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], Kildal[73], Rosen[76], Tarjan[76], Wegbreit[75]) have introduced abstract approaches to program analysis which are tantamount to the use of a program analysis framework \((A, t, Y)\) where \(A\) is a lattice of (approximate) assertions, \(t\) is an (approximate) predicate transformer and \(Y\) is an often implicit function specifying the meaning of the elements of \(A\). This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics.

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

The design of a space of approximate assertions \(A\) is studied in Section 5. We first justify the very reasonable assumption that \(A\) must be chosen such that the exact invariant assertions of any program must have an upper approximation in \(A\) and that the approximate analysis of any program must be performed using a deterministic process. These assumptions are shown to imply that \(A\) is a Moore family, that the approximation operator (which defines the least upper approximation of any assertion) is an upper closure operator and that \(A\) is necessarily a complete lattice. We next show that the connection between a space of approximate assertions and a computer representation is naturally made using a pair of isotope adjoint functions. This type of connection between two complete lattices is related to Galois connections thus making available classical mathematical results. Additional 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-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[73a]'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
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 isotope 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.

where $\text{path}(l)$ is the set of paths from the entry point $n_1$ to the vertex $i$ and $\overline{\pi} \in (E^+ \rightarrow (A \rightarrow A))$ is recursively defined as follows: if $p$ is an empty path then $\overline{\pi}(p)$ is the identity map on $A$ else $p = (q, e)$ where $q e < a$, $a e$ and $\overline{\pi}(p) = \lambda \phi. t[C(a)][\overline{\pi}(q)(\phi)]$.

The system of equations $P = \overline{\pi}(t, \phi)(P)$ associated with the program $\pi$ using $(A, t)$ and $\phi$ is defined as follows:

$$
\begin{align*}
P_1 &= \phi \\
\cap p_j &= \bigcup_{i \in \text{pred}(j)} t[C(<i, j>, P_1)] \\
\cap j[n_1] &= (n_1)
\end{align*}
$$

Notation: If $M([s, i, t], [\lambda, \Pi])$ is a complete lattice then the set $\{L \rightarrow M\}$ of total maps from the set $L$ into...
Example 3.1.0.1

The system of forward semantic equations may be mapped to:
\[ P_1 = \lambda x.(x \leq 101) \]

\[ P_1 = \emptyset \quad P_2 = \lambda x.(x \leq 101) \]
does not satisfy the ascending chain condition (Cousot & Cousot[77a]).

The design of $A, t, \gamma$ 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 $(A, t, \gamma)$ given $(V, \bar{A}, \tau)$ where $\tau$ is sp for forward and $\bar{sp}$ 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 $\bar{A} \times A$ of assertions has been found to provide meaningful information.

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

Let $P \times \bar{A}$ be an assertion and assume that we want to analyze a program $\Pi$ using the merge over all paths semantic analysis and an entry condition $Q$ which is an upper approximation of $P$ in $\bar{A}$. What is the best program for $\Pi$? It is defined by $\Pi = \{ \lambda, \bar{A} \rightarrow \bar{A}, \bar{A} \rightarrow \bar{A}, x \rightarrow y \}$.

$F_{\Pi}(\tau, Q) \rightarrow F_{\Pi}(\tau, Q)$ and by isotony the analysis $\bar{F}_{\Pi}(F_{\Pi}(\tau, Q))$ is more precise than $\bar{F}_{\Pi}(F_{\Pi}(\tau, Q))$.

Hence $Q$ must be a minimal upper approximation of $P$ in $\bar{A}$ which is such that $(P \rightarrow Q \land Q \subseteq \bar{A}) : (P \rightarrow Q \land Q \subseteq \bar{A})$. Assume that the set $U$ of minimal upper approximations of $P$ in $\bar{A}$ has cardinality greater than 1. What is the best possible choice of $Q$ in $U$? If $Q_1, Q_2 \in U$ and $Q_1 \subseteq Q_2$ then $Q_1$ and $Q_2$ are not necessarily comparable so that $\bar{F}_{\Pi}(\tau, Q_1)$ and $\bar{F}_{\Pi}(\tau, Q_2)$ may be not comparable. Hence "best" cannot be defined using the preciseness criterion provided by $\rightarrow$. The only way to determine which of the two alternatives will be the most useful in question about the program is to try both of them. Also the best choice may vary from one

