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

Chapter 10

Semantic Foundations of Program Analysis

Patrick Cousot

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}
        $x := x + y$;
{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 $(x_0 < 1000) \vee (y_0 < 0)$.
2. The execution of the program never terminates if and only if $(1000 \leq x_0 \leq b) \wedge (y_0 = 0)$.
3. The execution of the program leads to a run-time error (by overflow) if and only if $(x_0 \geq 1000) \wedge (y_0 > 0)$.
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:

$$P_1 = \lambda\langle x, y\rangle.[(-b - 1 \leq x \leq b) \wedge (-b - 1 \leq y \leq b)]$$
$$P_2 = \lambda\langle x, y\rangle.[(1000 \leq x \leq b) \wedge (-b - 1 \leq y \leq b)]$$
$$P_3 = \lambda\langle x, y\rangle.[(1000 + y \leq x \leq min(b, b + y)) \\ \wedge (-b - 1 \leq y \leq b)]$$
$$P_4 = \lambda\langle x, y\rangle.[(-b - 1 \leq x < 1000) \wedge (-b - 1 \leq y \leq b)]$$

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, Pnue77] 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 ascendants 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 Radhia 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 aprroximate 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 ϵ is of in-degree 0, ω is of out-degree 0, and every vertex lies on a path from ϵ to ω.

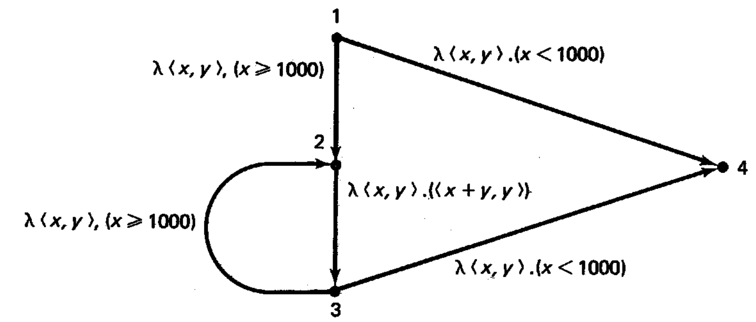

Figure 10-1

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 $L \in (E \longrightarrow I(U))$ 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 π specifies the sequence of successive states of the computation defined by π.

10-3.2.1. States

The set S of *states* is the set of pairs $\langle c, m\rangle$ 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 $\nu_\epsilon = \lambda\langle c, m\rangle.(c = \epsilon)$, $\nu_\omega = \lambda\langle c, m\rangle.(c = \omega)$ and $\nu_\xi = \lambda\langle c, m\rangle.(c = \xi)$.

10-3.2.2. State transition function

A program $\pi = (G, U, V)$ defines a *state transition function* $\bar{\tau} \in (S \longrightarrow S)$ as follows:

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

4. If $c_1 \in V$ is of out-degree 2, $\langle c_1, c_2 \rangle \in E, \langle c_1, c_3 \rangle \in E, L(\langle c_1, c_2 \rangle) = p, L(\langle c_1, c_3 \rangle) = \neg p, p \in I_t(U)$ then if $m \notin dom(p)$ then $\bar{\tau}(\langle c_1, m \rangle) = \langle \xi, m \rangle$ else if $p(m)$ then $\bar{\tau}(\langle c_1, m \rangle) = \langle c_2, m \rangle$ else $\bar{\tau}(\langle c_1, m \rangle) = \langle c_3, m \rangle$.

The *state transition relation* $\tau \in ((S \times S) \longrightarrow B)$ defined by π is $\lambda\langle s_1, s_2\rangle.(s_2 = \bar{\tau}(s_1))$.

10-3.2.3. Transitive closure of a binary relation

If $\alpha, \beta \in (S \times S \longrightarrow B)$ are two binary relations on S, their *product* $\alpha \circ \beta$ is defined as $\lambda\langle s_1, s_2\rangle.[\exists\, s_3 \in S: \alpha(s_1, s_3) \wedge \beta(s_3, s_2)]$. For any natural number n, the *n-extension* α^n of α is defined recursively as $\alpha^0 = e\alpha = \lambda\langle s_1, s_2\rangle.[s_1 = s_2]$, $\alpha^{n+1} = \alpha \circ \alpha^n$. The *(reflexive) transitive closure* of α is $\alpha^* = \lambda\langle s_1, s_2\rangle.[\exists\, n \geq 0: \alpha^n(s_1, s_2)]$.

10-3.2.4. Execution and output of a program

The execution of the syntactically valid program π starting from an initial state $s_1 \in S$ is said *to lead to an error* iff $[\exists\, s_2 \in S: \tau^*(s_1, s_2) \wedge \nu_\xi(s_2)]$, and to *terminate* iff $[\exists\, s_2 \in S: \tau^*(s_1, s_2) \wedge \nu_\omega(s_2)]$. Otherwise it is said to *diverge*. The *output* of the execution of a syntactically valid program π 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 τ 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 $\langle S, \tau, \nu_\epsilon, \nu_\omega, \nu_\xi \rangle$ such that S is a nonvoid set of *states*, $\tau \in ((S \times S) \longrightarrow B)$ where $B = \{true, false\}$ is the *transition relation* holding between a state and its possible successors, $\nu_\epsilon \in (S \longrightarrow B)$ characterizes the *entry states*, $\nu_\omega \in (S \longrightarrow B)$ characterizes the *exit states*, and $\nu_\xi \in (S \longrightarrow B)$ characterizes the *erroneous states*. It is assumed that the entry, exit, and erroneous states are disjoint $(\forall i, j \in \{\epsilon, \omega, \xi\}, (i \neq j) \longrightarrow (\forall s \in S, \neg(\nu_i(s) \wedge \nu_j(s))))$.

The following study is devoted to *total* ($\forall s_1 \in S, \exists s_2 \in S: \tau(s_1, s_2)$) and *deterministic* ($\forall s_1, s_2, s_3 \in S, (\tau(s_1, s_2) \wedge \tau(s_1, s_3)) \Rightarrow (s_2 = s_3)$) 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* ($\forall s_1, s_2 \in S, \tau(s_1, s_2) \Rightarrow \neg(\nu_\epsilon(s_2))$), the exit states are *stable* ($\forall s_1, s_2 \in S, (\nu_\omega(s_1) \wedge \tau(s_1, s_2)) \Rightarrow (s_1 = s_2)$), and the system is *without error recovery* ($\forall s_1, s_2 \in S, (\nu_\xi(s_1) \wedge \tau(s_1, s_2)) \Rightarrow \nu_\xi(s_2)$).

The *inverse* of $\tau \in ((S \times S) \rightarrow B)$ is $\tau^{-1} = \lambda\langle s_1, s_2\rangle.[\tau(s_2, s_1)]$. A system is *injective* if τ^{-1} is deterministic; it is *invertible* if it is injective and τ^{-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* ($\forall a \in L, a \sqsubseteq a$), *antisymmetric* ($\forall a, b \in L, (a \sqsubseteq b \wedge b \sqsubseteq a) \Rightarrow (a = b)$) and *transitive* ($\forall a, b, c \in L, (a \sqsubseteq b \wedge b \sqsubseteq c) \Rightarrow (a \sqsubseteq c)$). Given $H \subseteq L$, $a \in L$ 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 $\bigsqcup H$, 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*. $L(\sqsubseteq)$ is a *complete lattice* if the least upper bound $\bigsqcup H$ of H and the greatest lower bound $\bigsqcap H$ of H exist for all $H, H \subseteq L$. A complete lattice L has an *infimum* $\perp = \bigsqcap L$ and a *supremum* $\top = \bigsqcup L$.

An operator f on L is *strict* if $f(\perp) = \perp$, 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 $L(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ is a (nonempty) complete lattice with partial ordering $\sqsubseteq$. The *least fixed point* of f, in symbols $lfp(f)$, is $\bigsqcap\{x \in L: f(x) \sqsubseteq x\}$. Dually the *greatest fixed point* of f, in symbols $gfp(f)$ is $\bigsqcup\{x \in L: x \sqsubseteq f(x)\}$. 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 ($\forall x \in L, (f(x) \sqsubseteq x) \Rightarrow (lfp(f) \sqsubseteq x)$). The *Dual Recursion Induction Principle* is ($\forall x \in L, (x \sqsubseteq f(x)) \Rightarrow (x \sqsubseteq gfp(f))$).

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

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

A poset $L(\sqsubseteq)$ 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 $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$, then for some m we have $x_m = x_{m+1} = \ldots$. An operator f on $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ which is semi-$\sqcup$-continuous is necessarily isotone, 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-$\sqcup$-continuous. Also an operator f on a complete lattice L which is a *complete-$\sqcup$-morphism* (i.e., $\forall H \subseteq L, f(\bigsqcup H) = \bigsqcup f(H)$) is obviously semi-$\sqcup$-continuous.

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

Suppose $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$, $L'(\sqsubseteq', \bot', \top', \sqcup', \sqcap')$ are complete lattices and we have the commuting diagram of isotone functions shown in Fig. 10-2, where h is strict $(h(\bot) = \bot')$ and semi-$\sqcup$-continuous. Then $h(lfp(f)) = lfp(g)$.

$$\begin{tikzcd} L \arrow[r, "f"] \arrow[d, "h"'] & L \arrow[d, "h"] \\ L' \arrow[r, "g"'] & L' \end{tikzcd}$$

Figure 10-2

In a complete lattice $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$, a is a *complement* of b if $a \sqcap b = \bot$ and $a \sqcup b = \top$. A *uniquely complemented complete lattice* $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap, \neg)$ 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 $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ then $\lambda x.[\neg f(\neg x)]$ is an isotone operator on L, $gfp(f) = \neg lfp(\lambda x.[\neg f(\neg x)])$.

Let $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a complete lattice, $n \geq 1$, and F a semi-$\sqcup$-continuous operator on L^n. The system of equations

$$X = F(X)$$

which can be detailed as

$$X_j = F_j(X_1, \ldots, X_n) \qquad j = 1, \ldots, n$$

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:

$$X_j^{i+1} = F_j(X_1^i, \ldots, X_n^i) \qquad j = 1, \ldots, n$$

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] $lfp(F)$ is the least upper bound of any chaotic iteration sequence $\{X^i: i \geq 0\}$ where $X^0 = \langle \bot, \ldots, \bot \rangle$ and

$$\begin{aligned} X_j^{i+1} &= F_j(X_1^i, \ldots, X_n^i) && \text{if } j \in J_i \\ X_j^{i+1} &= X_j^i && \text{if } j \notin J_i \end{aligned}$$

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 $(S, \tau, \nu_\epsilon, \nu_\omega, \nu_\xi)$, the set of *descendants of the states satisfying a condition* $\beta \in (S \longrightarrow B)$ is by definition the set characterized by

$$\lambda s_2.[\exists s_1 \in S: \beta(s_1) \wedge \tau^*(s_1, s_2) = post(\tau^*)(\beta)$$

using the notation

$$\begin{aligned} &post \in (((S \times S) \longrightarrow B) \longrightarrow ((S \longrightarrow B) \longrightarrow (S \longrightarrow B))) \\ &post = \lambda\theta.[\lambda\beta.[\lambda s_2.[\exists s_1 \in S: \beta(s_1) \wedge \theta(s_1, s_2)]]] \end{aligned}$$

Example 10-3. Let π be a program defining a total and deterministic system $(S, \tau, \nu_\epsilon, \nu_\omega, \nu_\xi)$. Assume that $\phi, \Psi \in (S \longrightarrow B)$ specify what it is that π is intended to do: the execution of the program π starting with an entry state satisfying ϕ terminates and the exit state satisfies Ψ on termination of π. A *partial correctness proof* consists in showing that:

$$\nu_\omega \wedge post(\tau^*)(\nu_\epsilon \wedge \phi) \Rightarrow \Psi$$

In words, every exit state which is a descendant of an entry state satisfying ϕ must satisfy Ψ. The question of termination is not involved.

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

Theorem 10-4.

1. $((S \times S) \longrightarrow B)(\Longrightarrow, \lambda(s_1, s_2).false, \lambda(s_1, s_2).true, \vee, \wedge, \neg)$ and $(S \longrightarrow B)(\Longrightarrow, \lambda s.false, \lambda s.true, \vee, \wedge, \neg)$ are uniquely complemented complete lattices.
2. $\forall \theta \in ((S \times S) \longrightarrow B)$, $post(\theta)$ is a strict complete $\vee$-morphism. $\forall \beta \in (S \longrightarrow B)$, $\lambda\theta.[post(\theta)(\beta)]$ is a strict complete $\vee$-morphism.
3. $\forall \tau \in ((S \times S) \longrightarrow B)$, $\forall \beta \in (S \longrightarrow B)$,
$$post(\tau^*)(\beta) = \bigvee_{n \geq 0} post(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee post(\tau)(\alpha)])$$

Proof. The diagram of isotone functions shown in Fig. 10-3 is com-

$$\begin{tikzcd}
((S \times S) \rightarrow B) \arrow[r, "\lambda\alpha.[eq \vee \alpha \circ \tau]"] \arrow[d, "\lambda\theta.[post(\theta)(\beta)]"'] & ((S \times S) \rightarrow B) \arrow[d, "\lambda\theta.[post(\theta)(\beta)]"] \\
(S \rightarrow B) \arrow[r, "\lambda\alpha.[post(eq)(\beta) \vee post(\tau)(\alpha)]"'] & (S \rightarrow B)
\end{tikzcd}$$

Figure 10-3

muting and $\lambda\theta.[post(\theta)(\beta)]$ is a strict complete $\vee$-morphism. Therefore
$$\begin{aligned} post(lfp(\lambda\alpha.[eq \vee \alpha \circ \tau]))(\beta) &= post(\tau^*)(\beta) \\ &= lfp(\lambda\alpha.[post(eq)(\beta) \vee post(\tau)(\alpha)]) \\ &= lfp(\lambda\alpha.[\beta \vee post(\tau)(\alpha)]). \end{aligned}$$
Also $post(\tau^*)(\beta) = post(\bigvee_{n\geq 0} \tau^n)(\beta) = \bigvee_{n \geq 0} post(\tau^n)(\beta)$. ■

Example 10-5. Floyd [Floy67] and Naur's [Naur66] method of inductive assertions for proving the partial correctness of π with respect to ϕ, Ψ, consists in guessing an assertion ι and showing that $(((\nu_\epsilon \wedge \phi) \Longrightarrow \iota) \wedge (post(\tau)(\iota) \Longrightarrow \iota) \wedge ((\nu_\omega \wedge \iota) \Longrightarrow \Psi))$.

