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

Introduction ............................................. 1

Chapter 1: Mathematical Background

1.1 The (First-Order) Predicate Calculus .............. 4
1.2 The Validity-Problem of the Predicate Calculus .... 11
1.3 Directed Graphs ..................................... 12

Chapter 2: Definitions

2.1 Abstract Programs ................................... 14
2.2 Programs ............................................. 17
2.3 Interpreted Programs ................................. 19

Chapter 3: Termination of Programs and Abstract Programs

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

Chapter 4: Equivalence of Programs and Abstract Programs

4.1 The Algorithm to Construct $W_{AP,AP^1}$ (Algorithm 2)... 41
4.2 Equivalence of Programs (Theorem 3) ................. 46
4.3 Equivalence of Abstract Programs (Theorem 4) ....... 50

Chapter 5: Termination of Non-Deterministic Programs and Non-Deterministic Abstract Programs

5.1 Definitions .......................................... 54
5.2 Weak Termination (Theorems 5 and 6) ................. 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 The Algorithm to Construct $u_{up}$ (Algorithm 3)</td>
<td>61</td>
</tr>
<tr>
<td>5.4 Strong Termination of Non-Deterministic Programs (Theorem 7)</td>
<td>64</td>
</tr>
<tr>
<td>5.5 Strong Termination of Non-Deterministic Abstract Programs (Theorem 8)</td>
<td>68</td>
</tr>
<tr>
<td>References</td>
<td>70</td>
</tr>
</tbody>
</table>

**PART II**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>72</td>
</tr>
<tr>
<td>Chapter 1: Mathematical Background</td>
<td></td>
</tr>
<tr>
<td>1.1 Well-Ordered Sets</td>
<td>74</td>
</tr>
<tr>
<td>1.2 Directed Graphs</td>
<td>75</td>
</tr>
<tr>
<td>Chapter 2: Definitions</td>
<td>79</td>
</tr>
<tr>
<td>Chapter 3: Termination of Interpreted Graphs</td>
<td></td>
</tr>
<tr>
<td>3.1 Theorem 1</td>
<td>83</td>
</tr>
<tr>
<td>3.2 Theorem 2</td>
<td>88</td>
</tr>
<tr>
<td>Chapter 4: Applications</td>
<td></td>
</tr>
<tr>
<td>4.1 Example 1</td>
<td>90</td>
</tr>
<tr>
<td>4.2 Example 2</td>
<td>96</td>
</tr>
<tr>
<td>References</td>
<td>99</td>
</tr>
</tbody>
</table>
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 $\mathcal{I}$ for the symbols of AP, a program $(AP, \mathcal{I})$ 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 \((AP, \mathcal{I})\) is said to terminate if all possible executions of the program terminate. An abstract program \(AP\) is said to terminate if for every interpretation \(\mathcal{I}\), the program \((AP, \mathcal{I})\) 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., \(\neg W_{AP}\) 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 \((AP, \mathcal{I})\) and \((AP', \mathcal{I})\) 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 \(\mathcal{I}\), the corresponding programs \((AP, \mathcal{I})\) and \((AP', \mathcal{I})\) 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., \(\neg W_{AP, AP'}\) is valid. Consequently, conclusions
CHAPTER I: 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, \( \sim \land \lor \equiv \)
- primitive constants, T and F.

(b) Constants

- \( n \)-adic function constants, \( f^n_i \) (\( i \geq 1, n \geq 0 \))
  \([f^0_i \text{ are called individual constants}].\)
- \( n \)-adic predicate constants, \( p^n_i \) (\( i \geq 1, n \geq 0 \))
  \([p^0_i \text{ are called propositional constants}].\)

(c) Variables

- individual variables, \( x^i_1 \) (\( i \geq 1 \))
- \( n \)-adic predicate variables, \( q^n_1 \) (\( i \geq 1, n \geq 0 \))
  \([q^0_1 \text{ are called propositional variables}.\)

\[1\text{In the following, we shall use also } y_1 \text{ as individual variables and } a_1 \text{ as individual constants.}\]
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 \( x_1 \) and each individual constant \( f_0 \) is a term;
2. If \( t_1, t_2, \ldots, t_n \) \((n \geq 1)\) are terms, then so is \( f^n(t_1, t_2, \ldots, t_n) \);
3. The terms consist exactly of the expressions generated by 1 and 2.

(b) **Atomic formulas**

1. \( T, F, p_1^0 \) and \( q_1^0 \) are atomic formulas.
2. If \( t_1, t_2, \ldots, t_n \) \((n \geq 1)\) are terms, then the expressions \( p_1^n(t_1, t_2, \ldots, t_n) \) and \( q_1^n(t_1, t_2, \ldots, t_n) \) 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 \( \neg R, (x_1)R \) \([x_1 \text{ is said to be universally quantified}]\), and \( (\exists x_1)R \) \([x_1 \text{ is said to be existentially quantified}]\).
3. If \( R \) and \( S \) are wffs, then so are \( (R \supset S), (R \land S), (R \lor S) \), and \( (R = S) \).
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 interference 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 \( f^n \) which occurs in \( W \), we assign a total function of \( n \) variables ranging over \( D_3 \), whose values are in \( D_3 \). [If \( n = 0 \), the individual constant \( f^0 \) is assigned some fixed element of \( D_3 \).]

2. To each predicate constant \( p^n \) which occurs in \( W \), we assign a total function of \( n \) variables ranging over \( D_3 \), whose values are \( T \) or \( F \). [If \( n = 0 \), the propositional constant \( p^0 \) is assigned the value \( T \) or \( F \).]

Given a wff \( W \) and an interpretation \( \mathfrak{I} \) for \( W \) [notation: \((W,\mathfrak{I})\)]. An assignment \( \Gamma \) for \((W,\mathfrak{I})\) consists of assignments to the variables of \( W \):

1. To each free individual variable \( x_i \) in \( W \), we assign some fixed element of \( D_3 \).

2. To each predicate variable \( q^n \) which occurs in \( W \), we assign a total function of \( n \) variables ranging over \( D_3 \), whose values are \( T \) or \( F \). [If \( n = 0 \), the propositional variable \( q^0 \) is assigned the value \( T \) or \( F \).]

Let \( W \) be a wff. Then given an interpretation \( \mathfrak{I} \) for \( W \) and an assignment \( \Gamma \) for \((W,\mathfrak{I})\) [notation: \((W,\mathfrak{I},\Gamma)\)], a value \( T \) or \( F \) will be assigned to \((W,\mathfrak{I},\Gamma)\). This value is obtained simply by using the assignments of \( \mathfrak{I} \) and \( \Gamma \), interpreting \( F \) as falsehood and \( T \) as truth,
using the usual truth tables of ~, \( \land \), \( \lor \), \( \supset \), and \( = \), and interpreting the universally and existentially quantified variables in the standard way.

(W,3) is said to be:
1. **valid**, if for every assignment \( \Gamma \), (W,3,\( \Gamma \)) has the value T.
2. **satisfiable** (or **consistent**), if (W,3,\( \Gamma \)) has the value T for some assignment \( \Gamma \).
3. **unsatisfiable**, if it is not satisfiable.

Clearly, (W,3) is valid if and only if \( \neg (W,3) \) is unsatisfiable.

A wff W is said to be:
1. **valid**, if for every interpretation 3, (W,3) is valid.
2. **satisfiable** (or **consistent**), if (W,3) is satisfiable for some interpretation 3.
3. **unsatisfiable**, if it is not satisfiable.

Clearly, W is valid if and only if \( \neg W \) 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 \textit{prefix}), and if the sequence is followed by a quantifier free wff (called the \textit{matrix}).

The \textit{disjunction} of the wff's $R_1, R_2, \ldots, R_n$, $n \geq 1$, is the wff $R_1 \lor R_2 \lor \ldots \lor R_n$; their \textit{conjunction} is the wff $R_1 \land R_2 \land \ldots \land R_n$.

A \textit{l literal} is a wff which is either an atomic formula or of the form $\neg R$, where $R$ is atomic.

A \textit{clause} is a disjunction $R_1 \lor R_2 \lor \ldots \lor R_n$ 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 \textit{wff in conjunctive normal form}.

