Conference Record
of the
FOURTH ACM SYMPOSIUM ON
PRINCIPLES OF PROGRAMMING LANGUAGES

Papers Presented at the Symposium
Los Angeles, California
January 17-19, 1977

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

A program denotes computations in some universe of objects. Abstract interpretation of programs consists in using that denotation to describe computations in another universe of abstract objects, so that the results of abstract execution give some informations on the actual computations. An intuitive example (which we borrow from Sintzoff [72]) is the rule of signs. The text \(-1515 \times 17\) may be understood to denote computations on the abstract universe \((+), (-), (2)\) where the semantics of arithmetic operators is defined by the rule of signs. The abstract execution \(-1515 \times 17\) \(\Rightarrow -(-) \times (+) \Rightarrow (-) \times (+) \Rightarrow (-)\), proves that \(-1515 \times 17\) is a negative number. Abstract interpretation is concerned by a particular underlying structure of the usual universe of computations (the sign, in our example). It gives a summary of some facets of the actual executions of a program. In general this summary is simple to obtain but inaccurate (e.g., \(-1515 + 17 \Rightarrow -(+ + (+ \Rightarrow (-) + (+) \Rightarrow (2))\)). Despite its fundamentally incomplete results abstract interpretation allows the programmer or the compiler to answer questions which do not need full knowledge of program executions or which tolerate an imprecise answer, (e.g., partial correctness proofs of programs ignoring the termination problems, type checking, program optimizations which are not carried in the absence of certainty about their feasibility, ...).

Abstract program properties are modeled by a complete semilattice, Birkhoff[61]. Elementary program constructs are locally interpreted by order preserving functions which are used to associate a system of recursive equations with a program. The program global properties are then defined as one of the extreme fixpoints of that system, Tarski[55]. The abstraction process is defined in section 6. It is shown that the program properties obtained by an abstract interpretation of a program are consistent with those obtained by a more refined interpretation of that program. In particular, an abstract interpretation may be shown to be consistent with the formal semantics of the language. Levels of abstraction are formalized by showing that consistent abstract interpretations form a lattice (section 7). Section 8 gives a constructive definition of abstract properties of programs based on constructive definitions of fixpoints. It shows that various classical algorithms such as Kildall [73], Wegbreit [75] compute program properties as limits of finite Kleene[52]'s sequences. Section 9 introduces finite fixpoint approximation methods to be used when Kleene's sequences are infinite, Cousot[76]. They are shown to be consistent with the abstraction process. Practical examples illustrate the various sections. The conclusion points out that abstract interpretation of programs is a unified approach to apparently unrelated program analysis techniques.

2. Summary

We will use finite flowcharts as a language inde-
3.2 Semantics of Programs

This section develops a simple "mathematical semantics" of programs, in the style of Scott and Strachey [71].

- If $S$ is a set we denote $S^0$ the complete lattice obtained from $S$ by adjoining $\{\bot, \top\}$ to it, and imposing the ordering $\forall x, y \in S^0$, $x \leq y$ if and only if $x \leq y$ for all $x \in S$.

- The semantic domain "Values" is a complete lattice which is the sum of the lattice $\text{Bool} = \{\text{true}, \text{false}\}$ and some other primitive domains.

- Environments are used to hold the bindings of identifiers to their values:

  $$\text{Env} = \text{Ident} \rightarrow \text{Values}$$

  We assume that the meaning of an expression $e$ in the environment $e \in \text{Env}$ is given by $\text{val} = e$, which implies that:

  $$\text{val} : e \rightarrow [\text{Env} \rightarrow \text{Values}]$$

  In particular the projection $\text{val} |_{\text{Bexpr}}$ of the function $\text{val}$ in domain $\text{Bexpr}$ has the functionality:

  $$\text{val} |_{\text{Bexpr}} : e \rightarrow [\text{Env} \rightarrow \text{Booll}]$$

  The state set "States" consists of the set of all information configurations that can occur during computations:

  $$\forall e \in \text{States}, s = \langle e(s), \text{env}(s) \rangle$$

  A state $s$ consists of a control state $e(s)$ and an environment $\text{env}(s)$, such that:

  $$\forall e \in \text{States}, s = \langle e(s), \text{env}(s) \rangle$$

  - We use a continuous conditional function $\text{cond}(b, e_1, e_2)$ equal to $b$, $e_1$, $e_2$ or $\bot$ respectively as the value of $b$ is true, false or $\bot$. We also use $\text{if } b \text{ then } e_1 \text{ else } e_2$ to denote $\text{cond}(b, e_1, e_2)$.

  - If $e \in \text{Env}$, $v \in \text{Values}$, $x \in \text{Ident}$ then $e[v/x] = \lambda y. \text{cond}(y = x, v, e(y))$.

