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

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

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

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

Several recent papers (among others Cousot & Cousot[77a], Graham & Wegman[76], Kam & Ullman[76], Kildall[73], Rosen[76], Tarjan[76], Wegbreit[76]) have introduced abstract approaches to program analysis which are tantamount to the use of a program analysis framework \((A,\tau,Y)\) where \(A\) is a lattice of (approximate) assertions, \(\tau\) is an (approximate) predicate transformer and \(Y\) is an often implicit function specifying the meaning of the elements of \(A\). This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics.

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

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

In Section 6 we study and exemplify various methods which can be used in order to define a space of approximate assertions or equivalently an approximation function. They include the 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 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-isoton 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
Example 3.1.0.1

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

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

taking \(\phi = \lambda x.(x = 1)\) its least fixpoint characterizes the descendents 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 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 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:

\[
\begin{align*}
\text{wp}(q) &= \{ x \in A \mid \forall x \in V, \exists \pi(X) \wedge \pi(x) \wedge \pi(q(x)) \} \\
\text{wp}(a) &= \{ x \in A \mid \lambda x \in V, \exists \pi(x) \wedge \pi(a x) \wedge \pi(\text{e}(x)) \}
\end{align*}
\]

(notice that \(\subseteq L\), \(\text{wp}(S)\) is a complete join and meet morphism and the inverted program graph \(G' = (n', E', \eta', \psi', c')\) where \(E = \{<i, j> \mid i < j, i \in E\}, c' = \lambda x, j < j', c' = \lambda x, j \leq x\).

Example 3.2.0.1

The inverted program graph corresponding to 2.0.1 is:

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

The corresponding system of backward semantic equations is:

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

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

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

End of Example.

In the following we make no distinction 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 \(P, Q < A\) then "\(Q\) approximate \(P\)" iff \(P \Rightarrow Q\).

This definition of "approximate" is the one which is useful in logical analyses 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 \(A = (\wedge, \vee, \wedge, 1, 0)\). The meaning of the elements of \(A\) is specified by \(a\) (too often implicit) order morphism \(\pi\) mapping \(A\) to a subset of assertions \(A = y(A) \subseteq A\). The intention is that \(A\) is an implementable image of those aspects \(A\) of the program properties which are to be understood at each program point whereas the assertions belonging to \(A\) are ignored (that is approximated from above in \(y(A)\)). To each elementary command \(S\) is associated an isotope map \(i(S)\) from \(A\) to \(A\). The intent is that \(i(S)\) is an approximate predicate transformer such that \(i(S)(1)\) represents the propagation of the information \(i(A)\) through the statement \(S\).

The ideal merge over all paths program analysis (Graham-Wegman[76], Kam & Ulman[77], Rosen[76], Tarjan[76]) is often approximated by a fixpoint solution (Cousot-Cousot[77]), Jones & Muchnick[78], Kaplan & Ulman[78], Kiilsl[79], Tenenbaum[79], Wegbreit[75]). A fixpoint system of isotope equations \(X = \pi(X)\) where \(\pi(A) = \pi(A)\) is associated with the program graph. The approximate invariant assertions are generated by computing iteratively the least fixpoint of \(\pi\) starting from the initial of \(A\) and using any chaotic or asynchronous iteration strategy (Cousot[77]) or the least fixpoint is approximated above using an extrapolation technique in order to accelerate the convergence of the iterates whenever \(A\)
does not satisfy the ascending chain condition (Couso & Couso, 1973).

The design of A, t, the implicit γ and the determination of the construction rules for F are often empirical. The correctness of the least fix-point analysis is usually proved with respect to the approximate merge over all paths analysis, the correctness of which is taken for granted. As opposed to this empirical approach we now provide a formal approach to the systematic design of an approximate program analysis framework (A, t, γ) given (V, A, τ) where τ is sp for forward and wp for backward program analyses.

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 of assertions has been found to provide meaningful information.

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

Let P ≠ A be an assertion and assume that we want to analyze a program \( \tau \) using the merge over all paths two semantic analyses (i.e., \( sp(x=x+y)(Q_1) = \lambda(x,y).(x>y \land y\geq 0) \) and \( sp(x=x+y)(Q_2) = \lambda(x,y).(x>y \land y\geq 0) \) and next comparing them. Since these analyses are not related by the ordering \( \preceq \), the comparison criterion must be application dependent. For example using \( Q_1 \) we can prove that \( sp(λx.x+y)(Q) = λ(x,y).(x>y \land y\geq 0) \) whereas this is impossible with \( Q_2 \). On the contrary the best choice is \( Q_2 \) for the program \( x:=x\); \( x:=x+y \) since \( sp(x=x; x:=x+y)(Q_2) = λ(x,y).(x>y \land y\geq 0) \) which implies \( λ(x,y).(x>y \land y\geq 0) \) whereas \( sp(x=x; x:=x+y)(Q_1) = λ(x,y).(x<y \land y>0) \) does not imply \( λ(x,y).(x<y \land y>0) \).

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 of approximate assertions must be chosen such that for all P ≠ A the set \( \{Q \cup A : P \rightarrow Q\} \) of upper approximations of P in A has a least element.

