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[78], Wegbreit[75]) have introduced abstract approaches to program analysis which are tantamount to the use of a program analysis framework \((A, \text{trans})\) where \(A\) is a lattice of (approximate) assertions, \(\text{trans}\) is an (approximate) predicate transformer and \(\text{trans}\) 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 theorem 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 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 lattice 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

---

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

The set \(L\) of elementary commands consists in elementary tests and elementary assignments:

\[ l = l_1 \cup l_2 \cup \ldots \]

An elementary test \( q \in L \) is a total map from \( dom(q) \subseteq V \) into \( B\{true, false\} \). An elementary assignment \( e \in L \) is a total map from \( dom(e) \subseteq V \) into \( V \).

A program graph \( G \) is a tuple \((n, E, n_0, n')\) where \( n \) is the number of vertices (therefore \( n' > 1 \)), \( E \subseteq [1,n] \) is the (non-empty) set of edges, \( n_0 \in [1,n] \) is the entry point, \( n_1 \in [1,n] \) is the exit point and \( C \subseteq (E \rightarrow L) \) defines the command \( C(l, j) \) associated with each \( l:j \) in \( E \). Let \( pred(e) = 2^{[1,n]} \) be \( \lambda j.\{l \in [1,n] : l(j) \in E\} \) and \( succ(e) = 2^{[1,n]} \) be \( \lambda j.\{l \in [1,n] : j \in e(l)\} \), then we assume that \( pred(e) = P \), \( succ(e) = S \) for any \( e \in [1,n] \).

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 \( V \) into \( E \). The set \( A = (V \rightarrow B) \cup \lambda x \in V.x \) defines the set \( A \subseteq (V \rightarrow B) \cup \lambda x \in V.true \) of assertions is a complete boolean lattice partially ordered by the implication \( \rightarrow \).

Let \( sp(S)(P) \) be Floyd's strongest post-condition derived from the pre-condition \( P' \) for the elementary command \( S \). We assume that the operational semantics of the elementary commands is such that for an elementary test we have:

\[ sp(\neg e)(\lambda x = e(x) \rightarrow \neg e(x)) \]

whereas for an elementary assignment \( e \) we have:

\[ sp(e)(\lambda x = e(x) \rightarrow e(x)) \]

Notice that for all \( S \), \( sp(S) \) is a complete join-semilattice \( \{ 0 \leq S \leq S \} \).

We assume that the operational semantics of the program \( \pi \) is such that at each program point \( \eta \) the invariant assertion \( P_1 \) characterizes the set of states which are the descendants of the input states satisfying a given entry assertion \( P_0 \) which is the merge over all paths analysis of \( \pi \) using \( sp \) and \( \Phi \).

\( P_0 \) is the least fixpoint \( \nu P_0 = \nu P_0 \) of the system of equations \( P = P_0 \) associated with the program \( \pi \) using \( sp \) and \( \Phi \).
Example 3.1.0.1

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

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

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

\[
\begin{align*}
P_1 & = \lambda x.(x = 1) \\
P_2 & = \lambda x.(1 \leq x \leq 100) \\
P_3 & = \lambda x.(25 \leq x \leq 101) \\
P_4 & = \lambda x.(x > 101)
\end{align*}
\]

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 ancestors 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 formalism is necessary in order to treat backward program analysis. Instead of Floyd's forward predicate transformer we just have to consider Hoare's \(\uparrow p\)'s backward predicate transformer:

\[
\begin{align*}
\uparrow p[q] & = \lambda x. (x \in A, [x \in V, (P(X) \land \exists \theta \in \Theta(q) \land q(x))] \\
\uparrow p[a] & = \lambda x. (x \in A, [x \in V, (\exists \theta \in \Theta(a) \land P(e(x))] \\
\end{align*}
\]

(notice that \(\Theta, \uparrow p\) is a complete join and meet morphism) and the inverted program graph \(G' = \{n, e' \in \Theta, n', \Theta, e' \}\)

Example 3.2.0.1

The inverted program graph corresponding to 2.0.1 is

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

The corresponding system of backward semantic equations is:

\[
\begin{align*}
P_1 & = \uparrow p(\lambda x.(x \leq 100)) (P_2) \lor \uparrow p(\lambda x.(x > 100)) (P_4) \\
P_2 & = \uparrow p(\lambda x.(x = 1)) (P_3) \\
P_3 & = \uparrow p(\lambda x.(x \leq 100)) (P_2) \lor \uparrow p(\lambda x.(x > 100)) (P_4) \\
P_4 & = \phi
\end{align*}
\]

The merge over all paths and least fixpoint characterizations of the ancestors 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 automatized 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 mechanical analysis of programs: data type determination and data flow analysis (the dual one might be useful [e.g., for proving termination]).

The new 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 \(A = \{l, \lor, l, \land\}\); the meaning of the elements of \(A\) is specified by a (too often implicit) order morphism \(Y\) mapping \(A\) to a subset of assertions \(A = Y(A) \subseteq A\). The intention is that \(A\) is an implementable image of those aspects \(Y(A)\) of the program properties which are to be understood at each program point whereas the assertions belonging to \(A \setminus Y(A)\) are ignored (that is approximated from above in \(Y(A)\)). To each elementary command \(S\) is associated an isomorphism \(t(S)\) from \(A\) to \(A\). The intent is that \(t(S)\) is an approximate predicate transformer such that \(t(S)(\uparrow p)\) represents the propagation of the information \(\{a\} \in A\) through the statement \(S\).

The ideal merge over all paths program-wide analysis [Graham & Wegbreit'76, Knuth & Ulman'77, Rosen'76, Tarjan'76] is often approximated by a fixpoint solution [Cousot & Cousot'77a, Jonas & Muchnick'78, Kaplan & Ulman'78, Kildas'79, Tenenbaum'74]. A fixpoint system of isomorphism equations \(X = F(X)\) where \(F = (A^0 \rightarrow A^0)\) is associated with the program graph. The approximate invariant assertions are generated by computing iteratively the least fixpoint of \(F\) starting from the initial of \(A\) 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 \( A \in A \) of assertions has been found to provide meaningful information.