  - The state transition function defines for each state a next state (we consider deterministic programs):

    $$\text{n-state} : e(s) \rightarrow \text{States}$$

    $$\text{n-state}(s) =$$

    $$\ldots$$
- A "computation sequence" with initial state $i_0 \in I$-states is the sequence:
  
  $$ s_n = n\text{-state}_0(i_0) \quad \text{for } n = 0, 1, \ldots $$
  
  where $f^n$ is the identity function and $f^{n+1} = f \circ f^n$.

- The initial to final state transition function:

Since the equation $C_v(r) = n\text{-context}(r, C_v)$ must be valid for each arc, $C_v$ is a solution to the system of "forward" equations:

$$ C_v = F\text{-cont}(C_v) $$

where $F\text{-cont} : \text{Context-Vectors} \to \text{Context-Vectors}$ is defined by:

$$ F\text{-cont}(C_v) = n\text{-context}(r, C_v) $$
This implies that A-Cont is in fact a complete lattice, but we need only one of the two join and meet operations. The set of context vectors is defined by \( A-\text{Cont} = \text{Arcs}^2 \rightarrow A-\text{Cont} \).

Whatever \((Cv', Cv'') \in A-\text{Cont}^2\) may be, we define:

\[
Cv' \preceq Cv'' = \lambda r . Cv'(r) \circ Cv''(r)
\]

\[
\sim = \lambda r . \top \quad \text{and} \quad \perp = \lambda r . \bot
\]

\(<A-\text{Cont}, \preceq, \sim, \perp, \top>\) can be shown to be a complete lattice. The function:

\[
\text{Int} : \text{Arcs}^2 \times A-\text{Cont} \rightarrow A-\text{Cont}
\]

defines the interpretation of basic instructions. If \(\{Cq[q \in a-\text{prod}(n)]\}\) is the set of input contexts of node \(n\), then the output context on exit arc \(r\) of \(n\) \((r \in a-\text{succ}(n))\) is equal to \(\text{Int}(r, C)\). \(\text{Int}\) is supposed to be order-preserving:

\[
\forall a \in \text{Arcs}, \forall (Cv', Cv'') \in A-\text{Cont}^2,
\]

\[
(Cv' \preceq Cv'') \implies (\text{Int}(a, Cv') \preceq \text{Int}(a, Cv''))
\]

The local interpretation of elementary program constructs which is defined by \(\text{Int}\) is used to associate a system of equations with the program. We define:

\[
\text{Int} : A-\text{Cont} \rightarrow A-\text{Cont} \mid \text{Int}(Cv) = \lambda r . \text{Int}(r, Cv)
\]

It is easy to show that \(\text{Int}\) is order-preserving. Hence it has fixpoints, Tarski[55]. Therefore the context vector resulting from the abstract interpretation \(I\) of program \(P\), which defines the global properties of \(P\), may be chosen to be one of the extreme solutions to the system of equations \(Cv = \text{Int}(Cv)\).

5.2 Typeology of Abstract Interpretations

The restriction that "A-Cont" must be a complete semi-lattice is not drastic since Mac Neille[37] showed that any ordered set \(S\) can be embedded in a complete lattice so that inclusion is preserved, together with all greatest lower bounds and least upper bounds existing in \(S\). Hence in practice the set of abstract contexts will be a lattice, which can be considered as a join (\(\cup\)) semi-lattice or a meet (\(\cap\)) semi-lattice, thus giving rise to two dual abstract interpretations.

It is a pure coincidence that in most examples (see 5.3.2) the \(n\) or \(u\) operator represents the effect of path converging. The real need for this operator is to define completeness which ensures \(\text{Int}\) to have extreme fixpoints (see 8.4).

The result of an abstract interpretation was defined as a solution to forward (\(>\)) equations: the output contexts on exit arcs of node \(n\) are defined as a function of the input contexts on entry arcs of node \(n\). One can as well consider a system of backward (\(<\)) equations: a context may be related to its successors. Both systems \((\leq, \geq)\) may also be used.

Examples:

Kildal[73] uses \((n, \rightarrow, +)\), Wegbreit[75] uses \((u, \rightarrow, \top)\). Tenenbaum[74] uses both \((u, \rightarrow, +)\) and \((n, \rightarrow, +)\).

5.3 Examples

5.3.1 Static Semantics of Programs

The static semantics of programs we defined in section 4 is an abstract interpretation:

\[
\lambda s = \langle \text{Contexts}, u, \leq, \text{Env}, \emptyset, n-\text{context} >
\]

where Contexts, \(u\), \(\leq\), \(\text{Env}\), \(\emptyset\), \(n-\text{context}\), Context-Vectors, \(u\), \(\leq\), \(\text{P-Cont}\) respectively correspond to \(A-\text{Cont}\), \(\leq\), \(\top\), \(\perp\), \(\text{Int}\), \(A-\text{Cont}\), \(\leq\), \(\perp\), \(\text{Int}\).