Let $W$ be a wff in prenex normal form. Then the \textit{functional form} of $W$ is defined as follows:

Let the variables in the prefix of $W$ (in order of occurrence) be $x_1, x_2, \ldots, x_N$. Let the existentially quantified variables in the prefix be $x_{i_1}, x_{i_2}, \ldots, x_{i_M}$. 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 $f^q_{i_j}(x_{k_1}, x_{k_2}, \ldots, x_{k_q})$, where $(x_{k_1}), (x_{k_2}), \ldots, (x_{k_q}), q \geq 0$, are all the universal quantifiers that precede $(\exists x_{i_j})$ in the prefix of $W$ and $f^q_{i_j}$
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. $W_1 = \{W | W$ is a wff in prenex-normal form, without function constants, and with prefix of the form $V...Vw...V\}$,
2. $W_2 = \{W | W$ is a wff in prenex-normal form, without function constants, and with prefix of the form $V...VwV...V\}$,
3. $W_3 = \{W | W$ is a wff in prenex-normal form, without function constants, and with prefix of the form $V...VwV...V\}$.

See Ackermann [1954] or Church [1956] Section 46.
1.3 Directed Graphs

A directed graph $G$ is an ordered triple $<V, L, A>$ 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 $(v, \ell, v')$, where $v \in V$, $v' \in V$, and $\ell \in L$. These triples are called the arcs of $G$.

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

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

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

$$(v_1, v_2, v_3, \ldots, v_n), (v_2, v_3, v_4), \ldots, (v_n, v_1)$$

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

We say that the vertices $v_1, v_2, \ldots, v_n$ are on the path, and that the path joins the vertices $v_1$ and $v_n$.
CHAPTER 2: DEFINITIONS

2.1 Abstract Programs

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

1. A finite directed graph \( <V,L,A> \), 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 

    \( \gamma = (\gamma_1, \gamma_2, \ldots, \gamma_m) \), called input variables; and

   (b) a set of \( n, n \geq 1 \), distinct individual variables 

    \( x = (x_1, x_2, \ldots, x_n) \), called program variables.

3. With each arc \( \alpha = (v,i,v') \in A \) there is associated:

   (a) a quantifier free wff \( \varphi_\alpha \) called the test predicate of \( \alpha \); and

   (b) an \( n \)-tuple \( \tau_\alpha = (t_1(\alpha), t_2(\alpha), \ldots, t_n(\alpha)) \) of terms called the assignment function of \( \alpha \).

The wff \( \varphi_\alpha \) does not contain any predicate variables.

\[ 1 \] The intended interpretation is

\( v: \) if \( \varphi_\alpha \) then [replace simultaneously each variable \( x_i \) by \( t_i(\alpha) \) and go to \( v' \)].
The wff $\varphi_\alpha$ and the terms $t^{(\alpha)}$ do not contain individual variables other than $\bar{y}$ and $\bar{x}$.

If $v = S$ (i.e., $\alpha$ is an arc leading from the start vertex) the wff $\varphi_\alpha$ and the terms $t^{(\alpha)}$ 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 $\alpha_1, \alpha_2, \ldots, \alpha_N$ is the set of all arcs leading from $v$, the set of the test predicates $\varphi_{\alpha_1}, \varphi_{\alpha_2}, \ldots, \varphi_{\alpha_N}$ is

- (a) complete, i.e., $(\exists x)(y) \left[ \varphi_{\alpha_1} \lor \varphi_{\alpha_2} \lor \ldots \lor \varphi_{\alpha_N} \right]$ is valid, and
- (b) mutually exclusive, i.e., $(\exists x)(\exists y) [\varphi_{\alpha_i} \land \varphi_{\alpha_j}]$ is unsatisfiable for every pair $(i, j)$, $1 \leq i \neq j \leq N$.

---

We have restricted $\varphi_\alpha$ to be a quantifier free wff. However, all the theorems presented in this work are true also in the case when $\varphi_\alpha$ 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 \( f^m \) 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 \( f^0_1 \) is assigned some fixed element of \( D_\mathfrak{I} \).]

2. To each predicate constant \( p^n \) 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 \( p^0_1 \) 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 \( \mathbb{I} \) (the domain of the Integers),
- \( f(x) \) is \( x + 1 \),
- \( p(x) \) is \( x = 0 \), and
- \( a \) is \( -1 \).
Then the program \((\text{AP*}, \text{S*})\) 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:

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

\[
1: \text{if } x=0 \text{ then } [x + x; \text{ go to 3}] \text{ else } [x + x + 1; \text{ go to 2}];
\]

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

\[
3: \text{if } x=0 \text{ then } [x + x; \text{ HALT}] \text{ else } [x + x + 1; \text{ go to 3}].
\]
2.3 **Interpreted Programs**

Let \((AP,\gamma)\) be a program. Then the result obtained by assigning values \(\gamma, \gamma \in (D_3)^m\), for the input variables \(\gamma\) of the program - is called the **interpreted program** \((AP,\gamma, \gamma)\). (1)

**Example**

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

---

Programs with no input variables (i.e., \(m = 0\)) will be considered as interpreted programs.
The interpreted program \((AP_3, Y)\) defines an execution sequence \(<AP, Y>\) which is a (finite or infinite) sequence of triples

\[(\ell(1), v(1), x(1)), (\ell(2), v(2), x(2)), (\ell(3), v(3), x(3)), \ldots\]

where,

1. \((\ell(j), v(j), x(j)) \in L \times V \times (D_3)^n\) for every \(j, j \geq 1\).

2. \((\ell(1), v(1), x(1))\) is the first triple in the sequence if and only if there exists an arc \(\alpha = (S, \ell(1), v(1)) \in A\) s.t.

   \[\varphi_\alpha(Y) = T \quad \text{and} \quad \overline{x}(1) = \overline{t}_\alpha(Y), (1)\]

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

   \[\varphi_\alpha(x(j), Y) = T \quad \text{and} \quad \overline{x}(j+1) = \overline{t}_\alpha(x(j), Y), (2)\]

4. The sequence is finite and \((\ell(q), v(q), x(q))\), \(q \geq 1\), is the last triple of the sequence if and only if \(v(q) = H\). In

---

1. \(\varphi_\alpha(Y)\) and \(\overline{t}_\alpha(Y)\) stand for the result of substituting \(Y\) for \(Y\) in \(\varphi_\alpha\) and \(\overline{t}_\alpha\).

2. \(\varphi_\alpha(x(j), Y)\) and \(\overline{t}_\alpha(x(j), Y)\) stand for the result of substituting \(x(j)\) for \(x\) and \(Y\) for \(Y\) in \(\varphi_\alpha\) and \(\overline{t}_\alpha\).
this case $x(q)$ is called the value of the execution sequence $<AP,3,Y>$ and is denoted by $\text{val} <AP,3,Y>$.

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 $a = (v(j-1), v(j))$, where $v(0) = S$, and $q_a$ 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 $t_{1,1}(a)$, simultaneously. So, $x(j)$ represents the current value of the program variables $x$ after executing the $j$th step. Execution stops whenever control reaches the halt vertex.

**Example**

The interpreted program $(AP^*,3^*,I)$ defines the following execution sequence $<AP^*,3^*,I>$:

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

Let $(AP,3,Y)$ be an interpreted program, and let $v e V$ be any vertex of $AP$. Let $\delta$ be a specified total predicate from $(D_3)^n$ into $\{T,F\}$. Then,

1. $\delta$ is called a valid predicate of $v$ for $(AP,3,Y)$ if
\(\forall \xi, \exists \xi \in (D_3)^n: \) If there exists a triple of the form \((A, V, \xi)\) in \(<AP, 3, \gamma>\), for some \(A \in L\), then \(6(\xi) = T\).

2. \(6\) is called the \textit{minimal valid predicate} of \(V\) for \((AP, 3, \gamma)\) if

\(\forall \xi, \exists \xi \in (D_3)^n: 6(\xi) = T \text{ if and only if there exists a triple of the form } (A, V, \xi) \text{ in } <AP, 3, \gamma>, \text{ for some } A \in L.\)

\textbf{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 \((AP*, 3*, 1)\).
CHAPTER 3: TERMINATION OF PROGRAMS AND ABSTRACT PROGRAMS

3.1 The Algorithm to Construct \( W_\text{AP} \)

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

Algorithm 1

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

Step 1

Associate with every vertex \( v_i \) of \( \text{AP} \) a predicate variable \( q_i \), where the \( q_i \)'s are distinct \( n \)-adic predicate variables.

