PROGRAM FLOW ANALYSIS:
THEORY AND APPLICATIONS

Steven S. Muchnick
University of California
Berkeley, California
University of Kansas
Lawrence, Kansas

Neil D. Jones
Aarhus University
Aarhus, Denmark
University of Kansas
Lawrence, Kansas

10-1. INTRODUCTION

In the first part we establish general mathematical techniques useful in the task of analyzing semantic properties of programs. In the second part, we describe an algorithmic and hence approximate solution to the problem of analyzing semantic properties of programs.

The term "program analysis" will be given a precise meaning, but is better introduced by the following:

Example 10-1. Consider the program:

[1] while $x \geq 1000$ do
[2] \quad $x := x + y$;
[3] od;
[4]
where \( x \) and \( y \) are integer variables taking their values in the set \( I \) of integers included between \(-b - 1\) and \( b \) where \( b \) is the greatest machine-representable integer.

By "analysis of the semantic properties" of that program we understand the determination that:

1. The execution of that program starting from the initial value \( x_0 \in I \) and \( y_0 \in I \) of \( x \) and \( y \) terminates without run-time error if and only if \((x_0 < 1000) \lor (y_0 < 0)\).
2. The execution of the program never terminates if and only if \((1000 \leq x_0 \leq b) \land (y_0 = 0)\).
3. The execution of the program leads to a run-time error (by overflow) if and only if \((x_0 \geq 1000) \land (y_0 > 0)\).
4. During any execution of the program the following assertions \( P_i \) characterize the only possible values that the variables \( x \) and \( y \) can possess at program point \( i \):

\[
P_1 = \langle x, y \rangle.[(\neg b - 1 \leq x \leq b) \land (\neg b - 1 \leq y \leq b)]
\]
\[
P_2 = \langle x, y \rangle.[(1000 \leq x \leq b) \land (\neg b - 1 \leq y \leq b)]
\]
\[
P_3 = \langle x, y \rangle.[(1000 + y \leq x \leq \text{min}(b, b + y))
\]
\[\land (\neg b - 1 \leq y \leq b)]
\]
\[
P_4 = \langle x, y \rangle.[(\neg b - 1 \leq x < 1000) \land (\neg b - 1 \leq y \leq b)]
\]

10-2. SUMMARY

In Section 10-3 we define what we mean by flowchart programs; that is, we define their abstract syntax and operational semantics. A program defines a dynamic discrete system [Kell76, Pnu77] that is a transition relation on states. In Section 10-4 we set up general mathematical methods useful in the task of analyzing the behavior of dynamic discrete systems. In order to make this mathematically demanding section self-contained, lattice-theoretical theorems on fixed points of isotone or continuous maps are first introduced in a separate subsection. The main result of Section 10-4 shows that the predicates characterizing the descendants of the entry states, the ancestors of the exit states, the states which lead to an error, and the states which cause the system to diverge are the least or greatest solution to forward or backward fixed point equations. This result is completed by the proof that whenever a forward equation (corresponding to postconditions) is needed, a backward equation (corresponding to preconditions) can be used instead, and vice versa. Finally we show that when the set of states of the dynamic discrete system is partitioned, the forward or backward equation can be decomposed into a system
of equations. Numerous examples of applications are given which provide for a very concise presentation and justification of classical [Floy67, Naur66, King69, Hoar69, Dijk76] or innovative program proving methods. Section 10-5 tailors the general mathematical techniques previously set up for analyzing the behavior of a deterministic discrete dynamic system to suit the particular case when the system is a program. Two main theorems make explicit the syntactic construction rules for obtaining the systems of semantic backward or forward equations from the text of a program. The facts that the extreme fixed points of these systems of semantic equations can lead to complete information about program behavior and that the backward and forward approaches are equivalent are illustrated on the simple introductory example.

In the second part we briefly survey our joint work with Radha Cousot on the automatic synthesis of approximate invariant assertions for programs. Because of well-known unsolvability problems, the semantic equations which have been used in Section 10-5 for program analysis cannot be algorithmically solved. Hence we must limit ourselves to constructive methods which automatically compute approximate solutions. Such approximate information about the program behavior is often useful, e.g., in program verification systems, program debugging systems, optimizing compilers, etc. Approximate solutions to the semantic equations can be obtained by first simplifying these equations (Section 6.1 of [Cous79]) and next solving the simplified equations associated with the program text, using any chaotic iteration technique [Cous77b, Cous77c]. In Section 10-6.2 we show that when the exact solution to the simplified equations is obtained only after an infinite number of iteration steps, the convergence of the iterates can be sped up using an extrapolation technique based on a widening or narrowing operator [Cous77a]. A hierarchy of examples taken from [Cous77a] and [Cous78] illustrates the approximate program analysis method.

10-3. ABSTRACT SYNTAX AND OPERATIONAL SEMANTICS OF PROGRAMS

10-3.1. Abstract Syntax

Informally, programs will be abstractly represented as single-entry, single-exit directed graphs with edges labeled with instructions.

Example 10-2. The program of Example 10-2 will be represented by Fig. 10-1.

A program graph is a quadruple \( \langle V, \epsilon, \omega, E \rangle \) where \( V \) is a finite set of vertices, \( E \subseteq V \times V \) is a finite set of edges, and \( \epsilon \in V, \omega \in V \) are distinct entry and exit vertices such that \( \epsilon \) is of in-degree 0, \( \omega \) is of out-degree 0, and every vertex lies on a path from \( \epsilon \) to \( \omega \).
Let $\vec{v}$ be a vector of variables taking their values in a universe $U$. The set $I(U)$ of instructions is partitioned into a subset $I_a(U)$ of assignments and a subset $I_t(U)$ of tests. An assignment $\vec{v} := f(\vec{v})$ is represented as a partial map from $U$ into $U$. A test is represented as a partial map from $U$ into $B = \{true, false\}$.

A program is a triple $\langle G, U, L \rangle$ where the program graph $G$, the universe $U$, and the labeling $L \in (E \rightarrow I(U))$ are such that for every nonexit vertex $n$ in $G$ either $n$ is of out-degree 1 and the edge leaving $n$ is labeled with an assignment or $n$ is of out-degree 2 and the edges leaving $n$ are labeled with tests $p$ and $\neg p$.

10.3.2. Operational Semantics

The operational semantics of a syntactically valid program $\pi$ specifies the sequence of successive states of the computation defined by $\pi$.

10.3.2.1. States

The set $S$ of states is the set of pairs $\angle c, m \rangle$ where $c \in V \cup \{\xi\}$ is the control state and $m \in U$ is the memory state. $\xi \notin V$ is the error control state.

The entry, exit, and erroneous states are respectively characterized by $v_e = \lambda(c, m). (c = e), v_o = \lambda(c, m). (c = o)$ and $v_\xi = \lambda(c, m). (c = \xi)$.

10.3.2.2. State transition function

A program $\pi = (G, U, V)$ defines a state transition function $T : (S \rightarrow S)$ as follows:

1. $T(\angle \xi, m \rangle) = \angle \xi, m \rangle$ (no run-time error recovery is available)
2. $T(\angle o, m \rangle) = \angle o, m \rangle$
3. If $c_1 \in V$ is of out-degree 1, $\angle c_1, c_2 \rangle \in E, L(\angle c_1, c_2 \rangle) = f, f \in I_a(U)$ then if $m \in dom(f)$ then $T(\angle c_1, m \rangle) = \angle c_2, f(m) \rangle$ else $T(\angle c_1, m \rangle) = \angle \xi, m \rangle$. 

306
4. If \( c_1 \in V \) is of out-degree 2, \( \langle c_1, c_2 \rangle \in E, \langle c_1, e_1 \rangle \in E, L(\langle c_1, c_2 \rangle) = p, L(\langle c_1, e_1 \rangle) = \neg p \), \( p \in I(U) \) then if \( m \notin \text{dom}(p) \) then
\[
\mathcal{T}(\langle c_1, m \rangle) = \langle \xi, m \rangle
\]
else if \( p(m) \) then \( \mathcal{T}(\langle c_1, m \rangle) = \langle c_2, m \rangle \) else
\[
\mathcal{T}(\langle c_1, m \rangle) = \langle c_2, m \rangle.
\]

The state transition relation \( \tau \in ((S \times S) \to B) \) defined by \( \pi \) is \( \lambda s_1, s_2 \rangle(s_2 = \mathcal{T}(s_1)) \).

10-3.2.3. Transitive closure of a binary relation

If \( \alpha, \beta \in (S \times S \to B) \) are two binary relations on \( S \), their product \( \alpha \circ \beta \) is defined as \( \lambda s_1, s_2 \rangle (\exists s_2 \in S: \alpha(s_1, s_3) \land \beta(s_3, s_2)) \). For any natural number \( n \), the \( n \)-extension \( \alpha^n \) of \( \alpha \) is defined recursively as \( \alpha^0 = \varepsilon = \lambda s_1, s_2 \rangle(1 \leq 0) : \alpha(s_1, s_2) \).

The (reflexive) transitive closure of \( \alpha \) is \( \alpha^* = \lambda s_1, s_2 \rangle (\exists n \geq 0 : \alpha^n(s_1, s_2)) \).

10-3.2.4. Execution and output of a program

The execution of the syntactically valid program \( \pi \) starting from an initial state \( s_1 \in S \) is said to lead to an error iff \( \exists s_2 \in S : \tau^*(s_1, s_2) \land v(s_2) \), and to terminate iff \( \exists s_2 \in S : \tau^*(s_1, s_2) \land v(s_2) \). Otherwise it is said to diverge. The output of the execution of a syntactically valid program \( \pi \) starting from an initial state \( \langle \epsilon, m_1 \rangle \in S \) is defined if and only if this execution terminates with \( m_2 \in U \) such that \( \tau^*(\langle \epsilon, m_1 \rangle, \langle \omega, m_2 \rangle) \), and \( m_2 \) is the output.

10-4. ANALYSIS OF THE BEHAVIOR
OF A DISCRETE DYNAMIC SYSTEM

In order to establish general mathematical techniques useful in analyzing semantic properties of programs, we use the model of discrete dynamic systems. The advantage is that the reasoning on a set \( S \) of states and a state transition relation \( \tau \) leads to very concise notations, terse results, and brief proofs. Another benefit is that the applications of the mathematical techniques for analyzing the behavior of a dynamic discrete system are not necessarily confined within computer science.

10-4.1. Discrete Dynamic Systems

A discrete dynamic system is a 5-tuple \( \langle S, \tau, v_e, v_m, v_f \rangle \) such that \( S \) is a nonvoid set of states, \( \tau \in ((S \times S) \to B) \) where \( B = \{ \text{true}, \text{false} \} \) is the transition relation holding between a state and its possible successors, \( v_e \in (S \to B) \) characterizes the entry states, \( v_m \in (S \to B) \) characterizes the exit states, and \( v_f \in (S \to B) \) characterizes the erroneous states. It is assumed that the entry, exit, and erroneous states are disjoint \( (\forall i, j \in \{ \epsilon, \omega, \xi \}, (i \neq j) \to (\forall s \in S, v_i(s) \land v_f(s))) \).
The following study is devoted to total ( \( \forall s_1, s_2 \in S \exists s_3 \in S: \tau(s_1, s_3) \) ) and deterministic ( \( \forall s_1, s_2, s_3 \in S, (\tau(s_1, s_2) \land \tau(s_1, s_3)) \Rightarrow (s_2 = s_3) \) ) dynamic discrete systems.

A program as defined in Section 10-3 defines a total and deterministic discrete dynamic system. Moreover the entry states are exogenous ( \( \forall s_1, s_2 \in S, \tau(s_1, s_2) \Rightarrow -(\nu_2(s_2)) \) ), the exit states are stable ( \( \forall s_1, s_2 \in S, (\nu_2(s_1) \land \tau(s_1, s_2)) \Rightarrow (s_1 = s_2) \) ), and the system is without error recovery ( \( \forall s_1, s_2 \in S, (\nu_2(s_1) \land \tau(s_1, s_2)) \Rightarrow s_2 \) ).

The inverse of \( \tau \in ((S \times S) \rightarrow B) \) is \( \tau^{-1} = \lambda (s_1, s_2)[\tau(s_2, s_1)] \). A system is injective if \( \tau^{-1} \) is deterministic; it is invertible if it is injective and \( \tau^{-1} \) is total. In general a program does not define an injective dynamic discrete system.

### 10-4.2. Fixed Point Theorems for Isotone and Continuous Operators on a Complete Lattice

This section recalls the lattice-theoretic definitions [Birk67] and theorems which are needed below.

A partially ordered set (poset) \( L(\sqsubseteq) \) consists of a nonempty set \( L \) and a binary relation \( \sqsubseteq \) on \( L \) which is reflexive (\( \forall a \in L, a \sqsubseteq a \) ) and antisymmetric (\( \forall a, b \in L, (a \sqsubseteq b \land b \sqsubseteq a) \Rightarrow (a = b) \) ) and transitive (\( \forall a, b, c \in L, (a \sqsubseteq b \land b \sqsubseteq c) \Rightarrow (a \sqsubseteq c) \) ). Given \( H \sqsubseteq L, a \in L \) is an upper bound of \( H \) if \( b \sqsubseteq a \) for all \( b \in H \). \( a \) is called the least upper bound of \( H \), in symbols \( \bigwedge H \), if \( a \) is an upper bound of \( H \) and if for any upper bound \( b \) of \( H, a \sqsubseteq b \). The dualized notions (that is all \( \sqsubseteq \) are replaced by the inverse \( \sqsupseteq \) ) are the ones of lower bound and greatest lower bound. \( L(\sqsupseteq) \) is a complete lattice if the least upper bound \( \bigwedge H \) of \( H \) and the greatest lower bound \( \bigvee H \) of \( H \) exist for all \( H, H \sqsubseteq L \). A complete lattice \( L \) has an infimum \( \bot = \bigwedge L \) and a supremum \( \top = \bigvee L \).

An operator \( f \) on \( L \) is strict if \( f(\bot) = \bot \), and isotone iff (\( \forall a, b \in L, (a \sqsubseteq b) \Rightarrow (f(a) \sqsubseteq f(b)) \)). \( a \in L \) is a fixed point of \( f \) iff \( f(a) = a \). Tarski's Fixed Point Theorem states that the set of fixed points of an isotone operator \( f \) on a complete lattice \( L(\sqsubseteq, \bot, \top, \sqcup, \sqcap) \) is a (nonempty) complete lattice with partial ordering \( \sqsubseteq \). The least fixed point of \( f \), in symbols \( \operatorname{lfp}(f) \), is \( \bigwedge \{x \in L : f(x) \sqsubseteq x \} \). Dually the greatest fixed point of \( f \), in symbols \( \operatorname{gfp}(f) \) is \( \bigvee \{x \in L : x \sqsubseteq f(x) \} \). An element \( a \) of \( L \) such that \( a \sqsubseteq f(a) \) (respectively \( f(a) \sqsubseteq a \) ) is called a pre-fixed point (post-fixed point) of \( f \).

Let \( f \) be an isotone operator on the complete lattice \( L \). The Recursion Induction Principle follows from Tarski's Fixed Point Theorem and states that (\( \forall x \in L, f(x) \sqsubseteq x \Rightarrow (\operatorname{lfp}(f) \sqsubseteq x) \)). The Dual Recursion Induction Principle is (\( \forall x \in L, (x \sqsubseteq f(x)) \Rightarrow (x \sqsubseteq \operatorname{gfp}(f)) \)).