Since any invariant assertion \( p \in A \) for any program must have an upper approximation \( Q \) in \( A \), the set \( \{ Q \in A : p \subseteq Q \} \) must be non-empty.

Let \( p \in A \) be an assertion and assume that we want to analyze a program using the merge over all paths semantic analysis and an entry condition \( Q \) which is an upper approximation of \( p \) in \( A \). What is the best choice for \( Q \)? It is clear that if \( p \subseteq Q \subseteq Q' \) then \( F_{p}(Q') \supseteq F_{p}(Q) \) and by isotony the analysis \( L[p](F_{p}(Q')) \) is more precise than \( L[p](F_{p}(Q)) \).

Hence \( Q \) must be a maximal upper approximation of \( p \) in \( A \) (that is such that \( \forall \ Q' \in A \ : \ p \subseteq Q' \subseteq Q \)). Assume that the set \( U \) of minimal upper approximations of \( p \) in \( A \) has a cardinality greater than 1. What is the "best" possible choice for \( Q \) in \( U \)? If \( Q_{1}, Q_{2} \in U \) and \( Q_{1} \neq Q_{2} \) then \( Q_{1} \) and \( Q_{2} \) are not necessarily comparable so that \( L[p](F_{p}(Q_{1})) \) and \( L[p](F_{p}(Q_{2})) \) may be not comparable. Hence "best"

cannot be defined using the preciseness criterion provided by the ordering \( \subseteq \). 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 \( A \) such that \( Card(U) = 1 \).

Example 5.1.0.1

Assume that \( A = \{ Z \} \) where \( Z \) is the set of integers and \( A = \{(x,y),(p(x),p(y)) : p \in P \subset \{ true \} \} \). The assertion \( p = \{ (x,y),(x \geq 0) \} \) has two distinct minimal upper approximations \( \bar{A} \) namely \( Q_{1} = (x,y),(x \geq 0) \) and \( Q_{2} = (x,y),(x > 0) \). 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., \( sp(x := x + y)(Q_{1}) = \lambda(x,y),(x \geq 0) \) and \( sp(x := x + y)(Q_{2}) = \lambda(x,y),(x > 0) \) and next comparing them). Since these analyses are not related by the ordering \( \subseteq \), the comparison criterion must be application dependent. For example using \( Q_{1} \) we can prove that \( sp((x,y),(x \geq 0)) = \lambda(x,y),(x \geq 0) \) whereas this is impossible with \( Q_{2} \). On the contrary the best choice is \( Q_{2} \) for the program \( x := x + y \) since \( sp((x,y),(x > 0)) = \lambda(x,y),(x > 0) \) whereas \( sp((x := x),(x < x + y),(x < y \land y < 0)) \) does not imply \( \lambda(x,y),(x \geq 0) \).

End of Example.

If any program must have an analysis which can be approximated from above using \( A \), 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 \( A \in A \) of approximate assertions must be chosen such that for all \( p \in A \) the set \( \{ Q \in A : p \subseteq Q \} \) of upper approximations of \( p \) in \( A \) is a least element.

THEOREM 5.1.0.3

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

5.2 The Approximation Operator

DEFINITION 5.2.0.1 Approximation Operator

\[ p \in A \rightarrow R \]

\[ p = \lambda p \cdot A(p \cdot A : p \subseteq Q) \]

0(p) is the least upper approximation of \( p \) in \( A \). Since \( A \) is a Moore family it follows from Monteiro & Reisheif [42, Th.5.3 and 5.1] that:

THEOREM 5.2.0.2

1) \( p \in A \rightarrow R \) is an upper closure operator (that is \( p \) is isotone (if \( p \subseteq Q \) then \( p \subseteq Q \)) and extensive (for all \( p \subseteq Q \), \( p \subseteq Q \)) and idempotent (\( p \subseteq p \)).

2) \( p(A) = A \)

3) \( p \) is the unique upper closure operator on \( A \) such that \( p(A) = A \).

Since \( A \) is equal to the image of the complete lattice \( A \rightarrow R(A, R) \) and \( A \rightarrow R(A, R) \) by the upper closure operator \( p \) we derive from Ward [42, Th.4.4] the following:

THEOREM 5.2.0.3

1) \( A \rightarrow R(A) = \lambda p \lambda x, y \cdot R(A, R), x, y \cdot R(A, R) \)

2) \( p \) is a quasi-complete join-morphism (i.e., \( p(V \sqcup A, p(V)) = p(V) \)).

3) \( A \rightarrow R(A) = \lambda p \lambda x, y \cdot R(A, R), x, y \cdot R(A, R) \)
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 \( \bar{A} \subseteq A \), the upper closure operator \( \rho \) on \( A \) such that \( \rho(A) \) is the least Moore family containing \( \bar{A} \) is:

\[
\rho = \lambda A. \lambda \{ x : (A \cup \{ \{x, \text{true}\}\}) \wedge \text{true} \}
\]

\( \rho(A) = \{ \{x, \text{true}\} : x \in A \} \wedge \text{true} \}

**Example 5.2.0.5**

Returning to example 5.1.0.1 where \( A = \{ \{a, \text{false}\}, \{b, \text{false}\}, \{c, \text{false}\}, \{d, \text{false}\}, \{e, \text{false}\}\} \), the least Moore family containing \( \bar{A} \) is the one containing \( \text{true} \), \( \bar{A} \) and the meet of the non-empty subsets of \( \bar{A} \) that is the complete lattice:

\[
\begin{array}{c}
\text{true} \\
\text{false}
\end{array}
\]

The corresponding approximation operator is:

\[
\rho = \lambda A. \lambda \{ x : \text{true} \wedge \text{false} \}
\]

**THEOREM 5.3.0.3**

**DEFINITION 5.3.0.3**

Let \( L_1(E_1) \) and \( L_2(E_2) \) be posets, \( \alpha \in (L_1 \rightarrow L_2) \), \( \gamma \in (L_2 \rightarrow L_1) \). The pair \( \langle \alpha, \gamma \rangle \) defines a Galois connection between \( L_1 \) and \( L_2 \) if and only if:

