PROGRAM FLOW ANALYSIS: THEORY AND APPLICATIONS

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

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

10-1. INTRODUCTION

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

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

Example 10-1. Consider the program:

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

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

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

\[
P_1 = \lambda(x, y).[(\neg b - 1 \leq x \leq b) \land (\neg b - 1 \leq y \leq b)]
\]

\[
P_2 = \lambda(x, y).[(1000 \leq x \leq b) \land (\neg b - 1 \leq y \leq b)]
\]

\[
P_3 = \lambda(x, y).[(1000 + y \leq x \leq \min(b, b + y))
\]

\[
\land (\neg b - 1 \leq y \leq b)]
\]

\[
P_4 = \lambda(x, y).[(\neg b - 1 \leq x < 1000) \land (\neg b - 1 \leq y \leq b)]
\]

10-2. SUMMARY

In Section 10-3 we define what we mean by flowchart programs, that is, we define their abstract syntax and operational semantics. A program defines a dynamic discrete system \([Kc76, Pnue77]\) 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 theo-
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
4. If \( c_1 \in V \) is of out-degree 2, \( \langle c_1, c_2 \rangle \in E, \langle c_1, c_3 \rangle \in E, L(\langle c_1, c_2 \rangle) = p, L(\langle c_1, c_3 \rangle) = \neg p, p \in I(U) \) then if \( m \notin \text{dom}(p) \) then \\
\( \mathcal{T}(\langle c_1, m \rangle) = \langle c_1, m \rangle \) else if \( p(m) \) then \\
\( \mathcal{T}(\langle c_1, m \rangle) = \langle c_2, m \rangle \) else \\
\( \mathcal{T}(\langle c_1, m \rangle) = \langle c_3, m \rangle \).

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

10-3.2.3. Transitive closure of a binary relation

If \( \alpha, \beta \in (S \times S \rightarrow B) \) are two binary relations on \( S \), their product \( \alpha \circ \beta \) is defined as \( \lambda(\langle s_1, s_2 \rangle, \exists s_3 \in S \colon \alpha(s_1, s_3) \land \beta(s_3, s_2)) \). For any natural number \( n \), the \( n \)-extension \( \alpha^n \) of \( \alpha \) is defined recursively as \( \alpha^0 = \epsilon = \lambda(\langle s_1, s_2 \rangle, [\exists s_1 = s_2]) \), \( \alpha^{n+1} = \alpha \circ \alpha^n \). The (reflexive) transitive closure of \( \alpha \) 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 \( \pi \) starting from an initial state \( s_1 \in S \) is said to lead to an error iff \( \exists s_2 \in S \colon \tau^*(s_1, s_2) \land v_\varnothing(s_2) \), and to terminate iff \( \exists s_2 \in S \colon \tau^*(s_1, s_2) \land v_\omega(s_2) \). Otherwise it is said to diverge. The output of the execution of a syntactically valid program \( \pi \) starting from an initial state \( \langle \epsilon, m_1 \rangle \in S \) is defined if and only if this execution terminates with \( m_2 \in U \) such that \( \tau^*(\langle \epsilon, m_1 \rangle, \langle \omega, m_2 \rangle) \), and \( m_2 \) is the output.

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

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

10-4.1. Discrete Dynamic Systems

A discrete dynamic system is a 5-tuple \( \langle S, \tau, v_\epsilon, v_\omega, v_\varnothing \rangle \) such that \( S \) is a
The following study is devoted to total (\( \forall s_1 \in S, \exists s_2 \in S: \tau(s_1, s_2) \)) and deterministic (\( \forall s_1, s_2, s_3 \in S, (\tau(s_1, s_2) \land \tau(s_1, s_3)) \Rightarrow (s_2 = s_3) \)) dynamic discrete systems.

A program as defined in Section 10.3 defines a total and deterministic discrete dynamic system. Moreover, the entry states are exogenous (\( \forall s_1, s_2 \in S, \tau(s_1, s_2) \Rightarrow -(\varphi(s_2)) \)), the exit states are stable (\( \forall s_1, s_2 \in S, (\varphi(s_1) \land \tau(s_1, s_2)) \Rightarrow (s_1 = s_2) \)), and the system is without error recovery (\( \forall s_1, s_2 \in S, (\varphi(s_1) \land \tau(s_1, s_2)) \Rightarrow \varphi(s_2) \)).

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

10.4.2. Fixed Point Theorems for Isotone and Continuous Operators on a Complete Lattice

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

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

An operator \( f \) on \( L \) is strict if \( f(\bot) = \bot \), and isotone iff (\( \forall a, b \in L, (a \equiv b) \Rightarrow (f(a) \equiv f(b)) \)). \( a \in L \) 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 \( L(\equiv, \bot, \top, \cup, \cap) \) is a (nonempty) complete lattice with partial ordering \( \equiv \). The least fixed point of \( f \), in symbols \( \text{lfp}(f) \), is \( \bigwedge \{x \in L: f(x) \equiv x \} \). Dually the greatest fixed point of \( f \), in symbols \( \text{gfp}(f) \) is \( \bigvee \{x \in L: x \equiv f(x) \} \). An element \( a \) of \( L \) such that \( a \equiv f(a) \) (respectively \( f(a) \equiv a \)) 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 (\( \forall x \in L, (f(x) \equiv x) \Rightarrow (\text{lfp}(f) \equiv x) \)). The Dual Recursion Induction Principle is (\( \forall x \in L, (x \equiv f(x)) \Rightarrow (x \equiv \text{gfp}(f)) \)).

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

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

A poset \(L(\subseteq)\) is said to satisfy the ascending chain condition if any increasing chain terminates, that is if \(x_i \in L, i = 0, 1, 2, \ldots\), and \(x_0 \subseteq x_1 \subseteq \ldots \subseteq x_n \subseteq \ldots\), then for some \(m\) we have \(x_m = x_{m+1} = \ldots\). An operator \(f\) on \(L(\subseteq, \perp, \top, \cup, \cap)\) which is semi-\(\cup\)-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-\(\cup\)-continuous. Also an operator \(f\) on a complete lattice \(L\) which is a complete-

Dual results hold for decreasing chains, semi-\(\cap\)-continuous operators, descending chain conditions, and complete-\(\cap\)-morphisms.