If \( L(\sqsubseteq, \bot, \top, \sqcup, \sqcap) \) is a complete lattice, then the set \( (M \rightarrow L) \) of total maps from the set \( M \) into \( L \) is a complete lattice \( (M \rightarrow L)(\sqsubseteq, \bot', \top', \sqcup', \sqcap') \) for the pointwise ordering \( \preceq g \) iff (\( \forall x \in L, f(x) \sqsubseteq g(x) \)). In the
following the distinction between $\subseteq, \sqsubseteq, \top, \sqcup, \cap$ and $\subseteq', \sqsubseteq', \top', \sqcup', \cap'$. The set $L^*$ of $n$-tuples of elements of $L$ is a complete lattice for the componentwise ordering $\langle a_1, \ldots, a_n \rangle \subseteq \langle b_1, \ldots, b_n \rangle$ iff $a_i \subseteq b_i$ for $i = 1, \ldots, n$. The set $2^n$ of subsets of $L$ is a complete lattice $2^n(\subseteq, \emptyset, L, \cup, \cap)$. A map $f \in (M \to L)$ will be extended to $(M^* \to L^*)$ as $\lambda x_1, \ldots, x_n \lambda \langle f(x_1), \ldots, f(x_n) \rangle$ and to $(2^M \to 2^n)$ as $\lambda S \{ f(x) : x \in S \}$.

A sequence $x_0, x_1, \ldots, x_n, \ldots$ of elements of $L(\subseteq)$ is an increasing chain iff $x_0 \subseteq x_1 \subseteq \ldots \subseteq x_n \subseteq \ldots$. An operator $f$ on $L(\subseteq, \top, \sqcup, \cap)$ is semi-\sqcup-continuous iff for any chain $C = \{ x_i : i \in \Delta, C \subseteq L, f(\sqcup C) = \sqcup f(C) \}$ Kleene's Fixed Point Theorem [Klee52] states that the least fixed point of a semi-\sqcup-continuous operator $f$ on $L(\subseteq, \top, \sqcup, \cap)$ is equal to $\sqcup [f^n(\bot) : i \geq 0]$ where $f^i$ is defined by recurrence as $f^0 = \lambda x. [x], f^{i+1} = \lambda x. [f(f^i(x))]$.

A poset $L(\subseteq)$ is said to satisfy the ascending chain condition if any increasing chain terminates, that is if $x_i \in L, i = 0, 1, 2, \ldots$, and $x_0 \subseteq x_1 \subseteq \ldots \subseteq x_n \subseteq \ldots$, then for some $m$ we have $x_m = x_{m+1} = \ldots$. An operator $f$ on $L(\subseteq, \top, \sqcup, \cap)$ which is semi-$\sqcup$-continuous is necessarily isotope, but the converse is not true in general. However if $f$ is an isotone operator on a complete lattice satisfying the ascending chain condition, then $f$ is semi-$\cup$-continuous. Also an operator $f$ on a complete lattice $L$ which is a complete-$\sqcup$-morphism (i.e., $\forall H \subseteq L, f(\sqcup H) = \sqcup f(H)$) is obviously semi-$\sqcup$-continuous.

Dual results hold for decreasing chains, semi-$\sqcap$-continuous operators, descending chain conditions, and complete-$\sqcap$-morphisms.

Suppose $L(\subseteq, \top, \sqcup, \cap), L'(\subseteq', \top', \sqcup', \cap')$ are complete lattices and we have the commuting diagram of isotone functions shown in Fig. 10-2, where $h$ is strict ($h(\top) = \top'$) and semi-$\sqcup$-continuous. Then $h(\text{ifp}(f)) = \text{ifp}(g)$.

![Figure 10-2](image)

In a complete lattice $L(\subseteq, \top, \sqcup, \cap)$, $a$ is a complement of $b$ if $a \sqcup b = \top$ and $a \sqcap b = \bot$. A uniquely complemented complete lattice $L(\subseteq, \top, \sqcup, \cap, \neg)$ is a complete lattice in which every element $a$ has a unique complement $\neg a$. Park's Theorem [Park69] states that if $f$ is an isotone
operator on a uniquely complemented complete lattice \( L(\subseteq, \bot, \top, \cup, \cap) \) then \( \lambda x.[\neg f(\neg x)] \) is an isotone operator on \( L \), \( gfp(f) = \neg \neg \lambda x.[\neg f(\neg x)] \).

Let \( L(\subseteq, \bot, \top, \cup, \cap) \) be a complete lattice, \( n \geq 1 \), and \( F \) a semi-\( \bot \)-continuous operator on \( L^n \). The system of equations
\[
X = F(X)
\]
which can be detailed as
\[
X_j = F(X_1, \ldots, X_n) \quad j = 1, \ldots, n
\]
has a least solution which is the least upper bound of the sequence \( \{X^i: i \geq 0\} \) where \( X^0 = \langle \bot, \ldots, \bot \rangle \) and \( X^{i+1} = F(X^i) \), which can be detailed as:
\[
X_j^{i+1} = F(X_1^i, \ldots, X_n^i) \quad j = 1, \ldots, n
\]

One can also use a chaotic iteration strategy and arbitrarily determine at each step which are the components of the system of equations which will evolve and in what order (as long as no component is forgotten indefinitely).

More precisely [Cous77b, Cous77c] \( gfp(F) \) is the least upper bound of any chaotic iteration sequence \( \{X^i: i \geq 0\} \) where \( X^0 = \langle \bot, \ldots, \bot \rangle \) and
\[
X_j^{i+1} = F(X_1^i, \ldots, X_n^i) \quad \text{if } j \in J_i
\]
\[
X_j^{i+1} = X_j^i \quad \text{if } j \notin J_i
\]
provided that \( \forall i \geq 0, J_i \subseteq [1, n] \) and \( \forall j \in [1, n], \exists k \geq 0: j \in J_{i+k} \).

A dual result holds for \( gfp(F) \).

10-4.3. Characterization of the Set of Descendants of the Entry States of a Discrete Dynamic System as a Least Fixed Point

Given a discrete dynamic system \((S, \tau, v_e, v_m, v_o)\), the set of descendants of the states satisfying a condition \( \beta \in (S \rightarrow B) \) is by definition the set characterized by
\[
\lambda s_2. [\exists s_1 \in S: \beta(s_1) \land \tau^*(s_1, s_2) = post(\tau^*)(\beta)]
\]
using the notation
\[
post \in (((S \times S) \rightarrow B) \rightarrow ((S \rightarrow B) \rightarrow (S \rightarrow B)))
\]
\[
post = \lambda B. [\lambda \beta. [\lambda s_2. [\exists s_1 \in S: \beta(s_1) \land \theta(s_1, s_2)]]]
\]

Example 10-3. Let \( \pi \) be a program defining a total and deterministic system \((S, \tau, v_e, v_m, v_o)\). Assume that \( \phi, \Psi \in (S \rightarrow B) \) specify what it is that \( \pi \) is intended to do: the execution of the program \( \pi \) starting with an entry state satisfying \( \phi \) terminates and the exit state satisfies \( \Psi \) on termination of \( \pi \). A partial correctness proof consists in showing that:
\[
v_m \land post(\tau^*)(v_e \land \phi) \Rightarrow \Psi
\]
In words, every exit state which is a descendant of an entry state satisfying $\phi$ must satisfy $\Psi$. The question of termination is not involved.

We now show that $post(\tau^*)(\beta)$ is a solution to the equation $\alpha = \beta \lor post(\tau)(\alpha)$; more precisely it is the least one for the implication $\Rightarrow$ considered as a partial ordering on $(S \to B)$.

**Theorem 10.4.**

1. $(S \times S) \to B)(\Rightarrow, \lambda(s_1, s_2).false, \lambda(s_1, s_2).true, \lor, \land, \neg)$ and $(S \to B)(\Rightarrow, \lambda.\neg.false, \lambda.\neg.true, \lor, \land, \neg)$ are uniquely complemented complete lattices.

2. $\forall \theta \in ((S \times S) \to B), post(\theta)$ is a strict complete $\lor$-morphism. $\forall \beta \in (S \to B), \lambda \theta. [post(\theta)(\beta)]$ is a strict complete $\lor$-morphism.

3. $\forall \tau \in ((S \times S) \to B), \forall \beta \in (S \to B),$

   $post(\tau^*)(\beta) = \lor_{\alpha \geq 0} post(\tau^*)(\beta) = lfp(\lambda \alpha. \beta \lor post(\tau)(\alpha))$

**Proof.** The diagram of isotone functions shown in Fig. 10-3 is complete and $lfp(\lambda \alpha. \beta \lor post(\tau)(\alpha))$ is a strict complete $\lor$-morphism. Therefore

   $post(lfp(\lambda \alpha. [eq \lor \alpha \circ \tau])(\beta)) = post(\tau^*)(\beta)$

   $= lfp(\lambda \alpha. [post(eq)(\beta) \lor post(\tau)(\alpha)])$

   $= lfp(\lambda \alpha. [\beta \lor post(\tau)(\alpha)])$.

   Also $post(\tau^*)(\beta) = lfp(\lor_{\alpha \geq 0} \tau^*)(\beta) = lfp(\lor_{\alpha \geq 0} post(\tau^*)(\beta))$.  

**Example 10-5.** Floyd [Floy67] and Naur’s [Naur66] method of inductive assertions for proving the partial correctness of $\pi$ with respect to $\phi, \Psi$, consists in guessing an assertion $t$ and showing that $(((v_e \land \phi) \Rightarrow t) \land (post(\tau)(t) \Rightarrow t) \land ((v_e \land t) \Rightarrow \Psi))$.

Using the recursion induction principle, from $(((v_e \land \phi) \Rightarrow t) \land$
(post(\tau)(i) \Rightarrow i) we infer (\text{ifp}(\lambda x. [v_r \land \phi] \lor post(\tau)(x)) \Rightarrow i). It follows from Theorem 10-4, Part 3 that (v_w \land post(\tau^*)(v_r \land \phi)) \Rightarrow (v_w \land i) \Rightarrow \Psi. The method is sound [ClaE77].

Reciprocally, if \pi is partially correct with respect to \phi, \Psi, then this can be proved using the Floyd-Naur method. This completeness result follows from the fact that one can choose i as \text{ifp}(\lambda x. [v_r \land \phi] \lor post(\tau)(x)).

10-4.4. Characterization of the Set of Descendants of the Exit States of a Deterministic Discrete Dynamic System as a Least Fixed Point

In the case of a deterministic discrete dynamic system, the set of ascendants of the states satisfying a condition \beta \in (S \rightarrow B) is characterized by

\lambda s_1. [\exists s_2 \in S: \tau^*(s_1, s_2) \land \beta(s_2)] = \text{pre}(\tau^*)(\beta)

using the notation

\text{pre} \in \text{((S \times S) \rightarrow (S \rightarrow B) \rightarrow ((S \rightarrow B) \rightarrow (S \rightarrow B)))}

\text{pre} = \lambda \theta. [\lambda \beta. [\lambda s_1. [\exists s_2 \in S: \theta(s_1, s_2) \land \beta(s_2)]]]

Example 10-6. Let \pi be a program defining a total and deterministic system (S, \tau, v_r, v_w, v_\Psi) and \phi, \Psi \in (S \rightarrow B) be respectively entry and exit specifications. A total correctness proof consists in showing

\forall \phi \Rightarrow \text{pre}(\tau^*)(v_w \land \Psi)

In words, every entry state satisfying \phi is the ascendant of an exit state satisfying \Psi. This is a proof of termination when \Psi = \lambda s. [\text{true}].

Once the mathematical properties of post have been studied, similar ones can be easily derived for \text{pre} since \text{pre}(\theta)(\beta) = \text{post}(\theta^{-1})(\beta) and \text{post}(\theta)(\beta) = \text{pre}(\theta^{-1})(\beta). This point is illustrated by the proof of the following:

Theorem 10-7.

1. \forall \theta \in (S \times S \rightarrow B), \text{pre}(\theta) is a strict complete \lor-morphism; \forall \beta \in (S \rightarrow B), \lambda \theta. \text{pre}(\theta)(\beta) is a strict complete \lor-morphism.

2. \forall \tau \in (S \times S \rightarrow B), \forall \beta \in (S \rightarrow B),

\text{pre}(\tau^*)(\beta) = \bigvee_{\kappa \in \theta} \text{pre}(\tau^*)(\beta) = \text{ifp}(\lambda x. [\lor \text{pre}(\tau^*)(x)])

Proof. \forall \tau, \tau_1, \ldots \in (S \times S \rightarrow B), (\tau_1 \circ \tau_2)^{-1} = (\tau_2^{-1} \circ \tau_1^{-1}) ; \forall n \in \mathbb{N}, (\tau^*)^{-1} = (\tau^{-1})^* ; (\lor \tau)^{-1} = \bigvee (\tau_i)^{-1} ; (\tau^*)^{-1} = (\tau^{-1})^*. Therefore it follows from Theorem 10-4 that \forall \theta \in (S \times S \rightarrow B), \text{pre}(\theta) =
post(θ⁻¹) is a strict complete \( \lor \)-morphism. \( \forall \beta \in (S \rightarrow S), \lambda \beta. \, \text{pre}(\theta)(\beta) = \lambda \theta. \, \text{post}(\theta⁻¹)(\beta) \) is a strict complete \( \lor \)-morphism. Also
\[
\text{pre}(\tau^*)(\beta) = \text{post}(\tau^*)(\beta) = \bigvee_{\alpha \in \mathcal{A}} \text{post}(\tau^*)(\alpha)(\beta)
\]
\[
= \bigvee_{\alpha \in \mathcal{A}} \text{post}(\tau^*)(\alpha)(\beta) \\
= \text{lfp}(\lambda \alpha. [\beta \lor \text{post}(\tau^*)(\alpha)]) = \text{lfp}(\lambda \alpha. [\beta \lor \text{pre}(\tau)(\alpha)]).
\]

10.4.5. Characterization of the States of a Total Deterministic System Which Do Not Lead to an Error as a Greatest Fixed Point

The entry states which are the origins of correctly terminating or diverging execution paths of a deterministic program \( \pi(S, \tau, \nu, \nu_e, \nu_e) \) are those which do not lead to a run-time error. They are characterized by \( \nu \land \neg \text{pre}(\tau^*)(\nu_e) \).