Using the recursion induction principle, from $(((\nu_\epsilon \wedge \phi) \Longrightarrow \iota) \wedge$

$(post(\tau)(\iota) \Rightarrow \iota))$ we infer $(lfp(\lambda\alpha.[(\nu_\epsilon \wedge \phi) \vee post(\tau)(\alpha)]) \Rightarrow \iota)$. It follows from Theorem 10-4, Part 3 that $(\nu_\omega \wedge post(\tau^*)(\nu_\epsilon \wedge \phi)) \Rightarrow (\nu_\omega \wedge \iota) \Rightarrow \Psi$. The method is sound [ClaE77].

Reciprocally, if π is partially correct with respect to ϕ, Ψ, then this can be proved using the Floyd-Naur method. This completeness result follows from the fact that one can choose ι as $lfp(\lambda\alpha.[\nu_\epsilon \wedge \phi) \vee post(\tau)(\alpha)])$.

10-4.4. Characterization of the Set of Ascendants 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* $\beta \in (S \longrightarrow B)$ is characterized by

$$\lambda s_1.[\exists\, s_2 \in S: \tau^*(s_1, s_2) \wedge \beta(s_2)] = pre(\tau^*)(\beta)$$

using the notation

$$pre \in (((S \times S) \longrightarrow B) \longrightarrow ((S \longrightarrow B) \longrightarrow (S \longrightarrow B)))$$
$$pre = \lambda\theta.[\lambda\beta.[\lambda s_1.[\exists\, s_2 \in S: \theta(s_1, s_2) \wedge \beta(s_2)]]]$$

Example 10-6. Let π be a program defining a total and deterministic system $(S, \tau, \nu_\epsilon, \nu_\omega, \nu_\xi)$ and $\phi, \Psi \in (S \longrightarrow B)$ be respectively entry and exit specifications. A *total correctness proof* consists in showing

$$\nu_\epsilon \wedge \phi \Rightarrow pre(\tau^*)(\nu_\omega \wedge \Psi)$$

In words, every entry state satisfying ϕ is the ascendant of an exit state satisfying Ψ. This is a proof of termination when $\Psi = \lambda s.[true]$.

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

Theorem 10-7.

1. $\forall \theta \in ((S \times S) \longrightarrow B)$, $pre(\theta)$ is a strict complete $\vee$-morphism; $\forall \beta \in (S \longrightarrow B)$, $\lambda\theta.pre(\theta)(\beta)$ is a strict complete $\vee$-morphism.
2. $\forall \tau \in ((S \times S) \longrightarrow B)$, $\forall \beta \in (S \longrightarrow B)$,
$$pre(\tau^*)(\beta) = \bigvee_{n \geq 0} pre(\tau^n)(\beta) = lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)])$$

Proof. $\forall \tau, \tau_1, \ldots \in ((S \times S) \longrightarrow B), (\tau_1 \circ \tau_2)^{-1} = (\tau_2^{-1} \circ \tau_1^{-1})$; $\forall n \in N$, $(\tau^n)^{-1} = (\tau^{-1})^n$; $(\bigvee_i \tau_i)^{-1} = \bigvee_i (\tau_i)^{-1}$, $(\tau^*)^{-1} = (\tau^{-1})^*$. Therefore it follows from Theorem 10-4 that $\forall \theta \in ((S \times S) \longrightarrow B), pre(\theta) =$

$post(\theta^{-1})$ is a strict complete $\vee$-morphism. $\forall \beta \in (S \longrightarrow S)$, $\lambda\theta.pre(\theta)(\beta) = \lambda\theta.post(\theta^{-1})(\beta)$ is a strict complete $\vee$-morphism. Also

$$pre(\tau^*)(\beta) = post((\tau^*)^{-1})(\beta) = post((\tau^{-1})^*)(\beta) = \bigvee_{n\geq 0} post((\tau^{-1})^n)(\beta)$$
$$= \bigvee_{n\geq 0} post((\tau^n)^{-1})(\beta) = \bigvee_{n\geq 0} pre(\tau^n)(\beta)$$
$$= lfp(\lambda\alpha.[\beta \vee post(\tau^{-1})(\alpha)]) = lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)]). \quad \blacksquare$$

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 $\pi(S, \tau, \nu_\epsilon, \nu_\omega, \nu_\xi)$ are those which do not lead to a run-time error. They are characterized by $\nu_\epsilon \wedge \neg pre(\tau^*)(\nu_\xi)$.

Theorem 10-8. Let $\tau \in ((S \times S) \longrightarrow B)$ be total and deterministic.

$$\forall \beta \in (S \longrightarrow B), \quad \neg pre(\tau^*)(\beta) = gfp(\lambda\alpha.[\neg\beta \wedge pre(\tau)(\alpha)])$$

Proof. $\neg pre(\tau^*)(\beta) = \neg lfp(\lambda\alpha.[\beta \vee pre(\tau)(\alpha)]) = \neg lfp(\neg\lambda\alpha.[\neg\beta \wedge pre(\tau)(\neg(\neg\alpha))])$. According to Park's Fixed Point Theorem, this is equal to $gfp(\lambda\alpha.[\neg\beta \wedge \neg pre(\tau)(\neg\alpha)])$. Let $\bar{\tau} \in (S \longrightarrow S)$ be such that $(\forall s_1, s_2 \in S, (\tau(s_1, s_2) \longleftrightarrow (\bar{\tau}(s_1) = s_2)))$. We have $\neg pre(\tau)(\neg\alpha) = \lambda s_1.[\neg\neg\alpha(\bar{\tau}(s_1))] = \lambda s_1.[\alpha(\bar{\tau}(s_1))] = pre(\tau)(\alpha)$. $\blacksquare$

10-4.6. Analysis of the Behavior of a Total Deterministic Discrete Dynamic System

Given a total deterministic system $\pi(S, \tau, \nu_\epsilon, \nu_\sigma, \nu_\xi)$ 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 $\phi \in (S \longrightarrow B)$ is characterized by:
$$post(\tau^*)(\nu_\epsilon \wedge \phi) = lfp(\lambda\alpha.[(\nu_\epsilon \wedge \phi) \vee post(\tau)(\alpha)])$$
2. The set of ascendants of the exit states satisfying an exit condition $\Psi \in (S \longrightarrow B)$ is characterized by:
$$pre(\tau^*)(\nu_\omega \wedge \Psi) = lfp(\lambda\alpha.[(\nu_\omega \wedge \Psi) \vee pre(\tau)(\alpha)])$$
3. The set of states leading to an error is characterized by:
$$pre(\tau^*)(\nu_\xi) = lfp(\lambda\alpha.[\nu_\xi \vee pre(\tau)(\alpha)])$$

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:

$$\neg pre(\tau^*)(v_\xi) = gfp(\lambda\alpha.[\neg v_\xi \wedge pre(\tau)(\alpha)])$$

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

$$\neg pre(\tau^*)(v_\omega \vee v_\xi) = gfp(\lambda\alpha.[\neg v_\omega \wedge \neg v_\xi \wedge pre(\tau)(\alpha)])$$

Example 10-10. The proof that a program $\pi(S, \tau, v_\epsilon, v_\sigma, v_\xi)$ does not terminate for the entry states satisfying a condition $\delta \in (S \longrightarrow B)$ consists in proving that $v_\epsilon \wedge \delta \Longrightarrow \neg pre(\tau^*)(v_\omega \vee v_\xi)$. It follows from Theorem 10-9(5) and the Dual Recursion Induction Principle that this can be done by guessing an assertion $\iota \in (S \longrightarrow B)$ and proving that $(((v_\epsilon \wedge \delta) \Longrightarrow \iota) \wedge (\iota \Longrightarrow \neg v_\omega \wedge \neg v_\xi \wedge pre(\tau)(\iota)))$.

10-4.7. Relationships Between *pre* and *post*

Theorem 10-11. Let $\theta \in ((S \times S) \longrightarrow B)$. $\forall \beta, \gamma \in (S \longrightarrow B)$,

1. $pre(\theta)(\beta) = post(\theta^{-1})(\beta), post(\theta)(\beta) = pre(\theta^{-1})(\beta)$
2. If θ is deterministic, then:
$$post(\theta)(pre(\theta)(\beta)) = (\beta \wedge post(\theta)(true)) \Longrightarrow \beta$$
3. If θ is total, then:
$$\beta \Longrightarrow pre(\theta)(post(\theta)(\beta))$$
4. If θ is total and deterministic, then:
$$(\beta \Longrightarrow pre(\theta)(\gamma)) \text{ iff } (post(\theta)(\beta) \Longrightarrow \gamma)$$
$$post(\theta)(\beta) = \bigwedge\{\gamma \in (S \longrightarrow B): \beta \Longrightarrow pre(\theta)(\gamma)\}$$
$$pre(\theta)(\beta) = \bigvee\{\gamma \in (S \longrightarrow B): post(\theta)(\gamma) \Longrightarrow \beta\}$$

Proof.

1. $pre(\theta)(\beta) = \lambda s_1.[\exists s_2: \theta(s_1, s_2) \wedge \beta(s_2)] = \lambda s_1.[\exists s_2: \beta(s_2) \wedge \theta^{-1}(s_2, s_1)] = post(\theta^{-1})(\beta)$. $post(\theta)(\beta) = post((\theta^{-1})^{-1})(\beta) = pre(\theta^{-1})(\beta)$.
2. If θ is deterministic, then there exists $\bar{\theta} \in (S \longrightarrow S)$ such that $\theta(s_1, s_2) \Longleftrightarrow s_2 = \bar{\theta}(s_1)$. Therefore $post(\theta)(pre(\theta)(\beta)) = \lambda s_3.[\exists s_1: (\exists s_2: s_2 = \bar{\theta}(s_1) \wedge \beta(s_2)) \wedge s_3 = \bar{\theta}(s_1)] = \lambda s_3.[\exists s_1: \beta(\bar{\theta}(s_1)) \wedge s_3 = \bar{\theta}(s_1)] = post(\theta)(true) \wedge \beta$.
3. If θ is total, then $\forall s_3 \in S, \beta(s_3) \Longrightarrow (\beta(s_3) \wedge (\exists s_2: \theta(s_3, s_2)) \Longrightarrow (\exists s_2: \theta(s_3, s_2) \wedge (\exists s_1: \theta(s_1, s_2) \wedge \beta(s_1))) = pre(\theta)(post(\theta)(\beta))(s_3)$.

4. If $(\beta \Rightarrow pre(\theta)(\gamma))$, then by isotony $post(\theta)(\beta) \Rightarrow post(\theta)(pre(\theta)(\gamma)) \Rightarrow \gamma$. If $(post(\theta)(\beta) \Rightarrow \gamma)$, then by isotony $\beta \Rightarrow pre(\theta)(post(\theta)(\beta)) \Rightarrow pre(\theta)(\gamma)$. $post(\theta)(\beta) = \bigwedge\{\gamma : post(\theta)(\beta) \Rightarrow \gamma\} = \bigwedge\{\gamma : \beta \Rightarrow pre(\theta)(\gamma)\}$. $pre(\theta)(\beta) = \bigvee\{\gamma : \gamma \Rightarrow pre(\theta)(\beta)\} = \bigvee\{\gamma : post(\theta)(\gamma) \Rightarrow \beta\}$. ■

Example 10-12. According to Theorem 10-11.4, Floyd-Naur's method for proving the partial correctness of π with respect to ϕ, Ψ which consists in guessing an assertion $\imath$ and showing that $(((v_\epsilon \wedge \phi) \Rightarrow \imath) \wedge (post(\tau)(\imath) \Rightarrow \imath) \wedge ((v_\omega \wedge \imath) \Rightarrow \Psi))$ is equivalent to Hoare's method [Hoar69], which consists of guessing an assertion $\imath$ and showing that $(((v_\epsilon \wedge \phi) \wedge (\imath \Rightarrow pre(\tau)(\imath)) \wedge ((v_\omega \wedge \imath) \Rightarrow \Psi))$.

We have seen that the analysis of a system consists of solving "forward" fixpoint equations of the form $\alpha = \beta \between post(\tau)(\alpha)$ or "backward" fixpoint equations of the form $\alpha = \beta \between pre(\tau)(\alpha)$ (where $\beta \in (S \rightarrow B)$ and $\between$ is either $\vee$ or $\wedge$). In fact whenever a forward equation is needed, a backward equation can be used instead, and vice versa.

Theorem 10-13.

$$\forall \theta \in ((S \times S) \rightarrow B),\ \forall \beta \in (S \rightarrow B),$$
$$post(\theta)(\beta) = \lambda\bar{s}.[\exists s_1 \in S: \beta(s_1) \wedge pre(\theta)(\lambda s.[s = \bar{s}])(s_1)]$$
$$pre(\theta)(\beta) = \lambda\bar{s}.[\exists s_2 \in S: post(\theta)(\lambda s.[s = \bar{s}])(s_2) \wedge \beta(s_2)]$$

Proof. $post(\theta)(\beta) = \lambda\bar{s}.[\exists s_1 \in S: \beta(s_1) \wedge \theta(s_1, \bar{s})] = \lambda\bar{s}.[\exists s_1 \in S: \beta(s_1) \wedge (\exists s \in S: (s = \bar{s}) \wedge \theta(s_1, s))] = \lambda\bar{s}.[\exists s_1 \in S: \beta(s_1) \wedge pre(\theta)(\lambda s.[s = \bar{s}])(s_1)]$. ■

Example 10-14. A total correctness proof of a program π with respect to ϕ, Ψ consists in showing that $((v_\epsilon \wedge \phi) \Rightarrow pre(\tau^*)(v_\omega \wedge \Psi))$, that is to say $((v_\epsilon \wedge \phi) \Rightarrow lfp(\lambda\alpha.[(v_\omega \wedge \Psi) \vee pre(\tau)(\alpha)]))$. Equivalently, using *post*, one can show that: $\forall \bar{s} \in S$, $(v_\epsilon(\bar{s}) \wedge \phi(\bar{s})) \Rightarrow (\exists s_2 \in S: v_\omega(s_2) \wedge \Psi(s_2) \wedge lfp(\lambda\alpha.[\lambda s.(s = \bar{s}) \vee post(\tau)(\alpha)])(s_2))$ More generally we have:

$$post(\tau^*)(\beta) = \lambda\bar{s}.[\exists s_1 \in S: \beta(s_1) \wedge lfp(\lambda\alpha.[\lambda s.(s = \bar{s}) \vee pre(\tau)(\alpha)])(s_1)]$$
$$pre(\tau^*)(\beta) = \lambda\bar{s}.[\exists s_2 \in S: \beta(s_2) \wedge lfp(\lambda\alpha.[\lambda s.(s = \bar{s}) \vee post(\tau)(\alpha)])(s_2)]$$

10-4.8. Partitioned Dynamic Discrete Systems

A dynamic discrete system $(S, \tau, v_\epsilon, v_\sigma, v_\xi)$ is said to be *partitioned* if there exist $n \geq 1, U_1, \ldots, U_n, \imath_1, \ldots, \imath_n$ such that $\forall i \in [1, n]$, $\imath_i$ is a partial one-to-one map from S onto U_i and $\{\imath_i^{-1}(U_i): i \in [1, n]\}$ is a partition of S, (therefore $S = \bigcup_{i=1}^{n} \imath_i^{-1}(U_i)$ and every $s \in S$ is an element of exactly one $\imath_i^{-1}(U_i)$).