Suppose \(L(\subseteq, \perp, \top, \cup, \cap), L'(\subseteq', \perp', \top', \cup', \cap')\) are complete lattices and we have the commuting diagram of isotope functions shown in Fig. 10.2 whose \(\delta\) is strict \((\delta(M) = 1)\) and semi-\(\cap\)-continuous. Then
operator on a uniquely complemented complete lattice $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ then $\lambda x.[-f(-x)]$ is an isotope operator on $L$, $\text{gfp}(f) = -\text{lfp}(\lambda x.[-f(-x)])$.

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

$$X = F(X)$$

which can be detailed as

$$X_j = F(X_1, \ldots, X_n) \quad j = 1, \ldots, n$$

has a least solution which is the least upper bound of the sequence $\{X^i; i \geq 0\}$ where $X^0 = \langle \bot, \ldots, \bot \rangle$ and $X^{i+1} = F(X^i)$, which can be detailed as:

$$X^{i+1}_j = F(X^i_1, \ldots, X^i_n) \quad j = 1, \ldots, n$$

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

More precisely [Cous77b, Cous77c] if $F$ is the least upper bound of any chaotic iteration sequence $\{X^i; i \geq 0\}$ where $X^0 = \langle \bot, \ldots, \bot \rangle$ and

$$X^{i+1}_j = F(X^i_1, \ldots, X^i_n) \quad \text{if } j \in J_i,$$

$$X^{i+1}_j = X^i_j \quad \text{if } j \notin J_i$$
In words, every exit state which is a descendant of an entry state satisfying $\phi$ must satisfy $\Psi$. The question of termination is not involved.

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

**Theorem 10-4.**

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

2. $\forall \theta \in ((S \times S) \rightarrow B)$, $\text{post}(\theta)$ is a strict complete $\lor$-morphism. $\forall \beta \in (S \rightarrow B), \lambda \theta.[\text{post}(\theta)(\beta)]$ is a strict complete
\((\text{post}(\tau)i) \Rightarrow i\)) we infer \((\text{iff}(\lambda x.[(\tau \land \phi) \lor \text{post}(\tau)(x)]) \Rightarrow i)\). It follows from Theorem 10-4, Part 3 that \((\nu \land \text{post}(\tau^*)((\tau \land \phi))) \Rightarrow (\nu \land \iota) \Rightarrow \Psi\). The method is sound [ClaE77].

Reciprocally, if \(\pi\) is partially correct with respect to \(\delta\). \(\Psi\) then this can
post(\theta^{-1}) is a strict complete \lor-morphism. \forall \beta \in (S \rightarrow S), \lambda \beta. \text{pre} (\theta)(\beta) = \lambda \theta. \text{post}(\theta^{-1})(\beta) is a strict complete \lor-morphism. Also

\text{pre}(\tau^*)(\beta) = \text{post}(\text{pre}(\tau)^{-1})(\beta) = \text{post}(\text{pre}(\tau^{-1})^*)(\beta) = \bigvee_{\alpha \in \mathbb{S}} \text{pre}(\tau^*)(\beta)

= \lfp(\lambda \alpha. [\beta \lor \text{post}(\tau^{-1})(\alpha)]) = \lfp(\lambda \alpha. [\beta \lor \text{pre}(\tau)(\alpha)])

10-4.5. Characterization of the States of a Total Deterministic System Which Do Not Lead to an Error as a Greatest Fixed Point

The entry states which are the origins of correctly terminating or diverging execution paths of a deterministic program \pi(S, \tau, \nu, \nu_0, \nu_e) are those which do not lead to a run-time error. They are characterized by \nu_e \land \neg \text{pre}(\tau^*)(\nu_e).

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

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

Proof. \neg \text{pre}(\tau^*)(\beta) = \neg \lfp(\lambda \alpha. [\beta \lor \text{pre}(\tau)(\alpha)]) = \neg \lfp(\lambda \alpha. [\neg \beta \land \text{pre}(\tau)(\neg \alpha)])

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

= \lambda x, [\neg \neg \alpha(\bar{\tau}(s_1))] = \lambda x, [\alpha(\bar{\tau}(s_1))] = \text{pre}(\bar{\tau})(\alpha).

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

Given a total deterministic system \pi(S, \tau, \nu, \nu_0, \nu_e) we have established that the analysis of the behavior of this system can be carried out by solving fixed point equations as follows:

Theorem 10-9.

1. The set of descendants of the entry states satisfying an entry condition \phi \in (S \rightarrow B) is characterized by:

   \text{post}(\tau^*)(\nu_e \land \phi) = \lfp(\lambda \alpha. [\nu_e \land \phi \lor \text{post}(\tau)(\alpha)])

2. The set of descendants of the exit states satisfying an exit condition \Psi \in (S \rightarrow B) is characterized by:

   \text{pre}(\tau^*)(\nu_e \land \Psi) = \lfp(\lambda \alpha. [\nu_e \land \Psi \lor \text{pre}(\tau)(\alpha)])

3. The set of states leading to an error is characterized by:

   \text{pre}(\tau^*)(\nu_e) = \lfp(\lambda \alpha. [\nu_e \lor \text{pre}(\tau)(\alpha)])
4. The set of states which do not lead to an error (i.e., cause the system either to properly terminate or to diverge) is characterized by:

$$-\text{pre}(\tau^*)(\nu_e) = \text{gfp}(\lambda x.[-\nu_e \land \text{pre}(\tau)(x)])$$
4. If $\beta \Rightarrow \text{pre}(\theta)(\gamma)$, then by isotony $\text{post}(\theta)(\beta) \Rightarrow \text{post}(\theta)$
   $(\text{pre}(\theta)(\gamma)) \Rightarrow \gamma$. If $(\text{post}(\theta)(\beta) \Rightarrow \gamma)$, then by isotony $\beta \Rightarrow
   \text{pre}(\theta)(\text{post}(\theta)(\beta)) \Rightarrow \text{pre}(\theta)(\gamma)$.
   $\text{post}(\theta)(\beta) = \lambda \gamma : \text{post}(\theta)(\beta) \Rightarrow \gamma = \lambda \gamma : \beta \Rightarrow \text{pre}(\theta)(\gamma)$.
   $\text{pre}(\theta)(\beta) = \lambda \gamma : \gamma \Rightarrow \text{post}(\theta)(\beta)$
   $= \lambda \gamma : \text{post}(\theta)(\gamma) \Rightarrow \beta$. ■