THEOREM 5.1.0.3

For all P ≠ A the set \( \{Q \cup A : P \rightarrow Q\} \) has a least
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**

If $\mathcal{A}$ is a Moore family containing $\mathcal{A}$ such that $\mu(A)$ is the least Moore family containing $\mathcal{A}$ is:

$$\rho(A) = \lambda S : S \subseteq (\mathcal{A} \cup \{x : true\}) \land S^\ast$$

**Example 5.2.0.5**

Returning to example 5.1.0.1 where $\mathcal{A} = (\mathbb{Z} + \mathbb{Z})$

and $\mathcal{K} = (\lambda u.f \rightarrow, \lambda u.u = 0, \lambda u.\neg u = 0, \lambda u.true)$ the least Moore family containing $\mathcal{A}$ is the one containing $\lambda u.true$, $\mathcal{A}$ and the meets of the non-empty subsets of $\mathcal{A}$ that is the complete lattice:

$$\lambda u.true$$

$\lambda u.u = 0$

$\lambda u.\neg u = 0$

$\lambda u.false$

The corresponding approximation operator is:

$$\rho = \lambda P : P \models \lambda u.false \rightarrow \lambda u.false$$

$$\text{else} \rightarrow \lambda u.u = 0 \rightarrow \lambda u.u = 0$$

$$\text{else} \rightarrow \lambda u.\neg u = 0 \rightarrow \lambda u.\neg u = 0$$

$$\text{else} \rightarrow \lambda u.true \wedge$$

$$\text{until}$$

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 of adjunction functions. Such a pair $\langle \alpha, \gamma \rangle$ defines a Galois connection between $\mathcal{A}$ and the dual of $\mathcal{A}$:

**Definition 5.3.0.3**

Let $L_1(\mathcal{E})$ and $L_2(\mathcal{E})$ be posets, $\alpha \in (L_1 + L_2)$, $\gamma \in (L_2 \cdot L_1)$. The pair $\langle \alpha, \gamma \rangle$ defines a Galois connection between $L_1$ and $L_2$ if and only if:

1. $\alpha$ is antitone ($\forall x_1, x_2 \in L_1, \{x_1 \in \mathcal{E} \implies \alpha(x_1) \leq \alpha(x_2)\}$)
2. $\gamma$ is antitone ($\forall y_1, y_2 \in L_2, \{y_1 \in \mathcal{E} \implies y_1 \leq \gamma(y_2)\}$)
3. $\forall x \in L_1, \{y \in \mathcal{E} \implies \gamma(y) \leq \gamma(y_2)\}$
4. $\forall y \in L_2, \{y \in \mathcal{E} \implies \alpha(x) \leq \gamma(y_2)\}$

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

$\forall x \in L_1, \forall y \in L_2, \{x \in \mathcal{E} \implies (\alpha(x) \leq \gamma(y))\}$

(Birkhoff 67), hence we have:

**Theorem 5.3.0.4**

Let $L_1(\mathcal{E_1})$ and $L_2(\mathcal{E_2})$ be posets, $\alpha \in (L_1 + L_2)$, $\gamma \in (L_2 \cdot L_1)$. $\langle \alpha, \gamma \rangle$ is a pair of adjointed functions if and only if $\langle \alpha, \gamma \rangle$ defines a Galois connection between $L_1(\mathcal{E_1})$ and $L_2(\mathcal{E_2})$. (i.e. iff $\alpha$ and $\gamma$ are isotonous, $\lambda x.\gamma(x) = 0, \alpha \circ \gamma \circ \beta$)

**Corollary 5.3.0.5**

Let $L_1(\mathcal{E_1})$ and $L_2(\mathcal{E_2})$ be posets and $\alpha \in (L_1 + L_2)$, $\gamma \in (L_2 \cdot L_1)$ be adjointed functions:

1. $\gamma \circ \alpha$ is an upper closure operator on $L_1$, $\alpha \circ \gamma$ is...
\[ \gamma = \lambda \xi \lambda \eta \eta_1 \eta_2 (x \xi) \cdot (\alpha(x) \cdot y) \]

\[ \alpha \text{ is an isomorphism from the complete lattice } \gamma \alpha(L_1) \text{ onto the complete lattice } L_2 \text{ the inverse of which is } \gamma. \]

(3) \( \gamma \) is one-to-one if and only if \( \gamma \) is onto and if and only if \( I(x) = \lambda \xi \lambda \eta \eta_1 \eta_2 (x) \)

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

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

It is often the case that \( \gamma \) is only assumed to be a (complete) join-semi-lattice \( A(\mathbb{E}, \mathbb{I}, \mathbb{L}) \) (or dual-meet-semi-lattice for some authors) but since an if-instruction is an instruction \( \gamma \) is in fact a complete lattice (even when the met-operation is not used or what is called meet is not \( \gamma \) [e.g., Wegbreit75]).

When \( \gamma \in (A \rightarrow A) \) is isomote but not a complete meet-morphism the set \( \gamma(A) \) does not fulfill assumption 5.1.0.2 with the consequences examined at paragraph 5.1. The design of \( \gamma(A) \) and \( A \) can be revis ed as stated by theorem 5.2.0.4.

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

\[ \text{Theorem 5.3.0.7} \]

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

- \( \forall x \in A, \gamma(y) = \gamma(\sigma(x)) \)
- \( \gamma \) is a lower closure operator on \( A \)
- \( \gamma \) is a one-to-one complete meet-morphism from \( \gamma(A) \) onto \( A \)

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

\[ \text{Theorem 5.3.0.8} \]

(1) \( \forall x \in A \), \( \{\lambda \xi \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \} = \{\lambda \xi \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(y) \}

(2) \( \forall x \in A \), \( \{\lambda \xi \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \} \subseteq \{\lambda \xi \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(y) \}

(3) \( \forall x \in A, \forall y \in A, \{\lambda \xi \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \} \subseteq \{\lambda \xi \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \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 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 or Equal to an Arbitrary Function

\[ \text{Theorem 6.1.0.1} \]

Let \( L(\mathbb{E}, \mathbb{I}, \mathbb{L}, \mathbb{N}) \) be a complete lattice and \( f \in (L \rightarrow L) \).