two semantic analyses (i.e., $sp(x := y)(Q_1) = \lambda(x, y), (x < y \land y = 0)$ and $sp(x := x+y)(Q_2) = \lambda(x, y), (x \neq y \land y = 0)$) and next comparing them. Since these analyses are not related by the ordering $\rightarrow$, the comparison criterion must be application dependent. For example using $Q_1$ we can prove that $\bar{F}(x := x+y)(Q) \equiv \lambda(\bar{A}, x \rightarrow y)$ whereas this is impossible with $Q_2$. On the contrary the best choice is $Q_2$ for the program $(x := x+y)$ since $sp(x := x+y)(Q_2) = \lambda(x, y), (x \neq y \land y = 0)$ implies $\lambda(x, y), (x \neq y \land y = 0)$ whereas $sp(x := x); (x := x+y)(Q_1) = \lambda(x, y), (x \neq y \land y = 0)$ does not imply $\lambda(x, y), (x \neq y \land y = 0)$.

End of Example.

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

THEOREM 5.1.0.3

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

5.2 The Approximation Operator

DEFINITION 5.2.0.1 Approximation Operator

$\rho \in A \rightarrow \bar{A}$

$\rho = \lambda(\bar{A}, P \rightarrow \bar{A})$

$P(Q)$ is the least upper approximation of $P$ in $\bar{A}$. Since $\bar{A}$ is a Moore family it follows from Monteiro & Ribeiro[42, Th. 5.3 and 5.1] that:

THEOREM 5.2.0.2

(1) $\rho$ is an upper closure operator (that is $\rho$ is isotone (if $P \times \bar{A}$ and $P \rightarrow Q$ then $\rho P \rightarrow \rho(Q)$)), extensive (for all $P \times \bar{A}$, $P \rightarrow \rho(P)$) and idempo
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 $\bar{A} \in \mathcal{A}$, the upper closure operator $\rho$ on $A$ such that $\rho(A)$ is the least Moore family containing $\bar{A}$ is:

$$\rho = \lambda p. \lambda x. \mu A(\bar{A} \cup \{x \mid x, true\}) \land p \rightarrow q$$

$A = \{s \mid s \subseteq \{x \cup \{x, true\}\} \land s \neq \emptyset\}$

**Example 5.2.0.5**

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

```
false
true, \mu, u = 0
true, \mu, u \neq 0
true, \mu, u = 0
```

The corresponding approximation operator is:

$$\rho = \lambda p. i f p = \mu, false then \mu, false$$

else if $p = \mu, u = 0$ then $\mu, u = 0$

else if $p = \mu, u \neq 0$ then $\mu, u \neq 0$

else $\mu, true$.

**End of Example.**

### 5.3 Representation of the Lattice of Approximate Assertions

In order to represent the approximate assertions in a computer memory we must use a complete lattice $A(\xi, 1, 1, 1, 1)$ such that the smaller algebras $\bar{A} = \rho(\xi) = \lambda x. false, \lambda x. true, \lambda x. \rho(\xi), \lambda x. \rho(\xi)$ and $A(\xi, 1, 1, 1, 1)$ be isomorphic. Let $\gamma \in \gamma(\xi)$ be the corresponding lattice isomorphism. Let $\alpha \in (A \times A)$ be $\gamma^{-1} \circ \rho$. $\alpha$ is the representation of the least upper approximation of the assertion $PrA$ whereas $\gamma(\xi)$ provides the meaning of $\xi A$. The connection $\alpha, \gamma$ between $A$ and $A$ has the following property:

**Definition 5.3.0.1**

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

- $\alpha \in (L_1 \times L_2)$ is isotope
- $\gamma \in (L_2 \times L_1)$ is isotope
- $\psi x, L_1, \psi y, L_2, \{x \in \xi y(x)\} \Rightarrow \alpha(x) \in \xi y(x)$

(Contrary to Scott[72]'s definition, $L_1$ and $L_2$ are not required to be continuous lattices and $\alpha, \gamma$ need not be continuous).