5.3.2 Data Flow Analysis

Data flow analysis problems (see references in Ullman[75]) may be formalized as abstract interpretations of programs.

"Available expressions" give a classical example. An expression is available on arc \(r\), if whenever control reaches \(r\), the value of the expression has been previously computed, and since the last computation of the expression, no argument of the expression has had its value changed.

Let \(\text{Expr}_P\) be the set of expressions occurring in a program \(P\). Abstract contexts will be sets of available expressions, represented by boolean vectors:

\[
\text{B-vec} : \text{Expr}_P \rightarrow \{\text{true}, \text{false}\}
\]

B-vec is clearly a complete boolean lattice. The interpretation of basic nodes is defined by:

\[
\text{avail}(r, By) = \begin{cases} 
\text{false} & \text{if } n \text{ is not in } \text{case } n \text{ in } \\
\text{true} & \text{otherwise}
\end{cases}
\]

Assignments, Tests, Functions:

\[
\lambda e. (\text{generated}(n)(e)) \text{ or } (\text{and } By(p)(e) \text{ and transparent}(n)(e))
\]

which is available on entry arc. An expression...
The determination of available expressions, back-
dominators, intervals, ... requires a forward sys-
tem of equations. Some global flow problems, nota-
Instead of the global hypothesis 6.0 we will use
the following local hypothesis on the concrete and
abstract interpretations of primitive language cons-
where $\text{n-pred}$ defines Floyd[67]'s strongest post condition:

$$\text{n-pred}(r, P_v) = \begin{cases} \text{let}(n \text{ be origin}(r), \langle p \text{ be a-pred(origin}(r))\rangle \text{ within case } n \text{ in} \\ \text{Entries} \Rightarrow (\forall x \in \text{Ident}, x = \text{Values}) \\ \text{Junctions} \Rightarrow \text{or} (P_v(q)) \\ \text{Tests} \Rightarrow \text{case } r \text{ in} \begin{align*} (\text{a-succ-t}(n)) & \Rightarrow P_v(n) \\ \text{esac} \end{align*} \text{and not test}(n) \end{cases}$$

The relation $\equiv$ on abstract interpretations defined by:

$$\{I \equiv I'\} \iff \{(I \leq I') \text{ and } (I' \leq I)\}$$

is an equivalence relation. We have:

$$\{I \equiv (\beta)I'\} \iff \{\beta \text{ is an isomorphism between the algebras } I \text{ and } I'\}$$

The proof gives some insight in the abstraction process:

$$\begin{align*} 1 \cdot \{I \equiv (\beta)I'\} & \Rightarrow \{(I \leq (\beta, \beta^{-1})I') \text{ and } \end{align*}$$

2 - reciprocally,

If $I \leq (\alpha, \gamma)I'$, let $\equiv (\alpha_i)$ be the equivalence relation defined on $I$ (appropriately speaking). On the
The performance of programs may be analyzed by deriving for each program point the final value of an imaginary counter which is incremented each time control goes through that point.

Let $A\text{-Cont}$ be the lattice $\mathbb{R}^+$ of positive real numbers augmented by the upper bound $\infty$, with natural ordering $\leq$. The abstract interpretation:

$$I_p = (\mathbb{R}^+, \max, \leq, 0, \infty, \text{Kir})$$

may be used to derive the mean values of the counters using Kirchhoff's law of conservation of flow:

$$\text{Kir}(r, Cv) =$$

\[
\text{let } n \text{ be origin}(r) \text{ within} \\
\text{case } n \text{ in} \\
\quad \begin{cases} 
\text{Entries} & \rightarrow 1 \ {\text{(unique entry node)}} \\
\text{Junctions} \cup \text{Assignments} & \rightarrow \sum_{p \in a-pred(n)} Cv(p) \\
\text{Tests} & \rightarrow \begin{cases} 
\text{case } r \text{ in} \\
\{a\text{-succ-f}(n)\} & \rightarrow Cv(a\text{-pred}(n)) * \\
\{a\text{-succ-t}(n)\} & \rightarrow \text{Prob}(\text{test}(n) = \text{true}) \\
\text{esac} & \text{esac} 
\end{cases}
\end{cases}
\]

The main difficulty is to obtain the probability $\text{Prob}(\text{test}(n) = \text{true})$ of taking the true path at a test node $n$. Suppose the values of these probabilities can be determined (from hypothesis on the input data).