Step 2

Let \( \alpha = (v_i, a, v_j) \) be any arc of \( \text{AP} \).

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 \( \alpha \)) as

\[
W_\alpha: q_i(\bar{x}) \land \varphi_\alpha \supset q_j(\bar{t}_\alpha).
\]
But,

1. If \( v_i = S \) (i.e., \( v_i \) is the start vertex of AP), then replace the occurrence of \( q_i(x) \) in \( W_\alpha \) by \( T \), and

2. if \( v_j = H \) (i.e., \( v_j \) is the halt vertex of AP), then replace the occurrence of \( q_j(x) \) in \( W_\alpha \) by \( F \).

Step 3

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

\[
\neg(x)[W_{\alpha_1} \land W_{\alpha_2} \land \ldots \land W_{\alpha_N}].(1)
\]

Note that the input variables \( y \) 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

$$\sim p(x) \land x + f(x)$$  \hspace{1cm} (3)

$$\sim p(x) \land p(x)$$  \hspace{1cm} (4)

$$\sim p(x)$$  \hspace{1cm} (5)

$$\sim p(x)$$  \hspace{1cm} (6)

$$\sim p(x)$$  \hspace{1cm} (7)

$$\sim p(x)$$  \hspace{1cm} (8)

$W_1$:  \hspace{1cm} $T \land \sim p(y) \supset q_1(a)$

$W_2$:  \hspace{1cm} $T \land p(y) \supset q_3(y)$

$W_3$:  \hspace{1cm} $q_1(x) \land \sim p(x) \supset q_2(f(x))$

$W_4$:  \hspace{1cm} $q_1(x) \land p(x) \supset q_3(x)$

$W_5$:  \hspace{1cm} $q_2(x) \land p(x) \supset q_3(a)$

$W_6$:  \hspace{1cm} $q_2(x) \land \sim p(x) \supset F$
$W_7: \ q_3(x) \land \neg p(x) \supset q_3(f(x))$

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

Then by step 3 it follows that,

$W_{AP^*} := (x)[W_1 \land W_2 \land W_3 \land W_4 \land W_5 \land W_6 \land W_7 \land W_8]$
3.2 Termination of Programs

Definition 1

The program \( (AP, \mathcal{S}) \) is said to terminate if \( \forall \gamma, \exists (D_{\mathcal{S}})^m \), the execution sequence \( <AP, \mathcal{S}, \gamma> \) is finite.

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

Theorem 1

The program \( (AP, \mathcal{S}) \) terminates if and only if

\( (W_{AP, \mathcal{S}}) \) is unsatisfiable [or equivalently, \( (\neg W_{AP, \mathcal{S}}) \) is valid].

Proof

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

1. \( (AP, \mathcal{S}) \) does not terminate \( \Rightarrow (W_{AP, \mathcal{S}}) \) is satisfiable.

If the program \( (AP, \mathcal{S}) \) does not terminate, there exists a \( \gamma, \exists (D_{\mathcal{S}})^m \), such that the execution sequence \( <AP, \mathcal{S}, \gamma> \) 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 \( (AP, \mathcal{S}, \gamma) \).

Note that since the execution sequence \( <AP, \mathcal{S}, \gamma> \) 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 \( (AP, \mathcal{S}, \gamma) \).
Let $\Gamma$ 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 $(W_{AP}, 3, \Gamma)$ is $T$, i.e., $(W_{AP}, 3)$ is satisfiable, and this completes the proof in one direction.

2. $(W_{AP}, 3)$ is satisfiable $\Rightarrow$ $(AP, 3)$ does not terminate.

If $(W_{AP}, 3)$ is satisfiable, it means that there exists an assignment $\Gamma$ for $(W_{AP}, 3)$ such that the value of $(W_{AP}, 3, \Gamma)$ is $T$. $\Gamma$ consists of assignments of specified total predicates $\delta_i$, mapping $(D_3)^n$ into $\{T, F\}$, for the predicate variables $q_i$, and an assignment $\bar{y}, \bar{y} \in (D_3)^m$, for the free variables $\bar{y}$.

By the construction of $W_{AP}$ (see Algorithm 1), this implies that each $\delta_i$ is a valid predicate of the vertex $v_i$ for $(AP, 3, \bar{y})$, and therefore that $F$ is a valid predicate of the halt vertex for $(AP, 3, \bar{y})$. This implies that the execution sequence $<AP, 3, \bar{y}>$ is infinite (i.e., execution does not reach the halt vertex). So, $(AP, 3)$ does not terminate.

q.e.d.
Example

Let us consider the program \((A^p, i^p)\), where

1. the abstract program \(A^p\) is

\[
\begin{align*}
S & \xrightarrow{T} x + y \\
& \xrightarrow{\sim p(x)} x + f(x) \\
& \xrightarrow{p(x)} 1 \\
& \xrightarrow{x = 0} H
\end{align*}
\]

and

2. the interpretation \(i^p\) is

\[
D_{i^p} = I^+ \quad (i.e., \text{the domain of the non-negative integers}),
\]

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

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

The program \((A^p, i^p)\) can be represented by the domain \(D_{i^p} = I^+\) and the diagram
Using Algorithm 1 we can construct $W_{AP}$, which is

$$(x)[ T \land T \supset q_1(y)]$$

$\land [q_1(x) \land \neg p(x) \supset q_1(f(x))]$

$\land [q_1(x) \land p(x) \supset F].$ 

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

$\tilde{\sigma}^{AP} : (x)[ T \land T \supset q_1(y)]$

$\land [q_1(x) \land x \neq 0 \supset q_1(x \downarrow 1)]$

$\land [q_1(x) \land x = 0 \supset F].$ 

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

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:

$T_1: \forall x_1[q_1(x_1) \supset \forall x_2[q_1(x_2) \land (x_3 < x_2 \supset \neg q_1(x_3))]]$

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

$T_2: (x)[x \neq 0 \supset x \downarrow 1 < x].$

Thus, in order to prove that $(W_{AP}, \tilde{\sigma})$ is unsatisifiable, we shall prove that $\tilde{\sigma}^{AP} \land T_1 \land T_2$ is unsatisifiable (considering $x = 0$, $x < y$ and $x \downarrow 1$ just as symbols, i.e., the predicates $x = 0$ and $x < y$ as predicate constants and the function $x \downarrow 1$ as function constant).

---

The Proof:

The prenex normal form of \( \exists \neg \exists \exists \forall \wedge T_1 \wedge T_2 \) is:

\[
(\exists x_1)(x_2)(x_3)(x_4) q_1(y)
\]

\[
\wedge \left[ \neg q_1(x) \land x \neq 0 \supset q_1(x \neq 1) \right]
\]

\[
\wedge \left[ q_1(x) \land x = 0 \supset F \right]
\]

\[
\wedge \left[ q_1(x_1) \supset \left[ q_1(x_2) \land \left[ x_3 < x_2 \supset \neg q_1(x_3) \right] \right] \right]
\]

\[
\wedge \left[ x \neq 0 \supset x \neq 1 < x \right].
\]

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 \( W^* \):

\[
(x_1)(x_3)(x_4) q_1(b)
\]

\[
\wedge \left[ \neg q_1(x) \lor x = 0 \lor q_1(x \neq 1) \right]
\]

\[
\wedge \left[ \neg q_1(x) \lor x \neq 0 \right]
\]

\[
\wedge \left[ \neg q_1(x_1) \lor q_1(a) \right]
\]

\[
\wedge \left[ \neg q_1(x_1) \lor x_3 \neq a \lor \neg q_1(x_3) \right]
\]

\[
\wedge \left[ x = 0 \lor x \neq 1 < x \right].
\]

Clearly, \( \exists \neg \exists \exists \forall \wedge T_1 \wedge T_2 \) 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. \( \neg q_1(x), x = 0, q_1(x \neq 1) \)
3. \( \neg q_1(x), x \neq 0 \)
4. \( \neg q_1(x), q_1(a) \)
5. \( \neg q_1(x_1), x_3 \neq a, \neg q_1(x_3) \)
6. \( x = 0, x \leq 1 < x. \)

Then by resolving we obtain:

7. \( q_1(a) \) by 1 and 4 \((x_1 = b)\)
8. \( a \neq 0 \) by 3 \((x = a)\) and 7
9. \( q_1(a + 1) \) by 2 \((x = a)\), 7 and 8
10. \( a + 1 < a \) by 6 \((x = a)\) and 8
11. \( \neg q_1(a + 1) \) by 5 \((x_1 = a, x_3 = a + 1)\), 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., \( (W_{AB}^{\overline{3}}) \) is unsatisfiable. Therefore it follows, by Theorem 1, that the program \( (A^*_{AB}^{\overline{3}}) \) terminates.
3.3 Termination of Abstract Programs

Definition 2

An abstract program $AP$ is said to terminate if for every interpretation $I$, the program $(AP, I)$ 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, $\neg W_{AP}$ is valid].

