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 \(((+), (-), (\&))\) 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...
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 \( \{ \bot_S, \top_S \} \) to it, and imposing the ordering \( \bot_S \leq x \leq \top_S \) 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:
  \[ \text{Env} = \text{Ident}^5 \to \text{Values} \]

  We assume that the meaning of an expression \( \text{expr} \in \text{Expr} \) in the environment \( e \in \text{Env} \) is given by \( \text{val}_{\text{Expr}}(e) \) so that:
  \[ \text{val}_{\text{Expr}} : \text{Expr} \to [\text{Env} \to \text{Values}] \]
  In particular the projection \( \text{val}_{\text{Expr}} \) of the function \( \text{val} \) in domain \( \text{Expr} \) has the functionality:
  \[ \text{val}_{\text{Expr}} : \text{Expr} \to [\text{Env} \to \text{Bool}] \]

  The state set "States" consists of the set of all information configurations that can occur during computations:
  \[ \text{States} = \text{Arcs}^5 \times \text{Env} \]
  A state \( (s \in \text{States}) \) consists in a control state \( (\text{cs}(s)) \) and an environment \( (\text{env}(s)) \), such that:
  \[ (s \in \text{States}, s = (\text{cs}(s), \text{env}(s))) \]

  We use a continuous conditional function \( \text{cond}(b, e_1, e_2) \) equal to \( i, e_1, e_2 \) or \( \top \) respectively as the value of \( b \) is \( i, \text{true}, \text{false} \) or \( \top \). We also use
A "computation sequence" with initial state $i_0 \in I\text{-states}$ is the sequence:
$$s_n = \text{n-state}^n(i_0)$$
for $n = 0, 1, \ldots$
where $\text{id}$ is the identity function and $\text{id}^n = f \circ f \circ \cdots \circ f$.

The initial to final state transition function:
$$\text{n-state}^\infty : \text{States} \rightarrow \text{States}$$
is the minimal fixpoint of the functional:
$$\lambda F. (\text{n-state} \circ F)$$
Therefore
$$\text{n-state}^\infty = \mu Y \text{States} \rightarrow \text{States} (\lambda F. (\text{n-state} \circ F))$$
where $Y_f(f)$ denotes the least fixpoint of $f : D \rightarrow D$, Tarski [55].

4. Static Semantics of Programs

The constructive or operational semantics of programs defined in section 3 considers the sequence in which states occur during execution. The fundamental remark of Floyd [67] is that to prove static properties of programs it is often sufficient to consider the sets of states associated with each program point.

Hence, we define the context $C_q$ at some program point $q \in \text{ArCs}$ of a program $P$ to be the set of all environments which may be associated to $q$ in all the possible computation sequences of $P$:
$$C_q \subseteq \text{Contexts} = \mathcal{E}_\text{env}$$
$$C_q = \{ e | (n \geq 0, \exists i_n \in I\text{-states} | <q, e> = \text{n-state}^n(i_n)\}$$

The context vector $C_v$ associates a context to each of the program points of a program $P$:
$$C_v : \text{Contexts} \rightarrow \text{Context-Vectors}$$
$$C_v = \lambda q. \{ e | (n \geq 0, \exists i_n \in I\text{-states} | <q, e> = \text{n-state}^n(i_n)\}$$

According to the semantics of programs, the context $C_v(r)$ associated to arc $r$ is related to the context $C_v(q)$ at arc $q$ adjacent to $r$,
$$(\text{end}(q) = \text{origin}(r))$$
from the definition of the state transition function we can prove the equation:
$$C_v(r) = \text{n-context}(r, C_v)$$

Since the equation $C_v(r) \in \text{n-context}(r, C_v)$ must be valid for each arc, $C_v$ is a solution to the system of "forward" equations:
$$C_v = \text{F-cont}(C_v)$$
where
$$\text{F-cont} : \text{Context-Vectors} \rightarrow \text{Context-Vectors}$$
is defined by:
$$\text{F-cont}(C_v) = \lambda r. \text{n-context}(r, C_v)$$

Context-Vectors is a complete lattice with union $\cup$ such that $C_v_1 \cup C_v_2 = \lambda r. (C_v_1(r) \cup C_v_2(r))$.