**Theorem 10.8.** Let \( \tau \in ((S \times S) \rightarrow B) \) be total and deterministic.

\[ \forall \beta \in (S \rightarrow B), \hspace{1em} \neg \text{pre}(\tau^*)(\beta) = \text{gfp}(\lambda \alpha. [\neg \beta \land \text{pre}(\tau)(\alpha)]) \]

**Proof.** \( \neg \text{pre}(\tau^*)(\beta) = \neg \text{lfp}(\lambda \alpha. [\beta \lor \text{pre}(\tau)(\alpha)]) = \neg \text{lfp}(\lambda \alpha. [\beta \land \text{pre}(\tau)(\neg \alpha)]) \) According to Park's Fixed Point Theorem, this is equal to \( \text{gfp}(\lambda \alpha. [\neg \beta \land \neg \text{pre}(\tau)(\neg \alpha)]) \). Let \( \tilde{\tau} \in (S \rightarrow S) \) be such that \( (\forall s_1, s_2 \in S, \tau(s_1, s_2) \leftrightarrow (\tilde{\tau}(s_1) = s_2)) \). We have \( \neg \text{pre}(\tau)(\neg \alpha) = \lambda \gamma_1, [\neg \alpha(\gamma_1)] = \lambda \gamma_1, [\alpha(\gamma_1)] = \text{pre}(\tau)(\alpha) \).}

10.4.6. Analysis of the Behavior of a Total Deterministic Discrete Dynamic System

Given a total deterministic system \( \pi(S, \tau, \nu, \nu_e, \nu_e) \) we have established that the analysis of the behavior of this system can be carried out by solving fixed point equations as follows:

**Theorem 10.9.**

1. The set of descendants of the entry states satisfying an entry condition \( \phi \in (S \rightarrow B) \) is characterized by:
   \[ \text{post}(\tau^*)(\nu \land \phi) = \text{lfp}(\lambda \alpha. [(\nu \land \phi) \lor \text{post}(\tau)(\alpha)]) \]
2. The set of descendants of the exit states satisfying an exit condition \( \Psi \in (S \rightarrow B) \) is characterized by:
   \[ \text{pre}(\tau^*)(\nu_e \land \Psi) = \text{lfp}(\lambda \alpha. [(\nu_e \land \Psi) \lor \text{pre}(\tau)(\alpha)]) \]
3. The set of states leading to an error is characterized by:
   \[ \text{pre}(\tau^*)(\nu_e \land \text{pre}(\tau)(\alpha)) \]
4. The set of states which do not lead to an error (i.e., cause the system either to properly terminate or to diverge) is characterized by:

\[-\text{pre}(\tau^*)(\nu_e) = \text{gfp}(\lambda \alpha. [\neg \nu_c \land \text{pre}(\tau)(\alpha)])\]

5. The set of states which cause the system to diverge is characterized by:

\[-\text{pre}(\tau^*)(\nu_m \lor \nu_c) = \text{gfp}(\lambda \alpha. [\neg \nu_m \land \neg \nu_c \land \text{pre}(\tau)(\alpha)])\]

Example 10-10. The proof that a program \(\pi(S, \tau, \nu, \nu_m, \nu_c)\) does not terminate for the entry states satisfying a condition \(\delta \in (S \rightarrow B)\) consists in proving that \(\nu_e \land \delta \Rightarrow -\text{pre}(\tau^*)(\nu_m \lor \nu_c)\). It follows from Theorem 10-9(5) and the Dual Recursion Induction Principle that this can be done by guessing an assertion \(i \in (S \rightarrow B)\) and proving that \(\((\nu_e \land \delta) \Rightarrow i) \land (i \Rightarrow \neg \nu_m \land \neg \nu_c \land \text{pre}(\tau)(i))\).

10-4.7. Relationships Between \textit{pre} and \textit{post}

\textbf{Theorem 10-11.} Let \(\theta \in ((S \times S) \rightarrow B)\). \(\forall \beta, \gamma \in (S \rightarrow B)\),

1. \(\text{pre}(\theta)(\beta) = \text{post}(\theta^{-1})(\beta), \text{post}(\theta)(\beta) = \text{pre}(\theta^{-1})(\beta)\)
2. If \(\theta\) is deterministic, then:

\(\text{post}(\theta)(\text{pre}(\theta)(\beta)) = (\beta \land \text{post}(\theta)(\text{true})) \Rightarrow \beta\)
3. If \(\theta\) is total, then:

\(\beta \Rightarrow \text{pre}(\theta)(\text{post}(\theta)(\beta))\)
4. If \(\theta\) is total and deterministic, then:

\((\beta \Rightarrow \text{pre}(\theta)(\gamma)) \text{ iff } (\text{post}(\theta)(\beta) \Rightarrow \gamma)\)

\(\text{post}(\theta)(\beta) = \bigwedge_{\gamma \in (S \rightarrow B)} (\beta \Rightarrow \text{pre}(\theta)(\gamma))\)

\(\text{pre}(\theta)(\beta) = \bigvee_{\gamma \in (S \rightarrow B)} (\beta \Rightarrow \text{post}(\theta)(\gamma)) = \beta\)

\textbf{Proof.}

1. \(\text{pre}(\theta)(\beta) = \lambda s_1. [\exists s_2 : \theta(s_1, s_2) \land \beta(s_2)] = \lambda s_1. [\exists s_2 : \beta(s_2) \land \theta^{-1}(s_2, s_1)] = \text{post}(\theta^{-1})(\beta). \text{ post}(\theta)(\beta) = \text{post}((\theta^{-1})^{-1})(\beta) = \text{pre}(\theta^{-1})(\beta)\).
2. If \(\theta\) is deterministic, then there exists \(\hat{\theta} \in (S \rightarrow S)\) such that \(\theta(s_1, s_2) \Rightarrow s_2 = \hat{\theta}(s_1)\). Therefore

\(\text{post}(\theta)(\text{pre}(\theta)(\beta)) = \lambda s_2. [\exists s_1 : (\exists s_2 : s_2 = \hat{\theta}(s_1) \land \beta(s_2)) \land s_2 = \hat{\theta}(s_1)] = \lambda s_2. [\exists s_1 : \beta(\hat{\theta}(s_1)) \land s_2 = \hat{\theta}(s_1)] = \text{post}(\theta)(\text{true}) \land \beta\).
3. If \(\theta\) is total, then \(\forall s_3 \in S, \beta(s_3) \Rightarrow (\beta(s_2) \land (\exists s_2 : \theta(s_3, s_2) \Rightarrow (\exists s_2 : \theta(s_3, s_2) \land (\exists s_1 : \theta(s_1, s_2) \land \beta(s_1)))) = \text{pre}(\theta)(\text{post}(\theta)(\beta))(s_3)\).
4. If \((β \Rightarrow \text{pre}(θ)(γ))\), then by isotony \(\text{post}(θ)(β) \Rightarrow \text{post}(θ)(γ)\)
   \((\text{pre}(θ)(γ)) \Rightarrow γ\). If \((\text{post}(θ)(β) \Rightarrow γ)\), then by isotony \(β \Rightarrow \text{pre}(θ)(\text{post}(θ)(β)) \Rightarrow \text{pre}(θ)(γ)\).
   \(\text{post}(θ)(β) = \bigwedge[y: \text{post}(θ)(β) \Rightarrow γ] = \bigwedge[y: β \Rightarrow \text{pre}(θ)(γ)]. \text{pre}(θ)(β) = \bigwedge[γ: γ \Rightarrow \text{pre}(θ)(β)]\)
   \(= \bigwedge[y: \text{post}(θ)(γ) \Rightarrow β]\).

Example 10.12. According to Theorem 10.11.4, Floyd-Naur's method for proving the partial correctness of \(π\) with respect to \(φ, \Psi\) which consists in guessing an assertion \(i\) and showing that \(((\nu \land φ) \Rightarrow i) \land \text{post}(π)(i) \Rightarrow i\)
\(\land ((\nu \land i) \Rightarrow \Psi)\) is equivalent to Hoare's method [Hoar69], which consists of guessing an assertion \(i\) and showing that \(((\nu \land φ) \land (i \Rightarrow \text{pre}(π)(i)) \land
\((\nu \land i) \Rightarrow \Psi)\).

We have seen that the analysis of a system consists of solving “forward”
fixpoint equations of the form \(α = β \uplus \text{post}(π)(α)\) or “backward” fixpoint equations of the form \(α = β \land \text{pre}(π)(α)\) (where \(β \in (S \to B)\) and \(\uplus\) is either \(\lor\) or \(\land\)). In fact whenever a forward equation is needed, a backward
equation can be used instead, and vice versa.

Theorem 10.13.

\[ ∀θ \in (S \times S) \to B, ∀β \in (S \to B), \]
\[ \text{post}(θ)(β) = \lambda s. [\exists s_1 \in S: β(s_1) \land \text{pre}(θ)(λs. [s = s_1])(s_1)] \]
\[ \text{pre}(θ)(β) = \lambda s. [\exists s_2 \in S: \text{post}(θ)(λs. [s = s_2])(s_2)] \land β(s_2)] \]

Proof. \(\text{post}(θ)(β) = \lambda s. [∀ s_1 \in S: β(s_1) \land θ(s_1, s)] = \lambda s. [∀ s_1 \in S: β(s_1) \land (∃ s \in S: (s_1) \land θ(s_1, s))] = \lambda s. [∃ s_1 \in S: β(s_1) \land \text{pre}(θ)(λs. [s = s_1])(s_1)].\)

Example 10.14. A total correctness proof of a program \(π\) with respect to \(φ, \Psi\) consists in showing that \(((\nu \land φ) \Rightarrow \text{pre}(π)(π)\land \Psi)\), that is to say
\(((\nu \land φ) \Rightarrow \lambda s. [\nu \land Ψ] \lor \text{pre}(π)(π)])\). Equivalently, using \(\text{post}\), one can show that:
\(∀ s \in S, (ν(π)(s) \land φ(s)) \Rightarrow (\exists s_2 \in S: ν(s_2) \land Ψ(s_2) \land \text{post}(π)(π)(s_2))\) More generally we have:
\(\text{pre}(π)(π) = \lambda s. [∃ s_2 \in S: β(s_2) \land \text{pre}(π)(π)(s_2)] \land β(s_2) \land \text{post}(π)(π)(s_2)] \)

10.4. Partitioned Dynamic Discrete Systems

A dynamic discrete system \((S, τ, ν_1,ν_n, ν_0)\) is said to be partitioned if there exist \(n ≥ 1, U_1, \ldots, U_n, \tau_1, \ldots, \tau_n\) such that \(∀ i \in [1, n], \tau_i\) is a partial one-to-one map from \(S\) onto \(U_i\) and \([τ_i^{-1}(U_i)]: i \in [1, n]\) is a partition of \(S\),
therefore \(S = \bigcup_{i=0}^{n} τ_i^{-1}(U_i)\) and every \(s \in S\) is an element of exactly one \(τ_i^{-1}(U_i)\).
When studying the behavior of a partitioned system, the equations \( \alpha = \beta \equiv post(\tau)(\alpha) \) or \( \alpha = \beta \equiv pre(\tau)(\alpha) \) can be replaced by systems of equations defined as follows. Let us define: \( \forall i \in [1, n], \sigma_i \in ((S \rightarrow B) \rightarrow (U_i \rightarrow B)), \sigma_i = \lambda \beta.[\beta \circ \iota_i^{-1}], \sigma_i^{-1} = \lambda \beta.[\lambda \epsilon, \epsilon \in \iota_i^{-1}(U_i) \land \beta(\iota_i(\epsilon))], \sigma \in ((S \rightarrow B) \rightarrow \prod_{i=1}^{n} (U_i \rightarrow B)), \sigma = \lambda \beta.(\prod_{i=1}^{n} \sigma_i(\beta)) = \lambda \beta.\langle \sigma_1(\beta), \ldots, \sigma_n(\beta) \rangle. \sigma \) is a strict isomorphism from \((S \rightarrow B)\) onto \(\prod_{i=1}^{n} (U_i \rightarrow B)\). Its inverse is \(\sigma^{-1} = \lambda \langle \beta_1, \ldots, \beta_n \rangle.\left[ \prod_{i=1}^{n} \iota_i^{-1}(\beta_i) \right]\).

For any isotone operator \(f\) on \((S \rightarrow B)\), the diagram in Fig. 10.4 commutes, so that the sets of pre-fixed points, fixed points and post-fixed points of \(f\) coincide (up to the isomorphism \(\sigma\)) with the pre-solutions, solutions and post-solutions to the direct decomposition of \(\alpha = f(\alpha)\) on \(\prod_{i=1}^{n} (U_i \rightarrow B)\) which is the system of equations:

\[
\begin{align*}
\alpha_1 &= \sigma_1 \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n) \\
\vdots \\
\alpha_n &= \sigma_n \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n)
\end{align*}
\]

In particular when \(f = \lambda \alpha.[\beta \equiv post(\tau)(\alpha)]\) or \(f = \lambda \alpha.[\beta \equiv pre(\tau)(\alpha)]\), we have the following:

**Theorem 10.15.** \(\forall i \in [1, n], \sigma_i \circ \lambda \alpha.[\beta \equiv post(\tau)(\alpha)] \circ \sigma^{-1}\) is equal to

\[
\lambda \langle \alpha_1, \ldots, \alpha_n \rangle.\left[ \sigma_i(\beta) \equiv \bigvee_{j \in \text{pred}_i(\alpha)} post(\tau)(\alpha_j) \right]
\]

whereas \(\sigma_i \circ \lambda \alpha.[\beta \equiv pre(\tau)(\alpha)] \circ \sigma^{-1}\) is equal to

\[
\lambda \langle \alpha_1, \ldots, \alpha_n \rangle.\left[ \sigma_i(\beta) \equiv \bigvee_{j \in \text{succ}_i(\alpha)} pre(\tau)(\alpha_j) \right]
\]
where
\[ \tau_{ij} \in (\mathcal{M} \times \mathcal{M}) \rightarrow \mathcal{M}, \tau_{ij} = \lambda (s_1, s_2).[\tau_1(s_1), \tau_2^{-1}(s_2)] \]
\[ \text{pred}_i = \lambda i.\{j \in [1, n]: ( \exists s_1 \in U_j, \exists s_2 \in U_i: \tau_j(s_1, s_2)) \} \]
\[ \text{succ}_i = \lambda i.\{j \in [1, n]: ( \exists s_1 \in U_j, \exists s_2 \in U_i: \tau_i(s_1, s_2)) \} \]

**Proof.**
\[ \sigma_i(\beta \% post(\tau)(\sigma_j^{-1}(a_1, \ldots, a_n))) = \sigma_i(\beta) \% \sigma_i\left(\bigvee_{j=1}^{n} \sigma_j^{-1}(a_j)\right) \]
\[ = \sigma_i(\beta) \% \bigvee_{j=1}^{n} (post(\tau)(\sigma_j^{-1}(a_j)) \circ \tau_j^{-1}) \].

Moreover
\[ post(\tau)(\sigma_j^{-1}(a_j)) \circ \tau_j^{-1} = \lambda s_2. \{ s_1 \in S: \sigma_j^{-1}(a_j)(s_1) \land \tau(s_1, \tau_j^{-1}(s_2)) \} \]
\[ = \lambda s_2. \{ s_1 \in U_j: a_j(s_1) \land \tau(s_1, \tau_j^{-1}(s_2)) \} \]
\[ = \lambda s_2. \{ s_1 \in U_j: a_j(s_1) \land \tau_j(s_1, s_2) \} \]
\[ = post(\tau_j)(a_j) \].

Therefore
\[ \bigvee_{j=1}^{n} (post(\tau)(\sigma_j^{-1}(a_j)) \circ \tau_j^{-1}) = \bigvee_{j \notin \text{pred}_i(0)} post(\tau_j)(a_j) \] since
\[ (j \notin \text{pred}_i(0)) \text{ implies } \forall s_1, s_2, \tau_j(s_1, s_2). Also \ \text{pre}(\tau) = post(\tau^{-1}), (\tau_j)^{-1} = \tau_j^{-1} \text{ and } \text{succ}_i = \text{pred}_i \].

**SEC. 10-5 / SEMANTIC ANALYSIS OF PROGRAMS**

The fixed point approach to the analysis of the behavior of total deterministic discrete dynamic systems is now applied to the case of programs as defined in Section 10-3.

A program \( \langle G, U, L \rangle \) where \( G = \langle V, \epsilon, \omega, E \rangle \) and \( V = [1, n] \setminus \{ \varepsilon \} \) defines a partitioned discrete dynamic system \( \langle \tau, S, v_s, v_n, v_c \rangle \) where \( S = ([1, n] \times U), \forall i \in [1, n], U_i = U, i, \tau_i = \lambda \langle c, m \rangle. m, \tau_i^{-1} = \lambda m. \langle i, m \rangle \). Hence two states \( \langle c_1, m_1 \rangle \) and \( \langle c_2, m_2 \rangle \) are in the same block of the partition iff \( c_1 = c_2 \), that is, iff both states correspond to the same program point or are both erroneous.