1. \( \alpha \) is monotone \( \{ x_1, x_2 \in E_1 \rightarrow \{ x_1 \subseteq E_2 \alpha(x_2) \} \}
2. \( \gamma \) is monotone \( \{ y_1, y_2 \in E_2 \rightarrow \{ y_1 \subseteq E_1 \gamma(y_2) \} \}
3. \( \alpha \) is onto \( \{ x_1 \subseteq E_1 \gamma(x_2) \}
4. \( \gamma \) is into \( \{ y_1 \subseteq E_2 \alpha(y_2) \}

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

\[
\alpha(x_1) \subseteq E_2 \alpha(y_1) \quad \gamma(y_1) \subseteq E_1 \gamma(x_2)
\]

**THEOREM 5.3.0.4**

Let \( L_1(E_1) \) and \( L_2(E_2) \) be posets, \( \alpha \in (L_1 \rightarrow L_2) \), \( \gamma \in (L_2 \rightarrow L_1) \). \( \langle \alpha, \gamma \rangle \) is a pair of adjoint functions if and only if \( \langle \alpha, \gamma \rangle \) defines a Galois connection between \( L_1(E_1) \) and \( L_2(E_2) \), i.e. if \( \alpha \) and \( \gamma \) are isomorphic, \( \lambda \lambda x \gamma(x, y) \alpha(x, y) \). 

**THEOREM 5.3.0.5**

**COROLLARY 5.3.0.5**

Let \( L_1(E_1) \) and \( L_2(E_2) \) be posets and \( \alpha \in (L_1 \rightarrow L_2) \), \( \gamma \in (L_2 \rightarrow L_1) \). \( \alpha \) is an upper closure operator on \( L_1 \), \( \gamma \) is a lower closure operator on \( L_2 \). \( \alpha \) and \( \gamma \) are adjoint functions if

\[
\begin{align*}
&1. \gamma \alpha \text{ is an upper closure operator on } L_1, \\
&2. \alpha \gamma \text{ is a lower closure operator on } L_2, \\
&3. \alpha \gamma \text{ is isomorphic to } \gamma \alpha \\
&4. \alpha \text{ is a complete join-morphism, } \alpha((L_1 \wedge L_2) = L_2, \alpha \text{ is a complete meet-morphism, } \gamma((L_2 \wedge L_1) = L_1)
\end{align*}
\]

In complement we will need the following:

**THEOREM 5.3.0.6**

Let \( L_1(E_1, T_1, \lambda_1, \Gamma_1) \) and \( L_2(E_2, T_2, \lambda_2, \Gamma_2) \) be complete lattices and \( \alpha \in (L_1 \rightarrow L_2) \), \( \gamma \in (L_2 \rightarrow L_1) \). \( \alpha \) and \( \gamma \) are adjoint functions if

\[
\begin{align*}
&1. \alpha \text{ is onto (surjective) if and only if } \gamma \text{ is one-to-one (injective) if and only if } \\
&\lambda \gamma \text{ is onto (surjective) if and only if } \alpha \text{ is one-to-one (injective) if and only if }
\end{align*}
\]

(2) if one of the above conditions holds then
We use the notation \( L_1 \sqsupset a, y \gtrless L_2 \) to state that \( L_1 \) and \( L_2 \) are connected by the pair \( a, y \) of adjoint functions which are respectively surjective and injective. If \( a \) is a complete join-morphism from \( L_1 \) onto \( L_2 \) (respectively \( y \) is a one-to-one complete meet-morphism from \( L_2 \) into \( L_1 \)) we write \( L_1 \sqsupset a, y \gtrless L_2 \) and assume that the adjoint \( y(a) \) 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 \( A(E, 1, T, L, \Pi) \) representing approximate assertions and to informally describe the meaning of its elements (e.g. constant propagation, Kildall [73], Kam & Ullman [77]). Hence the function \( y \in (A \to A) \) remains implicit.

It is often the case that \( y \) is only assumed to be a complete join-semi-lattice \( A(E, 1, T, L, \Pi) \) (or dually meet-semi-lattice for some authors) but since an infimum is adjoint to \( A \) it is in fact a complete lattice (even when the meet-operation is not used or what is called meet is not \( \Pi \) [e.g. Wegbreit [75]]).

When \( y \in (A \to A) \) is isotonous but not a complete meet-morphism 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 \( y \in (A \to A) \) 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 \( y \) can be revised as follows:

**Theorem 5.3.0.7**

Let \( A(E, 1, T, L, \Pi) \) be a complete lattice and \( y \in (A \to A) \) be a complete meet-morphism. Let \( \sigma \in (A \to A) \) be \( \lambda x \Pi \cap \{y \circ x \colon y(x) = y(\sigma(x))\} = \sigma(A), \) \( y = \{y \} \): 

- \( \forall x \in A, y(x) = y(\sigma(x)) \)
- \( \sigma \) is a closer closure operator on \( A \)
- \( y \) is a one-to-one complete meet-morphism from the complete lattice \( A(E, 1, T, L, \Pi \sigma(A)(\Pi)) \) into \( A \)

Since \( y(A) = y(\sigma(A)), \) \( A \) and \( \sigma(A) \) have the same expressive power. Among all subsets of \( A, \sigma(A) \) is one with minimal cardinality.

**Theorem 5.3.0.8**

1. \( \forall L \subseteq A, \{y(L) = y(a) \} \supset \{a(L) = A\} \)
2. \( \forall L \subseteq A, \{y(L) = y(\Pi) \} \supset \{\text{Card}(A) = \text{Card}(L)\} \)
3. \( \forall x \in A, \forall y \in A, \{y(x) = y(y)\} \supset \{x \subseteq y\} \)

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 theorems 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 \( L(E, 1, T, L, \Pi) \) be a complete lattice and \( f \subseteq (L \to L) \). Let \( \exists y \in \{f(L) \to (L \to L)\} \) be \( \lambda x \lambda y : f(y) \subseteq y \to x \} \) isoc(f) is the least isotonous operator on \( L \) greater than or equal to \( f \).

- Let \( \exists x \in \{f(L) \to (L \to L)\} \) be \( \lambda y : f(y) \to f(x) \} \) ext(f) is the least extensive operator on \( L \) greater than or equal to \( f \).

