SYSTEMATIC DESIGN OF PROGRAM ANALYSIS FRAMEWORKS

Patrick Cousot* and Radhia Cousot**
Laboratoire d'Informatique, U.S.M.G., BP.53X
38041 Grenoble cedex, France

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], Kildall[73], Rosen[78], Tarjan[76], Wegbreit[75]) have introduced abstract approaches to program analysis which are tantamount to the use of a program analysis framework \((A, \mathcal{T}, Y)\) where \(A\) is a lattice of (approximate) assertions, \(\mathcal{T}\) 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 anisotone 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 characterization 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 verified 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[78]'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 closure of a family of approximate assertions. It is

This work was supported in part by (*). CNRS, Laboratoire Associé n°7 and ATP-Informatique D3119.
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 isotonous 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 π is a pair (V, G) where G is a program graph and V is the universe in which the program variables take their values.

The set I of elementary commands consists in elementary tests and elementary assignments: $L = I + L + L$. An elementary test $c \in I$ is a total map from dom(c) \subseteq V into $B = \{true, false\}$. An elementary assignment (L) \in L is a total map from dom(L) \subseteq V into V.

A program graph G is a tuple $(n_E, n_i, n_o, C)$ where n is the number of vertices (therefore n≥1), $E \subseteq I \times I$ is the (non-empty) set of edges, $n_i \in [1, n]$ is the entry point, $n_o \in [1, n]$ is the exit point and $C \in E \rightarrow L$ defines the command $C(c, i, j)$ associated with each $c(i, j)$ in E. Let $pred(c) \in [1, n] \cup \{\}, succ(c) \in [1, n] \cup \{\}$ be $\lambda_i.\{\{1, n\} \subseteq I \times I \} \in [1, n] \cup \{\},$ then we assume that $pred(c) = \emptyset$, $succ(c) = \emptyset$ and for any $v \in [1, n]$, $\forall v \in [1, n]$, $v \in \emptyset$ and $\forall v \in [1, n]$, $v \in \emptyset$.

Example 2.1.1: The program:

\begin{align*}
\{1\} & \text{begin} \\
\{2\} & \text{while } x \leq 100 \text{ do } \{x \text{ is an integer variable} \} \\
\{3\} & \text{end} \\
\{4\} & \text{end}
\end{align*}

will be represented by its program graph:

$$
\begin{align*}
\lambda x.(x \leq 100) & \quad \lambda x.(x > 100) \\
\lambda x.(x = 100) & \quad \lambda y.(y > 100)
\end{align*}
$$

3. DEDUCTIVE SEMANTICS OF PROGRAMS

3.1 Forward Semantics

The forward semantic analysis of a program π 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 V into $\mathcal{R}$. The set $A = (V + R)^{\omega} \times V \times \{true, false\}$ of assertions is a complete boolean lattice partially ordered by the implication $\rightarrow$.

Let sp($\Sigma$)$\rightarrow$A be Floyd's[67]'s strongest post-condition derived from the pre-condition P$\rightarrow$A for the elementary command $\Sigma \in L$. We assume that the operational semantics of the elementary commands is such that for an elementary test we have:

$$
sp(c) = \lambda p \in A.\{\lambda x \in V.\{P(x) \land \neg x \in \text{dom}(\alpha) \land q(x)\}\}
$$

whereas for an elementary assignment $\alpha$ we have:

$$
sp(\alpha) = \lambda p \in A.\{\lambda x \in V.\{P(x) \land \neg x \in \text{dom}(\alpha) \land x = \alpha(y)\}\}
$$

(Notice that for all $\Sigma \in L$, sp($\Sigma$) is a complete join-morphism (i.e., $\text{WA} \rightarrow A$, sp($\Sigma(\text{WA})$) = sp($\Sigma$)(sp($\text{WA}$))).

We assume that the operational semantics of the program π is such that at each program point if $i \in [1, n]$ the invariant assertion $P_i$ which characterizes the set of states which are the descendants of the input states satisfying a given entry assertion $\Phi$ is the merge over all paths analysis of π using $sp$ and $\phi$. $P$ is the least fixpoint of $\Sigma P \rightarrow$A of the system of equations $P = F(\Sigma P \rightarrow A)(\pi)$ associated with the program π using $sp$ and $\phi$.

270
Example 3.1.0.1

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

\[
\begin{align*}
P_1 &= \phi \\
P_2 &= up(\lambda x. (x \leq 100))(P_1 \lor P_3) \\
P_3 &= up(\lambda x. (x \leq 100))(P_2) \\
P_4 &= up(\lambda x. (x \leq 100))(P_3)
\end{align*}
\]

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

\[
\begin{align*}
P_1 &= \lambda x. (x = 1) \\
P_2 &= \lambda x. (1 \leq x \leq 100) \\
P_3 &= \lambda x. (2 \leq x \leq 100) \\
P_4 &= \lambda x. (x = 101)
\end{align*}
\]

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[59]-Dijkstra[78]'s backward predicate transformer:

\[
\begin{align*}
up(q) &= \lambda x. (\lambda x. (P(X) \land \lambda x. \text{dom}(q) \land q(x))) \\
up(a) &= \lambda x. (\lambda x. (\lambda x. \text{dom}(a) \land P(x)))
\end{align*}
\]

(Notice that \(\text{seg}\), \(up(S)\) is a complete join and meet morphism and the inverted program graph \(G' = (\eta', \text{E}', \eta', \text{C}', \lambda')\) where \(\text{E}' = \{(i, j) : (j, i) \in \text{E}\}, \lambda' = \lambda, \text{C}' = \{(i, j) \in \text{C}\}\).)

Example 3.2.0.1

The inverted program graph corresponding to 2.0.1 is:

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

The corresponding system of backward semantic equations is:

\[
\begin{align*}
P_1 &= up(\lambda x. (x \leq 100))(P_2) \lor up(\lambda x. (x > 100))(P_3) \\
P_2 &= up(\lambda x. (x \leq 100))(P_1) \\
P_3 &= up(\lambda x. (x \leq 100))(P_3) \\
P_4 &= up(\lambda x. (x \leq 100))(P_3) \\
P_5 &= \phi
\end{align*}
\]

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:

\[
\begin{align*}
P_1 &= \lambda x. (x \leq 101) \\
P_2 &= \lambda x. (x \leq 100) \\
P_3 &= \lambda x. (x \leq 101) \\
P_4 &= \phi = \lambda x. (x = 101)
\end{align*}
\]

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 assertions \(P\) defined by the deductive semantics.

