Conference Record
of the
SIXTH ANNUAL ACM SYMPOSIUM ON
PRINCIPLES OF PROGRAMMING LANGUAGES

Papers Presented at the Symposium
San Antonio, Texas
January 29-31, 1979

Sponsored by the
ASSOCIATION FOR COMPUTING MACHINERY
SPECIAL INTEREST GROUP ON AUTOMATA AND COMPUTABILITY THEORY
SPECIAL INTEREST GROUP ON PROGRAMMING LANGUAGES
1. INTRODUCTION and SUMMARY

Semantic analysis of programs is essential in optimizing compilers and program verification systems. It encompasses data flow analysis, data type determination, generation of approximate invariant assertions, etc.

Several recent papers (among others Cousot & Cousot[77], Graham & Wegman[76], Kam & Ullman[76], Kildal[73], Rosen[76], Tarjan[76], Wegbreit[75]) have introduced abstract approaches to program analysis which are tantamount to the use of a program analysis framework \(A, \pi, Y\) where \(A\) is a lattice of (approximate) assertions, \(\pi\) is an (approximate) predicate transformer and \(Y\) is an often implicit function specifying the meaning of the elements of \(A\).

This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics.

Preliminary definitions are given in Section 2 concerning the merge over all paths and (least) fixpoint program-wide analysis methods. In Section 3 we briefly define the (forward and backward) deductive semantics of programs which is later used as a formal basis in order to prove the correctness of the approximate program analysis frameworks. Section 4 very shortly recall the main elements of the lattice theoretic approach to approximate semantic analysis of programs.

The design of a space of approximate assertions \(A\) is studied in Section 5. We first justify the very reasonable assumption that \(A\) must be chosen such that the exact invariant assertions of any program must have an upper approximation in \(A\) and that the approximate analysis of any program must be performed using a deterministic process. These assumptions are shown to imply that \(A\) is a Moore family, that the approximation operator (which defines the least upper approximation of any assertion) is an upper closure operator and that \(A\) is necessarily a complete lattice. We next show that the connection between a space of approximate assertions and a computer representation is naturally made using a pair of isotope adjoint functions. This type of connection between two complete lattices is related to Galois connections thus making available classical mathematical results. Additional results are proved, they hold when no two approximate assertions have the same meaning.

In Section 6 we study and exemplify various methods which can be used in order to define a space of approximate assertions or equivalently an approximation function. They include the characteristic of the least Moore family containing an arbitrary set of assertions, the construction of the least closure operator greater than or equal to an arbitrary approximation function, the definition of closure operators by composition, the definition of a space of approximate assertions by means of a complete join congruence relation or by means of a family of principal ideals.

Section 7 is dedicated to the design of the approximate predicate transformer induced by a space of approximate assertions. First we look for a reasonable definition of the correctness of approximate predicate transformers and show that a local correctness condition can be given which has to be verified for every type of elementary statement. This local correctness condition ensures that the (merge over all paths or fixpoint) global analysis of any program is correct. Since isotony is not required for approximate predicate transformers to be correct it is shown that non-isotone program analysis frameworks are manageable although it is later argued that the isotony hypothesis is natural. We next show that among all possible approximate predicate transformers which can be used with a given space of approximate assertions there exists a best one which provides the maximum information relative to a program-wide analysis method. The best approximate predicate transformer induced by a space of approximate assertions turns out to be isotone. Some interesting consequences of the existence of a best predicate transformer are examined. One is that we have in hand a formal specification of the programs which have to be written in order to implement a program analysis framework once a representation of the space of approximate assertions has been chosen. Examples are given, including ones where the semantics of programs is formalized using Hoare[71]'s sets of traces.

In Section 8 we show that a hierarchy of approximate analyses can be defined according to the fineness of the approximations specified by a program analysis framework. Some elements of the hierarchy are shortly exhibited and related to the relevant literature.

In Section 9 we consider global program analysis methods. The distinction between "distributive" and "non-distributive" program analysis frameworks is studied. It is shown that when the best approximate predicate transformer is considered the coincidence or not of the merge over all paths and least fixpoint global analyses of programs is a consequence of the choice of the space of approximate assertions. It is
shown that the space of approximate assertions can always be refined so that the merge over all paths analysis of a program can be defined by means of a least fixpoint of isotone equations.

Section 10 is devoted to the combination of program analysis frameworks. We study and exemplify how to perform the "sum", "product" and "power" of program analysis frameworks. It is shown that combined analyses lead to more accurate information than the conjunction of the corresponding separate analyses but this can only be achieved by a new design of the approximate predicate transformer induced by the combined program analysis frameworks.

2. PRELIMINARY DEFINITIONS

A program \( \pi \) is a pair \((\mathcal{V}, \mathcal{G})\) where \( \mathcal{G} \) is a program graph and \( \mathcal{V} \) is the universe in which the program variables take their values.

The set \( \mathcal{L} \) of elementary commands consists in elementary tests and elementary assignments:
\[ \mathcal{L} = \{ \text{true}, \text{false} \}. \]
An elementary test \( q \in \mathcal{L} \) is a total map from \( \text{dom}(q) \subseteq \mathcal{V} \) into \( \{ \text{true}, \text{false} \} \). An elementary assignment \( a \in \mathcal{L} \) is a total map from \( \text{dom}(a) \subseteq \mathcal{V} \) into \( \mathcal{V} \).

A program graph \( \mathcal{G} \) is a tuple \((n, E, n_0, n_e, C)\) where \( n \) is the number of vertices (therefore \( n \geq 1 \)), \( E \subseteq [1,n]^2 \) is the (non-empty) set of edges, \( n_0 \in [1,n] \) is the entry point, \( n_e \in [1,n] \) is the exit point and \( C \subseteq (E \to \mathcal{L}) \) defines the command \( C(\langle i,j \rangle) \) associated with each \( \langle i,j \rangle \) in \( E \). Let \( \text{pred} \in [1,n] \to 2^{[1,n]} \) be \( \lambda i. \{ j \in [1,n] : \langle i,j \rangle \in E \} \) and \( \text{succ} \in [1,n] \to 2^{[1,n]} \) be \( \lambda i. \{ j \in [1,n] : \langle i,j \rangle \in E \} \); if \( \text{pred}(n_0) = \emptyset \) and for any \( v \in [1,n] - \{ n_0 \}, \text{pred}(v) \neq \emptyset \) and \( \text{succ}(v) = \emptyset \).

Example 2.0.1

The program:

\[ \text{where path}(i) \text{ is the set of paths from the entry point } n_0 \text{ to the vertex } i \text{ and } T \in (E^* \to (A \to A)) \text{ is recursively defined as follows: if } p \text{ is an empty path then } T(p) \text{ is the identity map on } A \text{ else } p = (q,a) \text{ where } a \in \mathcal{A}, \mathcal{A} \subseteq E \text{ and } T(p) = \lambda q. t[C(qa)](\tilde{t}(q)(\phi)) \].