Let \( \exists d \in \{f(L) \to (L \to L)\} \) be \( \lambda x : f(x) \to f(x) \} \) where \( f(x) \) is the limit of the increasing and ultimately constant sequence \( \{x^d\} \) such that for all ordinals \( \delta \), \( x^\delta = f(x^\delta) \) for all \( \alpha < \delta \) and \( x^\delta = \{x^\alpha \} \). Hence the closure operator greater than or equal to \( f \) and \( d \circ f \) is the greatest Moore family contained in \( f(L) \)

### 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 [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 \( L(E, 1, T, L, \Pi) \) be a complete lattice, \( p \) an upper closure operator on \( L \) and \( \Pi \) an upper closure operator on \( P(L) \). Then \( \Pi \circ p \) is an upper closure operator on \( L \) and \( p \circ \Pi \).

**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 \( V = D^m \), let \( \Pi_m \) be \(+B\) and \( A_1 \) be \( B \to B \).

Let us define:

\[
\sigma_j = \forall x \in [1, m], \begin{cases} 
\lambda \Pi_j, [x \in V, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_m]>
\end{cases} \epsilon D^{m-1} = \{v_1, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_m\}
\]

\[
\sigma = \forall x \in [1, m], \begin{cases} 
\lambda \Pi_j, [x_1, \ldots, x_m] \in D^m, [A \in \sigma(P(x_j))] \end{cases}
\]

274
σ is an upper closure operator on \( A_m \) and an assertion \( \mathcal{P} \in \mathcal{A}_m \) does not state relationships among the program variables if and only if \( \mathcal{P} \in \mathcal{I} \). The approximation 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 \( \mathcal{A} \) is defined by \( \rho(\mathcal{P}) = \bigvee_{j=1}^m \rho_j^P(x_i) \) for \( \mathcal{P} \in \mathcal{A} \). It follows from theorem 6.2.0.1 that the composition:

\[
\rho \sigma = \lambda \rho \mathcal{P} \mathcal{A}_m \{\bigvee_{j=1}^m \rho_j^P(\sigma_j^P(x_i))\}
\]

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 \( (\rho, \ ltd \ ) \) induced by an upper closure operator \( \rho \) on \( A \) and defined as \( A \equiv \mathcal{P} \) if and only if \( \mathcal{P} \in \mathcal{I} \), the approximation process 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 a least one must exist (assumption 5.1.0.2), not all equivalence relations are acceptable.

DEFINITION 6.3.0.1

Let \( L(E, \ I, T, U, I) \) 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:
   \[
   \forall x, y, u, v \in L : x \uplus y \equiv u \implies \theta(x \uplus y) \equiv \theta(y) \]
3. \( \theta \) satisfies the join-complementarity property:
   \[
   \forall x \uplus y \in L : (x \uplus y) \uplus u \equiv v \implies \theta(x \uplus y) \equiv \theta(u) \equiv \theta(v)
   \]

THEOREM 6.3.0.2

If \( \rho \) is an upper closure operator on \( L(E, I, T, U, I) \) and \( \forall x, y \in L, x \equiv y(\rho) \), then:

1. \( 1 \) \( - \rho \) is a complete join congruence relation on \( L \)
2. \( 2 \) \( \rho = \lambda x \uplus I \{(x\uplus y)\} \)

Recursively, 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 \( L(E, I, T, U, I) \). \( \lambda x \uplus I \{x\theta\} \) is an upper closure operator on \( L \).

(Similar results were already proved in Cousot & Cousot[77b] except that the above definition of complete join congruence relations has been substantially simplified.)

The following result can sometimes facilitate the proof that a given relation is a join congruence relation (satisfying 6.3.0.1.1) and 6.3.0.1.2). It can be compared with Grätzer & Schmidt[58b]’s theorem which is relative to congruence relations.

THEOREM 6.3.0.4

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

1. \( 1 \) \( [x, y \in \theta(\theta(\theta))] \implies \forall u, v \in \theta(\theta(\theta)) \)
2. \( 2 \) \( x \equiv y(\theta) \equiv \theta(\theta(\theta)) \)
3. \( 3 \) \( x \equiv y(\theta) \equiv \theta(\theta(\theta)) \)

Example 6.3.0.5

Let \( U \) be a non-empty set of integers included between two bounds \( a \) and \( b \) (either \( \in \mathbb{Z} \)), \( x \in \mathbb{Z} \) and \( y \in \mathbb{Z} \). The binary relation \( \theta \) defined on \( A \) by:

\[
\theta \equiv \{x, y \in \mathbb{Z} \mid x \equiv y(\theta) \}
\]

is isomorphic to \( \rho(\mathcal{P}) \) for \( \rho \) an upper closure operator induced by \( \theta \):

\[
\rho = \lambda x \uplus I \{x \theta\}
\]

In conjunction with 6.2.0.2, \( \rho \) can be used for static analysis of the ranges of values of numerical variables (Cousot & Cousot[77a]).

End of Example.

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 \( \rho \) the following property:

THEOREM 6.4.0.1

Let \( \theta \) be a complete join congruence relation on the complete lattice \( L(E, I, T, U, I) \), then \( \forall x, y \in \theta(\theta) \) is a complete and convex subjoin-similallattic of \( L \).

End of Example.

Here is another representation of a convex subjoin-similallattic of \( L \) (which can be compared with Grätzer[74b]’s representation of convex sublattices):

An ideal is a nonvoid subset \( J \) of a lattice \( L(E, I, T, U, I) \) with the properties

(a) \( \{a \in J \mid a \leq a \} \subseteq \{x \in J \mid x \leq a \} \}

(b) \( \{a \in J \mid a \leq a \} \subseteq \{x \in J \mid x \leq a \} \}

It is easy to show that \( J \) is an ideal when \( \{a \in J \mid a \leq a \} \subseteq \{x \in J \mid x \leq a \} \}

Given an element \( a \) in a lattice \( L \), the set \( \{a \in J \mid a \leq a \} \}

is evidently an ideal; it is called a principal ideal of \( L \). If every ascending chain \( I \) in \( L \) is finite, every ideal is principal.