When studying the behavior of a partitioned system, the equations $\alpha = \beta \bowtie post(\tau)(\alpha)$ or $\alpha = \beta \bowtie pre(\tau)(\alpha)$ can be replaced by systems of equations defined as follows. Let us define: $\forall i \in [1, n], \sigma_i \in ((S \longrightarrow B) \longrightarrow (U_i \longrightarrow B)), \sigma_i = \lambda\beta.[\beta \circ \iota_i^{-1}]$, $\sigma_i^{-1} = \lambda\beta.[\lambda s.[s \in \iota_i^{-1}(U_i) \wedge \beta(\iota_i(s))]]$, $\sigma \in ((S \longrightarrow B) \longrightarrow \left(\prod_{i=1}^{n} (U_i \longrightarrow B)\right)$, $\sigma = \lambda\beta.\left(\prod_{i=1}^{n} \sigma_i(\beta)\right) = \lambda\beta.\langle\sigma_1(\beta), \ldots, \sigma_n(\beta)\rangle$. σ is a strict isomorphism from $(S \longrightarrow B)$ onto $\prod_{i=1}^{n} (U_i \longrightarrow B)$. Its inverse is $\sigma^{-1} = \lambda\langle\beta_1, \ldots, \beta_n\rangle.\left[\bigvee_{i=1}^{n} \sigma_i^{-1}(\beta_i)\right]$.

For any isotone operator f on $(S \longrightarrow B)$, the diagram in Fig. 10-4 commutes, so that the sets of pre-fixed points, fixed points and post-fixed

$$\begin{tikzcd}
(S \to B) \arrow[rrr, "f"] \arrow[d, shift right, "\sigma"'] & & & (S \to B) \arrow[d, shift right, "\sigma"'] \\
\prod_{i=1}^{n} (U_i \to B) \arrow[u, shift right, "\sigma^{-1}"'] \arrow[rrr, "\lambda\langle\alpha_1, \ldots, \alpha_n\rangle.[\sigma \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n)]"'] & & & \prod_{i=1}^{n} (U_i \to B) \arrow[u, shift right, "\sigma^{-1}"']
\end{tikzcd}$$

Figure 10-4

points of f coincide (up to the isomorphism σ) with the pre-solutions, solutions and post-solutions to the *direct decomposition* of $\alpha = f(\alpha)$ on $\prod_{i=1}^{n} (U_i \longrightarrow B)$ which is the system of equations:

$$\alpha_1 = \sigma_1 \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n)$$
$$\vdots$$
$$\alpha_n = \sigma_n \circ f \circ \sigma^{-1}(\alpha_1, \ldots, \alpha_n)$$

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

Theorem 10-15. $\forall i \in [1, n], \sigma_i \circ \lambda\alpha.[\beta \bowtie post(\tau)(\alpha)] \circ \sigma^{-1}$ is equal to

$$\lambda\langle\alpha_1, \ldots, \alpha_n\rangle.\left[\sigma_i(\beta) \bowtie \left(\bigvee_{j \in pred_\tau(i)} post(\tau_{ji})(\alpha_j)\right)\right]$$

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

$$\lambda\langle\alpha_1, \ldots, \alpha_n\rangle.\left[\sigma_i(\beta) \bowtie \left(\bigvee_{j \in succ_\tau(i)} pre(\tau_{ij})(\alpha_j)\right)\right]$$

where

$$\tau_{ij} \in ((U_i \times U_j) \longrightarrow B), \tau_{ij} = \lambda\langle s_1, s_2\rangle.[\tau(\iota_i^{-1}(s_1), \iota_j^{-1}(s_2))]$$
$$pred_\tau = \lambda i.\{j \in [1, n]: (\exists s_1 \in U_j, \exists s_2 \in U_i: \tau_{ji}(s_1, s_2))\}$$
$$succ_\tau = \lambda i.\{j \in [1, n]: (\exists s_1 \in U_j, \exists s_2 \in U_i: \tau_{ij}(s_1, s_2))\}$$

Proof.

$$\sigma_i(\beta \between post(\tau)(\sigma^{-1}(\alpha_1, \ldots, \alpha_n))) = \sigma_i(\beta) \between \sigma_i\Big(post(\tau)\Big(\bigvee_{j=1}^{n} \sigma_j^{-1}(\alpha_j)\Big)\Big)$$
$$= \sigma_i(\beta) \between \bigvee_{j=1}^{n} (post(\tau)(\sigma_j^{-1}(\alpha_j)) \circ \iota_i^{-1}).$$

Moreover

$$\begin{aligned} post(\tau)(\sigma_j^{-1}(\alpha_j)) \circ \iota_i^{-1} &= \lambda s_2.[\exists s_1 \in S: \sigma_j^{-1}(\alpha_j)(s_1) \wedge \tau(s_1, \iota_i^{-1}(s_2))] \\ &= \lambda s_2.[\exists s_1 \in U_j: \alpha_j(s_1) \wedge \tau(\iota_j^{-1}(s_1), \iota_i^{-1}(s_2))] \\ &= \lambda s_2.[\exists s_1 \in U_j: \alpha_j(s_1) \wedge \tau_{ji}(s_1, s_2)] \\ &= post(\tau_{ji})(\alpha_j). \end{aligned}$$

Therefore $\bigvee_{j=1}^{n} (post(\tau)(\sigma_j^{-1}(\alpha_j)) \circ \iota_i^{-1}) = \bigvee_{j \in pred_\tau(i)} post(\tau_{ji})(\alpha_j)$ since $(j \notin pred_\tau(i))$ implies $\forall s_1, s_2, \neg\tau_{ji}(s_1, s_2)$. Also $pre(\tau) = post(\tau^{-1})$, $(\tau_{ij})^{-1} = \tau_{ji}$ and $succ_\tau = pred_{\tau^{-1}}$. ■

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 $V = [1, n] - \{\xi\}$ defines a partitioned discrete dynamic system $\langle \tau, S, \nu_\epsilon, \nu_\omega, \nu_\xi\rangle$ where $S = ([1, n] \times U)$, $\forall i \in [1, n]$, $U_i = U$, $\iota_i = \lambda\langle c, m\rangle.m$, $\iota_i^{-1} = \lambda m.\langle i, m\rangle$. 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 $P = F_\pi(\phi)(P)$ associated with a program π and an entry specification $\phi \in (U \longrightarrow B)$ is the direct decomposition of $\alpha = (\nu_\epsilon \wedge \sigma_\epsilon^{-1}(\phi)) \vee post(\tau)(\alpha)$ on $(U \longrightarrow B)^n$; that is,

$$P_i = \sigma_i(\nu_\epsilon \wedge \sigma_\epsilon^{-1}(\phi)) \vee \Big(\bigvee_{j \in pred_\tau(i)} post(\tau_{ji})(P_j)\Big) \qquad i = 1, \ldots, n$$

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 $pred_\tau(\epsilon) = \phi$; therefore $P_\epsilon = \sigma_\epsilon(v_\epsilon \wedge \sigma_\epsilon^{-1}(\phi)) = \sigma_\epsilon(\lambda\langle c, m\rangle.((c = \epsilon) \wedge \phi(m))) = \phi$. Otherwise $i \neq \epsilon$, in which case $\sigma_i(v_\epsilon \wedge \sigma_\epsilon^{-1}(\phi)) = \lambda m.false$ and

$$\begin{aligned} P_i &= \bigvee_{j \in pred_\tau(i)} post(\tau_{ji})(P_j) \\ &= \bigvee_{j \in pred_\tau(i)} \lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge (\bar{\tau}(\langle j, m_1\rangle) = \langle i, m_2\rangle)] \end{aligned}$$

When $i \neq \epsilon$ and $i \neq \xi$, notice that $pred_\tau(i)$ is contained in the set of origins of the edges entering i, that is, the set $pred_\pi(i)$ of predecessors of the vertex i in the program graph G of π. The expression $\lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge (\bar{\tau}(\langle j, m_1\rangle) = \langle i, m_2\rangle)]$ 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 $\vec{v} = f(\vec{v})$, then

$$\begin{aligned} &\lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge (\bar{\tau}(\langle j, m_1\rangle) = \langle i, m_2\rangle)] \\ &\quad = \lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge m_1 \in dom(f) \wedge (m_2 = f(m_1))] \end{aligned}$$

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

$$\begin{aligned} &\lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge (\tau(\langle j, m_1\rangle) = \langle i, m_2\rangle)] \\ &\quad = \lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge (m_1 \in dom(p)) \wedge p(m_1) \\ &\qquad \wedge (m_1 = m_2)] \\ &\quad = \lambda m_2.[P_j(m_2) \wedge (m_2 \in dom(p)) \wedge p(m_2)] \end{aligned}$$

4. If $i = \xi$, then

$$\begin{aligned} P_\xi &= \bigvee_{j \in pred_\tau(\xi)} \lambda m_2.[\exists\, m_1 \in U: P_j(m_1) \wedge (\bar{\tau}(\langle j, m_1\rangle) = \langle \xi, m_2\rangle)] \\ &= P_\xi \vee \bigvee_{j \in at(\pi)} \lambda m_2.[P_j(m_2) \wedge (m_2 \notin dom(expr(j)))] \end{aligned}$$

where $at(\pi)$ is the set of program points j preceding an assignment $\vec{v} = f(\vec{v})$ or a test $p(\vec{v})$ and $expr(j)$ 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 π and an entry specification $\phi \in (U \longrightarrow B)$ is*

$$\begin{aligned} P_\epsilon &= \phi \\ P_i &= \bigvee_{j \in pred_\pi(i)} post(L(\langle j, i\rangle))(P_j) \qquad i \in ([1, n] - \{\epsilon, \xi\}) \\ P_\xi &= \Big(\bigvee_{j \in at(\pi)} \lambda m.[P_j(m) \wedge m \notin dom(expr(j))]\Big) \vee P_\xi \end{aligned}$$

where $\forall f \in I_a(U)$, $post(f) = \lambda P.[\lambda m.[\exists m' \in U: P(m') \wedge m' \in dom(f) \wedge m = f(m')]]$; $\forall p \in I_t(U), post(p) = \lambda P.[\lambda m.[P(m) \wedge m \in dom(p) \wedge p(m)]]$; $at(\pi)$ is the set of program points j preceding an assignment $\vec{v} = f(\vec{v})$ or a test $p(\vec{v})$ and $expr(j)$ is the corresponding f or p.

Theorem 10-17. The system of forward semantic equations $P = F_\pi(\phi)(P)$ associated with a program π and an entry specification $\phi \in (U \longrightarrow B)$ is the direct decomposition of $\alpha = (\nu_\epsilon \wedge \sigma_\epsilon^{-1}(\phi)) \vee post(\tau)(\alpha)$ on $(U \longrightarrow B)^n$.

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 π the systems of equations which are the direct decomposition of backward equations of type $\alpha = \beta \between pre(\tau)(\alpha)$ on $(U \longrightarrow B)^n$; that is,

$$P_i = \sigma_i(\beta) \between \bigvee_{j \in succ_\tau(i)} \lambda m_1.[\exists m_2 \in U: (\bar{\tau}(\langle m_1, i\rangle) = \langle m_2, j\rangle) \wedge P_j(m_2)] \qquad i = 1, \ldots, n$$

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

Definition 10-18. The *system of backward semantic equations* $P = B_\pi(\Psi)(P)$ *associated with a program* π *and an exit specification* $\Psi \in (U \longrightarrow B)$ is

$$P_i = \bigvee_{j \in succ_\pi(i)} pre(L(\langle i, j\rangle))(P_j) \qquad i \in ([1, n] - \{\omega, \xi\})$$

$$P_\sigma = \Psi$$

where $\forall f \in I_a(U), pre(f) = \lambda P.[\lambda m.[m \in dom(f) \wedge P(f(m))]]$; $\forall p \in I_t(U)$, $pre(p) = \lambda P.[\lambda m.[m \in dom(f) \wedge p(m) \wedge P(m)]]$; and $succ_\pi(i)$ is the set of successors of the vertex i in the program graph of π.

Theorem 10-19.

1. The direct decomposition $P = B(P)$ of $\alpha = (\nu_\omega \wedge \sigma_\omega^{-1}(\Psi)) \vee pre(\tau)(\alpha)$ on $(U \longrightarrow B)^n$ is

$$P_i = B_\pi(\Psi)_i(P) \vee error(i) \qquad \text{for } i \in ([1, n] - \{\xi, \omega\})$$

$$P_\omega = \Psi \vee P_\omega$$

$$P_\xi = P_\xi$$

where $error(i) = \lambda m \in U.[P_\xi(m) \wedge i \in at(\pi) \wedge m \notin dom(expr(i))]$; $\forall i \in ([1, n] - \{\xi\}), lfp(B)_i = lfp(B_\pi(\Psi))_i$; and $lfp(B)_\xi = \lambda m.[false]$.

2. The direct decomposition $P = B(P)$ of $\alpha = \neg v_\xi \wedge pre(\tau)(\alpha)$ on $(U \longrightarrow B)^n$ is

$$P_i = B_\pi(\lambda m.[true])_i(P) \qquad \text{for } i \in ([1, n] - \{\omega, \xi\})$$
$$P_\omega = P_\omega$$
$$P_\xi = \lambda m.[false]$$

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

3. The direct decomposition $P = B(P)$ of $\alpha = v_\xi \vee pre(\tau)(\alpha)$ on $(U \longrightarrow B)^n$ is

$$P_i = B_\pi(\lambda m.[false])_i(P) \vee error(i) \quad \text{for } i \in ([1, n] - \{\omega, \xi\})$$
$$P_\omega = P_\omega$$
$$P_\xi = \lambda m.[true]$$

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

$$P_i = B_\pi(\lambda m.[false])_i(Q) \vee \lambda m.[m \notin dom(expr(i))] \qquad \text{for } i \in ([1, n] - \{\omega, \xi\})$$
$$P_\omega = \lambda m.[false]$$
$$P_\xi = \lambda m.[true]$$

where Q_i stands for P_i when $i \in ([1, n] - \{\omega, \xi\})$, Q_ω stands for $\lambda m.[false]$ and Q_ξ stands for $\lambda m.[true]$.