**Theorem 5.3.0.2**

If $\rho$ is an upper closure operator on $A$, the image $\gamma(A)$ of $A(\xi, 1, 1, 1, 1)$ through the lattice isomorphism $\gamma$ is equal to $\rho(A) = \rho(\lambda, false), \lambda, true$.

$\lambda, \rho(\lambda, \rho(\xi), \lambda, true), \lambda, \rho(\lambda, false), \lambda, true$ and $\alpha = \gamma^{-1} \circ \rho$ then

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

Reciprocally the approximation process can be defined by the lattice $A(\xi, 1, 1, 1, 1)$ and a pair of adjoined functions. Such a pair $\langle \alpha, \gamma \rangle$ defines a Galois connection between $A$ and the dual of $A$.

**Definition 5.3.0.3**

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

- $\alpha$ is antitone $\{x_1, y_1 \in E_1, \{x_1 \in \alpha(x_1) \} \Rightarrow \{\alpha(x_1) \in \gamma(x_1) \} \}$
- $\gamma$ is antitone $\{x_1, y_1 \in E_1, \{y_1 \in \gamma(y_1) \} \Rightarrow \{y_1 \in \alpha(y_1) \} \}$
- $\psi x_1, L_1, \{x_1 \in \xi y_1 \}$
- $\psi y_1, L_2, \{y_1 \in \xi \alpha(x_1) \}$

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

$\forall x_1:E_1, \psi y_1, L_2, \{x_1 \in \xi y_1 \} \Rightarrow \{\alpha(x_1) \in \gamma(x_1) \}$

(Birkhoff[67]), hence we have:

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

**Corollary 5.3.0.5**

Let $L_1(E_1)$ and $L_2(E_2)$ be posets and $\alpha \in (L_1 \times L_2)$, $\gamma \in (L_2 \times L_1)$ be adjoined functions:

- $\alpha$ is an upper closure operator on $L_1$, $\alpha, \gamma$ is a lower closure operator on $L_1 \times L_2$, isotonous, reductive (of $\alpha, \gamma$ on $x, x$), and idempotent.
- $L_1(E_1) = L_2(E_2)$ and $L_2(E_2) = L_1(E_1)$ are complete lattices then:

$\alpha, \gamma$ are complete lattices, $\alpha \in (L_1 \times L_2)$ is isotonous

Moreover if $L_1(E_1)$ and $L_2(E_2)$ are complete lattices then:

- $\alpha, \gamma$ are complete lattices, $\alpha \in (L_1 \times L_2)$ is isotonous

- Each function in the pair $\alpha, \gamma$ of adjoined functions uniquely determines the other, more precisely:

$\{\lambda \alpha \gamma \xi \alpha \gamma \xi \alpha \gamma \xi \alpha \gamma \xi \alpha \gamma \xi \alpha \gamma \xi \}$

$\{\lambda \alpha \gamma \xi \alpha \gamma \xi \alpha \gamma \xi \alpha \gamma \xi \}$

Also $\alpha$ is a complete join morphism, $\alpha: L_1 \cong L_2$, $\alpha$ is a complete meet morphism, $\gamma(T_2) = T_1$

In complement we will need the following:

**Theorem 5.3.0.6**

Let $L_1(E_1, L_1, E_1, L_1, E_1)$ and $L_2(E_2, L_2, E_2, L_2, E_2)$ be complete lattices and $\alpha \in (L_1 \times L_2)$, $\gamma \in (L_2 \times L_1)$ be adjoined functions:

- $\alpha$ is onto (surjective) if and only if $\gamma$ is one-to-one (injective) and if and only if $\alpha(\gamma \xi) = \gamma \xi L_1(T_1(y))$
- $\gamma$ is onto (surjective) if and only if $\alpha(\gamma \xi) = \gamma \xi L_2(T_2(y))$
We use the notation $L_1 \lessdot a \lessdot y \lessdot L_2$ to state that $L_1$ and $L_2$ are connected by the pair $<a, y>$ of adjoined 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 $L_1 \lessdot a \lessdot y \lessdot L_2$. If $(L_1, \lessdot a \lessdot y \lessdot L_2)$ and assume that the adjoined $y(a)$ 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 $A(E, I, T, F, P)$ representing approximate assertions and to informally describe the meaning of its elements (e.g., constant propagation, Kildall[73], Kamin[81]). Hence the function $y \in (A \rightarrow A)$ is implicit.