A semi-ideal is a nonvoid subset \( I \) of \( L \) with the property

\[
\{a \in J \mid a \leq a \} \subseteq \{x \in J \mid x \leq a \}
\]

The dual notion is the one of dual semi-ideal.
7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER
INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

In addition to A and y the specification of a program analysis framework also includes the choice of an approximate predicate transformer \( tc(L \rightarrow (A \rightarrow A)) \) (or a monoid of maps on A plus a rule for associating maps to program statements (e.g., Rosen (76))). We now show that in fact this is not indispensable since there exists a best correct choice of \( T \) 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 \( (V,A,\tau) \) the minimal assertion which is invariant at point 1 of a program \( \pi \) with entry specification \( \psi \) was defined as:

\[
P_1 = \bigvee_{p \in \text{Path}(1)} \psi(p)(\delta)
\]

Therefore the minimal approximate invariant assertion is the least upper approximation of \( P_1 \) in \( A \) that is:

\[
\rho(P_1) = \rho(\bigvee_{p \in \text{Path}(1)} \psi(p)(\delta))
\]

Even when \( \text{Path}(1) \) is a finite set of finite paths the evaluation of \( \psi(p)(\delta) \) is hardly machine-implementable since for each path \( p = p_1, \ldots, p_m \) the computation sequence \( X_0 = \phi, X_1 = \tau(C(p_1))(X_0), \ldots, X_m = \tau(C(p_m))(X_{m-1}) \) does not necessarily only involve elements of \( A \) and \( \tau(A,A) \). Therefore using \( \tau(A,A) \) and \( C : (L \rightarrow (A \times A)) \) a machine representable sequence \( X_0 = \phi, X_1 = \tau(C(p_1))(X_0), \ldots, X_m = \tau(C(p_m))(X_{m-1}) \) is used instead of \( X_0, \ldots, X_m \) which leads to the expression:

\[
\rho_1 = \rho(\bigvee_{p \in \text{Path}(1)} \tau(p)(\delta))
\]

The choice of \( \tau \) and \( \phi \) is correct if and only if \( \rho_1 \) is an upper approximation of \( P_1 \) in \( A \) that is if and only if:

\[
\bigvee_{p \in \text{Path}(1)} \tau(p)(\delta) \Rightarrow \rho(\bigvee_{p \in \text{Path}(1)} \tau(p)(\delta))
\]

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

**DEFINITION 7.1.0.1**

- An approximate predicate transformer \( tc(L \rightarrow (A \rightarrow A)) \) is said to be a correct upper approximation of \( tc(L \rightarrow (A \rightarrow A)) \) in \( A \) if and only if for all \( \phi, \gamma \) such that \( \phi \Rightarrow \gamma \) and program \( \pi \) we have:
  \[ \text{MOP}_{\pi}(\tau, \phi) \Rightarrow \text{MOP}_{\pi}(\tau, \gamma) \]

- Similarly if \( A \supseteq \alpha, \gamma \supseteq A \), \( tc(L \rightarrow (A \rightarrow A)) \) is said to be a correct upper approximation of \( tc(L \rightarrow (A \rightarrow A)) \) in \( A \) if and only if for all \( \phi, \psi \) such that:
  \[ \phi \Rightarrow \gamma(\psi), \forall \psi, (\text{MOP}_{\pi}(\tau, \phi)) \Rightarrow \text{MOP}_{\pi}(\tau, \gamma(\psi)) \]

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

5.2.0.5

End of Example.
Theorem 7.1.0.2

(1) \( -\bar{T}(L + (A^* \rightarrow A)) \) is a correct upper approximation of \( \tau \in L + (A^* \rightarrow A) \) in \( A^* \rightarrow A \) \iff \( \{w \in L, w^* \in A^*, \tau(S)(P) \Rightarrow \bar{T}(S)(P)\} \).

(2) \( -\bar{T}(L + (A^* \rightarrow A)) \) is a correct upper approximation of \( \tau \in L + (A^* \rightarrow A) \) in \( A^* \rightarrow A \) \( \) where \( A \leftarrow A^* \rightarrow A \) \( \) \iff \( \{w \in L, w^* \in A^*, \alpha(\tau(S)(P) \Rightarrow \bar{T}(S)(P)) \} \).

If \( -\bar{T}(L + (A^* \rightarrow A)) \) is a correct upper approximation of \( \tau \in L + (A^* \rightarrow A) \) in \( A^* \rightarrow A \) we have \( \text{MOP}_\tau(-\bar{T}(A^*)) \Rightarrow \text{MOP}_\tau(-\bar{T}(A^*)) \).

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 \( T \) is a correct upper approximation of \( \tau \in L + (A^* \rightarrow A) \) in \( A^* \rightarrow A \) then \( \forall P \subseteq L, \forall \alpha, \alpha(\tau(S)(P) \Rightarrow \bar{T}(S)(P)) \).

(2) If \( T \) is a correct upper approximation of \( \tau \in L + (A^* \rightarrow A) \) in \( A^* \rightarrow A \) where \( A \leftarrow A^* \rightarrow A \) and \( \forall P \subseteq L, \alpha(\tau(S)(P) \Rightarrow \bar{T}(S)(P)) \).

Similar results hold for fixpoint analysis of programs.

Theorem 7.1.0.4

Let \( \tau \in L + (A^* \rightarrow A) \) be an isotonous correct upper approximation of \( \tau \in L + (A^* \rightarrow A) \) in \( A^* \rightarrow A \) where \( A \leftarrow A^* \rightarrow A \) then \( \{w \in L, w^* \in A^*, \forall P \subseteq L, \alpha(\tau(S)(P) \Rightarrow \bar{T}(S)(P)) \).

Notice that in Theorem 7.1.0.2 the maps \( \overline{\tau}(S) : S \subseteq L \) are not assumed to be isotonous. Yet isotonous is assumed in Theorem 7.4.0.4 and is a customary hypothesis in the literature. An apparent justification of this A-weakly-\( \tau \)-requirement is to ensure that the system of equations \( x = F_T(x, \overline{\tau}(X)) \) associated with a program \( \tau \) has fixpoints which can be obtained as limits of iteration sequences. But this could also be achieved without isotonous hypothesis taking \( \lambda x. x \sqsubseteq F_T(x, \overline{\tau}(X)) \) instead of \( F_T(x, \overline{\tau}(X)) \).