For fixed probabilities, the function $\text{Kir}$ is clearly continuous (although it is not a complete morphism) since

$$\text{if } Cv_0 \leq Cv_1 \leq \ldots \leq Cv_n \leq \ldots$$

then

$$\lim_{n \to \infty} \sum_{1=0}^{\infty} p_{c=a-pred(n)} \frac{Cv_i(p)}{\sum_{1=0}^{\infty} (\max(Cv_i)(p))}$$

and

$$\max_{1 \leq i} (n_i \cdot q) = (\max_{1 \leq i} (n_i)) \cdot q.$$ 

The least fixpoint of $\text{Kir}$ is the limit of Kleene's

9. Fixpoints Approximation Methods

When the extreme fixpoints of the system of equations established for an abstract interpretation $I$ of a program $P$ cannot be computed in finitely many steps, they can be approximated. A more abstract interpretation $\tilde{I}$ ($1 \leq I$) may be used for that purpose (e.g., Ternenbaum's theorem). It is often better to make approximations in $I$, for example by "accelerating the convergence" of Kleene's sequences.

9.1 Finite Iterative and Increasing Approximation of the Least Fixpoint Starting from a Lower Bound

Let $I = \langle A\text{-Cont}, \preceq, \leq, I, \top, \text{Int} \rangle$ be an interpretation of $P$. When the least fixpoint $Cv$ of $\text{Int}$ is unreachable, we look for an upper bound $UB$ of $Cv$, since according to the correctness requirement $6.0$, $Cv \preceq \gamma(Cv)$ and $Cv \preceq UB$ implies $Cv \preceq \gamma(UB)$.

9.1.1 Increasing Approximation Sequence

Let $\sim \text{Int} : A\text{-Cont} \rightarrow A\text{-Cont}$ be such that:

9.1.1.1 $(\forall n \geq 0, \zeta = \sim \text{Int}(\zeta))$ and $\not\sim \text{Int}((\zeta) \preceq \zeta) \rightarrow (Cv \preceq \sim \text{Int}(Cv) \preceq \sim \text{Int}(C))$.

9.1.1.2 Every infinite sequence $I, \sim \text{Int}(I), \ldots, \sim \text{Int}^n(I), \ldots$ is not strictly increasing.

The approximation sequence $S_0, \ldots, S_n, \ldots$ is recursively defined by:

9.1.1.3 $S_0 = I$

$$S_{n+1} = \begin{cases} 
\text{if } \not\sim \text{Int}(S_n) \preceq S_n \text{ then } \\
\text{else } \\
\end{cases}$$

We now prove that $\exists m$ finite such that:

$$S_0 \preceq S_1 \preceq \ldots \preceq S_m = S_{m+1} = \ldots$$

Let $m$ be the least natural number (eventually, in--
9.1.2 Generalization of Kleene's Ascending Sequence

When A-Cont satisfies the ascending chain condition one can choose $\lambda \text{int}$ to be int and therefore the

As before, we define:

9.1.3.5 $\overline{\lambda \text{int}} = \lambda \text{int}. (\lambda q. \lambda \text{int}(q, C_v))$
Let us note \([a, b]\) where \(a \leq b\) the predicate \(a \leq x \leq b\). The system of equations corresponding to the example is:

(0) \(C0 = [1, 100]\)
(1) \(C1 = [1, 1]\)
(2) \(C2 = C1 \cup C4\)
(3) \(C3 = C2 \cap [-\infty, 100]\)
(4) \(C4 = C3 \cap [1, 1]\)
(5) \(C5 = C2 \cap [101, +\infty]\)

Assignment statements are treated using an interval arithmetic (e.g. \([i, j] + [k, \ell] = [i+k, j+\ell]\) naturally extended to include the case of the empty interval). Similarly tests are treated using an "interval logic". Since there exist infinite intervals 
\(\{x \in \mathbb{R} \mid |x| < 0.1\}\) there exist infinite relations \(x \in \mathbb{R} \mid |x| < 0.1\). The final context on each arc is marked by a star *.

In this example the widening is a very rough operation which introduces a great loss of information. However, it can be seen in the trace that tests behave like filters. Furthermore, for PASCAL like languages, one can first use the bounds given in the declaration of \(x\) before widening to infinite limits.

8.3 Finite Iterative and Decreasing Approximation of the Least Fixpoint Starting from an Upper Bound

The ascending approximation sequence leads to an upper bound \(s_n = \text{Int}^n(s)\) of the least fixpoint \(CV\) of \(\text{Int}\). Moreover \(\text{Int}(s_m) \leq s_m\). Since \(\text{Int}\) is order preserving, this implies that:

\[ s_0 \geq \text{Int}(s_m) \geq \ldots \geq \text{Int}^n(s_m) \geq \ldots \geq \text{CV}. \]
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound on \( n \)).

