ is called the value of the execution sequence $\langle AP, \mathfrak{J}, \bar{Y} \rangle$ and is denoted by val $\langle AP, \mathfrak{J}, \bar{Y} \rangle$.

In other words, execution always starts at the start vertex. On execution of the j^{th} step, $j \geq 1$, control moves along the arc $\alpha = (v^{(j-1)}, l^{(j)}, v^{(j)})$, where $v^{(0)} = S$, and φ_α represents the condition that this arc is entered. The value of each program variable x_i is replaced in the j^{th} step by the current value of $t_i^{(\alpha)}$, simultaneously. So, $\bar{x}^{(j)}$ represents the current value of the program variables $\bar{x}$ after executing the j^{th} step. Execution stops whenever control reaches the halt vertex.

Example

The Interpreted program $(AP^*, \mathfrak{J}^*, I)$ defines the following execution sequence $\langle AP^*, \mathfrak{J}^*, I \rangle$:

$(1, I, -1), (3, 2, 0), (5, 3, -1), (7, 3, 0), (8, H, 0)$.

Let $(AP, \mathfrak{J}, \bar{Y})$ be an interpreted program, and let $v \in V$ be any vertex of AP . Let δ be a specified total predicate from $(D_{\mathfrak{J}})^n$ into $\{T, F\}$. Then,

1. δ is called a valid predicate of v for $(AP, \mathfrak{J}, \bar{Y})$

if

$\forall \bar{\xi}, \bar{\xi} \in (D_{\mathcal{I}})^n$: if there exists a triple of the form $(l, v, \bar{\xi})$ in $\langle AP, \mathcal{I}, \bar{\gamma} \rangle$, for some $l \in L$, then $\delta(\bar{\xi}) = T$.

2. δ is called the minimal valid predicate of v for $(AP, \mathcal{I}, \bar{\gamma})$ if

$\forall \bar{\xi}, \bar{\xi} \in (D_{\mathcal{I}})^n$: $\delta(\bar{\xi}) = T$ if and only if there exists a triple of the form $(l, v, \bar{\xi})$ in $\langle AP, \mathcal{I}, \bar{\gamma} \rangle$, for some $l \in L$.

Example

The predicate $x \leq 0$ is a valid predicate, while the predicate $x = -1$ is the minimal valid predicate, of the vertex 1 for the interpreted program $(AP^*, \mathcal{I}^*, 1)$.

BLANK PAGE

CHAPTER 3: TERMINATION OF PROGRAMS AND ABSTRACT PROGRAMS

3.1 The Algorithm to Construct W_{AP}

In this section we shall describe an algorithm to construct from a given abstract program AP a wff W_{AP} , called the wff of AP. In section 3.3 we shall state results about the relation between AP and W_{AP} .

Algorithm 1

Let AP be any abstract program with program variables $\bar{x} = (x_1, x_2, \dots, x_n)$, $n \geq 1$, and input variables $(y_1, y_2, \dots, y_m)$, $m \geq 0$. We shall construct the wff W_{AP} in three steps:

Step 1

Associate with every vertex v_i of AP a predicate variable q_i , where the q_i 's are distinct n-adic predicate variables.

Step 2

Let $\alpha = (v_i, l, v_j)$ be any arc of AP.

In step 1 we have associated with the vertex v_i the predicate variable q_i , and with the vertex v_j the predicate variable q_j .

We shall define the wff W_α (the wff of the arc α) as

$$W_\alpha: q_i(\bar{x}) \wedge \varphi_\alpha \supset q_j(\bar{t}_\alpha).$$

But,

1. If $v_i = S$ (i.e., v_i is the start vertex of AP), then replace the occurrence of $q_i(\bar{x})$ in W_α by T, and
2. If $v_j = H$ (i.e., v_j is the halt vertex of AP), then replace the occurrence of $q_j(\bar{t}_\alpha)$ in W_α by F.

Step 3

Let $\alpha_1, \alpha_2, \dots, \alpha_N$ be the set of all the arcs of AP. Then define W_{AP} (the wff of AP) as:

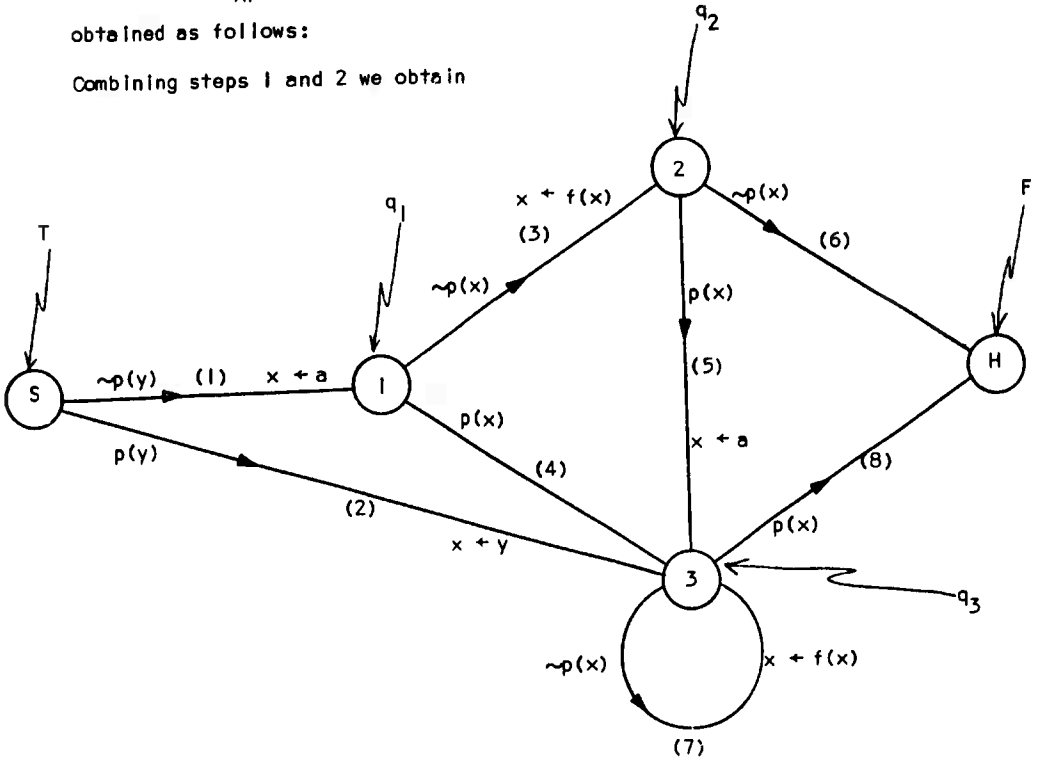
$$(\bar{x})[W_{\alpha_1} \wedge W_{\alpha_2} \wedge \dots \wedge W_{\alpha_N}]. \quad (1)$$

¹ Note that the input variables $\bar{y}$ are free variables in W_{AP} .

Example

The wff W_{AP^*} of the abstract program AP^* of sec. 2.1 will be obtained as follows:

Combining steps 1 and 2 we obtain



$$W_1: T \wedge \sim p(y) \supset q_1(a)$$

$$W_2: T \wedge p(y) \supset q_3(y)$$

$$W_3: q_1(x) \wedge \sim p(x) \supset q_2(f(x))$$

$$W_4: q_1(x) \wedge p(x) \supset q_3(x)$$

$$W_5: q_2(x) \wedge p(x) \supset q_3(a)$$

$$W_6: q_2(x) \wedge \sim p(x) \supset F$$

$$W_7: q_3(x) \wedge \sim p(x) \supset q_3(f(x))$$

$$W_8: q_3(x) \wedge p(x) \supset F$$

Then by step 3 it follows that,

$$W_{AP*}: (x)[W_1 \wedge W_2 \wedge W_3 \wedge W_4 \wedge W_5 \wedge W_6 \wedge W_7 \wedge W_8]$$

3.2 Termination of Programs

Definition 1

The program $(AP, \mathfrak{J})$ is said to terminate if $\forall \bar{y}, \bar{y} \in (D_{\mathfrak{J}})^m$, the execution sequence $\langle AP, \mathfrak{J}, \bar{y} \rangle$ is finite.

We are ready now to state the main result of this chapter.

Theorem 1

The program $(AP, \mathfrak{J})$ terminates

if and only if

$(W_{AP}, \mathfrak{J})$ is unsatisfiable [or equivalently, $(\sim W_{AP}, \mathfrak{J})$ is valid].

Proof

We shall prove that the program $(AP, \mathfrak{J})$ does not terminate if and only if $(W_{AP}, \mathfrak{J})$ is satisfiable.

1. $(AP, \mathfrak{J})$ does not terminate $\Rightarrow (W_{AP}, \mathfrak{J})$ is satisfiable.

If the program $(AP, \mathfrak{J})$ does not terminate, there exists a $\bar{y}, \bar{y} \in (D_{\mathfrak{J}})^m$, such that the execution sequence $\langle AP, \mathfrak{J}, \bar{y} \rangle$ is infinite.

Let us assign to each predicate variable q_i in W_{AP} , the minimal valid predicate of the vertex v_i for the interpreted program $(AP, \mathfrak{J}, \bar{y})$.

Note that since the execution sequence $\langle AP, \mathfrak{J}, \bar{y} \rangle$ is infinite, i.e., control never reaches the halt vertex, it follows that the predicate F is the minimal valid predicate of the vertex H for the interpreted program $(AP, \mathfrak{J}, \bar{y})$.

Let Γ consist of the above assignments for the q_i 's and with $\bar{y}$ assigned to $\bar{y}$. Following the construction of W_{AP} (see Algorithm 1), it is clear that the value of $(W_{AP}, \mathcal{J}, \Gamma)$ is T, i.e., $(W_{AP}, \mathcal{J})$ is satisfiable, and this completes the proof in one direction.

2. $(W_{AP}, \mathcal{J})$ is satisfiable $\Rightarrow$ $(AP, \mathcal{J})$ does not terminate.

If $(W_{AP}, \mathcal{J})$ is satisfiable, it means that there exists an assignment Γ for $(W_{AP}, \mathcal{J})$ such that the value of $(W_{AP}, \mathcal{J}, \Gamma)$ is T. Γ consists of assignments of specified total predicates δ_i , mapping $(D_{\mathcal{J}})^n$ into $\{T, F\}$, for the predicate variables q_i , and an assignment $\bar{y}, \bar{y} \in (D_{\mathcal{J}})^m$, for the free variables $\bar{y}$.

By the construction of W_{AP} (see Algorithm 1), this implies that each δ_i is a valid predicate of the vertex v_i for $(AP, \mathcal{J}, \bar{y})$, and therefore that F is a valid predicate of the halt vertex for $(AP, \mathcal{J}, \bar{y})$.

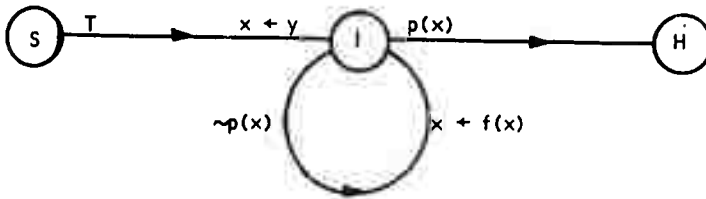
This implies that the execution sequence $\langle AP, \mathcal{J}, \bar{y} \rangle$ is infinite (i.e., execution does not reach the halt vertex). So, $(AP, \mathcal{J})$ does not terminate.

q.e.d.

Example

Let us consider the program $(\tilde{A}\tilde{P}, \tilde{\mathfrak{I}})$, where

1. the abstract program $\tilde{A}\tilde{P}$ is



and

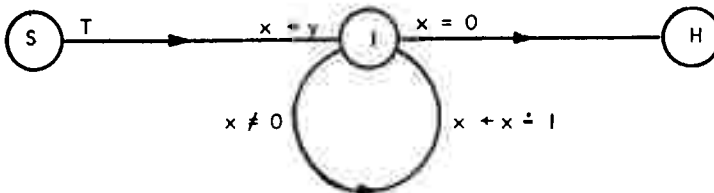
2. the interpretation $\tilde{\mathfrak{I}}$ is

$D_{\tilde{\mathfrak{I}}} = \mathbb{I}^+$ (i.e., the domain of the non-negative integers),

$p(x)$ is $x = 0$, and

$f(x)$ is $x \dot{-} 1$, where $x \dot{-} 1$ is defined as $\begin{cases} x-1 & \text{if } x > 0 \\ 0 & \text{if } x = 0. \end{cases}$

The program $(\tilde{A}\tilde{P}, \tilde{\mathfrak{I}})$ can be represented by the domain $D_{\tilde{\mathfrak{I}}} = \mathbb{I}^+$ and the diagram



Using Algorithm 1 we can construct $W_{AP}^{\sim}$, which is

$$(x) \{ [T \wedge T \supset q_1(y)] \\ \wedge [q_1(x) \wedge \sim p(x) \supset q_1(f(x))] \\ \wedge [q_1(x) \wedge p(x) \supset F] \}.$$

The pair $(W_{AP}^{\sim}, \tilde{S})$ can be represented by the domain $D_{\tilde{S}} = I^+$ and

$$W_{AP}^{\sim}: (x) \{ [T \wedge T \supset q_1(y)] \\ \wedge [q_1(x) \wedge x \neq 0 \supset q_1(x \pm 1)] \\ \wedge [q_1(x) \wedge x = 0 \supset F] \}.$$

We shall prove that the program $(AP^{\sim}, \tilde{S})$ terminates by using Theorem 1, i.e., by proving that $(W_{AP}^{\sim}, \tilde{S})$ unsatisfiable.

We shall use the first order theory N , which formalizes elementary number theory. We assume that the reader is familiar with this theory⁽¹⁾.

The theorems of N that we shall use are:

$$T1: \exists x_1 [q_1(x_1) \supset \exists x_2 [q_1(x_2) \wedge (x_3) [x_3 < x_2 \supset \sim q_1(x_3)]]]$$

(an instance of the Least-number Principle), and

$$T2: (x) [x \neq 0 \supset x \pm 1 < x].$$

Thus, in order to prove that $(W_{AP}^{\sim}, \tilde{S})$ is unsatisfiable, we shall prove that $W_{AP}^{\sim} \wedge T1 \wedge T2$ is unsatisfiable (considering $x = 0$, $x < y$ and $x \pm 1$ just as symbols, i.e., the predicates $x = 0$ and $x < y$ as predicate constants and the function $x \pm 1$ as function constant).