Example 7.2.0.5

Coming back to examples 8.2.0.2 and 6.3.0.5 assume that \( D \) is the subset of integers included between two bounds \( a \) and \( b \) and \( y = D \).

Since \( y \) is an injective complete meet-morphism the adjunction function \( \alpha^*(D = R) \vdash \alpha \) is determined by \( \alpha^*(D = R) \vdash \alpha \rightarrow \alpha \).
According to theorem 7.2.0.4 the best upper approximation of \( \alpha(\{x, y, z, (x, y)\}) \) in \( A \) is \( \alpha = \alpha(\{x, y, z\}) \setminus \{x, y\} = \{(x, z), (y, z), (x, y)\} \). If \( P \alpha \) equals \( <1, 1> \) then \( T \geq <1, 1> \) else \( P \leq \{x, y, z\} \) \& \( \alpha \). Therefore, the best upper approximation of \( \alpha(\{x, y, z, (x, y)\}) \) in \( A \) is \( \alpha = \alpha(\{x, y, z\}) \setminus \{x, y\} = \{(x, z), (y, z), (x, y)\} \). Hence, the best upper approximation of \( \alpha(\{x, y, z, (x, y)\}) \) in \( A \) is \( \alpha = \alpha(\{x, y, z\}) \setminus \{x, y\} = \{(x, z), (y, z), (x, y)\} \).

End of Example.

Example 7.2.0.6

Some program analyses (such as "teaching definitions", "available expressions", "live variables", ... Ahu & Ulman[77]) are "history sensitive" because the approximate assertions which are useful at each program point \( p \) characterize sets of sequences of states (or execution paths from the entry point to \( p \)) and not sets of states. In such a case Hoare[78] 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 universal \( V \) of values, a set \( L \) of elementary assignments, a set \( L \) of elementary tests, the set of sequential traces is the free monoid \( T(L, V, \alpha) \) generated by \( L = \langle L, V, \alpha \rangle \).

The concatenation operation "\( \cdot \)" is extended to elements of the complete lattice \( L = \{L, V, \alpha\} \) by \( \alpha(t) = \{t \in L \mid \alpha(t) \in \alpha(L) \} \).

Let us define a forward "set of traces transformer" \( \Phi(t) \) \( \in \{L = \{L, V, \alpha\} \} \) as \( \alpha(L, V, \alpha(L)) \) \( \Phi \). The set of traces associated with a program \( P \) and an entry specification \( \Phi(L) \in \Phi(t) \).

7.2.0.6.2. Approximating a Set of Traces by an Assertion Characterizing the Descendants of the Entry States

The connection with the deductive semantics of paragraph 3 is made using \( \alpha(\{L, V, \alpha(L)\}) \) such that for any set \( T \) of traces, \( \alpha(T) \) characterizes the possible descendants of the entry states (belonging to \( L \)) when the traces \( t \in T \) are executed. From an obvious reason not given here operationally semantics of sequential traces we derive the characteristic function \( \alpha(T) \) \( \alpha(T) = \{t \in L \mid \alpha(t) \in \alpha(T) \} \).

Since \( \alpha \) is a complete join-morphism from \( L \) onto \( A \), theorem 3.0.4.3.(2) defines an adjoint function \( \Psi(T) \) \( \{L, V, \alpha(L)\} \).

According to theorem 7.2.0.4 the best correct upper approximation of \( \Phi(t) \) \( \in \Phi(T) \) \( \{L, V, \alpha(L)\} \) is \( \alpha(\Phi(T)(\Psi(T))) = \alpha(\Phi(T)(\Psi(T))) \) \( \Phi(T) \).

Since \( \alpha \) is a complete join-morphism from \( L \) onto \( A \), theorem 3.0.4.3.(2) defines an adjoint function \( \Psi(T) \).

Theorem 7.2.0.4.3. Justifying the Data Flow Equations of "Available Expressions"

Let \( E \) be the set of expressions. The set \( \alpha(T) \) \{expressions which are available at exit of a path \( t \in T \) in \( T \) \} is defined by \( \alpha(T) = \alpha(T) \cap \alpha(T) \) \( \alpha(T) \) \( \alpha(T) \).

An expression \( E \) is available in some program point \( q \) if it is available at exit of every path from the entry point to \( q \). Therefore the set of expressions available at \( q \) is \( \alpha(T) \cap \alpha(T) \).

Since \( \alpha \) is a complete join-morphism from \( L \) \{expressions at \( q \)\} \( \{L, V, \alpha(L)\} \) \( \{L, V, \alpha(L)\} \).

According to theorem 7.2.0.4 the best correct upper approximation of \( T \) \( \{L, V, \alpha(L)\} \) is \( \alpha(\Phi(T)(\Psi(T))) = \alpha(\Phi(T)(\Psi(T))) \) \( \Phi(T) \).

Since \( \alpha \) is a complete join-morphism from \( L \) onto \( A \), theorem 3.0.4.3.(2) defines an adjoint function \( \Psi(T) \).

THEOREM 8.0.1

The set of upper closure operators on a complete lattice \( L = \{L, V, \alpha(L), \} \) is a complete lattice \( \alpha(\{L, V, \alpha(L)\}) \).

Example 8.0.2

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 \( y \) the characteristic set of which is the following:

\[ y = \frac{P(x, y)}{x} \]

\[ y = \frac{P(x, y)}{x} \]

\[ y = \frac{P(x, y)}{x} \]

\[ y = \frac{P(x, y)}{x} \]
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:

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

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:

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

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

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

The corresponding framework was used for the automatic discovery of linear restraints among variables of programs [Cousot & Halbwachs '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-wise analysis problem has various solutions including the merge over all paths and least fixpoint solutions. It is known (Kam & Lilliemel '77) that when \( A \) satisfies the ascending chain condition \( \mathcal{W} \mathcal{S}L \) \( t(S) \) is isotope we have \( \text{MOP}_P \{t, \phi\} \subseteq \mathcal{L}_P \{F_P(t, \phi)\} \). Also the additional hypothesis that \( \mathcal{W} \mathcal{S}L \) \( t(S) \) is a join-morphism (sometimes called join-distributive map) implies \( \text{MOP}_P \{t, \phi\} = \mathcal{L}_P \{F_P(t, \phi)\} \). Slightly more general is the following:

**THEOREM 9.1.0.1**

If \( A(\mathcal{W} \mathcal{S}L, t, \mathcal{W} \mathcal{L}) \) is a complete lattice and \( t(\mathcal{L} \{A + A\}) \) is such that \( \mathcal{W} \mathcal{S}L \) \( t(S) \) is isotope then for all programs \( \Psi \) and \( \phi \in A \), \( \text{MOP}_P \{t, \phi\} \subseteq \mathcal{L}_P \{F_P(t, \phi)\} \). If moreover \( \mathcal{W} \mathcal{S}L \) \( t(S) \) is a complete \( \mathcal{L} \mathcal{L} \)-morphism then \( \text{MOP}_P \{t, \phi\} = \mathcal{L}_P \{F_P(t, \phi)\} \).