Proof

$AP$ terminates, if and only if (follows by Definition 2)

for every interpretation $I$, the program $(AP, I)$ terminates, if and only if (follows by Theorem 1)

for every interpretation $I$, $(W_{AP}, I)$ is unsatisfiable, if and only if

$W_{AP}$ is unsatisfiable. $\quad$ 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

$$(x)| \begin{array} {c} T \land \neg p(y) \supset q_1(a)  \\ \land \begin{array} {c} T \land p(y) \supset q_3(y)  \\ \land [q_1(x) \land \neg p(x) \supset q_2(f(x))]  \\ \land [q_1(x) \land p(x) \supset q_3(x)]  \\ \land [q_2(x) \land p(x) \supset q_3(a)]  \\ \land [q_2(x) \land \neg p(x) \supset F]  \\ \land [q_3(x) \land \neg p(x) \supset q_3(f(x))]  \\ \land [q_3(x) \land p(x) \supset F] \end{array} \end{array}$$

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 $W_{AP^*}^!$:

$$(x)| \begin{array} {c} \mu(b, \lor q_1(a))  \\ \land \begin{array} {c} \neg p(b) \lor q_3(b)  \\ \land [\neg q_1(x) \lor p(x) \lor q_2(f(x))]  \\ \land [\neg q_1(x) \lor \neg p(x) \lor q_3(x)]  \\ \land [\neg q_2(x) \lor \neg p(x) \lor q_3(a)]  \\ \land [\neg q_2(x) \lor p(x)] \end{array} \end{array}$$
Clearly, \( W' \) is satisfiable if and only if \( W_{AP^*} \) is satisfiable.

We are going to prove that \( W_{AP^*} \) 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. \( \neg p(b), q_3(b) \)
3. \( \neg q_1(x), p(x), q_2(f(x)) \)
4. \( \neg q_1(x), \neg p(x), q_3(x) \)
5. \( \neg q_2(x), \neg p(x), q_3(a) \)
6. \( \neg q_2(x), p(x) \)
7. \( \neg q_3(x), p(x), q_3(f(x)) \)
8. \( \neg q_3(x), \neg p(x) \).

Then by resolving we obtain

9. \( \neg p(b) \) by 2 & 8 (with \( x = b \))
10. \( q_1(a) \) by 1 & 9
11. \( \neg q_1(x), q_2(f(x)), q_3(x) \) by 3 & 4
12. \( q_2(f(a)), q_3(a) \) by 10 & 11 (with \( x = a \))
13. \( \neg q_2(x), q_3(a) \) by 5 & 6
14. $q_3(a)$ by 12 & 13 (with $x = f(a)$)
15. $\neg q_3(x), q_3(f(x))$ 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. $\neg q_3(a), p(f(a))$ by 8 (with $x = a$) & 18
20. $\neg q_3(a), \neg q_3(f(a))$ by 8 (with $x = f(a)$) & 19
21. $\neg q_3(a)$ 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 Peterson [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. $C_1 = \{AP | AP \text{ is an abstract program without function constants } f^n_i, n \geq 1\}$,

2. $C_2 = \{AP | AP \text{ is an abstract program which has only one program variable } x \text{ (i.e., } n = 1), \text{ and all the occurrences of function constants in } AP \text{ are in terms of the form } f^o_i \text{ or } f^1_i(x)\}$.

3. $C_3 = \{AP | AP \text{ is an abstract program which has only two program variables } x_1 \text{ and } x_2 \text{ (i.e., } n = 2), \text{ and all the occurrences of function constants in } AP \text{ are in terms of the form } f^o_i \text{ or } f^2_i(x_1, x_2)\}$.

**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 $I = 2$, 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

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

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 $f_1^0, f_2^0, \ldots, f_k^0$ and $f_1^1(x), f_2^1(x), \ldots, f_k^1(x)$ ($k, \ell \geq 0$).

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 $f_1^0, f_2^0, \ldots, f_k^0$ and $f_1^1(x), f_2^1(x), \ldots, f_k^1(x)$.

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

$W_{AP}^1$ is satisfiable if and only if $W_{AP}$ is satisfiable, since $W_{AP}$ is the functional form of $W_{AP}^1$. 
Let $W'_{AP}$ be the wff $(w_1)\ldots(w_k)x(z_1)\ldots(z_s)[M']$, i.e., $W'_{AP}$ is just $\neg W'_{AP}$. Clearly, $W''_{AP}$ is valid if and only if $W'_{AP}$ is unsatisfiable.

Since $W''_{AP}$ is in prenex normal form, without function constants, and with prefix of the form $V\ldots V y V\ldots V$, 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 $x = (x_1, \ldots, x_n)$, and
2. they have the same set of input variables $y = (y_1, \ldots, y_m)$.\(^1\)

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:

\(^1\)Note 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 \( W_\alpha \) (the wff of the arc \( \alpha \)) as

\[
W_\alpha: q_i(x) \land q_\alpha \supset q_j(f_\alpha).
\]

But,

if \( v_i = S \) (i.e., \( v_i \) is the start vertex of AP), then replace the occurrence of \( q_i(x) \) in \( W_\alpha \) 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 \( \alpha' \)) as

\[
W_{\alpha'}: q'_i(x) \land q_{\alpha'} \supset q'_j(f_{\alpha'}).
\]

But,

1. if \( v'_i = S' \) (i.e., \( v'_i \) is the start vertex of AP'), then replace the occurrence of \( q'_i(x) \) in \( W_{\alpha'} \) by \( T \), and
2. if \( v_j' = H' \) (i.e., \( v_j' \) is the halt vertex of \( AP' \)), then replace the occurrence of \( q_j'(t, \alpha_j) \) in \( W_{\alpha_j} \) by \( \sim q_j'(t, \alpha_j) \).

**Step 4**

Let \( \alpha_1, \alpha_2, \ldots, \alpha_N \) be the set of all the arcs of \( AP \), and \( \alpha_1', \alpha_2', \ldots, \alpha_M' \) be the set of all the arcs of \( AP' \). Then define \( W_{AP,AP'} \) as

\[
W_{AP,AP'} = (\exists x)[\bigwedge_{\alpha_1} \bigwedge_{\alpha_2} \cdots \bigwedge_{\alpha_N} \bigwedge_{\alpha_1'} \bigwedge_{\alpha_2'} \cdots \bigwedge_{\alpha_M'}].
\]

**Example**

Consider the abstract program \( AP^{**} \):

\[
\sim p(y) \land \sim p(a) \quad (1) \quad x + f(a)
\]

where,

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

\[\text{Note that the input variables of } AP \text{ and } AP' \text{ are free variables in } W_{AP,AP'} \]
Using Algorithm 2 we shall construct the wff $W_{A_{P^*}, A_{P^*}}$, where $A_{P^*}$ is the abstract program that was presented in sec. 2.1.
\[ W_{A_{\mathfrak{P}^*}, A_{\mathfrak{P}^{**}}} : (x) \{ \begin{array}{l} [ \ T \land \neg \phi(y) \supset q_1(a) ] \\
[ \ T \land p(y) \supset q_3(y) ] \\
[ q_1(x) \land \neg p(x) \supset q_2(f(x)) ] \\
[ q_1(x) \land p(x) \supset q_3(x) ] \\
[ q_2(x) \land p(x) \supset q_3(a) ] \\
[ q_2(x) \land \neg p(x) \supset q_H(x) ] \\
[ q_3(x) \land \neg p(x) \supset q_3(f(x)) ] \\
[ q_3(x) \land p(x) \supset q_H(x) ] \\
[ T \land \neg \phi(y) \land \neg p(a) \supset \neg q_H(f(a)) ] \\
[ T \land \neg \phi(y) \land p(a) \supset \neg q_H(a) ] \\
[ T \land p(y) \supset \neg q_H(y) ] \}. \]
4.2 Equivalence of Programs

Definition 4

Let AP and AP' be any two comparable abstract programs.
Let $\mathcal{I}$ be an interpretation that contains assignments for all the constants that occur in AP or AP'.
Then the programs $(AP, \mathcal{I})$ and $(AP', \mathcal{I})$ are said to be comparable.