**Definition 4.0.1**

If \(P, Q \in A\) then "\(Q\) approximate \(P\)" iff \(P \Rightarrow Q\).

This definition of "approximate" 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 \(A(\eta, \text{E}, \text{C}, \lambda)\). The meaning of the elements of \(A\) is specified by \(\eta\) (too often implicit) order morphism \(\gamma\) mapping \(A\) to a subset of assertions \(\gamma(A) \subseteq A\). The intention is that \(\gamma(A)\) 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 \(s\) is associated an isotone map \(t(s)\) from \(A\) to \(A\). The intent is that \(t(s)\) is an approximate predicate transformer such that \(t(s)(A)\) represents the propagation of the information \(s\) through the statement \(s\).

The ideal merge over all paths program analysis [Graham & Wegman76], Kam & Ulman[77], Rosen[76], Tarjan[76] is often approximated by a fixpoint solution [Cousot & Cousot[77a], Jones & Muchnick[78], Kaplan & Ullman[78], Kildal[79], Tenenbaum[74], Wegbreit[75]]. A fixpoint system of isotone equations \(X = \gamma(x)\) where \(\gamma\) (\(A^0 \Rightarrow A^0\)) is associated with the program graph. The approximate invariant assertions are generated by computing iteratively the least fixpoint of \(\gamma\) starting from the infimum of \(A^0\) and using any chaotic or asynchronous iteration strategy [Cousot[77b]] or the least fixpoint is approximated from above using an extrapolation technique in order to accelerate the convergence of the iterates whenever \(A\)
5. DESIGN OF A SPACE OF APPROXIMATE ASSERTIONS

5.1 A Very Reasonable Assumption

Assume that for a specific-purpose analysis of programs a subset \( \mathcal{P} \) of assertions has been found to provide meaningful information.

Since any invariant assertion \( \mathcal{P} \) for any program must have an upper approximation \( Q \) in \( \mathcal{R} \), the set \( \{Q \in \mathcal{R} : P \Rightarrow Q\} \) must be non-empty.

Let \( \mathcal{P} \) be an assertion and assume that we want to analyze a program \( P \) using the merge over all paths semantic analysis and an entry condition \( Q \) which is an upper approximation of \( P \) in \( \mathcal{R} \). What is the best choice for \( Q \)? It is clear that if \( P \Rightarrow Q \Rightarrow Q \) then \( P \Rightarrow Q \Rightarrow Q \) and by isotony the analysis \( \mathcal{I}(P,F(P,F(Q))) \) is more precise than \( \mathcal{I}(P,F(Q)) \).

Hence \( Q \) must be a minimal upper approximation of \( P \) in \( \mathcal{R} \) (that is such that \( P \Rightarrow Q \Rightarrow Q \Rightarrow Q \Rightarrow Q \)). Assume that the set \( U \) of minimal upper approximations of \( P \) in \( \mathcal{R} \) has a cardinality greater than 1. What is the "best" possible choice for \( Q \) in \( U \)? If \( Q_1, Q_2 \in \mathcal{R} \) then \( Q_1 \neq Q_2 \) and \( Q_1 \) are not necessarily comparable so that \( \mathcal{I}(P,F(P,Q_1)) \) and \( \mathcal{I}(P,F(P,Q_2)) \) may be not comparable. Hence "best" cannot be defined using the preciseness criterion provided by the ordering \( \Rightarrow \). The only way to determine which of the two alternatives will be the most useful in order to answer a given set of application dependent questions about the program is to try both of them. Also the best choice may vary from one program to another. This try and see choice method leads to a non-deterministic analysis method which is unacceptable because of obvious efficiency considerations. Therefore it is reasonable to choose \( \mathcal{R} \) such that Card(\( U \)) = 1.

Example 5.1.1

Assume that \( A = \{Z \} \) where \( Z \) is the set of integers and \( \mathcal{R} = \{\lambda(x,y), (P, x) \Rightarrow (P, y) \} \). The assertion \( P \Rightarrow \lambda(x,y),(x > 0 \land y > 0) \) has two distinct minimal upper approximations in \( \mathcal{R} \) namely \( Q_1 = \lambda(x,y),(x > 0 \land y > 0) \) and \( Q_2 = \lambda(x,y),(x < 0 \land y < 0) \). Now the choice of the most useful upper approximation of the entry assertion \( P \) is program-dependent. For example the best choice is \( Q_1 \) for the program \( x := x + y \). This positive declaration can only be justified by performing the two semantic analyses (i.e., \( sp(x := x + y)(Q_1) = \lambda(x,y),(x < 0 \land y < 0) \) and \( sp(x := x + y)(Q_2) = \lambda(x,y),(x > 0 \land y > 0) \)) and next comparing them. Since these analyses are not related by the ordering \( \Rightarrow \), the comparison criterion must be application dependent. For example using \( Q_1 \) we can prove that \( sp(x := x + y)(Q_1) = \lambda(x,y),(x > 0 \land y > 0) \) whereas this is impossible with \( Q_2 \). On the contrary the best choice is \( Q_2 \) for the program \( x := x + y \). Since \( sp(x := x + y)(Q_2) = \lambda(x,y), (x < 0 \land y < 0) \) which implies \( \lambda(x,y),(x > 0 \land y > 0) \) whereas \( sp(x := x + y)(Q_1) = \lambda(x,y),(x < 0 \land y < 0) \) does not imply \( \lambda(x,y),(x > 0 \land y > 0) \).

End of Example.

If any program must have an analysis which can be approximated from above using \( \mathcal{R} \) and the process for deriving the most useful approximate analysis of any program is required to be deterministic then it is reasonable to make the following:

ASSUMPTION 5.1.0.2

The set \( \mathcal{R} \) of approximate assertions must be chosen such that for all \( P \in \mathcal{R} \) the set \( \{Q \in \mathcal{R} : P \Rightarrow Q\} \) of upper approximations of \( P \) in \( \mathcal{R} \) has a least element.