¹See Kleene [1950] Chapter 8, Mendelson [1964] Chapter 3, or Kleene [1967] Section 38.

The Proof:

The prenex normal form of $\mathcal{W}_{AP}^3 \wedge T_1 \wedge T_2$ is:

$$\begin{aligned} \exists x_2)(x_1)(x_3)(x)\{ & \quad q_1(y) \\ & \wedge [q_1(x) \wedge x \neq 0 \supset q_1(x \div 1)] \\ & \wedge [q_1(x) \wedge x = 0 \supset F] \\ & \wedge [q_1(x_1) \supset [q_1(x_2) \wedge [x_3 < x_2 \supset \sim q_1(x_3)]]] \\ & \wedge [x \neq 0 \supset x \div 1 < x]\}. \end{aligned}$$

Then by changing the matrix to conjunctive normal form and replacing x_2 by a and y by b (a and b are individual variables), we obtain the wff W^* :

$$\begin{aligned} (x_1)(x_3)(x)\{ & \quad q_1(b) \\ & \wedge [\sim q_1(x) \vee x = 0 \vee q_1(x \div 1)] \\ & \wedge [\sim q_1(x) \vee x \neq 0] \\ & \wedge [\sim q_1(x_1) \vee q_1(a)] \\ & \wedge [\sim q_1(x_1) \vee x_3 \neq a \vee \sim q_1(x_3)] \\ & \wedge [x = 0 \vee x \div 1 < x]\}. \end{aligned}$$

Clearly, $\mathcal{W}_{AP}^3 \wedge T_1 \wedge T_2$ is satisfiable if and only if W^* is satisfiable.

We are going to prove that W^* is unsatisfiable by using the resolution principle. We assume that the reader is familiar with this technique (see Robinson [1965]).

The list of clauses is:

1. $q_1(b)$
2. $\sim q_1(x), x = 0, q_1(x \div 1)$

3. $\sim q_1(x), x \neq 0$
4. $\sim q_1(x_1), q_1(a)$
5. $\sim q_1(x_1), x_3 \neq a, \sim q_1(x_3)$
6. $x = 0, x \neq 1 < x.$

Then by resolving we obtain:

- | | |
|--------------------------|--|
| 7. $q_1(a)$ | by 1 and 4 ($x_1 = a$) |
| 8. $a \neq 0$ | by 3 ($x = a$) and 7 |
| 9. $q_1(a \neq 1)$ | by 2 ($x = a$), 7 and 8 |
| 10. $a \neq 1 < a$ | by 6 ($x = a$) and 8 |
| 11. $\sim q_1(a \neq 1)$ | by 5 ($x_1 = a, x_3 = a \neq 1$), 7 and 10 |
| 12. $\square$ | by 9 and 11 |

So, by resolving, we inferred the empty clause $\square$, which implies that W^* is unsatisfiable, i.e., (W_{APV}) is unsatisfiable. Therefore it follows, by Theorem 1, that the program (APV) terminates.

3.3 Termination of Abstract Programs

Definition 2

An abstract program AP is said to terminate if for every interpretation $\mathfrak{I}$, the program $(AP, \mathfrak{I})$ terminates.

The following theorem follows from Theorem 1 and Definition 2.

Theorem 2

An abstract program AP terminates
if and only if

W_{AP} is unsatisfiable [or equivalently, $\sim W_{AP}$ is valid].

Proof

AP terminates,

if and only if (follows by Definition 2)

for every interpretation $\mathfrak{I}$, the program $(AP, \mathfrak{I})$ terminates,

if and only if (follows by Theorem 1)

for every interpretation $\mathfrak{I}$, $(W_{AP}, \mathfrak{I})$ is unsatisfiable,

if and only if

W_{AP} is unsatisfiable.

q.e.d.

Theorem 2 transforms completely the problem of termination of abstract programs to an equivalent problem in logic. This enables us to obtain many results about the problem of termination of abstract programs, just by using well-known results in logic. The following example illustrates one of them. Other results are presented in the next section.

Example

We shall prove that the abstract program AP^* (see sec. 2.1) terminates, by using Theorem 2, i.e., by proving that W_{AP^*} is unsatisfiable.

In sec. 3.1 we have already constructed W_{AP^*} , which is

$$\begin{aligned}
 (x)\{ & [\quad T \wedge \sim p(y) \supset q_1(a) \quad] \\
 & \wedge [\quad T \wedge p(y) \supset q_3(y) \quad] \\
 & \wedge [q_1(x) \wedge \sim p(x) \supset q_2(f(x))] \\
 & \wedge [q_1(x) \wedge p(x) \supset q_3(x) \quad] \\
 & \wedge [q_2(x) \wedge p(x) \supset q_3(a) \quad] \\
 & \wedge [q_2(x) \wedge \sim p(x) \supset F \quad] \\
 & \wedge [q_3(x) \wedge \sim p(x) \supset q_3(f(x))] \\
 & \wedge [q_3(x) \wedge p(x) \supset F \quad] \}.
 \end{aligned}$$

By changing the matrix of W_{AP^*} to conjunctive normal form, and replacing y by b (where b is a new individual variable), we obtain

W'_{AP^*} :

$$\begin{aligned}
 (x)\{ & [\quad p(b, \vee q_1(a) \quad] \\
 & \wedge [\quad \sim p(b) \vee q_3(b) \quad] \\
 & \wedge [\sim q_1(x) \vee p(x) \vee q_2(f(x))] \\
 & \wedge [\sim q_1(x) \vee \sim p(x) \vee q_3(x) \quad] \\
 & \wedge [\sim q_2(x) \vee \sim p(x) \vee q_3(a) \quad] \\
 & \wedge [\sim q_2(x) \vee p(x) \quad]
 \end{aligned}$$

$$\begin{aligned} & \wedge [\sim q_3(x) \vee p(x) \vee q_3(f(x))] \\ & \wedge [\sim q_3(x) \vee \sim p(x) \quad]\}. \end{aligned}$$

Clearly, W_{AP*}^1 is satisfiable if and only if W_{AP*} is satisfiable.

We are going to prove that W_{AP*}^1 is unsatisfiable by using the resolution principle. We assume that the reader is familiar with this technique (see Robinson [1965]).

The list of clauses is:

1. $p(b), q_1(a)$
2. $\sim p(b), q_3(b)$
3. $\sim q_1(x), p(x), q_2(f(x))$
4. $\sim q_1(x), \sim p(x), q_3(x)$
5. $\sim q_2(x), \sim p(x), q_3(a)$
6. $\sim q_2(x), p(x)$
7. $\sim q_3(x), p(x), q_3(f(x))$
8. $\sim q_3(x), \sim p(x)$.

Then by resolving we obtain

- | | |
|--------------------------------------|----------------------------|
| 9. $\sim p(b)$ | by 2 & 8 (with $x = b$) |
| 10. $q_1(a)$ | by 1 & 9 |
| 11. $\sim q_1(x), q_2(f(x)), q_3(x)$ | by 3 & 4 |
| 12. $q_2(f(a)), q_3(a)$ | by 10 & 11 (with $x = a$) |
| 13. $\sim q_2(x), q_3(a)$ | by 5 & 6 |

- | | |
|-----------------------------------|-------------------------------|
| 14. $\frac{q_3(a)}{}$ | by 12 & 13 (with $x = f(a)$) |
| 15. $\sim q_3(x), q_3(f(x))$ | by 7 & 8 |
| 16. $q_3(f(a))$ | by 14 & 15 (with $x = a$) |
| 17. $p(a), q_2(f(a))$ | by 3 (with $x = a$) & 10 |
| 18. $p(a), p(f(a))$ | by 6 (with $x = f(a)$) & 17 |
| 19. $\sim q_3(a), p(f(a))$ | by 8 (with $x = a$) & 18 |
| 20. $\sim q_3(a), \sim q_3(f(a))$ | by 8 (with $x = f(a)$) & 19 |
| 21. $\frac{\sim q_3(a)}{}$ | by 16 & 20 |
| 22. $\square$ | by 14 & 21. |

So, by resolving, we inferred the empty clause $\square$, which implies that W_{AP^*} is unsatisfiable, i.e., W_{AP^*} is unsatisfiable. Therefore it follows, by Theorem 2, that AP^* terminates.

3.4 The Termination Problem of Abstract Programs

It is a well-known result that the termination problem of abstract programs is undecidable (see Luckham, Park and Paterson [1967]). That is, there can be no algorithm which takes as input any abstract program AP and in all cases stops with a decision as to whether the abstract program terminates or not.

But,

Corollary 1: The termination problem of abstract programs is semi-decidable.

That is, there are algorithms (called semi-decision procedures), which take as input any abstract program AP, and

1. If AP terminates, the algorithm will stop and say so;
2. If AP does not terminate, the algorithm will never stop.

Since the validity problem of the predicate calculus is semi-decidable, Corollary 1 follows directly by Theorem 2.

Moreover, any known semi-decision procedure for solving the validity problem of the predicate calculus can be used, together with Algorithm 1, as a semi-decision procedure for solving the termination problem of abstract programs. In fact, in sec. 3.3, we have used the resolution principle, which is a semi-decision procedure for solving

the validity problem of the predicate calculus, to prove the termination of the abstract program AP^* of sec. 2.1.

Though the termination problem of abstract programs is undecidable, there nevertheless exist subclasses of abstract programs for which the termination problem is decidable.

Corollary 2

The termination problem for the following classes is decidable:

1. $C_1 = \{AP \mid AP \text{ is an abstract program without function constants } f_i^n, n \geq 1\},$
2. $C_2 = \{AP \mid AP \text{ is an abstract program which has only one program variable } x \text{ (i.e., } n = 1), \text{ and all the occurrences of function constants in } AP \text{ are in terms of the form } f_i^0 \text{ or } f_i^1(x)\}.$
3. $C_3 = \{AP \mid AP \text{ is an abstract program which has only two program variables } x_1 \text{ and } x_2 \text{ (i.e., } n = 2), \text{ and all the occurrences of function constants in } AP \text{ are in terms of the form } f_i^0 \text{ or } f_i^2(x_1, x_2)\}.$

Proof

For each i , $1 \leq i \leq 3$, the decidability of the termination problem for the class C_i follows, by using Theorem 2, from the decidability of the validity problem for the class W_i (see sec. 1.2).

Let us prove this assertion for $l = 2$, i.e., we shall prove the decidability of the termination problem for the class C_2 by using Theorem 2 and the decidability of the validity problem for the class W_2 , where

$W_2 = \{W \mid W \text{ is a wff in prenex normal form, without function constants, and with prefix of the form } \forall \dots \forall \exists \forall \dots \forall\}$.

The proof of the assertion for the other classes is similar.

Let AP be any member of the class C_2 , i.e., AP is an abstract program which has only one program variable x (i.e., $n = 1$), and all the occurrences of function constants in AP are in terms of the form $f_1^0, f_2^0, \dots, f_k^0$ and $f_1^1(x), f_2^1(x), \dots, f_l^1(x)$ ($k, l \geq 0$).

Then W_{AP} is of the form $(x)M$, where M is a quantifier free wff and all the occurrences of function constants in M are in terms of the form $f_1^0, f_2^0, \dots, f_k^0$ and $f_1^1(x), f_2^1(x), \dots, f_l^1(x)$.

Let W'_{AP} be the wff $(\exists w_1) \dots (\exists w_k)(x)(\exists z_1) \dots (\exists z_l)M'$, where M' is the result of substituting w_i , $i = 1, 2, \dots, k$, for each occurrence of f_i^0 in M and substituting z_i , $i = 1, 2, \dots, l$, for each occurrence of $f_i^1(x)$ in M , i.e., M' contains no function constants.

W'_{AP} is satisfiable if and only if W_{AP} is satisfiable, since W_{AP} is the functional form of W'_{AP} .

Let W''_{AP} be the wff $(w_1) \dots (w_k) (\exists x) (z_1) \dots (z_l) [\sim M']$, i.e., W''_{AP} is just $\sim W'_1$. Clearly, W''_{AP} is valid if and only if W'_1 is unsatisfiable.

Since W''_{AP} is in prenex normal form, without function constants, and with prefix of the form $\forall \dots \forall \exists \forall \dots \forall$, it follows that W''_{AP} is a member of W_2 . But the validity problem for the class W_2 is decidable, so it is decidable whether W''_{AP} is valid or not.

Since by the previous assertions W''_{AP} is valid if and only if AP terminates, this implies that it is decidable whether AP terminates or not.

q.e.d.

Known decision procedures for solving the validity problem for the class W_1 can be used, together with Algorithm 1, as a decision procedure for solving the termination problem for the class C_1 . For example, we can use Friedman's semi-decision procedure for the predicate calculus (see Friedman [1963]), which is a decision procedure for the classes W_1 , W_2 , and W_3 .

Note that the abstract program AP^* of sec. 2.1 belongs to the class C_2 .

CHAPTER 4: EQUIVALENCE OF PROGRAMS AND ABSTRACT PROGRAMS

4.1 The Algorithm to Construct $W_{AP,AP'}$ Definition 3

Two abstract programs AP and AP' are said to be comparable if

1. they have the same set of program variables $\bar{x} = (x_1, \dots, x_n)$,
and
2. they have the same set of input variables $\bar{y} = (y_1, \dots, y_m)$.⁽¹⁾

In this section we shall first describe an algorithm to construct from two given comparable abstract programs AP and AP', a wff $W_{AP,AP'}$ (the wff of AP and AP'). In section 4.3 we shall state results about the relation between AP, AP' and $W_{AP,AP'}$.

Algorithm 2

Let AP and AP' be any two comparable abstract programs. We shall construct the wff $W_{AP,AP'}$ in four steps:

¹Note that any two abstract programs can be considered as satisfying condition 2, for if the two abstract programs do not have the same sets of input variables, just add to each program an appropriate set of dummy input variables.

Step 1

Associate with every vertex v_i of AP a predicate variable q_i [we shall denote by q_H the predicate variable associated with the halt vertex H of AP], and associate with every vertex v_i' of AP' a predicate variable q_i' , where all the q_i and the q_i' are distinct.

Step 2

Let $\alpha = (v_i, l, v_j)$ be any arc of AP.

In step 1 we have associated with the vertex v_i the predicate variable q_i , and with the vertex v_j the predicate variable q_j .