**10-5.1. System of Forward Semantic Equations Associated with a Program and an Entry Specification**

The system of forward semantic equations \( P = F_i(\phi)(P) \) associated with a program \( \pi \) and an entry specification \( \phi \in (U \rightarrow B) \) is the direct decomposition of \( \alpha = (\nu_\epsilon \land \sigma_i^{-1}(\phi)) \lor \text{post}(\tau)(\alpha) \) on \( (U \rightarrow B)^\pi \); that is,
\[ P_i = \sigma_i(\nu_\epsilon \land \sigma_i^{-1}(\phi)) \lor \left( \bigvee_{j \notin \text{pred}_i(0)} \text{post}(\tau_j)(P_j) \right) \quad i = 1, \ldots, n \]
From the abstract syntax and operational semantics of programs we derive a set of construction rules for obtaining this system of equations from the program text:

1. If $i$ is the program entry point, $i = \epsilon$ and $\text{pred}_i(\epsilon) = \emptyset$; therefore $P_\epsilon = \sigma_i(\epsilon \land \sigma^{-1}_i(\emptyset)) = \sigma_i(\lambda(c, m), ((c = \epsilon) \land \emptyset(m))) = \emptyset$. Otherwise $i \neq \epsilon$, in which case $\sigma_i(\epsilon \land \sigma^{-1}_i(\emptyset)) = \lambda m. \text{false}$ and

$$P_i = \bigvee_{j \in \text{pred}_i(\epsilon)} \text{post}(\tau_j)(P_j)$$

$$= \bigvee_{j \in \text{pred}_i(\epsilon)} \lambda m_2. [\exists m_1 \in U: P_j(m_1) \land (\tau(\langle j, m_1 \rangle) = \langle i, m_2 \rangle)]$$

When $i \neq \epsilon$ and $i \neq \xi$, notice that $\text{pred}_i(i)$ is contained in the set of origins of the edges entering $i$, that is, the set $\text{pred}_i(i)$ of predecessors of the vertex $i$ in the program graph $G$ of $\pi$. The expression $\lambda m_2. [\exists m_1 \in U: P_j(m_1) \land (\tau(\langle j, m_1 \rangle) = \langle i, m_2 \rangle)]$ depends on the instruction $L(\langle j, i \rangle)$ labeling the edge $\langle j, i \rangle$.

2. If $\langle j, i \rangle$ is labeled with an assignment $\hat{v} = f(\hat{v})$, then

$$\lambda m_2. [\exists m_1 \in U: P_j(m_1) \land (\tau(\langle j, m_1 \rangle) = \langle i, m_2 \rangle)]$$

$$= \lambda m_2. [\exists m_1 \in U: P_j(m_1) \land m_1 \in \text{dom}(f) \land m_2 = f(m_1)]$$

3. If $\langle j, i \rangle$ is labeled with a test $p$, then

$$\lambda m_2. [\exists m_1 \in U: P_j(m_1) \land (\tau(\langle j, m_1 \rangle) = \langle i, m_2 \rangle)]$$

$$= \lambda m_2. [\exists m_1 \in U: P_j(m_1) \land (m_1 \in \text{dom}(p) \land p(m_1)$$

$$\land (m_1 = m_2)]$$

$$= \lambda m_2. [P_j(m_2) \land (m_2 \in \text{dom}(p) \land p(m_2)]$$

4. If $i = \xi$, then

$$P_\xi = \bigvee_{j \in \text{pred}_\xi(\epsilon)} \lambda m_2. [\exists m_1 \in U: P_j(m_1) \land (\tau(\langle j, m_1 \rangle) = \langle \xi, m_2 \rangle)]$$

$$= P_\xi \lor \bigvee_{j \in \text{pred}_\xi(\epsilon)} \lambda m_2. [P_j(m_2) \land (m_2 \notin \text{dom}(\text{expr}(j)))]$$

where $\text{at}(\pi)$ is the set of program points $j$ preceding an assignment $\hat{v} = f(\hat{v})$ or a test $p(\hat{v})$ and $\text{expr}(j)$ is the corresponding $f$ or $p$.

The above analysis can be summarized by the following.

Definition 10-16. The system of forward semantic equations $P = F_\pi(\phi)(P)$ associated with a program $\pi$ and an entry specification $\phi \in (U \rightarrow B)$ is

$$P_\pi = \phi$$

$$P_i = \bigvee_{j \in \text{pred}_i(\epsilon)} \text{post}(L(\langle j, i \rangle))(P_j) \quad i \in ([1, n] \setminus \{\epsilon, \xi\})$$

$$P_\xi = \bigvee_{j \in \text{pred}_\xi(\epsilon)} \lambda m. [P_j(m) \land m \notin \text{dom}(\text{expr}(j)) \lor P_\xi]$$

$$\lor \text{post}(L(\langle j, \xi \rangle))(P_j) \quad i \in ([1, n] \setminus \{\epsilon, \xi\})$$
where \( \forall f \in I(U), \text{post}(f) = \lambda P.[\lambda m. [\exists m' \in U: P(m') \land m = f(m')]] \); \( \forall p \in I(U), \text{post}(p) = \lambda P.[\lambda m. [P(m) \land \text{dom}(p) \land m(m)]] \); \( \text{at}(\pi) \) is the set of program points \( j \) preceding an assignment \( v = f(\overline{v}) \) or a test \( p(\overline{v}) \) and \( \text{expr}(j) \) is the corresponding \( f \) or \( p \).

**Theorem 10-17.** The system of forward semantic equations \( P = F_{\phi}(\phi)(\pi) \) associated with a program \( \pi \) and an entry specification \( \phi \in (U \rightarrow B) \) is the direct decomposition of \( \alpha = (v, \wedge \sigma^{-1}(\phi)) \) \( \forall \text{post}(\pi)(\alpha) \) on \((U \rightarrow B)^{\ast}\).

### 10-5.2. System of Backward Semantic Equations Associated with a Program and an Exit Specification

As above the abstract syntax and operational semantics of programs can be used in order to derive sets of construction rules for associating with any program \( \pi \) the systems of equations which are the direct decompositions of backward equations of type \( \alpha = \beta \times \text{pre}(\pi)(\alpha) \) on \((U \rightarrow B)^{\ast}\); that is,

\[
P_{\pi} = \sigma_{\pi}(\beta) \times \bigvee_{j \in \text{succ}\_\pi(0)} \lambda m. [\exists m_{2} \in U: (\psi(m_{1}, i) = \langle m_{2}, j \rangle) \wedge P_{\pi}(m_{2})] \quad i = 1, \ldots, n
\]

The result of this study can be summarized by the following.

**Definition 10-18.** The system of backward semantic equations \( P = B_{\pi}(\pi)(\pi) \) associated with a program \( \pi \) and an exit specification \( \psi \in (U \rightarrow B) \) is

\[
P_{\pi} = \bigvee_{i \in \text{succ}\_\pi(0)} \text{pre}(\psi)(i)(\pi) \quad i \in ([1, n] - \{\xi, \omega\})
\]

\[
P_{\psi} = \psi
\]

where \( \forall f \in I(U), \text{pre}(f) = \lambda P.[\lambda m. [m \in \text{dom}(f) \land P(f(m))]] \); \( \forall p \in I(U), \text{pre}(p) = \lambda P.[\lambda m. [m \in \text{dom}(f) \land p(m) \land P(m)]] \); and \( \text{succ}_{\pi}(i) \) is the set of successors of the vertex \( i \) in the program graph of \( \pi \).

**Theorem 10-19.**

1. The direct decomposition \( P = B(P) \) of \( \alpha = (v, \wedge \sigma_{\pi}(\psi)) \) \( \forall \text{pre}(\pi)(\alpha) \) on \((U \rightarrow B)^{\ast}\) is

\[
P_{\pi} = B_{\pi}(\psi)(\pi) \vee \text{error}(i) \quad \text{for } i \in ([1, n] - \{\xi, \omega\})
\]

\[
P_{\psi} = \psi \vee P_{0}
\]

\[
P_{z} = P_{z}
\]

where \( \text{error}(i) = \lambda m \in U. [P_{\pi}(m) \land i \in \text{at}(\pi) \land m \notin \text{dom}(\text{expr}(i))] \); \( \forall i \in ([1, n] - \{\xi, \omega\}), \text{lfp}(B)_{i} = \text{lfp}(B_{\pi})_{i} \); and \( \text{lfp}(B)_{z} = \lambda m. [\text{false}] \).
2. The direct decomposition \( P = B(P) \) of \( \alpha = \neg v_\xi \land \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is

\[
P_i = B_\alpha(\lambda m.[true])(P) \quad \text{for} \ i \in ([1, n] - \{\omega, \xi\})
\]

\[
P_\omega = P_\alpha
\]

\[
P_\xi = \lambda m.[\text{false}]
\]

\( \forall i \in ([1, n] - \{\xi\}), gfp(B)_i = gfp(B_\alpha(\lambda m.[true]))_i \), and \( gfp(B) = \lambda m.[\text{false}] \).

3. The direct decomposition \( P = B(P) \) of \( \alpha = v \lor \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is

\[
P_i = B_\alpha(\lambda m.[false])(P) \lor \text{error}(i) \quad \text{for} \ i \in ([1, n] - \{\omega, \xi\})
\]

\[
P_\omega = P_\alpha
\]

\[
P_\xi = \lambda m.[true]
\]

The least solution to the above system of equations is equal to the least solution to

\[
P_i = B_\alpha(\lambda m.[false])(Q) \lor \lambda m. [m \notin \text{dom}(\text{expr}(i))] \quad \text{for} \ i \in ([1, n] - \{\omega, \xi\})
\]

\[
P_\omega = \lambda m.[\text{false}]
\]

\[
P_\xi = \lambda m.[\text{true}]
\]

where \( Q \) stands for \( P_i \) when \( i \in ([1, n] - \{\omega, \xi\}) \), \( Q_\omega \) stands for \( \lambda m.[\text{false}] \) and \( Q_\xi \) stands for \( \lambda m.[\text{true}] \).

4. The direct decomposition of \( \alpha = \neg w_m \land \neg w_\xi \land \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is

\[
P_i = B_\alpha(\lambda m.[false])(P) \quad \text{for} \ i \in ([1, n] - \{\xi\})
\]

\[
P_\omega = \lambda m.[\text{false}]
\]

5. The direct decomposition of \( \alpha = \lambda s. [s = \bar{s}] \lor \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is

\[
P_i = \lambda m. [i, m'] = \bar{s} \lor B_\alpha(\lambda m.[false])(P) \lor \text{error}(i) \quad \text{for} \ i \in ([1, n] - \{\omega, \xi\})
\]

\[
P_\omega = \lambda m. [\sigma, m'] = \bar{s} \lor P_\omega
\]

\[
P_\xi = \lambda m. [\xi, m'] = \bar{s} \lor P_\xi
\]

10-5.3. Analysis of the Behavior of a Program

In order to illustrate the application of Theorem 10-9 to the analysis of the behavior of a program, we choose the introductory example program \( \pi \):

\[
\]
while \( x \geq 1000 \) do
\[ x := x + y; \]
end;

It is assumed that the domain of values of the variables \( x \) and \( y \) is \( I = \{ n \in \mathbb{Z} : -b - 1 \leq n \leq b \} \) where \( b \) is the greatest and \(-b - 1\) the least machine-representable integer.

10.5.3.1. Forward semantic analysis

The system \( P = F_e(\phi)(P) \) (where \( F_e(\phi) \in ((P \to B)^{2} \to (I \to B)^{2}) \)) of forward semantic equations associated with the above program \( \pi \) and an entry specification \( \phi \in (I \to B) \) is the following:
\[
P_1 = \phi
\]
\[
P_2 = \lambda \langle x, y \rangle, [(P_1 \lor P_3)(x, y) \land (x \in I) \land (x \geq 1000)]
\]
\[
P_3 = \lambda \langle x, y \rangle, [(\exists x' \in I : P_2(x', y) \land ((x' + y) \in I) \land (x = x' + y)]
\]
\[
P_4 = \lambda \langle x, y \rangle, [(P_1 \lor P_3)(x, y) \land (x \in I) \land (x < 1000)]
\]
\[
P_e = \lambda \langle x, y \rangle, {[((P_1 \lor P_3)(x, y) \land (x \notin I)) \lor (P_3(x, y) \land ((x + y) \notin I))]
\]

The set of entry states which are descendants of the exit states (i.e., cause the program to terminate properly) is characterized by:
\[
\sigma_e(v_e, \land \text{pre}(\tau^*)(v_{e}))
\]
\[
= \sigma_e(\lambda \delta, [\exists s_2 \in S: v_e(s_2) \land \text{post}(\tau^*)(\lambda s. [y = \delta])(s_2)])
\]
\[
= \lambda \bar{m}. [\exists s_2 \in S: v_e(s_2) \land \text{post}(\tau^*)(\lambda s. [y = \bar{m}])(s_2)])
\]
\[
= \lambda \bar{m}. [\exists s_2 \in S: v_e(s_2) \land \text{lpf}(\lambda a. [v_e \land \sigma_e^{-1}(\lambda m. [m = \bar{m}])])
\]
\[\lor \text{post}(\tau(\alpha))(s_2)] \quad \text{(Theorem 10-13)}
\]
\[
= \lambda \bar{m}. [\exists s_2 \in S: v_e(s_2) \land \sigma_e^{-1}(\text{lpf}(F_e(\lambda m. [m = \bar{m}])))(s_2)]
\]
\[
= \lambda \bar{m}. [\exists s_2 \in S: v_e(s_2) \land \left(\bigvee_{i=1}^{n} \text{lpf}(F_e(\lambda m. [m = \bar{m}]))(t_i(s_2))\right)]
\]
\[
= \lambda \bar{m}. [\exists m_2 \in U_e: v_e(g^{-1}(m_2)) \land \text{lpf}(F_e(\lambda m. [m = \bar{m}]))(m_2)]
\]
\[
= \lambda \bar{m}. [\exists m_2 \in U_e: \text{lpf}(F_e(\lambda m. [m = \bar{m}]))(m_2)]
\]
The least fixed point \( P^\omega \) of \( F_\nu(\lambda\langle x, y \rangle.\{ (x = \bar{x}) \land (y = \bar{y}) \}) \) is computed iteratively using a chaotic iteration sequence as follows:

\[
P_1 = \lambda\langle x, y \rangle.\{ \text{false} \} \\
P_i = \lambda\langle x, y \rangle.\{ (x = \bar{x}) \land (y = \bar{y}) \} \quad \text{where} \quad \langle \bar{x}, \bar{y} \rangle \in P^i \\
P_i = \lambda\langle x, y \rangle.\{ (x = \bar{x}) \land (y = \bar{y}) \} \land (x \in I) \land (x \geq 1000) \\
= \lambda\langle x, y \rangle.\{ (x = \bar{x}) \land (y = \bar{y}) \} \land (x \in I) \land (x \geq 1000) \\
\]

Assume as induction hypothesis that

\[
P_k = \lambda\langle x, y \rangle.\{ \exists j \in [0, k-1] : \bigwedge_{i=0}^{j} ((\bar{x} + ij) \in I \land 1000 \leq (\bar{x} + ij)) \\
\land (x = \bar{x} + jy) \land (y = \bar{y}) \}
\]

then

\[
P_{k+1} = \lambda\langle x, y \rangle.\{ \exists j \in [1, k] : \bigwedge_{i=0}^{j-1} ((\bar{x} + ij) \in I \land 1000 \leq (\bar{x} + ij)) \\
\land ((\bar{x} + jy) \in I) \land (x = \bar{x} + jy) \land (y = \bar{y}) \}
\]

proving by induction on \( k \) that \( P^\omega \) is of the form assumed in the induction hypothesis. Then passing to the limit,

\[
P^\omega = \bigvee_{k=0}^{\infty} P_k \\
= \lambda\langle x, y \rangle.\{ \exists j \geq 0 : \bigwedge_{i=0}^{j} ((\bar{x} + ij) \in I \land 1000 \leq (\bar{x} + ij)) \\
\land (x = \bar{x} + jy) \land (y = \bar{y}) \}
\]

\[
= \lambda\langle x, y \rangle.\{ \exists j \geq 0 : (1000 \leq \min(\bar{x}, x)) \land (\max(\bar{x}, x) \leq b) \\
\land (x = \bar{x} + jy) \land (y = \bar{y}) \}
\]
(It is worth noting that the use of the symbolic entry condition $\lambda(x, y).[(x = \bar{x}) \land (y = \bar{y})]$ and of the above iteration strategy corresponds to a symbolic execution of the program loop [Hant76] with the difference that all possible execution paths are considered simultaneously and the induction step as well as the passage to the limit deal with infinite paths.) The remaining components of $\text{fp}(F_a(\lambda(x, y).[(x = \bar{x}) \land (y = \bar{y})]))$ are:

$P_T = \lambda(x, y).[(x = \bar{x}) \land (y = \bar{y})]$

$P_\bar{y} = \lambda(x, y).[(\exists x' \in I: P_\bar{y}(x', y) \land ((x' + y) \in I) \land (x = x' + y))]
\quad \land (x = \bar{x} + j\bar{y}) \land (y = \bar{y})$

$P_\bar{x} = \lambda(x, y).[(P_T \lor P_\bar{y})(x, y) \land (x \in I) \land (x < 1000)]$

$P_\bar{x} = \lambda(x, y).[(\exists x' \in I: P_\bar{x}(x', y) \land ((x' + y) \in I) \land (x = x' + y))
\quad \lor ((\bar{x} < 1000) \land (\bar{y} < 0)
\quad \land (x = \bar{x} + ((\bar{x} - 1000) \text{ div } \bar{y}) + 1)\bar{y}) \land (y = \bar{y})]]$

$P_\bar{y} = \lambda(x, y).[(P_T \lor P_\bar{x})(x, y) \land (x \notin I) \lor (P_\bar{y}(x, y) \land ((x + y) \notin I))]
\quad \lor P_\bar{y}$

$P_\bar{y} = \lambda(x, y).[((\exists x' \in I: P_\bar{y}(x', y) \land (x \notin I)) \lor (P_\bar{y}(x, y) \land ((x + y) \notin I))]
\quad \lor P_\bar{y}$

The set of entry states which cause the program to terminate properly is characterized by

$\lambda(\bar{x}, \bar{y}).[(\exists x_3, y_3 \in I: P_\bar{y}(x_3, y_3)) = \lambda(x, y).[((x < 1000) \lor (\bar{y} < 0)]$

The set of entry states leading to a run-time error is characterized by

$\sigma_v(\nu_v \land \text{pre}(\tau^e)(\nu_v)) = \lambda(m).[(\exists m_2 \in U_o: \text{fp}(F_a(\lambda m. [m = \bar{m}])))(m_2)]$

that is,

$\lambda(\bar{x}, \bar{y}).[(\exists x_3, y_3 \in I: P_\bar{y}(x_3, y_3)) = \lambda(\bar{x}, \bar{y}).[((\exists m_2 \in U_o: \text{fp}(F_a(\lambda m. [m = \bar{m}])))(m_2)]$

The set of entry states which cause the program to diverge is characterized by

$\sigma_v(\nu_v \land \neg\text{pre}(\tau^e)(\nu_v))$

$= \lambda(\bar{m}.\neg[\sum_{i=1}^n (\exists m_2 \in U_1: (\nu_v \lor \nu_v)(\langle i, m_2 \rangle)
\quad \land \text{fp}(F_a(\lambda m. [m = \bar{m}])))(m_2))$

$= \lambda(\bar{m}.\neg(\exists m_2 \in U_o: \text{fp}(F_a(\lambda m. [m = \bar{m}])))(m_2))$

$\land \neg(\exists m_2 \in U_1: \text{fp}(F_a(\lambda m. [m = \bar{m}])))(m_2))$
that is,
\[ \lambda<\bar{x}, \bar{y}>. [\neg(\exists \bar{x}_2, \bar{y}_2 \in I: P^\phi(x_2, y_2)) \land \neg(\exists \bar{x}_2, \bar{y}_2 \in I: P^\phi(x_2, y_2))] \]
\[ = \lambda<\bar{x}, \bar{y}>. [\bar{x} \geq 1000 \land (y = 0)] \]

The set of descendants of the entry states satisfying the entry condition \( \phi \in (P \to B) \) is characterized by \( post(\tau^*(v_\sigma \land \sigma^{-1}(\phi))) \); that is, (Theorem 10-9 and Theorem 10-17) up to the isomorphism \( \sigma \) by \( Q^\sigma = lfp(F_\sigma(\phi)) \):

\[ Q^\phi = \phi \]
\[ Q^1 = \lambda<\bar{x}, \bar{y}>. [\exists j \geq 0: \phi(x - jy, y) \land (1000 \leq \text{min}(x - jy, x)) \land (\text{max}(x - jy, x) \leq b)] \]
\[ Q^2 = \lambda<\bar{x}, \bar{y}>. [\exists j \geq 1: \phi(x - jy, y) \land (1000 \leq \text{min}(x - jy, x - y)) \land (\text{max}(x - jy, x) \leq b)] \]
\[ Q^3 = \lambda<\bar{x}, \bar{y}>. [\phi(x, y) \land (x < 1000)] \lor ((y < 0) \land (\exists j \geq 1: \phi(x - jy, y) \land (x - jy \leq b) \land (x < 1000 \leq x - y))] \]
\[ Q^4 = \lambda<\bar{x}, \bar{y}>. [(y > 0) \land (\exists j \geq 0: \phi(x - jy, y) \land (1000 \leq x - jy < x \leq b < x + y))] \]

Equivalently \( Q^\phi \) can be obtained from \( P^\phi \) as follows:

\[ \sigma_\phi(post(\tau^*(v_\sigma \land \sigma^{-1}(\phi)))) \]
\[ = \sigma_\phi(\lambda\bar{s}_2. [\exists \bar{z}: v_\phi(\bar{z}) \land \sigma^{-1}(\phi)(\bar{z}) \land post(\tau^*)(\lambda\bar{s}. [\bar{s} = \bar{z}](\bar{s}_2))]) \]
\[ = \lambda\bar{m}_2. [\exists \bar{m}: \phi(\bar{m}) \land post(\tau^*)(\lambda\bar{s}. [\bar{s} = \langle e, \bar{m} \rangle](\bar{s}_2))] \]
\[ = \lambda\bar{m}_2. [\exists \bar{m}: \phi(\bar{m}) \land \sigma^{-1}(lfp(F_\phi(\lambda m. [m = \bar{m}])))(\bar{s}_2)] \]
\[ = \lambda\bar{m}_2. [\exists \bar{m}: \phi(\bar{m}) \land lfp(F_\phi(\lambda m. [m = \bar{m}]))(\bar{s}_2)] \]

Therefore at each program point \( i \) the set of descendants of the entry states satisfying the entry condition \( \phi \in (P \to B) \) is characterized by

\[ Q^\phi = \lambda<\bar{x}, \bar{y}>. [\exists \bar{x}, \bar{y} \in I: \phi(\bar{x}, \bar{y}) \land P^\phi(x, y)] \]

For example:

\[ Q^2 = \lambda(x, y). [\exists \bar{x}, \bar{y} \in I: \phi(\bar{x}, \bar{y}) \land (\bar{x} \geq 1000) \land (\bar{y} > 0) \land (x = \bar{x} + (b - \bar{x}) \text{div} \bar{y}) \land (y = \bar{y})] \]
\[ = \lambda(x, y). [\exists \bar{x} \in I: (\exists j: \phi(x - jy, y) \land (x - jy \geq 1000) \land (y > 0) \land (x = \bar{x} + jy) \land j = (b - \bar{x}) \text{div} y)] \]
\[ = \lambda(x, y). [(y > 0) \land (\exists j \geq 0: \phi(x - jy, y) \land (1000 \leq x - jy < x \leq b) \land (j = j + (b - x) \text{div} y))] \]
\[ = \lambda(x, y). [(y > 0) \land (\exists j \geq 0: \phi(x - jy, y) \land (1000 \leq x - jy < x \leq b < x + y))] \]
We now recommence the semantic analysis of this program, but this time using backward equations.

10-5.3.2. Backward semantic analysis

The system $P = B_\pi(\Psi)(P)$ (where $B_\pi(\Psi) \in ((P \rightarrow B) \rightarrow (P \rightarrow B))$) of backward semantic equations associated with the example program $\pi$ and an exit specification $\Psi \in (P \rightarrow B)$ is the following:

$P_1 = \lambda \langle x, y \rangle. \{ ((x \in I) \land (x \geq 1000) \land P_4(x, y)) \lor ((x \in I) \land (x < 1000) \land P_4(x, y)) \}$

$P_2 = \lambda \langle x, y \rangle. \{ ((x + y) \in I) \land P_4(x + y, y) \}$

$P_3 = \lambda \langle x, y \rangle. \{ ((x \in I) \land (x \geq 1000) \land P_4(x, y)) \lor ((x \in I) \land (x < 1000) \land P_4(x, y)) \}$

$P_4 = \Psi$

The set of entry states which are ascendants of the exit states (i.e., cause the program to terminate properly) is characterized by

$\sigma_e(\langle v, \land pre(\tau^*)(v) \rangle)$

$= \sigma_e(\langle v, \land ifp(\lambda x. [v \lor pre(\tau)(x)]) \rangle)$  [Theorem 10-9(2)]

$= \sigma_e(\langle v, \land ifp(B_\pi(\lambda m. true)) \rangle)$  [Theorem 10-19(1)]

$= \sigma_e(\langle \bigvee_{s \in \lambda} \lambda x. [v(x) \land s \in \lambda^{-1}(U) \land ifp(B_\pi(\lambda m. true))(\lambda s(x))] \rangle)$

$= \sigma_e(\langle ifp(B_\pi(\lambda m. true))(\circ i_4) \rangle)$

The least fixed point $P^=\pi$ of the above system of equations where $\Psi = \lambda \langle x, y \rangle. [true]$ is

$P_1^= = \lambda \langle x, y \rangle. \{ (x < 1000) \lor (y < 0) \}$

$P_2^= = \lambda \langle x, y \rangle. \{ ((x + y) \in I) \land ((x + y < 1000) \lor (y < 0)) \}$

$P_3^= = \lambda \langle x, y \rangle. \{ (x < 1000) \lor (y < 0) \}$

$P_4^= = \lambda \langle x, y \rangle. [true]$

The set of entry states which do not lead to a run-time error (i.e., cause the program to properly terminate or diverge) is characterized by

$\sigma_e(\langle v, \land \neg pre(\tau^*)(v) \rangle)$

$= \sigma_e(\langle v, \land gfp(\lambda x. [v \land pre(\tau)(x)]) \rangle)$  [Theorem 10-9(3)]

$= \sigma_e(\langle v, \land gfp(B_\pi(\lambda m. true)) \rangle)$  [Theorem 10-19(2)]

$= gfp(B_\pi(\lambda m. true))$, 

"SEC. 10-5 / SEMANTIC ANALYSIS OF PROGRAMS" 325
The greatest fixed point \( Q^n \) of the above system of equations where \( \Psi = \lambda(x, y).[\text{true}] \) can be computed iteratively starting from \( Q^0 = \lambda(x, y).[\text{true}] \), \( i = 1, \ldots, 4 \), inventing the general term of a chaotic iteration sequence, and passing to the limit:

\[
Q^1 = \lambda(x, y).[(y \leq 0) \vee (x < 1000)]
\]
\[
Q^2 = \lambda(x, y).[((x + y \leq 0) \vee (x + y < 1000)) \wedge ((x + y) \in I)]
\]
\[
Q^3 = \lambda(x, y).[(y \leq 0) \wedge (x < 1000)]
\]
\[
Q^4 = \lambda(x, y).[\text{true}]
\]

The set of entry states leading to a run-time error is characterized by \( \lambda(x, y) \in P_1 . [\neg Q^n(x, y)] = \lambda(x, y) \in P_1 . [(y > 0) \wedge (x \geq 1000)] \).

Equivalently, the set of ascendants of the run-time error states is characterized by \( \text{pre}(\rho^*)(v_c) \), which, according to Theorem 10-9(3) and Theorem 10-19(3), is equal (up to the isomorphism \( \sigma \)) to the least solution \( R^\omega \) to

\[
P_1 = \lambda(x, y).[(x \geq 1000) \wedge P_2(x, y)]
\]
\[
P_2 = \lambda(x, y).[((x + y) \in I) \in P_3((x + y, y)) \vee ((x + y) \notin I)]
\]
\[
P_3 = \lambda(x, y).[(x \geq 1000) \wedge P_4(x, y)]
\]
\[
P_4 = \lambda(x, y).[\text{false}]
\]
\[
P_5 = \lambda(x, y).[\text{true}]
\]

that is, \( R_1 = R_2 = \lambda(x, y).[(x \geq 1000) \wedge (y > 0)] \), \( R_2 = \lambda(x, y).[((x + y) \geq 0 \wedge (y > 0)) \vee ((x + y) \notin I)] \), \( R_3 = \lambda(x, y).[\text{false}] \), \( R_4 = \lambda(x, y).[\text{true}] \).

The set of entry states which cause the program to diverge is characterized by \( \lambda(x, y) . [Q^n(x, y) \wedge \neg P^n(x, y)] = \lambda(x, y) . [(x \geq 1000) \wedge (y = 0)] \).

Equivalently, the states which cause the program to diverge can be characterized by \( \neg \text{pre}(\rho^*)(v_c) \vee v_c = gfp(\lambda x . [\neg w_x \wedge \neg w_c \vee \text{pre}(\rho)(a)]) \), which, according to Theorem 10-19(4) is equal (up to the isomorphism \( \sigma \)) to \( D^\omega = gfp(\lambda x . [\text{false}]) \), that is, \( D_1 = D_2 = D_3 = \lambda(x, y).[(x \geq 1000) \wedge (y = 0)] \) and \( D_4 = D_5 = \lambda(x, y).[\text{false}] \).