THEOREM 5.1.0.3

For all \( P \in \mathcal{R} \) the set \( \{Q \in \mathcal{R} : P \Rightarrow Q\} \) has a least element if and only if \( \mathcal{R} \) is a Moore family (i.e., \( \mathcal{R} \) contains the supremum of \( A \) and is closed under conjunction).

5.2 The Approximation Operator

DEFINITION 5.2.0.1 Approximation Operator

\( \rho = \rho(\mathcal{R} : P \Rightarrow Q) \)

(0) is the least upper approximation of \( P \) in \( \mathcal{R} \). Since \( \mathcal{R} \) is a Moore family it follows from Monteiro & Reishein [42, Th. 5.3 and 5.1] that:

THEOREM 5.2.0.2

(1) \( \rho \) is an upper closure operator (that is \( \rho \) is isotone if \( P \Rightarrow Q \) then \( \rho(P) \Rightarrow \rho(Q) \), extensive (for all \( P \in \mathcal{R} \), \( \rho(P) \Rightarrow \rho(P) \)) and idempotent (\( \rho(P) = \rho(P) \)).

(2) \( \rho(A) = \mathcal{R} \)

(3) \( \rho \) is the unique upper closure operator on \( \mathcal{R} \) such that \( \rho(A) = \mathcal{R} \).

Since \( \mathcal{R} \) is equal to the image of the complete lattice \( \mathcal{A} = \{\lambda(x,y), (x < 0 \land y < 0) \} \) by the upper closure operator \( \rho \) we derive from Ward [42, Th. 4.1] the following:

THEOREM 5.2.0.3

(1) \( \lambda(x,y), (x < 0 \land y < 0) \) is a complete lattice \( \rho(\mathcal{A}) = \rho(\lambda(x,y), (x < 0 \land y < 0)) \).

(2) \( \rho \) is a quasi-complete join-morphism (i.e., \( \rho(\mathcal{A}) = \rho(\mathcal{R}) \)).

(3) \( \mathcal{R} \) is a complete sub-lattice of \( \mathcal{A} \) if and only \( \rho \) is a complete join-morphism (i.e., \( \rho(\mathcal{R}) = \rho(\mathcal{A}) \)).
If the initial choice of $\mathcal{A}$ does not satisfy assumption 5.1.0.2 we can use the following:

**Theorem 5.2.0.4**

| $\mathcal{A} \in A$, the upper closure operator $\rho$ on $A$ such that $\rho(A)$ is the least Moore family containing $\mathcal{A}$ is: |
| $\rho = \lambda P. \lambda (Q: (\lambda x. true) : P \Rightarrow Q)$ |
| $\rho(A) = \lambda S : P \subseteq (\lambda x. true) \cup S \Rightarrow P$ |

**Example 5.2.0.5**

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

![Diagram of lattice]

The corresponding approximation operator is:

\[ \rho = \lambda P. if P = \lambda u. false then \lambda u. false \\
elif P = \lambda u. u \neq 0 then \lambda u. u \neq 0 \\
elif P = \lambda u. u \neq 0 then \lambda u. u \neq 0 \\
else \lambda u. true \triangleright P \]

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 $A(\epsilon, i, T, U, P)$ such that the smaller algebras $\mathcal{A} = \rho(A) \Rightarrow (\lambda x. false, \lambda x. true, \lambda x. true, \lambda x. \rho(\epsilon), \lambda x, \lambda y)$ and $A(\epsilon, i, T, U, P)$ be isomorphic. Let $\gamma \in (\mathcal{A}, \mathcal{A})$ be the corresponding lattice isomorphism. Let $\alpha \in (A \Rightarrow A)$ be $\lambda S. (\rho(\psi), \lambda x. true)$ and $\alpha = \gamma^{-1} \rho$ then:

- $\langle x, \psi \rangle$ is a pair of adjoint functions
- $\alpha$ is onto, $\gamma$ is one-to-one

Reciprocally the approximation process can be defined by the lattice $A(\epsilon, i, T, U, P)$ and a pair of adjoint functions. Such a pair $\langle x, \psi \rangle$ defines a Galois connection between $A$ and the dual of $A$.

### Definition 5.3.0.3

Let $L_1(\epsilon_1)$ and $L_2(\epsilon_2)$ be posets, $\alpha \in (L_1 \Rightarrow L_2)$, $\gamma \in (L_2 \Rightarrow L_1)$. The pair $\langle x, \psi \rangle$ defines a Galois connection between $L_1$ and $L_2$ if and only if:

1. $\alpha$ is antitone ($\forall x_1, x_2 \in L_1, \{x_1 \in L_1 \Rightarrow \{\alpha(x_1) \neq \alpha(x_2)\}\}$)
2. $\gamma$ is antitone ($\forall y_1, y_2 \in L_2, \{y_1 \in L_2 \Rightarrow \{\gamma(y_1) \neq \gamma(y_2)\}\}$)
3. $\lambda x \in L_1. (x) \in L_1 \Rightarrow \gamma \alpha = \gamma \gamma \alpha$
4. $\lambda y \in L_2. (y) \in L_2 \Rightarrow \alpha \gamma = \alpha \alpha \gamma$

The above conditions (3) and (4) are equivalent to:

$\forall x \in L_1. \gamma \alpha \in L_2. (x \in L_1 \Rightarrow \gamma \alpha) \Rightarrow (\alpha(x) \in L_2)$(Birkhoff[67]), hence we have:

**Theorem 5.3.0.4**

Let $L_1(\epsilon_1)$, $L_2(\epsilon_2)$ be posets, $\alpha \in (L_1 \Rightarrow L_2)$, $\gamma \in (L_2 \Rightarrow L_1)$. $\langle x, \psi \rangle$ is a pair of adjoint functions if and only if $\langle x, \psi \rangle$ defines a Galois connection between $L_1$ and $L_2$ if $L_1(\epsilon_1)$ and $L_2(\epsilon_2)$ are isomorphic, i.e., if $\alpha$ and $\gamma$ are isotonous, $\lambda x \in L_1 \gamma \alpha = \gamma \alpha \gamma \alpha$