Example 10-12. According to Theorem 10-11.4, Floyd-Naur’s method for proving the partial correctness of $\pi$ with respect to $\phi$, $\Psi$ which consists in
   guessing an assertion $i$ and showing that $(((\nu, \land \phi) \Rightarrow i) \land (\text{post}(i) \Rightarrow i) 
   \land ((\nu, \land i) \Rightarrow \Psi))$ is equivalent to Hoare’s method [Hoar69], which consists of
   guessing an assertion $i$ and showing that $(((\nu, \land \phi) \land (i \Rightarrow \text{pre}(\pi)(i)) \land
   ((\nu, \land i) \Rightarrow \Psi))$.

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

Theorem 10-13.

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

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

Example 10-14. A total correctness proof of a program $\pi$ with respect to $\phi$, $\Psi$ consists in showing that $(((\nu, \land \phi) \Rightarrow \text{pre}(\pi)^*(\nu, \land \Psi))$, that is to say $(\nu, \land \phi) \Rightarrow \text{lp}(\lambda \alpha.[(\nu, \land \Psi) \lor \text{pre}(\pi)(\alpha)])$. Equivalently, using $\text{post}$, one can
   show that: $\forall \delta \in S, (\nu, \delta) \land \phi(\delta) \Rightarrow (\exists s_2 \in S: \nu(s_2) \land \Psi(s_2) \land \text{lp}(\lambda \alpha.[\lambda s.(s = \delta) \lor \text{post}(\pi)(\alpha)])(s_2))$ More generally we have:
   $\text{post}(\pi^*)(\beta) = \lambda \delta \exists s_1 \in S: \beta(s_1) \land \text{lp}(\lambda \alpha.[\lambda s.(s = \delta) \lor \text{pre}(\pi)(\alpha)])(s_1)$
   $\text{pre}(\pi^*)(\beta) = \lambda \delta \exists s_2 \in S: \beta(s_2) \land \text{lp}(\lambda \alpha.[\lambda s.(s = \delta) \lor \text{post}(\pi)(\alpha)])(s_2)$

10-4.8. Partitioned Dynamic Discrete Systems

A dynamic discrete system $(S, \tau, \nu, \nu_a, \nu_a)$ is said to be partitioned if
   there exist $n \geq 1, U_1, \ldots, U_a, t_1, \ldots, t_a$ such that $\forall i \in [1, n], t_i$ is a partial
   one-to-one map from $S$ onto $U_i$ and $\{t_i^{-1}(U_i): i \in [1, n]\}$ is a partition of $S$,
   (therefore $S = \bigcup_{i=1}^n t_i^{-1}(U_i)$ and every $s \in S$ is an element of exactly one
   $t_i^{-1}(U_i)$).
When studying the behavior of a partitioned system, the equations \( \alpha = \beta \equiv \text{post}(\tau)(\alpha) \) or \( \alpha = \beta \equiv \text{pre}(\tau)(\alpha) \) can be replaced by systems of equations defined as follows. Let us define: \( \forall i \in [1,n], \sigma_i \equiv ((S \rightarrow B) \rightarrow (U_i \rightarrow B)), \sigma_i = \lambda \beta. [\beta \circ i^{-1}], \sigma_i^{-1} = \lambda \beta. [\lambda \text{x}. [\epsilon \in i^{-1}(U_i) \land \beta(\epsilon)]], \sigma \equiv ((S \rightarrow B) \rightarrow (\prod_{i=1}^n (U_i \rightarrow B)), \sigma = \lambda \beta. \left( \prod_{i=1}^n \sigma_i(\beta) \right) = \lambda \beta. \langle \sigma_i(\beta), \ldots, \sigma_n(\beta) \rangle. \sigma \) is a strict isomorphism from \( (S \rightarrow B) \) onto \( \prod_{i=1}^n (U_i \rightarrow B) \). Its inverse is \( \sigma^{-1} = \lambda\langle \beta_1, \ldots, \beta_n \rangle. [\prod_{i=1}^n \sigma_i^{-1}(\beta_i)] \).

For any isotone operator \( f \) on \( (S \rightarrow B) \), the diagram in Fig. 10-4 commutes, so that the sets of pre-fixed points, fixed points and post-fixed points of \( f \) coincide (up to the isomorphism \( \sigma \)) with the pre-solutions, solutions and post-solutions to the direct decomposition of \( \alpha = f(\alpha) \) on \( \prod_{i=1}^n (U_i \rightarrow B) \) which is the system of equations:

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

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

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

\[
\lambda\langle \alpha_1, \ldots, \alpha_n \rangle. [\sigma_i(\beta) \equiv (\bigvee_{\epsilon \in \text{pred}(\tau)(\alpha_i)} \text{post}(\tau)(\alpha_i))]
\]

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

\[
\lambda\langle \alpha_1, \ldots, \alpha_n \rangle. [\sigma_i(\beta) \equiv (\bigvee_{\epsilon \in \text{succ}(\tau)(\alpha_i)} \text{pre}(\tau)(\alpha_i))]
\]
where
\[ \tau_{ij} \in (\langle U_i \times U_j \rangle \rightarrow B) \]  
\[ \text{pred}_i = \lambda i. [j \in [1, n]: (\exists s_1 \in U_j, \exists s_2 \in U_i : \nu i (s_1, s_2))] \]  
\[ \text{succ}_i = \lambda i. [j \in [1, n]: (\exists s_1 \in U_j, \exists s_2 \in U_i : \nu j (s_1, s_2))] \]