It is often the case that $A$ is only assumed to be a complete semi-lattice $A(E, I, T, F, P)$ (or dually meet-semi-lattice for some authors) but since an infimum is adjoined to $A$, it is in fact a complete lattice (even when the meet-operation is not used). What is called meet is not $\sqcap$ (e.g., Wegbreit[75]).

When $y \in (A \rightarrow A)$ is isotone 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 $y \in (A \rightarrow A)$ 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 $y$ can be revised as follows:

**THEOREM 5.3.0.7**

Let $A(E, I, T, F, P)$ be a complete lattice and $y \in (A \rightarrow A)$ be a complete meet-morphism. Let $\sigma \in (A \rightarrow A)$ be $\lambda x. y(x) = y(y(x))$, $\bar{A} = \sigma(A)$, $\bar{y} = y(\bar{A})$.

$y$ is a lower closure operator on $A$.

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

**THEOREM 6.1.0.1**

Let $L(E, I, T, F, P)$ be a complete lattice and $f \in (L \rightarrow L)$.

1. Let $\sigma = (L \rightarrow L) \rightarrow (L \rightarrow L)$ be $\lambda f. \lambda x. y(f(y(x)) = y(f(x))$.
2. $\sigma$ is the least isotone operator on $L$ greater than or equal to $f$.
3. Let $\tau = (L \rightarrow L) \rightarrow (L \rightarrow L)$ be $\lambda f. \lambda x. y(f(x))$.
4. $\tau$ is the least isotone operator on $L$ greater than or equal to $f$.
5. Let $\delta \in (L \rightarrow L) \rightarrow (L \rightarrow L)$ be $\lambda x. y(f(x))$ where $\delta(f)(x)$ is the limit of the increasing and ultimately constant sequence $\{x_\delta\}$ such that $x_\delta = x, y$ for every ordinal $\delta$, $x_\delta = y(x_\delta)$, and for every limit ordinal $\delta, x_\delta = \lim_{\alpha < \delta} x_\delta$.
6. $\delta_{\text{iso}}(f) = \delta_{\text{iso}}(f)$.
7. $\delta_{\text{iso}}(f) = \delta_{\text{iso}}(f)$.
8. $\delta_{\text{iso}}(f)$ is the least closure operator greater than or equal to $f$ and $\delta_{\text{iso}}(f)(L)$ 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 (Ored43). However, the space of approximate assertions can be designed by successive approximations using the following composition of upper closure operators:

**THEOREM 6.2.0.1**

Let $L(E, I, T, F, P)$ be a complete lattice, $\rho$ an upper closure operator on $L$ and $\eta$ be an upper closure operator on $\rho(L)$. Then $\eta \circ \rho$ is an upper closure op-
$\sigma$ is an upper closure operator on $A_m$ and an assertion $P \leq A_m$ does not state relationships among the program variables if and only if $\sigma(P) = P$. The approximate assertions on each individual program variable $x_i$ are next defined using an upper closure operator $\rho^j$ on

$A_m$. The induced closure operator $\rho$ on $\sigma(A_m)$ is defined by $\rho(P) = \lambda(x_1, \ldots, x_m), \bigwedge_{j=1}^{m} \rho^j(P_j)(x_j)$ where $P \in \sigma(A_m)$ is (necessarily) of the form $P = \lambda(x_1, \ldots, x_m), \bigwedge_{j=1}^{m} P_j(x_j)$. It follows from theorem 6.2.1.4 that the composition

$\rho \circ \sigma = \lambda P \in \sigma(A_m), \{ \lambda(x_1, \ldots, x_m) \in \rho[\sigma(P)](x_j) \}$

is an upper closure operator on $A_m$.