**Theorem 5.3.0.4**, Ore[44,Th.21] and Pickert[52] imply:

**Corollary 5.3.0.5**

Let $L_1(\epsilon_1)$ and $L_2(\epsilon_2)$ be posets and $\alpha \in (L_1 \Rightarrow L_2)$, $\gamma \in (L_2 \Rightarrow L_1)$ be adjoint functions:

1. $\gamma \alpha \in$ an upper closure operator on $L_1$, $\gamma \alpha \psi$ is a lower closure operator on $L_1$
2. $\alpha \gamma \in$ reductive (i.e., isotonous, $\alpha \gamma = \alpha \gamma \alpha$ and idempotent)

Moreover if $L_1(\epsilon_1, T_1, U_1, P_1)$ and $L_2(\epsilon_2, T_2, U_2, P_2)$ are complete lattices then:
- \gamma = \lambda x \in L_1. \exists y \in L_2. \alpha(x, y)

\alpha is an isomorphism from the complete lattice \(L_2\) into the complete lattice \(L_1\) the inverse of which is \(\gamma\).

(3) \alpha is one-to-one if and only if \(\gamma\) is onto and if and only if \(\gamma \circ \alpha = \lambda x \in L_1. x\)

We use the notation \(L_1 \triangleright \alpha, \gamma \triangleright L_2\) to state that \(L_1\) and \(L_2\) are connected by the pair <\(\alpha, \gamma\)> of adjoint functions which are respectively surjective and injective. If \(\alpha\) is a complete join-isomorphism from \(L_1\) onto \(L_2\), then \(\gamma\) is an onto one-one complete meet-isomorphism from \(L_2\) into \(L_1\). We write \(L_1 \triangleright \alpha, \gamma \triangleright L_2\) and assume that the adjointed \(\gamma(\alpha)\) is determined by \(5.3.0.5. (3.2) \, (5.3.0.5. (3.1))\).

In the literature the most useful method for defining a program analysis framework is to specify the complete lattice \(A\) of program assertions and to informally describe the meaning of its elements, e.g., constant propagation, Kildall[73], Kam & Suliman[77]). Hence the function \(\gamma A\) of \((A \rightarrow A)\) remains implicit.

It is often the case that \(A\) is only assumed to be a complete join-semi-lattice \((A, \sqcup, \sqcap, \bot, 1, \top)\) or a dual meet-semi-lattice for some authors, but since an infimum is adjoined to \(A\), it is in fact a complete lattice (even when the meet-operation is not used or what is called meet is not \(\sqcap\) e.g., Wegbreit[75]).

When \(\gamma A\) is a complete lattice but not a complete meet-lattice the set \(Y(A)\) does not fulfill assumption 5.1.0.2 with the consequences examined at paragraph 5.1. The design of \(Y(A)\) and \(A\) can be revised as stated by theorem 5.2.0.4. When \(\gamma A\) is a complete meet-lattice but not one-to-one, several distinct elements of \(A\) have the same minimal. Since this is useless, the design of \(A\) and \(Y\) can be revised as follows:

**THEOREM 5.3.0.7**

Let \((A, \sqcup, \sqcap, \bot, 1, \top)\) be a complete lattice and \(\gamma A\) be a complete meet-lattice. Let \(\sigma (A \rightarrow A)\) be \(\lambda x \in \bigcup \{y \in A : y(x) = \gamma(y)\}, \bigcup = \sigma(A), \gamma = \{y | \gamma x\} : \sigma x A, Y(x) = \gamma(\sigma x)\).

- \(\forall x A, Y(x) = \gamma(\sigma x)\)
- \(\sigma\) is a lower closure operator on \(A\)
- \(\gamma\) is a one-to-one complete meet-lattice from the complete lattice \((A, \sqcup, \sqcap, \bot, 1, \top)\) into \(\sigma x A, Y(x) = \gamma(\sigma x)\)

Since \(Y(A) = \gamma(\sigma x A)\), \(A\) and \(\sigma x A\) have the same expressive power. Among all subsets of \(A\), \(\sigma x A\), one has the minimal cardinality,

**THEOREM 5.3.0.8**

(1) \(\forall x A, \gamma(\sigma x) = \gamma(\sigma x)\) \(\Rightarrow \gamma(\sigma x) = \gamma(\sigma x)\)
(2) \(\forall x A, \gamma(\sigma x) = \gamma(\sigma x)\) \(\Rightarrow \gamma(\sigma x) \leq \gamma(\sigma x)\)
(3) \(\forall x A, \forall y A, \gamma(x) = \gamma(y)\) \(\Rightarrow \gamma(y) \leq \gamma(y)\)

6. EQUIVALENT METHODS FOR SPECIFYING A SPACE OF APPROXIMATE ASSERTIONS

A space of approximate assertions can be specified either by a Moore family or by an upper closure operator. Moore families can be characterized using definition 5.1.0.2 or theorem 5.1.0.3 and 5.2.0.4. In addition to theorems 5.2.0.2 (1) and 5.3.0.8 we now study and exemplify various equivalent methods which can be used to define an upper closure operator.

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

**THEOREM 6.1.0.1**

Let \(L(E, l, E, l, \bot, \top, 1, \top)\) be a complete lattice and \(\forall f \in (L, \top)\).
- \(\forall x \in L \Rightarrow f(x) = \lambda y [\lambda x \in f(y) \land \forall x \in f(x)]\)
- \(\forall f \in L \land f \leq \top\)
- \(\forall f \in L \land f \leq \top\)
- \(\forall f \in L \land f \leq \top\)
- \(\forall f \in L \land f \leq \top\)
- \(\forall f \in L \land f \leq \top\)

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 not a closure operator [Ore64]). 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 \(L(E, l, E, l, \bot, \top, 1, \top)\) be a complete lattice, \(A\) an upper closure operator on \(L\) and \(\eta A\) be an upper closure operator on \(R\). Then \(\eta A\) is an upper closure operator on \(L\) and \(R\).