**Proof.**

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

Moreover

\[ \text{post}(\nu)(\sigma_j^{-1}(a_j)) \circ \nu_i^{-1} = \lambda s_2. [\exists s_1 \in S : \sigma_j^{-1}(a_j)(s_1) \land \nu(s_1, s_2)] \]
\[ = \lambda s_2. [\exists s_1 \in U_j : a_j(s_1) \land \nu(s_1, s_2)] \]
\[ = \lambda s_2. [\exists s_1 \in U_j : a_j(s_1) \land \nu j (s_1, s_2)] \]
\[ = \text{post}(\nu j)(a_j). \]

Therefore

\[ \bigvee_{j=1}^{n} \left( \text{post}(\nu)(\sigma_j^{-1}(a_j)) \circ \nu_i^{-1} \right) = \bigvee_{j \neq \text{pred}_i (l)} \text{post}(\nu j)(a_j) \]

since \((j \neq \text{pred}_i (l))\) implies \(\forall s_1, s_2, \neg \nu j (s_1, s_2). \) Also \( \text{pre}(\nu) = \text{post}(\nu^{-1}), \)
\((\nu j)^{-1} = \nu j, \) and \(\text{succ}_i = \text{pred}_i. \) \[\blacksquare\]

### 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 \( \langle \tau, S, v_\nu, v_\omega \rangle \) where \( S = ([1, n] \times U), \forall i \in [1, n], U_i = U, \nu_i = \lambda \langle c, m \rangle, m, \nu_i^{-1} = \lambda m. \langle i, m \rangle. \) 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 \( P = F_\nu (\phi)(P) \) associated with a program \( \pi \) and an entry specification \( \phi \in (U \rightarrow B) \) is the direct decomposition of \( \alpha = (\nu \land \sigma_i^{-1}(\phi)) \lor \text{post}(\nu)(\alpha) \) on \((U \rightarrow B)^*; \) that is,

\[ P_i = \sigma_i (\nu \land \sigma_i^{-1}(\phi)) \lor \bigvee_{j \neq \text{pred}_i (l)} \text{post}(\nu j)(P_j) \]

\( i = 1, \ldots, n \)
From the abstract syntax and operational semantics of programs we derive a set of construction rules for obtaining this system of equations from the program text:

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

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

$$= \bigvee_{j \in \text{pred}_c(i)} \lambda m_1. [\exists m_1 \in U: P_j(m_1) \land (\bar{e}(\langle j, m_1 \rangle) = \langle i, m_1 \rangle)]$$

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

$$\lambda m. [\exists m_\epsilon \in U: P_i(m_\epsilon) \land \bar{e}(\langle j, m_\epsilon \rangle) = \langle i, m_\epsilon \rangle]$$
where \( \forall f \in I(U), \text{post}(f) = \lambda P.\{ \lambda m. [ \exists m' \in U: P(m') \land m = f(m')] \}; \quad \forall p \in I(U), \text{post}(p) = \lambda P.\{ \lambda m. [P(m) \land m \in \text{dom}(p) \land p(m)] \}; \quad \text{at}(x) \) is the set of program points \( j \) preceding an assignment \( v = f(\varphi) \) or a test \( p(\varphi) \) and \( \text{expr}(j) \) is the corresponding \( f \) or \( p \).

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

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

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

\[
P_i = \sigma_i(\beta) \land \bigvee_{j \in \text{succ}(i)} \lambda m_i. [\exists m_{j+1} \in U: (\uparrow L(m_i, i) = \langle m_{j+1}, j \rangle \land P(m_{j+1})]
\]

\[i = 1, \ldots, n\]

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

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

\[
P_i = \bigvee_{j \in \text{succ}(i)} \text{pre}((L(i, j))(P)) \quad i \in ([1, n] - [\xi, \omega])
\]

\[P_\alpha = \Psi \land P_\omega\]

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

**Theorem 10-19.**

1. The direct decomposition \( P = B(P) \) of \( \alpha = (v \land \sigma^{-1}_e(\Psi)) \land \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is

\[
P_i = B_\alpha(\Psi)(P) \land \text{error}(i) \quad \text{for } i \in ([1, n] - [\xi, \omega])
\]

\[P_\alpha = \Psi \land P_\omega \land P_\xi = P_\xi \]

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

3. The direct decomposition \( P = B(P) \) of \( \alpha = \nu_\chi \lor \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is
\[
P_i = B_s(\lambda m.[\text{false}])((P)) \lor \text{error}(i) \quad \text{for } i \in ([1, n] - \{\omega, \xi\})
P_\omega = P_\omega
\]
\[
P_\xi = \lambda m.[\text{true}]
\]
The least solution to the above system of equations is equal to the least solution to
\[
P_i = B_s(\lambda m.[\text{false}])((Q)) \lor \lambda m.[m \notin \text{dom}(\text{expr}(i))] \quad \text{for } i \in ([1, n] - \{\omega, \xi\})
P_\omega = \lambda m.[\text{false}]
P_\xi = \lambda m.[\text{true}]
\]
where \( Q \) stands for \( P_i \) when \( i \in ([1, n] - \{\omega, \xi\}) \), \( Q_\omega \) stands for \( \lambda m.[\text{false}] \) and \( Q_\xi \) stands for \( \lambda m.[\text{true}] \).