End of Example.

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

Considering the equivalence relation $[P]$ induced by an upper closure operator $\rho$ on $A$ and defined as $P \equiv Q$ if and only if $\rho(P) = \rho(Q)$, 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 5.4.0.2) not all equivalence relations are acceptable.

DEFINITION 6.3.0.1

Let $(L, \cup, \cap, 0, 1)$ be a complete lattice. A binary relation $\theta$ on $L$ is a complete join-congruence relation if and only if:

- The domain $\rho$ is an upper closure operator on $A$ and defined as $P \equiv Q$ if and only if $\rho(P) = \rho(Q)$.

End of Example.

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

A reflexive and symmetric binary relation $\theta$ on a complete lattice $(L, \cup, \cap, 0, 1)$ is a join-congruence relation if the following three properties are satisfied for $x, y, z, t \in L$:

- $(1) - x \theta y \Rightarrow \forall u \in L : u \cup x \theta(y \cup u) \wedge u \cup y \theta(x \cup u)$
- $(2) - x \theta y \wedge y \theta z \Rightarrow x \theta(z \cup y)$
- $(3) - x \theta y \wedge x \neq y \Rightarrow \{x, y \cup t \cup (y \cap t) \}$

Example 6.3.0.5

Let $V$ be a non-empty set of integers included between two bounds $\min(V)$ and $\max(V)$, $\forall x \in V : x \in [\min(V), \max(V)]$ and $V = \{i \in Z : -\infty < i < \infty \}$. The binary relation $\theta$ defined on $A = (V \times \mathbb{Z})$ by:

$\rho = \{ (x \in \rho) : x \in \rho \}$

is isomorphic to $\rho(A)$ where $\rho$ is the upper closure operator induced by $\theta$.

In conjunction with 6.2.0.2, $\rho$ can be used for static analysis of the ranges of values of numerical variables.
THEOREM 6.4.0.2

(1) *Let I be a principal ideal and J be a dual semi-ideal of a complete lattice \( L \subseteq \{1, x, y, z, 0, 1\} \). If\n
*In J is monoid then In J 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\n
*\( I = \{x \in L : x \leq j\}\) and \( \{x \in C : y \leq x\} \subseteq J.\)*

Example 6.4.0.4

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

![Lattice Diagram]

(where \( 1, 0, 1, x, y, z, 0, 1 \) respectively stand for \( \lambda x, \downarrow, \uparrow, \lambda x \leq 0, \lambda x > 0, \lambda x > 0, \lambda x = 0, \lambda x = 0, \lambda x = 0 \). A further approximation can be defined by the following family of principal ideals:

\[
I_1, I_2, I_3, I_4, I_5, I_6
\]

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

\[
\begin{align*}
I_1 & \subseteq I_2 \\
I_2 & \subseteq I_3 \\
I_3 & \subseteq I_4 \\
I_4 & \subseteq I_5 \\
I_5 & \subseteq I_6
\end{align*}
\]

THEOREM 6.4.0.3

Let \( \{I_v \in \Delta \} \) be a family of principal ideals of the complete lattice \( L \subseteq \{1, x, y, z, 0, 1\} \) containing \( L \). Then

\[
\lambda x \in L(n_{I_v} : i \in \Delta \land x \in I_v)
\]

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

7. DESIGN OF THE APPROXIMATE PREDICATE TRANSFORMER
    INDUCED BY A SPACE OF APPROXIMATE ASSERTIONS

In addition to \( A \) and \( \gamma \) the specification of a program analysis framework also includes the choice of an approximate predicate transformer \( Tc(L \rightarrow (A \rightarrow A)) \) (or a monoid of maps on \( A \) plus a rule for associating maps to program statements (e.g., Rosenblum78)). 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 \( (V,A,\gamma) \) the minimal assertion which is invariant at point \( i \) of a program \( \pi \) with entry specification \( \phi \) was defined as:

\[
P_i = \bigvee_{P \in \text{Path}(i)} \phi(p)
\]