The system of equations \( \mathcal{P} = \mathcal{F}_L(t, \phi)(P) \) associated with the program \( \pi \) using \( (A, t) \) and \( \phi \) is defined as follows:
\[
\begin{align*}
\mathcal{P}_n & = \phi \\
\mathcal{P}_j & = \bigcup_{i \in \text{pred}(j)} \text{t}([C(\langle i,j \rangle)](\mathcal{P}_i)) \text{ if } j \in [1,n] - \{ n_0 \}
\end{align*}
\]

Notation: \( M = (\sigma_1, \tau_1, \Pi) \) is a complete lattice then the set \( (L \to M) \) of total maps from the set \( L \) into \( M \) is a complete lattice \((L \to M)(S', L', \tau', \Pi', \Pi') \) for the pointwise ordering \( f \leq g \text{ iff } \forall x \in L, f(x) \leq g(x) \). In the following the distinction between \( \sigma, \tau, L, \Pi \) and \( \sigma', \tau', L', \Pi' \) will be determined by the context. Also a map \( f \in (L \to M) \) will be extended to \( (2^L \to 2^n) \) as \( \lambda \sigma \in 2^L, f(\sigma) = \sigma S \) and to \( (L \to M^n) \) as \( \lambda \sigma \in 1^n, f(\sigma) = (f(x_1), \ldots, f(x_n)) \).

3. DEDUCTIVE SEMANTICS OF PROGRAMS

3.1 Forward Semantics

The forward semantic analysis of a program \( \pi \) consists in determining at each program point an invariant assertion which characterizes the set of states which are the descendants of the input states satisfying a given entry assertion \( \Phi \).

More precisely an assertion is a total map from \( \mathcal{V} \) into \( \mathcal{R} \). The set \( A = (\mathcal{V} \to \mathcal{R}) = \{ \alpha \in \mathcal{V}, \text{true} \} \) of assertions is a complete boolean lattice partially ordered by the implication \( \supseteq \).
Example 3.1.0.1

The system of forward semantic equations associated with the program 2.0.1 is:

\[
\begin{align*}
\Phi_1 &= \phi \\
\Phi_2 &= \text{sp}(\lambda x.(x\leq 100))\Phi_3 \\
\Phi_3 &= \text{sp}(\lambda x.(x+1))\Phi_2 \\
\Phi_4 &= \text{sp}(\lambda x.(x\geq 100))\Phi_3 \\
\end{align*}
\]

taking \(\phi=\lambda x.(x=1)\) its least fixpoint characterizes the descendants of the input states satisfying \(\phi\):

\[
\begin{align*}
\Phi_1 &= \lambda x.(x=1) \\
\Phi_2 &= \lambda x.(1\leq x\leq 100) \\
\Phi_3 &= \lambda x.(25\leq x\leq 101) \\
\Phi_4 &= \lambda x.(x=101) \\
\end{align*}
\]

End of Example.

3.2 Backward Semantics

The backward semantic analysis of a program consists in determining at each program point an invariant assertion which characterizes the set of states which are the ascendants of the output states satisfying a given exit specification \(\phi\).

Since we can consider the inverse of the state transition relation defined by the operational semantics no new formalism is necessary in order to treat backward program analysis. Instead of Floyd's forward predicate transformer we just have to consider Hoare[58]-Dijkstra[76]'s backward predicate transformer:

\[
\begin{align*}
\text{up}(q) &= \{x \in A_0 \mid [x \in V, (P(x) \land x \in dom(q) \land q(x))] \} \\
\text{up}(a) &= \{x \in A_0 \mid [x \in V, (x \in dom(a) \land P(x))] \}
\end{align*}
\]

(notice that \(\text{set}(\text{up}(S))\) is a complete join and meet morphism and the inverted program graph \(G'=(n, E', n_0, n_0')\text{C'}\) where \(E'=\{(i,j) : j \in E\}, C'=\lambda x<1,j \in E,\ \text{C'}\).

Example 3.2.0.1

The inverted program graph corresponding to 2.0.1 is:

\[
\begin{align*}
\lambda x.(x\leq 100) \\
\lambda x.(x\leq 100) \\
\lambda x.(x+1) \\
\lambda x.(x\geq 100)
\end{align*}
\]

The corresponding system of backward semantic equations is:

\[
\begin{align*}
\Phi_1 &= \text{up}(\lambda x.(x\leq 100))\Phi_2 \lor \text{up}(\lambda x.(x=10))\Phi_4 \\
\Phi_2 &= \text{up}(\lambda x.(x+1))\Phi_3 \\
\Phi_3 &= \text{up}(\lambda x.(x\geq 100))\Phi_2 \lor \text{up}(\lambda x.(x=100))\Phi_4 \\
\Phi_4 &= \phi \\
\end{align*}
\]

The merge over all paths and least fixpoint characterizations of the ascendants of the output states satisfying the exit specification \(\phi=\lambda x.(x=101)\) are both equal to:

\[
\begin{align*}
\Phi_1 &= \lambda x.(x\leq 101) \\
\Phi_2 &= \lambda x.(x\leq 100) \\
\Phi_3 &= \lambda x.(x\leq 101) \\
\Phi_4 &= \phi = \lambda x.(x=101) \\
\end{align*}
\]

End of Example.

In the following no distinction will be made between forward and backward program analyses because of the above mentioned symmetry.

4. APPROXIMATE ANALYSIS OF PROGRAMS

The semantic analysis of programs cannot be automated since neither the merge over all paths nor the least fixpoint characterization of the invariant assertions to be generated leads to a computable function. Therefore optimizing compilers and program verification systems are only concerned with the discovery of approximate invariants assertions. Here an approximate invariant assertion \(Q\) will be one which is implied by the exact invariant assertion \(P\) defined by the deductive semantics.

DEFINITION 4.0.1

If \(P,Q\leq\phi\) then "approximate \(P\) iff \(P\Rightarrow\phi\)"

This definition of "approximates" is the one which is useful in logical analysis of programs, data type determination and data flow analysis. (The dual one might be useful e.g., for proving termination).

The now classical lattice theoretic approach to approximate analysis of programs can be briefly sketched as follows: the representation of an approximate assertion is an element of a complete lattice \(\text{Af}(\text{C},\leq,\land,\lor,\bot)\). The meaning of the elements of \(\text{A}\) is specified by a (too often implicit) order morphism \(\gamma\) mapping \(\text{A}\) to a subset of assertions \(\gamma(\gamma(A))\subseteq\text{A}\). The intention is that \(\gamma\) is an implementable image of those aspects \(\gamma(A)\) of the program properties which are to be understood at each program point whereas the assertions belonging to \(\gamma(A)\) are ignored (that is approximated from above in \(\gamma(A)\)). To each elementary command \(\text{set}\) is associated an isotope map \(t(S)\) from \(\text{A}\) to \(\text{A}\). The intent is that \(t(S)\) is an approximate predicate transformer such that \(t(S)\) represents the propagation of the information \(\text{ieA}\) through the statement \(S\).