The set of descendants of the input states satisfying an entry condition \( \phi \in (P \rightarrow B) \) is characterized by

\[
\text{post}(\rho^*)(v_c, \sigma^{-1}(\phi)) = \lambda[s_i \in S : v_c(s_i) \wedge \sigma^{-1}(\phi)(s_i) \wedge \text{lfp}(\lambda x . [\text{ls}.[s \in \bar{s}]) \vee \text{pre}(\tau)(a))(s_i))]
\]
\[
= \lambda[s_i \in S : v_c(s_i) \wedge \sigma^{-1}(\phi)(s_i) \wedge \text{lfp}(\lambda x . [\text{ls}.[s \in \bar{s}]) \vee \text{pre}(\tau)(a))(s_i)]
\]
\[
= \lambda[s_i \in S : v_c(s_i) \wedge \text{lfp}(\lambda x . [\text{ls}.[s \in \bar{s}]) \vee \text{pre}(\tau)(a))(s_i)]
\]

According to Theorem 10-19(5), the direct decomposition of \( \lambda x . [\text{ls}.[s \in \bar{s}]) \vee \text{pre}(\tau)(a)] \) is the following when \( \tau \) is defined by our example program.
\[ P_1 = \lambda \langle x, y \rangle. [\langle 1, \langle x, y \rangle \rangle = \bar{s}] \lor (x \in I \land x \geq 1000 \land P_2(x, y)) \]
\[ \lor (x \in I \land x \leq 1000 \land P_4(x, y)) \lor (P_5(x, y) \land x \notin I) \]
\[ P_2 = \lambda \langle x, y \rangle. [\langle 2, \langle x, y \rangle \rangle = \bar{s}] \lor (x + y \in I \land P_3(x + y, y)) \]
\[ \lor (P_6(x, y) \land x + y \notin I) \]
\[ P_3 = \lambda \langle x, y \rangle. [\langle 3, \langle x, y \rangle \rangle = \bar{s}] \lor (x \in I \land x \geq 1000 \land P_4(x, y)) \]
\[ \lor (x \in I \land x < 1000 \land P_4(x, y)) \lor (P_6(x, y) \land x \notin I) \]
\[ P_4 = \lambda \langle x, y \rangle. [\langle 4, \langle x, y \rangle \rangle = \bar{s}] \lor P_5(x, y) \]
\[ P_5 = \lambda \langle x, y \rangle. [\langle 5, \langle x, y \rangle \rangle = \bar{s}] \lor P_6(x, y) \]

If \( P^\alpha(\bar{s}) \) denotes \( \lambda s. [\lambda \alpha. [s = \bar{s}] \lor \text{pred}(\alpha)(s) \lor \sigma^{-1}] \), we determine that
\[ P^\alpha(\bar{s}) = \lambda \langle x, y \rangle. [\langle 1, \langle x, y \rangle \rangle = \bar{s}] \]
\[ \lor (\exists j \geq 0: (\forall i \in [0, j], 1000 \leq x + iy \leq b) \]
\[ \land (\langle 2, \langle x + jy, y \rangle \rangle = \bar{s}) \]
\[ \lor (\exists j \geq 1: (\forall i \in [0, j - 1], 1000 \leq x + iy \leq b) \]
\[ \land (x + jy \in I) \land (\langle 3, \langle x + jy, y \rangle \rangle = \bar{s}) \]
\[ \lor (\exists j \geq 0: (\forall i \in [0, j - 1], 1000 \leq x + iy \leq b) \]
\[ \land (x + jy < 1000) \land (\langle 4, \langle x + jy, y \rangle \rangle = \bar{s}) \]
\[ \lor (\exists j \geq 1: (\forall i \in [0, j - 1], 1000 \leq x + jy \leq b) \]
\[ \land (x + jy \notin I) \land (\langle 5, \langle x + (j - 1)y, y \rangle \rangle = \bar{s}) \]}

At program point \( i \), the set of descendants of the input states satisfying \( \phi \) is
\[ \sigma_i(\lambda \tilde{s}. [\exists m_r \in U_r: \phi(m_r) \land P^\alpha(\bar{s})(m_r)]) = \lambda m. [\exists m_r \in U_r: \phi(m_r) \land P^\alpha(\bar{s})(m_r)] \]

For our example
\[ \lambda \langle x, y \rangle. [\exists \langle x, y \rangle \in P^\alpha: \phi(x, y) \land P^\alpha(\langle i, \langle x, y \rangle \rangle)(\langle x, y \rangle)] \]
when \( i = 2 \), this is equal to
\[ \lambda \langle x, y \rangle. [\exists \langle x, y \rangle \in P^\alpha: \phi(x, y) \land (\exists j \geq 0: (\forall i \in [0, j], \]
\[ 1000 \leq x + iy \leq b) \land (\langle 2, \langle x + jy, y \rangle \rangle = (\langle 2, \langle x, y \rangle \rangle))] \]
\[ = \lambda \langle x, y \rangle. [\exists j \geq 0: \phi(x - jy, y) \land 1000 \leq \text{min}(x - jy, x) \]
\[ \land (\text{max}(x - jy, x) \leq b)] \]

10.5.3.3. Forward versus backward semantic analysis of programs

In the literature on program verification, backward program analysis is often preferred to forward analysis (e.g., [Dijk76]). Theorem 10-19(1)-(2) clearly shows that the two approaches are not strictly equivalent, but this point of view is complemented by Theorem 10-13 and Section 10-5.3, which show that, using symbolic variables, one approach can serve as a substitute for the other.
10-6. APPROXIMATE ANALYSIS OF PROGRAMS

Although the above approach to solving semantic fixed point equations can lead to complete information about program behavior, it is essentially mathematically ideal since it is well known that decision problems connected with programs such as termination are algorithmically unsolvable. The trouble is that fixed points are obtained as limits of infinitely long iteration sequences and that machines are unable (and humans, as well, for nontrivial examples) to guess a suitable induction hypothesis to be used in the induction step which avoids the need to compute all terms of the iteration sequence. Hence we must limit ourselves to constructive methods which automatically compute approximate solutions to the semantic equations. Such approximate information is often useful. For example in program verification systems, a total correctness proof (Examples 10-6, 10-14) can be approximated by a partial correctness proof (Examples 10-3, 10-5, 10-12). Optimizing compilers only need a conservative approximate analysis of programs, since whenever insufficient information is available no optimization will take place. For example if an array index cannot be proved to be within the array bounds, a runtime check can be generated. In the same way, a large part of the debugging process can be automated to detect some but not all programming errors.

An example of approximate information about the behavior of a program \( \pi(S, \tau, \nu_e, \nu_m, \nu) \) is given by a predicate \( P \in (S \rightarrow B) \) characterizing a superset of the descendants of the entry states satisfying an input specification \( \phi \), that is, such that \( \text{post}(\tau^*)(\nu_e \land \phi) \Rightarrow P \). \( P \) gives a partial answer to the termination problem since if \( P \land \nu_e \) is false, then \( \pi \) surely always diverges, whereas if \( P \land \nu_e \) is not false, then one cannot conclude whether \( \pi \) terminates or not. Also \( P \) can be used for a partial correctness proof (Example 10-3).

It follows from Theorem 10-9(1) that \( P \) must be an upper approximation of the least solution to the equation \( \alpha = (\nu_e \land \phi) \lor \text{post}(\tau(\alpha)) \Rightarrow P \).

Another example of approximate information about the behavior of a program \( \pi(S, \tau, \nu_e, \nu_m, \nu) \) is given by a predicate \( Q \in (S \rightarrow B) \) which is a lower approximation of the greatest solution to the equation \( \alpha = -\nu_m \land -\nu_e \land \text{pre}(\tau(\alpha)) \), that is, (Theorem 10-9(5)) such that \( Q \Rightarrow -\text{pre}(\tau^*) (\nu_e \land \nu_m) \). The approximation is that \( Q \) characterizes only a subset of the states which cause \( \pi \) to diverge.

Hence an approximate program analysis can be automated by effectively computing a lower or upper approximation of the least or greatest solution to the fixed point equations \( x = f(x) \) on \( (S \rightarrow B) \) considered in Theorem 10-9.

Our approach consists in solving a simplified equation \( x = f'(x) \) on a complete lattice \( L \) which is a simpler and computer-representable image of \( (S \rightarrow B) \). A solution to the simplified equation \( x = f'(x) \) will be computed iteratively using an extrapolation technique in order to accelerate the con-
vergence when necessary. We will consider only the upper approximation of least fixed points since the other cases are duals to it.

10-6.1. Considering Simplified Equations

Assume we have to compute an upper approximation of the least fixed point \( lfp(f) \) of an isotone operator \( f \) on a complete lattice \( L(\equiv, \perp, \top, \sqcap, \sqcup, \sqsubseteq) \). We will consider a simplified and computer-representable image \( \tilde{L} \) of \( L \) and solve a less complex equation \( x = f(x) \) on \( \tilde{L} \). [Cous79] justifies the fact that the following connection must be established between \( L \) and \( \tilde{L} \).

Definition 10-20. Let \( L(\equiv, \perp, \top, \sqcup, \sqsubseteq) \) and \( L(\equiv, \perp, \top, \sqcup, \sqsubseteq, \sqsupseteq) \) be complete lattices. \( \tilde{L} \) is an upper approximation of \( L \) iff there exists a one-to-one complete \( \sqsubseteq \)-morphism \( y \) from \( \tilde{L} \) onto \( L \).

Example 10-21. Let \( \pi \) be a program over \( n \) integer variables \( X_1, \ldots, X_n \). Assume that at each program point \( i \) we have to determine the signs of the variables \( X_1, \ldots, X_n \). This consists in finding out \( Q_i \) such that \( lfp(F_i(\phi)) \Rightarrow Q_i \), where \( Q_i \) belongs to the subset \( L_i' \sqsubseteq L_i = (\mathbb{P} \rightarrow B) \) of predicates of the form \( \lambda(X_1, \ldots, X_n). \left( \bigwedge_{j=1}^{n} P_j(X_j) \right) \), where \( P_j \) belongs to the subset \( L_i' = \{ \lambda X. [false], \lambda X. [X = 0], \lambda X. [-b - 1 \leq X \leq 0], \lambda X. [0 \leq X \leq b], \lambda X. [true] \} \) of \( L_i = (\mathbb{P} \rightarrow B) \). We can define an upper approximation \( \tilde{L}_2 \) of \( L_2 \) as shown in Fig. 10-5.

\[
\tilde{L}_2 = \begin{array}{c}
\top \\
\downarrow \rightarrow \lambda X. [false] \\
\downarrow \rightarrow \lambda X. [-b - 1 \leq X \leq 0] \\
0 \rightarrow \lambda X. [X = 0] \\
\downarrow \rightarrow \lambda X. [0 \leq X \leq b] \\
\top \rightarrow \lambda X. [true] \\
\end{array}
\]

Figure 10-5

where \( \gamma_2 \in (\tilde{L}_2 \rightarrow L_2) \) is defined by

\[
\gamma_2 = \lambda s. \text{case } s \text{ in}
\begin{align*}
\perp & \rightarrow \lambda X. [false] \\
\ast & \rightarrow \lambda X. [-b - 1 \leq X \leq 0] \\
0 & \rightarrow \lambda X. [X = 0] \\
\ast & \rightarrow \lambda X. [0 \leq X \leq b] \\
\top & \rightarrow \lambda X. [true] \\
\text{esac}
\end{align*}
\]
The upper approximation $\bar{L}_1$ of $L_1$ can be defined as

$$\bar{L}_1 = \{ \langle s_1, \ldots, s_n \rangle : (\forall j \in [1, n], s_j \in \bar{L}_2 - \{ \perp \}) \cup \{ \perp, \ldots, \perp \} \}$$

and $\gamma_1 \in (L_1 \rightarrow L_1)$ is such that

$$\gamma_1 = \lambda \langle s_1, \ldots, s_n \rangle \left[ \lambda \langle X_1, \ldots, X_n \rangle \left[ \bigwedge_{j=1}^n \gamma_j(s_j)(X_j) \right] \right]$$

Intuitively, $\gamma$ gives the meaning $\gamma(x)$ of the elements $\bar{x}$ of $\bar{L}$. $\gamma$ is assumed to be one-to-one so that no two distinct elements of $\bar{L}$ have the same meaning. The hypothesis that $\gamma$ is a complete $\sqcap$-morphism implies that for all $x \in L$ the set $\{ \bar{y} \in \bar{L} : x \sqsubseteq \gamma(\bar{y}) \}$ of upper approximations of $x$ in $\bar{L}$ has at least one element, namely $\alpha(x) = \bigcap\{ \bar{y} \in \bar{L} : x \sqsubseteq \gamma(\bar{y}) \}$. The close relationship between $\alpha$ and $\gamma$ is described by the following:

**Theorem 10.22.** Let $\alpha \in (L \rightarrow \bar{L})$ be $\lambda x. \bigcap\{ \bar{y} \in \bar{L} : x \sqsubseteq \gamma(\bar{y}) \}$. $\alpha$ is a complete $\sqcap$-morphism from $L$ onto $\bar{L}$. $\alpha \circ \gamma = \lambda x. [\alpha(\gamma(x))]$ is the identity function on $\bar{L}$. $\gamma \circ \alpha$ is an isotone, idempotent ($\gamma \circ \alpha \circ \gamma = \gamma \circ \alpha$) and extensive ($\forall x \in L, x \sqsubseteq \gamma \circ \alpha(x)$) operator on $L$. Moreover $\gamma = \lambda \bar{y}. \bigcap\{ x \in L : \alpha(x) \subseteq \bar{y} \}$.

**Example 10.23.** Coming back to Example 10.21, we have:

$$\alpha_2 = \lambda P \in (I \rightarrow B). [\text{if } P = \lambda X. \text{false} \text{ then } \perp \text{ else if } P \Rightarrow \lambda X. [X = 0] \text{ then } 0 \text{ else if } P \Rightarrow \lambda X. [X \leq 0] \text{ then } \perp \text{ else if } P \Rightarrow \lambda X. [X \geq 0] \text{ then } \perp \text{ else } \top \text{ if}]$$

$$\alpha_1 = \lambda P \in (I^* \rightarrow B). \left[ \prod_{j=1}^n \alpha_2(\lambda X. [\exists X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_n] \in I^{n-1}: \right. \left. P(X_1, \ldots, X_{j-1}, X, X_{j+1}, \ldots, X_n) ) \right]$$

For example,

$$\alpha_1(\lambda \langle x, y \rangle. [\langle x \geq 1 \rangle \land (x + y = 2) \land \text{even}(y) ]\rangle$$

$$= \langle \alpha_2(\lambda x. [\langle 1 \leq x \leq 0 \rangle \land \text{even}(x) ]), \alpha_2(\lambda y. [\langle 0 \leq 2 \leq y \leq 1 \rangle \land \text{even}(y) ]) \rangle$$

$$= \langle \top, \bot \rangle$$

An upper approximation of the least fixed point $\text{lp}(f)$ of an isotone operator $f$ on the complete lattice $L$ can be defined using an approximate image $\bar{f}$ of $f$ on $(L \rightarrow \bar{L})$ as follows:
Definition 10-24. Let $\bar{L}(\subseteq, \sqsubseteq, \top, \sqcup, \sqcap)$ be an upper approximation of $L(\subseteq, \sqsubseteq, \top, \sqcup, \sqcap)$ and $f$ be an isitone operator on $L$. $f$ is an upper approximation of $f$ on $\bar{L}$ if $f$ is an isitone operator on $\bar{L}$ such that $(\forall \bar{x} \in L, \alpha(f(\gamma(\bar{x})))) \subseteq f(\bar{x}))$.

Theorem 10-25. If $f$ is an upper approximation of $f$ on $\bar{L}$, then $\text{lfp}(f) \equiv \gamma(\text{lfp}(f))$ and $\text{gfp}(f) \equiv \gamma(\text{gfp}(f))$.

Once an application-dependent upper approximation has been chosen (Definition 10-20) and knowing the construction rules for associating a system of semantic equations with a program (Theorems 10-17 and 10-19), one can determine the construction rules for associating a system of approximate equations with a program.