Therefore the minimal approximate invariant assertion is the least upper approximation of \( P_i \) in \( \Gamma \) that is.

\[
\rho(P_i) = \rho(\bigvee_{P \in \text{Path}(i)} \phi(p))
\]

Even when \( \text{Path}(i) \) is a finite set of finite paths the evaluation of \( \phi(p) \) is hard to machine-implementable since for each path \( p = a_1 \ldots a_n \) the computation sequence \( X_0 = 0, X_1 = \gamma(C(a_1))X_0, \ldots, X_n = \gamma(C(a_n)X_{n-1}) \) does not necessarily only involve elements of \( A \) and \( (A \rightarrow A) \). Therefore using \( \bar{X} = X \) and \( \gamma(=) \) we can represent \( \gamma(C(a_n)X_{n-1}) \) instead of \( X_0, \ldots, X_n \) which leads to the expression:

\[
Q_i = \rho(\bigvee_{P \in \text{Path}(i)} \phi(p))
\]

The choice of \( \Gamma \) and \( \phi \) is correct if and only if \( Q_i \) is an upper approximation of \( P_i \) in \( \Gamma \) that is if and only if:

\[
\bigvee_{P \in \text{Path}(i)} \phi(p) \Rightarrow \rho(\bigvee_{P \in \text{Path}(i)} \phi(p))
\]

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

DEFINITION 7.1.0.1

(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 \( A \rightarrow \rho(A) \) if and only if for all \( \phi \in \Gamma \) and program \( \pi \) we have:\n
\[
\rho(p, \phi) = \rho(\phi) = \phi
\]

and program \( \pi \) we have:

\[
\rho(p, \phi) = \rho(\phi) = \phi
\]

(2) *Similarly if \( A \rightarrow \phi, \gamma \rightarrow \phi \), \( Tc(L \rightarrow (A \rightarrow A)) \) is said to be a correct upper approximation of \( Tc(L \rightarrow (A \rightarrow A)) \) in \( A \rightarrow \rho(A) \) if and only if \( \forall \phi, \forall \psi : \phi \Rightarrow \gamma(\phi), \forall \psi, \forall \phi : \rho(\phi) = \gamma(\phi), \rho(\phi) = \gamma(\rho(\phi)) \in \rho(\phi) \),\n
(1.e., \( \rho(\phi) = \gamma(\rho(\phi)) = \gamma(\rho(\phi)) \)))

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

End of Example.
THEOREM 7.1.0.2

approximate predicate transformer which happens to be isotone. This property also explains the fact
According to theorem 7.2.0.4 the best upper approximation of $ap(\lambda <x,y>, (x<y))$ in $A$ is $t = a \circ ap(\lambda <x,y>, (x<y)) \circ y$. If $P \circ A$ equals $<1,1>$ then $t(P) = <1,1>$ else $P = [a,b], [c,d]$ where $a \circ b$ and $c \circ d$ in which case $t(P) = q(Q)$ where $Q = \lambda <x,y>, (x<y) \circ a \circ b \circ c \circ d \circ x \circ y$.

7.2.0.6.3. Justifying the Data Flow Equations of "Available Expressions"

Let $E$ be the set of expressions. The set $\text{avail}(t)$ of expressions which are available at exit of a path $t \in T$ is defined by $\text{avail}(t) = \emptyset$ and $\text{avail}(x)$.
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:

A more precise approximation (example 6.3.0.5) consists in approximating the characteristic set of solution to program-wide analysis problems since whenever some $t(S)$ is not a complete join-morphism $\text{MOP}_t(t,\emptyset)$ can be strictly better than $\text{MOP}_t(t,\emptyset)$. When $A$ satisfies the ascending chain condition $\text{MOP}_t(t,\emptyset)$ is computable, which is not necessarily the case of $\text{MOP}_t(t,\emptyset)$. 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 $Y$.