The ideal merge over all paths program-wide analysis [Graham & Wegman[76], Kam & Ullman[77], Rosen[76], Tarjan[76]] is often approximated by a fixpoint solution [Cousot & Cousot[77a], Jones & Muchnick[78], Kaplan & Ullman[78], Kildas[79], Tenenbaum[79]]. A fixpoint system of isotope equations \(X\in P(X)\) where \(F(x)\in A(x)\) is associated with the program graph. The approximate invariant assertions are generated by computing iteratively the least fixpoint of \(F\) starting from the inimum of \(\text{A}\) and using any chaotic or asynchronous iteration strategy [Cousot[77a]] or the least fixpoint is approximated above using an extrapolation technique in order to accelerate the convergence of the iterates whenever \(A\)
does not satisfy the ascending chain condition (Cousot & Cousot 77a).

The design of $\Lambda$, $t$, the implicit $\gamma$ and the determination of the continuation value for $\epsilon$.

two semantic analyses (i.e., $sp(x:=x+y)(\epsilon_1) = \lambda(x,y),(x>y \wedge y \geq 0)$ and $sp(x:=x+y)(\epsilon_2) = \lambda(x,y),(x>y \wedge y \geq 0)$) and next comparing them. Since these analyses are not related by the ordering $\preceq$, the comparison and $x$.
If the initial choice of $\bar{A}$ does not satisfy assumption 5.1.0.2 we can use the following:

**Theorem 5.2.0.4**

If $\bar{A} \leq A$, the upper closure operator $p$ on $A$ such that $p(A)$ is the least Moore family containing $\bar{A}$ is:

$$p = \lambda \alpha, \rho \{ \alpha \in (\bar{A} \cup \{\lambda x, true\}) \land \rho(\alpha) \}$$

$$p(A) = \{ s : s \leq (\bar{A} \cup \{\lambda x, true\}) \land s \neq \}$$

**Example 5.2.0.5**

Returning to example 5.1.0.1 where $A = (\mathbb{Z} \times \mathbb{B})$ and $\bar{A} = (\lambda u, false, \lambda u, u \neq 0, \lambda u, u \neq 0, \lambda u, true)$ the least Moore family containing $\bar{A}$ is the one containing $\lambda u, true$, $\bar{A}$ and the meets of the non-empty subsets of $\bar{A}$ that is the complete lattice:

$$\lambda u, true$$

$$\lambda u, u \neq 0$$

$$\lambda u, u \neq 0$$

$$\lambda u, false$$

The corresponding approximation operator is:

$$\rho = \lambda \alpha, p \left\{ \alpha \in false, \lambda u, true \land \alpha(p) \right\}$$

$$\text{else if } P = \lambda u, true \text{ then } \lambda u, true$$

$$\text{else if } P = \lambda u, u \neq 0 \text{ then } \lambda u, u \neq 0$$

$$\text{else if } P = \lambda u, u \neq 0 \text{ then } \lambda u, u \neq 0$$

$$\text{else } \lambda u, true$$

**End of Example.**

5.3 Representation of the Lattice of Approximate Assertions

In order to represent the approximate assertions in a computer memory we must use a complete lattice $\Lambda(R, I, T, U, \Pi)$ such that the smaller algebras $\bar{A} = p(A)(\Rightarrow, \lambda x, false, \lambda x, true, \lambda x, true, p(vs) \alpha) \land A \in (\bar{A}, I, T, \Pi)$ be isomorphic. Let $\gamma \in (\Lambda(R, I, T, U, \Pi))$ be the corresponding lattice isomorphism. Let $\alpha \in (\Lambda(R, I, T, U, \Pi))$ be $\gamma \Rightarrow p$. $\alpha(p)$ is the representation of the least upper approximation of the assertion $p \Lambda A$ whereas $\gamma(p)$ provides the meaning of $p \Lambda A$. The connection $\langle \alpha, \gamma \rangle$ between $\Lambda(R, I, T, U, \Pi)$ and $\Lambda(R, I, T, U, \Pi)$ has the following property:

**Definition 5.3.0.1**

Let $L_1(E_1)$ and $L_2(E_2)$ be posets. $\langle \alpha, \gamma \rangle$ is a pair of adjoined functions if and only if:

1. $\gamma$ is an upper closure operator on $L_1$, $\alpha \gamma$ is a lower closure operator on $L_2$,
2. $\alpha$ is isomoprhic to $\gamma$.

Moreover if $L_1(E_1, I_1, T_1, U_1, \Pi_1)$ and $L_2(E_2, I_2, T_2, U_2, \Pi_2)$ are complete lattices then:

**Theorem 5.3.0.2**

If $p$ is an upper closure operator on $A$, the image $\gamma(A)$ of $A \in (\bar{A}, I, T, U, \Pi)$ on the lattice isomorphism $\gamma$ is equal to $\rho(A)(\Rightarrow, \lambda x, false, \lambda x, true, p(vs) \alpha)$.

$\langle \alpha, \gamma \rangle$ is a pair of adjoined functions if and only if:

1. $\alpha$ is onto $L_1 \times L_2$, $\gamma$ is one-to-one (injective) and if only if $\alpha \gamma = \lambda y \lambda x \gamma(y) \alpha(x)$
2. $\gamma$ is onto $L_1 \times L_2$.
\[ \gamma = \lambda y \psi L_4 L_5 (x \psi L_4 : a(x) \psi y) \]
\[ \alpha \text{ is an isomorphism from the complete lattice } \gamma \psi L_4 \text{ onto the complete lattice } L_2 \text{ the inverse of which is } \gamma \]

We use the notation \( L_1 \prec \alpha, \gamma \succ L_2 \) to state that \( L_1 \) and \( L_2 \) are connected by the pair \( \langle \alpha, \gamma \rangle \) of adjointed functions which are respectively surjective and injective, if \( \alpha \) is a complete join-morphism from \( L_1 \) onto \( L_2 \) (respectively \( \gamma \) is a one-to-one complete meet-morphism from \( L_2 \) into \( L_1 \)) we write \( L_1 \prec \alpha, \gamma \succ L_2 \) and assume that the adjoint \( \gamma \alpha \) is determined by \( 5.3.0.5.3.2 \) \( 5.3.0.5.3.1 \).

In the literature the most usual method for defining a program analysis framework is to specify the complete lattice \( A(E, I, \mathcal{L}, \mathcal{P}) \) representing approximate assertions and to informally describe the meaning of its elements (e.g., constant propagation, Kildall[73], Kam \\& Ullman[77]). Hence the function \( \gamma \in (A \rightarrow A) \) remains implicit.