- \( \forall x \in L \), \( f \gamma \epsilon (L \rightarrow L) \) be \( \lambda \gamma \epsilon \lambda x \gamma(\gamma(y \epsilon f(x))) = \gamma \epsilon (f \gamma(\gamma(y \epsilon f(x)))) \)

- \( \forall x \in L \), \( f \gamma \epsilon (L \rightarrow L) \) be \( \lambda \gamma \epsilon \lambda x \gamma(\gamma(y \epsilon f(x))) \)

- \( \forall x \in L \), \( f \gamma \epsilon = \gamma \epsilon (L \rightarrow L) \) be \( \lambda \gamma \epsilon \lambda x \gamma(\gamma(y \epsilon f(x))) \)

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 [Oreb43]. However the space of approximate assertions can be designed by successive approximations using the following composition of upper closure operators:

\[ \text{Theorem 6.2.0.1} \]

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

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'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 \( A \) be \((V \rightarrow B)\) and \( A_1 \) be \((D \rightarrow B)\). Let us define:

\[ \forall x \in [1, m], \sigma \in (A \rightarrow A_1) \]

\[ \sigma \in \{ \lambda x \in D{[1, m]} \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \} \subseteq \{ \lambda x \in D{[1, m]} \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \}\]

\[ \sigma \in (A \rightarrow A_1) \]

\[ \sigma \in \{ \lambda x \in D{[1, m]} \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \} \subseteq \{ \lambda x \in D{[1, m]} \lambda \eta \lambda \zeta \lambda \eta_1 \lambda \eta_2 \gamma(x) \}\]

274
\( \sigma \) is an upper closure operator on \( A \) and an assertion \( \text{Pe}_A \) does not state relationships among the program variables if and only if \( \sigma(\text{Pe}_A) \). The approximation assertions on each individual program variable \( x_i \) are next defined using an upper closure operator \( \text{Pe}_j \) on \( A_j \). The induced closure operator \( \sigma \) on \( d(A) \) is defined by \( \rho(x) = \bigwedge_{j=1}^m \text{Pe}_j(x_j) \) where \( x = \bigwedge_{j=1}^m \rho_j \sigma_j(x_j) \). It follows from Theorem 6.20.1 that the \( j = 1 \) composition:
\[
\rho \sigma = \lambda \rho \beta \text{Pe}_m \big[ \lambda(x_1, \ldots, x_m) \in \sigma \big( \bigwedge_{j=1}^m \rho_j \sigma_j(x_j) \big) \big]
\]
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 \( \{p\} \) induced by an upper closure operator \( \beta \) on \( A \) and defined as \( \{p\} = \{q\} \) if and only if \( \beta(q) = \beta(p) \), the approximation process can be understood as essentially consisting in partitioning the space of assertions so that no distinction is made between 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 \( (E, l, l, l, l) \) be a complete lattice. A binary relation \( \theta \) on \( A \) is a complete join congruence relation if and only if:

1. \( \theta \) is an equivalence relation
2. \( \theta \) satisfies the join-substitution property:
\[
\forall x, y, u, v \in \theta \quad x \wedge y \in \theta \implies x \wedge u \in \theta \\
\forall x, y, v \in \theta \quad x \wedge v \in \theta \implies x \wedge u \in \theta
\]
3. \( \theta \) satisfies the join-completeness property:
\[
\forall x, y, u \in \theta \quad x \wedge u \in \theta \implies x \wedge v \in \theta
\]

**THEOREM 6.3.0.2**

If \( \rho \) is an upper closure operator on \( (E, l, l, l, l) \) and \( \forall x, y \in \theta, x \wedge y \in \rho(y) \) if and only if \( \rho(x \wedge y) \), then:

1. \( \{\rho\} = \{\rho \} \) is a complete join congruence relation on \( (E, l, l, l, l) \)
2. \( \rho = \lambda x, y \in \theta(\{x\}) \)

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

**THEOREM 6.3.0.3**

Let \( \theta \) be a complete join congruence relation on the complete lattice \( (E, l, l, l, l) \). \( \lambda x, \theta(\{x\}) \) is an upper closure operator on \( L \).

(Similar results were already proved in Cousot & Cousot[77a] 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).

**THEOREM 6.3.0.4**

A reflexive and symmetric binary relation \( \rho \) on a complete lattice \( (E, l, l, l, l) \) is a join congruence relation if the following three properties are satisfied for \( x, y, z, t \in L \):

1. \( \{x \wedge y \in \theta \} \iff \{x \wedge y \in \theta \} \wedge u \in \theta \iff \theta(\{x \wedge y \in \theta \}) \wedge u \in \theta \}
2. \( \{x \wedge y \in \theta \} \wedge u \in \theta \}
3. \( \{x \wedge y \in \theta \} \wedge u \in \theta \}

**Example 6.3.0.5**

Let \( V \) be a non-empty set of integers included between two bounds \(-\infty \) and \( \infty \) and \( (\infty, \infty) \) or \( (\infty, \infty) \) \( \in V \). The binary relation \( \theta \) on \( A \) is defined by \( \theta \) if and only if \( \theta \) is an isomorphic to \( \theta \) where \( \rho(x) \) is the upper closure operator induced by \( \theta \):

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

In conjunction with 6.20.2, \( \rho \) can be used for static analysis of the range of values of numerical variables (Cousot & Cousot[77b]).

**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 \( \theta \) 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 \( (E, l, l, l, l) \), then \( \forall x, y \in \theta \) and \( \{x \wedge y \in \theta \} \) is a complete and convex sub-join-satellite of \( L \) (let us recall that \( \forall \in L \) is convex and \( a, b, c \in L \) and \( a \subseteq b \) imply that \( c \subseteq b \)).

Here is another representation of convex sub-join-satellites of \( L \) which can be compared with Grätzer & Schmidt[56b]'s theorem which is relative to congruence relations.

**THEOREM 6.4.0.2**

An ideal is a nonvoid subset \( J \) of a lattice \( L \) with the properties that \( \{a \in J \quad | \quad a \leq x \in E \} \) is nonvoid and \( \{a \in J \quad | \quad a \leq x \in E \} \) is a complete and convex sub-join-satellite of \( L \). Given an element \( a \) in a lattice \( L \), the set \( \{x \in L \quad | \quad a \leq x \in E \} \) is evidently an ideal; it is called the principal ideal of \( L \). If every ascending chain in \( L \) is finite, every ideal is principal.