In our opinion the above argument is not entire-
10. COMBINATION OF PROGRAM ANALYSIS FRAMEWORKS

The ideal method in order to construct a program analyser (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 analyser 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 \(A_1 \supset Y_1 \supset A_2 \supset Y_2 \supset A\) and \(t_1, t_2\) are correct upper approximations of \(\gamma\) in \(A_1, A_2\). Then the direct product \((A, t, Y)\) of \((A_1, t_1, \gamma_1)\) and \((A_2, t_2, \gamma_2)\) is defined as:

\[
\begin{align*}
A &= A_1 \times A_2, \\
t &= t_1 \times t_2, \\
Y &= \lambda x. (y_1(x) \times y_2(x))
\end{align*}
\]
Example 10.1.0.3

\[ A_1 = \text{ev} \quad A_2 = \text{od} \]

\[ Y_1(t) = \text{true}, \quad Y_2(t) = \text{false}, \quad Y_3(t) = \text{false}, \quad Y_4(t) = \text{true}, \quad Y_5(t) = \text{true}, \quad Y_6(t) = \text{true} \]

The following program (Manna 74,p.179) computes \( y_1 = x_1^2 \) (with the convention \( 0^0 = 1 \)) for every integer \( x_1 \) and natural number \( x_2 \):

1. \( Y_1, Y_2, Y_3 \): if \( x_1 = 0 \) then \( y_2 = 0 \) else \( y_2 = 1 \)
2. \( Y_4 \): if \( x_1 = 0 \) then \( y_2 = 0 \) else \( y_2 = 1 \)
3. \( Y_5 \): \( y_2 = x_1 \)
4. \( Y_6 \): \( y_2 = x_1 \)

The fixpoint analysis with entry condition \( \lambda(y_1, y_2, y_3, x_1, x_2) \) using \( A_1 \) leads to the following result for the variable \( y_2 \):

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>0</td>
</tr>
</tbody>
</table>

The fixpoint analysis using \( A_2 \) leads to the following result for the variable \( y_2 \):

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>ev</td>
</tr>
</tbody>
</table>

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 \( (A_1 \times A_2)_R \) and the corresponding optimal approximate predicate transformer (which takes account of the rule \( f_{\text{od}} = f_{\text{ev}} \)) we get:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_2</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>0</td>
</tr>
</tbody>
</table>

End of Example.

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 \( L_1(E_1, T_1, \text{od}, \text{ev}) \) and \( L_2(E_2, T_2, \text{od}, \text{ev}) \) are complete lattices we can define the disjoint sum \( L_1 \cup L_2 \) as \( L_1 \cup L_2 = \{ i, T \} \) with ordering \( x \leq y \iff (x \leq y_1) \vee (y_2 \leq y) \) or \( (x \leq y_1) \wedge (y_2 \leq y) \). The meaning of elements of \( L_1 \cup L_2 \) can be defined as \( Y_1(y) \cdot Y_2(y) \cdot \text{true} \). Even when \( Y_1 \) and \( Y_2 \) 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 \( \gamma(L_1 \cup L_2) \) must be completed using theorem 5.2.0.4. Then it turns out that the least Moore family containing \( Y(L_1 \cup L_2) \) is equal to \( Y(L_1 \cup L_2) \) (as defined in theorem 10.1.0.2). Therefore the use of disjoint sums amounts to the use of reduced products.

End of Remark.

10.2 Reduced Cardinal Power of Program Analysis Frameworks

The cardinal power \( L_1^L \) with base \( L_1(L_2, T_1, \text{od}, \text{ev}) \) and exponent \( L_2(T_1, \text{od}, \text{ev}) \) (henceforth noted \( \text{foc}(L_1, L_2)(E_1, T_1, \text{od}, \text{ev}) \)) is the set of all isomorphism maps from \( L_1 \) to \( L_2 \) with \( \text{foc} \) and only if \( \text{foc}(L_3) \). Two program analysis frameworks \( (A_1, E_1, Y_1) \) and \( (A_2, E_2, Y_2) \) can be combined by letting \( y \in \text{foc}(L_1 \cup L_2) \) mean that for all \( x \in A_1 \), \( y(x) \) holds whenever \( Y_1(x) \) holds.