It is often the case that \( A \) is only assumed to be a (complete) join-semi-lattice \( A(E, I, \mathcal{L}) \) (or dual meet-semi-lattice for some authors) but since an infimum is adjointed to \( A \) it is in fact a complete lattice (even when the meet-operation is not used or what is called meet is not \( \cap \) e.g., Wegbreit[75]).

When \( \gamma \in (A \rightarrow A) \) is isotone but not a complete meet-morphism the set \( \gamma(A) \) does not fulfill assumption 5.1.0.1 with the consequences examined at paragraph 5.1. The design of \( \gamma(A) \) and \( A \) can be revised as stated by theorem 5.2.0.4.

When \( \gamma \in (A \rightarrow A) \) is a complete meet-morphism but not one-to-one, several distinct elements of \( A \) have the same meaning. Since this is useless, the design of \( A \) and \( \gamma \) can be revised as follows:

**Theorem 5.3.0.7**

Let \( A(E, I, \mathcal{L}, \mathcal{P}) \) be a complete lattice and \( \gamma \in (A \rightarrow A) \) be a complete meet-morphism. Let \( \sigma \in (A \rightarrow A) \) be \( \lambda x \psi (y \psi A : \gamma(y)) \), \( \tilde{A} = \sigma(A) \), \( \tilde{\gamma} = \{ \gamma(y) \} \).

1. \( \forall x \psi A, \gamma(y) = \gamma(\gamma(y)) \)
2. \( \sigma \) is a lower closure operator on \( A \)
3. \( \gamma \) is a one-to-one complete meet-morphism from the complete lattice \( A(E, I, \mathcal{L}, \mathcal{P}) \) into \( A \)

Since \( \gamma(A) = \tilde{\gamma}(\tilde{A}) \), \( A \) and \( \tilde{A} \) have the same expressive power. Among all subsets of \( A \), \( \tilde{A} \) is one with minimal cardinality.

**Theorem 5.3.0.8**

1. \( \forall L \psi \subseteq L, \gamma(L) = \gamma(y) \Rightarrow \{ \gamma(L) \} = \tilde{A} \)
2. \( \forall L \psi \subseteq L, \gamma(y) = \gamma(y) \Rightarrow \{ \gamma(y) \} \leq \gamma(L) \)
3. \( \forall x \psi A, \gamma(y) = \gamma(y) \Rightarrow \{ y \psi x \} \)

**6.1 Least Closure Operator Greater than or Equal to an Arbitrary Function**

**Theorem 6.1.0.1**

Let \( L(E, I, \mathcal{L}, \mathcal{P}) \) be a complete lattice and \( f \in (L \rightarrow L) \). Let \( \gamma \in \{ L \rightarrow L \} \). Let \( \tilde{A} \psi (L \psi L) \Rightarrow (L \psi L) \) be \( \lambda x \psi (L \psi L) \Rightarrow (L \psi L) \). Let \( \gamma \) is the least isotone operator on \( L \) greater than or equal to \( f \).

- \( \forall x \psi L, \gamma(y) \Rightarrow (L \psi L) \Rightarrow (L \psi L) \) be \( \lambda x \psi (L \psi L) \Rightarrow (L \psi L) \).

- \( \forall x \psi L, \gamma(y) \Rightarrow (L \psi L) \Rightarrow (L \psi L) \) be \( \lambda x \psi (L \psi L) \Rightarrow (L \psi L) \).

- \( \forall x \psi L, \gamma(y) \Rightarrow (L \psi L) \Rightarrow (L \psi L) \) be \( \lambda x \psi (L \psi L) \Rightarrow (L \psi L) \).

- \( \gamma \) is the least closure operator greater than or equal to \( f \) and \( \gamma \) is the greatest Moore family contained in \( f \).

**6.2. Definition of a Space of Approximate Assertions by Composition of Upper Closure Operators**

The composition of two upper closure operators on \( A \) is usually a closure operator [Ore43]). However the space of approximate assertions can be designed by successive approximations using the following composition of upper closure operators:

**Theorem 6.2.0.1**

Let \( L(E, I, \mathcal{L}, \mathcal{P}) \) be a complete lattice, \( L \) an upper closure operator on \( L \) and \( \gamma \) be an upper closure operator on \( \gamma(L) \). Then \( \gamma \) is an upper closure operator on \( L \) and \( \gamma \) is a closure operator on \( L \).

**Example 6.2.0.2**

Many program analysis frameworks are designed in order to describe some properties of each program variable but so that the relationships among the values of these variables are ignored. An example is Jones & Muchnick[78]'s type determination scheme. A counter-example is the determination of linear relationships among numerical variables, Cousot & Halbwachs[78]. The corresponding approximation can be characterized as follows:

Assume that \( V = D \), let \( A \) be \( (V + B) \) and \( A_1 \) be \( (D + B) \). Let us define:

\[ \gamma(y) \in \{ y \psi x \}, \sigma \in (A \rightarrow A) \]
σ is an upper closure operator on \(A_m\) and an assertion \(\rho A_m\) does not state relationships among the program variables if and only if \(\sigma(\rho)\). The approximation assertions on each individual variable \(x_i\) are next defined using an upper closure operator \(\rho_j\) on \(A_j\). The induced closure operation \(\rho\) on \(\sigma(A_m)\) is defined by \(\rho(\rho)(x_{j1},\ldots,x_{jm}) = \bigcup_{j=1}^m \rho_j(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\ldots(\rho_j(\rho_j(\ldots(\rho_j(x_{j1}))\ldots)))\ldots)))\ldots)))\ldots)))\ldots)))\ldots)))\ldots)))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots)))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots))))\ldots)})
THEOREM 6.4.0.2

1. Let I be a principal ideal and \( J \) be a dual semi-ideal of a complete lattice \( L([x], [y], [z], [w]) \). If \( \text{InJ} \) is nonvoid then \( \text{InJ} \) is a complete and convex sub-join-semilattice of \( L \).

2. Every complete and convex sub-join-semilattice \( C \) of \( L \) can be expressed in this form with
   \( I = \{ x \in L : x \in [I, J] \} \) and \( \{ y \in C : y \subseteq x \} \subseteq J \).

THEOREM 6.4.0.3

Let \( \{ I_i \} \) be a family of principal ideals of the complete lattice \( L([x], [y], [z], [w]) \) containing \( L \). Then \( \lambda x \in [I_i] : i \in \Delta \wedge x \in [I_i] \) is an upper closure operator on \( L \).