A semi-ideal is a nonvoid subset \( I \) of \( L \) with the property that \( \{a \in J \quad | \quad a \leq x \in E \} \) is a semi-ideal. The dual notion is the one of dual semi-ideal.
THEOREM 6.4.0.2

(1) - Let $I$ be a principal ideal and $J$ be a dual semi-ideal of a complete lattice $L = \{I_1, I_2, I_3, I_4\}$. If $INJ$ is nonvoid then $INJ$ is a complete and convex sub-join-semilattice of $L$.

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

THEOREM 6.4.0.3

Let $\{I_1, I_2\}$ be a family of principal ideals of the complete lattice $L = \{I_1, I_2, I_3, I_4\}$ containing $L$. Then $\lambda x(I_1, I_2, x \in L)$ is an upper closure operator on $L$.

Example 6.4.0.4

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

\[ \begin{array}{c c c c}
 & X_1 & X_2 & X_3 \\
X_0 & & & \\
X_1 & & & \\
X_2 & & & \\
X_3 & & & \\
X_4 & & & \\
\end{array} \]

(\text{where } I, \land, \lor, \neg, =, \neq, \tau \text{ respectively stand for } \lambda x, false, \lambda x, \neq 0, \lambda x, \neq 0, \lambda x, \neq 0, \lambda x, \neq 0, \lambda x, \neq 0, \lambda x, \neq 0). A further approximation can be defined by the following family of principal ideals:

\[ \{I_1, I_2, I_3, I_4\} \]

which induces an upper closure operator $p$:

\[ \begin{array}{c c c c}
 & X_1 & X_2 & X_3 \\
X_0 & & & \\
X_1 & & & \\
X_2 & & & \\
X_3 & & & \\
X_4 & & & \\
\end{array} \]

End of Example.

7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER

INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

In addition to $\lambda$ and $\gamma$ the specification of a program analysis framework also includes the choice of an approximate predicate transformer $\tau_f(L \rightarrow \lambda (A \rightarrow A))$ (or a monoid of maps on $A$ plus 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 $\lambda$ and the formal semantics of the considered programming language.

7.1 A Reasonable Definition of Correct Approximate Predicate Transformers

At paragraph 3, given $(\forall, A, \tau)$ the minimal assertion which is invariant at point $i$ of a program $\pi$ with entry specification $\phi A$ was defined as:

\[ p_1 = \forall p \in \text{Path}(i) \]

Therefore the minimal approximate invariant assertion is the least upper approximation of $p_1$ in $\mathfrak{R}$ that is:

\[ p_1 = \rho(\forall p \in \text{Path}(i)) \]

Even when $\text{Path}(i)$ is a finite set of finite paths the evaluation of $\tau_f(p)(\delta)$ is hardly machine-implementable since for each path $p = \sigma_1, \ldots, \sigma_m$ the computation sequence $X_0 = \delta, X_1 = \tau_f(\sigma_1)(X_0), \ldots, X_m = \tau_f(\sigma_m)(X_{m-1})$ does not necessarily only involve elements of $\mathfrak{R}$ and $\mathfrak{R}$. Therefore using $\mathfrak{R}$ and $\tau_f(L \rightarrow \mathfrak{R}(X))$ a machine-representable sequence $X_0 = \delta, X_1 = \tau_f(\sigma_1)(X_0), \ldots, X_m = \tau_f(\sigma_m)(X_{m-1})$ is used instead of $X_0, \ldots, X_m$ which leads to the expression:

\[ q_1 = \rho(\tau_f(\delta)) \]

The choice of $\tau$ and $\delta$ is correct if and only if $q_1$ is an upper approximation of $p_1$ in $\mathfrak{R}$ that is if and only if:

\[ \forall p \in \text{Path}(i) \]

For the particular for the entry point we must have $\phi = \tau_f(\delta)$ so that we can state the following:

**DEFINITION 7.1.0.1**

(1) - An approximate predicate transformer $\tau_f(L \rightarrow \lambda (A \rightarrow A))$ is said to be a correct upper approximation of $\tau_f(L \rightarrow \lambda (A \rightarrow A))$ in $\mathfrak{A}$ if and only if for all $\Phi \in \mathfrak{A}$ such that $\phi = \tau_f(\delta)$ and program $\pi$ we have $\operatorname{MOP}_\pi(\tau_f(\delta)) = \operatorname{MOP}_\pi(\tau_f(\delta))$.

(2) - Similarly if $\Phi \in \lambda (A \rightarrow A)$, $\tau_f(L \rightarrow \lambda (A \rightarrow A))$ is said to be a correct upper approximation of $\tau_f(L \rightarrow \lambda (A \rightarrow A))$ in $\mathfrak{A}$-\textit{A} if and only if for all $\Psi$, $\Psi \in \mathfrak{A}$ the following equivalence condition holds:

\[ \forall p \in \text{Path}(i) \]

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

End of Example.
**Theorem 7.1.0.2**

1. \( \overline{t} : (L + (A \rightarrow \overline{A})) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \rho_0 \) iff \( (s \subseteq L, w \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

2. \( \overline{t} \) is a correct upper approximation of \( \tau : (L + (A \rightarrow A)) \) in \( \alpha \) iff \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

If \( \overline{t} \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \rho_0 \), we have:

\[
\rho(\overline{t} \circ \rho_0) = \rho(t_0) = \rho_0 \circ \rho_0.
\]

Theorem 7.1.0.3

1. If \( t \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \rho_0 \), then \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

2. If \( t \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \), then \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

Similar results hold for fixpoint analysis of programs.

**Theorem 7.1.0.4**

Let \( \overline{t} : (L + (A \rightarrow A)) \) be an isotope correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \) where \( A \rightarrow A \). Let \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) be a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

- \( F_\overline{t}(s, \phi) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

- \( F_\overline{t}(s, \phi) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

Notice that in theorem 7.1.0.2 the maps \( \tau : S \rightarrow P(t(s))) \) are not assumed to be isomorphic. Yet, isomorphism is assumed in theorem 7.1.0.4 and is a customary hypothesis in the literature. An apparent justification of this requirement is to ensure that the system of equations \( \lambda_\overline{t}(s, \phi) \) associated with a program \( \lambda_\overline{t} \) has fixpoints which can be obtained as limits of iteration sequences. But this could also be achieved without isomorphism hypothesis taking \( \lambda_\overline{t}(s, \phi) \) instead of \( F_\overline{t}(s, \phi) \).