$\text{F-cont}$ is order preserving for the ordering $\subseteq$ of Context-Vectors which is defined by:
$$C_v_1 \subseteq C_v_2 \iff (\forall r \in \text{Arcs}, C_v_1(r) \subseteq C_v_2(r))$$

Hence it is known that $\text{F-cont}$ has fixpoints, Tarski [55]. However, it is trivial to exhibit examples which show that these fixpoints are not always unique. Fortunately, it can be shown that $C_v$ is included in any solution $\mathcal{E}$ to the system of equations $X = \text{F-cont}(X)$, $(C_v \leq \mathcal{E})$. Tarski [55] shows that this property uniquely determines $C_v$ as the least fixpoint of $\text{F-cont}$. Thus $C_v$ can be equivalently defined by:
$$D_1 : C_v = \lambda q. \{ e | (n \geq 0, \exists i_n \in I\text{-states} | <q, e> = \text{n-state}^n(i_n)\}$$
$$D_2 : C_v = \mu Y \text{Context-Vectors} (\text{F-cont})$$

The concrete context vector $C_v$ is such that for any program point $q \in \text{ArCs}$ of the program $P$,

(a) $C_v(q)$ contains at least the environments $e$ which may be associated to $q$ during any execution of $P$:
$$(\exists i_n \geq 0, \exists i_n \in I\text{-states} | <q, e> = \text{n-state}^n(i_n) \rightarrow (e \in C_v(q)))$$

(b) $C_v(q)$ contains only the environments $e$ which may be associated to $q$ during an execution of $P$:
$$(e \in C_v(q)) \rightarrow (\exists i_n \geq 0, \exists i_n \in I\text{-states} | <q, e> = \text{n-state}^n(i_n))$$

$C_v$ is merely a static summary of the possible executions of the program. However, our definitions $D_1$ or $D_2$ of $C_v$ cannot be utilized at compile time since the computation of $C_v$ consists in fact in running the program (for all the possible input data). In practice compilers may consider states which can never occur during program execution (e.g. some compilers consider that any program may always perform a division by zero although this is not the case for most programs). Hence compilers may use "abstract" contexts satisfying (a) but not necessarily (b), which therefore correctly approximate the concrete contexts we considered until now.

5. Abstract Interpretation of Programs

5.1 Formal Definition

An abstract interpretation $I$ of a program $P$ is a tuple
$$I = \langle A\text{-Cont}, \leq, \tau, I, \iota, \text{Int} \rangle$$
where the set of abstract contexts is a complete $o$-semilattice with ordering $\leq$, $(x \leq y) \iff (x \circ y = y)$. This implies that $A\text{-Cont}$ has a supremum $\tau$. We suppose also $A\text{-Cont}$ to have an infimum $\iota$. 

240
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 $A$-Cont = Arccs → $A$-Cont.

Whatever $(Cv', Cv'')$ ∈ $A$-Cont² may be, we define:

$Cv' \triangleright Cv'' = \lambda r. Cv'(r) \circ Cv''(r)$

$Cv' \triangleright= Cv'' = \{ r ∈ Arccs^0, Cv'(r) ≤ Cv''(r) \}$

$\tilde{\top} = \lambda r. \top$ and $\tilde{\bot} = \lambda r. \bot$
The determination of available expressions, backdominators, intervals, ... requires a forward system of equations. Some global flow problems, notably the live variables and very busy expressions require propagating information backward through the program graphs. These are examples of backward.