We shall define the wff W_α (the wff of the arc α) as

$$W_\alpha: q_i(\bar{x}) \wedge \varphi_\alpha \supset q_j(\bar{t}_\alpha).$$

But,

if $v_i = S$ (i.e., v_i is the start vertex of AP), then replace the occurrence of $q_i(\bar{x})$ in W_α by T.

Step 3

Let $\alpha' = (v_i', l, v_j')$ be any arc of AP'.

In step 1 we have associated with the vertex v_i' the predicate variable q_i' , and with the vertex v_j' the predicate variable q_j' .

We shall define the wff $W_{\alpha'}$ (the wff of the arc α') as

$$W_{\alpha'}: q_i'(\bar{x}) \wedge \varphi_{\alpha'} \supset q_j'(\bar{t}_{\alpha'}).$$

But,

1. if $v_i' = S'$ (i.e., v_i' is the start vertex of AP'), then replace the occurrence of $q_i'(\bar{x})$ in $W_{\alpha'}$ by T, and

2. if $v_j^1 = H^1$ (i.e., v_j^1 is the halt vertex of AP^1), then replace the occurrence of $q_j^1(\bar{t}_{\alpha^1})$ in W_{α^1} by $\sim q_H(\bar{t}_{\alpha^1})$.

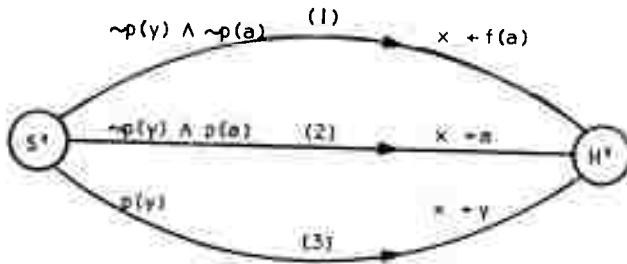
Step 4

Let $\alpha_1, \alpha_2, \dots, \alpha_N$ be the set of all the arcs of AP , and $\alpha_1', \alpha_2', \dots, \alpha_M'$ be the set of all the arcs of AP^1 . Then define W_{AP, AP^1} as

$$W_{AP, AP^1} := (\bar{x}) [W_{\alpha_1} \wedge W_{\alpha_2} \wedge \dots \wedge W_{\alpha_N} \wedge W_{\alpha_1'} \wedge W_{\alpha_2'} \wedge \dots \wedge W_{\alpha_M'}]. \quad (1)$$

Example

Consider the abstract program AP^{**} :

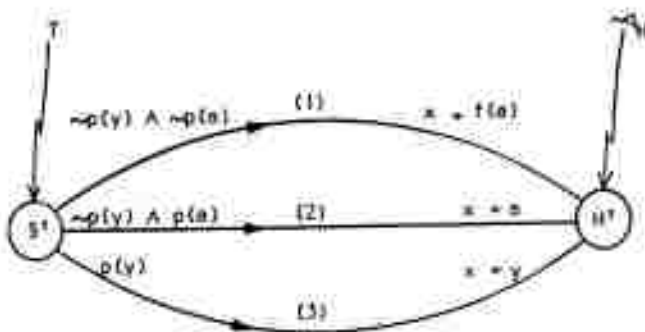
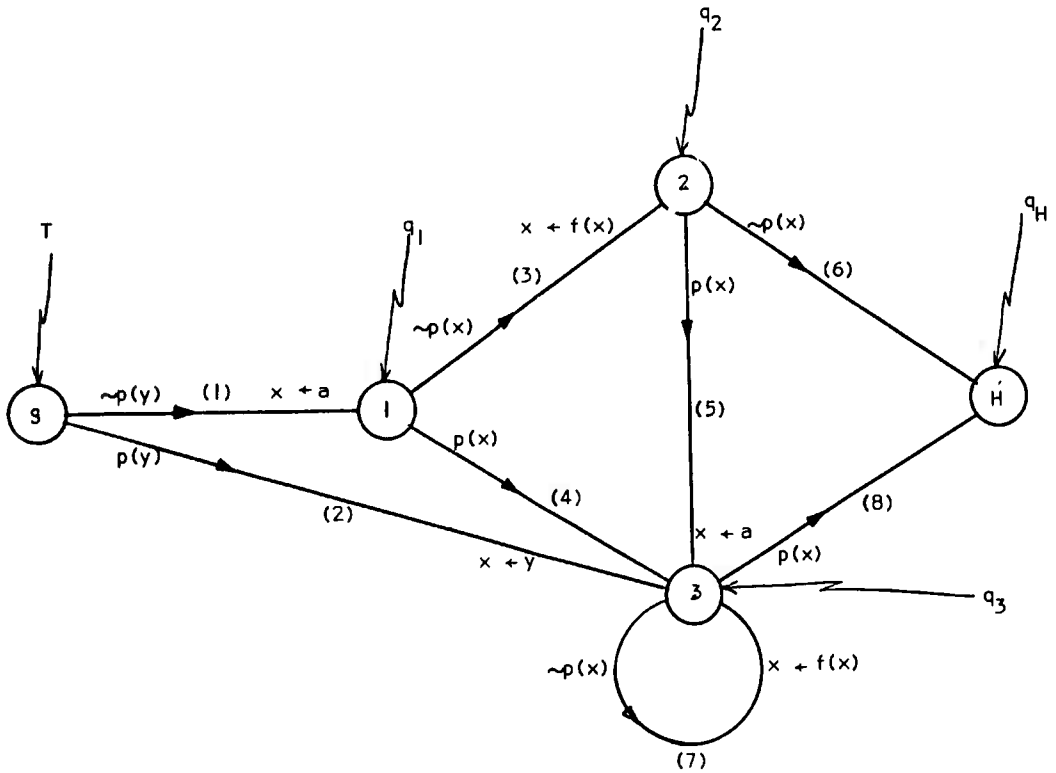


where,

- a - individual variable,
- f - monadic function constant,
- p - monadic predicate constant,
- y - input variable,
- x - program variable.

¹ Note that the input variables of AP and AP^1 are free variables in W_{AP, AP^1} .

Using Algorithm 2 we shall construct the wff $W_{AP^*, AP^{**}}$, where AP^* is the abstract program that was presented in sec. 2.1.



$$\begin{aligned}
 W_{AP^*, AP^{**}}: (x) \{ & [\quad T \quad \wedge \quad \sim p(y) \supset q_1(a)] \\
 & \wedge [\quad T \quad \wedge \quad p(y) \supset q_3(y)] \\
 & \wedge [q_1(x) \wedge \sim p(x) \supset q_2(f(x))] \\
 & \wedge [q_1(x) \wedge p(x) \supset q_3(x)] \\
 & \wedge [q_2(x) \wedge p(x) \supset q_3(a)] \\
 & \wedge [q_2(x) \wedge \sim p(x) \supset q_H(x)] \\
 & \wedge [q_3(x) \wedge \sim p(x) \supset q_3(f(x))] \\
 & \wedge [q_3(x) \wedge p(x) \supset q_H(x)] \\
 & \wedge [\quad T \quad \wedge \quad \sim p(y) \wedge \sim p(a) \supset \sim q_H(f(a))] \\
 & \wedge [\quad T \quad \wedge \quad \sim p(y) \wedge p(a) \supset \sim q_H(a)] \\
 & \wedge [\quad T \quad \wedge \quad p(y) \supset \sim q_H(y)] \}.
 \end{aligned}$$

4.2 Equivalence of Programs

Definition 4

Let AP and AP' be any two comparable abstract programs.

Let $\mathfrak{I}$ be an interpretation that contains assignments for all the constants that occur in AP or AP'.

Then the programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are said to be comparable.

Definition 5

Two comparable programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are said to be equivalent, if

$\bar{w}, \bar{y} \in (D_{\mathfrak{I}})^m$, both execution sequences $\langle AP, \mathfrak{I}, \bar{w} \rangle$ and $\langle AP', \mathfrak{I}, \bar{y} \rangle$ are finite and $\text{val } \langle AP, \mathfrak{I}, \bar{w} \rangle = \text{val } \langle AP', \mathfrak{I}, \bar{y} \rangle$.

Theorem 3

Two comparable programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are equivalent, if and only if

$(w_{AP, AP', \mathfrak{I}})$ is unsatisfiable [or equivalently, $(\sim w_{AP, AP', \mathfrak{I}})$ is valid].

Proof

We shall prove that:

$\bar{w}, \bar{y} \in (D_{\mathfrak{I}})^m$, such that 1. $\langle AP, \mathfrak{I}, \bar{w} \rangle$ is infinite,
or 2. $\langle AP', \mathfrak{I}, \bar{y} \rangle$ is infinite,
or 3. both $\langle AP, \mathfrak{I}, \bar{w} \rangle$ and $\langle AP', \mathfrak{I}, \bar{y} \rangle$ are finite,
and $\text{val } \langle AP, \mathfrak{I}, \bar{w} \rangle \neq \text{val } \langle AP', \mathfrak{I}, \bar{y} \rangle$,

if and only if

$(W_{AP, AP', \mathfrak{S}})$ is satisfiable.

(i) $\Rightarrow$

We have to consider three cases:

1. If the execution sequence $\langle AP, \mathfrak{S}, \bar{Y} \rangle$ is infinite, then $(W_{AP, AP', \mathfrak{S}})$ is satisfiable, since the value of $(W_{AP, AP', \mathfrak{S}, \Gamma})$ is T, where Γ consists of the following assignments:

- (a) $\bar{Y}$ assigned to $\bar{y}$,
- (b) to each occurrence of q_i in $W_{AP, AP'}$ assign the minimal valid predicate of v_i for $(AP, \mathfrak{S}, \bar{Y})$, and
- (c) to each occurrence of $q_i^!$ in $W_{AP, AP'}$ assign the minimal valid predicate of $v_i^!$ for $(AP', \mathfrak{S}, \bar{Y})$.

The result then follows from the construction of $W_{AP, AP'}$ (Algorithm 2). Note that, since $\langle AP, \mathfrak{S}, \bar{Y} \rangle$ is infinite, the minimal valid predicate of H for $(AP, \mathfrak{S}, \bar{Y})$ is F, i.e., by our assignment $q_H \equiv F$, and therefore $\sim q_H \equiv T$.

2. If the execution sequence $\langle AP', \mathfrak{S}, \bar{Y} \rangle$ is infinite, then $(W_{AP, AP', \mathfrak{S}})$ is satisfiable, since the value of $(W_{AP, AP', \mathfrak{S}, \Gamma})$ is T, where Γ consists of the following assignments:

- (a) $\bar{Y}$ assigned to $\bar{y}$,
- (b) to each occurrence of q_i [except q_H] in $W_{AP, AP'}$ assign the minimal valid predicate of v_i for $(AP, \mathfrak{S}, \bar{Y})$,
- (c) to each occurrence of $q_i^!$ in $W_{AP, AP'}$ assign the minimal valid predicate of $v_i^!$ for $(AP', \mathfrak{S}, \bar{Y})$, and
- (d) $q_H \equiv T$.

The result then follows from the construction of $W_{AP, AP'}$ (Algorithm 2). Note that $\sim q_H \equiv F$, and since $\langle AP', \bar{y} \rangle$ is infinite, F is the minimal valid predicate of H' for $(AP', \bar{y})$.

3. If both the execution sequences $\langle AP, \bar{y} \rangle$ and $\langle AP', \bar{y} \rangle$ are finite and $\text{val } \langle AP, \bar{y} \rangle \neq \text{val } \langle AP', \bar{y} \rangle$ then $(W_{AP, AP'}, \bar{y})$ is satisfiable, since the value of $(W_{AP, AP'}, \bar{y}, \Gamma)$ is T , where Γ consists of the following assignments:

- (a) $\bar{y}$ assigned to $\bar{y}$,
- (b) to each occurrence of q_i in $W_{AP, AP'}$ assign the minimal valid predicate of v_i for $(AP, \bar{y})$, and
- (c) to each occurrence of q'_i in $W_{AP, AP'}$ assign the minimal valid predicate of v'_i for $(AP', \bar{y})$.

The result then follows from the construction of $W_{AP, AP'}$ (Algorithm 2). Note that we assigned to q_H the minimal valid predicate δ of H for $(AP, \bar{y})$, i.e., $\delta(x) = T$ if and only if $\bar{x} = \text{val } \langle AP, \bar{y} \rangle$. Now, since $\text{val } \langle AP, \bar{y} \rangle \neq \text{val } \langle AP', \bar{y} \rangle$, it follows that $\delta(\text{val } \langle AP', \bar{y} \rangle) = F$, i.e., $\sim \delta(\text{val } \langle AP', \bar{y} \rangle) = T$.

(ii) =

We shall prove that if $(W_{AP, AP'}, \bar{y})$ is satisfiable with $\bar{y}, \bar{y} \in (D_y)^m$, assigned to $\bar{y}$, and both execution sequences $\langle AP, \bar{y} \rangle$ and $\langle AP', \bar{y} \rangle$ are finite, then $\text{val } \langle AP, \bar{y} \rangle \neq \text{val } \langle AP', \bar{y} \rangle$.

If $(W_{AP, AP'}, \bar{y})$ is satisfiable with $\bar{y}$ assigned to $\bar{y}$, it means that there exist an assignment Γ such that $(W_{AP, AP'}, \bar{y}, \Gamma)$ is T , where Γ

consists of the assignment of $\bar{y}$ to $\bar{y}$ and assignments of specified total predicates δ_i and δ'_i (mapping $(D_{\mathfrak{S}})^n$ into $\{T, F\}$) for q_i and q'_i respectively.

By the construction of $W_{AP, AP'}$ (Algorithm 2), this implies that each δ_i is a valid predicate of the vertex v_i for $(AP, \mathfrak{S}, \bar{y})$, especially δ_H is a valid predicate of the halt vertex H for $(AP, \mathfrak{S}, \bar{y})$, and therefore $\delta_H(\text{val } \langle AP, \mathfrak{S}, \bar{y} \rangle) = T$. Moreover, each δ'_i is a valid predicate of the vertex v'_i for $(AP', \mathfrak{S}, \bar{y})$, and $\sim\delta_H$ is a valid predicate of the halt vertex H' for $(AP', \mathfrak{S}, \bar{y})$, and therefore $\sim\delta_H(\text{val } \langle AP', \mathfrak{S}, \bar{y} \rangle) = T$, i.e., $\delta_H(\text{val } \langle AP', \mathfrak{S}, \bar{y} \rangle) = F$.