Let \( \mathcal{D} \)-\text{Int} : A-\text{Cont} \rightarrow A-\text{Cont} be such that:

9.3.2.1 \( \{ \forall c \in A-\text{Cont} \mid (c \not\subseteq \text{Int}(c)) \rightarrow (c \not\subseteq \mathcal{D} \text{-Int}(c) \not\subseteq \text{Int}(c)) \} \)

9.3.2.2 \( \forall c \in A-\text{Cont}, \) every infinite sequence \( c, d; \text{Int}(c), \ldots, d; \text{Int}(d), \ldots \) is not strictly decreasing.

The truncated decreasing sequence \( s_0', \ldots, s_n' \) is recursively defined by:

9.3.2.3 \[ s_0' = s_m, \quad s_{n+1}' = \begin{cases} s_n' & \text{if } (s_n' \not\subseteq \text{Int}(s_n')) \text{ and } (s_n' \not\subseteq \mathcal{D} \text{-Int}(s_n')) \\ \text{else} & \end{cases} \]

Let us now prove that the truncated decreasing sequence is a finite strictly decreasing chain which terms are greater than \( \text{CV} \) the least fixpoint of \( \text{Int} \).

Let \( p \) be the least natural number (eventually infinite) such that \( s_p' = s_{p+1}' \). Trivially from 9.1.1:

\[ s_0' = s_m \not\subseteq \text{Int}(s_0') \not\subseteq \text{CV} \]

If \( p > 0 \) then \( s_0' \not\subseteq \text{Int}(s_0') \), therefore \( s_0' \not\subseteq \text{Int}(s_0') \).

Then applying 9.3.2.1 we have:

\[ s_0' \not\subseteq \mathcal{D} \text{-Int}(s_0') = s_1' \not\subseteq \text{Int}(s_1') \not\subseteq \text{CV} \]

But 9.3.2.3 implies \( s_0' \not\subseteq \text{Int}(s_0') \), hence:

\[ s_0' = s_1' \not\subseteq \text{Int}(s_1') \not\subseteq \text{CV} \]

For the induction step, let us suppose that for \( k < p \), we have:

\[ s_k' \not\subseteq \text{Int}(s_k') \not\subseteq \mathcal{D} \text{-Int}(s_k') \not\subseteq \text{CV} \]

Since \( \text{Int} \) is order preserving we have:

\[ \text{Int}(s_{k+1}') \subseteq \text{Int}(s_k') \not\subseteq \text{CV} \]

By transitivity \( s_k' \not\subseteq \text{Int}(s_k') \) and since 9.3.2.3 implies \( s_k' \not\subseteq \mathcal{D} \text{-Int}(s_k') \) we have from 9.3.2.1:

\[ s_k' \not\subseteq \mathcal{D} \text{-Int}(s_k') = s_{k+1}' \not\subseteq \text{Int}(s_k') \]

Since 9.3.2.3 implies \( s_k' \not\subseteq \text{Int}(s_k') \) we have:

\[ s_k' \not\subseteq s_{k+1}' \not\subseteq \mathcal{D} \text{-Int}(s_k') \not\subseteq \text{CV} \]

By recurrence on \( k \) the result is true for \( k = p \). Moreover 9.3.2.2 implies that \( p \) is finite. Q.E.D.

9.3.3 Generalization of Kleene’s Descending Sequence

When \( A-\text{Cont} \) satisfies the descending chain condition, one can choose \( \mathcal{D} \)-\text{Int} to be \( \text{Int} \), in which case the final result \( s_p' = \mathcal{D} \text{-Int}(c_p') \) is a fixpoint greater or equal to the least fixpoint \( \text{CV} \) of \( \text{Int} \).

The limit of the descending sequence \( s_0' = s_1', \ldots, s_p' = \mathcal{D} \text{-Int}(c_p') \) is an upper bound of the greatest fixpoint of \( \text{Int} \).

9.3.4 Narrowing in Truncated Decreasing Sequences

By analogy with 9.1.3 we define now the narrowing operation in order to build a possible construction of \( \mathcal{D} \)-\text{Int} by local modifications to \( \text{Int} \):

9.3.4.1 \( \Delta : A-\text{Cont} \times A-\text{Cont} \rightarrow A-\text{Cont} \)

9.3.4.2 \( \forall (c, c') \in A-\text{Cont}^2, \quad \{ c \sqsupseteq c' \} \Rightarrow \{ c \sqsupseteq c \sqsupseteq c' \} \)

9.3.4.3 Every infinite sequence \( s_0, \ldots, s_n, \ldots \) of the form \( s_0 = c_0, s_1 = s_0 \Delta c_1, \ldots, s_n = s_{n-1} \Delta c_n, \ldots \) for arbitrary abstract contexts \( c_0, c_1, \ldots, c_n, \ldots \) is not strictly decreasing.