**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 \(A = D^m\), let \(A_1\) be \((V + B)\) and \(A_1\) be \((D + B)\). Let us define:

\(\forall y \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)

\(\forall v_j \in [1, m], \sigma_j = \lambda x \in D^m [\lambda x \in D^m [x_j = v_j]\)
\( \sigma \) is an upper closure operator on \( A_m \) and an assertion \( \text{Pe}_A \) does not state relationships among the program variables if and only if \( \text{Pe}(P) = \perp \). The approximate assertions on each individual program variable \( x_i \) are next defined using an upper closure operator \( \rho_j \) on \( A_j \). The induced closure operator \( \rho \) on \( A_m \) is defined by \( \rho(P) = \{ \lambda(x_1, \ldots, x_m) \mid \exists \rho_j(P_j)(x_j) \text{ where } P = \{ x_1, \ldots, x_m \}. \}

A complete upper closure \( \lambda \) on \( A_m \) is equivalent to \( \rho \) if and only if \( \sigma(A_m) = \{ x \mid \rho(x) = \perp \} \). It follows from theorem 6.2.0.1 that the composition:

\[
\rho \circ \sigma = \lambda \text{Pe}_A \lambda(x_1, \ldots, x_m) \in \rho(P) \exists \rho_j(P_j)(x_j)
\]

is an upper closure operator on \( A_m \).

**End of Example.**

**6.3 Definition of a Space of Approximate Assertions by Means of a Complete Join Congruence Relation**

Considering the equivalence relation \( \rho(x) \) induced by an upper closure operator \( \rho \) on \( A \) and defined as \( \rho \in \equiv \rho(P) \) if and only if \( \rho(P) = \rho(Q) \), the approximation proofs can be understood as essentially consisting in partitioning the space of assertions so that no distinction is made between equivalent assertions which are all approximated by a representative of their equivalence class. Since the approximation is from above and at least one must exist (assumption 5.1.0.2) all not equivalent assertions are acceptable:

**DEFINITION 6.3.0.1**

Let \( L(E,\cup,\cap,T,U) \) be a complete lattice. A binary relation \( \theta \) on \( L \) is a complete join congruence relation if and only if:

1. \( \theta \) is an equivalence relation.
2. \( \theta \) satisfies the join-substitution property:
   \[
   \forall x, y, u, v, w \in L, x \cap y \in \theta \Rightarrow x \cup (y \cap u) \in \theta
   \]
3. \( \theta \) satisfies the join-completeness property:
   \[
   \forall x \in L, x \cup \{x\} \theta \Rightarrow \{x\} \cup y \in \theta \text{ for all } y \in L.
   \]

**THEOREM 6.3.0.2**

If \( \rho \) is an upper closure operator on \( L(E,\cup,\cap,U,T) \) and \( \forall x, y \in L, x \equiv y \in \theta \) if and only if \( \rho(x) = \rho(y) \) then:

1. \( \rho \) is a complete join-congruence relation on \( L \)
2. \( \rho \) is an upper closure operator on \( L \).

Reciprocally a complete join-congruence relation on \( A \) defines an upper closure operator on \( A \) whence a space of approximate assertions:

**THEOREM 6.3.0.3**

Let \( \theta \) be a complete join-congruence relation on the complete lattice \( L(E,\cup,\cap,T,U) \). \( \lambda x \cup \{x\} \in \theta \) is an upper closure operator on \( L \).

(Similar results were already proved in Coutos & Cousinot[71] except that the above definition of complete join-congruence relations has been substantially simplified.)

The following result can sometimes facilitate

**THEOREM 6.3.0.4**

A reflexive and symmetric binary relation \( R \) on a complete lattice \( L(E,\cup,\cap,T,U) \) is a join-congruence relation iff the following three properties are satisfied for \( x, y, z \in L \):

1. \( x \cap y \in \theta \Rightarrow \{x\} \cup \{y\} \in \theta \)
2. \( x \cup y \in \theta \Rightarrow x \cap y \in \theta \)
3. \( x \cup y \in \theta \Rightarrow x \cap z \in \theta \)

**Example 6.3.0.5**

Let \( V \) be a non-empty set of integers included between two bounds \( \alpha \) and \( \beta \) (\( \alpha \leq \beta \)). The binary relation \( \theta \) on \( A = (V \cup \beta) \)

\[
(\{x \in V \mid \alpha \leq x \leq \beta \}) \Rightarrow \{x \in V \mid \alpha \leq x \leq \beta \}
\]

The quotient lattice \( L/\theta \) is isomorphic to \( A \) where \( \rho \) is the upper closure operator induced by \( \theta \):

\[
\theta = \lambda x \in L, x \in \rho(\{x\})
\]

**6.4 Definition of a Space of Approximate Assertions by Means of a Family of Principal Ideals**

The equivalence classes of the complete join-congruence relation \( \rho \) induced by a closure operator \( \rho \) have the following property:

**THEOREM 6.4.0.1**

Let \( \theta \) be a complete join-congruence relation on the complete lattice \( L(E,\cup,\cap,T,U) \). Then \( x \in L \) is a complete join-similarity lattice of \( L \) if and only if \( x \in L \) is a complete join-similarity lattice of \( L \).

The notion of a principal ideal is the same as in algebra, except that it is defined in a lattice rather than in a ring. In a lattice \( L \), a principal ideal is a subset \( I \) of \( L \) such that \( I \) is an ideal and \( I \) is a sublattice of \( L \).

Here is another representation of convex sub-lattices of \( L \) (which can be compared with Gröbner[71]'s representation of convex sub-lattices):

An **ideal** is a nonvoid subset \( J \) of \( L \), where \( L(E,\cup,\cap,T,U) \)

\[
(\{x \in L \mid x \in J \}) \Rightarrow \{x \in L \mid x \in J \}
\]

It is easy to show that \( J \) is an ideal, if and only if \( \alpha \leq \beta \) and \( \beta \leq \gamma \), and \( \alpha \leq \beta \) and \( \beta \leq \gamma \) for all \( \alpha, \beta, \gamma \). Since \( L \) has an infimum \( \bot \), the intersection of all ideals of \( L \) is the ideal of \( L \).