But since $\delta_H(\text{val } \langle AP, \mathfrak{S}, \bar{y} \rangle) = T$, while $\delta_H(\text{val } \langle AP', \mathfrak{S}, \bar{y} \rangle) = F$, it follows that $\text{val } \langle AP, \mathfrak{S}, \bar{y} \rangle \neq \text{val } \langle AP', \mathfrak{S}, \bar{y} \rangle$.

q.e.d.

4.3 Equivalence of Abstract Programs

Definition 6

Two comparable abstract programs AP and AP' are said to be equivalent if for every interpretation $\mathfrak{I}$ that contains assignments for all the constants that occur in AP or AP' , the programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are equivalent.

Theorem 4

Two comparable abstract programs AP and AP' are equivalent, if and only if

$W_{AP, AP'}$ is unsatisfiable [or equivalently, $\sim W_{AP, AP'}$ is valid].

Proof

AP and AP' are equivalent,
 if and only if (by Definition 6)
 for every interpretation $\mathfrak{I}$, the programs $(AP, \mathfrak{I})$ and $(AP', \mathfrak{I})$ are
 equivalent,
 if and only if (by Theorem 3)
 for every interpretation $\mathfrak{I}$, $(W_{AP, AP'}, \mathfrak{I})$ unsatisfiable,
 if and only if
 $W_{AP, AP'}$ is unsatisfiable.

Theorem 4 transforms completely the equivalence problem of abstract programs to an equivalent problem in logic. So, by Theorem 4 we can obtain many results about the equivalence problem of abstract programs, just by applying well-known results in logic. In the remainder of this section we shall present several such results.

It is a well-known result that
the equivalence problem of abstract programs is undecidable.

That is, there can be no algorithm which takes as input any two comparable abstract programs and in all cases stops with a decision as to whether the abstract programs are equivalent or not.

This result follows directly from the undecidability of the termination problem of abstract programs (see sec. 3.4), since an abstract program terminates if and only if it is equivalent to itself.

But, by Theorem 4 it follows that

Corollary 3

the equivalence problem of abstract programs is semi-decidable.

That is, there is an algorithm (called a semi-decision procedure), which takes as input any two comparable abstract programs, and

1. if they are equivalent, the algorithm will stop and say so,
2. if they are not equivalent, the algorithm will never stop.

Since the validity problem of the predicate calculus is semi-decidable, Corollary 3 follows directly by Theorem 4. Moreover, any known semi-decision procedure for solving the validity problem of the predicate calculus can be used, together with Algorithm 2, as a semi-decision procedure for solving the equivalence problem of abstract programs.

Though the equivalence problem of abstract programs is undecidable, there nevertheless exist subclasses of abstract programs for which the equivalence problem is decidable.

Corollary 4

The equivalence problem for the following classes is decidable:

1. $C_1 = \{AP \mid AP \text{ is an abstract program without function constants } f_i^n, n \geq 1\},$
2. $C_2 = \{AP \mid AP \text{ is an abstract program which has only one program variable } x \text{ (i.e., } n = 1), \text{ and all the occurrences of function constants in } AP \text{ are in terms of the form } f_i^0 \text{ or } f_i^1(x)\},$
3. $C_3 = \{AP \mid AP \text{ is an abstract program which has only two program variables } x_1 \text{ and } x_2 \text{ (i.e., } n = 2), \text{ and all the occurrences of function constants in } AP \text{ are in terms of the form } f_i^0 \text{ or } f_i^2(x_1, x_2)\}.$

That is, for each i , $1 \leq i \leq 3$, there is an algorithm which takes as input any two comparable abstract programs AP , $AP' \in C_i$, and in all cases stops with a decision as to whether AP and AP' are equivalent or not. This follows, by using Theorem 4, from the decidability of the validity problem for the class W_i (sec. 1.2).⁽¹⁾

Most of the results for the termination problem presented in Chapter 3 are special cases of the results presented in this chapter, especially corollaries 1 and 2 follows from corollaries 3 and 4 respectively, since every abstract program AP terminates if and only if it is equivalent to itself.

¹ See the proof of Corollary 2 in sec. 3.4.

BLANK PAGE

CHAPTER 5: TERMINATION OF NON-DETERMINISTIC PROGRAMS AND NON-DETERMINISTIC ABSTRACT PROGRAMS

5.1 Definitions

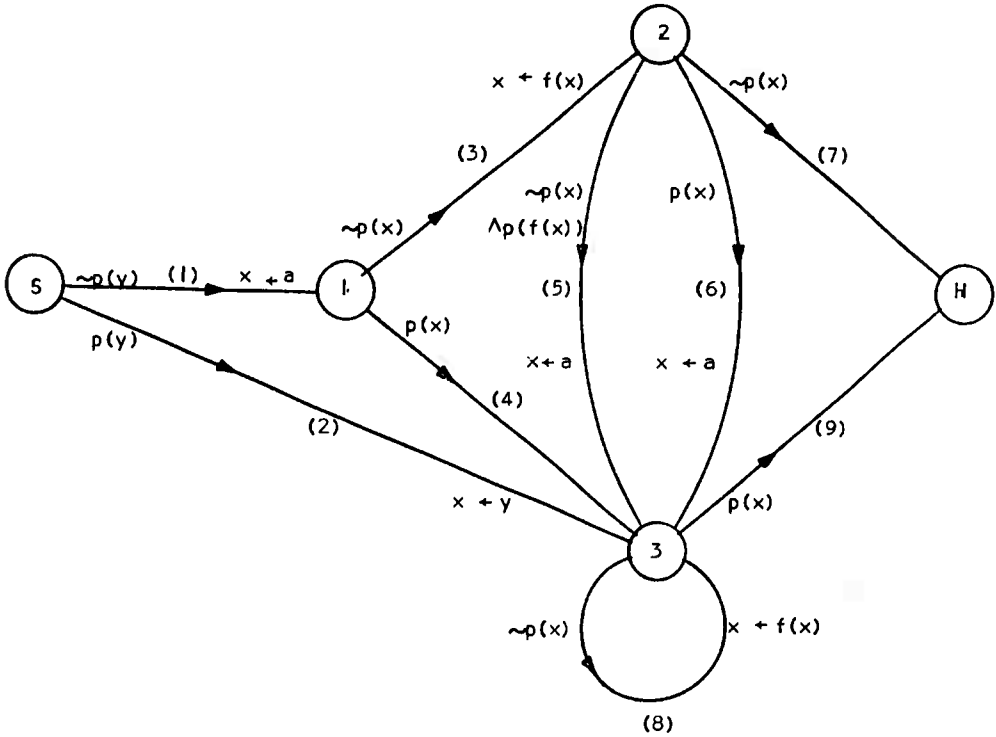
A non-deterministic abstract program GP is defined exactly as an abstract program (see sec. 2.1), but without restriction 4(b), i.e., without the restriction that for every vertex $v (v \neq H)$, the test predicates on all the arcs leading from v are mutually exclusive.

This implies that the class of all the non-deterministic abstract programs includes as a proper subclass the class of all the abstract programs.

The notions of non-deterministic program $(GP, \mathcal{S})$ and non-deterministic interpreted program $(GP, \mathcal{S}, \bar{Y})$ are defined exactly as for abstract programs (see sections 2.2 and 2.3).

Example

The following diagram represents a non-deterministic abstract program. We shall later refer to it as GP^* :



where

- a - individual constant,
- f - monadic function constant,
- p - monadic predicate constant,
- y - input variable,
- x - program variable.

Since the test predicates on all the arcs leading from vertex 2 [i.e., $\sim p(x)$, $p(x)$, and $\sim p(x) \wedge p(f(x))$], are not mutually exclusive - $\mathcal{AP}^*$ is not an abstract program.

Let $\mathcal{I}^*$ be the following interpretation of $\mathcal{AP}^*$:

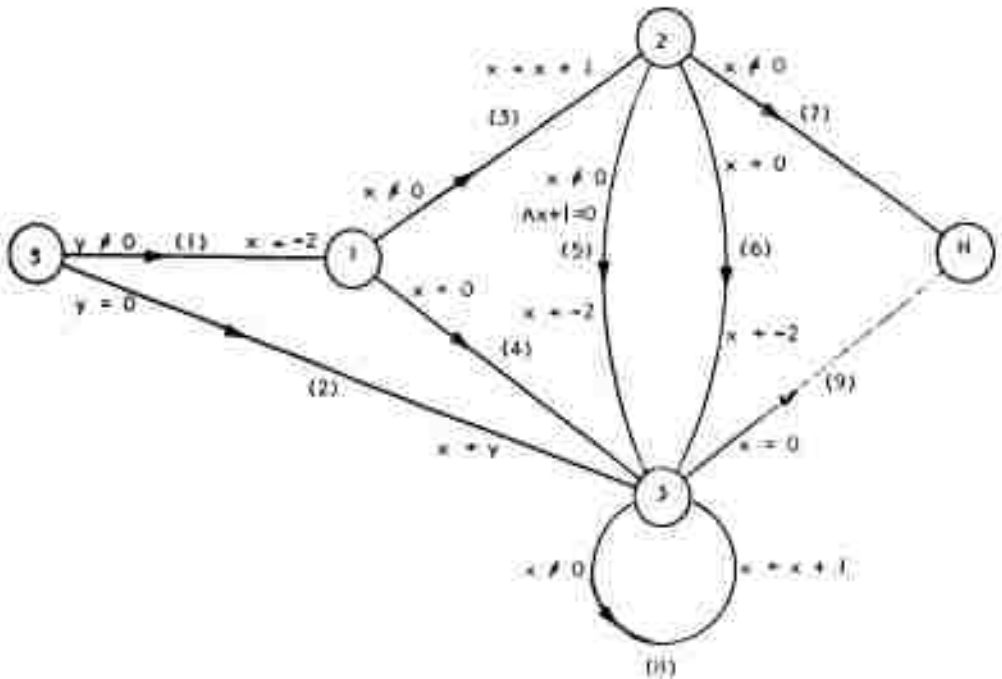
D is I (the domain of the integers),

$f(x)$ is $x + 1$,

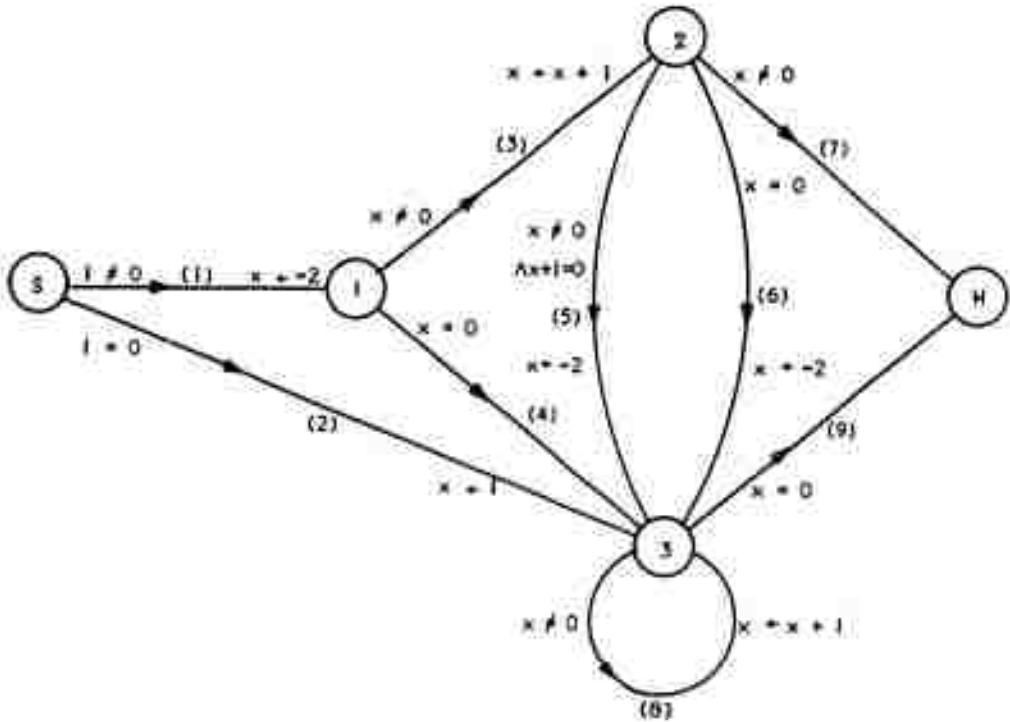
$p(x)$ is $x = 0$, and

a is -2 .

Then the non-deterministic program $(\mathcal{AP}^*, \mathcal{I}^*)$ can be represented by the domain $D = I$ and the diagram



By assigning the value 1 to the variable y of $(\mathcal{M}^*, \mathcal{V}^*)$, we obtain the non-deterministic interpreted program $(\mathcal{M}^*, \mathcal{V}^*, 1)$:



In a non-deterministic interpreted program $(\mathcal{M}^*, \mathcal{V}^*, \bar{y})$ there may exist a vertex v and two distinct arcs α_1 and α_2 leading from v , such that control may reach vertex v with $\bar{x} = \bar{\xi}$, $\bar{\xi} \in (D_y)^n$, while both $\varphi_{\alpha_1}(\bar{\xi}) = T$ and $\varphi_{\alpha_2}(\bar{\xi}) = T$.⁽¹⁾

⁽¹⁾ $\varphi_{\alpha_1}(\bar{\xi})$ and $\varphi_{\alpha_2}(\bar{\xi})$ stand for the result of substituting $\bar{\xi}$ for $\bar{y}$ in φ_{α_1} and φ_{α_2} respectively.

It follows that in general a non-deterministic interpreted program $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$ does not define a unique execution sequence $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$ as for interpreted programs (see sec. 2.3), but a set $\{ \langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle \}$ of execution sequences.

Example

The interpreted program $\langle \mathcal{G}^*, \mathcal{I}^*, l \rangle$ defines two execution sequences:

$(1, l, -2) (3, 2, -1) (7, H, -1)$, and

$(1, l, -2) (3, 2, -1) (5, 3, -2) (8, 3, -1) (8, 3, 0) (9, H, 0)$.

Let $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$ be a non-deterministic interpreted program, and $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$ be any fixed execution sequence of $\{ \langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle \}$.

Let $v \in V$ be any vertex of $\mathcal{G}$, and δ be a specified total predicate from $(D_{\mathcal{I}})^n$ into $\{T, F\}$.