4. The direct decomposition of $\alpha = \neg v_\omega \wedge \neg v_\xi \wedge pre(\tau)(\alpha)$ on $(U \longrightarrow B)^n$ is

$$P_i = B_\pi(\lambda m.[false])_i(P) \qquad \text{for } i \in ([1, n] - \{\xi\})$$
$$P_\xi = \lambda m.[false]$$

5. The direct decomposition of $\alpha = \lambda s.[s = \bar{s}] \vee pre(\tau)(\alpha)$ on $(U \longrightarrow B)^n$ is

$$P_i = \lambda m.[\langle i, m\rangle = \bar{s}] \vee B_\pi(\lambda m.[false])_i(P) \vee error(i) \qquad \text{for } i \in ([1, n] - \{\omega, \xi\})$$
$$P_\omega = \lambda m.[\langle \sigma, m\rangle = \bar{s}] \vee P_\omega$$
$$P_\xi = \lambda m.[\langle \xi, m\rangle = \bar{s}] \vee P_\xi$$

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 π:

```
{1}
    while x ≥ 1000 do
{2}
        x := x + y;
{3}
    od;
{4}
```

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

10-5.3.1. Forward semantic analysis

The system $P = F_\pi(\phi)(P)$ (where $F_\pi(\phi) \in ((I^2 \longrightarrow B)^5 \longrightarrow (I^2 \longrightarrow B)^5))$ of forward semantic equations associated with the above program π and an entry specification $\phi \in (I^2 \longrightarrow B)$ is the following:

$$P_1 = \phi$$
$$P_2 = \lambda\langle x, y\rangle.[(P_1 \vee P_3)(x, y) \wedge (x \in I) \wedge (x \geq 1000)]$$
$$P_3 = \lambda\langle x, y\rangle.[\exists\, x' \in I: P_2(x', y) \wedge ((x' + y) \in I) \wedge (x = x' + y)]$$
$$P_4 = \lambda\langle x, y\rangle.[(P_1 \vee P_3)(x, y) \wedge (x \in I) \wedge (x < 1000)]$$
$$P_\xi = \lambda\langle x, y\rangle.[((P_1 \vee P_3)(x, y) \wedge (x \notin I)) \vee (P_2(x, y) \wedge ((x + y) \notin I))]$$

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

$$\sigma_\epsilon(v_\epsilon \wedge pre(\tau^*)(v_\omega))$$
$$= \sigma_\epsilon(\lambda\bar{s}.[\exists\, s_2 \in S: v_\omega(s_2) \wedge post(\tau^*)(\lambda s.[s = \bar{s}])(s_2)]) \qquad \text{(Theorem 10-13)}$$
$$= \lambda\bar{m}.[\exists\, s_2 \in S: v_\omega(s_2) \wedge post(\tau^*)(\lambda s.[s = \langle\epsilon, \bar{m}\rangle])(s_2)])$$
$$= \lambda\bar{m}.[\exists\, s_2 \in S: v_\omega(s_2) \wedge lfp(\lambda\alpha.[(v_\epsilon \wedge \sigma_\epsilon^{-1}(\lambda m.[m = \bar{m}])) \vee post(\tau)(\alpha)])(s_2)] \qquad \text{(Theorem 10-4(3))}$$
$$= \lambda\bar{m}.[\exists\, s_2 \in S: v_\omega(s_2) \wedge \sigma^{-1}(lfp(F_\pi(\lambda m.[m = \bar{m}])))(s_2)] \qquad \text{(Theorem 10-17)}$$
$$= \lambda\bar{m}.\left[\exists\, s_2 \in S: v_\omega(s_2) \wedge \left(\bigvee_{i=1}^{n} lfp(F_\pi(\lambda m.[m = \bar{m}]))_i(\iota_i(s_2))\right)\right]$$
$$= \lambda\bar{m}.\left[\bigvee_{i=1}^{n} (\exists\, m_2 \in U_i: v_\omega(\iota_i^{-1}(m_2)) \wedge lfp(F_\pi(\lambda m.[m = \bar{m}]))_i(m_2))\right]$$
$$= \lambda\bar{m}.[\exists\, m_2 \in U_\omega: lfp(F_\pi(\lambda m.[m = \bar{m}]))_\omega(m_2)]$$

The least fixed point P^∞ of $F_\pi(\lambda\langle x, y\rangle.[(x = \bar{x}) \wedge (y = \bar{y})])$ is computed iteratively using a chaotic iteration sequence as follows:

$$P_i^0 = \lambda\langle x, y\rangle.[\mathit{false}] \qquad i = 1, \ldots, 4, \xi$$

$$P_1^1 = \lambda\langle x, y\rangle.[(x = \bar{x}) \wedge (y = \bar{y})] \qquad \text{where } \langle \bar{x}, \bar{y}\rangle \in I^2$$

$$P_2^1 = \lambda\langle x, y\rangle.[(P_1^1 \vee P_3^0)(x, y) \wedge (x \in I) \wedge (x \geq 1000)]$$
$$= \lambda\langle x, y\rangle.[(\bar{x} \in I \wedge 1000 \leq \bar{x}) \wedge (x = \bar{x}) \wedge (y = \bar{y})]$$

$$P_3^1 = \lambda\langle x, y\rangle.[\exists\, x' \in I: P_2^1(x', y) \wedge ((x' + y) \in I) \wedge (x = x' + y)$$
$$= \lambda\langle x, y\rangle.[(\bar{x} \in I \wedge (\bar{x} + \bar{y}) \in I \wedge 1000 \leq \bar{x}) \wedge (x = \bar{x} + \bar{y}) \wedge (y = \bar{y})]$$

$$P_2^2 = \lambda\langle x, y\rangle.[(P_1^1 \vee P_3^1)(x, y) \wedge (x \in I) \wedge (x \geq 1000)]$$
$$= \lambda\langle x, y\rangle.[((\bar{x} \in I \wedge 1000 \leq \bar{x}) \wedge (x = \bar{x}) \wedge (y = \bar{y}))$$
$$\vee ((\bar{x} \in I \wedge (\bar{x} + \bar{y}) \in I \wedge 1000 \leq \bar{x} \wedge 1000 \leq (\bar{x} + \bar{y}))$$
$$\wedge (x = \bar{x} + \bar{y}) \wedge (y = \bar{y}))]$$

Assume as induction hypothesis that

$$P_2^k = \lambda\langle x, y\rangle.[\exists j \in [0, k-1]: \bigwedge_{i=0}^{j} ((\bar{x} + i\bar{y}) \in I \wedge 1000 \leq (\bar{x} + i\bar{y}))$$
$$\wedge (x = \bar{x} + j\bar{y}) \wedge (y = \bar{y})]$$

then

$$P_3^k = \lambda\langle x, y\rangle.[\exists\, x' \in I: P_2^k(x', y) \wedge ((x' + y) \in I) \wedge (x = x' + y)]$$
$$= \lambda\langle x, y\rangle.[\exists j \in [1, k]: \bigwedge_{i=0}^{j-1} ((\bar{x} + i\bar{y}) \in I \wedge 1000 \leq (\bar{x} + i\bar{y}))$$
$$\wedge ((\bar{x} + j\bar{y}) \in I) \wedge (x = \bar{x} + j\bar{y}) \wedge (y = \bar{y})]$$

$$P_2^{k+1} = \lambda\langle x, y\rangle.[(P_1^1 \vee P_3^k)(x, y) \wedge (x \in I) \wedge (x \geq 1000)$$
$$= \lambda\langle x, y\rangle.[\exists j \in [0, k]: \bigwedge_{i=0}^{j} ((\bar{x} + i\bar{y}) \in I \wedge 1000 \leq (\bar{x} + i\bar{y}))$$
$$\wedge (x = \bar{x} + j\bar{y}) \wedge (y = \bar{y})]$$

proving by induction on k that P_2^k is of the form assumed in the induction hypothesis. Then passing to the limit,

$$P_2^\infty = \bigvee_{k \geq 0} P_2^k$$
$$= \lambda\langle x, y\rangle.[\exists j \geq 0: \left[\bigwedge_{i=0}^{j} ((\bar{x} + i\bar{y}) \in I \wedge 1000 \leq (\bar{x} + i\bar{y}))\right.$$
$$\left.\wedge (x = \bar{x} + j\bar{y}) \wedge (y = \bar{y})\right]$$
$$= \lambda\langle x, y\rangle.[\exists j \geq 0: (1000 \leq \mathit{min}(\bar{x}, x)) \wedge (\mathit{max}(\bar{x}, x) \leq b)$$
$$\wedge (x = \bar{x} + j\bar{y}) \wedge (y = \bar{y})]$$

(It is worth noting that the use of the symbolic entry condition $\lambda\langle x, y\rangle.[(x = \bar{x}) \wedge (y = \bar{y})]$ 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 $lfp(F_\pi(\lambda\langle x, y\rangle.[(x = \bar{x}) \wedge (y = \bar{y})]))$ are:

$$P_1^\omega = \lambda\langle x, y\rangle.[(x = \bar{x}) \wedge (y = \bar{y})]$$

$$\begin{aligned}P_3^\omega &= \lambda\langle x, y\rangle.[\exists\, x' \in I: P_2^\omega(x', y) \wedge ((x' + y) \in I) \wedge (x = x' + y)]\\ &= \lambda\langle x, y\rangle.[\exists\, j \geq 1: (1000 \leq min(\bar{x}, x - \bar{y})) \wedge (max(\bar{x}, x) \leq b)\\ &\qquad\qquad \wedge (x = \bar{x} + j\bar{y}) \wedge (y = \bar{y})]\end{aligned}$$

$$\begin{aligned}P_4^\omega &= \lambda\langle x, y\rangle.[(P_1^\omega \vee P_3^\omega)(x, y) \wedge (x \in I) \wedge (x < 1000)]\\ &= \lambda\langle x, y\rangle.[((\bar{x} < 1000) \wedge (x = \bar{x}) \wedge (x = \bar{y}))\\ &\qquad \vee ((\bar{x} \geq 1000) \wedge (\bar{y} < 0)\\ &\qquad \wedge (x = \bar{x} + (((\bar{x} - 1000)\, div\, |\bar{y}|) + 1)\bar{y}) \wedge (y = \bar{y}))]\end{aligned}$$

$$\begin{aligned}P_\xi^\omega &= \lambda\langle x, y\rangle.[((P_1^\omega \vee P_3^\omega)(x, y) \wedge (x \notin I)) \vee (P_2^\omega(x, y) \wedge ((x + y) \notin I))]\\ &\qquad \vee P_\xi^0\\ &= \lambda\langle x, y\rangle.[(\bar{x} \geq 1000) \wedge (\bar{y} > 0) \wedge (x = \bar{x} + ((b - \bar{x})div\, \bar{y})\bar{y})\\ &\qquad \wedge (y = \bar{y})]\end{aligned}$$

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

$$\lambda\langle \bar{x}, \bar{y}\rangle.[\exists\, x_2, y_2 \in I: P_4^\omega(x_2, y_2)] = \lambda\langle x, y\rangle.[(\bar{x} < 1000) \vee (\bar{y} < 0)]$$

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

$$\sigma_\epsilon(v_\epsilon \wedge pre(\tau^*)(v_\xi)) = \lambda\bar{m}.[\exists\, m_2 \in U_\xi: lfp(F_\pi(\lambda m.[m = \bar{m}]))_\xi(m_2)]$$

that is,

$$\lambda\langle \bar{x}, \bar{y}\rangle.[\exists\, x_2, y_2 \in I: P_\xi^\omega(x_2, y_2)] = \lambda\langle \bar{x}, \bar{y}\rangle.[(\bar{x} \geq 1000) \wedge (\bar{y} > 0)]$$

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

$$\begin{aligned}&\sigma_\epsilon(v_\epsilon \wedge \neg pre(\tau^*)(v_\omega \vee v_\xi))\\ &\quad = \lambda\bar{m}.\neg\Big[\bigvee_{i=1}^{n}(\exists\, m_2 \in U_i: (v_\omega \vee v_\xi)(\langle i, m_2\rangle)\\ &\qquad\qquad \wedge lfp(F_\pi(\lambda m.[m = \bar{m}]))_i(m_2))\Big]\\ &\quad = \lambda\bar{m}.[\neg(\exists\, m_2 \in U_\omega: lfp(F_\pi(\lambda m.[m = \bar{m}]))_\omega(m_2))\\ &\qquad\qquad \wedge \neg(\exists\, m_2 \in U_\xi: lfp(F_\pi(\lambda m.[m = \bar{m}]))_\xi(m_2))]\end{aligned}$$

that is,

$$\lambda\langle\bar{x},\bar{y}\rangle.[\neg(\exists\, x_2, y_2 \in I: P_4^\infty(x_2, y_2)) \wedge \neg(\exists\, x_2, y_2 \in I: P_\xi^\infty(x_2, y_2))] = \lambda\langle\bar{x},\bar{y}\rangle.[(\bar{x} \geq 1000) \wedge (y = 0)]$$

The *set of descendants of the entry states satisfying the entry condition* $\phi \in (I^2 \longrightarrow B)$ is characterized by $post(\tau^*)(v_\epsilon \wedge \sigma_\epsilon^{-1}(\phi))$; that is (Theorem 10-9 and Theorem 10-17) up to the isomorphism σ by $Q^\infty = lfp(F_\pi(\phi))$:

$$Q_1^\infty = \phi$$

$$Q_2^\infty = \lambda\langle x, y\rangle.[\exists\, j \geq 0: \phi(x - jy, y) \wedge (1000 \leq min(x - jy, x)) \wedge (max(x - jy, x) \leq b)]$$

$$Q_3^\infty = \lambda\langle x, y\rangle.[\exists\, j \geq 1: \phi(x - jy, y) \wedge (1000 \leq min(x - jy, x - y)) \wedge (max(x - jy, x) \leq b)]$$

$$Q_4^\infty = \lambda\langle x, y\rangle.[(\phi(x, y) \wedge (x < 1000)) \vee ((y < 0) \wedge (\exists\, j \geq 1: \phi(x - jy, y) \wedge (x - jy \leq b) \wedge (x < 1000 \leq x - y)))]$$

$$Q_\xi^\infty = \lambda\langle x, y\rangle.[(y > 0) \wedge (\exists\, j \geq 0: \phi(x - jy, y) \wedge (1000 \leq x - jy < x \leq b < x + y))]$$