**Theorem 7.1.0.5**

Let \( \overline{t} : (L + (A \rightarrow A)) \) be a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \). Then \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

- \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) is a correct upper approximation of \( t : (L + (A \rightarrow A)) \) in \( \alpha \).

Hence the isotope hypothesis is even not necessary for technical purposes. However, the profound justification of this hypothesis can be found in the fact that among all possible approximate predicate transformers which can be used with a given set \( A \) of approximate assertions the designer of a program analysis framework intuitively thinks to the best approximate predicate transformer which happens to be isomorphic. This property also explains the fact that no significant counter-examples to the isomorphism hypothesis have ever been found.

**7.2. The Best Approximate Predicate Transformer**

**Induced by a Space of Approximate Assertions**

**Definition 7.2.0.1**

If \( \overline{t}_1, \overline{t}_2 \) are correct upper approximations of \( t : (L + (A \rightarrow A)) \) in \( \alpha \), then we say that \( \overline{t}_1 \) is better than \( \overline{t}_2 \) if for all \( s \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) and all programs \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) holds:

\[
\nu(\overline{t}_1(s, \phi)) = \nu(t(s, \phi)).
\]

**Lemma 7.2.0.2**

Let \( \overline{t}_1, \overline{t}_2 \) be correct upper approximations of \( t : (L + (A \rightarrow A)) \) in \( \alpha \). If \( \nu \subseteq L, \nu \subseteq \alpha, \tau : S \rightarrow P(t(s))) \) and \( \overline{t}_1 \) or \( \overline{t}_2 \) is isomorphic, then \( \overline{t}_1 \) is better than \( \overline{t}_2 \).

**Corollary 7.2.0.4**

If \( A \rightarrow A \) and \( \lambda_\overline{t}(s, \phi) \) then \( \lambda_\overline{t}(s, \phi) : \tau \subseteq P(t(s))) \) is isomorphic, it is the best correct upper approximation of \( t : (L + (A \rightarrow A)) \).

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

**Example 7.2.0.5**

Coming back to examples 6.2.0.2 and 6.3.0.5 assume that \( \lambda \) is the set of integers included between two bounds \( \Rightarrow \) and \( \Leftarrow \).

For simplicity we shall assume that \( m = 2 \). Let \( \lambda \) be the complete lattice \( \lambda \{a, b : a \subseteq A \neq \emptyset \} \) with ordering \( [a, b] \) \( \subseteq [c, d] \) iff \( a \subseteq b \subseteq c \subseteq d \). Let \( \lambda \) be the infimum. Let \( \lambda \) be the suprema. Let \( \gamma \) be such that every \( \gamma \subseteq [a, b] \neq \emptyset \) be such that every \( \gamma \subseteq [a, b] \neq \emptyset \) let \( \gamma \neq [a, b] \) be such that \( \gamma \subseteq [a, b] \neq \emptyset \).

Let \( x \subseteq \lambda \) and \( x \neq \emptyset \) let \( \lambda \neq [a, b] \neq \emptyset \) let \( \lambda \neq [a, b] \neq \emptyset \).

Given \( \gamma \) let us determine the best correct upper approximation of \( \gamma \subseteq \lambda \) again for lack of space we just study the case of \( \gamma \subseteq \lambda \).

Since \( \gamma \) is a injective complete meet-morphism the adjoint function \( \gamma^* \) of \( \gamma \) is determined by \( \lambda \).

The same way, \( \gamma \) is a injective complete meet-morphism the adjoint function \( \gamma^* \) of \( \gamma \) is determined by \( \lambda \).

Since \( \gamma \) is a injective complete meet-morphism the adjoint function \( \gamma^* \) of \( \gamma \) is determined by \( \lambda \).

Thus \( \gamma = \gamma \cdot \gamma \) and \( \gamma = \gamma \cdot \gamma \) and \( \gamma = \gamma \cdot \gamma \) and \( \gamma = \gamma \cdot \gamma \).
According to theorem 7.2.0.4 the best upper approximation of \( p(\lambda x_1 y_1, \lambda x_2 y_2) \) in \( A \) is \( t = \lambda x_1 y_1 \lambda x_2 y_2 \) where \( P \subseteq \{[a, b], [c, d] \} \) where \( a \) and \( b \) in which case \( t(P) = a(Q) \) where \( Q = \lambda x_y \lambda y_1 \lambda y_2 \lambda x_1 y_2 \). Therefore \( t(P) = \lambda x_1 y_1 \lambda x_2 y_2 \lambda x_1 y_2 \lambda x_2 y_1 \lambda x_2 y_2 \). The same way \( a(Q) = \lambda x_1 y_1 \lambda x_2 y_2 \lambda x_2 y_1 \lambda x_2 y_2 \). Therefore \( t(P) = \lambda x_1 y_1 \lambda x_2 y_2 \lambda x_2 y_1 \lambda x_2 y_2 \lambda x_2 y_2 \). The same way \( a(Q) = \lambda x_1 y_1 \lambda x_2 y_2 \lambda x_2 y_1 \lambda x_2 y_2 \). Therefore \( t(P) = \lambda x_1 y_1 \lambda x_2 y_2 \lambda x_2 y_1 \lambda x_2 y_2 \lambda x_2 y_2 \).

End of Example.

**Example 7.2.0.8**

Some program analyses (such as "reaching definitions", "available expressions", "live variables", "... Aho & Ullman[77]) are "history sensitive" because the approximate assertions which are useful at each program point \( p \) characterize sets of sequences of states for execution paths from the entry point to \( p \) and not sets of states. In such a case Hoare[70]'s 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 \( L \) of elementary atoms, the set of sequential traces is the free monoid \( T(1, C) \) generated by \( L = L_0 \cup L_1 \).