Definition 5

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

\[
\forall \gamma, \gamma \in (D_{\mathcal{I}})^m, \text{ both execution sequences } <AP, \mathcal{I}, \gamma> \text{ and } <AP', \mathcal{I}, \gamma> \text{ are finite and } \text{val}<AP, \mathcal{I}, \gamma> = \text{val}<AP', \mathcal{I}, \gamma>.
\]

Theorem 3

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

$(W_{AP, AP', \mathcal{I}})$ is unsatisfiable [or equivalently, $(\neg W_{AP, AP', \mathcal{I}})$ is valid].

Proof

We shall prove that:

1. $\exists \gamma, \gamma \in (D_{\mathcal{I}})^m$, such that $<AP, \mathcal{I}, \gamma>$ is infinite,

or 2. $<AP', \mathcal{I}, \gamma>$ is infinite,

or 3. both $<AP, \mathcal{I}, \gamma>$ and $<AP', \mathcal{I}, \gamma>$ are finite, and $\text{val}<AP, \mathcal{I}, \gamma> \neq \text{val}<AP', \mathcal{I}, \gamma>$. 
if and only if

\((W_{AP,AP',\exists})\) is satisfiable.

(i) =

We have to consider three cases:

1. If the execution sequence \(<AP,3,\overline{y}>\) is infinite, then \((W_{AP,AP',\exists})\) is satisfiable, since the value of \((W_{AP,AP',\exists},\Gamma)\) is \(T\), where \(\Gamma\) consists of the following assignments:
   
   (a) \(\overline{y}\) assigned to \(\overline{y}\),
   
   (b) to each occurrence of \(q_i\) in \(W_{AP,AP'}\) assign the minimal valid predicate of \(v_i\) for \((AP,3,\overline{y})\), and
   
   (c) to each occurrence of \(q'_i\) in \(W_{AP,AP'}\) assign the minimal valid predicate of \(v'_i\) for \((AP',3,\overline{y})\).

   The result then follows from the construction of \(W_{AP,AP'}\) (Algorithm 2). Note that, since \(<AP,3,\overline{y}>\) is infinite, the minimal valid predicate of \(H\) for \((AP,3,\overline{y})\) is \(F\), i.e., by our assignment \(q_H = F\), and therefore \(\neg q_H = T\).

2. If the execution sequence \(<AP',3,\overline{y}>\) is infinite, then \((W_{AP,AP',\exists})\) is satisfiable, since the value of \((W_{AP,AP',\exists},\Gamma)\) is \(T\), where \(\Gamma\) consists of the following assignments:
   
   (a) \(\overline{y}\) assigned to \(\overline{y}\),
   
   (b) to each occurrence of \(q_i\) [except \(q_H\)] in \(W_{AP,AP'}\) assign the minimal valid predicate of \(v_i\) for \((AP,3,\overline{y})\),
   
   (c) to each occurrence of \(q'_i\) in \(W_{AP,AP'}\) assign the minimal valid predicate of \(v'_i\) for \((AP',3,\overline{y})\), and
   
   (d) \(q_H = T\).
The result then follows from the construction of \( W_{AP, AP'} \) (Algorithm 2). Note that \( q_H = F \), and since \( <AP', \exists, \gamma> \) is infinite, \( F \) is the minimal valid predicate of \( H' \) for \( <AP', \exists, \gamma> \).

3. If both the execution sequences \( <AP, \exists, \gamma> \) and \( <AP', \exists, \gamma> \) are finite and \( \text{val} <AP, \exists, \gamma> \neq \text{val} <AP', \exists, \gamma> \) then \( (W_{AP, AP'}, \exists) \) is satisfiable, since the value of \( (W_{AP, AP'}, \exists, \gamma) \) is \( T \), where \( \Gamma \) consists of the following assignments:
   
   (a) \( \exists \) assigned to \( \gamma \),
   
   (b) to each occurrence of \( q_i \) in \( W_{AP, AP'} \) assign the minimal valid predicate of \( v_i \) for \( <AP, \exists, \gamma> \), and
   
   (c) to each occurrence of \( q'_i \) in \( W_{AP, AP'} \) assign the minimal valid predicate of \( v'_i \) for \( <AP', \exists, \gamma> \).

The result then follows from the construction of \( W_{AP, AP'} \) (Algorithm 2). Note that we assigned to \( q_H \) the minimal valid predicate \( \delta \) of \( H \) for \( <AP, \exists, \gamma> \), i.e., \( \delta(x) = T \) if and only if \( x = \text{val} <AP, \exists, \gamma> \). Now, since \( \text{val} <AP, \exists, \gamma> \neq \text{val} <AP', \exists, \gamma> \), it follows that \( \delta(\text{val} <AP', \exists, \gamma>) = F \), i.e., \( \neg \delta(\text{val} <AP', \exists, \gamma>) = T \).

\((ii) \Rightarrow \)

We shall prove that if \( (W_{AP, AP'}, \exists) \) is satisfiable with \( \gamma, \gamma \in (D^\exists)^m \), assigned to \( \gamma \), and both execution sequences \( <AP, \exists, \gamma> \) and \( <AP', \exists, \gamma> \) are finite, then \( \text{val} <AP, \exists, \gamma> \neq \text{val} <AP', \exists, \gamma> \).

If \( (W_{AP, AP'}, \exists) \) is satisfiable with \( \gamma \) assigned to \( \gamma \), it means that there exist an assignment \( \Gamma \) such that \( (W_{AP, AP'}, \exists, \gamma) \) is \( T \), where \( \Gamma \)
consists of the assignment of $\bar{y}$ to $\bar{y}$ and assignments of specified total predicates $\delta_i$ and $\delta'_i$ (mapping $D_\overline{3}$ into $\{T,F\}$) for $q_i$ and $q'_i$ respectively.

By the construction of $W_{AP,AP'}$ (Algorithm 2), this implies that each $\delta_i$ is a valid predicate of the vertex $v_i$ for $(AP,\overline{3},\overline{y})$; especially $\delta_H$ is a valid predicate of the halt vertex $H$ for $(AP,\overline{3},\overline{y})$, and therefore $\delta_H(\text{val} \, <AP,\overline{3},\overline{y}>) = T$. Moreover, each $\delta'_i$ is a valid predicate of the vertex $v'_i$ for $(AP',\overline{3},\overline{y})$, and $\overline{\delta}_H$ is a valid predicate of the halt vertex $H'$ for $(AP',\overline{3},\overline{y})$, and therefore $\overline{\delta}_H(\text{val} \, <AP',\overline{3},\overline{y}>) = T$, i.e., $\overline{\delta}_H(\text{val} \, <AP',\overline{3},\overline{y}>) = F$.

But since $\delta_H(\text{val} \, <AP,\overline{3},\overline{y}>) = T$, while $\delta_H(\text{val} \, <AP',\overline{3},\overline{y}>) = F$, it follows that $\text{val} \, <AP,\overline{3},\overline{y}> \neq \text{val} \, <AP',\overline{3},\overline{y}>$.