Equivalently Q^∞ can be obtained from P^∞ as follows:

$$\begin{aligned}
&\sigma_i(post(\tau^*)(v_\epsilon \wedge \sigma_\epsilon^{-1}(\phi))) \\
&= \sigma_i(\lambda s_2.[\exists\, \bar{s}: v_\epsilon(\bar{s}) \wedge \sigma_\epsilon^{-1}(\phi)(\bar{s}) \wedge post(\tau^*)(\lambda s.[s = \bar{s}])(s_2)]) \\
&= \lambda m_2.[\exists\, \bar{m}: \phi(\bar{m}) \wedge post(\tau^*)(\lambda s.[s = \langle\epsilon, \bar{m}\rangle])(\langle i, m_2\rangle)] \\
&= \lambda m_2.[\exists\, \bar{m}: \phi(\bar{m}) \wedge \sigma^{-1}(lfp(F_\pi(\lambda m.[m = \bar{m}])))(\langle i, m_2\rangle)] \\
&= \lambda m_2.[\exists\, \bar{m}: \phi(\bar{m}) \wedge lfp(F_\pi(\lambda m.[m = \bar{m}]))_i(m_2)]
\end{aligned}$$

Therefore at each program point i the set of descendants of the entry states satisfying the entry condition $\phi \in (I^2 \longrightarrow B)$ is characterized by

$$Q_i^\infty = \lambda\langle x, y\rangle.[\exists\, \bar{x}, \bar{y} \in I^2: \phi(\bar{x}, \bar{y}) \wedge P_i^\infty(x, y)]$$

For example:

$$\begin{aligned}
Q_\xi^\infty &= \lambda(x, y).[\exists\, \bar{x}, \bar{y} \in I^2: \phi(\bar{x}, \bar{y}) \wedge (\bar{x} \geq 1000) \wedge (\bar{y} > 0) \\
&\qquad \wedge (x = \bar{x} + ((b - \bar{x})\, div\, \bar{y})\bar{y}) \wedge (y = \bar{y})] \\
&= \lambda(x, y).[\exists\, \bar{x} \in I: (\exists\, j: \phi(x - jy, y) \wedge (x - jy \geq 1000) \\
&\qquad \wedge (y > 0) \wedge (x = \bar{x} + jy) \wedge j = (b - \bar{x})\, div\, y)] \\
&= \lambda(x, y).[(y > 0) \wedge (\exists\, j \geq 0: \phi(x - jy, y) \\
&\qquad \wedge (1000 \leq x - jy < x \leq b) \wedge (j = j + (b - x)\, div\, y))] \\
&= \lambda(x, y).[(y > 0) \wedge (\exists\, j \geq 0: \phi(x - jy, y) \\
&\qquad \wedge (1000 \leq x - jy < x \leq b < x + y))]
\end{aligned}$$

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 $B_\pi(\Psi) \in ((I^2 \longrightarrow B)^4 \longrightarrow (I^2 \longrightarrow B)^4))$ of backward semantic equations associated with the example program π and an exit specification $\Psi \in (I^2 \longrightarrow B)$ is the following:

$$P_1 = \lambda\langle x, y\rangle.[((x \in I) \wedge (x \geq 1000) \wedge P_2(x, y)) \vee ((x \in I) \wedge (x < 1000) \wedge P_4(x, y))]$$

$$P_2 = \lambda\langle x, y\rangle.[((x + y) \in I) \wedge P_3(x + y, y)]$$

$$P_3 = \lambda\langle x, y\rangle.[((x \in I) \wedge (x \geq 1000) \wedge P_2(x, y)) \vee ((x \in I) \wedge (x < 1000) \wedge P_4(x, y))]$$

$$P_4 = \Psi$$

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

$$\begin{aligned}
&\sigma_\epsilon(\nu_\epsilon \wedge pre(\tau^*)(\nu_\omega))\\
&\quad = \sigma_\epsilon(\nu_\epsilon \wedge lfp(\lambda\alpha.[\nu_\omega \vee pre(\tau)(\alpha)])) \qquad \text{[Theorem 10-9(2)]}\\
&\quad = \sigma_\epsilon(\nu_\epsilon \wedge \sigma^{-1}(lfp(B_\pi(\lambda m.[true])))) \qquad \text{[Theorem 10-19(1)]}\\
&\quad = \sigma_\epsilon\Big(\bigvee_{i \in [1,n]} \lambda s.[\nu_\epsilon(s) \wedge s \in \iota_i^{-1}(U) \wedge lfp(B_\pi(\lambda m.[true]))_i(\iota_i(s))]\Big)\\
&\quad = \sigma_\epsilon(lfp(B_\pi(\lambda m.[true]))_\epsilon \circ \iota_\epsilon)\\
&\quad = lfp(B_\pi(\lambda m.[true]))_\epsilon
\end{aligned}$$

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

$$\begin{aligned}
P_1^\infty &= \lambda\langle x, y\rangle.[(x < 1000) \vee (y < 0)]\\
P_2^\infty &= \lambda\langle x, y\rangle.[((x + y) \in I) \wedge ((x + y < 1000) \vee (y < 0))]\\
P_3^\infty &= \lambda\langle x, y\rangle.[(x < 1000) \vee (y < 0)]\\
P_4^\infty &= \lambda\langle x, y\rangle.[true]
\end{aligned}$$

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

$$\begin{aligned}
&\sigma_\epsilon(\nu_\epsilon \wedge \neg pre(\tau^*)(\nu_\xi))\\
&\quad = \sigma_\epsilon(\nu_\epsilon \wedge gfp(\lambda\alpha.[\neg\nu_\xi \wedge pre(\tau)(\alpha)])) \qquad \text{[Theorem 10-9(3)]}\\
&\quad = \sigma_\epsilon(\nu_\epsilon \wedge \sigma^{-1}(gfp(B_\pi(\lambda m.[true])))) \qquad \text{[Theorem 10-19(2)]}\\
&\quad = gfp(B_\pi(\lambda m.[true]))_\epsilon
\end{aligned}$$

The greatest fixed point Q^{ω} of the above system of equations where $\Psi = \lambda\langle x, y\rangle.[true]$ can be computed iteratively starting from $Q_i^0 = \lambda\langle x, y\rangle.[true]$, $i = 1, \ldots, 4$, inventing the general term of a chaotic iteration sequence, and passing to the limit:

$$Q_1^{\omega} = \lambda\langle x, y\rangle.[(y \leq 0) \vee (x < 1000)]$$

$$Q_2^{\omega} = \lambda\langle x, y\rangle.[((x + y \leq 0) \vee (x + y < 1000)) \wedge ((x + y) \in I)]$$

$$Q_3^{\omega} = \lambda\langle x, y\rangle.[(y \leq 0) \vee (x < 1000)]$$

$$Q_4^{\omega} = \lambda\langle x, y\rangle.[true]$$

The *set of entry states leading to a run-time error* is characterized by $\lambda\langle x, y\rangle \in I^2.[\neg Q_1^{\omega}(x, y)] = \lambda\langle x, y\rangle \in I^2.[(y > 0) \wedge (x \geq 1000)]$.

Equivalently, the set of ascendants of the run-time error states is characterized by $pre(\tau^*)(v_{\xi})$, which, according to Theorem 10-9(3) and Theorem 10-19(3), is equal (up to the isomorphism σ) to the least solution R^{ω} to

$$P_1 = \lambda\langle x, y\rangle.[(x \geq 1000) \wedge P_2(x, y)]$$

$$P_2 = \lambda\langle x, y\rangle.[(((x + y) \in I) \in P_3(x + y, y)) \vee ((x + y) \notin I)]$$

$$P_3 = \lambda\langle x, y\rangle.[(x \geq 1000) \wedge P_2(x, y)]$$

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

$$P_{\xi} = \lambda\langle x, y\rangle.[true]$$

that is, $R_1^{\omega} = R_3^{\omega} = \lambda\langle x, y\rangle.[(x \geq 1000) \wedge (y > 0)]$, $R_2^{\omega} = \lambda\langle x, y\rangle.[((x + y) \geq 0 \wedge (y > 0)) \vee ((x + y) \notin I)]$, $R_4^{\omega} = \lambda\langle x, y\rangle.[false]$, $R_{\xi}^{\omega} = \lambda\langle x, y\rangle.[true]$.

The *set of entry states which cause the program to diverge* is characterized by $\lambda\langle x, y\rangle.[Q_1^{\omega}(x, y) \wedge \neg P_1^{\omega}(x, y)] = \lambda\langle x, y\rangle.[(x \geq 1000) \wedge (y = 0)]$.

Equivalently, the states which cause the program to diverge can be characterized by $\neg pre(\tau^*)(v_{\xi} \vee v_{\sigma}) = gfp(\lambda\alpha.[\neg v_{\sigma} \wedge \neg v_{\xi} \wedge pre(\tau)(\alpha)])$, which, according to Theorem 10-19(4) is equal (up to the isomorphism σ) to $D^{\omega} = gfp(B_{\pi}(\lambda\langle x, v\rangle.[false]))$, that is, $D_1^{\omega} = D_2^{\omega} = D_3^{\omega} = \lambda\langle x, y\rangle.[(x \geq 1000) \wedge (y = 0)]$ and $D_4^{\omega} = D_{\xi}^{\omega} = \lambda\langle x, y\rangle.[false]$.

The *set of descendants of the input states satisfying an entry condition* $\phi \in (I^2 \longrightarrow B)$ is characterized by

$$\begin{aligned}
&post(\tau^*)(v_{\epsilon} \wedge \sigma_{\epsilon}^{-1})\phi)) \\
&\quad = \lambda\bar{s}.[\exists\, s_1 \in S\colon v_{\epsilon}(s_1) \wedge \sigma_{\epsilon}^{-1}(\phi)(s_1) \wedge lfp(\lambda\alpha.[\lambda s.[s = \bar{s}] \\
&\qquad\qquad \vee pre(\tau)(\alpha)])(s_1)] \\
&\quad = \lambda\bar{s}.[\exists\, m_{\epsilon} \in U_{\epsilon}\colon \phi(m_{\epsilon}) \wedge \sigma^{-1}(lfp(\sigma \circ \lambda\alpha.[\lambda s.[s = \bar{s}] \\
&\qquad\qquad \vee pre(\tau(\alpha)] \circ \sigma^{-1}))(\langle \epsilon, m_{\epsilon}\rangle)] \\
&\quad = \lambda\bar{s}.[\exists\, m_{\epsilon} \in U_{\epsilon}\colon \phi(m_{\epsilon}) \wedge lfp(\sigma \circ \lambda\alpha.[\lambda s.[s = \bar{s}] \vee pre(\tau)(\alpha)] \circ \sigma^{-1})_{\epsilon}(m_{\epsilon})]
\end{aligned}$$

According to Theorem 10-19(5), the direct decomposition of $\lambda\alpha.[\lambda s.[s = \bar{s}] \vee pre(\tau)(\alpha)]$ is the following when τ is defined by our example program.

$$P_1 = \lambda\langle x, y\rangle.[(\langle 1, \langle x, y\rangle\rangle = \bar{s}) \vee (x \in I \wedge x \geq 1000 \wedge P_2(x, y)) \vee (x \in I \wedge x \leq 1000 \wedge P_4(x, y)) \vee (P_\xi(x, y) \wedge x \notin I)]$$

$$P_2 = \lambda\langle x, y\rangle.[(\langle 2, \langle x, y\rangle\rangle = \bar{s}) \vee (x + y \in I \wedge P_3(x + y, y)) \vee (P_\xi(x, y) \wedge x + y \notin I)]$$

$$P_3 = \lambda\langle x, y\rangle.[(\langle 3, \langle x, y\rangle\rangle = \bar{s}) \vee (x \in I \wedge x \geq 1000 \wedge P_2(x, y)) \vee (x \in I \wedge x < 1000 \wedge P_4(x, y)) \vee (P_\xi(x, y) \wedge x \notin I)]$$

$$P_4 = \lambda\langle x, y\rangle.[(\langle 4, \langle x, y\rangle\rangle = \bar{s}) \vee P_4(x, y)]$$

$$P_\xi = \lambda\langle x, y\rangle.[(\langle \xi, \langle x, y\rangle\rangle = \bar{s}) \vee P_\xi(x, y)]$$

If $P^\omega(\bar{s})$ denotes $lfp(\sigma \circ \lambda\alpha.[\lambda s.[s = \bar{s}] \vee pre(\tau)(\alpha)] \circ \sigma^{-1})$, we determine that

$$\begin{aligned} P_1^\omega(\bar{s}) = \lambda\langle x, y\rangle.[&(\langle 1, \langle x, y\rangle\rangle = \bar{s}) \\ &\vee (\exists j \geq 0: (\forall i \in [0, j], 1000 \leq x + iy \leq b) \\ &\qquad \wedge (\langle 2, \langle x + jy, y\rangle\rangle = \bar{s})) \\ &\vee (\exists j \geq 1: (\forall i \in [0, j - 1], 1000 \leq x + iy \leq b) \\ &\qquad \wedge (x + jy \in I) \wedge (\langle 3, \langle x + jy, y\rangle\rangle = \bar{s})) \\ &\vee (\exists j \geq 0: (\forall i \in [0, j - 1], 1000 \leq x + iy \leq b) \\ &\qquad \wedge (x + jy \in I) \wedge (x + jy < 1000) \wedge (\langle 4, \langle x + jy, y\rangle\rangle = \bar{s})) \\ &\vee (\exists j \geq 1: (\forall i \in [0, j - 1], 1000 \leq x + jy \leq b) \\ &\qquad \wedge (x + jy \notin I) \wedge (\langle \xi, \langle x + (j - 1)y, y\rangle\rangle = \bar{s}))] \end{aligned}$$

At program point i, the set of descendants of the input states satisfying ϕ is

$$\sigma_i(\lambda\bar{s}.[\exists m_\epsilon \in U_\epsilon: \phi(m_\epsilon) \wedge P_\epsilon^\omega(\bar{s})(m_\epsilon)]) = \lambda m.[\exists m_\epsilon \in U_\epsilon: \phi(m_\epsilon) \wedge P_\epsilon^\omega(\langle i, m\rangle)(m_\epsilon)]$$

For our example

$$\lambda\langle x, y\rangle.[\exists \langle x_\epsilon, y_\epsilon\rangle \in I^2: \phi(x_\epsilon, y_\epsilon) \wedge P_1^\omega(\langle i, \langle x, y\rangle\rangle)(\langle x_\epsilon, y_\epsilon\rangle)]$$

when $i = 2$, this is equal to

$$\begin{aligned} &\lambda\langle x, y\rangle.[\exists \langle x_\epsilon, y_\epsilon\rangle \in I^2: \phi(x_\epsilon, y_\epsilon) \wedge (\exists j \geq 0: \forall i \in [0, j], \\ &\qquad 1000 \leq x_\epsilon + iy_\epsilon \leq b) \wedge (\langle 2, \langle x_\epsilon + jy_\epsilon, y_\epsilon\rangle\rangle = \langle 2, \langle x, y\rangle\rangle))] \\ &= \lambda\langle x, y\rangle.[\exists j \geq 0: \phi(x - jy, y) \wedge 1000 \leq min(x - jy, x) \\ &\qquad \wedge (max(x - jy, x) \leq b)] \end{aligned}$$

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 run-time 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 $\pi(S, \tau, v_\epsilon, v_\omega, v_\xi)$ is given by a predicate $P \in (S \longrightarrow B)$ characterizing a superset of the descendants of the entry states satisfying an input specification ϕ, that is, such that $post(\tau^*)(v_\epsilon \wedge \phi) \Longrightarrow P$. P gives a partial answer to the termination problem since if $P \wedge v_\sigma$ is false, then π surely always diverges, whereas if $P \wedge v_\sigma$ is not false, then one cannot conclude whether π 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 $\alpha = (v_\epsilon \wedge \phi) \vee post(\tau)(\alpha)$, that is, such that $lfp(\lambda\alpha.[v_\epsilon \wedge \phi) \vee post(\tau)(\alpha)]) \Longrightarrow P$.