Given an element \( a \) in a lattice \( L \), the set \( \{a \uparrow \cup \{a\} \} \) is evidently an ideal, if it is called a principal ideal of \( L \). If \( \alpha \) is an element of \( L \), then \( \{a \uparrow \cup \{a\} \} \) is the dual notion of the dual principal ideal.
7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER
INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

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

7.1 A Reasonable Definition of Correct Approximate Predicate Transformers

At paragraph 3, given \((\mathcal{V}, A, \tau)\) minimal assertion which is invariant at point \( i \) of a program \( \pi \) with entry specification \( \phi \) \( A \) was defined as:
\[
P_i = \forall \ p \in \text{Path}(i)
\]
Therefore the minimal approximate invariant assertion is the least upper approximation of \( P_i \) in \( \mathcal{V} \) that is:
\[
\rho(P_i) = \rho(\forall \ p \in \text{Path}(i))
\]
Even when \( \text{Path}(i) \) is a finite set of finite paths the evaluation of \( \tau(p)(\phi) \) is hardly machine-implementable since for each path \( p = a_1 \ldots a_m \) the computation sequence \( X_0 = \phi, X_1 = \tau(C(a_1))(X_0), \ldots, X_m = \tau(C(a_m))(X_{m-1}) \) does not necessarily only involve elements of \( A \) and \( (\mathcal{V}, A) \). Therefore using \( \tau_0 = \tau \) and \( \tau(\mathcal{V}, A) \) a machine representable sequence \( X_0 = \phi, X_1 = \tau(C(a_1))(X_0), \ldots, X_m = \tau(C(a_m))(X_{m-1}) \) is used instead of \( X_0, \ldots, X_m \) which leads to the expression:
\[
Q_i = \rho(\forall \ p \in \text{Path}(i))
\]
The choice of \( \tau_0 \) and \( \phi_0 \) is correct if and only if \( Q_i \) is an upper approximation of \( P_i \) in \( \mathcal{V} \) that is if and only if:
\[
\rho(\forall \ p \in \text{Path}(i)) \Rightarrow \rho(\forall \ p \in \text{Path}(i))
\]
In particular for the entry point we must have \( \phi_0 \Rightarrow \phi_0 \) so that we can state the following:

DEFINITION 7.1.0.1

(1) An approximate predicate transformer \( \tau \) (of \( (\mathcal{V}, A) \)) is said to be a correct upper approximation of \( \tau \) in \( (\mathcal{V}, A) \) if and only if for all \( \phi \in \mathcal{V}, \mathcal{V} \) such that \( \phi \Rightarrow \phi \) and program \( \pi \) we have:\n\[\text{MOP}_{\pi}(\tau, \phi) \Rightarrow \text{MOP}_{\pi}(\tau, \phi)\]
(2) Similarly if \( A \) and \( \mathcal{V} \), \( \tau \) (of \( (\mathcal{V}, A) \)) is said to be a correct upper approximation of \( \tau \) in \( (\mathcal{V}, A) \) if and only if for all \( \phi \in \mathcal{V}, \mathcal{V} \) such that \( \phi \Rightarrow \phi \) \[\text{MOP}_{\pi}(\tau, \phi) \Rightarrow \text{MOP}_{\pi}(\tau, \phi)\]

(i.e., \( \text{MOP}_{\pi}(\tau, \phi) \Rightarrow \text{MOP}_{\pi}(\tau, \phi)\))

This global correctness condition for \( \tau \) is very difficult to check since for any program \( \pi \) and any
THEOREM 7.1.0.2

(1) \( \exists e (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) is a correct upper approximation of \( e \in (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) in \( (A=e(A)) \) iff \( \{\forall S \subseteq L, \forall P \in A, \tau(S)(P) \Rightarrow \bar{e}(S)(P)\} \).

(2) \( \exists e (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) is a correct upper approximation of \( t \in (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) in \( (A=t(A)) \) (where \( A=t(A), \forall y \in A \) iff \( \{\forall S \subseteq L, \forall P \in A, \alpha(S)(y(P)) \in t(S)(P)\} \).

If \( \exists e (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) is a correct upper approximation of \( t \in (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) in \( A=t(A) \) we have \( \text{MOP}_n(t, \psi) \Rightarrow \text{MOP}_n(e, \psi) \) where \( \text{MOP}_n(t, \psi) \Rightarrow \text{MOP}_n(e, \psi) \).

The cases when equality holds are not easy to distinguish. Yet the following sufficient condition turns

DEFINITION 7.2.0.1

If \( t_1, t_2 \) are correct upper approximations of \( t \in (L \rightarrow (\bar{A} \rightarrow \bar{A})) \) in \( A=t(A) \) then we say that \( t_1 \) is better than \( t_2 \) iff for all \( \forall P \in A \) and all programs \( \tau \),
According to theorem 7.2.0.4 the best upper approximation of $ap(\lambda x y, y, x(y))$ in $A$ is $t = a < ap(\lambda x y, y, x(y))$. If $P \wedge A$ equals $\langle 1, 1 \rangle$ then $t(P) = \langle 1, 1 \rangle$ else $P = \langle \alpha, \beta, [\alpha, \beta, d] \rangle$ where $\alpha$ and $\beta$ in which case $t(P) = (\alpha, \beta) \wedge x, y, z, \lambda x y, z (\alpha x y z \wedge \beta x y z) = \lambda x (\alpha x y z \wedge \beta x y z)$. The same way $t(\langle 1, 1 \rangle, \alpha y, \lambda x y, z) = \langle 1, 1 \rangle, \alpha y, \lambda x y, z) = \lambda x (\alpha y x y z \wedge \beta x y z)$. Therefore $t(P) = 1, \langle 1, 1 \rangle, \alpha y, \lambda x y, z)$ and then $\langle 1, 1 \rangle, \alpha y, \lambda x y, z) = \langle 1, 1 \rangle, \alpha y, \lambda x y, z) = \lambda x (\alpha y x y z \wedge \beta x y z)$. The same way $t(\langle 1, 1 \rangle, \alpha y, \lambda x y, z) = \langle 1, 1 \rangle, \alpha y, \lambda x y, z) = \lambda x (\alpha y x y z \wedge \beta x y z)$ proving that this choice in Cousot & Cousot [77a] was optimal.

End of Example.

Example 7.2.0.6

Some program analyses (such as "recovering definitions", "available expressions", "live variables", ... Aho & Ullman [77]) are "history sensitive" because the approximate assertions which are useful at each program point $p$ characterize sets of states or execution paths of the program point $p$ and not sets of states. In such a case Hoare [78] in his formal definition of languages using sets of sequential traces is more convenient that the deductive 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 $L_0$ of elementary assignments, a set $L_0$ of elementary tests, the set of sequential traces is the free monoid $T_1, \cdot_1$ generated by $L = L_0 \cup L_1$.