The approximation interpretation:

\[ \mathcal{D} \text{-Int} : \text{Arcs}^3 \times A-\text{Cont} \rightarrow A-\text{Cont} \] is defined by:

9.3.4.4 \( \mathcal{D} \text{-Int} = \lambda(q, \text{CV}) \cdot \begin{cases} \text{CV}(q) \Delta \text{Int}(q, \text{CV}) & \text{if } q \in \text{W-arcs} \\ \text{else} & \end{cases} \)

\[ \mathcal{D} \text{-Int} = \lambda(q, \text{CV}) 

This definition of \( \mathcal{D} \)-\text{Int} trivially satisfies the requirement 9.3.2.1 since \( \text{CV} \sqsupseteq \text{Int}(\text{CV}) \) implies \( \text{CV}(q) \sqsupseteq \text{Int}(q, \text{CV}) \), \( q \in \text{W-arcs} . \quad \) If \( q \in \text{W-arcs} \) then 9.3.4.2 implies that \( \text{CV}(q) \sqsupseteq \text{CV}(q) \Delta \text{Int}(q, \text{CV}) = \mathcal{D} \text{-Int}(q, \text{CV}) \not\supseteq \text{Int}(q, \text{CV}) \). Otherwise, if \( q \not\in \text{W-arcs} \) then \( \text{CV}(q) \not\subseteq \text{Int}(q, \text{CV}) \). Hence \( \text{CV} \not\subseteq \mathcal{D} \text{-Int}(\text{CV}) \not\subseteq \text{Int}(\text{CV}) \).

The proof of termination (requirement 9.3.2.2) is very similar to the one outlined for A-\text{Int} in section 9.1.3.

9.4 Example: Bounds of Integer Variables

Let us come back to example 9.2. The system of equations was:

\[
\begin{align*}
(1) \quad C_1 &= [1, 1] \\
(2) \quad C_2 &= C_1 \cup C_4 \\
(3) \quad C_3 &= C_2 \cap [-\infty, 100] \\
(4) \quad C_4 &= C_3 + [1, 1] \\
(5) \quad C_5 &= C_2 \cap [101, +\infty]
\end{align*}
\]

The ascending approximation sequence led to the approximate solution:

+ \( C_1 = [1, 1] \)
+ \( C_2 = [1, +\infty] \)
+ \( C_3 = [1, 100] \)
+ \( C_4 = [2, 101] \)
+ \( C_5 = [101, +\infty] \)

Let us define the narrowing \( \Delta \) of intervals by:

\[ \Delta[i, j] = \begin{cases} \text{null} & \text{if } i = j \\
[i, j] & \text{if } i < j \leq \text{min}(i, j) \text{ or } \text{max}(i, j) < j
\end{cases} \]

248
Thus narrowing just discards infinite bounds and makes no improvement on finite bounds, it satisfies the requirements of 9.3.4. According to 9.3.4.4 the system of equations is modified by:

(2) $C_2 = C_2 \land (C_1 \lor C_4)$

The descending approximation sequence is:

- $C_2 = C_2 \land (C_1 \lor C_4)$
  - $[1, +\infty] \land ([1, 1) \lor [2, 101])$
  - $[1, +\infty] \land [1, 101]$

- $C_2 = [1, 101]$
- $C_3 = C_2 \cap [-\infty, 100]$
- $C_3 = [1, 101] \cap [-\infty, 100] = [1, 100]$
  - stop on that path.
- $C_5 = C_2 \cap [101, +\infty]$
- $C_5 = [1, 101] \cap [101, +\infty] = [101, 101]$
  - exit.

On that example the approximate solution has been improved so that the least fixpoint is reached but this is not the case in general.

9.5 Dual Approximation Methods

The lattice $\mathcal{A}$-cont may be partitioned as follows:

$\llcorner\Phi$ and $\lrcorner\Phi$ are the least and greatest fixpoints of $\llcorner\Phi$, and $\lrcorner\Phi$ is its dual.

When $X \gtrless Y$ we have noted $X \xrightarrow{-\longrightarrow} Y$.

The truncated descending sequence TDS is fundamentally different from AAS, since it ensures that the successive approximations starting from $S_m$ remain in the partition $\{X | X \gtrless \llcorner\Phi(X)\}$, so that their limit $S'_n$ is greater than $\llcorner\Phi$:

It is clear that the ascending approximation sequence AAS when starting from 1 leads to an upper bound of the least fixpoint $\llcorner$ of $\llcorner\Phi$, and the truncated descending sequence TDS when starting.
Any of the AAS, TDS, DAS, TAS methods may yields a fixed point \( fp \) which is not the fixed point \( ffp \) or \( gfp \) of interest. None of these methods can improve \( fp \) to reach \( ffp \) or \( gfp \), therefore a "fixed point improvement method" is necessary. It is our feeling that such a method could be designed only when considering that A-Gont possesses a richer structure (i.e. for particular applications).