Another example of approximate information about the behavior of a program $\pi(S, \tau, v_\epsilon, v_\omega, v_\xi)$ is given by a predicate $Q \in (S \longrightarrow B)$ which is a lower approximation of the greatest solution to the equation $\alpha = \neg v_\omega \wedge \neg v_\xi \wedge pre(\tau)(\alpha)$, that is, (Theorem 10-9(5)) such that $Q \Longrightarrow \neg pre(\tau^*)(v_\omega \wedge v_\xi)$. The approximation is that Q characterizes only a subset of the states which cause π 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 $(S \longrightarrow B)$ 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 $(S \longrightarrow B)$. A solution to the simplified equation $x = f'(x)$ will be computed iteratively using an extrapolation technique in order to accelerate the convergence 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 $L(\sqsubseteq, \bot, \top, \sqcap, \sqcup)$. We will consider a simplified and computer-representable image $\bar{L}$ of L and solve a less complex equation $x = \bar{f}(x)$ on $\bar{L}$. [Cous79] justifies the fact that the following connection must be established between L and $\bar{L}$.

Definition 10-20. Let $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ and $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be complete lattices. $\bar{L}$ is *an upper approximation* of L iff there exists a one-to-one complete $\sqcap$-morphism γ from $\bar{L}$ onto L.

Example 10-21. Let π 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 $lfp(F_\pi(\phi))_i \Rightarrow Q_i$ where Q_i belongs to the subset $L_1' \subseteq L_1 = (I^n \longrightarrow B)$ of predicates of the form $\lambda\langle X_1, \ldots, X_n\rangle.\left[\bigwedge_{j=1}^{n} P_j(X_j)\right]$, where P_j belongs to the subset $L_2' = \{\lambda X.[false], \lambda X.[X = 0], \lambda X.[-b-1 \leq X \leq 0], \lambda X.[0 \leq X \leq b], \lambda X.[true]\}$ of $L_2 = (I \longrightarrow B)$. We can define an upper approximation $\bar{L}_2$ of L_2 as shown in Fig. 10-5.

$\bar{L}_2$ =

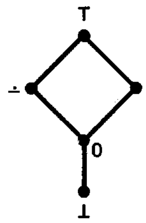

Figure 10-5

where $\gamma_2 \in (\bar{L}_2 \longrightarrow L_2)$ is defined by

$$
\begin{aligned}
\gamma_2 = \lambda s.[\textbf{case } s \textbf{ in}\\
&\bot \longrightarrow \lambda X.[false]\\
&\dot{-} \longrightarrow \lambda X.[-b-1 \leq X \leq 0]\\
&0 \longrightarrow \lambda X.[X = 0]\\
&\dot{+} \longrightarrow \lambda X.[0 \leq X \leq b]\\
&\top \longrightarrow \lambda X.[true]\\
\textbf{esac}]
\end{aligned}
$$

The upper approximation $\bar{L}_1$ of L_1 can be defined as

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

and $\gamma_1 \in (\bar{L}_1 \longrightarrow L_1)$ is such that

$$\gamma_1 = \lambda\langle s_1, \ldots, s_n\rangle.\left[\lambda\langle X_1, \ldots, X_n\rangle.\left[\bigwedge_{j=1}^{n} \gamma_2(s_j)(X_j)\right]\right]$$

Intuitively, γ gives the meaning $\gamma(\bar{x})$ of the elements $\bar{x}$ of $\bar{L}$. γ is assumed to be one-to-one so that no two distinct elements of $\bar{L}$ can have the same meaning. The hypothesis that γ 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 a least element, namely $\alpha(x) = \sqcap\{\bar{y} \in \bar{L} : x \sqsubseteq \gamma(\bar{y})\}$. The close relationship between α and γ is described by the following:

Theorem 10-22. Let $\alpha \in (L \longrightarrow \bar{L})$ be $\lambda x.\sqcap\{\bar{y} \in \bar{L} : x \sqsubseteq \gamma(\bar{y})\}$. α is a complete $\sqcup$-morphism from L onto $\bar{L}$. $\alpha \circ \gamma = \lambda\bar{x}.[\alpha(\gamma(\bar{x}))]$ is the identity function on $\bar{L}$. $\gamma \circ \alpha$ is an isotone, idempotent ($\gamma \circ \alpha \circ \gamma \circ \alpha = \gamma \circ \alpha$) and extensive ($\forall x \in L, x \sqsubseteq \gamma \circ \alpha(x)$) operator on L. Moreover $\gamma = \lambda\bar{y}.\sqcup\{x \in L : \alpha(x) \sqsubseteq \bar{y}\}$.

Example 10-23. Coming back to Example 10-21, we have:

$$\begin{aligned}
\alpha_2 = \lambda P \in (I \longrightarrow B).[\textbf{if}\ \ & P = \lambda X.[\mathit{false}]\ \textbf{then}\ \bot \\
& \textbf{elsif}\ P \Longrightarrow \lambda X.[X = 0]\ \textbf{then}\ 0 \\
& \textbf{elsif}\ P \Longrightarrow \lambda X.[X \leq 0]\ \textbf{then}\ \dot{-} \\
& \textbf{elsif}\ P \Longrightarrow \lambda X.[X \geq 0]\ \textbf{then}\ \dot{+} \\
& \textbf{else}\ \top\ \textbf{fi}]
\end{aligned}$$

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

For example,

$$\begin{aligned}
&\alpha_1(\lambda\langle x, y\rangle.[(x \geq 1) \wedge (x + y = 2) \wedge \mathit{even}(y)]) \\
&\quad = \langle \alpha_2(\lambda x.[(1 \leq x \leq b) \wedge \mathit{even}(x)]), \alpha_2(\lambda y.[(-b + 2 \leq y \leq 1) \wedge \mathit{even}(y)])\rangle \\
&\quad = \langle \dot{+}, \dot{-}\rangle
\end{aligned}$$

An upper approximation of the least fixed point $\mathit{lfp}(f)$ of an isotone operator f on the complete lattice L can be defined using an approximate image $\bar{f}$ of f on $(\bar{L} \longrightarrow \bar{L})$ as follows:

Definition 10-24. Let $\bar{L}(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ be an upper approximation of $L(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ and f be an isotone operator on L. $\bar{f}$ is an *upper approximation* of f on $\bar{L}$ iff $\bar{f}$ is an isotone operator on $\bar{L}$ such that $(\forall \bar{x} \in \bar{L}, \alpha(f(\gamma(\bar{x}))) \sqsubseteq \bar{f}(\bar{x}))$.

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

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.

$$P_i = \bigvee_{j \in pred_\pi(i)} post(\lambda x.[x + 1])(P_j) \qquad \text{where } P_i, P_j \in (I \to B)$$

The approximate rule is Definition 10-24.

$$\begin{aligned} P_i &= \alpha_2\Big(\bigvee_{j \in pred_\pi(i)} post(\lambda x.[x + 1])(\gamma_2(P_j))\Big) && \text{where } P_i, P_j \in \bar{L}_2 \\ &= \bigsqcup_{j \in pred_\pi(i)} \alpha_2(post(\lambda x.[x + 1])(\gamma_2(P_j))) && \text{(Theorem 10-22)} \end{aligned}$$

It remains to analyze:

$$\begin{aligned} &\alpha_2(post(\lambda x.[x + 1])(\gamma_2(P_j))) \\ &\quad = \alpha_2(\lambda x.[\exists x' \in I: \gamma_2(P_j)(x') \wedge x' \in dom(\lambda x.[x + 1]) \wedge x = x' + 1]) \\ &\quad = \alpha_2(\lambda x.[\gamma_2(P_j)(x - 1) \wedge (-b - 1 \leq x \leq b - 1)] \end{aligned}$$

According to the possible values of P_j we can distinguish the following cases:

$$\begin{aligned} &\alpha_2(post(\lambda x.[x + 1])(\gamma_2(\perp))) = \alpha_2(\lambda x.[false]) = \perp \\ &\alpha_2(post(\lambda x.[x + 1])\ (\gamma_2(0))) = \alpha_2(\lambda x.[x = 1]) = \dot{+} \\ &\alpha_2(post(\lambda x.[x + 1])(\gamma_2(\dot{-}))) = \alpha_2(\lambda x.[-b \leq x \leq 1]) = \top \\ &\alpha_2(post(\lambda x.[x + 1])(\gamma_2(\dot{+}))) = \alpha_2(\lambda x.[1 \leq x \leq b - 1]) = \dot{+} \\ &\alpha_2(post(\lambda x.[x + 1])(\gamma_2(\top))) = \alpha_2(\lambda x.[-b \leq x \leq b - 1]) = \top \end{aligned}$$

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 ($\dot{+} + \dot{+} = \dot{+}, \dot{+} + \dot{-} = \top$, $\dot{-} - \dot{+} = \dot{-}, \dot{-} * \dot{-} = \dot{+}, \ldots$) 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 π, constructs the associated system of approximate equations $x = \bar{f}(x)$. 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

{1}
  while $x \geq 1000$ **do**
{2}
    $x := x + y;$
{3}
  od;
{4}

is the following:

$$\langle x_1, y_1\rangle = \phi$$
$$\langle x_2, y_2\rangle = smash(\langle if\ (x_1 \sqcup x_3) \sqsubseteq \dot{-}\ then\ \bot\ else\ \dot{+}\ fi,\ y_1 \sqcup y_3\rangle)$$
$$\langle x_3, y_3\rangle = smash(\langle x_2 + y_2, y_2\rangle)$$
$$\langle x_4, y_4\rangle = smash(\langle x_1 \sqcup x_3, y_1 \sqcup y_3\rangle)$$
$$\langle x_\xi, y_\xi\rangle = if\ (x_2 \sqsubseteq 0) \vee (y_2 \sqsubseteq 0) \vee ((x_2 = \dot{+}) \wedge (y_2 = \dot{-})) \vee ((x_2 = \dot{-}) \wedge (y_2 = \dot{+}))\ then\ \langle\bot, \bot\rangle\ else\ \langle x_2, y_2\rangle\ fi$$

where $smash(\langle x, y\rangle) = if\ (x = \bot)\ or\ (y = \bot)\ then\ \langle\bot, \bot\rangle\ else\ \langle x, y\rangle\ fi$. Taking $\phi = \langle\dot{+}, \dot{-}\rangle$, a chaotic iterative resolution corresponding to a symbolic execution of the program is the following:

$$\langle x_i^0, y_i^0\rangle = \langle\bot, \bot\rangle \qquad i = 1, \ldots, 4, \xi$$
$$\langle x_1^1, y_1^1\rangle = \phi = \langle\dot{+}, \dot{-}\rangle$$
$$\langle x_2^1, y_2^1\rangle = smash(\langle if\ (x_1^1 \sqcup x_3^0) \sqsubseteq \dot{-}\ then\ \bot\ else\ \dot{+}\ fi, y_1^1 \sqcup y_3^0\rangle) = \langle\dot{+}, \dot{-}\rangle$$
$$\langle x_3^1, y_3^1\rangle = smash(\langle x_2^1 + y_2^1, y_2^1\rangle) = \langle\top, \dot{-}\rangle$$
$$\langle x_2^2, y_2^2\rangle = smash(\langle if\ (x_1^1 \sqcup x_3^1) \sqsubseteq \dot{-}\ then\ \bot\ else\ \dot{+}\ fi, y_1^1 \sqcup y_3^1\rangle) = \langle\dot{+}, \dot{-}\rangle$$

Stabilizing around the loop, the remaining components are

$$\langle x_4^2, y_4^2\rangle = smash(\langle x_1^1 \sqcup x_3^1, y_1^1 \sqcup y_3^1\rangle) = \langle\top, \dot{-}\rangle$$
$$\langle x_\xi^2, y_\xi^2\rangle = if\ (x_2^2 \sqsubseteq 0) \vee (y_2^2 \sqsubseteq 0) \vee ((x_2^2 = \dot{+}) \wedge (y_2^2 = \dot{-})) \vee ((x_2^2 = \dot{-}) \wedge (y_2^2 = \dot{+}))\ then\ \langle\bot, \bot\rangle\ else\ \langle x_2, y_2\rangle\ fi$$
$$= \langle\bot, \bot\rangle$$

The results are approximate but not useless since, e.g., they prove that no overflow can occur when the entry specification $(x \geq 0 \wedge y \leq 0)$ 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 $\bigsqcup_{i \geq 0} \bar{f}^i(\perp)$ to the least fixed point $lfp(\bar{f})$. 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 $\bar{L}_2 = \{\perp\} \cup \{[l, u]: -b - 1 \leq l \leq u \leq b\}$ where $\perp$ is the infimum and $([l_1, u_1] \sqsubseteq [l_2, u_2])$ iff $(l_2 \leq l_1 \leq u_1 \leq u_2)$. The meaning of the elements of $\bar{L}_2$ is given by $\gamma_2(\perp) = \lambda X.[false]$ and $\gamma_2([l, u]) = \lambda X.[l \leq X \leq u]$.
2. The complete lattice $\bar{L}_1 = \{\langle i_1, \ldots, i_n\rangle: (\forall j \in [1, n], i_j \in (\bar{L}_2 - \{\perp\}))\} \cup \{\langle\perp, \ldots, \perp\rangle\}$ with $\gamma_1 = \lambda\langle i_1, \ldots, i_n\rangle.[\lambda\langle X_1, \ldots, X_n\rangle.\left[\bigwedge_{j=1}^{n} \gamma_2(i_j)(X_j)\right]]$.
3. The resolution of the equation $x = [0, 0] \sqcup (x + 1)$ on $\bar{L}_2$ where $\perp + 1 = \perp$ and $[l, u] + 1 = [l + 1, \ min(u + 1, b)]$ involves a very large number of iterates $x^0 = \perp, x^1 = [0, 0], x^2 = [0, 1], x^3 = [0, 2], \ldots, x^{b+1} = [0, b], x^{b+2} = [0, b]$.