Example 10-26. Let us determine the construction rules of the approximate equations corresponding to Example 10-21. For simplicity we consider only the forward rule for the assignment $x := x + 1$ in a program with a single variable $x$. The exact rule is Definition 10-16.

$$P_i = \bigvee_{j \in \text{pred}_a(i)} \text{post}(\lambda x.[x + 1](P_j)) \text{ where } P_i, P_j \in \bar{L}$$

The approximate rule is Definition 10-24.

$$P_i = a_2\left(\bigvee_{j \in \text{pred}_a(i)} \text{post}(\lambda x.[x + 1](\gamma_2(P_j)))\right) \text{ where } P_i, P_j \in \bar{L}$$

$$= \bigwedge_{j \in \text{pred}_a(i)} a_2(\text{post}(\lambda x.[x + 1](\gamma_2(P_j)))) \quad (\text{Theorem 10-22})$$

It remains to analyze:

$$a_2(\text{post}(\lambda x.[x + 1](\gamma_2(P_j))))$$

$$= a_2(\lambda x. [\exists x' \in I: \gamma_2(P)(x') \land x' \in \text{dom}(\lambda x.[x + 1]) \land x = x' + 1])$$

$$= a_2(\lambda x. \gamma_2(P)(x - 1) \land (-b - 1 \leq x \leq b - 1))$$

According to the possible values of $P$, we can distinguish the following cases:

$$a_2(\text{post}(\lambda x.[x + 1](\gamma_2(\bot)))) = a_2(\lambda x.[\text{false}] = \bot$$

$$a_2(\text{post}(\lambda x.[x + 1](\gamma_2(0)))) = a_2(\lambda x.[x = 1]) = +$$

$$a_2(\text{post}(\lambda x.[x + 1](\gamma_2(-)))) = a_2(\lambda x.[-b \leq x \leq b - 1]) = \top$$

$$a_2(\text{post}(\lambda x.[x + 1](\gamma_2(\bot)))) = a_2(\lambda x.[-b \leq x \leq b - 1]) = +$$

$$a_2(\text{post}(\lambda x.[x + 1](\gamma_2(\top)))) = a_2(\lambda x.[-b \leq x \leq b - 1]) = \top$$

More generally the approximate forward assignment rule consists, given the signs of the variables before the assignment, in applying the rules of signs to the right-hand-side expression $(\downarrow + \downarrow = \downarrow, \downarrow + \downarrow = \top, \neg \downarrow = \downarrow = \neg \downarrow, \neg \downarrow \ast \downarrow = \downarrow, \ldots)$ and assigning the result to the left-hand-side variable. The signs of the other variables are left unchanged.
We can now write a program, which, given a program text $\pi$, constructs the associated system of approximate equations $x = f(x)$. Whenever $\bar{L}$ satisfies the ascending chain condition, we can write a program for computing the least fixed point $lfp(\bar{f})$ of any isotone operator $\bar{f}$ on $\bar{L}$. Starting from the infimum of $\bar{L}$, the algorithm can proceed iteratively using any efficient chaotic strategy until the iterates stabilize.

Example 10-27. The system of approximate equations associated with the introductory example program

\[
\begin{align*}
\{1\} & \quad \text{while } x \geq 1000 \text{ do} \\
\{2\} & \quad x := x + y; \\
\{3\} & \quad \text{od}; \\
\{4\} & \quad \text{is the following:}
\end{align*}
\]

\[
\begin{align*}
\langle x_1, y_1 \rangle &= \phi \\
\langle x_2, y_2 \rangle &= \text{smash}(\langle \text{if } (x_1 \sqcup x_2) \equiv \bot \text{ then } \bot \quad \text{else } y_1 \sqcup y_2 \rangle) \\
\langle x_3, y_3 \rangle &= \text{smash}(\langle x_2 + y_2, y_2 \rangle) \\
\langle x_4, y_4 \rangle &= \text{smash}(\langle x_2 \sqcup y_2, y_1 \sqcup y_2 \rangle) \\
\langle x_5, y_5 \rangle &= \text{if } (x_3 \equiv 0) \lor (y_3 \equiv 0) \lor ((x_2 = \bot) \land (y_3 = \bot)) \lor ((x_2 = \bot) \\
& \quad \land (y_3 = \bot)) \text{ then } \langle \bot, \bot \rangle \text{ else } \langle x_2, y_2 \rangle \text{ fi}
\end{align*}
\]

where $\text{smash}(\langle x, y \rangle) = \text{if } (x = \bot) \text{ or } (y = \bot) \text{ then } \langle \bot, \bot \rangle \text{ else } \langle x, y \rangle \text{ fi}$.

Taking $\phi = \langle \bot, \bot \rangle$, a chaotic iterative resolution corresponding to a symbolic execution of the program is the following:

\[
\begin{align*}
\langle x_2^0, y_2^0 \rangle &= \langle \bot, \bot \rangle \quad i = 1, \ldots, 4, \xi \\
\langle x_1, y_1 \rangle &= \phi = \langle \bot, \bot \rangle \\
\langle x_2^1, y_2^1 \rangle &= \text{smash}(\langle \text{if } (x_1^1 \sqcup x_1^2) \equiv \bot \text{ then } \bot \quad \text{else } y_1^1 \sqcup y_1^2 \rangle) = \langle \bot, \bot \rangle \\
\langle x_3, y_3 \rangle &= \text{smash}(\langle x_1^1 + y_1^1, y_1^1 \rangle) = \langle \top, \bot \rangle \\
\langle x_2^2, y_2^2 \rangle &= \text{smash}(\langle \text{if } (x_1^1 \sqcup x_1^2) \equiv \bot \text{ then } \bot \quad \text{else } y_1^1 \sqcup y_1^2 \rangle) = \langle \bot, \bot \rangle
\end{align*}
\]

Stabilizing around the loop, the remaining components are

\[
\begin{align*}
\langle x_2^3, y_2^3 \rangle &= \text{smash}(\langle x_1^1 \sqcup x_1, y_1^1 \sqcup y_1^2 \rangle) = \langle \top, \bot \rangle \\
\langle x_2^4, y_2^4 \rangle &= \text{if } (x_4^2 \equiv 0) \lor (y_4^2 \equiv 0) \lor ((x_3^2 = \bot) \land (y_3^2 = \bot)) \\
& \quad \lor ((x_4^2 = \bot) \land (y_3^2 = \bot)) \text{ then } \langle \bot, \bot \rangle \text{ else } \langle x_2, y_2 \rangle \text{ fi} \\
& = \langle \bot, \bot \rangle
\end{align*}
\]
The results are approximate but not useless since, e.g., they prove that no overflow can occur when the entry specification \((x \geq 0 \land y \leq 0)\) is true.

10-6.2. Speeding Up the Convergence of Chaotic Iterations Using Extrapolation Techniques

When \(\bar{L}\) does not satisfy the ascending chain condition, infinitely many steps may be necessary in order to ensure the convergence of a chaotic version of the iterates \(\bigcup_{i \geq 0} \bar{L}^i(\bot)\) to the least fixed point \(\mu p(f)\). It may also be the case that the number of iterations required is not infinite but so large as to be unacceptable in practice.

Example 10-28. In order to analyze programs over \(n\) integer variables \(X_1, \ldots, X_n\) let us consider:

1. The complete lattice \(\bar{L}_2 = \{\bot\} \cup \{\mathcal{I}, \mathcal{U}\}; -b - 1 \leq \mathcal{I} \leq \mathcal{U} \leq b\) where \(\bot\) is the infimum and \(\{(\mathcal{I}, \mathcal{U}) \subseteq I_2, U_2\}\) iff \((\mathcal{I} \leq \mathcal{I} \leq \mathcal{U}) \leq U\). The meaning of the elements of \(\bar{L}_2\) is given by \(\gamma_2(\bot) = \lambda X [\text{false}]\) and \(\gamma_2(\mathcal{I}, \mathcal{U}) = \lambda X [\mathcal{I} \leq X \leq \mathcal{U}]\).

2. The complete lattice \(\bar{L}_1 = \{\langle i_1, \ldots, i_n\rangle: (\forall j \in [1, n], i_j \in (\bar{L}_2 - \{\bot\})) \cup \{\bot, \ldots, \bot\}\}\) with \(\gamma_1 = \lambda\langle i_1, \ldots, i_n\rangle [\lambda X_1, \ldots, X_n \mapsto (\bigcup_{j=1}^{n} \gamma_2(i_j)(X_j))\].

3. The resolution of the equation \(x = [0, 0] \sqcup (x + 1)\) on \(\bar{L}_2\) where \(\bot + 1 = \bot\) and \(\mathcal{I} + 1 = [\mathcal{I} + 1, \min(\mathcal{I} + 1, b)]\) involves a very large number of iterates \(x^0 = \bot, x^1 = [0, 0], x^2 = [0, 1], x^3 = [0, 2], \ldots, x^{n+1} = [0, b], x^{n+2} = [0, b]\).

However, using the notion of extrapolation [Cous77a], we are able to speed up the convergence of the iteration and obtain a close approximation to the least fixed point in a finite number of steps.

Definition 10-29. A widening operator \(\mathcal{V} \in (N \rightarrow (\bar{L} \times \bar{L} \rightarrow \bar{L}))\) is such that:

1. \(\forall j > 0, \forall x, y \in \bar{L}, x \sqcup y \subseteq x \mathcal{V}(j) y\)

2. For any ascending chain \(y^0 \subseteq y^1 \subseteq \ldots \subseteq y^\omega \subseteq \ldots\) of elements of \(\bar{L}\), the ascending chain \(x^0 = y^0, x^1 = x^0 \mathcal{V}(1) y^1, \ldots, x^\omega = x^{\omega-1} \mathcal{V}(n) y^\omega, \ldots\) is eventually stable; i.e., there exists a \(k \geq 0\) such that for \(i \geq k, x^i = x^k\). \(x^k\) is called the limit of the chain.
A widening operator will be used to extrapolate each iterate until a post-fixed point is reached, in which case an upper approximation of the least fixed point has been found.

**Theorem 10-30.** Let \( \tilde{f} \) be an isotone operator on \( \tilde{L}(\subseteq, \perp, \top, \sqcup, \sqcap) \) and \( \lor \) be a widening operator. The limit \( u \) of the sequence
\[
\begin{align*}
x^0 &= d \\
x^{n+1} &= x^n \lor (n + 1) \tilde{f}(x^n) \text{ if } \neg(\tilde{f}(x^n) \subseteq x^n) \\
x^{n+1} &= x^n \text{ if } \tilde{f}(x^n) \subseteq x^n
\end{align*}
\]
can be computed in a finite number of steps. Moreover \( \text{lfp}(\tilde{f}) \subseteq u \) and \( \tilde{f}(u) \subseteq u \).

**Example 10-31.** When \( \tilde{L} \) satisfies the ascending chain condition, one can choose \( \forall f > 0, \forall (j) = \top \), in which case \( x^* = \text{lfp}(\tilde{f}) \). However the widening operator may be necessary when the convergence of the iterates must be speeded up as in Example 10-28. Although the above definition allows a different extrapolation to be applied at each step, the widening operator will most often be independent of \( j \). For Example 10-28, one can choose for all \( f > 0 \):
\[
\begin{align*}
\forall x \in \tilde{L}_2, \perp \lor \forall (j) x &= x \lor \forall (j) \perp = x \\
[l_1, u_1] \lor \forall (j) [l_2, u_2] &= [\text{if } 0 \leq l_2 < l_1 \text{ then } 0 \text{ elseif } l_2 < l_1 \text{ then } -b - 1 \text{ else } l_1, fi, \\
&\text{if } u_1 < u_2 \leq 0 \text{ then } 0 \text{ elseif } u_1 < u_2 \text{ then } b \text{ else } u_1, fi]
\end{align*}
\]
A widening \( \lor_1 \) on \( \tilde{L}_1 \) is obtained by applying \( \lor_2 \) componentwise:
\[
\langle i_1, \ldots, i_n \rangle \lor_1 \langle j_1, \ldots, j_n \rangle = \langle i_1 \lor_2 (j_1), \ldots, i_n \lor_2 (j_n) \rangle
\]

With regard to systems of equations it is not necessary to use a widening for each component. Considering the program graph as defined in Section 10-3.1 (or the reversed graph for backward equations), the widening operation need only be used for the components corresponding to loop head nodes. A set \( S \) of loop head nodes is a minimal set of vertices such that any oriented cycle in the graph contains an element of \( S \). In general such a set is not unique, and an arbitrary choice may be made (but when the graph is reducible, interval headers [Alle70] should be preferred).

**Example 10-32.** The loop head nodes of the program graph in Fig. 10-6 are marked \( \odot \).

**Example 10-33.** When considering the approximation defined in Example 10-28, the system of simplified forward equations corresponding to the program \( \odot \).
\textbf{Figure 10-6}

\begin{align*}
\{ & 1 \} \\
& \text{while } (10 \leq x) \land (x \leq 100) \text{ do} \\
& \{ & 2 \} \\
& \quad x := x + y; \\
& \{ & 3 \} \\
& \quad \text{od}; \\
& \{ & 4 \} \\
& \end{align*}

is the following:

\begin{align*}
\langle x_1, y_1 \rangle &= \phi \\
\langle x_2, y_2 \rangle &= \text{smash}(\langle (x_1 \lor x_3) \cap [10, 100] \rangle, \langle y_1 \lor y_3 \rangle) \\
\langle x_3, y_3 \rangle &= \text{smash}(\langle x_3 + y_2, y_3 \rangle) \\
\langle x_4, y_4 \rangle &= \text{smash}(\langle (x_1 \lor x_3) \cap (-b - 1, 9) \rangle, \langle y_1 \lor y_3 \rangle) \\
\langle x_5, y_5 \rangle &= \text{if } \langle x_2, y_2 \rangle = \langle \bot, \bot \rangle \text{ then } \langle \bot, \bot \rangle \text{ else underflow}(x_2, y_2) \\
& \quad \sqcup \text{overflow}(x_2, y_2) \text{ fi}
\end{align*}

335
where
\[ \forall x \in L_2, \bot \sqcup x = x \sqcup \bot = x, \]
\[ \bot \sqcap x = x \sqcap \bot = \bot, \]
\[ \bot + x = x + \bot = \bot, \]
\[ [l_1, u_1] \sqcup [l_2, u_2] = [\min(l_1, l_2), \max(u_1, u_2)] \]
\[ [l_1, u_1] \sqcap [l_2, u_2] = \text{if } \max(l_1, l_2) \leq \min(u_1, u_2) \text{ then } [\max(l_1, l_2), \min(u_1, u_2)] \]
\[ \text{else } \bot \]
\[ [l_1, u_1] + [l_2, u_2] = [l_1 + l_2, u_1 + u_2] \sqcap [-b - 1, b] \]
overflow([l_1, u_1], [l_2, u_2]) = if (u_1 \geq 1) \land (u_2 \geq 1) \text{ then } ([l_1, u_1]
\[ \sqcup [b - u_2 + 1, b]), ([l_2, u_2] \sqcap [b - u_1 + 1, b]) \text{ else } \bot \]
underflow([l_1, u_1], [l_2, u_2]) = if (l_1 \leq -1) \land (l_2 \leq -1) \text{ then } ([l_1, u_1]
\[ \sqcap [-b - 1, -b - 2 - l_2]), ([l_2, u_2] \]
\[ \sqcap [-b - 1, -b - 2 - l_1]) \text{ else } \bot \]
smash(⟨x, y⟩) = if (x = \bot) \lor (y = \bot) \text{ then } \bot \text{ else } ⟨x, y⟩fi

Taking \( \phi = ⟨9, 11⟩, [−1, +1]⟩ \) the resolution uses the widening operators of Example 10-31 for the loop head node 2:

\[ ⟨x^9_i, y^9_i⟩ = ⟨\bot, \bot⟩ \quad i = 1, \ldots, 4, \xi \]
\[ ⟨x^1_i, y^1_i⟩ = \phi = ⟨9, 11⟩, [−1, 1]⟩ \]
\[ ⟨x^2_i, y^2_i⟩ = ⟨x^3_i, y^3_i⟩ \lor V_i(1) \lor x^1_i \lor y^1_i⟩ \]
\[ = ⟨\bot \lor V_1(1) [10, 11], [−1, 1]⟩ = ⟨10, 11⟩, [−1, 1]⟩ \]
\[ ⟨x^4_i, y^4_i⟩ = smash(⟨x^9_i + y^9_i, y^9_i⟩) = ⟨9, 12⟩, [−1, 1]⟩ \]
\[ ⟨x^2_i, y^2_i⟩ = ⟨x^1_i, y^1_i⟩ \lor V_i(2) \lor smash(⟨x^1_i \lor x^1_i⟩ \lor [10, 100], [y^1_i \lor y^1_i⟩)⟩ \]
\[ = ⟨10, 11⟩, [−1, 1]⟩ \]

The effect of the widening is to extrapolate to zero or infinity (\( −b - 1 \) or \( b \)) the bounds which are not stable around the loop.

\[ ⟨x^3_i, y^3_i⟩ = smash(⟨x^9_i + y^9_i, y^9_i⟩) = ⟨9, 12⟩, [−1, 1]⟩ \]