This theorem is implicitly used at paragraph 3 taking \( A(\mathcal{W} \mathcal{L} \{L \rightarrow B\}) \rightarrow, \Lambda \mathcal{L} \{false \}, \Lambda \mathcal{L} \{true \}, \Lambda \mathcal{L} \{\Lambda \} \) for \( A(\mathcal{W} \mathcal{L}, t, \mathcal{L} \mathcal{L}) \) and either \( ap \) or \( wp \) for \( t \).

If \( A = (L \rightarrow B) \rightarrow, \Lambda \mathcal{L} \{false \}, \Lambda \mathcal{L} \{true \}, \Lambda \mathcal{L} \{\Lambda \} \) for \( A(\mathcal{W} \mathcal{L}, t, \mathcal{L} \mathcal{L}) \) and either \( ap \) or \( wp \) for \( t \).

If \( A = L \rightarrow B \rightarrow \Lambda \mathcal{L} \{false \}, \Lambda \mathcal{L} \{true \}, \Lambda \mathcal{L} \{\Lambda \} \) then the above theorem establishes the correctness of \( \mathcal{L}_P \{F_P(t, \phi)\} \) with respect to \( \text{MOP}_P \{t, \phi\} \). In the literature the correctness of \( \text{MOP}_P \{t, \phi\} \) is generally taken for granted. Also \( \text{MOP}_P \{t, \phi\} \) is considered as the desired solution to program-wide analysis problems since whenever some \( t(S) \) is not a complete join-morphism \( \text{MOP}_P \{t, \phi\} \) can be strictly better than \( \mathcal{L}_P \{F_P(t, \phi)\} \). When \( A \) satisfies the ascending chain condition \( \mathcal{L}_P \{F_P(t, \phi)\} \) is computable, which is not necessarily the case of \( \text{MOP}_P \{t, \phi\} \). In that case a variety of methods can be used (e.g. Rosen '78) 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 \( t_j \), \( t_k \in \mathcal{L} \{L \rightarrow (A + A)\} \) it may be the case that \( \mathcal{L}_P \{F_P(t_j, \phi)\} \subseteq \text{MOP}_P \{t_k, \phi\} \). 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 \( \mathcal{L}_P \{L \rightarrow (A \rightarrow A)\} \) be the best correct upper approximation of \( \mathcal{L}_P \{L \rightarrow (A \rightarrow A)\} \) in \( A \rightarrow \mu \{D\} \). If \( \mu \{D\} \) is a complete sublattice of \( A \) then \( \text{MOP}_P \{t, \phi\} = \mathcal{L}_P \{F_P(t, \phi)\} \).

**Example 9.1.0.3**

If \( A = (L + B) \) and \( \bar{\gamma} = \gamma(A) \) where:

\[
\bar{\gamma} = \begin{cases}
0 & \text{if } x, y \in A \text{ and } x \neq y \text{ or } (x, y) \in \mathcal{W} \mathcal{L} \{false \}, \\
1 & \text{otherwise}
\end{cases}
\]

and \( \gamma(L) = \lambda u, u \neq u \), \( \gamma(-) = \lambda u, u \neq u \), \( \gamma(0) = \lambda u, u = 0 \), etc.

then \( \bar{\gamma} \) is not a sublattice of \( A \) since \( \gamma(-) \gamma(+) \neq \gamma(+) \gamma(-) \). The merge over all paths analysis of the program:

\[
\text{if } x > 0 \text{ then while } x \neq 0 \text{ do } x := x - 1 \text{ end if}
\]

(which is 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 \( x \) 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 isotope equations associated with that program:

**THEOREM 9.2.0.1**

Let \( A(\mathcal{W} \mathcal{L}, t, \mathcal{W} \mathcal{L}) \) be a complete lattice, \( \mathcal{L}_P \{L \rightarrow (A \rightarrow A)\} \) be an approximate predicate transformer, \( 2^p(x, p, A, v, n) \) be the complete lattice of all subsets of \( A \), \( \mathcal{L}_P \{L \rightarrow (2^A + 2^A)\} \) be \( \mathcal{A}(\mathcal{L}_P(t(S)\{x, x \in p\})) \) and \( \mathcal{W} \mathcal{L} \{L \rightarrow (2^A + 2^A)\} \) be \( \mathcal{L}_P(t(S)\{x, x \in p\}) \).

- \( \mathcal{W} \mathcal{L} \{L \} \) is a complete \( \mathcal{L} \mathcal{L} \)-morphism

\[
\Psi \rightarrow \mathcal{W} \mathcal{L} \{L \} = \mathcal{L}_P(t(S)\{x, x \in p\}) = \text{MOP}_P(t, \phi)
\]

The above construction is not fully satisfactory since \( 2^A \) is not isomorphic to \( A \) when \( t \) is a complete join-morphism, so that the choice of \( (2^A, \gamma) \) in order to define \( \text{MOP}_P(t, \phi) \) as a least
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 $(A_1, t_1, \gamma_1)$, $(A_2, t_2, \gamma_2)$ be two program analysis frameworks such that $A_1 \triangleleft t_1 \gamma_1 \triangleleft A_2$ and $A_2 \triangleleft t_2 \gamma_2 \triangleleft A_1$. The direct product $(A, t, \gamma)$ of $(A_1, t_1, \gamma_1)$ and $(A_2, t_2, \gamma_2)$ is defined as $A = A_1 \times A_2$, $t = t_1 + t_2$, $\gamma = \gamma_1 + \gamma_2$. If $t_1$ and $t_2$ are correct upper approximations of $\tau$ in $A_1$ and $A_2$, respectively, then $t_1$ and $t_2$ are also correct upper approximations of $\tau$ in $A$.