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* $\nabla \in (\mathrm{N} \rightarrow (\bar{L} \times \bar{L} \rightarrow \bar{L}))$ is such that:

1. $\forall j > 0, \forall x, y \in \bar{L}, x \sqcup y \sqsubseteq x \nabla(j)\, y$
2. For any ascending chain $y^0 \sqsubseteq y^1 \sqsubseteq \ldots \sqsubseteq y^n \sqsubseteq \ldots$ of elements of $\bar{L}$, the ascending chain $x^0 = y^0, x^1 = x^0 \nabla(1)\, y^1, \ldots, x^n = x^{n-1} \nabla(n)\, y^n, \ldots$ 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 $\bar{f}$ be an isotone operator on $\bar{L}(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ and ∇ be a widening operator. The limit u of the sequence

$$\begin{aligned} x^0 &= d \\ x^{n+1} &= x^n \nabla(n+1) \bar{f}(x^n) && \text{if } \neg(\bar{f}(x^n) \sqsubseteq x^n) \\ x^{n+1} &= x^n && \text{if } \bar{f}(x^n) \sqsubseteq x^n \end{aligned}$$

can be computed in a finite number of steps. Moreover $lfp(\bar{f}) \sqsubseteq u$ and $\bar{f}(u) \sqsubseteq u$.

Example 10-31. When $\bar{L}$ satisfies the ascending chain condition, one can choose $\forall j > 0, \nabla(j) = \sqcup$, in which case $x^k = lfp(\bar{f})$. 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 $j > 0$:

$$\forall x \in \bar{L}_2, \perp \nabla_2(j)\, x = x\, \nabla_2(j) \perp = x$$

$$\begin{aligned} &[l_1, u_1] \nabla_2(j) [l_2, u_2] \\ &\qquad = [\mathit{if}\ 0 \leq l_2 < l_1\ \mathit{then}\ 0\ \mathit{elsif}\ l_2 < l_1\ \mathit{then}\ -b-1\ \mathit{else}\ l_1\ \mathit{fi}, \\ &\qquad\quad \mathit{if}\ u_1 < u_2 \leq 0\ \mathit{then}\ 0\ \mathit{elsif}\ u_1 < u_2\ \mathit{then}\ b\ \mathit{else}\ u_1\ \mathit{fi}] \end{aligned}$$

A widening ∇_1 on $\bar{L}_1$ is obtained by applying ∇_2 componentwise:

$$\langle i_1, \ldots, i_n \rangle \nabla_1(j) \langle i'_1, \ldots, i'_n \rangle = \langle i_1 \nabla_2(j)\, i'_1, \ldots, i_n \nabla_2(j)\, i'_n \rangle$$

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 $\odot$.

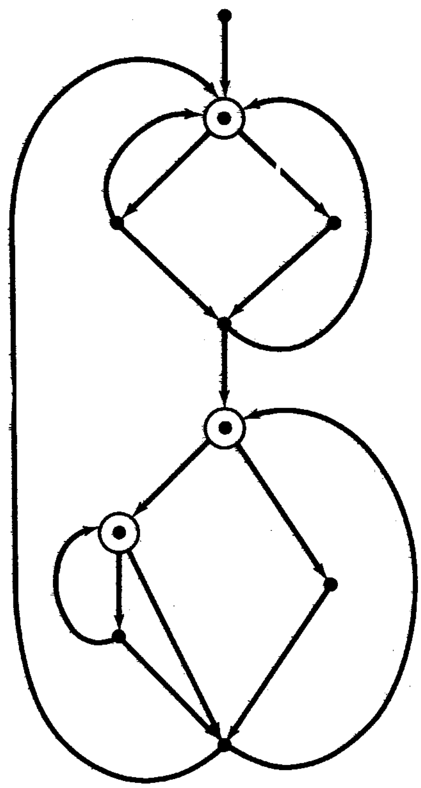

Figure 10-6

{1}

while $(10 \leq x) \wedge (x \leq 100)$ **do**

{2}

$x := x + y;$

{3}

od;

{4}

is the following:

$$\begin{cases} \langle x_1, y_1 \rangle = \phi \\ \langle x_2, y_2 \rangle = smash(\langle((x_1 \sqcup x_3) \sqcap [10, 100]), (y_1 \sqcup y_3)\rangle) \\ \langle x_3, y_3 \rangle = smash(\langle x_2 + y_2, y_2 \rangle) \\ \langle x_4, y_4 \rangle = smash(\langle((x_1 \sqcup x_3) \sqcap (-b - 1, 9]), (y_1 \sqcup y_3)\rangle) \\ \langle x_\xi, y_\xi \rangle = if\ \langle x_2, y_2 \rangle = \langle \bot, \bot \rangle\ then\ \langle \bot, \bot \rangle\ else\ underflow(x_2, y_2) \\ \qquad \sqcup\ overflow(x_2, y_2)\ fi \end{cases}$$

where

$\forall x \in \bar{L}_2, \perp \sqcup x = x \sqcup \perp = x,$

$\quad \perp \sqcap x = x \sqcap \perp = \perp,$

$\quad \perp + x = x + \perp = \perp,$

$[l_1, u_1] \sqcup [l_2, u_2] = [min(l_1, l_2), max(u_1, u_2)]$

$[l_1, u_1] \sqcap [l_2, u_2] = \mathit{if}\ max(l_1, l_2) \leq min(u_1, u_2)\ \mathit{then}\ [max(l_1, l_2), min(u_1, u_2)]$
$\mathit{else} \perp \mathit{fi}$

$[l_1, u_1] + [l_2, u_2] = [l_1 + l_2, u_1 + u_2] \sqcap [-b - 1, b]$

$\mathit{overflow}([l_1, u_1], [l_2, u_2]) = \mathit{if}\ (u_1 \geq 1) \wedge (u_2 \geq 1)\ \mathit{then} \langle([l_1, u_1]$
$\sqcap [b - u_2 + 1, b]), ([l_2, u_2] \sqcap [b - u_1 + 1, b])\rangle\ \mathit{else} \langle\perp, \perp\rangle\ \mathit{fi}$

$\mathit{underflow}([l_1, u_1], [l_2, u_2]) = \mathit{if}\ (l_1 \leq -1) \wedge (l_2 \leq -1)\ \mathit{then} \langle([l_1, u_1]$
$\sqcap [-b - 1, -b - 2 - l_2]), ([l_2, u_2]$
$\sqcap [-b - 1, -b - 2 - l_1])\rangle\ \mathit{else} \langle\perp, \perp\rangle\ \mathit{fi}$

$\mathit{smash}(\langle x, y\rangle) = \mathit{if}\ (x = \perp) \vee (y = \perp)\ \mathit{then} \langle\perp, \perp\rangle\ \mathit{else} \langle x, y\rangle \mathit{fi}$

Taking $\phi = \langle[9, 11], [-1, +1]\rangle$ the resolution uses the widening operators of Example 10-31 for the loop head node 2:

$\langle x_i^0, y_i^0\rangle = \langle\perp, \perp\rangle \qquad i = 1, \ldots, 4, \xi$

$\langle x_1^1, y_1^1\rangle = \phi = \langle[9, 11], [-1, 1]\rangle$

$\langle x_2^1, y_2^1\rangle = \langle x_2^0, y_2^0\rangle \nabla_1(1)\ \mathit{smash}(\langle(x_1^1 \sqcup x_3^0) \sqcap [10, 100], (y_1^1 \sqcup y_3^0)\rangle)$
$= \langle\perp \nabla_2(1) [10, 11], \perp \nabla_2(1) [-1, 1]\rangle = \langle[10, 11], [-1, 1]\rangle$

$\langle x_3^1, y_3^1\rangle = \mathit{smash}(\langle x_2^1 + y_2^1, y_2^1\rangle) = \langle[9, 12], [-1, 1]\rangle$

$\langle x_2^2, y_2^2\rangle = \langle x_2^1, y_2^1\rangle \nabla_1(2)\ \mathit{smash}(\langle(x_1^1 \sqcup x_3^1) \sqcap [10, 100], (y_1^1 \sqcup y_3^1)\rangle)$
$= \langle[10, 11) \nabla_2(2) [10, 12], [-1, 1] \nabla_2(2) [-1, 1]\rangle = \langle[10, b], [-1, 1]\rangle$

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.

$\langle x_3^2, y_3^2\rangle = \mathit{smash}(\langle x_2^2 + y_2^2, y_2^1\rangle) = \langle[9, b], [-1, +1]\rangle$

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

$\langle x_2^3, y_2^3\rangle = \langle x_2^2, y_2^2\rangle = \langle[10, b], [-1, 1]\rangle$

since $\langle[10, 100], [-1, 1]\rangle = \mathit{smash}(\langle(x_1^1 \sqcup x_3^2) \sqcap [10, 100], (y_1^1 \sqcup y_3^2)\rangle) \sqsubseteq \langle x_2^2, y_2^2\rangle$. The remaining components are:

$$\langle x_4^3, y_4^3 \rangle = smash(\langle ((x_1^1 \sqcup x_3^2) \sqcap [-b-1, 9]), (y_1^1 \sqcup y_3^2) \rangle)$$
$$= \langle [9, 9], [-1, +1] \rangle$$
$$\langle x_\xi^3, y_\xi^3 \rangle = \mathit{if}\ \langle x_2^3, y_2^3 \rangle = \langle \perp, \perp \rangle\ \mathit{then}\ \langle \perp, \perp \rangle\ \mathit{else}\ underflow(x_2^3, y_2^3)$$
$$\sqcup\ overflow(x_2^3, y_2^3)\ \mathit{fi}$$
$$= \langle \perp, \perp \rangle \sqcup \langle [b, b], [1, 1] \rangle = \langle [b, b], [1, 1] \rangle$$

The approximate result is

$$\langle x_1^1, y_1^1 \rangle = \langle [9, 11], [-1, 1] \rangle$$
$$\langle x_2^3, y_2^3 \rangle = \langle [10, b], [-1, 1] \rangle$$
$$\langle x_3^2, y_3^2 \rangle = \langle [9, b], [-1, 1] \rangle$$
$$\langle x_4^3, y_4^3 \rangle = \langle [9, 9], [-1, 1] \rangle$$
$$\langle x_\xi^3, y_\xi^3 \rangle = \langle [b, b], [1, 1] \rangle$$

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

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* $\Delta \in (\mathrm{N} \longrightarrow ((\bar{L} \times \bar{L}) \longrightarrow \bar{L}))$ is such that:

1. $\forall j > 0, (\forall x, y \in \bar{L}^2 : y \sqsubseteq x), y \sqsubseteq x\, \Delta(j)\, y \sqsubseteq x$
2. For any descending 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 $j > 0$:

$$\forall x \in \bar{L}_2, \perp \Delta_2(j)\, x = x\, \Delta_2(j) \perp = \perp$$
$$[l_1, u_1]\, \Delta_2(j)\, [l_2, u_2] = [\mathit{if}\ (l_1 = b - 1) \vee (l_1 = 0)\ \mathit{then}\ l_2\ \mathit{else}\ min(l_1, l_2)\ \mathit{fi},$$
$$\mathit{if}\ (u_1 = b) \vee (u_1 = 0)\ \mathit{then}\ u_2\ \mathit{else}\ max(u_1, u_2)\ \mathit{fi}]$$

This narrowing operator attempts to improve the zero or infinite bounds ($-b - 1$ and b) which might have been too imprecisely extrapolated by the widening operator.

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

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

$$\begin{aligned}
\langle x_2^4, y_2^4\rangle &= \langle x_2^3, y_2^3\rangle\, \Delta_1(1)\, smash(\langle((x_1^1 \sqcup x_3^2) \sqcap [10, 100], (y_1^1 \sqcup y_3^2)\rangle) \\
&= \langle [10, b]\, \Delta_2(1)\, [10, 100], [-1, 1]\, \Delta_2(1)\, [-1, 1]\rangle \\
&= \langle [10, 100], [-1, 1]\rangle \\
\langle x_3^4, y_3^4\rangle &= smash(\langle x_2^4 + y_2^4, y_2^4\rangle) = \langle [9, 101], [-1, 1]\rangle \\
\langle x_2^5, y_2^5\rangle &= \langle x_2^4, y_2^4\rangle\, \Delta_1(2)\, smash(\langle((x_1^1 \sqcup x_3^4) \sqcap [10, 100], (y_1^1 \sqcup y_3^2)\rangle) \\
&= \langle [10, 100]\, \Delta_2(2)\, [10, 100], [-1, 1]\, \Delta_2(2)\, [-1, 1]\rangle \\
&= \langle [10, 100], [-1, 1]\rangle = \langle x_2^4, y_2^4\rangle
\end{aligned}$$

Stabilization around the loop has been achieved. The components depending on $\langle x_2^5, y_2^5\rangle$ remain to be evaluated. The final result is

$$\begin{aligned}
\langle x_1^1, y_1^1\rangle &= \langle [9, 11], [-1, 1]\rangle \\
\langle x_2^5, y_2^5\rangle &= \langle [10, 100], [-1, 1]\rangle \\
\langle x_3^4, y_3^4\rangle &= \langle [9, 101], [-1, 1]\rangle \\
\langle x_4^5, y_4^5\rangle &= \langle [9, 9], [-1, 1]\rangle \\
\langle x_\xi^5, y_\xi^5\rangle &= \langle \perp, \perp\rangle
\end{aligned}$$