Then,

1. δ is called a valid predicate of v for $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$,

if

$\forall \bar{\xi}, \bar{\xi} \in (D_{\mathcal{I}})^n$: if for some $l \in L$, there exists a triple of the form $(l, v, \bar{\xi})$ in $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$, then $\delta(\bar{\xi}) = T$.

2. δ is called the minimal valid predicate of v for $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$

if

$\forall \bar{\xi}, \bar{\xi} \in (D_{\mathcal{I}})^n$: $\delta(\bar{\xi}) = T$ if and only if for some $l \in L$, there exists a triple of the form $(l, v, \bar{\xi})$ in $\langle \mathcal{G}, \mathcal{I}, \bar{\gamma} \rangle$.

5.2 Weak Termination

Let GP be any abstract program, and W_{GP} be the wff obtained from GP by applying Algorithm 1 (see sec. 3.1).

Definition 7

A non-deterministic program $(GP, \mathfrak{J})$ is said to terminate weakly, if $\bar{w}_Y, \bar{w}_e(D_{\mathfrak{J}})^m$, there exists at least one finite execution sequence in $\{ \langle GP, \mathfrak{J}, \bar{w} \rangle \}$.

The proof of the following theorem is similar to the proof of Theorem 1 in sec. 3.2.

Theorem 5

The non-deterministic program $(GP, \mathfrak{J})$ terminates weakly, if and only if

$(W_{GP}, \mathfrak{J})$ is unsatisfiable [or equivalently, $(\sim W_{GP}, \mathfrak{J})$ is valid].

Definition 8

A non-deterministic abstract program GP is said to terminate weakly if

for every interpretation $\mathfrak{J}$, the program $(GP, \mathfrak{J})$ terminates weakly.

The proof of the following theorem follows from Theorem 5 and Definition 8 (see the proof of Theorem 2 in sec. 3.3).

Theorem 6

The non-deterministic abstract program Q_P terminates weakly,
if and only if

W_{Q_P} is unsatisfiable [or equivalently, $\sim W_{Q_P}$ is valid].

5.3 The Algorithm to Construct W_{GP}

In this section we shall describe an algorithm to construct from a given abstract program GP a wff W_{GP} . In the next section we shall state results about the relation between GP and W_{GP} .

Algorithm 3

Let GP be any non-deterministic abstract program with program variables $\bar{x} = (x_1, x_2, \dots, x_n)$, $n \geq 1$, and input variables $\bar{y} = (y_1, y_2, \dots, y_m)$, $m \geq 0$. We shall construct the wff W_{GP} in three steps:

Step 1

Associate with every vertex v_i of GP a predicate variable q_i , where the q_i 's are distinct n-adic predicate variables.

Step 2

Let v_i be any vertex of GP ($v_i \neq H$).

Let $\alpha_1, \alpha_2, \dots, \alpha_N$ be the set of all the arcs leading from v_i to $v_{i_1}, v_{i_2}, \dots, v_{i_N}$ respectively. In step 1 we have associated with the vertex v_i the predicate variable q_i and with the vertex v_{i_j} , $1 \leq j \leq N$, the predicate variable q_{i_j} .

We shall define the wff W_{v_i} (the wff of the vertex v_i) as

$$W_{v_i} : q_i(\bar{x}) \supset \bigvee_{j=1}^N [q_{\alpha_j} \wedge q_{i_j}(\bar{t}_{\alpha_j})]$$

But,

1. If $v_i = S$ (i.e., v_i is the start vertex of GP), then replace the occurrence of $q_i(\bar{x})$ in W_{v_i} by T , and
2. If $v_{ij} = H$ (i.e., v_{ij} is the halt vertex of GP), replace the occurrence of $q_{ij}(\bar{t}_{\alpha_j})$ in W_{v_i} by F .

Step 3

Let $v_1, v_2, \dots, v_M$ be the set of all the vertices of GP (except H), then define W_{GP} as

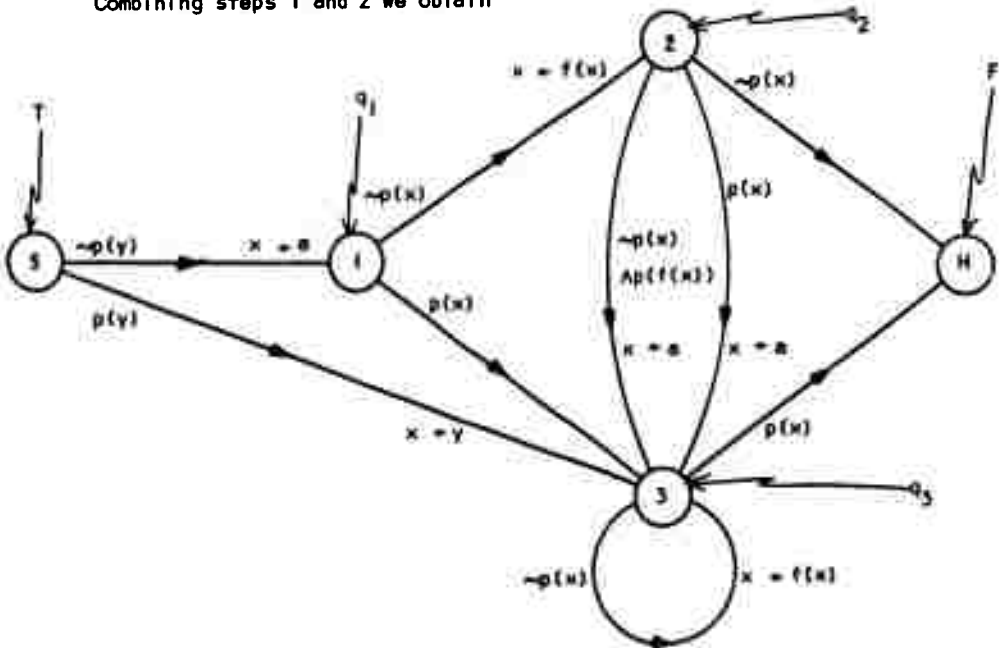
$$W_{GP}: (\bar{x}) [W_{v_1} \wedge W_{v_2} \wedge \dots \wedge W_{v_M}]. \quad (1)$$

¹ Note that the input variables $\bar{y}$ are free variables in W_{GP} .

Example

The wff W_{Gp^*} of the non-deterministic abstract program Gp^* of sec. 5.1 will be constructed as follows:

Combining steps 1 and 2 we obtain



$$W_S: T \supset \{[\sim p(y) \wedge q_1(a)] \vee [p(y) \wedge q_3(y)]\}$$

$$W_I: q_1(x) \supset \{[\sim p(x) \wedge q_2(f(x))] \vee [p(x) \wedge q_3(x)]\}$$

$$W_Z: q_2(x) \supset \{[\sim p(x) \wedge p(f(x)) \wedge q_3(a)] \vee [p(x) \wedge q_3(a)] \vee [\sim p(x) \wedge F]\}$$

$$W_3: q_3(x) \supset \{[\sim p(x) \wedge q_3(f(x))] \vee [p(x) \wedge F]\}.$$

Then by step 3 it follows that

$$W_{Gp^*} \text{ is } (x)[W_S \wedge W_I \wedge W_Z \wedge W_3].$$

5.4 Strong Termination of Non-Deterministic Programs

Definition 2

A non-deterministic program (G_P, γ) is said to terminate strongly if

$\bar{\gamma}, \bar{\gamma} \in (D_\gamma)^m$, all the execution sequences in $\{ \langle G_P, \gamma, \bar{\gamma} \rangle \}$ are finite.

Theorem 7

The non-deterministic program (G_P, γ) terminates strongly

if and only if

$(L_{G_P, \gamma})$ is unsatisfiable [or equivalently, $(\neg L_{G_P, \gamma})$ is valid].

Proof

We shall prove that (G_P, γ) does not terminate strongly if and only if $(L_{G_P, \gamma})$ is satisfiable.

1. (G_P, γ) does not terminate strongly $\Rightarrow (L_{G_P, \gamma})$ is satisfiable.

If (G_P, γ) does not terminate strongly, there exists a

$\bar{\gamma}, \bar{\gamma} \in (D_\gamma)^m$, and an execution sequence $\langle G_P, \gamma, \bar{\gamma} \rangle, \langle G_P, \gamma, \bar{\gamma} \rangle \in \{ \langle G_P, \gamma, \bar{\gamma} \rangle \}$, which is infinite.

Let us assign to each predicate variable q_i in L_{G_P} the minimal valid predicate of the vertex v_i for the execution sequence $\langle G_P, \gamma, \bar{\gamma} \rangle$.

Note that since the execution sequence $\langle G_P, \gamma, \bar{\gamma} \rangle$ is infinite, i.e., control never reaches the halt vertex, it follows that the predicate F is the minimal valid predicate of the vertex H for $\langle G_P, \gamma, \bar{\gamma} \rangle$.

Let Γ consists of the above assignments for the q_i 's and with $\bar{\gamma}$ assigned to $\bar{\gamma}$. Following the construction of L_{G_P} (see sec. 5.3,

especially note the $\vee$ connective used in step 2), it is clear that the value of $(\mathcal{U}_{\mathcal{A}}, \mathcal{V}, \Gamma)$ is T, i.e., $(\mathcal{U}_{\mathcal{A}}, \mathcal{V})$ is satisfiable. This completes the proof in one direction.

2. $(\mathcal{U}_{\mathcal{A}}, \mathcal{V})$ is satisfiable $\Rightarrow (\mathcal{A}, \mathcal{V})$ does not terminate strongly.

If $(\mathcal{U}_{\mathcal{A}}, \mathcal{V})$ is satisfiable, there exist an assignment Γ for $(\mathcal{U}_{\mathcal{A}}, \mathcal{V})$ such that the value $(\mathcal{U}_{\mathcal{A}}, \mathcal{V}, \Gamma)$ is T. Γ consists of assignments of specified total predicates δ_i , mapping $(D_y)^n$ into $\{T, F\}$, for the predicate variables q_i , and an assignment $\bar{y}$, $\bar{y} \in (D_y)^m$, for the free variables $\bar{y}$.

By the construction of $\mathcal{U}_{\mathcal{A}}$, this implies that each δ_i is a valid predicate of the vertex v_i for some execution sequence $\langle \mathcal{A}, \mathcal{V}, \bar{y} \rangle$, $\langle \mathcal{A}, \mathcal{V}, \bar{y} \rangle \in \langle \mathcal{A}, \mathcal{V}, \bar{y} \rangle$, and therefore that F is a valid predicate of the halt vertex for $\langle \mathcal{A}, \mathcal{V}, \bar{y} \rangle$.

This implies that the execution sequence $\langle \mathcal{A}, \mathcal{V}, \bar{y} \rangle$ is infinite (i.e., execution does not reach the halt vertex). So, $(\mathcal{A}, \mathcal{V})$ does not terminate strongly.

q.e.d.

The above result can be used to prove the convergence of recursively defined functions.

Let us consider, for example, the functions $F_1(x)$ and $F_2(x)$ defined recursively by the following Algol conditional statements:

$$F_1(x) = \text{if } x = 0 \text{ then } 1 \\ \quad \text{else if } x > 0 \text{ then } 2 * F_1(x-1) \\ \quad \text{else } F_2(-x) * F_1(x+1);$$

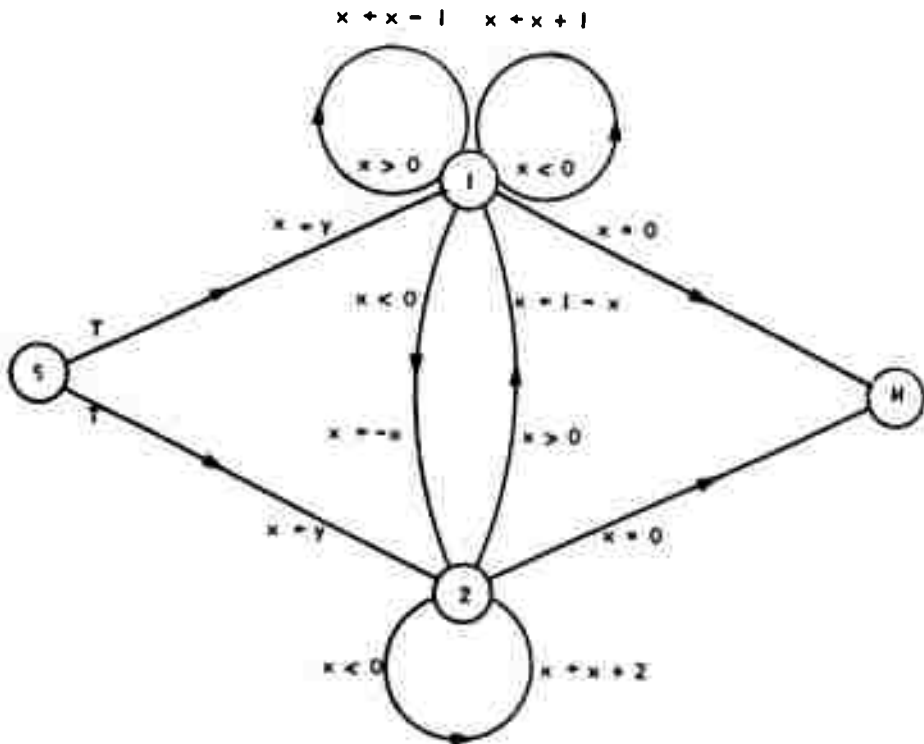
$$F_2(x) = \text{if } x = 0 \text{ then } 2 \\ \quad \text{else if } x < 0 \text{ then } 3 * F_2(x+2) + 7 \\ \quad \text{else } \{F_1(1-x)\}^2.$$

Suppose that we want to prove that for every integer x , the recursive process of computing $F_1(x)$ and $F_2(x)$ terminates. We can use Theorem 7, since:

for every integer x , the recursive process for computing $F_1(x)$ and $F_2(x)$ terminates,

if and only if

the following non-deterministic program (over I) terminates strongly.



[Consider vertex 1 as representing the start of the computation of $F_1(x)$ and vertex 2 as representing the start of the computation of $F_2(x)$.]

5.5 Strong Termination of Non-Deterministic Abstract Programs

Definition 10

A non-deterministic abstract program GP is said to terminate strongly, if for every interpretation $\mathfrak{I}$, the non-deterministic program $GP, \mathfrak{I}$ terminates strongly.

The following theorem follows from Theorem 7 and Definition 10.