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 than 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 7.2.0.4.:
Example 10.1.0.3

\[ A_1 = \begin{array}{c|c|c|c|c|c|c|c|c} \hline \end{array} \]

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 \( L_1(E_1, T_1, \mathcal{L}_1, \mathcal{H}_1) \) and \( L_2(E_2, T_2, \mathcal{L}_2, \mathcal{H}_2) \) are complete lattices, we can define the disjoint sum \( L_1 \cup L_2 \) of \( L_1 \) and \( L_2 \) with ordering \( x \leq y \) iff \((x \leq y_1) \lor (y \leq y_2)\) or \((x \leq y_1 \land y \leq y_2)\). The meaning of elements of \( L_1 \cup L_2 \) can be defined as \( y \downarrow L_1 = y_1 \downarrow L_1 \land y \downarrow L_2 \). \( y \uparrow L_1 = y_1 \uparrow L_1 \lor y \uparrow L_2 \) trivially. Even when \( L_1 \) and \( L_2 \) are one-to-one complete meet-morphisms, \( y \) may be neither one-to-one nor a complete meet-morphism. In order to satisfy assumption 5.1.0.2 the \( \mathcal{L}(L_1 \cup L_2) \) must be completed using theorem 5.2.0.4. Then it turns out that the least Moore family containing \( \mathcal{L}(L_1 \cup L_2) \) is equal to \( \mathcal{L}(L_1 \cup L_2) \) (\( y \) 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 \( L_1^2 \) with base \( \mathbb{E}_2, T_2, L_2 \) and exponent \( L_1(E_1, T_1, L_1, H_1) \) (hereafter noted \( \mathcal{L}(L_1 \cup L_2)(E_2, T_2, L_2, H_2) \)) is the set of all isomorphism maps from \( L_1 \) to \( L_2 \) with \( f \in \mathcal{G}(L_1 \cup L_2) \) and only if \( f(x) = g(x) \) for all \( x \in L_1 \). Two program analysis frameworks \( A_1(E_1, T_1, L_1) \) and \( A_2(T_2, L_2) \) can be combined by letting \( g \in \mathcal{L}(L_1 \cup L_2) \) mean that for all \( x \in A_1 \), \( y \in g(x) \) holds whenever \( y \in L_1(x) \) holds.

\[ \text{THEOREM 10.2.0.1} \]

The reduced cardinal power with base \( \mathbb{E}_2, T_2, L_2 \) and exponent \( A_1,E_1,T_1 \) is \( (A_1, E_1, Y_1) \), and \( Y_2 \). Let \( \mathcal{A} \) and \( \mathcal{B} \) be two program analysis systems \( (A_1, E_1, T_1, L_1) \) and \( (A_2, T_2, L_2) \). The corresponding optimal approximate predicate transformer (which takes account of the rule \( f, g \in \mathcal{G}(L_1 \cup L_2) \)) with \( \mathcal{A} \) and \( \mathcal{B} \) is defined as

\[ \mathcal{A} \times \mathcal{B} \]

Example 10.2.0.2

\[ \text{Example 10.2.0.2} \]

\[ \begin{array}{c|c|c|c|c|c|c|c|c} \hline \end{array} \]

According to theorem 10.1.0.1 the direct product of the above analyses cannot yield sharper information. On the other hand using the reduced direct product \( (A_1, E_1, T_1, L_1) \times (A_2, E_2, T_2, L_2) \) and the corresponding optimal approximate predicate transformer (which takes account of the rule \( f, g \in \mathcal{G}(L_1 \cup L_2) \)) we get:

\[ \begin{array}{c|c|c|c|c|c|c|c|c} \hline \end{array} \]

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 
$g(x_1:A_2) = \lambda \gamma \cdot \gamma (x_1(A_2))\gamma (y_2) \gamma (x_2) \gamma (f_2)$. 
Therefore $g(x) = h(x_1) \cdot h(x_2) \cdot h(x_3) \cdot h(x_4) \cdot h(x_5) = g(x_1) \cdot g(x_2) \cdot g(x_3) \cdot g(x_4)$. 
It follows that $t(\lambda g(x_{1,A_2}))$ is isomorphic to $\tau(f_{1,A_2})$ for $A_1 < A_2$.

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

\[
\begin{align*}
  g_1 &= \lambda b \cdot f_1 \cdot b \cdot t \
  g_2 &= \lambda b \cdot f_2 \cdot b \cdot t \cdot g_1 \cdot g_2 \cdot t \
  g_3 &= \lambda b \cdot \text{de} \cdot \text{ar}(g_2(b)) \
  g_4 &= \lambda b \cdot f_1 \cdot b \cdot t \cdot g_1 \cdot g_2 \cdot t \
  g_5 &= \lambda b \cdot f_2 \cdot b \cdot t \cdot g_1 \cdot g_2 \cdot t \
\end{align*}
\]

where $\text{de} (t) = \text{de} \cdot \text{ar}(t) < \text{de} \cdot \text{ar}(t)$, $\text{de} \cdot \text{ar}(t) = \text{de} \cdot \text{ar}(t)$, $\text{de} \cdot \text{ar}(t) = \text{de} \cdot \text{ar}(t)$.

The iterative resolution of this system of equations starting from the initial $\lambda b$, $\tau$ yields $Y(g_1) = Y(g_2) = \lambda (x_1, x_2) \cdot (x_1, x_2) = 0$, $Y(g_3) = \lambda (x_1, x_2) \cdot (x_1, x_2 \cdot 0) = 0$, $Y(g_4) = \lambda (x_1, x_2) \cdot (x_1, x_2 \cdot 0) = 0$, $Y(g_5) = \lambda (x_1, x_2) \cdot (x_1, x_2 \cdot 0) = 0$.

End of Example.

11. REFERENCES


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.


282