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.

```
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);
```

```
{1}
        while bi ≤ bs do
{2}
          m := (bi + bs) div 2;
{3}
          if k = R(m) then
            bi := bs + 1;
{4}
          elsif k < R(m) then
            bs := m − 1;
{5}
          else
            bi := m + 1;
{6}
          fi;
{7}
        od;
{8}
        if R(m) ≠ k then m := lb(R) − 1 fi;
{9}
      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 $\bar{P}$ 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 $\bar{P}$, 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

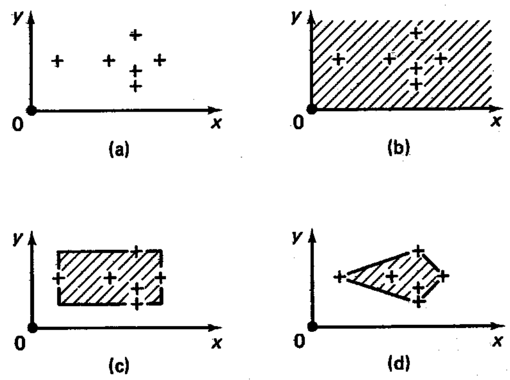

Figure 10-7

of the variables *bi*, *bs*, and *m* then the corresponding approximate analysis of the procedure *binary-search* is the following:

$$P_1 = (bi > 0) \wedge (bs > 0)$$
$$P_2 = (bi > 0) \wedge (bs > 0)$$
$$P_3 = (bi > 0) \wedge (bs > 0) \wedge (m > 0)$$
$$P_4 = (bi > 0) \wedge (bs > 0) \wedge (m > 0)$$
$$P_5 = (bi > 0) \wedge (bs \geq 0) \wedge (m > 0)$$
$$P_6 = (bi > 0) \wedge (bs > 0) \wedge (m > 0)$$
$$P_7 = (bi > 0) \wedge (bs \geq 0) \wedge (m > 0)$$
$$P_8 = (bi > 0) \wedge (bs \geq 0)$$
$$P_9 = (bi > 0) \wedge (bs \geq 0)$$

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:

$$P_1 = (bi = 1) \wedge (bs = 100)$$
$$P_2 = (1 \leq bi \leq 100) \wedge (1 \leq bs \leq 100)$$
$$P_3 = (1 \leq bi \leq 100) \wedge (1 \leq bs \leq 100) \wedge (1 \leq m \leq 100)$$
$$P_4 = (2 \leq bi \leq 101) \wedge (1 \leq bs \leq 100) \wedge (1 \leq m \leq 100)$$
$$P_5 = (1 \leq bi \leq 100) \wedge (0 \leq bs \leq 99) \wedge (1 \leq m \leq 100)$$
$$P_6 = (2 \leq bi \leq 101) \wedge (1 \leq bs \leq 100) \wedge (1 \leq m \leq 100)$$
$$P_7 = (1 \leq bi \leq 101) \wedge (0 \leq bs \leq 100) \wedge (1 \leq m \leq 100)$$
$$P_8 = (1 \leq bi \leq 101) \wedge (0 \leq bs \leq 100) \wedge (1 \leq m \leq 100)$$
$$P_9 = (1 \leq bi \leq 101) \wedge (0 \leq bs \leq 100) \wedge (0 \leq m \leq 100)$$

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, *bi* and *bs* should have been declared *bi*: 1 .. 101; *bs*: 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:

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 *bi*, *bs*, *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:

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

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

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

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

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

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

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

$$\begin{aligned} P_7 = {} & (m \leq bs + 1) \wedge (3bi \leq 2bs + n + 3) \wedge (3bi \leq 2bs + 2m + 3) \\ & \wedge (2bi \leq 2bs + 3) \wedge (bs + m + 1 \leq bi + n) \\ & \wedge (bi + m \leq bs + n + 1) \wedge (1 \leq n) \wedge (bs \leq n) \wedge (bs \leq 2m) \\ & \wedge (bs + 4 \leq 3bi + m) \wedge (1 \leq 2m) \wedge (bs + 1 \leq bi + m) \end{aligned}$$

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

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, Wegb75] 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, ReiJ77, ReiJ78, Rose77a, Rose78a, Tarj75b]. Boolean techniques for interprocedural analysis 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

Aho76 AHO, ALFRED V., and JEFFREY D. ULLMAN, "Node Listings for Reducible Flow Graphs," *J. Comput. Syst. Sci.*, 13, no. 3 (December 1976), 286–299.

Aho77 AHO, ALFRED V., and JEFFREY D. ULLMAN, *Principles of Compiler Design.* Reading, MA: Addison-Wesley, 1977.

Alle74 ALLEN, FRANCES E., "Interprocedural Data Flow Analysis," *Information Processing 74*, Proc. IFIP Congress 74, Stockholm, Sweden (August 1974), ed. J. L. Rosenfield, pp. 398–408. Amsterdam: North-Holland, 1974.

Bart77a BARTH, JEFFREY M., "An Interprocedural Data Flow Analysis Algorithm," *Conf. Rec. 4th ACM Symp. on Principles of Programming Languages*, Los Angeles, CA (January 1977), pp. 119–131.

Berm76 BERMAN, LEONARD, and GEORGE MARKOWSKY, "Linear and Non-linear Approximate Invariants," IBM RC7241 (February 1976), T. J. Watson Research Center, Yorktown Heights, NY.

Birk67 BIRKHOFF, G., *Lattice Theory* (3rd ed.), Vol. 25. Providence, RI: AMS Colloquium Publications, 1967.

ClaE77 CLARKE, E. M., JR., "Program Invariants as Fixed Points," *Proc. 18th Ann. Symp. on Foundations of Computer Science*, Providence, RI (October–November 1977), pp. 18–29.

ClaE79 ———, "Synthesis of Resource Invariants for Concurrent Programs," *Conf. Rec. 6th ACM Symp. on Principles of Programming Languages*, San Antonio, TX (January 1979), pp. 211–221.

Cous77a COUSOT, PATRICK, and RADHIA COUSOT, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," *Conf. Rec. of 4th ACM Symp. on Principles of Programming Languages*, Los Angeles, CA (January 1977), pp. 238–252.

Cous77b COUSOT, PATRICK, "Asynchronous Iterative Methods for Solving a Fixed Point System of Monotone Equations in a Complete Lattice," Rapport de Recherche No. 88 (September 1977), Laboratoire d'Informatique, Grenoble, France.

Cous77c COUSOT, PATRICK, and RADHIA COUSOT, "Automatic Synthesis of Optimal Invariant Assertions: Mathematical Foundations," *Proc. ACM Symp. on Artificial Intelligence and Programming Languages*, Rochester, NY, *SIGPLAN Notices*, 12, no. 8 (August 1977), 1–12.

Cous77d ———, "Constructive Versions of Tarski's Fixed Point Theorems," *Pacific J. Math*, 82, no. 1 (May 1979), 43–57.

Cous77e ———, "Static Determination of Dynamic Properties of Recursive Procedures," *IFIP Working Conf. on Programming Concepts*, St. Andrews, N.B., Canada (August 1977), ed. Erich J. Neuhold. New York: North-Holland, 1978, pp. 237–277.

Cous77f ———, "Static Determination of Dynamic Properties of Generalized Type Unions," *SIGPLAN Notices*, 12, no. 3 (March 1977), 77–94.

Cous78 COUSOT, PATRICK, and N. HALBWACHS, "Automatic Discovery of Linear Restraints Among Variables of a Program," *Conf. Rec. 5th ACM Symp. on Principles of Programming Languages*, Tucson, AZ (January 1978), pp. 84–97.

Cous79 COUSOT, PATRICK, and RADHIA COUSOT, "Systematic Design of Program Analysis Frameworks," *Conf. Rec. 6th ACM Symp. on Principles of Programming Languages*, San Antonio, TX (January 1979), pp. 269–282.

Dijk76 DIJKSTRA, EDSGER W., *A Discipline of Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1976.

Floy67 FLOYD, R. W., "Assigning Meanings to Programs," *Proc. Symp. in Applied Mathematics of the AMS*, ed. J. T. Schwartz, Providence, RI (1967), 19–32.

Fong75 FONG, AMELIA C., J. KAM, and JEFFREY D. ULLMAN, "Application of Lattice Algebra to Loop Optimization," *Conf. Rec. 2nd ACM Symp. on Principles of Programming Languages*, Palo Alto, CA (January 1975), pp. 1–9.

Fong76 FONG, AMELIA C., and JEFFREY D. ULLMAN, "Induction Variables in Very High Level Languages," *Conf. Rec. 3rd ACM Symp. on Principles of Programming Languages*, Atlanta, GA (January 1976), pp. 104–112.

Fong77 FONG, AMELIA C., "Generalized Common Subexpressions in Very High Level Languages," *Conf. Rec. 4th ACM Symp. on Principles of Programming Languages*, Los Angeles, CA (January 1977), pp. 48–57.

Fosd76 FOSDICK, L. D., and L. J. OSTERWEIL, "Data Flow Analysis in Software Reliability," *Comput. Surv.*, 8, no. 3 (September 1976), 305–330.

Germ78 GERMAN, S., "Automating Proofs of the Absence of Common Runtime Errors," *Conf. Rec. 5th ACM Symp. on Principles of Programming Languages*, Tucson, AZ (January 1978), pp. 105–118.

Gill77 GILLETT, W. D., "Iterative Global Flow Techniques for Detecting Program Anomalies," Ph.D. thesis UIUCDCS-R-77-868, (January 1977), University of Illinois at Urbana-Champaign.

Grah76 GRAHAM, S. L., and M. WEGMAN, "A Fast and Usually Linear Algorithm for Global Flow Analysis," *J. ACM*, 23, no. 1 (January 1976), 172–202.

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.

Harr77a HARRISON, WILLIAM H., "Compiler Analysis of the Value Ranges for Variables," *IEEE Trans. Software Eng.*, SE-3, no. 3 (May 1977), 243–250.

Hech77 HECHT, MATTHEW S., *Flow Analysis of Computer Programs*. New York: Elsevier North-Holland, 1977.

Hoar69 HOARE, C. A. R., "An Axiomatic Basis for Computer Programming," *Commun. ACM*, 12, no. 10 (October 1969), 576–583.

Jone76 JONES, NEIL D., and STEVEN S. MUCHNICK, "Binding Time Optimization in Programming Languages: Some Thoughts Toward the Design of an Ideal Language," *Conf. Rec. 3rd ACM Symp. on Principles of Programming Languages*, Altanta, GA (January 1976), pp. 77–94.

Kam77 KAM, J. B., and JEFFREY D. ULLMAN, "Monotone Data Flow Analysis Frameworks," *Acta Inf.*, 7, fasc. 3 (1977), 305–317.

Kapl78a KAPLAN, M. A., and JEFFREY D. ULLMAN, "A General Scheme for the Automatic Inference of Variable Types," *Conf. Rec. 5th ACM Symp. on Principles of Programming Languages*, Tucson, AZ (January 1978), pp. 60–75.

Karr76 KARR, M., "Affine Relationships Among Variables of a Program," *Acta Inf.*, 6, fasc. 2 (April 1976), 133–151.

Kell76 KELLER, R. M., "Formal Verification of Parallel Programs," *Commun. ACM*, 19, no. 7 (July 1976), 371–384.

Kenn75b KENNEDY, KENNETH W., "Node Listing Applied to Data Flow Analysis," *Conf. Rec. 2nd ACM Symp. on Principles of Programming Languages*, Palo Alto, CA (January 1975), pp. 10–21.

Kild73 KILDALL, G. A., "A Unified Approach to Global Program Optimization," *Conf. Rec. ACM Symp. on Principles of Programming Languages*, Boston, MA (October 1973), pp. 194–206.

King69 KING, J., "A Program Verifier," Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1969.

Klee52 KLEENE, STEPHEN COLE, *Introduction to Metamathematics*. New York: D. Van Nostrand. 1952.

Lome75 LOMET, D. B., "Data Flow Analysis in the Presence of Procedure Calls," IBM Research Report RC-5728 (1975), T. J. Watson Research Center, Yorktown Heights, NY.

Naur66 NAUR, P. "Proof of Algorithms by Generalized Snapshots," *BIT*, 6(1966), 310–316.

Park69 PARK, DAVID, "Fixpoint Induction and Proofs of Program Properties," in *Machine Intelligence 5*, ed. Bernard Meltzer and Donald Michie. New York: American Elsevier, 1969, pp. 59–78.

Pnue77 PNUELI, A., "The Temporal Logic of Programs," *Proc. 18th Ann. Symp. on Foundations of Computer Science*, Providence, RI (October–November 1977), pp. 46–57.

ReiJ77 REIF, JOHN H., and HARRY R. LEWIS, "Symbolic Evaluation and the Global Value Graph," *Conf. Rec. 4th ACM Symp. on Principles of Programming Languages*, Los Angeles, CA (January 1977), pp. 104–118.

ReiJ78 REIF, JOHN H., "Symbolic Program Analysis in Almost Linear Time," *Conf. Rec. 5th ACM Symp. on Principles of Programming Languages*, Tucson, AZ (January 1978), pp. 76–83.

Rose77a ROSEN, BARRY K., "Applications of High Level Control Flow," *Conf. Rec. 4th ACM Symp. on Principles of Programming Languages*, Los Angeles, CA (January 1977), pp. 38–47.

Rose78a ———, "Monoids for Rapid Data Flow Analysis," *Proc. 5th ACM Symp. on Principles of Programming Languages*, Tucson, AZ (January 1978), pp. 47–59.

Rose79 ———, "Data Flow Analysis for Procedural Languages," *J. ACM*, 26, no. 2 (April 1979), 322–344.

Scha73 SCHAEFER, MARVIN, *A Mathematical Theory of Global Program Optimization*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

Schw75b SCHWARTZ, JACOB T. "Optimization of Very High Level Languages I: Value Transmission and its Corollaries," *J. Comput. Languages*, 1 (1975), 161–194.

Schw75c ———, "Optimization of Very High Level Languages II: Deducing Relationships of Inclusion and Membership," *J. Comput. Languages*, 1 (1975), 197–218.

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

Sint72 SINTZOFF, M., "Calculating Properties of Programs by Valuations on Specific Models," *Proc. ACM Conf. on Proving Assertions about Programs*, New Mexico (1972), pp. 203–207.

Spil72 SPILLMAN, THOMAS C., "Exposing Side-Effects in a PL/I Optimizing Compiler," *Information Processing 71*, Proc. IFIP Congress 71, Ljubljana, Yugoslavia (August 1971), ed. C. V. Freiman, 376–381. Amsterdam: North-Holland, 1972.

Suzu77 SUZUKI, NORIHISA, and KIYOSHI ISHIHATA, "Implementation of Array Bound Checker," *Conf. Rec. of 4th ACM Symp. on Principles of Programming Languages*, Los Angeles, CA (January 1977), pp. 132–143.

Tarj75b TARJAN, ROBERT ENDRE, "Solving Path Problems on Directed Graphs," STAN-CS-75-528 (November 1975), Computer Science Department, Stanford University, Stanford, CA.

Tarj76 ———, "Iterative Algorithms for Global Flow Analysis," in *Algorithms and Complexity, New Directions and Recent Results*, ed. J. F. Traub. New York: Academic Press, 1976, pp. 11–101.

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

Wegb75 WEGBREIT, BEN, "Property Extraction in Well-founded Property Sets," *IEEE Trans. Software Eng.*, SE-1, no. 3 (September 1975), 270–285.

Wels77 WELSH, J., "Economic Range Checking in PASCAL," Department of Computer Science, Queen's University, Belfast, Northern Ireland, October 1977.