The concatenation operation \( \ast \) is extended to elements of the complete lattice \( 2^V(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \) by \( S = \{\ast, \ast \ast, \ast \ast \ast \ast \} \). Let us define a function \( "\ast \ast \ast \" \) of traces in the set of states \( T(1, \{1, 2\}) \) associated with a program \( T \) and an entry specification \( \Phi \). The set \( T \) is the classical system of state data flows for available expressions (Aho & Ullman[77]) and that the largest possible solution (least for 3) is desired.

End of Example.

8. **HIERARCHY OF PROGRAM ANALYSIS FRAMEWORKS**

Once the semantics of programs has been defined by \( (A, T) \) all program analysis frameworks \( (A, T, \gamma) \) are specified up to the isomorphism \( \gamma \) by \( p(A), \lambda x(\alpha_1, \alpha_2) \) such that \( p \ast \gamma a \) is an upper closure operator on \( A \) and \( A \ast \gamma a \ast \gamma a \). Program analysis frameworks can be partially ordered using the ordering of the corresponding closure operators on \( A \) since whenever \( p \ast \gamma a \ast \gamma a \ast \gamma a \) is so that program analysis frameworks corresponding to \( p \ast \gamma a \ast \gamma a \) yield sharper information than the ones corresponding to \( p \ast \gamma a \ast \gamma a \) (whichever global analysis method is used).

The following theorem is a constructive version of War[42, 71, 72]:

**Theorem 8.0.1**

The set of upper closure operators on a complete lattice \( L(\mathbb{E}, C, D, E) \) is a complete lattice \( L(\mathbb{E}, C, D, E) \ast \mathbb{E} \) where \( \mathbb{E} \ast \mathbb{E} \ast \mathbb{E} \) is a complete lattice.
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:

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:

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

The corresponding framework was used for the automatic discovery of linear restraints among variables of programs [Cousot & Halbwachs [78]].

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,E) 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 81] that when A satisfies the ascending chain condition and \( \forall s \in L, t(s) \) is isotope we have \( \text{MOP}(t,A) \subseteq \text{MOP}(t,(A,A)) \). Also the additional hypothesis that \( \forall s \in L, t(s) \) is a join-morphism (sometimes called join-distributive map) implies \( \text{MOP}(t,A) \subseteq \text{MOP}(t,A,A) \). Slightly more general is the following:

**THEOREM 9.1.0.1**

If \( A(1,\leq,\leq) \) is a complete lattice and \( t: L \to (A,A) \) is such that \( \forall s \in L, t(s) \) is isotope then for all programs \( v \) and \( \phi A, \text{MOP}(t,A) \subseteq \text{MOP}(t,A,A) \). If \( \forall s \in L, t(s) \) is a complete \( \mu \)-morphism then \( \text{MOP}(t,A) \subseteq \text{MOP}(t,A,A) \).

(This theorem is implicitly used at paragraph 3 taking \( A = (V,\leq) \to (\mu A,\leq) \) for \( A(1,\leq,\leq) \) and \( \mu A = \text{MOP}(t,A) \).

If \( A(1,\leq,\leq) \) and \( t: L \to (A,A) \) then the above theorem establishes the correctness of \( \text{MOP}(t,A) \) with respect to \( \text{MOP}(t,A) \). In the literature the correctness of \( \text{MOP}(t,A) \) is generally taken for granted. Also \( \text{MOP}(t,A) \) is considered as the desired solution to program-wide analysis problems since whenever some \( t(s) \) is not a complete join-morphism \( \text{MOP}(t,A) \) can be strictly better than \( \text{MOP}(t,A,A) \). When \( A \) satisfies the ascending chain condition \( \text{MOP}(t,A,A) \) is computable, which is not necessarily the case of \( \text{MOP}(t,A) \). 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 \( A, t, e, \) and \( \gamma \).

In our opinion the above argument is not entirely convincing since for different correct approximate predicate transformers \( t_1, t_2: L \to (A,A) \) it may be the case that \( \text{MOP}(t_1,A) \subseteq \text{MOP}(t_2,A) \). In order to retrieve 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 \( \alpha: L \to (A,A) \) be the best correct upper approximation of \( \alpha: L \to (A,A) \) in \( A(1,\leq) \). If \( \phi A \) is a complete sublattice of \( A \) then \( \text{MOP}(t,A) \subseteq \text{MOP}(t,A,A) \).

**Example 9.1.0.3**

If \( A = (Z,+) \) and \( A = \gamma(A) \) then:

\[ A = \{0,1,2\} \]

and \( \gamma(0) = 0, \gamma(1) = 1, \gamma(2) = 2 \). The merge over all paths analysis of the program:

\[ \text{if } x > 0 \text{ then } y := x ; \text{ else } y := 0 \]

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 \( A(e,\leq,\leq,\leq) \) be a complete lattice, \( t: L \to (A,A) \) be an approximate predicate transformer, \( \mu(\varepsilon \in \mu A, \leq) : A(1,\leq) \) be \( \text{AS}(\varepsilon: t(x); x : \varepsilon P) \) and \( \mu(\varepsilon \in \mu A) : \varepsilon P \) be \( \text{AS}(\varepsilon : t(x); x : \varepsilon P) \).

- \( \forall s \in L, t(s) \) is a complete \( \mu \)-morphism:
- \( \forall s \in L, t(s) \) is a complete \( \mu \)-morphism:

The above construction is not fully satisfactory since \( (2,1) \) is not isomorphic to \( (A,t) \) when \( t \) is a complete join-morphism, so that the choice of \( (2,1) \) in order to define \( \text{MOP}(t,A) \) as a least
fixpoint is unnecessarily too complicated. The following construction is preferable:

**LEMMA 9.2.0.2**

Let \( L \cup L \cup \{ 0, 1 \} \) be a complete lattice and \( U \), \( \mu \in 2^{(L \cup L \cup \{ 0, 1 \})} \) be defined as:

\[
\begin{align*}
\mu &= \lambda \psi \cdot \mu(\psi) \lor \mu(\neg \psi) \land \mu(\psi) \\
\mu &= \lambda \psi \cdot \mu(\neg \psi) \lor \mu(\psi) \land \mu(\psi)
\end{align*}
\]

\( \mu \) and \( \nu \) are upper closure operators on \( L \).