Theorem 8

A non-deterministic abstract program GP terminates strongly
if and only if

$\mathcal{M}_{GP}$ is unsatisfiable [or equivalently, $\sim \mathcal{M}_{GP}$ is valid].

Proof

GP terminates strongly,
if and only if (follows by Definition 10)
for every interpretation $\mathfrak{I}$, the non-deterministic program $GP, \mathfrak{I}$
terminates strongly,
if and only if (follows by Theorem 7)
for every interpretation $\mathfrak{I}$, $(\mathcal{M}_{GP}, \mathfrak{I})$ is unsatisfiable,
if and only if
 $\mathcal{M}_{GP}$ is unsatisfiable.

Theorem 8 is a generalization of Theorem 2 of sec. 3.3. Moreover, all the results presented in sec. 3.4 (Corollaries 1 and 2) can also be generalized for the strong termination of non-deterministic abstract programs.

REFERENCES

Ackerman [1954]

Ackermann, W., Solvable Cases of the Decision Problem, North-Holland Publishing Company, Amsterdam (1954).

Church [1956]

Church, A., Introduction to Mathematical Logic, Volume I, Princeton University Press, Princeton, New Jersey (1956).

Davis and Putnam [1960]

Davis, M. and H. Putnam, "A Computing Procedure for Quantification Theory," J. ACM 7 (3), 201-215 (July, 1960).

Friedman [1963]

Friedman, J., "A Semi-Decision Procedure for the Functional Calculus," J. ACM 10 (1), 1-24 (January, 1963).

Kleene [1950]

Kleene, S. C., Introduction to Mathematics, D. Van Nostrand Company, Inc., Princeton, New Jersey (1950).

Kleene [1967]

Kleene, S. C., Mathematical Logic, John Wiley & Sons, Inc., New York (1967).

Luckham, Park and Paterson [1967]

Luckham, D. C., D. M. R. Park and M. S. Paterson, "On Formalised Computer Programs," Programming Research Group, Oxford University (August, 1967).

Mendelson [1964]

Mendelson, E., Introduction to Mathematical Logic, D. Van Nostrand Company, Inc., Princeton, New Jersey (1964).

Robinson [1965]

Robinson, J. A., "A Machine-Oriented Logic Based on the Resolution Principle," J. ACM 12 (1), 23-41 (January, 1965).

PART II

Introduction

Since Part I and Part II of the thesis are intended to be self-contained units, the background information necessary to understand Part II is entirely contained in this part.

- An interpreted graph IG consists of a finite directed graph, and
1. With each vertex v , there is associated a domain D_v , and
 2. With each arc a leading from vertex v to vertex v' , there are associated a total test predicate $P_a (D_v \rightarrow \{T, F\})$, and a total function $f_a (D_v \wedge P_a \rightarrow D_{v'})$.

Let us represent by a state vector x the current values of the variables during an execution of an interpreted graph IG. An execution sequence of IG may start from any vertex v with any initial state vector $x_0 \in D_v$. The domain D_v is the set of all possible state vectors at vertex v , P_a represents the condition that arc a may be entered from its origin, and f_a represents the operation of changing the state vector x to $f_a(x)$ when control moves along arc a . In general, the flow of control through an interpreted graph is a non-deterministic process, i.e., more than one arc may be entered from a given vertex with a given state vector. Execution will halt on vertex v , with state vector x , if and only if no predicate on any arc leading from v is true for x .

An interpreted graph terminates if and only if all the execution sequences of IG terminate.

In this part, two necessary and sufficient conditions for the termination of interpreted graphs are described. The first condition (Theorem 1) is defined by means of well-ordered sets and the properties of the cycles of the graph, while the second condition (Theorem 2) is defined by means of the strongly connected components of the graph.

Floyd [1967] has discussed the use of well-ordered sets for proving the termination of programs.

These results have applications in proving termination of various classes of algorithms, such as deterministic and non-deterministic programs and recursively defined functions.

CHAPTER 1: MATHEMATICAL BACKGROUND

1.1 Well-Ordered Sets

A pair $(S, >)$ is called an ordered set, provided that S is a set and $>$ is a relation defined for every pair of distinct elements a and b of S (and only between distinct elements), and satisfies the following two conditions:

1. If $a \neq b$, then either $a > b$ or $b > a$;
2. If $a > b$ and $b > c$, then $a > c$ (i.e., the relation is transitive).

A well-ordered set W is an ordered set $(S, >)$ in which every non-empty subset has a first element; equivalently, in which every decreasing sequence of elements $a > b > c \dots$ has only finitely many elements.

Examples:

1. I_1^+ - the set of all non-negative integers well-ordered by its natural order, i.e., $\{0, 1, 2, 3, \dots\}$.
2. I_n^+ - the set of all n -tuples of non-negative integers for some fixed n , $n \geq 1$, well-ordered by the usual lexicographic order, i.e.,

$$(a_1, a_2, \dots, a_n) > (b_1, b_2, \dots, b_n)$$

if and only if

$$a_1 = b_1, a_2 = b_2, \dots, a_{k-1} = b_{k-1}, a_k > b_k \text{ for some } k, 1 \leq k \leq n.$$

3. I_{∞}^{+} - the set of all infinite monotone non-increasing sequences of non-negative integers with finitely many non-zero entries⁽¹⁾ well-ordered by the usual lexicographic order, i.e.,

$$(a_1, a_2, a_3, \dots) > (b_1, b_2, b_3, \dots)$$

if and only if

$$a_1 = b_1, a_2 = b_2, \dots, a_{k-1} = b_{k-1}, a_k > b_k \text{ for some } k, 1 \leq k.$$

1.2 Directed Graphs

A directed graph G (graph, for short) is an ordered triple $\langle V, L, A \rangle$ where:

1. V is a non-empty set of elements called the vertices of G ;
2. L is a non-empty set of elements called the labels of G ; and
3. A is a set of ordered triples (v, l, v') , where $v \in V$, $v' \in V$ and $l \in L$. These triples are called the arcs of G .

If V and L are finite sets, G is called a finite directed graph.

¹ i.e., the infinite sequence $(a_1, a_2, a_3, \dots)$ is in the set if and only if $\exists l, 1 \leq l$, s.t.

$$\forall (i < l): a_i \text{ is a positive integer and } a_i \geq a_{i+1}, \text{ and}$$

$$\forall (i \geq l): a_i = 0.$$

For example, $(5, 5, 4, 3, 3, 3, 3, 1, 0, 0, \dots)$ is an element in this set.

Let $a = (v, l, v')$ be an arc of a directed graph. Then we define:

1. v - the initial vertex of the arc,
2. l - the label of the arc,
3. v' - the terminal vertex of the arc.

And we shall say that the arc a leads from the vertex v to the vertex v' .

Let v be a vertex of a directed graph. Then,

1. The number (finite or infinite) of all arcs $a \in A$, s.t. v is the initial vertex of a , is called the out-degree of v .
2. The number (finite or infinite) of all arcs $a \in A$, s.t. v is the terminal vertex of a , is called the in-degree of v .

A finite path of a graph G (path, for short) is a finite sequence of n , $n \geq 1$, arcs of G

$$(v_{i_1}, l_{i_1}, v_{i_2}), (v_{i_2}, l_{i_2}, v_{i_3}), \dots, (v_{i_n}, l_{i_n}, v_{i_{n+1}})$$

$$[\text{notation: } v_{i_1} \xrightarrow{l_{i_1}} v_{i_2} \xrightarrow{l_{i_2}} v_{i_3} \dots v_{i_n} \xrightarrow{l_{i_n}} v_{i_{n+1}}],$$

s.t. the terminal vertex of each arc coincides with the initial vertex of the succeeding arc.

We say that:

1. The path meets the vertices $v_{i_1}, v_{i_2}, \dots, v_{i_{n+1}}$, and these vertices are on the path.

2. The path joins the vertices v_{i_1} and $v_{i_{n+1}}$.
3. The path is elementary if the vertices $v_{i_1}, v_{i_2}, \dots, v_{i_{n+1}}$ are distinct.
4. The path is a cycle if the vertex v_{i_1} coincides with the vertex $v_{i_{n+1}}$, further it is an elementary cycle if in addition the vertices $v_{i_1}, v_{i_2}, \dots, v_{i_n}$ are distinct.

An infinite path of a graph G is an infinite sequence of arcs of G s.t. the terminal vertex of each arc coincides with the initial vertex of the succeeding arc. A subpath of an infinite path is a consecutive subsequence (finite or infinite) of its arcs.

We define a cut set of a graph G as a set of vertices having the property that every cycle meets at least one vertex of the set.

A graph G is said to be strongly connected if there is a path joining any ordered pair of distinct vertices of G .

Let G be a graph $\langle V, L, A \rangle$. We define a subgraph $G_1 = \langle V_1, L, A_1 \rangle$ of G as the triple consisting of V_1 , L and A_1 , where V_1 is a subset of V and A_1 is defined by $A_1 = A \cap (V_1 \times L \times V_1)$.

A subgraph $G_1 = \langle V_1, L, A_1 \rangle$ of G is said to be a strongly connected component of G if,

1. G_1 is strongly connected, and
2. For all subsets $V_2 \subseteq V$ s.t. $V_2 \neq V_1$ and $V_2 \supset V_1$, the subgraph $G_2 = \langle V_2, L, A_2 \rangle$ is not strongly connected.

A tree $T = \langle V, L, A, r \rangle$ is a directed graph $\langle V, L, A \rangle$ with a distinguished root $r \in V$, s.t. for every $v \in V$ ($v \neq r$), there is at least one path from r to v .

We shall use the following version of König's Infinity Lemma:

A tree with no infinite paths and with finite out-degree for every vertex - is finite.

BLANK PAGE

CHAPTER 2: DEFINITIONS

An interpreted graph IG consists of a finite directed graph $\langle V, L, A \rangle$, and

1. With each vertex $v \in V$, there is associated a domain D_v , and
2. With each arc $a = (v, l, v') \in A$, there is associated a total test predicate $P_a (D_v \rightarrow \{T, F\})$, and a total function $f_a (D_v \wedge P_a \rightarrow D_{v'})$.

Let $(v^{(0)}, x^{(0)}) \in V \times D_{v^{(0)}}$ be an arbitrary vector of an interpreted graph IG.

An $(v^{(0)}, x^{(0)})$ - execution-sequence of IG is a (finite or infinite) sequence of the form

$$(v^{(0)}, x^{(0)}) \xrightarrow{l^{(0)}} (v^{(1)}, x^{(1)}) \xrightarrow{l^{(1)}} (v^{(2)}, x^{(2)}) \xrightarrow{l^{(2)}} \dots,$$

where,

1. $v^{(j)} \in V$, $l^{(j)} \in L$ and $x^{(j)} \in D_{v^{(j)}}$ for all $j \geq 0$.
2. If $(v^{(j)}, x^{(j)}) \xrightarrow{l^{(j)}} (v^{(j+1)}, x^{(j+1)})$ is in the sequence, then there exists an arc $a = (v^{(j)}, l^{(j)}, v^{(j+1)}) \in A$ s.t. $P_a x^{(j)} = \text{True}$ and $f_a x^{(j)} = x^{(j+1)}$.
3. If the sequence is finite and the last vector in the sequence is $(v^{(n)}, x^{(n)})$, then for all arcs $a \in A$ leading from $v^{(n)}$: $P_a x^{(n)} = \text{False}$.

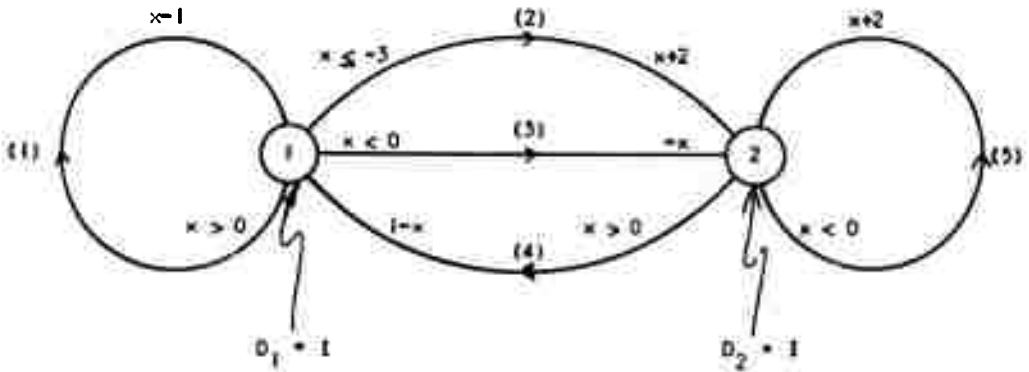
By the definition of interpreted graphs, there may exist in an interpreted graph IG: a vertex $v \in V$, a state vector $x \in D_v$, and two distinct arcs $a, b \in A$ leading from v - s.t. both $P_a x = \text{True}$ and $P_b x = \text{True}$, i.e., the predicates on all arcs leading from the vertex v are not necessarily mutually exclusive. It follows, that for the fixed vector $(v^{(0)}, x^{(0)}) \in V \times D_{v^{(0)}}$, there may exist many distinct $(v^{(0)}, x^{(0)})$ - execution sequences of IG. For this reason, the execution process of an interpreted graph, starting with the vector $(v^{(0)}, x^{(0)})$, is described by a tree.

The execution tree $T(v^{(0)}, x^{(0)})$ is the tree $\langle V', L, A', (v^{(0)}, x^{(0)}) \rangle$, where,

1. The set of vertices V' is the set of all vectors $(v, x) \in V \times D_v$ s.t. there exists an $(v^{(0)}, x^{(0)})$ - execution sequence of IG that contains the vector (v, x) .
2. L is the set of labels of IG.
3. The set of arcs A' is the set of all triples $((v, x), l, (v', y)) \in V' \times L \times V'$ s.t. there exists an $(v^{(0)}, x^{(0)})$ - execution sequence of IG that contains $(v, x) \xrightarrow{l} (v', y)$.
4. $(v^{(0)}, x^{(0)}) \in V'$ is the root-vertex of the tree.

Example

Consider the interpreted graph IG^*



(where I is the set of the integers).