4. The direct decomposition of \( \alpha = \neg \nu_m \land \neg \nu_\xi \land \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is
\[
P_i = B_s(\lambda m.[\text{false}])((P)) \quad \text{for } i \in ([1, n] - \{\xi\})
P_\omega = \lambda m.[\text{false}]
\]

5. The direct decomposition of \( \alpha = \lambda s.[s = \xi] \lor \text{pre}(x)(\alpha) \) on \( (U \rightarrow B)^* \) is
\[
P_i = \lambda m.[i, m'] = \xi] \lor B_s(\lambda m.[\text{false}])((P)) \lor \text{error}(i) \quad \text{for } i \in ([1, n] - \{\omega, \xi\})
P_\omega = \lambda m.[\sigma, m'] = \xi] \lor P_\omega
\]
\[
P_\xi = \lambda m.[\xi, m'] = \xi] \lor P_\xi
\]

10-5.3. Analysis of the Behavior of a Program

In order to illustrate the application of Theorem 10-9 to the analysis of the behavior of a program, we consider the introduction of variables...
while $x \geq 1000$ do
\[ x := x + y; \]
\[ \text{od;} \]

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

**10.5.3.1. Forward semantic analysis**

The system $P = F_s(\bar{\phi})(P)$ (where $F_s(\bar{\phi}) \in ((P \rightarrow B)^s \rightarrow (P \rightarrow B)^s)$) of forward semantic equations associated with the above program $\pi$ and an entry specification $\bar{\phi} \in (P \rightarrow B)$ is the following:

$P_1 = \bar{\phi}$

$P_2 = \lambda \langle x, y \rangle. [\forall (x \in I) \land (x \geq 1000)]$

$P_3 = \lambda \langle x, y \rangle. [\exists x' \in I: P_2(x', y) \land (x' + y) \in I] \land (x \neq x' + y)]$

$P_4 = \lambda \langle x, y \rangle. [\forall (x \in I) \land (x < 1000)]$

$P_5 = \lambda \langle x, y \rangle. [\forall (x \neq 1) \land (x \neq y)]$

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

$\sigma_s(v_s) \cap \text{pre}(\tau^*)(v_m)$

$= \sigma_s(\lambda s. [\exists x \in S: v_s(x) \land \text{post}(\tau^*)(\lambda s. [v_m = \overline{m}])])$

(\text{Theorem 10-13})

$= \lambda \overline{m}. [\exists x \in S: v_s(x) \land \text{post}(\tau^*)(\lambda s. [v_m = \overline{m}])]$

(\text{Theorem 10-4(3)})

$= \lambda \overline{m}. [\exists x \in S: v_s(x) \land \sigma^{-1}(\text{lfp}(F_s(\lambda m. [v_m = \overline{m}])))]$

(\text{Theorem 10-17})
The least fixed point $P^n$ of $F((x = \bar{x}) \land (y = \bar{y}))$ is computed iteratively using a chaotic iteration sequence as follows:

\[ P_i^0 = \lambda(x, y).[\text{false}] \quad i = 1, \ldots, 4, \xi \]

\[ P_i^1 = \lambda(x, y).[(x = \bar{x}) \land (y = \bar{y})] \quad \text{where } \langle \bar{x}, \bar{y} \rangle \in P^2 \]

\[ P_i^2 = \lambda(x, y).[((P_i^1 \lor P_i^2)(x, y) \land (x \in I) \land (x \geq 1000)) \lor \text{false}] \]

\[ \vdots \quad \lambda(x, y).[(x = \bar{x}) \land (y = \bar{y})] \land (x \geq 1000) \]

\[ \text{where } \langle \bar{x}, \bar{y} \rangle \in P^2 \]
(It is worth noting that the use of the symbolic entry condition $\lambda^<x, y^>.[(x = \tilde{x}) \land (y = \tilde{y})]$ and of the above iteration strategy corresponds to a symbolic execution of the program loop [Hant76] with the difference that all possible execution paths are considered simultaneously and the induction step as well as the passage to the limit deal with infinite paths.) The remaining components of $\text{fpp}(\mathcal{P}_s(\lambda^<x, y^>.[(x = \tilde{x}) \land (y = \tilde{y})]))$ are:

$$P^p_1 = \lambda^<x, y^>.[(x = \tilde{x}) \land (y = \tilde{y})]$$

$$P^p_2 = \lambda^<x, y^>.[\exists x' \in I: P^p_1(x', y) \land ((x' + y) \in I) \land (x = x' + y)]$$

$$= \lambda^<x, y^>.[\exists j \geq 1: (1000 \leq \text{min}(\tilde{x}, x - \tilde{y})) \land (\text{max}(\tilde{x}, x) \leq b) \land (x = \tilde{x} + j\tilde{y}) \land (y = \tilde{y})]$$

$$P^p_3 = \lambda^<x, y^>.[(P^p_1 \lor P^p_2)(x, y) \land (x \in I) \land (x < 1000)]$$

$$= \lambda^<x, y^>.[[((\tilde{x} < 1000) \land (x = \tilde{x}) \land (x = \tilde{y})) \lor ((\tilde{x} \geq 1000) \land (\tilde{y} < 0) \land (x = \tilde{x} + (((\tilde{x} - 1000) \text{ div } |\tilde{y}|) + 1)\tilde{y}) \land (y = \tilde{y}))]]$$
that is,

\[ \downarrow (\exists \ y. \ u \in I \cdot \text{prop}(y, u)) \land \neg (\exists \ y. \ u \in I \cdot \text{prop}(y, u)) \]
We now recommence the semantic analysis of this program, but this time using backward equations.

10.5.3.2. Backward semantic analysis

The system $P = B_5(\Psi)(P)$ (where $B_5(\Psi) \in ((P \rightarrow B)^4 \rightarrow (P \rightarrow B)^4)$) of backward semantic equations associated with the example program $\pi$ and an exit specification $\Psi \in (P \rightarrow B)$ is the following:

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

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

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

$$P_4 = \Psi$$

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

$$\sigma(v_e \land \text{pre}(\pi^e)(v_\text{in}))$$

$$= \sigma(v_e \land \text{gfp}(\lambda x. [v_\text{in} \lor \text{pre}(\pi(x))])) \quad \text{[Theorem 10-9(2)]}$$

$$= \sigma(v_e \land \sigma^{-1}(\text{gfp}(B_5(\lambda m. [\text{true}])))) \quad \text{[Theorem 10-19(1)]}$$

$$= \sigma(v_e \land \sigma^{-1}(\text{gfp}(B_5(\lambda m. [\text{true}]))) \circ i_\text{in})$$

$$= \text{gfp}(B_5(\lambda m. [\text{true}]))$$

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

$$P^e_1 = \lambda \langle x, y \rangle. ((x < 1000) \lor (y < 0))$$

$$P^e_2 = \lambda \langle x, y \rangle. (((x + y) \in I) \land ((x + y < 1000) \lor (y < 0)))$$

$$P^e_3 = \lambda \langle x, y \rangle. ((x < 1000) \lor (y < 0))$$

$$P^e_4 = \lambda \langle x, y \rangle. [\text{true}]$$

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

$$\sigma(v_e \land \neg \text{pre}(\pi^e)(v_2))$$

$$= \sigma(v_e \land \text{gfp}(\lambda x. [\neg v_2 \land \text{pre}(\pi(x))])) \quad \text{[Theorem 10-9(3)]}$$

$$= \sigma(v_e \land \sigma^{-1}(\text{gfp}(B_5(\lambda m. [\text{true}])))) \quad \text{[Theorem 10-19(2)]}$$

$$= \text{gfp}(B_5(\lambda m. [\text{true}])), $$
The greatest fixed point $Q^\omega$ of the above system of equations where $\Psi = \lambda \langle x, y \rangle. \{true\}$ can be computed iteratively starting from $Q^0 = \lambda \langle x, y \rangle. \{true\}$, $i = 1, \ldots, 4$, inventing the general term of a chaotic iteration sequence, and passing to the limit:

$$Q^\omega = \lambda \langle x, y \rangle. \{y < 0 \land \forall r \leq 1000\}$$
\[ P_1 = \lambda \langle x, y \rangle.[(\langle 1, \langle x, y \rangle \rangle = \bar{s}) \lor (x \in I \land x \geq 1000 \land P_2(x, y)) \lor (x \in I \land x \leq 1000 \land P_4(x, y)) \lor (P_4(x, y) \land x \notin I)] \]
\[ P_2 = \lambda \langle x, y \rangle.[(\langle 2, \langle x, y \rangle \rangle = \bar{s}) \lor (x + y \in I \land P_3(x + y, y)) \lor (P_4(x, y) \land x + y \notin I)] \]
\[ P_3 = \lambda \langle x, y \rangle.[(\langle 3, \langle x, y \rangle \rangle = \bar{s}) \lor (x \in I \land x \geq 1000 \land P_4(x, y)) \lor (P_4(x, y) \land x \notin I)] \]
\[ P_4 = \lambda \langle x, y \rangle.[(\langle 4, \langle x, y \rangle \rangle = \bar{s}) \lor P_4(x, y)] \]
\[ P_5 = \lambda \langle x, y \rangle.[(\langle 5, \langle x, y \rangle \rangle = \bar{s}) \lor P_4(x, y)] \]

If \( P^w(\bar{s}) \) denotes \( \text{ifp}(\sigma \circ \lambda \alpha[\lambda s.s = \bar{s}] \lor \text{pre}(r)(\alpha)) \circ \sigma^{-1} \), we determine that
\[ P^w(\bar{s}) = \lambda \langle x, y \rangle.[(\langle 1, \langle x, y \rangle \rangle = \bar{s}) \lor (\exists j \geq 0: (\forall i \in [0, j], 1000 \leq x + iy \leq b) \land (\langle 2, \langle x + jy, y \rangle \rangle = \bar{s})) \lor (\exists j \geq 1: (\forall i \in [0, j - 1], 1000 \leq x + iy \leq b) \land (x + jy \in I) \land (\langle 3, \langle x + jy, y \rangle \rangle = \bar{s})) \lor (\exists j \geq 0: (\forall i \in [0, j - 1], 1000 \leq x + iy \leq b) \land (x + jy \in I) \land (x + jy < 1000) \land (\langle 4, \langle x + jy, y \rangle \rangle = \bar{s})) \lor (\exists j \geq 1: (\forall i \in [0, j - 1], 1000 \leq x + jy \leq b) \land (x + jy \notin I) \land (\langle 5, \langle x + (j - 1)y, y \rangle \rangle = \bar{s}))]]
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.
vergence when necessary. We will consider only the upper approximation of least fixed points since the other cases are duals to it.

10-6.1. Considering Simplified Equations

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

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

**Example 10-21.** Let \( x \) be a program over integer variables \( X \). . . . . .
The upper approximation $\mathcal{L}_1$ of $L_1$ can be defined as

$$\mathcal{L}_1 = \{\langle s_1, \ldots, s_n > : (\forall j \in [1, n], s_j \in L_2 - \{\bot\}) \cup \{\langle \bot, \ldots, \bot >\} \}$$

and $\gamma_1 \in (\mathcal{L}_1 \rightarrow \mathcal{L}_1)$ is such that

$$\gamma_1 = \lambda \langle s_1, \ldots, s_n \rangle \cdot \lambda \langle X_1, \ldots, X_n \rangle \cdot \left[ \bigwedge_{j=1}^n \gamma_3(s_j)(X_j) \right]$$

Intuitively, $\gamma$ gives the meaning $\gamma(\tilde{x})$ of the elements $\tilde{x}$ of $\mathcal{L}$. $\gamma$ is assumed to be one-to-one so that no two distinct elements of $\mathcal{L}$ can have the same meaning. The hypothesis that $\gamma$ is a complete $\sqcap$-morphism implies that for
We can now write a program, which, given a program text \( x = f(x) \), constructs the associated system of approximate equations \( x = \varphi(x) \). Whenever \( \overline{L} \) satisfies the ascending chain condition, we can write a program for computing the least fixed point \( lfp(f) \) of any isotone operator \( f \) on \( \overline{L} \). Starting from the infimum of \( \overline{L} \), the algorithm can proceed iteratively using any efficient chaotic strategy until the iterates stabilize.

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

\begin{verbatim}
[1] while \( x \geq 1000 \) do
[2] \hspace{1em} \( x := x + y \); \\
[3] \hspace{1em} \textbf{od};
[4]
\end{verbatim}

is the following:

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

where \( \text{smash}(\langle x, y \rangle) = \langle \text{if } x = \bot \text{ or } y = \bot \text{ then } \bot \rangle \text{ else } \langle x, y \rangle \). Taking \( \phi = \langle \bot, \bot \rangle \), a chaotic iterative resolution corresponding to a symbolic execution of the program is the following:
10.6.2. Speeding Up the Convergence of Chaotic Iterations

Using Extrapolation Techniques

The results are approximate but not useless since, e.g., they prove that no
overflow can occur when the entry specification ($\alpha \geq 0 \land y \leq 0$) is true.