Example 6.4.0.4

The following lattice can be used for static analysis of the signs of values of numerical variables:

```
\[ \begin{array}{c}
    \top & [0, \infty) & [0, \infty) & [0, \infty) & \bot \\
    [0, \infty) & \cap & \cup & \cap & [0, \infty) \\
    [0, \infty) & \cap & \cup & [0, \infty) & \cup \\
    [0, \infty) & \cap & [0, \infty) & \cup & \cup \\
    \bot & [0, \infty) & [0, \infty) & [0, \infty) & \top \\
\end{array} \]
```

(\( \wedge \) stands for \( \cap \), \( \vee \) stands for \( \cup \), \( \emptyset \) stands for \( \varnothing \), \( \tau \) stands for \( \tau \).

A further approximation can be defined by the following family of principal ideals:

\[ I_1, I_2, I_3, I_4, I_5 \]

which induces an upper closure operator \( I \):

```
\[ \begin{array}{c}
    \top & I_1 & I_2 & I_3 & I_4 & I_5 & \bot \\
    I_1 & \cap & \cup & \cap & \cup & \cap & \top \\
    I_2 & \cap & \cup & \cap & \cup & \cap & \top \\
    I_3 & \cap & \cup & \cap & \cup & \cap & \top \\
    I_4 & \cap & \cup & \cap & \cup & \cap & \top \\
    I_5 & \cap & \cup & \cap & \cup & \cap & \top \\
    \bot & [0, \infty) & [0, \infty) & [0, \infty) & [0, \infty) & [0, \infty) & \top \\
\end{array} \]
```

7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER

INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

In addition to \( A \) and \( \gamma \), the specification of a program analysis framework also includes the choice of an approximate predicate transformer \( \tau_c(L \to [A \to A]) \) (or a monoid of maps on \( A \) plus a rule for associating maps to program statements (e.g., Rosenberg)). We now show that in fact our method is not indispensable since there exists a best correct choice of \( \gamma \) which is induced by \( \tau_c \) and the formal semantics of the considered programming language.

7.1 A Reasonable Definition of Correct Approximate Predicate Transformers

At paragraph 3, given \( (V, A, \gamma) \) the minimal assertion which is invariant at point \( 1 \) of a program \( \pi \) with entry specification \( \phi^A \) was defined as:

\[ P_1 = \forall \pi \in \text{Path}(1) \]

Therefore the minimal approximate invariant assertion is the least upper approximation of \( P_1 \) in \( \mathcal{L} \) that is:

\[ \rho(P_1) = \rho(\forall \pi \in \text{Path}(1)) \]

Even when \( \phi \) is finite set of finite paths the evaluation of \( \tau_c(\phi) \) is hardly machine-implementable since for each path \( \pi = \pi_1, \ldots, \pi_m \) the computation sequence \( X_0, X_1 = \tau_c(\pi_1)(X_0), \ldots, X_m = \tau_c(\pi_m)(X_{m-1}) \) does not necessarily only involve the elements of \( A \) and \( (A \to A) \). Therefore using the abstract methods from \( \tau_c(L \to (A \to A)) \) a machine-representable sequence \( X_0, X_1 = \tau_c(\pi_1)(X_0), \ldots, X_m = \tau_c(\pi_m)(X_{m-1}) \) is used instead of \( X_0, \ldots, X_m \) which leads to the expression:

\[ Q_1 = \rho(\forall \pi \in \text{Path}(1)) \]

The choice of \( \tau_c \) and \( \phi \) is correct if and only if \( Q_1 \) is an upper approximation of \( P_1 \) in \( \mathcal{L} \) if and only if:

\[ \forall \pi \in \text{Path}(1) \]

In particular for the entry point we must have \( \phi \Rightarrow \phi \) so that we can state the following:

DEFINITION 7.1.0.1

1. An approximate predicate transformer \( \tau_c(L \to (A \to A)) \) is said to be a correct upper approximation of \( \tau_c(L \to (A \to A)) \) in \( A \to \gamma(A) \) if and only if for all \( \phi \in A \to \gamma(A) \) such that \( \phi \Rightarrow \phi \) and program \( \pi \) we have:

\[ \text{MOP}_\pi(\tau_c(\phi)) \Rightarrow \text{MOP}_\pi(\phi) \]

2. Similarly if \( \text{A} \gg \gamma \), \( \text{MOP}_\pi(\phi_1) \Rightarrow \text{MOP}_\pi(\phi_2) \)

This global correctness condition for \( \tau_c \) is very difficult to check since for any program \( \pi \) and any program point \( 1 \) all paths \( \pi \in \text{Path}(1) \) must be considered. However it is possible to use instead the following equivalent local condition which can be checked for every type of statements:

End of Example.
THEOREM 7.1.0.2

(1) \(\mathbf{t} \in (L \to (A \to A))\) is a correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) if and only if \(\forall \mathbf{w}, \mathbf{v}. (\mathbf{w} \subseteq \mathbf{v}, \mathbf{w} \in \mathbf{A}, \mathbf{v}(\mathbf{t}(\mathbf{S})) \subseteq \mathbf{t}(\mathbf{S})) \Rightarrow \mathbf{t}(\mathbf{S}).\)

(2) \(\mathbf{t} \in (L \to (A \to A))\) is a correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) if and only if \(\forall \mathbf{w}, \mathbf{v}, \mathbf{a}. (\mathbf{w} \subseteq \mathbf{v}, \mathbf{w} \in \mathbf{A}, \mathbf{a}(\mathbf{t}(\mathbf{S})) \subseteq \mathbf{t}(\mathbf{S})) \Rightarrow \mathbf{t}(\mathbf{S}).\)

If \(\mathbf{t} \in (L \to (A \to A))\) is a correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\), we have \(\mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S}) \Rightarrow \mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S})\) whenever \(\mathbf{p}(\mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S})) \Rightarrow \mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S})\). The cases when equality holds are not easy to distinguish. Yet the following sufficient condition turns out to be useful afterwards:

THEOREM 7.1.0.3

(1) If \(\mathbf{t} \in (L \to (A \to A))\) is a correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) then \(\mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S}) \Rightarrow \mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S})\).

(2) If \(\mathbf{t} \in (L \to (A \to A))\) is a correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) then \(\mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S}) \Rightarrow \mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S})\).

Similar results hold for fixed-point analysis of programs.

THEOREM 7.1.0.4

Let \(\mathbf{t} \in (L \to (A \to A))\) be an isotone correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) where 
\[A \Rightarrow A \Rightarrow A \Rightarrow A\] then \(\mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S}) \Rightarrow \mathbf{MDF}_{\mathbf{t}}(\mathbf{t}, \mathbf{S})\).

DEFINITION 7.2.0.1

If \(\mathbf{t}_1, \mathbf{t}_2\) are correct upper approximations of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) then we say that \(\mathbf{t}_1\) is better than \(\mathbf{t}_2\) if \(\forall \mathbf{v}. (\mathbf{v}(\mathbf{t}_1) \Rightarrow \mathbf{v}(\mathbf{t}_2))\) and \(\mathbf{t}_1\) is a better approximation than \(\mathbf{t}_2\).