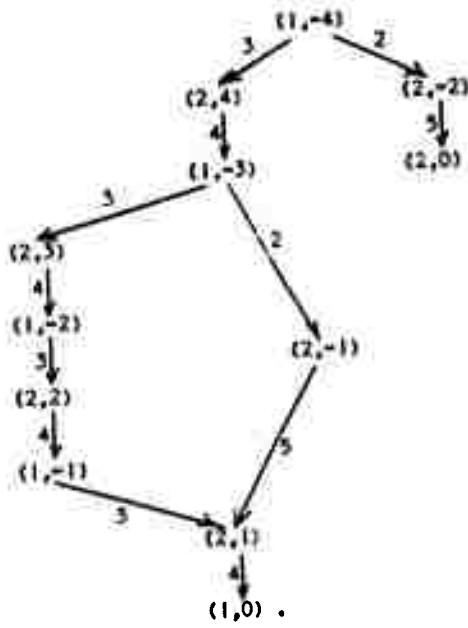
There are three $(1, -4)$ - execution sequences in IG^* , i.e., three execution sequences that start from the vertex 1 with $x = -4$,

$$(I) \quad (1, -4) \xrightarrow{2} (2, -2) \xrightarrow{5} (2, 0),$$

$$(II) \quad (1, -4) \xrightarrow{3} (2, 4) \xrightarrow{4} (1, -3) \xrightarrow{2} (2, -1) \xrightarrow{5} (2, 1) \xrightarrow{4} (1, 0), \text{ and}$$

$$(III) \quad (1, -4) \xrightarrow{3} (2, 4) \xrightarrow{4} (1, -3) \xrightarrow{3} (2, 3) \xrightarrow{4} (1, -2) \xrightarrow{3} (2, 2) \xrightarrow{4} (1, -1) \xrightarrow{3} (2, 1) \xrightarrow{4} (1, 0).$$

The execution tree $T(1, -4)$ of IG^* is:



CHAPTER 3: TERMINATION OF INTERPRETED GRAPHS

3.1 Termination of Interpreted Graphs (Theorem 1)

Definition

An interpreted graph is said to terminate if all its execution sequences are finite⁽¹⁾.

Notations

Let $\alpha = (a_1, a_2, \dots, a_q)$, where $a_j = (v^{(j)}, l^{(j)}, v^{(j+1)}) \in A$ for $1 \leq j \leq q$, be any path of an interpreted graph. Then let

1. $f_\alpha x$ stand for $f_{a_q}(\dots(f_{a_2}(f_{a_1} x))\dots)$, and
2. $P_\alpha x$ stand for

$$x \in D_v(l) \wedge P_{a_1} x \wedge P_{a_2}(f_{a_1} x) \wedge P_{a_3}(f_{a_2}(f_{a_1} x)) \wedge \dots$$

$$\wedge P_{a_q}(f_{a_{q-1}}(\dots(f_{a_2}(f_{a_1} x))\dots)) \wedge f_\alpha x \in D_{v^{(q+1)}}.$$

Lemma

If an interpreted graph IG terminates,

then there exists for every vertex $v \in V$ a total function F_v

which maps D_v into I_1^+ , such that for every arc $a = (v, l, v')$ of IG and

for every x s.t. $P_a x = \text{True}$:

$$F_v(x) > F_{v'}(f_a(x)).$$

⁽¹⁾ i.e., $\forall (v, x), (v, x) \in V \times D_v$, all the (v, x) - execution sequences are finite.

Proof

Assuming that IG terminates, we have to specify $F_v(x)$ for arbitrary $v \in V$ and $x \in D_v$.

Since IG terminates, we know that the execution tree $T(v, x)$ has no infinite paths. Moreover, since every vertex of $T(v, x)$ has a finite out-degree it follows by König's Lemma that $T(v, x)$ is finite, i.e., has finitely many vertices.

So, let $F_v(x)$ be the number of vertices in $T(v, x)$.

Now, it is easy to verify that for this choice of F_v the condition is satisfied.

q.e.d.

Theorem 1

An interpreted graph IG terminates if and only if there exist:

1. A cut set V^* of the vertices V of IG, and
2. For every vertex $v \in V^*$, a well-ordered set $W_v = (S_v, \succ_v)$ and a total function F_v which maps D_v into S_v ,
such that,
3. For every cycle α of IG:

$$v^{(1)} \xrightarrow{L^{(1)}} v^{(2)} \xrightarrow{L^{(2)}} v^{(3)} \dots v^{(q-1)} \xrightarrow{L^{(q-1)}} v^{(q)} \xrightarrow{L^{(q)}} v^{(1)}$$

(where $v^{(1)} \in V^*$ and $v^{(k)} \neq v^{(l)}$ for all $1 < k \leq q$), and for every x s.t. $P_\alpha x = \text{True}$:

$$F_{v^{(1)}}(x) >_{v^{(1)}} F_{v^{(1)}}(f_\alpha x).$$

Proof

⇒ Necessary condition for termination.

Follows directly from the lemma (with $V^* = V$ and $W_V = I_1^+$ for every $v, v \in V$).

⇐ Sufficient condition for termination.

Proof by contradiction.

Let us assume that IG does not terminate, i.e., there exists an infinite execution sequence γ in IG,

$$\gamma: (v^{(0)}, x^{(0)}) \xrightarrow{L^{(0)}} (v^{(1)}, x^{(1)}) \xrightarrow{L^{(1)}} (v^{(2)}, x^{(2)}) \xrightarrow{L^{(2)}} \dots$$

Let γ' be the infinite path

$$\gamma': v^{(0)} \xrightarrow{L^{(0)}} v^{(1)} \xrightarrow{L^{(1)}} v^{(2)} \xrightarrow{L^{(2)}} \dots$$

Since IG, by definition, consists of a finite directed graph, and since γ' is an infinite sequence - it follows, that there exists at least one elementary cycle β in IG, that occurs (as a subpath) infinitely many times in γ' .

Since v^* is a cut set, it follows that there exists a vertex $v^* \in v^*$ that is on β . This implies that v^* must occur infinitely many times in γ' .

Let $v^{(n_1)}, v^{(n_2)}, v^{(n_3)}, \dots$ ($0 \leq n_j < n_{j+1}$ for $j \geq 1$), be the infinite sequence of all occurrences of the vertex v^* in γ' . Therefore, the infinite execution sequence γ can be written as

$$\begin{aligned} \gamma: & (v^{(0)}, x^{(0)}) \xrightarrow{L} \dots (v^{(n_1)}, x^{(n_1)}) \xrightarrow{L} \dots \\ & (v^{(n_2)}, x^{(n_2)}) \xrightarrow{L} \dots (v^{(n_3)}, x^{(n_3)}) \xrightarrow{L} \dots \end{aligned}$$

Then, by condition (3) it follows that

$$F_{v^*}(x^{(n_1)}) >_{v^*} F_{v^*}(x^{(n_2)}) >_{v^*} F_{v^*}(x^{(n_3)}) >_{v^*} \dots$$

i.e., there is an infinite decreasing sequence in W_{v^*} . But this contradicts the fact that W_{v^*} is a well-ordered set.

q.e.d.

The following corollaries follow directly from the lemma and Theorem 1.

Corollary 1

An interpreted graph IG, which has a vertex v^* common to all its (elementary) cycles, terminates

if and only if

there exist a well-ordered set $W = (S, >)$ and a total function F which maps D_{v^*} into S , such that for every elementary cycle

$\alpha: v^* \rightarrow \dots \rightarrow v^*$ and for every x s.t. $P_\alpha x = \text{True}$:

$$F(x) > F(f_\alpha(x)).$$

Corollary 2

An interpreted graph IG terminates

if and only if

there exist:

1. A cut set V^* of the vertices V of IG,
2. A well-ordered set $W = (S, >)$, and
3. For every vertex $v \in V^*$, a total function F_v that maps D_v into S ,

such that

4. For every elementary path α of IG:

$$v^{(1)} \xrightarrow{f^{(1)}} v^{(2)} \xrightarrow{f^{(2)}} v^{(3)} \dots v^{(q-1)} \xrightarrow{f^{(q-1)}} v^{(q)}$$

(where $v^{(1)}, v^{(q)} \in V^*$ and $v^{(j)} \notin V^*$ for all $j, 1 < j < q$),

and for every x s.t. $P_\alpha(x) = \text{True}$:

$$F_{v(l)}(x) > F_{v(q)}(f_\alpha(x)).$$

3.2 Termination of Interpreted Graphs (Theorem 2)

Let IG be an interpreted graph constructed from the finite directed graph G.

Then a strongly connected component IG' of IG consists of a strongly connected component $G' = \langle V', L, A' \rangle$ of G, and in addition,

1. With each vertex $v \in V'$, there is associated the domain D_v of IG, and
2. With each arc $a \in A'$, there are associated the test-predicate P_a and the function f_a of IG.

Theorem 2

An interpreted graph IG terminates
if and only if
 all its strongly connected components terminate.

Proof

⇒ Necessary Condition for Termination

Follows directly from the definition of termination of interpreted graphs.

= Sufficient Condition for Termination

Proof by Contradiction.

Let's assume that IG does not terminate, i.e., there exists an infinite execution sequence γ in IG,

$$\gamma: (v^{(0)}, x^{(0)}) \xrightarrow{L^{(0)}} (v^{(1)}, x^{(1)}) \xrightarrow{L^{(1)}} (v^{(2)}, x^{(2)}) \xrightarrow{L^{(2)}} \dots$$

Let γ' be the infinite path

$$\gamma': v^{(0)} \xrightarrow{L^{(0)}} v^{(1)} \xrightarrow{L^{(1)}} v^{(2)} \xrightarrow{L^{(2)}} \dots$$

Since IG, by definition, consists of a finite directed graph G - it follows that IG contains finitely many vertices. So clearly, there are only finitely many vertices of G that meet γ' only a finite number of times. Let $v^{(n_1)}, v^{(n_2)}, \dots, v^{(n_q)}$ ($0 \leq n_j < n_{j+1}$ for $1 \leq j < q$), be the list of their occurrences in γ' .

It follows that all the vertices $v^{(j)}$ ($j > n_q$) of γ' , are in some strongly connected component G' of G.

This implies that there exists a strongly connected component IG' of IG, s.t. the infinite subsequence of γ :

$$(v^{(n_{q+1})}, x^{(n_{q+1})}) \xrightarrow{L^{(n_{q+1})}} (v^{(n_{q+2})}, x^{(n_{q+2})}) \xrightarrow{L^{(n_{q+2})}} \dots$$

is an infinite execution sequence of IG' , i.e., IG' does not terminate. Contradiction.

q.e.d.

BLANK PAGE

CHAPTER 4: APPLICATIONS

The results of Chapter 3 can be used for proving termination of various classes of algorithms. In this section we shall illustrate the use of those results for proving termination of:

1. Programs, and
2. Recursively defined functions.

In the first example, we shall use the notion of valid interpretation. Roughly speaking, a valid interpretation of a flowchart is a mapping of its test-boxes to propositions, such that, if the test-box B is mapped to the proposition q, and if the flow of control through the flowchart can reach the test-box B with ξ as the value of the state vector, then $q(\xi) = \text{True}$ (see Floyd [1967]).

4.1 Example 1:

Consider the program (Figure 1) for evaluating a determinant $|a_{ij}|$ of order n , $n \geq 1$, by Gaussian elimination. Where,

- D - real variable,
- $(a_{ij})_{1 \leq i, j \leq n}$ - real array,
- i, j, k - integer variables,
- n - integer constant.

[We consider the division operator over the real domain as a total function, by interpreting, for example, $\frac{r}{0}$ as $\frac{r}{10^{-10}}$ for every real r .]