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, \( \neg W_{AP,AP'} \) 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 \(1 \leq i \leq 3\), 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.
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 $v (v \notin H)$, 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 $(GP, Q)$ and non-deterministic interpreted program $(GP, Q, \gamma)$ 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 \( G_{\Phi^*} \):

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., \(\neg p(x), p(x), \text{ and } \neg p(x) \land p(f(x))\)], are not mutually exclusive -
\(GP^*\) is not an abstract program.

Let \(\mathcal{G}^*\) be the following interpretation of \(GP^*\):

- \(D = \mathbb{I}\) (the domain of the integers),
- \(f(x) = x + 1\),
- \(p(x) = x = 0\), and
- \(a = -2\).

Then the non-deterministic program \((GP^*, \mathcal{G}^*)\) can be represented by
the domain \(D = \mathbb{I}\) and the diagram.
By assigning the value 1 to the variable $y$ of $(\mathcal{G}_y, \mathcal{J}_y)$, we obtain the non-deterministic interpreted program $(\mathcal{G}_y, \mathcal{J}_y, 1)$:

In a non-deterministic interpreted program $(\mathcal{G}_y, \mathcal{J}_y)$ there may exist a vertex $v$ and two distinct arcs $\alpha_1$ and $\alpha_2$ leading from $v$, such that control may reach vertex $v$ with $x = \bar{x}, \bar{x} \in \mathcal{D}_y^n$, while both $\varphi_{\alpha_1}(\bar{x}) = T$ and $\varphi_{\alpha_2}(\bar{x}) = T$. (1)

---

$\varphi_{\alpha_1}(\bar{x})$ and $\varphi_{\alpha_2}(\bar{x})$ stand for the result of substituting $\bar{x}$ for $y$ in $\varphi_{\alpha_1}$ and $\varphi_{\alpha_2}$, respectively.
It follows that in general a non-deterministic interpreted program \( (Q_P, \mathcal{I}, \gamma) \) does not define a unique execution sequence \( <Q_P, \mathcal{I}, \gamma> \) as for interpreted programs (see sec. 2.3), but a set \( \{<Q_P, \mathcal{I}, \gamma>\} \) of execution sequences.

**Example**

The interpreted program \( (Q_P*, \mathcal{I}*, 1) \) defines two execution sequences:

\[(1, 1, -2) (3, 2, -1) (7, H, -1), \text{ and} \]
\[(1, 1, -2) (3, 2, -1) (5, 3, -2) (8, 3, -1) (8, 3, 0) (9, H, 0).\]

Let \( (Q_P, \mathcal{I}, \gamma) \) be a non-deterministic interpreted program, and \( <Q_P, \mathcal{I}, \gamma> \) be any fixed execution sequence of \( \{<Q_P, \mathcal{I}, \gamma>\} \).

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

Then,

1. \( \delta \) is called a **valid predicate of** \( v \) for \( <Q_P, \mathcal{I}, \gamma> \).
   
   If \( \forall \xi, \exists (D_\mathcal{I})^n: \text{ if for some } \xi \in \mathcal{L}, \text{ there exists a triple of the form } (L, v, \xi) \text{ in } <Q_P, \mathcal{I}, \gamma>, \text{ then } \delta(\xi) = T.\)

2. \( \delta \) is called the **minimal valid predicate of** \( v \) for \( <Q_P, \mathcal{I}, \gamma> \).

   If \( \forall \xi, \exists (D_\mathcal{I})^n: \delta(\xi) = T \text{ if and only if } \text{ for some } \xi \in \mathcal{L}, \text{ there exists a triple of the form } (L, v, \xi) \text{ in } <Q_P, \mathcal{I}, \gamma>.\)
5.2 Weak Termination

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

**Definition 7**

A non-deterministic program $(G_P, G)$ is said to **terminate weakly**, if

$$\forall \gamma, \exists (D, \gamma)^m, \text{ there exists at least one finite execution sequence }$$

in $\{<G_P, G, \gamma>\}$.

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

**Theorem 5**

The non-deterministic program $(G_P, G)$ terminates weakly, if and only if

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

**Definition 8**

A non-deterministic abstract program $G_P$ is said to **terminate weakly** if

for every interpretation $G$, the program $(G_P, G)$ 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 $GP$ terminates weakly, if and only if

$W_{GP}$ is unsatisfiable [or equivalently, $\neg W_{GP}$ is valid].
5.3 The Algorithm to Construct $\mathbf{U}_{\mathbf{GP}}$

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

Algorithm 3

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

**Step 1**

Associate with every vertex $v_i$ of $\mathbf{GP}$ 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 $\mathbf{GP}$ ($v_i \neq H$).

Let $\alpha_1, \alpha_2, \ldots, \alpha_N$ be the set of all the arcs leading from $v_i$ to $v_1, v_2, \ldots, v_N$ respectively. In step 1 we have associated with the vertex $v_i$ the predicate variable $q_i$ and with the vertex $v_{ij}$, $1 \leq j \leq N$, the predicate variable $q_{ij}$.

We shall define the wff $\mathbf{W}_{v_i}$ (the wff of the vertex $v_i$) as

$$
\mathbf{W}_{v_i} : q_i(\mathbf{x}) \supset \bigvee_{j=1}^{N} [q_{ij} \land q_{ij}(\mathbf{y})]
$$

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

$$W_{G_P}: \overline{(x)[W_{v_1} \land W_{v_2} \land \ldots \land W_{v_M}]. \tag{1}$$

Note that the input variables $y$ are free variables in $W_{G_P}$. 

1. If $v_1 = S$ (i.e., $v_1$ is the start vertex of $G_P$), then replace the occurrence of $q_1(x)$ in $W_{v_1}$ by $T$, and

2. If $v_1 = H$ (i.e., $v_1$ is the halt vertex of $G_P$), replace the occurrence of $q_j(\alpha_j)$ in $W_{v_1}$ by $F$. 

**Step 3**

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

$$W_{G_P}: \overline{(x)[W_{v_1} \land W_{v_2} \land \ldots \land W_{v_M}]. \tag{1}$$

Note that the input variables $y$ are free variables in $W_{G_P}$. 

1. If $v_1 = S$ (i.e., $v_1$ is the start vertex of $G_P$), then replace the occurrence of $q_1(x)$ in $W_{v_1}$ by $T$, and

2. If $v_1 = H$ (i.e., $v_1$ is the halt vertex of $G_P$), replace the occurrence of $q_j(\alpha_j)$ in $W_{v_1}$ by $F$. 

**Step 3**

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

$$W_{G_P}: \overline{(x)[W_{v_1} \land W_{v_2} \land \ldots \land W_{v_M}]. \tag{1}$$

Note that the input variables $y$ are free variables in $W_{G_P}$. 

1. If $v_1 = S$ (i.e., $v_1$ is the start vertex of $G_P$), then replace the occurrence of $q_1(x)$ in $W_{v_1}$ by $T$, and
Example

The wff \( \mathcal{G}_{D} \) of the non-deterministic abstract program \( G_{D} \) of sec. 5.1 will be constructed as follows:

Combining steps 1 and 2 we obtain

\[
\begin{align*}
\mathcal{W}_2: & \quad T \supset \neg p(x) \land q_1(x) \land \neg p(f(x)) \lor \neg p(x) \land q_3(x) \\
\mathcal{W}_1: & \quad q_1(x) \supset \neg p(x) \land q_2(f(x)) \lor \neg p(x) \land q_3(x) \\
\mathcal{W}_2: & \quad q_2(x) \supset \neg p(x) \land p(f(x)) \land q_3(x) \lor \neg p(x) \land q_3(x) \lor \neg p(x) \land q_3(x) \\
\mathcal{W}_3: & \quad q_3(x) \supset \neg p(x) \land q_3(f(x)) \lor \neg p(x) \land q_3(x) \\
\end{align*}
\]

Then by step 3 it follows that

\( \mathcal{G}_{D} \) is \((x)[\mathcal{W}_1 \land \mathcal{W}_2 \land \mathcal{W}_3] \).
5.4 Strong Termination of Non-Deterministic Programs

**Definition 9**

A non-deterministic program \((G_P, \gamma)\) is said to **terminate strongly** if

\[
\forall \gamma, \gamma \in (D_{\gamma})^m, \text{ all the execution sequences in } \langle G_P, \gamma \rangle \text{ are finite.}
\]

**Theorem 7**

The non-deterministic program \((G_P, \gamma)\) terminates strongly if and only if

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

**Proof**

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

1. \((G_P, \gamma)\) does not terminate strongly if \((G_P, \gamma)\) is satisfiable.