According to the definition of the approximate iteration sequence in Theorem 10-30, we have

\[ ⟨x^2_i, y^2_i⟩ = ⟨x^2_i, y^2_i⟩ = ⟨10, b⟩, [−1, 1]⟩ \]

since \[ ⟨10, 100⟩, [−1, 1]⟩ = smash(⟨x^1_i \lor x^1_i⟩ \lor [10, 100], [y^1_i \lor y^1_i⟩)⟩ \]
\[ ≡ ⟨x^2_i, y^2_i⟩ \]. The remaining components are:
\( \langle x_1, y_1 \rangle = \text{smash}(\langle \langle x \sqcap x \rangle \cap [-b - 1, 9], \langle y \sqcup y \rangle \rangle) \)

\( = \langle [9, 9], [-1, +1] \rangle \)

\( \langle x_2, y_2 \rangle = \text{if} \left( \langle x_1, y_1 \rangle = \langle \bot, \bot \rangle \text{ then } \langle \bot, \bot \rangle \text{ else underflow}(x_2, y_2) \right) \)

\( = \langle \bot, \bot \rangle \cup \langle [b, b], [1, 1] \rangle = \langle [b, b], [1, 1] \rangle \)

The approximate result is

\( \langle x_1, y_1 \rangle = \langle [9, 11], [-1, 1] \rangle \)
\( \langle x_2, y_2 \rangle = \langle [10, b], [-1, 1] \rangle \)
\( \langle x_3, y_3 \rangle = \langle [9, b], [-1, 1] \rangle \)
\( \langle x_4, y_4 \rangle = \langle [9, 9], [-1, 1] \rangle \)
\( \langle x_5, y_5 \rangle = \langle [b, b], [1, 1] \rangle \)

Any upper approximation \( u \) of the least fixed point \( \text{lfp}(f) \) of an isotone operator \( f \) on a complete lattice \( \bar{L} (\sqsubseteq, \bot, \top, \sqcup, \sqcap) \) can be improved by any term of the decreasing chain \( x = u, \ldots, x^{n+1} = x^n \cap f(x^n), \ldots \). (We have \( \text{lfp}(f) \sqsubseteq u = x^n \). If by induction hypothesis \( \text{lfp}(f) \sqsubseteq x^n \), then by the fixed point property and isotony \( \text{lfp}(f) = f(\text{lfp}(f)) \sqsubseteq f(x^n) \), so that \( \text{lfp}(f) \sqsubseteq x^n \cap f(x^n) = x^{n+1} \), proving by recurrence that for all \( k \geq 0 \), \( \text{lfp}(f) \sqsubseteq x^k \).

Yet notice that when \( u \) is already a fixed point no improvement is possible. In general the chain is strictly decreasing, and when \( \bar{L} \) does not satisfy the descending chain condition, it may not stabilize. However one can stop the iteration process after any number of steps or use the following extrapolation technique.

Definition 10-34. A narrowing operator \( \Delta \in (\mathbb{N} \rightarrow ((\bar{L} \times \bar{L}) \rightarrow \bar{L})) \) is such that:

1. \( \forall j > 0, (\forall x, y \in \bar{L} : y \sqsubseteq x), y \sqsubseteq x \Delta(j) y \sqsubseteq x \)

2. For any decreasing chain \( y_0 \sqsupseteq y_1 \sqsupseteq \ldots \sqsupseteq y_n \sqsupseteq \ldots \) of elements of \( \bar{L} \), the descending chain \( x^0 = y_0, \ldots, x^n = x^{n-1} \Delta(n) y^n, \ldots \) is eventually stable.

Example 10-35. Coming back to Example 10-28, one can choose for all \( j > 0 \):

\( \forall x \in \bar{L}_2, \bot \Delta(j) x = x \Delta(j) \bot = \bot \)

\([l_1, u_1] \Delta_2(j) [l_2, u_2] = [\text{if} (l_1 = b - 1) \vee (l_1 = 0) \text{ then } l_2 \text{ else min}(l_1, l_2)_f, \text{ if} (u_1 = b) \vee (u_1 = 0) \text{ then } u_2 \text{ else max}(u_1, u_2)_f] \)
This narrowing operator attempts to improve the zero or infinite bounds \((-b - 1) \text{ and } b\) which might have been too imprecisely extrapolated by the widening operator.

**Theorem 10-36.** Let \(u \in L\) be such that \((\text{lfp}(\bar{f}) \subseteq u)\) and \((\bar{f}(u) \subseteq u)\). The decreasing chain \(x^0 = u, \ldots, x^k = x^{k-1} \Delta(n) \bar{f}(x^{k-1}), \ldots\) is eventually stable. Moreover \(\forall k \geq 0, \text{lfp}(\bar{f}) \subseteq x^k\).

**Example 10-37.** This is a continuation of Example 10-33.

\[
\langle x_4^1, y_4^1 \rangle = \langle x_3^1, y_3^1 \rangle \Delta(1) \text{ smash} \langle ((x_1 \cup x_2) \cap [10, 100], [y_1 \cup y_2]) \rangle
\]

\[
= \langle [10, 100], [\{1, 1\} \Delta(1)] [10, 100], [-1, 1] \Delta(1) [-1, 1] \rangle
\]

\[
= \langle [10, 100], [-1, 1], [1] \rangle
\]

\[
\langle x_4^2, y_4^2 \rangle = \text{smash} \langle x_4^1 + y_4^1, y_4^1 \rangle = \langle [9, 101], [-1, 1] \rangle
\]

\[
\langle x_4^2, y_4^2 \rangle = \langle x_4^1, y_4^1 \rangle \Delta(2) \text{ smash} \langle ((x_1 \cup x_2) \cap [10, 100], [y_1 \cup y_2]) \rangle
\]

\[
= \langle [10, 100], [\{1, 1\} \Delta(2)] [10, 100], [-1, 1] \Delta(2) [-1, 1] \rangle
\]

\[
= \langle [10, 100], [-1, 1], [1] \rangle
\]

Stabilization around the loop has been achieved. The components depending on \(\langle x_4^2, y_4^2 \rangle\) remain to be evaluated. The final result is

\[
\langle x_4^3, y_4^3 \rangle = \langle [9, 11], [-1, 1] \rangle
\]

\[
\langle x_4^4, y_4^4 \rangle = \langle [10, 100], [-1, 1] \rangle
\]

\[
\langle x_4^5, y_4^5 \rangle = \langle [9, 101], [-1, 1] \rangle
\]

\[
\langle x_4^6, y_4^6 \rangle = \langle [9, 9], [-1, 1] \rangle
\]

\[
\langle x_4^7, y_4^7 \rangle = \langle \bot, \bot \rangle
\]

10-6.3. Hierarchy of Approximate Program Analyses

Let us give three examples of approximate analysis of the same program. Given an array \(R\) of integers whose elements are sorted in increasing order, the following procedure searches for a given argument \(k\) and returns the position \(m\) such that \(R(m) = k\). When the search is unsuccessful, \(m = lb(R) - 1\) where \(lb(R)\) and \(ub(R)\) are respectively the least and greatest indices of \(R\).

```plaintext
type table = array [1, 100] of integer;
procedure binary-search (var R: table; value k: integer; result m: integer) =
    var bi, bs: integer;
begin
    bi := lb(R); bs := ub(R);
```
while $bi \leq bs$ do

    $m := (bi + bs) \text{ div } 2$;

    if $k = R(m)$ then
        $bi := bs + 1$;
    elseif $k < R(m)$ then
        $bs := m - 1$;
    else
        $bi := m + 1$;
    fi;

od;

if $R(m) \neq k$ then $m := lb(R) - 1$ fi;

end;

The approximation considered in Examples 10-21, 10-23, 10-26, and 10-27 can be briefly sketched using a geometrical analogy. A predicate $P$ over two numerical variables $x$ and $y$, whose characteristic set $\overline{P}$ is shown in Fig. 10-7(a), is approximated from above by the predicate characterizing the quadrant of the plane containing all the points of $\overline{P}$, as shown in Fig. 10-7(b). If, contrary to Example 10-21, we make a distinction between predicates such as $\lambda x. [x \geq 0]$ and $\lambda x. [x > 0]$ and if we are only concerned by the behavior
of the variables $b_i, b_s,$ and $m$ then the corresponding approximate analysis
of the procedure binary-search is the following:

\[
\begin{align*}
P_1 &= (b_i > 0) \land (b_s > 0) \\
P_2 &= (b_i > 0) \land (b_s > 0) \\
P_3 &= (b_i > 0) \land (b_s > 0) \land (m > 0) \\
P_4 &= (b_i > 0) \land (b_s > 0) \land (m > 0) \\
P_5 &= (b_i > 0) \land (b_s \geq 0) \land (m > 0) \\
P_6 &= (b_i > 0) \land (b_s > 0) \land (m > 0) \\
P_7 &= (b_i > 0) \land (b_s \geq 0) \land (m > 0) \\
P_8 &= (b_i > 0) \land (b_s > 0) \\
P_9 &= (b_i > 0) \land (b_s \geq 0)
\end{align*}
\]

The approximation considered at Examples 10-28, 10-31, 10-33, 10-35, and 10-37 is more precise and consists of approximating the characteristic set of $P$ by the smallest rectangle including it and whose sides run parallel with the axes, as shown in Fig. 10-7(c).

The corresponding analysis of the procedure binary-search is the following:

\[
\begin{align*}
P_{1} &= (b_{i} = 1) \land (b_{s} = 100) \\
P_{2} &= (1 \leq b_{i} \leq 100) \land (1 \leq b_{s} \leq 100) \\
P_{3} &= (1 \leq b_{i} \leq 100) \land (1 \leq b_{s} \leq 100) \land (1 \leq m \leq 100) \\
P_{4} &= (2 \leq b_{i} \leq 101) \land (1 \leq b_{s} \leq 100) \land (1 \leq m \leq 100) \\
P_{5} &= (1 \leq b_{i} \leq 100) \land (0 \leq b_{s} \leq 99) \land (1 \leq m \leq 100) \\
P_{6} &= (2 \leq b_{i} \leq 101) \land (1 \leq b_{s} \leq 100) \land (1 \leq m \leq 100) \\
P_{7} &= (1 \leq b_{i} \leq 101) \land (0 \leq b_{s} \leq 100) \land (1 \leq m \leq 100) \\
P_{8} &= (1 \leq b_{i} \leq 101) \land (0 \leq b_{s} \leq 100) \land (1 \leq m \leq 100) \\
P_{9} &= (1 \leq b_{i} \leq 101) \land (0 \leq b_{s} \leq 100) \land (0 \leq m \leq 100)
\end{align*}
\]

This analysis shows that all array accesses are correct, neither underflow nor overflow can occur, the integer division by 2 can be implemented by a logical right shift, $b_i$ and $b_s$ should have been declared $b_i$: 1..101; $b_s$: 0..100, and the result $m$ returned by the procedure is always included between the bounds 0 and 100.

Some programming languages allow a declaration such as:

```plaintext
type table: array[1,n] of integer;
```

where $n$ is a symbolic constant the value of which is known only at run time. In such a case a more precise program analysis might be necessary in order to discover relationships between $b_i$, $b_s$, $m$, and $n$. For example [Cous78], it can
be useful to look for linear equality or inequality relationships between the numerical variables of the program. This consists in approximating the characteristic set of a predicate $P$ by the convex-hull of this set, as shown in Fig. 10-7(d).

The corresponding analysis of the procedure binary-search is now:

$P_1 = (bi = 1) \land (bs = n)$

$P_2 = (1 \leq bi \leq bs \leq n)$

$P_3 = (1 \leq bi \leq bs \leq n) \land (2m \leq bi + bs \leq 2m + 1)$

(hence $1 \leq m \leq n$ since $m$ is integer)

$P_4 = (1 \leq bs \leq n) \land (bs \leq 2m \leq 2bs) \land (bi = bs + 1)$

$P_5 = (1 \leq bi \leq n) \land (2bi - 1 \leq 2m \leq bi + n) \land (m = bs + 1)$

$P_6 = (1 \leq bs \leq n) \land (bs \leq 2m \leq 2bs) \land (bi = m + 1)$

$P_7 = (m \leq bs + 1) \land (3bi \leq 2bs + n + 3) \land (3bi \leq 2bs + 2m + 3)$

$\land (2bi \leq 2bs + 3) \land (bs + m + 1 \leq bi + n)$

$\land (bi + m \leq bs + n + 1) \land (1 \leq n) \land (bs \leq n) \land (bs \leq 2m)$

$\land (bs + 4 \leq 3bi + m) \land (1 \leq 2m) \land (bs + 1 \leq bi + m)$

$P_8 = (1 \leq bi) \land (bs \leq n) \land (bs \leq bi - 1)$

Notice that at program point [8] nothing is known about $m$. Contrary to the previous analysis, it may be the case that $n < 1$, in which case $m$ is not initialized.

10-7. BIBLIOGRAPHIC NOTES

References [Schae73, Aho77 (chapter 14), Hech77] are introductions to program flow analysis which put emphasis on the boolean techniques (which historically appeared first). This bibliography is devoted to program analysis methods which do not make the hypothesis that the information to be gathered about programs can be naturally represented as boolean vectors.

Nonboolean program analysis techniques can be traced back to [Naur66] and [Sint72]; [Kild73, Wieg75] introduced iterative algorithms using a lattice satisfying the ascending chain condition in order to represent the information to be gathered about programs. The fixed point theory of approximate program analysis is discussed by [Kam77, Cous77a]. The problem of using iterative algorithms with lattices not satisfying the ascending chain condition is treated by [Cous77a]. [Tarj76] presents an efficient (but not general) implementation of iterative algorithms which extends [Kenn75b, Aho76]. Noniterative program analysis methods include [Grah76, Rei77, Rei78, Rose77a, Rose78a, Tarj75b]. Boolean techniques for interprocedural analy-
sis of recursive programs are proposed by [Spil72, Alle74, Lome75, Rose79, Bart77a], whereas [Cous77a, Shar80] handle more ambitious analyses related to program verification.

Automatic methods for program analysis have numerous applications including type determination [Jone76, Kapl78a, Tene74b], gathering information for automatic data structure selection in very-high-level languages [Schw75b, Schw75c], detection of induction variables and strength reduction [Fong75, Fong76] or discovery of generalized common subexpressions [Fong77] for set-theoretic languages, determination of affine equality relationships among variables of a program [Karr76], detection of programming errors [Fosd76, Gill77], static array bound checking [Cous77a, Cous78, Germ78, Harr77a, Suzu77, Wels77], determination of linear [Cous78] or nonlinear [Berm76] invariant assertions, synthesis of resource invariants for concurrent programs [ClaE79], and so on.

ACKNOWLEDGMENT

I owe a deep debt to C. Pair whose argumented advice to study program analysis techniques using the model of discrete dynamic systems was very helpful. I wish to thank Radhia Cousot for her collaboration.
Bibliography


BIBLIOGRAPHY


Hant76  HANTLER, S. L., and J. C. KING, "An Introduction to Proving the Correctness of Programs," *Comp. Surv.*, 8, no. 3 (September 1976), 331–353.


---


Shar81 SHARIR, M., and A. Pnueli, "Two Approaches to Interprocedural Data Flow Analysis," this volume, chap. 7.


Taj75b TARJAN, ROBERT ENDRE, "Solving Path Problems on Directed Graphs," STAN-CS-75-528 (November 1975), Computer Science Department, Stanford University, Stanford, CA.


Tene74b TENENBAUM, AARON, "Type Determination for Very High Level Languages," Report NSO-3 (October 1974), Computer Science Department, New York University.


Wel77 WELSH, J., "Economic Range Checking in PASCAL," Department of Computer Science, Queen's University, Belfast, Northern Ireland, October 1977.