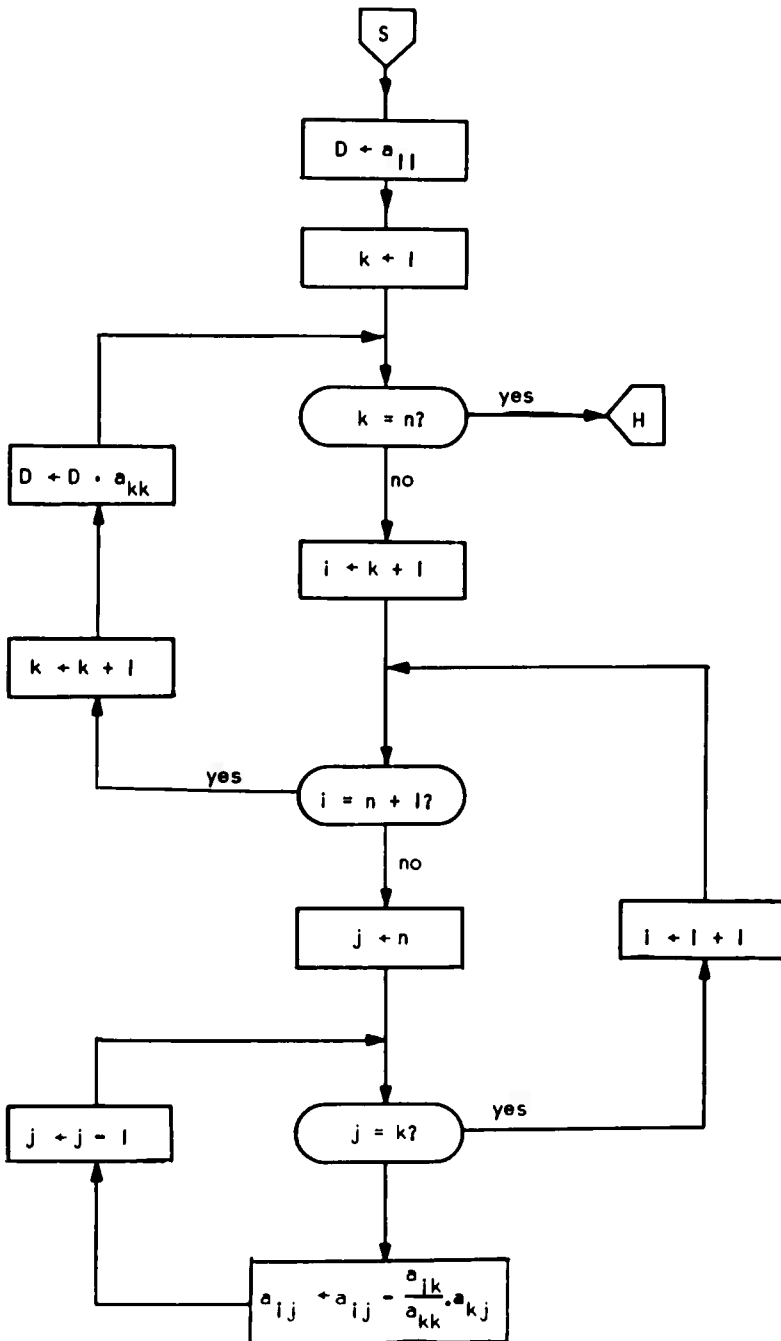


Figure 1

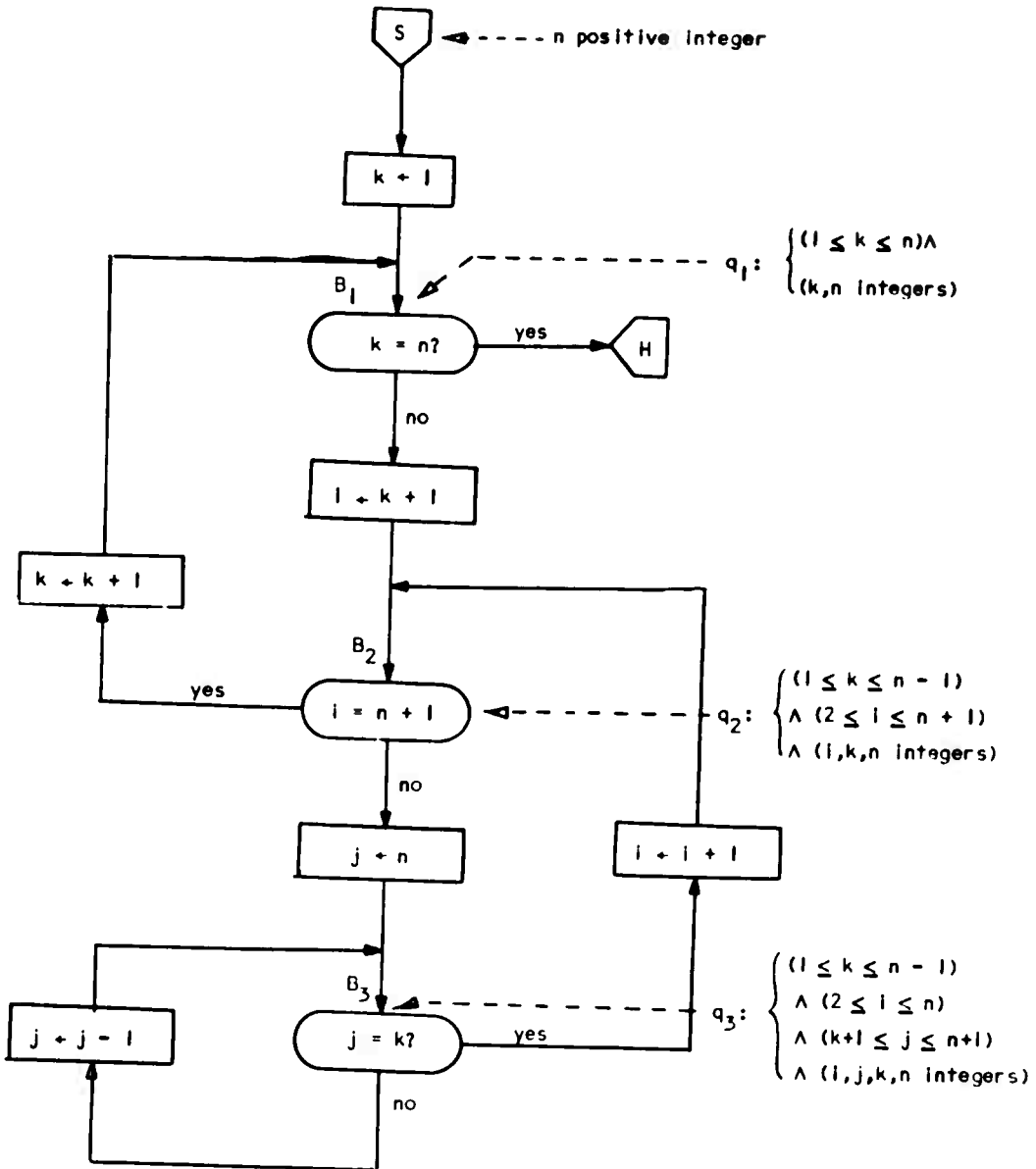


Figure 2

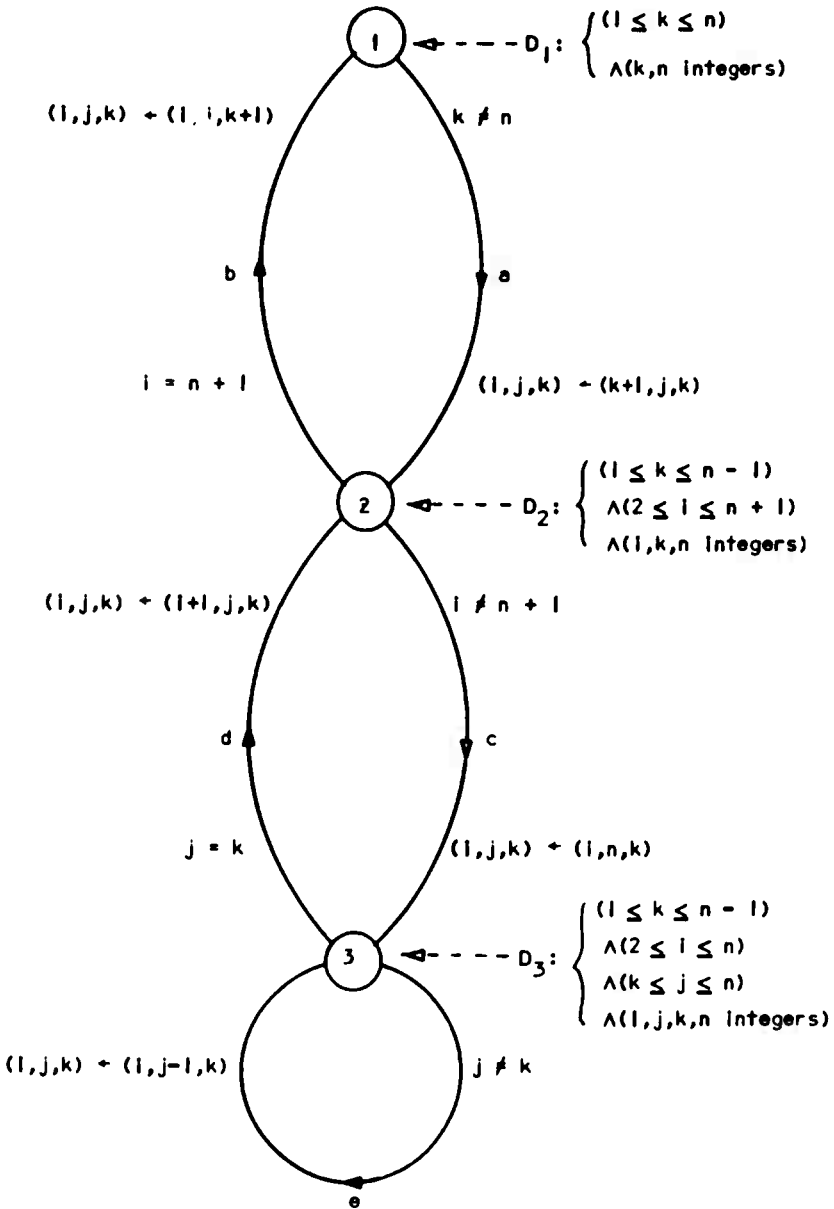


Figure 3

We want to show that the program terminates for every positive integer n .

Since neither D nor any a_{ij} occurs in a test-box or affect the value of any variable that occurs in a test-box, it is clear that by erasing the following three assignment boxes:

$$\begin{aligned} D &\leftarrow a_{11} \quad , \\ D &\leftarrow D \cdot a_{kk} \quad , \text{ and} \\ a_{ij} &\leftarrow a_{ij} - \frac{a_{ik}}{a_{kk}} \cdot a_{kj} \end{aligned}$$

we do not change the termination properties of the program. In other words,

For every integer n , the original program (Figure 1) terminates if and only if the reduced program (Figure 2) terminates.

One can verify easily that the set of predicates attached to the test-boxes of the flowchart of Figure 2 - considering the initial predicate " n positive integer" - is a valid interpretation.

Let's construct now, from the reduced program (Figure 2), the appropriate interpreted graph (Figure 3), s.t. each vertex i , $1 \leq i \leq 3$, of Figure 3 corresponds to the test-box B_i of Figure 2, and its domain D_i is exactly the valid interpretation q_i of Figure 2.

Note that we have used theorem 2 here, by considering only the strongly connected component of our graph.

It is clear that,

If the interpreted graph (Figure 3) terminates, then the reduced program (Figure 2) terminates for every positive integer n .

Now, use corollary 2, where

$V^* = \{2,3\}$ is the cut set,

$W = I_3^+$ is the well-ordered set,

$F_2(i,j,k) = (n-1-k, n+1-i, n+1)$ is the mapping of D_2 into W , and

$F_3(i,j,k) = (n-1-k, n+1-i, j)$ is the mapping of D_3 into W .

Note that when control moves:

- (i) along the path ba , the value of k is increased by 1
(i.e., the value of $n-1-k$ is decreased by 1),
- (ii) along the arc d , the value of k is not changed while the value of i is increased by 1 (i.e., the value of $n+1-i$ is decreased by 1),
- (iii) along the arc c , the values of k and i are not changed while j is assigned the value n , and
- (iv) along the arc e , the values of k and i are not changed while the value of j is decreased by 1.

Therefore it follows, by Corollary 2, that

The interpreted graph (Figure 3) terminates.

This implies that our Gaussian elimination program (Figure 1) terminates for every positive integer n .

4.2 Example 2:

Consider the function $\text{gcd}(x,y)$ (McCarthy [1960]). $\text{gcd}(x,y)$ computes the greatest common divisor of x and y (where x and y are positive integers), and is defined recursively using the Euclidean Algorithm by

$$\begin{aligned}\text{gcd}(x,y) &= [x > y \rightarrow \text{gcd}(y,x); \\ &\quad \text{rem}(y,x) = 0 \rightarrow x; \\ &\quad T \rightarrow \text{gcd}(\text{rem}(y,x),x)],\end{aligned}$$

where $\text{rem}(u,v)$ is the remainder of $\frac{u}{v}$.

The Algol meaning of this definition is:

$$\begin{aligned}\text{gcd}(x,y) &= \text{if } x > y \text{ then } \text{gcd}(y,x) \\ &\quad \text{else if } \text{rem}(y,x) = 0 \text{ then } x \\ &\quad \text{else } \text{gcd}(\text{rem}(y,x),x).\end{aligned}$$

We want to show that for every pair (x,y) of positive integers, the recursive process for computing $\text{gcd}(x,y)$ always terminates.

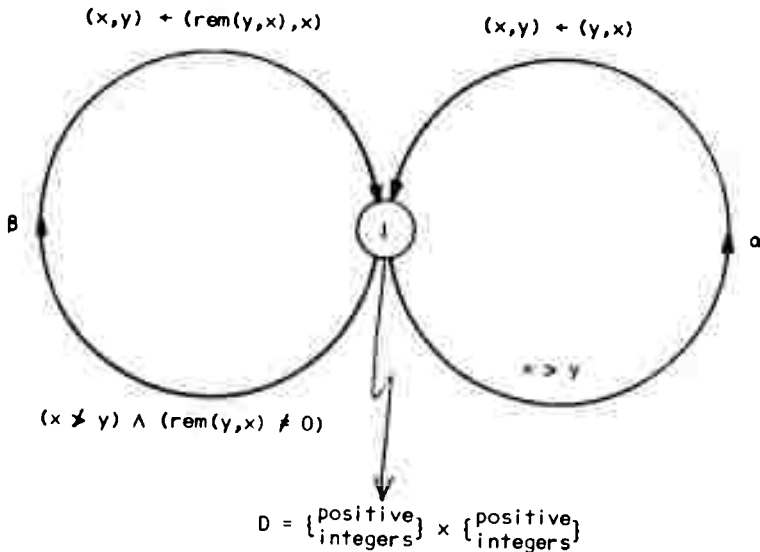


Figure 4

By considering vertex 1 in Figure 4 as representing the start of the computation of gcd, for each pair (x, y) , it follows that:

For every pair of positive integers (x, y) , the recursive process for computing $\text{gcd}(x, y)$ terminates, if and only if

the interpreted graph (Figure 4) terminates.

Since this interpreted graph consists only of one vertex, we shall use Corollary 1 to show its termination.

So, let $\underline{W} = \underline{I}_1^+$ be the well-ordered set, and $\underline{F}(x, y) = \text{rem}(y, x)$ the mapping of D into W .

Since the graph contains two elementary cycles, α and β , we have to show:

1. $\forall(x,y): P_{\alpha}(x,y) = \text{True} \Rightarrow F(x,y) > F(y,x)$, and
2. $\forall(x,y): P_{\beta}(x,y) = \text{True} \Rightarrow F(x,y) > F(\text{rem}(y,x),x)$.

Proof:

1. $P_{\alpha}(x,y) = \text{True} \Rightarrow (x,y) \in D \wedge (x > y)$
 $\Rightarrow (\text{rem}(y,x) = y) \wedge (y > \text{rem}(x,y) \geq 0)$
 $\Rightarrow \text{rem}(y,x) > \text{rem}(x,y)$
 $\Rightarrow F(x,y) > F(y,x)$.
2. $P_{\beta}(x,y) = \text{True} \Rightarrow (x,y) \in D \wedge (x \not> y) \wedge (\text{rem}(y,x) \neq 0) \wedge (\text{rem}(y,x),x) \in D$
 $\Rightarrow (x \text{ positive integer}) \wedge \text{rem}(y,x) \text{ positive integer}$
 $\stackrel{*}{\Rightarrow} \text{rem}(y,x) > \text{rem}(x, \text{rem}(y,x))$
 $\Rightarrow F(x,y) > F(\text{rem}(y,x),x)$.

So by corollary 1, it follows that the interpreted graph (Figure 4) terminates, which implies the desired result.

*Note that for every non-negative integer x , and for every positive z : $z > \text{rem}(x,z) \geq 0$.

REFERENCES

Floyd [1967]

Floyd, R. W., "Assigning Meanings to Programs," Proceedings of Symposia in Applied Mathematics, Volume 19, American Mathematical Society, 19-32 (1967).

McCarthy [1960]

McCarthy, J., "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I," Communications of the ACM 3 (4), 184-195 (April, 1960).