The concatenation operation $\cdot_1$ is extended to elements of the complete lattice $\mathcal{L}(\mathcal{S}, \wedge, \vee, \cdot_1)$ by $S \cdot_1 T = \{s \cdot_1 t : s \in S, t \in T \}$.

Let us define a forward "set of trace transformer" $F \in (\mathcal{L}(\mathcal{S}, \wedge, \vee, \cdot_1))$ by $F = \{(T_1, \cdot_1, T_1, s) \}$. The set of traces associated with a program $P$ and an entry

8. HIERARCHY OF PROGRAM ANALYSIS FRAMEWORKS
9. MERGE OVER ALL PATHS VERSUS LEAST FIXEDPOINT GLOBAL ANALYSIS OF PROGRAMS

9.1 "Distributive" Program Analysis Frameworks

We recalled at paragraph 4 that once a program analysis framework \((A, t, y)\) has been designed, the program-wide analysis problem has various solutions including the merge over all paths and least fixpoint solutions. It is known (Kam [11, 1977]) that when \(A\) satisfies the ascending chain condition and \(\forall x \in C, t(x) = 0\) we have \(\mathbb{M}P(t, \phi) \subseteq I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\). Also the additional hypothesis that \(\forall \phi \in C, t(S)\) is a join-morphism (sometimes called join-distributive map) implies \(\mathbb{M}P(t, \phi) = I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\). Slightly more general is the following:

**Theorem 9.1.0.1**

If \(\forall \in C, t(S) = 0\) for all programs \(\forall \in C, \mathbb{M}P(t, \phi) \subseteq I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\). If moreover \(\forall \in C, t(S)\) is a complete \(\mathbb{M}P(t, \phi) = I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\).

This theorem is implicitly used at paragraph 3 taking \(A = (V, \mathcal{L}) = (\forall, \lambda x. \text{false}, \lambda x. \text{true}, \lambda x. y, \lambda x. y)\) for \(A, \epsilon, \lambda \in C\) and either \(\epsilon \) or \(\forall \) for \(t\).

If \(A = (V, \mathcal{L}) = (\forall, \lambda x. \text{false}, \lambda x. \text{true}, \lambda x. y, \lambda x. y)\) then the above theorem establishes the correctness of \(I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\) with respect to \(\mathbb{M}P(t, \phi)\). In the literature the correctness of \(\mathbb{M}P(t, \phi)\) is generally taken for granted. Also \(\mathbb{M}P(t, \phi)\) is considered as the desired solution to program-wide analysis problems since whenever some \(t(S)\) is not a complete join-morphism \(\mathbb{M}P(t, \phi)\) can be strictly better than \(I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\). When \(A\) satisfies the ascending chain condition \(I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\) is computable, which is not necessarily the case of \(\mathbb{M}P(t, \phi)\). In that case a variety of methods can be used (e.g., Rosen [76]) 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 \(y\).

In our opinion the above argument is not entirely convincing since for different correct approximate predicate transformers \(t_1, t_2 \in (L \to (A \to A))\) it may be the case that \(I_{\mathbb{P}}(F_{\mathbb{P}}(t_1, \phi)) \subseteq I_{\mathbb{P}}(F_{\mathbb{P}}(t_2, \phi))\). 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 \(\tau_c(\epsilon = (\overline{A} \to \overline{A})\) be the best correct upper approximation of \(\tau_c(\epsilon = (\overline{A} \to \overline{A})\) in \(\overline{A} = \tau(A)\). If \(\tau(A)\) is a complete sublattice of \(\overline{A}\) then \(\mathbb{M}P(t, \phi) = I_{\mathbb{P}}(F_{\mathbb{P}}(t, \phi))\).
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 assembler 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 \( (A_1, t_1, \gamma_1) \) and \( (A_2, t_2, \gamma_2) \) be two program analysis frameworks such that \( A_1 \leq \gamma_1 \leq A_2 \) and \( t_1 \geq t_2 \) are correct upper approximations of \( t \) in \( A_1 \). The direct product \( (A, t, \gamma) \) of \( (A_1, t_1, \gamma_1) \) and \( (A_2, t_2, \gamma_2) \) is defined as \( A = A_1 \times A_2 \), \( t = t_1 \cup t_2 \), \( \gamma = \gamma_1 \cup \gamma_2 \).

(1) \( \forall \psi \in A_1, \psi \phi \in A_2, \text{MDP}(t, \phi, \psi) \) is a product of \( \text{MDP}(t_1, \phi, \psi_1), \text{MDP}(t_2, \phi, \psi_2) \),

(2) If moreover \( t_1 \) and \( t_2 \) are isomorphic, then \( \text{MDP}(t_1, \phi, \psi) = \text{MDP}(t_2, \phi, \psi) \).

This definition of direct product is not satisfactory since \( \gamma \) 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 9.2.0.3 and 9.2.0.4.

THEOREM 10.1,0.2

Let \( A_1 \leq A_2 \leq A_3 \) and \( \gamma_1 \leq \gamma_2 \leq \gamma_3 \) be two program analysis frameworks such that \( A_1 \leq A_2 \leq A_3 \) and \( \gamma_1 \leq \gamma_2 \leq \gamma_3 \). The reduced product \( (A, t, \gamma) \) is defined as \( A = A_1 \times A_2 \times A_3 \), \( t = t_1 \cup t_2 \cup t_3 \), \( \gamma = \gamma_1 \cup \gamma_2 \cup \gamma_3 \).

The reduced product \( (A, t, \gamma) \) is a refinement of the direct product \( (A, t, \gamma) \) which can be constructed as the infimum of the closure operators \( \gamma_1 \) and \( \gamma_2 \) (Theorem 8.0.1) viz. the join \( \{1 \cup \{p \in A_1 \cup A_2 \} \} \).

Since \( \gamma \) is a representation of the space of abstract states corresponding to the meet of the closure operators \( \gamma_1 \) and \( \gamma_2 \) (Theorem 8.0.1) viz. the join \( \{1 \cup \{p \in A_1 \cup A_2 \} \} \).