Let \( \alpha(2^{L \cup L \cup \{ 0, 1 \}}) \) be the join lattice \( 2^{L \cup L \cup \{ 0, 1 \}} \) and \( \mu \) and \( \nu \) in the lattice of upper closure operators on \( 2^{L \cup L \cup \{ 0, 1 \}} \).

**THEOREM 9.2.0.3**

Let \( \tau \) be an upper closure operator on \( L \cup L \cup \{ 0, 1 \} \), then \( \eta(\lambda \psi \cdot \nu(\psi) \lor \nu(\neg \psi) \land \nu(\psi)) \) is the greatest upper closure on \( L \) which is less than or equal to \( \tau \) and which is a complete join morphism.

**THEOREM 9.2.0.4**

\( \mu \) is a complete join morphism and \( \tau = 2^{L \cup L \cup \{ 0, 1 \}} \).

**Example 9.2.0.5**

Coming back to example 9.1.0.3, where \( A = 2 \) and \( \mu(A) = \{ \lambda \psi \cdot \nu(\psi) \lor \nu(\neg \psi) \land \nu(\psi) \} \), \( \lambda \psi \cdot \nu(\psi) \lor \nu(\neg \psi) \land \nu(\psi) \) and applying theorem 9.2.0.4, we get \( \nu(\psi) \) as \( \{ \lambda \psi \cdot \nu(\psi) \lor \nu(\neg \psi) \land \nu(\psi) \} \) such that \( \nu(\psi) \) is the best upper approximation of \( \tau \) in \( A \).

**End of Example.**

It is clear that when \( \tau(A) \) is the satisfying ascending chain condition, the construction of theorem 9.2.0.4 may lead to a refined space of approximate assertions \( \tau(A) \) which does not satisfy the ascending chain condition. Then the iterative computation of \( \nu \) is not naturally converging in a finite number of steps. Nevertheless this least fixpoint can be approximated from above using an extrapolation technique in order to accelerate the convergence of the iterates. Such a technique was developed in Cousot & Cousot [77] using a "widening operator" \( \psi \cdot \mu(\psi) \land \phi(\psi) \).

**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 \( A_1 \neq A_2 \) and \( t_1, t_2 \) are correct upper approximations of \( \tau \) in \( A_1, A_2 \). The direct product \( A, t, \gamma \) is defined as \( A = A_1 \times A_2, t = t_1 \times t_2, \gamma = \gamma_1 \times \gamma_2 \).

\( \gamma_1 \) is a subset of \( A_1 \) and \( \gamma_2 \) is a subset of \( A_2 \). The direct product \( \gamma \) is the direct product of \( \gamma_1 \) and \( \gamma_2 \).

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 9.2.0.7 and 9.2.0.8.

**THEOREM 10.1.0.2**

Let \( (A_1, t_1, \gamma_1), (A_2, t_2, \gamma_2) \) be two program analysis frameworks such that \( A_1 \neq A_2 \) and \( t_1, t_2 \) are correct upper approximations of \( \tau \) in \( A_1, A_2 \). The direct product \( A, t, \gamma \) is defined as \( A = A_1 \times A_2, t = t_1 \times t_2, \gamma = \gamma_1 \times \gamma_2 \) and \( \gamma_1 \) and \( \gamma_2 \) are respectively the best upper approximation of \( \tau \) in \( A \).

The reduced product \( (A_1, t_1, \gamma_1), (A_2, t_2, \gamma_2) \) is \( A, t, \gamma \) where \( A = A_1 \times A_2, t = t_1 \times t_2, \gamma = \gamma_1 \times \gamma_2 \).

The reduced product is a special case of the product of the space of approximations corresponding to the meet of the closure operators \( \gamma_1 \) and \( \gamma_2 \) (Theorem 8.0.6) viz. to the join \( \{ \gamma_1 \cup \gamma_2 \} \) of the Moore families \( \gamma_1 \) and \( \gamma_2 \).

Since \( \gamma = \gamma_1 \times \gamma_2 \) is a representation of the space of approximations corresponding to the meet of the closure operators \( \gamma_1 \) and \( \gamma_2 \), this inequality can be strict.
Example 10.1.3.4

\[ A_1 = \begin{array}{c}
T \\
1 \\
\end{array} \quad A_2 = \begin{array}{c}
T \\
1 \\
\od \\
\end{array} \]

\[ \gamma_1(1) = \lambda x, \text{false}, \quad \gamma_1(0) = \lambda x, (x = 0), \quad \gamma_1(1) = \lambda x, (x \neq 0), \quad \gamma_1(1) = \lambda x, (x \text{ modulo } 2 = 0), \quad \gamma_2(0) = \lambda x, (x \text{ modulo } 2 = 1), \quad \gamma_2(1) = \lambda x, \text{true}, \quad \gamma_2(1) = \lambda x, \text{false}, \quad \gamma_2(1) = \lambda x, (x = 0), \quad \gamma_2(1) = \lambda x, (x \neq 0) \]

Remark 10.1.4

Let \( L_1(\text{E}_1), L_2(\text{E}_2) \) 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 \( L_1(\text{E}_1, T_1, \eta_1, \lambda_1) \) and \( L_2(\text{E}_2, T_2, \eta_2, \lambda_2) \) are complete lattices, we can define the disjoint sum \( L_1 + L_2 \) as \( L_1 \cup L_2 \cup \{ \top, \bot \} \) with ordering \( x \leq y \) iff \( (x \leq_1 y) \) or \( (y \leq_2 x) \) or \( (x \leq_1 \top, y \leq_2 \bot) \) or \( (y \leq_1 \top, x \leq_2 \bot) \). The meaning of elements of \( L_1 + L_2 \) can be defined as \( Y(1) = \gamma_1(1) \wedge \gamma_2(1) \), \( Y(x) = \gamma_1(x) \) if \( x \leq_1 L_1 \), \( Y(x) = \gamma_2(x) \) if \( x \leq_2 L_2 \). Even when \( \gamma_1 \) and \( \gamma_2 \) 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 set \( Y(L_1 + L_2) \) must be completed using theorem 5.2.0.4. Then it turns out that the least Moore family containing \( Y(L_1 + L_2) \) is equal to \( Y(L_1 \cup L_2) \) (\( Y \) as defined in theorem 10.1.3.2). Therefore the use of disjoint sums amounts to the use of reduced products.