When $T$ is a vector of $s$ elements, the regular Runge-Kutta algorithms
inhibit linear
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 \( V \) be a widening operator. The limit \( u \) of the sequence

\[
\begin{align*}
x^0 &= d \\
x^{n+1} &= x^n \forall (n + 1) \bar{f}(x^n) \quad \text{if} \quad \neg(\bar{f}(x^n) \sqsubseteq x^n) \\
x^{n+1} &= x^n \quad \text{if} \quad \bar{f}(x^n) \sqsubseteq x^n
\end{align*}
\]

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 \( \forall j > 0, \forall (j) = \perp \), in which case \( x^* = lfp(\bar{f}) \). 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 \( j > 0 \):

\[
\forall x \in \bar{L}_2, \perp \forall (j) x = x \forall (j) \perp = x
\]

\[
[l_1, u_1] \forall (j) [l_2, u_2]
\]

\[
= [\text{if } 0 \leq l_1 < l_1 \text{ then } 0 \text{ elseif } l_2 < l_1 \text{ then } -b - 1 \text{ else } l_1, f_1,
\]

\[
\text{if } u_1 < u_2 \leq 0 \text{ then } 0 \text{ elseif } u_1 < u_2 \text{ then } b \text{ else } u_1, f_1]
\]

A widening \( V_1 \) on \( \bar{L}_1 \) is obtained by applying \( V_2 \) componentwise:

\[
\langle i_1, \ldots, i_n \rangle \forall (j) \langle i'_1, \ldots, i'_n \rangle = \langle i_1 \forall (j) i'_1, \ldots, i_n \forall (j) i'_n \rangle
\]

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 \( \odot \).
while (10 ≤ x) ∧ (x ≤ 100) do
x := x + y;
end;

is the following:
\[
\begin{array}{l}
\langle x_1, y_1 \rangle = \phi \\
\langle x_2, y_2 \rangle = \text{smash}(\langle (x_1 \sqcup x_3) \cap [10, 100], (y_1 \sqcup y_3) \rangle) \\
\langle x_3, y_3 \rangle = \text{smash}(\langle x_2 + y_2, y_3 \rangle) \\
\langle x_4, y_4 \rangle = \text{smash}(\langle (x_1 \sqcup x_3) \cap (-b - 1, 9), (y_1 \sqcup y_3) \rangle) \\
\langle x_5, y_5 \rangle = \text{if} \langle x_2, y_2 \rangle = \langle \bot, \bot \rangle \text{ then } \langle \bot, \bot \rangle \text{ else underflow}(x_2, y_2) \text{ fi}
\end{array}
\]
where
\[\forall x \in \mathcal{L}_2, \bot \sqcup x = x \sqcup \bot = x,\]
\[\bot \sqcap x = x \sqcap \bot = \bot,\]
\[\bot + x = x + \bot = \bot,\]
\[\mathcal{L}_1 \sqcup \mathcal{L}_2 = [\min(\mathcal{L}_1, \mathcal{L}_2), \max(\mathcal{L}_1, \mathcal{L}_2)]\]
\[\mathcal{L}_1 \sqcap \mathcal{L}_2 = \text{if } \max(\mathcal{L}_1, \mathcal{L}_2) \leq \min(\mathcal{L}_1, \mathcal{L}_2) \text{ then } [\max(\mathcal{L}_1, \mathcal{L}_2), \min(\mathcal{L}_1, \mathcal{L}_2)] \text{ else } \bot \]
\[\mathcal{L}_1 + \mathcal{L}_2 = [\mathcal{L}_1 + \mathcal{L}_2, \mathcal{L}_1 + \mathcal{L}_2] \sqcap [-b - 1, b]\]
\[\text{overflow}(\mathcal{L}_1, \mathcal{L}_2) = \text{if } (u_1 \geq 1) \land (u_2 \geq 1) \text{ then } ((\mathcal{L}_1, \mathcal{L}_1) \sqcap [b - u_2 + 1, b]), (\mathcal{L}_2, \mathcal{L}_2) \sqcap [b - u_1 + 1, b]) \text{ else } \langle \bot, \bot \rangle \]
\[\text{underflow}(\mathcal{L}_1, \mathcal{L}_2) = \text{if } (l_1 \leq -1) \land (l_2 \leq -1) \text{ then } ((\mathcal{L}_1, \mathcal{L}_1) \sqcap [-b - 1, -b - 2 - l_2]), (\mathcal{L}_2, \mathcal{L}_2) \sqcap [-b - 1, -b - 2 - l_2]) \text{ else } \langle \bot, \bot \rangle \]
\[\text{smash}(x, y) = \text{if } (x = \bot) \lor (y = \bot) \text{ then } \langle \bot, \bot \rangle \text{ else } \langle x, y \rangle \]