Furthermore, in the AAS, TDS, DAS, TAS sequences the term of rank \( n \) is computed only as a function of the term of rank \( n-1 \), hence these are "separate steps" methods. One can as well imagine to use "bound steps" methods, where the term of rank \( n \) is computed as a function of the terms of rank \( n-1, n-2, ..., n-k \). In this last case the Kleene's sequences may be used to compute the first \( k \) terms. After \( k \) steps more informations about the program would be available to heuristically accelerate the convergence so that the definition of \( A_{-int} \) and \( B_{-int} \) could be more refined.

Finally, going deeply into the comparison with numerical analysis methods, it is clear that some measure is necessary to control the accuracy of the result. Its definition would certainly also necessitate some additional properties of the abstract contexts.

10. Conclusion

It is our feeling that most program analysis techniques may be understood as abstract interpretations of programs. Let us point out global data flow analysis in optimizing compilers (Kildall[73], Morel and Renvoize[76], Schwartz[75], Ullman[75], Wegbreit[75], ...), type discovery (Cousot[76], Sintzoff[72], Tenenbaum[74], ...), program testing (Henderson [75], ...) symbolic evaluation of programs (Hewitt et al.[73], Karr[76], ...), program performance analysis (Wegbreit[76], ...), formalization of program semantics (Hoare and Lauer[74], Liger[75], Manna and Shamir[75], ...), verification of program correctness (Floyd[67], Park[69], Sintzoff[75], ...), discovery of inductive invariants (Katz and Manna[76], ...), proofs of program termination (Sintzoff[76], ...), program transformation (Sintzoff [76], ...).

There is a fundamental unity between all these apparently unrelated program analysis techniques: a new interpretation is given to the program text which allows to build an often implicit system of equations. The problem is either to verify that a solution provided by the user is correct, or to discover or approximate such a solution.

The mathematical model we studied in this paper is certainly the weakest which is necessary to unify these techniques, and therefore should be of very general scope. It can be considerably enriched for particular applications so that more powerful results may be obtained.

Acknowledgements

We wish to thank M. Sintzoff for stimulating discussions. We were very lucky to have F. Blanc do the typing for us.

11. References


Cousot[76*]. Static determination of dynamic properties of generalized type unions. Submitted for publication. (Sept.)


Kam and Ullman[75]. Monotone data flow analysis frameworks. TR.169, C.S. Lab., Princeton Univ.

Karr[76]. Affine relationships among variables of a program. Acta Inf. 6, 133-151.


Naur[65]. Checking of operand types in ALGOL compilers, BIT 5, 151-163.


Scott[71]. The lattice of flow diagrams. Symp. on Semantics of Programming Languages. Springer-Verlag Lecture Notes in Math. (E. Engeler, ed.), Vol. 188.


Sintzoff[76]. Eliminating blind alleys from backtrack programs. Proc. of the third Int. Coll. on Automata, Languages and Programming, Edinburgh, (July).


Tenenbaum[74]. Type determination for very high level languages. MSD-3, Courant Inst. of Math. Sc., New York U., (Oct.).


\[
\{\forall (x, y) \in L^2, (x \leq y) \Rightarrow \{f(x) \leq f(y)\}\} \\
\iff \{\forall (x, y) \in L^2, (f(x) \cup f(y)) \geq f(x) \cup f(y)\}\}
\]

(H1): Let \( F \) be an order-preserving function from the complete semi-lattice \( \langle L, \leq, \top, \bot \rangle \) in itself.

(H1): Let \( \overline{F} \) be an order-preserving function from the complete semi-lattice \( \langle L, \leq, \top, \bot \rangle \) in itself.

(L1): The fixpoints of \( F \) form a non-empty complete lattice with supremum \( \gamma \) and infimum \( \alpha \) such that:

\[
\alpha = \gamma(x) = \{x \in L \mid (x \leq F(x))\}
\]

\[
\gamma = \{x \in L \mid (F(x) \leq x)\}
\]

(This result is proved in Tarski[55], pp. 286-287). Note that the fixpoints of \( F \) need not form a sublattice of \( L \).

We note \( g \) and \( \gamma \) the greatest and least fixpoints of \( F \).