LEMMA 7.2.0.2

Let \(\mathbf{t}_1, \mathbf{t}_2\) be correct upper approximations of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\). If \(\forall \mathbf{w}, \mathbf{v}, \mathbf{a}. (\mathbf{w} \subseteq \mathbf{v}, \mathbf{w} \in \mathbf{A}, \mathbf{a}(\mathbf{t}(\mathbf{S})) \subseteq \mathbf{t}(\mathbf{S})) \Rightarrow \mathbf{t}(\mathbf{S}) \Rightarrow \mathbf{t}(\mathbf{S})\) then \(\mathbf{t}_1\) is better than \(\mathbf{t}_2\).

THEOREM 7.2.0.3

Let \(\mathbf{t}\) be an isotone correct upper approximation of \(\mathbf{t} \in (L \to (A \to A))\) in \(A \Rightarrow A\) if and only if \(\forall \mathbf{w}, \mathbf{v}, \mathbf{a}. (\mathbf{w} \subseteq \mathbf{v}, \mathbf{w} \in \mathbf{A}, \mathbf{a}(\mathbf{t}(\mathbf{S})) \subseteq \mathbf{t}(\mathbf{S})) \Rightarrow \mathbf{t}(\mathbf{S}) \Rightarrow \mathbf{t}(\mathbf{S})\).

COROLLARY 7.2.0.4

If \(A \Rightarrow A \Rightarrow A \Rightarrow A\) then \(\forall \mathbf{w}, \mathbf{v}, \mathbf{a}. (\mathbf{w} \subseteq \mathbf{v}, \mathbf{w} \in \mathbf{A}, \mathbf{a}(\mathbf{t}(\mathbf{S})) \subseteq \mathbf{t}(\mathbf{S})) \Rightarrow \mathbf{t}(\mathbf{S}) \Rightarrow \mathbf{t}(\mathbf{S})\).

The most interesting consequence is that we have in hand a formal specification of the programs which have to be written in order to implement any specific program analysis framework once \(A\) and \(\mathbf{Y}\) have been chosen. As a challenge to automatic program synthesizers let us consider a simple example.

Example 7.2.0.5

Coming back to examples 6.2.0.2 and 6.3.0.5 we assume that \(\mathbf{D}\) is the set of integers included between two bounds \(\mathbf{a}\) and \(\mathbf{b}\). For simplicity we shall assume that \(\mathbf{D} = [\mathbf{a}, \mathbf{b}]\) with ordering \(\mathbf{a} \leq \mathbf{b}\) and \(\mathbf{a}\) is the infimum. Let \(\mathbf{t}\) be the partial function that \(\mathbf{t}(\mathbf{a}) = \mathbf{a}\).
According to theorem 7.2.0.4 the best upper approximation of \( \alpha(x;y) \) in \( A \) is \( \alpha(x;y) \wedge \alpha(y;x) \), where \( \alpha(x;y) \) and \( \alpha(y;x) \) are the lower and upper approximations of \( \alpha(\{x\};\{y\}) \) and \( \alpha(\{y\};\{x\}) \), respectively.

Example 7.2.0.6

Some program analyses (such as "reaching definitions", "available expressions", "live variables", ... Aho & Ullman[77]) are "history sensitive" because the approximate assertions which are useful at some program point can be derived from sequences of states from some path through the program. In such a case Hoare[79] formal definition of states using a set of sequential traces is more convenient than the state machine semantics of paragraph 3.

7.2.0.6.1. Associating a Set of Traces with a Program

Given a universe \( V \) of values, a set \( L \) of elementary assignments, a set \( E \) of elementary tests, the set of sequential traces is a directed graph \( \mathcal{T}(V,E) \) generated by \( L \cup E \cup E \cup V \).

The concatenation operation "::" is extended to elements of the complete lattice \( 2^C(V,E;\mathcal{T};\mathcal{L};\mathcal{V};\mathcal{U}) \) by \( S \cdot T = \{s, t | s \in S, t \in T \} \).

Let us define a forward "set of traces transformer" \( ft: L \rightarrow (2^C(V,E;\mathcal{T};\mathcal{L};\mathcal{V};\mathcal{U})) \) as \( \lambda s.\mathcal{L} \cdot \mathcal{T}(s;\mathcal{T}(s)) \). The set of traces associated with a program \( P \) and an entry specification \( \Phi \in L \) is \( \mathcal{M}_{\Phi}(ft(P)) \).

7.2.0.6.2. Approximating a Set of Traces with an Assertion Characterizing the Descentants of the Entry States

The connection with the deductive semantics of paragraph 3 is made using \( \alpha(T;A) \) such that for any set of traces, \( \alpha(T;A) \) characterizes the possible descentants of the entry states (belonging to \( T \)) when the traces \( t \in T \) are executed. From an obvious (hence not given here) operational semantics of sequential traces we derive that \( \alpha(T;A) = \lambda t.\mathcal{V}(\alpha(t;A)) \) where \( \alpha(t;A) = \lambda v.\mathcal{V}(\alpha(v;A)) \).

Since \( \alpha \) is a complete join-morphism from \( 2^C(V,E;\mathcal{T};\mathcal{L};\mathcal{V};\mathcal{U}) \) to \( \mathcal{M}_{\Phi}(ft(P)) \), the correct upper approximation of \( ft \) in \( A \) is \( \mathcal{F}(ft(P)) \).

Example 7.2.0.7

In order to briefly illustrate the hierarchy of program analysis frameworks, let us consider three comparable examples the approximation function of which can be sketched using a geometrical analogy. Let \( P \) be a predicate over two numerical variables \( x \) and \( y \) and the characteristic set of which is the following:

\[ \{x, y | x < y \} \]
The upper closure operator of example 5.2.0.5 defines a very rough approximation consisting in approximating this set by the quarter of plane containing all its points:

\[ y \leq p(x, y) \]

A more precise approximation (example 6.3.0.5) consists in approximating the characteristic set of \( P \) by the smallest rectangle including it and whose sides run parallel with the axes:

\[ y \leq p(x, y) \]

A refinement consists in approximating the characteristic set of \( P \) by its convex-hull:

\[ y \leq p(x, y) \]

The corresponding framework was used for the automatic discovery of linear restraints among variables of programs [Cucos & Hallwachs [76]].

End of Example.