Taking \(\phi = \langle [9, 11], [-1, 1] \rangle\) the resolution uses the widening operators of Example 10-31 for the loop head node 2:
\[\langle x_i, y_i \rangle = \langle \bot, \bot \rangle \quad i = 1, \ldots, 4, \xi\]
\[\langle x_i, y_i \rangle = \phi = \langle [9, 11], [-1, 1] \rangle\]
\[\langle x_i, y_i \rangle = \langle x_i, y_i \rangle \lor \langle x_i, y_i \rangle \text{ smash}(\langle x_i \sqcup x_i \rangle \sqcap [10, 100], (y_i \sqcup y_i))\]
\[= \langle \bot \lor \langle x_i, y_i \rangle \rangle \lor \langle x_i, y_i \rangle \text{ smash}(\langle x_i \sqcup x_i \rangle \sqcap [10, 100], (y_i \sqcup y_i))\]
\[\langle x_i, y_i \rangle = \text{smash}(\langle x_i + y_i, y_i \rangle) = \langle [9, 12], [-1, 1] \rangle\]
\[\langle x_i, y_i \rangle = \langle x_i, y_i \rangle \text{ smash}(\langle x_i \sqcup x_i \rangle \sqcap [10, 100], (y_i \sqcup y_i))\]
\[\langle x_i, y_i \rangle = \langle x_i, y_i \rangle \lor \langle x_i, y_i \rangle \text{ smash}(\langle x_i \sqcup x_i \rangle \sqcap [10, 100], (y_i \sqcup y_i))\]
\[ \langle x_1, y_1 \rangle = smash(\langle ((x_1 \sqcup x_1) \cap [\neg b - 1, 9]), (y_1 \sqcup y_1) \rangle) \]
\[ = \langle 9, 9 \rangle, [\neg 1, \neg 1] \rangle \]
\[ \langle x_2, y_2 \rangle = if \langle x_2, y_2 \rangle = \langle \bot, \bot \rangle then \langle \bot, \bot \rangle else underflow(x_2, y_2) \]
\[ \text{\sqcup overflow}(x_2, y_2) fi \]
\[ = \langle \bot, \bot \rangle \sqcup \langle [b, b], [1, 1] \rangle = \langle [b, b], [1, 1] \rangle \]

The approximate result is
\[ \langle x_1, y_1 \rangle = \langle [9, 11], [-1, 1] \rangle \]
This narrowing operator attempts to improve the zero or infinite bounds \((-b - 1 \text{ and } b)\) which might have been too imprecisely extrapolated by the widening operator.

**Theorem 10-36.** Let \(u \in \tilde{L}\) be such that \((\text{lf}(\tilde{f}) \subseteq u)\) and \((\tilde{f}(u) \subseteq u)\). The decreasing chain \(x_0 = u, \ldots, x^* = x^{n-1} \Delta(u) \tilde{f}(x^{n-1}), \ldots\) is eventually stable. Moreover \(\forall k \geq 0, \text{lf}(\tilde{f}) \subseteq x^*\).

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

\[
\langle x_2, y_4 \rangle = \langle x_3, y_2 \rangle \text{ smash} (\langle (x_1 \sqcup x_3) \cap [10, 100], (y_2 \sqcup y_3) \rangle)
\]

\[
= \langle [10, b], \Delta_2(1) [10, 100], [-1, 1] \Delta_2(1) [-1, 1] \rangle
\]

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

\[
\langle x_1, y_1 \rangle = \text{ smash} (\langle x_1 + y_2, y_3 \rangle) = \langle [9, 101], [-1, 1] \rangle
\]

\[
\langle x_2, y_1 \rangle = \langle x_2, y_2 \rangle \text{ smash} (\langle (x_1 \sqcup x_4) \cap [10, 100], (y_2 \sqcup y_3) \rangle)
\]

\[
= \langle [10, 100], \Delta_2(2) [10, 100], [-1, 1] \Delta_2(2) [-1, 1] \rangle
\]

\[
= \langle [10, 100], [-1, 1] \rangle = \langle x_2, y_1 \rangle
\]

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

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

\[
\langle x_2, y_1 \rangle = \langle [10, 100], [-1, 1] \rangle
\]

\[
\langle x_2, y_2 \rangle = \langle [9, 101], [-1, 1] \rangle
\]

\[
\langle x_2, y_3 \rangle = \langle [9, 9], [-1, 1] \rangle
\]

\[
\langle x_3, y_2 \rangle = \langle \bot, \bot \rangle
\]

**10-6.3. Hierarchy of Approximate Program Analyses**

Let us give three examples of approximate analysis of the same program.

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

```plaintext
type table = array [1, 100] of integer;
procedure binary-search (var R: table; value k: integer;
result m: integer) =

var bi, bs: integer;
begin
bi := lb(R); bs := ub(R);
```
while $bi \leq bs$ do

\[ m := (bi + bs) \div 2; \]

if $k = R(m)$ then
\[ bi := bs + 1; \]

elsif $k < R(m)$ then
\[ bs := m - 1; \]

else
\[ bi := m + 1; \]

fi;

od;

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

end;

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

\[ P_1 = (bi > 0) \land (bs > 0) \]
\[ P_2 = (bi > 0) \land (bs > 0) \]
\[ P_3 = (bi > 0) \land (bs > 0) \land (m > 0) \]
\[ P_4 = (bi > 0) \land (bs > 0) \land (m > 0) \]
\[ P_5 = (bi > 0) \land (bs > 0) \land (m > 0) \]
\[ P_6 = (bi > 0) \land (bs > 0) \land (m > 0) \]
\[ P_7 = (bi > 0) \land (bs > 0) \land (m > 0) \]

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:

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

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.
be useful to look for linear equality or inequality relationships between the numerical variables of the program. This consists in approximating the characteristic set of a predicate $P$ by the convex-hull of this set, as shown in Fig. 10-7(d).

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

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

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

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

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

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

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

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

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

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

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

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

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

Notice that at program point [8] nothing is known about $m$. Contrary to the

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
sis of recursive programs are proposed by [Spil72, Alle74, Lome75, Rose79, Bart77a], whereas [Cous77a, Shar80] handle more ambitious analyses related to program verification.

Automatic methods for program analysis have numerous applications including type determination [Jone76, Kapl78a, Tene74b], gathering information for automatic data structure selection in very-high-level languages [Schw75b, Schw75c], detection of induction variables and strength reduction [Fong75, Fong76] or discovery of generalized common subexpressions [Fong77] for set-theoretic languages, determination of affine equality relationships among variables of a program [Karr76], detection of programming errors [Fosd76, Gill77], static array bound checking [Cous77a, Cous78, Germ78, Harr77a, Suzu77, Wels77], determination of linear [Cous78] or nonlinear [Berm76] invariant assertions, synthesis of resource invariants for concurrent programs [ClAE79], and so on.

ACKNOWLEDGMENT

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


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


BIBLIOGRAPHY


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


Klee52  KLEENE, STEPHEN COLE, Introduction to Metamathematics. New York:


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