280
Example 10.1.0.3

\[ Y_1(1) = \lambda x. false, \quad Y_1(0) = \lambda x. (x=0), \quad Y_1(\top) = \lambda x. (x \geq 0), \quad Y_1(\bot) = \lambda x. (x < 0), \quad Y_2(1) = \lambda x. (x \equiv 0, \mod 2), \quad Y_2(0) = \lambda x. (x \equiv 1, \mod 2) \]

A disjunctive sum \( L_1 \cup L_2 \cup \{ I, T \} \) with ordering \( x \leq y \iff (x = 1) \lor (y = T) \) or \((x \equiv y, \mod 2 ) \). The meaning of elements of \( L_1 \cup L_2 \) can be defined as \( Y_1(1) = Y_1(\top), Y_2 = Y_2(0) = Y_2(\bot) \). True may be either one-to-one or one meets-morphism. In order to satisfy assumption 5.1.0.2 the set \( Y(1) = Y_1(1) \cup Y_2(1) \). Therefore the use of disjunctive sum amounts to the use of reduced products.

End of Remark.

10.2 Reduced Cardinal Power of Program Analysis Frameworks

The cardinal power \( L_1^2 \) with base \( L_2[\{ E_2, T_2, T \}, \{ \bot, \top, I \}) \) and exponent \( L_1[\{ 1, 2, T \}, \{ \bot, \top, I \}) \) is the set of all isotope maps from \( L_1 \) to \( L_2 \) with \( f \in L_2 \). Two program analysis frameworks \( (A_1, E_1) \) and \( (A_2, E_2) \) can be combined by letting \( g \in A_2\{ \{ x \} \} \) mean that for all \( x \in A_1 \), \( A_2(g(x)) \) holds whenever \( Y(x) \) holds.

THEOREM 10.2.0.1

The reduced cardinal power with base \( (A_2, E_2, Y_2) \) and exponent \( (A_1, T_1, Y_1) \) is \( (A_2, Y_2) \) where:

- \( \lambda_{A}(\{ x \}) = A(x) \). For all \( x \in A_1 \), \( Y_1(x) \) and \( Y_2(\{ x \}) \) is
- \( \lambda_{A}\{ \{ x \} \} = \lambda_{A}(A(x)) \) is
- \( \lambda_{A}\{ \{ x \} \} = \lambda_{A}(A(x)) \) is
- \( \lambda_{A}\{ \{ x \} \} = \lambda_{A}(A(x)) \) is
- \( \lambda_{A}\{ \{ x \} \} = \lambda_{A}(A(x)) \) is
- \( \lambda_{A}\{ \{ x \} \} = \lambda_{A}(A(x)) \) is
- \( \lambda_{A}\{ \{ x \} \} = \lambda_{A}(A(x)) \) is

Example 10.2.0.2

\[ Y_1(1) = Y_2(1) = \lambda_{x}(b, x). (x \equiv 0, \mod 2), \quad Y_1(\top) = Y_2(\top) = \lambda_{x}(b, x). (x \equiv 0, \mod 2), \quad Y_1(\bot) = Y_2(\bot) = \lambda_{x}(b, x). (x \equiv 0, \mod 2) \]

The analysis of the program:

\[ x = 100; \quad b := true; \]
\[ \text{while} \ b \ do \ x := \ x - 1; \]
\[ \text{if} \ b \ then \ x := \ (x \times 0); \]
\[ \text{if} \ b \ then \ od; \]

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 10.2.1.1 we determine that if $g(x)A_1 \rightarrow A_2$ then $\gamma(g) = (y_1(t) \wedge y_2(g(t))) \vee (y_1(t) \wedge y_2(g(f)))$. Therefore $\delta(h) = \delta(h_1) \wedge h(t) = (g(t), h(f)) = (g(t), h(t) \wedge g(f))$. It follows that $\alpha'(\delta \circ \alpha_1 \alpha_2)$ is isomorphic to $\langle t(f), f \rangle \rightarrow A_3$ for $A_2 \times A_2$.

The system of equations associated with the above program and the entry specification $\lambda_2.2$ is then:

$$
\begin{align*}
\lambda_1 & = \lambda(b, f \cdot b \cdot t \cdot \text{then} \cdot + \text{else} \cdot \lambda_2 \cdot f) \\
\lambda_2 & = \lambda(b, f \cdot b \cdot t \cdot \text{then} \cdot \lambda_1 \cdot \lambda_2 \cdot \text{else} \cdot \lambda_2 \cdot f) \\
\lambda_3 & = \lambda(b, \text{dear}(\lambda_2(b))) \\
\lambda_4 & = \lambda(b, f \cdot b \cdot t \cdot \text{then} \cdot (\lambda_3(\lambda_2) \vee \lambda_3(\lambda_2)) \vee \lambda_3(\lambda_2) \vee \lambda_4(\lambda_2)) \vee \lambda_2(f) \vee \lambda_2(f) \vee \lambda_2(f) \\
\lambda_5 & = \lambda(b, f \cdot b \cdot f \cdot \text{then} \cdot \lambda_3(\lambda_2) \vee \lambda_3(\lambda_2) \vee \lambda_4(\lambda_2) \vee \lambda_4(\lambda_2) \vee \lambda_2(f) \vee \lambda_2(f) \vee \lambda_2(f) \\
\end{align*}
$$

where $\text{dear}(\lambda_2) = \lambda_2$, $\text{dear}(\lambda_1) = \text{dear}(\lambda_2) = \text{dear}(\lambda_3) = \text{dear}(\lambda_4) = \lambda_2$.

The iterative resolution of this system of equations starting from the initial $\lambda_2.2$ yields $\gamma(\lambda_1) = Y(\lambda_2) = \lambda(b, x) \cdot (b \wedge x = 0)$, $Y(\lambda_2) = \lambda(b, x) \cdot (b \wedge x = 0)$, $Y(\lambda_3) = \lambda(b, x) \cdot (b \wedge x = 0)$.

End of Example.

11. REFERENCES


Cousett P. & Halbwachs N. [1978], Automatic discovery


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

Ore O. [1944], Galois connections, Trans. AMS, 55 (1944), 493-513.


Scott D. [1976], Data types as lattices, Siam Comp. 5, 3 (Sept. 1976), 522-567.