If \((G_P, \gamma)\) does not terminate strongly, there exists a \(\gamma, \gamma \in (D_{\gamma})^m, \) and an execution sequence \(\langle G_P, \gamma, \gamma \rangle, \langle G_P, \gamma, \gamma \rangle \in \langle G_P, \gamma \rangle, \) which is infinite.

Let us assign to each predicate variable \(q_i\) in \(G_P\) the minimal valid predicate of the vertex \(v_i\) for the execution sequence \(\langle G_P, \gamma, \gamma \rangle\).

Note that since the execution sequence \(\langle G_P, \gamma, \gamma \rangle\) 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 \(\langle G_P, \gamma, \gamma \rangle\).

Let \(\Gamma\) consists of the above assignments for the \(q_i's\) and with \(\gamma\) assigned to \(\gamma\). Following the construction of \(G_P\) (see sec. 5.3,
especially note the \( V \) connective used in step 2), it is clear that the value of \( \Phi_{GP,S}^T \) is \( T \), i.e., \( \Phi_{GP,S} \) is satisfiable. This completes the proof in one direction.

2. \( \Phi_{GP,S} \) is satisfiable \( \Rightarrow \) \( \Phi_{GP} \) does not terminate strongly.

If \( \Phi_{GP,S} \) is satisfiable, there exist an assignment \( \Gamma \) for \( \Phi_{GP,S} \) such that the value \( \Phi_{GP,S}^{\Gamma} \) is \( T \). \( \Gamma \) consists of assignments of specified total predicates \( \delta_i \), mapping \( (P_j) \) into \( \{T,F\} \), for the predicate variables \( q_i \), and an assignment \( \overline{y}, \overline{v} \in (D_j) \) for the free variables \( \overline{y} \).

By the construction of \( U_{GP} \), this implies that each \( \delta_i \) is a valid predicate of the vertex \( v_i \) for some execution sequence \( \langle GP, S, \overline{y} \rangle \), \( \langle GP, S, \overline{y} \rangle \in \langle GP, S, \overline{y} \rangle \), and therefore that \( F \) is a valid predicate of the halt vertex for \( \langle GP, S, \overline{y} \rangle \).

This implies that the execution sequence \( \langle GP, S, \overline{y} \rangle \) is infinite (i.e., execution does not reach the halt vertex). So, \( \Phi_{GP,S} \) 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:

\[
F_1(x) = \begin{cases} 
1 & \text{if } x = 0 \\
2 \cdot F_1(x-1) & \text{if } x > 0 \\
F_2(-x) \cdot F_1(x+1) & \text{else}
\end{cases}
\]

\[
F_2(x) = \begin{cases} 
2 & \text{if } x = 0 \\
3 \cdot F_2(x+2) + 7 & \text{if } x < 0 \\
(F_1(1-x))^2 & \text{else}
\end{cases}
\]

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 $\mathcal{I}$, the non-deterministic program $(GP, \mathcal{I})$ 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

$\mathcal{L}_{GP}$ is unsatisfiable (or equivalently, $\neg \mathcal{L}_{GP}$ is valid).

Proof

$GP$ terminates strongly,

if and only if (follows by Definition 10)

for every interpretation $\mathcal{I}$, the non-deterministic program $(GP, \mathcal{I})$

terminates strongly,

if and only if (follows by Theorem 7)

for every interpretation $\mathcal{I}$, $(\mathcal{L}_{GP}, \mathcal{I})$ is unsatisfiable,

if and only if

$\mathcal{L}_{GP}$ 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]

Church [1956]

Davis and Putnam [1960]

Friedman [1963]

Kleene [1950]

Kleene [1967]

Luckham, Park and Paterson [1967]
Mendelson [1964]

Robinson [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 $P_a (D_v + \{T,F\})$, and a total function $f_a (D_v \land P_a \rightarrow D_{v'})$.

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 I: MATHEMATICAL BACKGROUND

1.1 Well-Ordered Sets

A pair \((S, \succ)\) is called an \textit{ordered set}, provided that \(S\) is a set and \(\succ\) 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 \(a \succ b\) or \(b \succ a\);
2. If \(a \succ b\) and \(b \succ c\), then \(a \succ c\) (i.e., the relation is transitive).

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

\textbf{Examples:}

1. \(\mathbb{I}_1^{+}\) - the set of all non-negative integers well-ordered by its natural order, i.e., \([0, 1, 2, 3, \ldots]\).

2. \(\mathbb{I}_n^{+}\) - 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.,

\[
(a_1, a_2, \ldots, a_n) \succ (b_1, b_2, \ldots, b_n)
\]

if and only if

\[a_1 = b_1, a_2 = b_2, \ldots, a_{k-1} = b_{k-1}, a_k > b_k\] for some \(k, 1 \leq k \leq n\).
3. \( I_e^+ \) - 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.,

\[(a_1, a_2, a_3, \ldots) > (b_1, b_2, b_3, \ldots)\]

if and only if

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

1.2 Directed Graphs

A directed graph \( G \) (graph, for short) is an ordered triple \( <V, L, A>\) 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 \( (v, v', v') \), where \( v \in V, v' \in V \) and \( \& \in L \). 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 \((a_1, a_2, a_3, \ldots)\) is in the set if and only if \( \exists L, 1 \leq l, \) s.t.

\[\forall l (1 < l) : a_1 \text{ is a positive integer and } a_l \geq a_{l+1}, \text{ and} \]

\[\forall l (l \geq l) : a_1 = 0.\]

For example, \((5, 5, 4, 3, 3, 3, 3, 1, 0, 0, \ldots)\) is an element in this set.
Let $a = (v, \ell, v')$ be an arc of a directed graph. Then we define:

1. $v$ - the initial vertex of the arc,
2. $\ell$ - 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 $G$

$$(v_1, \ell_1, v_2), (v_2, \ell_2, v_3), \ldots, (v_n, \ell_n, v_{n+1})$$

[notation: $v_1 \xrightarrow{\ell_1} v_2 \xrightarrow{\ell_2} v_3 \ldots v_n \xrightarrow{\ell_n} v_{n+1}$],

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 $v_1, v_2, \ldots, v_{n+1}$, and these vertices are on the path.
2. The path joins the vertices \( v_i \) and \( v_{i+n+1} \).

3. The path is **elementary** if the vertices \( v_i, v_{i+1}, \ldots, v_{i+n+1} \) are distinct.

4. The path is a cycle if the vertex \( v_i \) coincides with the vertex \( v_{i+n+1} \), further it is an **elementary cycle** if in addition the vertices \( v_i, v_{i+1}, \ldots, v_{i+n} \) 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 \( <V,L,A> \). We define a subgraph \( G_1 = <V_1,L,A_1> \) 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 \( G_1 = <V_1,L,A_1> \) 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 \( V_2 \supseteq V_1 \), the subgraph \( G_2 = <V_2,L,A_2> \) is not strongly connected.
A tree $T = <V, L, A, r>$ is a directed graph $<V, L, A>$ 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.
CHAPTER 2: DEFINITIONS

An interpreted graph \( IG \) consists of a finite directed graph \(<V,L,A>\), and

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

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

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

\[
(v^{(o)},x^{(o)}) \xrightarrow{L^{(0)}} (v^{(1)},x^{(1)}) \xrightarrow{L^{(1)}} (v^{(2)},x^{(2)}) \xrightarrow{L^{(2)}} \ldots ,
\]

where,

1. \( v^{(j)} \in V \), \( L^{(j)} \in L \) and \( x^{(j)} \in D_v \) 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. \( P_a \times^{(j)} = \text{True and } f_a \times^{(j)} = x^{(j+1)} \).
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 \( v^{(n)} \):
   \( P_a \times^{(n)} = \text{False} \).
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 \( (v^{(0)}, x^{(0)}) \in V \times D^{(0)}_v \), 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 \( V', L', (v^{(0)}, x^{(0)}) \), 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 \( ((v, x), \ell, (v', y)) \in V' \times L \times V' \) s.t. there exists an \( (v^{(0)}, x^{(0)}) \) - execution sequence of IG that contains \( (v, x) \ell (v', y) \).
4. \( (v^{(0)}, x^{(0)}) \in V' \) is the root-vertex of the tree.
Example

Consider the interpreted graph $I_{\mathbb{Z}}$

(1, -4) → (2, -2) → (2, 0),

(11) (1, -4) → (2, 4) → (1, -3) → (2, -1) → (2, 1) → (1, 0), and

(111) (1, -4) → (2, 4) → (1, -3) → (2, 3) → (1, -2) → (2, 2) → (1, -1) → (2, 1) → (1, 0).