9. MERGE OVER ALL PATHS VERSUS LEAST FIXPOINT GLOBAL ANALYSIS OF PROGRAMS

9.1 "Distributive" Program Analysis Frameworks

We recalled at paragraph 4 that once a program analysis framework \((A, t, \gamma)\) has been designed, the program-analysis problem has various solutions including the merge over all paths and least fixpoint solutions. It is known (Kam [111, 113]) that when \( A \) satisfies the ascending chain condition, \( \mathcal{W}S\mathcal{L} \) is isotope we have \( \text{MOP}_A(t, \phi) \subseteq \mathcal{L}\mathcal{F}(F_M(t, \phi)) \). Also the additional hypothesis that \( \mathcal{W}S\mathcal{L} \), \( t(S) \) is a join-morphism (sometimes called join-distributive map) implies \( \text{MOP}_A(t, \phi) \subseteq \mathcal{L}\mathcal{F}(F_M(t, \phi)) \). Slightly more general is the following:

**THEOREM 9.1.0.1**

If \((A, \phi, t, \mathcal{L}, \mathcal{U})\) is a complete lattice and \( t(S) \) is isotope then for all programs \( t' \) and \( \phi' \neq A \), \( \text{MOP}_A(t', \phi') \subseteq \mathcal{L}\mathcal{F}(F_M(t', \phi')) \). Moreover \( \mathcal{W}S\mathcal{L} \), \( t(S) \) is a complete \( \mathcal{L}\mathcal{F}\) then \( \text{MOP}_A(t, \phi) \subseteq \mathcal{L}\mathcal{F}(F_M(t, \phi)) \).

(This theorem is implicitly used at paragraph 3 taking \( A = (V \cup B) = (x, y, \text{false}, x, \text{true}, y, A) \) for \( A, \mathcal{L}, \mathcal{U}, \mathcal{N} \) and either \( s \) or \( u \) for \( t \).

If \( \Delta A \cup \gamma \neq A \) and \( t(S) \) then the above theorem establishes the correctness of \( \mathcal{L}\mathcal{F}(F_M(t, \phi)) \) with respect to \( \text{MOP}_A(t, \phi) \). In the literature the correctness of \( \text{MOP}_A(t, \phi) \) is always left open for general solutions to program-wide analysis problems since whenever some \( t(S) \) is not a complete join-morphism \( \text{MOP}_A(t, \phi) \) can be strictly better than \( \mathcal{L}\mathcal{F}(F_M(t, \phi)) \).

When \( A \) satisfies the ascending chain condition \( \mathcal{L}\mathcal{F}(F_M(t, \phi)) \) is computable, which is not necessarily the case of \( \text{MOP}_A(t, \phi) \). In that case a variety of methods can be used (e.g. Rosen [783]) which can find sharper information that fixpoint methods and therefore approach the ideal merge over all paths solution which provides the maximum information relevant to \( A, t \), and \( \gamma \).

In our opinion the above argument is not entirely convincing since for different correct approximate predicate transformers \( t_1, t_2 : (L \to (A \to A)) \) it may be the case that \( \mathcal{L}\mathcal{F}(F_M(t_1, \phi)) \subseteq \text{MOP}_A(t_2, \phi) \). In order to relieve from the burden of badly chosen approximate predicate transformers the argument must consider the best approximate predicate transformer relevant to \( A \) (Theorem 7.2.0.4). Then the following result is a useful complement to theorem 9.1.0.1:

**THEOREM 9.1.0.2**

Let \( t_c((L \to (A \to A))) \) be the best correct upper approximation of \( t_c((L \to (A \to A))) \) in \( A \). Then \( (A, t_c) \) is a complete sublattice of \( A \) then \( \text{MOP}_A(t, \phi) = \mathcal{L}\mathcal{F}(F_M(t, \phi)) \).

**Example 9.1.0.3**

If \( A = (Z \cup B) \) and \( A \) are \( y(A) \) where:

\[ A = \]

and \( \gamma(S) = \alpha \cup \beta \), \( \gamma(S) = \gamma \cup \beta \), \( \gamma(S) = \gamma \cup \beta \), etc. then \( A \) is not a sublattice of \( A \) since \( \gamma \cup \gamma \gamma \cup \gamma \gamma \cup \gamma \gamma \), etc.

The merge over all paths analysis of the program:

\[ \gamma(S) = \alpha \cup \beta \]

which (powerful enough in order to determine that the while-loop does not terminate) is strictly better than the least fixpoint analysis (which fails to discover that \( i \) is invariant on the exit path of the loop).

End of Example.

9.2 "Non-Distributive" Program Analysis Frameworks

The merge over all paths analysis of a program using some "non-distributive" program analysis framework can always be defined by means of the least fixpoint of a system of isotone equations associated with that program:

**THEOREM 9.2.0.1**

Let \( (A, \phi, t, \mathcal{L}, \mathcal{U}) \) be a complete lattice, \( t_c((L \to (A \to A))) \) be an approximate predicate transformer, \( 2^A \in \phi, \chi \cup \alpha, \phi, \chi \) be the complete lattice of all subsets of \( A \), \( t((L \to (2^A \to 2^A))) = t((L \to (2^A \to 2^A))) \) be \( \lambda \mathcal{L} \mathcal{F}(F_M(t, \phi)) \) and \( \mu(2^A \to 2^A) \) be \( \lambda \mathcal{L} \mathcal{F}(F_M(t, \phi)) \).

The above construction is not fully satisfactory since \( 2^A \) is not isomorphic to \( (A, \phi) \) when the definition of [111, 113] is used.
10. COMBINATION OF PROGRAM ANALYSIS FRAMEWORKS

The ideal method in order to construct a program analyzer (to be integrated in optimizing compilers or program verification systems) would consist in a separate design and implementation of various complementary program analysis frameworks which could then be systematically combined using a once for all implemented assembler. In this section, we show that such an automatic combination of independently designed parts would not lead to an optimal analyzer and that unfortunately the efficient combination of program analysis frameworks often necessitates the revision of the original design phase.

10.1 Reduced Cardinal Product of Program Analysis Frameworks

**THEOREM 10.1.0.1**

Let \( \{A_1,t_1,Y_1\} \) and \( \{A_2,t_2,Y_2\} \) be two program analysis frameworks such that \( A_1 \supset \supset Y_1 \supset A \) and \( A_2 \supset \supset Y_2 \supset A \), and \( t_1, t_2 \) are correct upper approximations of \( t \in A_1 \), \( A_2 \). The direct product \( \langle A,t,Y \rangle \) of \( \{A_1,t_1,Y_1\} \) and \( \{\overline{A_2},t_2,Y_2\} \) is defined as \( A = A_1 \times A_2 \), \( t = t_1 \times t_2 \), \( Y = \perp \in P_2 \), \( Y' = Y_1 \times Y_2 \).

- If \( t \) is both elementwise and isotone, then \( \sum (t_1 \times t_2) = \sum t_1 \times \sum t_2 \).

This definition of direct product is not satisfactory since \( Y \) is not necessarily injective and \( t \) is not necessarily optimal. Hence given a global program analysis algorithm we can get sharper information then the one obtained by the separate analyses just by revising the definition of \( A \) and \( t \) as stated in theorems 2.0.0.7 and 2.0.0.8.