THEOREM 10.2.0.1

The reduced cardinal power with base \( (A_2, T_2, Y_2) \) and exponent \( (A_1, T_1, Y_1) \) is \( (A_1 \times A_2) \) where \( A = \foc(A_1 \times A_2), \text{foc}(A_1 \times A_2) = \foc(A_1) \times \foc(A_2) \) and \( \lambda_{\alpha}([\alpha_1 \in \text{foc}(A_1), \alpha_2 \in \text{foc}(A_2)]) = \lambda_{\alpha}[\alpha_1 \in \text{foc}(A_1), \alpha_2 \in \text{foc}(A_2)] = \lambda_{\alpha}[\alpha_1 \in \text{foc}(A_1), \alpha_2 \in \text{foc}(A_2)] = \lambda_{\alpha}[\alpha_1 \in \text{foc}(A_1), \alpha_2 \in \text{foc}(A_2)] = \lambda_{\alpha}[\alpha_1 \in \text{foc}(A_1) \times \alpha_2]. \]

Example 10.2.0.2

\[ Y_1(t_1) = Y_2(t_1) = \lambda(b, x), \text{false}, \quad Y_1(t_2) = Y_2(t_2) = \lambda(b, x), \text{true}, \]

The analysis of the program:

\{ 1 \} \quad \text{while} \quad \text{do} \quad \text{end}
using the reduced cardinal product of $A_1$ and $A_2$ yields no information since no relationship can be discovered between $b$ and $x$.

Following theorem 10.2.0.1 we determine that if $y \in (A_1 \rightarrow A_2)$ then $y(y_1 \in \alpha_1(y) \land \alpha_2(y_1 \in \alpha_2(y))$. Therefore $\delta(y) = \delta$ where $\delta(x_1) = x_2$, $\delta(x_1) = x_2$, $\delta(x_1) = x_2$. It follows that $\alpha_1(y_1 \in \alpha_2(y_1) \land \alpha_2(y_1 \in \alpha_2(y))$ is isomorphic to $(t, f) \rightarrow A_1$ for $A_2 \times A_2$.

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

\[
\begin{align*}
g_1 &= \lambda b. t \cdot f \cdot b \cdot t + e \cdot s \cdot l_2 \cdot f \cdot g \\
g_2 &= \lambda b. t \cdot f \cdot b \cdot t \cdot g \cdot (t) \cdot e \cdot s \cdot l_2 \cdot f \\
g_3 &= \lambda b. \text{decr}(g \cdot (b)) \\
g_4 &= \lambda b. t \cdot f \cdot b \cdot t \cdot \text{decr}((t) \cdot f \cdot b \cdot t \cdot g \cdot (f) \cdot T_2) \\
g_5 &= \lambda b. t \cdot f \cdot b \cdot f \cdot t \cdot g \cdot (f) \cdot e \cdot s \cdot l_2 \cdot f \\
\end{align*}
\]

where $\text{decr}(l_2) = l_2$, $\text{decr}(0) = \text{decr}(-) = \text{decr}(2) = -$, $\text{decr}(+) = -$, $\text{decr}(0) = \text{decr}(e) = \text{decr}(T_2) = T_2$.

The iterative resolution of this system of equations starting from the initial $\lambda b. \tau$ yields $y(y_1) = Y(y_1) = \lambda(b, x) \cdot (b \land x = 0)$, $y(y_2) = \lambda(b, x) \cdot (b \land x = 0)$, $y(g) = \lambda(b, x) \cdot (b \land x = 0)$, $Y(y_3) = \lambda(b, x) \cdot (\neg b \land x = 0)$.

End of Example.

11. REFERENCES


Cousot P. & Cousot R. [1977b], Static determination of dynamic properties of recursive procedures, IFIP WG6.2 Working Conf. on Formal Description of Programming Concepts, St.-Andrews, Canada.


Ore O. [1945], Combinatorial closure relations, Ann. of Math., 44 (1945), 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.

Shamir A. & Waks W.W. [1977], Data types as objects.