End of Remark.

10.2 Reduced Cardinal Power of Program Analysis Frameworks

The cardinal power \( L_1^{L_2} \) with base \( L_2(\text{E}_2, T_2, \eta_2, \lambda_2) \) and exponent \( L_1(\text{E}_1, T_1, \eta_1, \lambda_1) \) (hereafter noted \( \text{fac}(L_1 + L_2)(\text{E}_1, T_1, \eta_1, \lambda_1) \)) is the set of all iso-tone maps from \( L_1 \) to \( L_2 \) with \( f \in \text{EG} \) if and only if \( f(x) \leq_2 g(x) \) for all \( x \in L_1 \). Two program analysis frameworks \( (A_1, \text{E}_1, \gamma_1) \) and \( (A_2, \text{E}_2, \gamma_2) \) can be combined by letting \( g \leq \gamma_1 \circ f \circ \lambda_1 \). Let \( f(x) \geq_2 g(x) \) for all \( x \in A_1 \). \( \gamma_1 \circ f \circ \lambda_1 \) holds whenever \( \gamma_1(x) \) holds.

Theorem 10.2.0.1

The reduced cardinal power with base \( (A_2, \text{E}_2, \gamma_2) \) and exponent \( (A_1, \text{E}_1, \gamma_1) \) is \( (A_1, \text{E}_1, \gamma_1) \) where

\[ f = \alpha \circ (\text{fac}(A_1 + A_2)), \quad \gamma_1 \circ (\text{fac}(A_1 + A_2) \circ \gamma_2) \circ (\text{fac}(A_1 + A_2)) \text{ is } \lambda_1, \text{if } \neq \text{fac}(A_1 + A_2) \circ \gamma_2 \circ (\text{fac}(A_1 + A_2)), \quad \gamma_1 \circ (\text{fac}(A_1 + A_2) \circ \gamma_2) \circ (\text{fac}(A_1 + A_2)) \text{ is } \lambda_2, \text{if } \neq \text{fac}(A_1 + A_2) \circ \gamma_2 \circ (\text{fac}(A_1 + A_2)), \quad \gamma_1 \circ (\text{fac}(A_1 + A_2) \circ \gamma_2) \circ (\text{fac}(A_1 + A_2)) \text{ is } \lambda_1, \text{if } \neq \text{fac}(A_1 + A_2) \circ \gamma_2 \circ (\text{fac}(A_1 + A_2)), \quad \gamma_1 \circ (\text{fac}(A_1 + A_2) \circ \gamma_2) \circ (\text{fac}(A_1 + A_2)) \text{ is } \lambda_1, \text{if } \neq \text{fac}(A_1 + A_2) \circ \gamma_2 \circ (\text{fac}(A_1 + A_2)). \]
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 $g(x_1+A_2)$ then $Y(\gamma) = Y(\gamma_1) + 2Y(\gamma_2) + 2Y(\gamma_3) - 2Y(\gamma_4)$. Therefore $h(\gamma) = h(\gamma_1)/2$, $h(\gamma_2) = h(\gamma_3)/2$, $h(\gamma_4) = h(\gamma_5)/2$. It follows that $g(\gamma) = \gamma(1) + \gamma(2) + \gamma(3)$ is isomorphic to $(\gamma_1/2) + (\gamma_2/2)$ for $A_3 \approx A_2$.

The system of equations associated with the above program and the entry specification $\lambda_b \gamma$ is then:

$$
\begin{align*}
g_1 &= \lambda_b \gamma \text{ if } b \text{ then } + \text{ else } + \text{ else } \gamma \text{ else } \gamma \text{ else } \\
g_2 &= \lambda_b \gamma \text{ if } b \text{ then } g(\gamma) \text{ else } \gamma \text{ else } \gamma \text{ else } \\
g_3 &= \lambda_b \gamma \text{ else } \gamma \text{ else } \gamma \text{ else } \gamma \text{ else } \\
g_4 &= \lambda_b \gamma \text{ else } \gamma \text{ else } \gamma \text{ else } \gamma \text{ else } \\
g_5 &= \lambda_b \gamma \text{ else } \gamma \text{ else } \gamma \text{ else } \gamma \text{ else }
\end{align*}
$$

where $\lambda_b(\gamma_1) = \gamma_2$, $\lambda_b(\gamma_1) = \gamma_3$, $\lambda_b(\gamma_1) = \gamma_4$, $\lambda_b(\gamma_1) = \gamma_5$, $\lambda_b(\gamma_1) = \gamma_6$, $\lambda_b(\gamma_1) = \gamma_7$.

The iterative resolution of this system of equations starting from the initial $\lambda_b \gamma$ yields $Y(\gamma) = Y(\gamma) + Y(\gamma) + Y(\gamma) + Y(\gamma) - Y(\gamma)$.

End of Example.

11. REFERENCES


Cousot P. [1977], Asynchronous iterative methods for solving a fixed point system of motions equations in a complete lattice, Rapport de Recherche n°68, Laboratoire IMAG, Grenoble, (Sept. 1977).


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. [1978], Data types as lattices, SIAM Comp. 5, 3 (Sept. 1978), 522-567.