**THEOREM 10.1.0.2**

Let \( \{A_1,t_1,Y_1\} \) and \( \{A_2,t_2,Y_2\} \) be two program analysis frameworks such that \( A_1 \supset \supset Y_1 \supset A_2 \), \( A_2' \supset \supset Y_2 \supset A \), \( \overline{A_2} \supset \supset \overline{Y_2} \supset A \), \( t_1 = \overline{A_1} \supset \supset \overline{Y_1} \supset t_2 \), \( t_1 = \overline{A_1} \supset \supset \overline{Y_1} \supset t_2 = \overline{A_2} \supset \supset \overline{Y_2} \supset t_2 \).

- The reduced product \( \langle A_1,t_1,Y_1 \rangle \times \langle A_2,t_2,Y_2 \rangle \) is \( \langle A_1,t_1 \rangle \times \langle A_2,t_2 \rangle \) where \( A_1 = A_1 \times A_2 \), \( t_1 = t_1 \times t_2 \), \( t_2 = \overline{A_1} \supset \supset \overline{Y_1} \supset \overline{Y_1} \).

Since \( Y_1 \supset \supset Y_2 \supset A \), \( A_1 \times A_2 \) is a representation of the space of approximations corresponding to the meet of the closure operators \( Y_1 \) and \( Y_2 \) (Theorem 8.0.1) viz. to the join \( \{t,P \mid P \in Y_1(A_1) \cup Y_2(A_2)\} \) of the Moore families \( Y_1(A_1) \) and \( Y_2(A_2) \).

280
Remark 10.1.0.4

Let $L_1(E_1)$, $L_2(E_2)$ be posets. The cardinal sum of $L_1$ and $L_2$ is the set of all elements in $L_1$ or $L_2$, considered as disjoint. When $L_1(E_1), L_1(T_1), L_2(E_2)$, $L_2(T_2)$ are complete lattices we can define the disjoint sum $L_1 + L_2$ as $L_1 \cup L_2$ with ordering $x \leq y$ if and only if $x \leq y$ or $x \leq y$. The meaning of elements of $L_1 + L_2$ can be defined as $\gamma(1) = \gamma(y_1) \wedge \gamma(z_2)$, $\gamma(x) = \gamma(y_1) = \gamma(z_2)$ if $x \leq y_1, y_2(x) \leq y_2(z_2)$, $\gamma(1) = \gamma(x)$, $\gamma(2) = \gamma(y)$, $\gamma(x) = \gamma(y)$. Even when $\gamma_1$ and $\gamma_2$ are one-to-one complete meet-morphisms, $\gamma$ may be neither one-to-one nor a complete meet-morphism. In order to satisfy assumption 5.1.0.2 the set $\gamma(L_1 + L_2)$ must be complete using theorem 5.2.0.4. Then it turns out that the least Moore family containing $\gamma(L_1 + L_2)$ is equal to $\gamma(L_1 \cup L_2)$ ($\gamma$ as defined in theorem 10.1.0.2). Therefore the use of disjoint sums amounts to the use of reduced products.

End of Remark.

10.2 Reduced Cardinal Power of Program Analysis Frameworks

The cardinal power $L_1^L$ with base $L_2(E_2, I_2, T_2, \lambda, \mu_2, \nu_2)$ and exponent $L_1(E_1, I_1, T_1, \lambda_1, \mu_1)$ (henceforth noted $\text{Iso}(L_1 L_2(E_1, I_1, T_1, \lambda, \mu_2))$) is the set of all isomorphisms maps from $L_1$ to $L_2$ with $f \in \text{Iso}$ and only if $f(x) \in \mu_2(g)(x)$ for all $x \in L_1$. Two program analysis frameworks $(A_1, T_1, \mu_1)$ and $(A_2, T_2, \mu_2)$ can be combined by letting $g \in \text{Iso}(L_1 L_2)$ mean that for all $x \in A_1$, $g(x) \in \mu_2(g)(x)$ holds whenever $f(x)$ holds.

Theorem 10.2.0.1

The reduced cardinal power with base $(A_2, T_2, \lambda_2)$ and exponent $(A_1, T_1, \mu_1)$ is $(A_1, T_1, \lambda_2)$ where $\lambda_2 = \lambda_1 \circ \lambda_2 \circ \lambda_1 \circ A_1 \circ A_2$.

Example 10.2.0.2

The analysis of the program:

\[
\begin{align*}
\gamma(1) & = \gamma(y_1) \wedge \gamma(z_2), & \gamma(y_1) & = \gamma(y_2) = \gamma(z_2), & \gamma(1) & = \gamma(x), & \gamma(2) & = \gamma(y), & \gamma(x) & = \gamma(y).
\end{align*}
\]

End of Example.
using the reduced cardinal product of $A_1$ and $A_2$ yields no information since no relationship can be discovered between $b$ and $x$.

Following theorem 10.2.0.1 we determine that if $g(x_1, x_2) = \gamma(g_1(y_1, \tau(x_1)) \oplus g_2(y_2), \tau(x_2)) \gamma g_1, g_2, \tau(x_1), \tau(x_2))$. Therefore $g_1(x_1) = \gamma(x_1) \oplus g_2(x_2)$, $g_2(x_2) = g_1(x_1), g_2(x_1), \tau(x_2)$, and $g(x_1, x_2) = \gamma g_1, g_2, \tau(x_1), \tau(x_2))$. It follows that $g(x_1, x_2)$ is isomorphic to $(\ell_1, \ell_2, \tau(x_1), \tau(x_2))$ for $A_1 \times A_2$.

The system of equations associated with the above program and the entry specification $\lambda b.2$ is then:

$g_1 = \lambda b.f \cdot b \cdot then \cdot + \cdot else \cdot i_1 \cdot f^t$
$g_2 = \lambda b.\cdot f \cdot then \cdot + \cdot g_1(f) \cdot \ell_1 \cdot g_2(f) \cdot else \cdot i_2 \cdot f^t$
$g_3 = \lambda b.\cdot \ell 0 \cdot g_2(f)$
$g_4 = \lambda b.\cdot f \cdot then \cdot g_4(f) \cdot \ell_0 \cdot g_2(f) \cdot \ell_0 \cdot g_2(f) \cdot else \cdot b \cdot f \cdot then \cdot g_4(f) \cdot \ell_0 \cdot g_2(f) \cdot \ell_0 \cdot g_2(f) \cdot f^t$
$g_5 = \lambda b.\cdot f \cdot then \cdot g_5(f) \cdot \ell_0 \cdot g_2(f)$

where $\ell_0(1) = i_2$, $\ell_0(0) = \ell 0(-)$, $\ell 0(2) = \ell 0(-)$, $\ell 0(1) = \ell 0(-)$, $\ell 0(0) = \ell 0(-)$, $\ell 0(2) = \ell 0(-)$.

End of Example.

### 11. REFERENCES


Ore O. [1943], *Combination of closure relations*, Ann. of Math., 44 (1943), 514-533.