The execution tree $T(1, -4)$ of $I_{\mathbb{Z}}$ is:
(1,0)

(2,1)

(2,-1)

(2,2)

(2,-2)

(1,-1)

(1,-2)

(2,4)

(1,-3)

(2,4)

(1,-4)
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 \( \alpha = (a_1, a_2, \ldots, a_{q}) \), where \( a_j = (v(j), e(j), v(j+1)) \) \( \epsilon \mathbb{A} \) for \( 1 \leq j \leq q \), be any path of an interpreted graph. Then let

1. \( f^\alpha x \) stand for \( f(\ldots(f(a_2 x))\ldots) \), and
2. \( P^\alpha x \) stand for

\[
\forall v, x \in V : (\forall a_q \in A : (\exists a_{q-1} \in A : (\ldots (f(a_2 x) a_1 x \ldots))) \land (f^\alpha x) v \in D(v+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 \( a = (v, e, v') \) of \( IG \) and for every \( x \) s.t. \( P^\alpha x = True \):

\[
F_v(x) > F_{v'}(f_a(x)).
\]

\(^1\) i.e., \( (v, x), (v, x) \epsilon V \times D_v \), all the \( (v, x) \) - execution sequences are finite.
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 $W_v = (S_v, >_v)$ and a total function $F_v$ which maps $D_v$ into $S_v$, such that,
3. For every cycle $\alpha$ of IG:

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 Konig'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.
\[ \text{Proof} \]

- **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 \( \gamma \) in IG,

  \[ \gamma: (v(0), x(0)) \xrightarrow{l(0)} (v(1), x(1)) \xrightarrow{l(1)} (v(2), x(2)) \xrightarrow{l(2)} \ldots \]

  Let \( \gamma' \) be the infinite path

  \[ \gamma': v(0) \xrightarrow{l(0)} v(1) \xrightarrow{l(1)} v(2) \xrightarrow{l(2)} \ldots \]

  Since IG, by definition, consists of a finite directed graph, and since \( \gamma' \) is an infinite sequence - it follows, that there exists at least one elementary cycle \( B \) in IG, that occurs (as a subpath) infinitely many times in \( \gamma' \).
Since $V^*$ is a cut set, it follows that there exists a vertex $v^* \in V^*$ that is on $B$. This implies that $v^*$ must occur infinitely many times in $\gamma'$.

Let $v^{(1)}, v^{(2)}, v^{(3)}, \ldots$ ($0 \leq n_j < n_{j+1}$ for $j \geq 1$), be the infinite sequence of all occurrences of the vertex $v^*$ in $\gamma'$. Therefore, the infinite execution sequence $\gamma$ can be written as

$$
\gamma: \quad (v^{(o)}, x^{(o)}) \xrightarrow{\gamma} (v^{(1)}, x^{(1)}) \xrightarrow{\gamma} \ldots
$$

Then, by condition (3) it follows that

$$
F_{v^*}(x^{(1)}) >_{v^*} F_{v^*}(x^{(2)}) >_{v^*} F_{v^*}(x^{(3)}) >_{v^*} \ldots
$$

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 \( \alpha: v^* \rightarrow \cdots \rightarrow v^* \) 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 \( \alpha \) of IG:

\[ v^{(1)} \xrightarrow{L(1)} v^{(2)} \xrightarrow{L(2)} \cdots \xrightarrow{L(q-1)} v^{(q)} \]

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

\[ F(x) = \text{True} \]
and for every $x$ s.t. $P^x(x) = \text{True}$:

$$F^v(I)(x) > F^v(q)(f^v(x)).$$

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 $G' = <V',L,A'>$ 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 \( \gamma \) in IG,

\[
\gamma: (v^{(0)}, x^{(0)}) \xrightarrow{(0)} (v^{(1)}, x^{(1)}) \xrightarrow{(1)} (v^{(2)}, x^{(2)}) \xrightarrow{(2)} \ldots
\]

Let \( \gamma' \) be the infinite path

\[
\gamma': v^{(0)} \xrightarrow{(0)} v^{(1)} \xrightarrow{(1)} v^{(2)} \xrightarrow{(2)} \ldots
\]

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 \( \gamma' \) only a finite number of times. Let \( v^{(1)}, v^{(2)}, \ldots, v^{(q)} \) (\( 0 \leq n_j < n_{j+1} \) for \( 1 \leq j < q \)), be the list of their occurrences in \( \gamma' \).

It follows that all the vertices \( v^{(j)} \) (\( j > n_q \)) of \( \gamma' \), 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 \( \gamma \):

\[
(n_{q+1}) \xrightarrow{(n_{q+1})} (n_{q+2}) \xrightarrow{(n_{q+2})} (n_{q+2}) \xrightarrow{(n_{q+2})} \ldots
\]

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

q.e.d.
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 $\xi$ 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:

\[
D + a_{ij} , \quad D + D \cdot a_{kk} , \quad \text{and} \quad a_{ij} + a_{ij} = \frac{a_{ik}}{a_{kk}} \cdot a_{kj},
\]

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

\[ W^* = \{2, 3\} \] is the cut set,

\[ W = I_3^+ \] is the well-ordered set,

\[ F_2(i,j,k) = (n-l-k, n+l-i, n+l) \] is the mapping of \( D_2 \) into \( W \), and

\[ F_3(i,j,k) = (n-l-k, n+l-i, j) \] 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 \( n-l-k \) 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 \( n+l-i \) 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

\[
\text{gcd}(x,y) = \begin{cases} 
  x > y & \Rightarrow \text{gcd}(y,x); \\
  \text{rem}(y,x) = 0 & \Rightarrow x; \\
  T & \Rightarrow \text{gcd}(\text{rem}(y,x),x),
\end{cases}
\]

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

The Algol meaning of this definition is:

\[
\text{gcd}(x,y) = \begin{cases} 
  \text{if } x > y \text{ then } \text{gcd}(y,x); \\
  \text{if } \text{rem}(y,x) = 0 \text{ then } x; \\
  \text{else } \text{gcd}(\text{rem}(y,x),x).
\end{cases}
\]

We want to show that for every pair \((x,y)\) of positive integers, the recursive process for computing \( \text{gcd}(x,y) \) always terminates.
By considering vertex \( I \) in Figure 4 as representing the start of the computation of \( \text{gcd} \), 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 \( W = I_1^+ \) be the well-ordered set, and \( F(x,y) = \text{rem}(y,x) \) the mapping of \( D \) into \( W \).
Since the graph contains two elementary cycles, \( \alpha \) and \( \beta \), 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. \( P_\alpha(x,y) = \text{True} \Rightarrow (x,y) \in D \land (x > y) \)
   \[ \Rightarrow (\text{rem}(y,x) = y) \land (y > \text{rem}(x,y) \geq 0) \]
   \[ \Rightarrow \text{rem}(y,x) > \text{rem}(x,y) \]
   \[ \Rightarrow F(x,y) > F(y,x). \]

2. \( P_\beta(x,y) = \text{True} \Rightarrow (x,y) \in D \land (x \neq y) \land (\text{rem}(y,x) \neq 0) \land (\text{rem}(y,x),x) \in D \)
   \[ \Rightarrow (x \text{ positive integer}) \land (\text{rem}(y,x) \text{ positive integer}) \]
   \[ \Rightarrow \text{rem}(y,x) > \text{rem}(x,\text{rem}(y,x)) \]
   \[ \Rightarrow F(x,y) > F(\text{rem}(y,x),x). \]

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]


McCarthy [1960]