(H2): Let \( \alpha \) and \( \beta \) be such that:

\[
\begin{align*}
\alpha &: L \to L \\
\beta &: L \to L
\end{align*}
\]

(H2): \( \alpha \) is order preserving

(H2): \( \gamma \) is order preserving

(H2.5): \( \forall x \in L, x \leq \gamma(\alpha(x)) \)

(H2.6): \( \forall x \in L, x \leq \alpha(\gamma(x)) \)

(H3.1): \( (H1), (H2), (H1) \) and \( \forall x \in L, F(\alpha(x)) = \alpha(F(x)) \)

(H3.2): \( (H1), (H2), (H2) \) and \( \forall x \in L, \gamma(F(x)) \geq F(\gamma(x)) \)

(L2): \( (H3.1) \iff (H3.2) \)

Proof:

\[
\forall x \in L,
\begin{align*}
F(\alpha(x)) &\leq F(\gamma(x)) \iff \alpha(\gamma(x)) = \alpha(F(x)) \text{ in H3.1} \\
F(x) &\leq F(\gamma(x)) \iff \gamma(F(x)) \geq F(\gamma(x)) \text{ from H2.2} \\
\gamma(F(x)) &\leq F(\gamma(x)) \iff H2.6 \text{ and transitivity.}
\end{align*}
\]

\[
\forall x \in L,
\begin{align*}
\gamma(F(\alpha(x))) &\geq F(\gamma(x)) \iff \alpha(\gamma(x)) = \alpha(F(x)) \text{ in H3.2} \\
\gamma(\alpha(x)) &\geq x \iff H2.6 \text{ order preserving in (H1)} \\
\gamma(F(\alpha(x))) &\geq F(\gamma(x)) \iff \text{transitivity} \\
\alpha(\gamma(F(\alpha(x)))) &\geq \alpha(F(\gamma(x))) \iff H2.3 \text{ transitivity} \\
\alpha(F(\alpha(x))) &\geq \alpha(F(x)) \iff H2.5 \text{ transitivity}
\end{align*}
\]

O.E.D.
(T1) \( H_1, \overline{H_1}, H_2, H_3 \) imply that the greatest fixpoints \( g \) and \( \bar{g} \) of \( F \) and \( \overline{F} \) are related by:
\[
(\alpha(g) \leq \bar{g}) \quad \text{and} \quad (\bar{g} \leq \gamma(\bar{g}))
\]

Proof:

The existence of \( g \) and \( \bar{g} \) is stated by (L1).

\[
\begin{align*}
\overline{\gamma(g)} & \geq \alpha(g) & \text{trivially} \\
\overline{\gamma(g)} & \geq \gamma(\alpha(g)) & \text{since } g = F(g) \\
\overline{\gamma(\alpha(g))} & \geq \alpha(g) & \text{H3.1, } \overline{\gamma}, \text{ isotone, } \overline{\gamma} \text{ transitive} \\
\overline{g} & \geq \alpha(g) & \text{L3} \\
\gamma(\overline{g}) & \geq \gamma(\alpha(g)) & \text{H2.4} \\
\gamma(\overline{g}) & \geq g & \text{H2.6, } \overline{\gamma} \text{ transitive.} \\
\end{align*}
\]

Q.E.D.

Replacing \( g, \overline{g}, \gamma, \overline{\gamma}, \geq, \leq, F, \overline{F}, \alpha, \gamma, H3.1, H2.4, H2.6 \) respectively by \( \ell, \overline{\ell}, \leq, \geq, F, \overline{F}, \alpha, \gamma, H3.2, H2.3, H2.5 \) in the above proof, we get the "dual" theorem:

(\( T2 \)) \( H_1, \overline{H_1}, H_2, H_3 \) imply that the least fixpoints \( \ell \) and \( \overline{\ell} \) of \( F \) and \( \overline{F} \) are related by:

\[
(\gamma(\overline{\ell}) \geq \ell) \quad \text{and} \quad (\ell \geq \alpha(\ell))
\]

According to Scott[71] a subset \( X \subseteq L \) is called directed if every finite subset of \( X \) has an upper bound (in the sense of \( \leq \)) belonging to \( X \). (An obvious example of a directed subset is a non-empty ascending chain.) A function \( f : D \to D \) is called continuous if whenever \( X \subseteq L \) is directed, then \( f(\bigcup \{ x | x \in X \}) = \bigcup \{ f(x) | x \in X \} \).

(H4): Let \( F \) be a continuous function from the complete semi-lattice \( \langle L, \leq, \varepsilon, \tau, \iota \rangle \) in itself.

(\( H4' \)): Let \( \overline{F} \) be a continuous function from the complete semi-lattice \( \langle L, \geq, \varepsilon, \tau, \iota \rangle \) in itself.

We note \( F^0(x) = x \) and \( F^{n+1}(x) = F(F^n(x)) \).

(L4): \( H4(\overline{H4}) \) implies that \( F(\overline{F}) \) has a least fixpoint \( \ell(\overline{\ell}) \) which is the limit \( \lim_{n \to \infty} F^n(\ell) \) of the Kleene's sequence \( \ell \leq F(\ell) \leq \ldots \leq F^n(\ell) \leq \ldots \).

(The proof is easy to adapt from Kleene[52]'s proof of the first recursion theorem pp. 348–349).

\[
\text{\( \ell \)}
\]

252