Instead of the global hypothesis 6.0 we will use the following local hypothesis on the concrete and abstract interpretations of primitive language constructs:

\[(\forall (a, x) \in \text{Arens} \times \text{A-Cont}, \sim)\]
where \( n\text{-pred} \) defines Floyd's strongest postcondition:

\[
\text{n-pred}(r, P) = \begin{cases} 
\text{let}(a \text{ be origin}(r), (p \text{ be } n\text{-pred(origin}(r)))) \text{within} \\
\text{let}(s \text{ be } \text{origin}(r), (p \text{ be } n\text{-pred(origin}(r)))) \text{within} \\
\end{cases}
\]

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:

\[
\{I \leq I' \} \iff \{(I \leq I') \text{ and } (I' \leq I)\}
\]
A further abstraction may be:

\[\alpha([a, b]) = \begin{cases} \text{if } a \neq b \text{ then } a & \text{else if } a \geq 0 \text{ then } + \\
& \text{else if } b \leq 0 \text{ then } - \\
& \text{else } \text{ all } \end{cases}
\]

The abstract contexts are then:

\[I_C = \ldots 4 -3 -2 -1 0 1 2 3 4 \ldots\]

This interpretation may be abstracted by two non-comparable abstractions:

\[I_{CP} = \ldots -3 -2 -1 0 1 2 3 \ldots\]

\[I_{RS} = \ldots 0 1 \ldots\]

\[T_I = \{I, t, I, \lambda(x, y), I, \lambda(a, C), I\}\]

where \(t\) is the relation which is always true. We have exhibited a sublattice of \(I\) which is:

\[T_I = I_R \rightarrow I_{CP} \rightarrow I_{CS} \rightarrow I_I \rightarrow I_{SS}\]

8. Abstract Evaluation of Programs

The system of equations:

\[C_v : \text{Int}(C_v)\]

resulting from an interpretation \(I = \langle A, \text{Cont}, \leq, \tau, i, \text{Int} \rangle\) of a program \(P\) may be solved by "elimination" methods, (e.g. Tarjan[75]). Otherwise, one can use an "iterative" algorithm which computes Kleene's sequence (L4 of Appendix 12):

\[C_v := (C := I; \text{until } C = \text{Int}(C) \text{ do } C := \text{Int}(C) \text{ repeat}; C)\]

8.1 Correctness

If \(\text{Int}\) is supposed to be a complete morphism (i.e. infinitely distributive over \(\ast\)) then \(C_v\) is the least fixpoint of \(\text{Int}\), (e.g. Kildall[75]), since in a semi-lattice of finite length, any distributive function is a complete morphism. Under the weaker assumption that \(\text{Int}\) is continuous, the limit \(C_v\) of Kleene's sequence can also be shown to be the least fixpoint of \(\text{Int}\), (e.g. Wegbreit[75], since in a well-founded semi-lattice, any isotope function is continuous). Finally, if \(\text{Int}\) is only supposed to be isotope, \(C_v\) is an approximation (\(\hat{2}\)) of the least fixpoint (e.g. Kam and Ullman[75]).

8.2 Termination

The abstract evaluation terminates iff Kleene's sequence is finite. This may be the case because \(A\)-Cont is finite (e.g. type checking in ALGOL 60, Naur[65]), or a finite subset only is to be considered for any particular program (e.g. type checking in ALGOL 68), or \(A\)-Cont may be of finite length \(m\), the length of any strictly increasing chain is bounded by \(m\), Kildall[73], Wegbreit[75]) or \(A\)-Cont may satisfy the ascending chain condition (every strictly increasing chain is finite, although not bounded). A lattice may have infinite chains, although \(\text{Int}\) is chosen so that Kleene's sequences are finite. Finally an infinite Kleene's sequence may be arbitrarily truncated (to get a lower bound of its limit), some induction principle (Sintoff [75]) or heuristics (Katz and Manual[76]) may be used to pass to the limit, or approximate it, (Coqot[76]).

8.3 Efficiency

In practice efficient versions of the Kleene's sequence are used. These consist in a symbolic execution of the program which propagates information along paths of the program until stabilization. A specification of order of information propagation may lead to optimal algorithms for specific applications (references in Tarjan[76]).
8.4 Example: Performance Analysis of Programs

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-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:

$$\text{Kir}(r, Cv) = \begin{cases} \text{let } n \text{ be origin}(r) \text{ in } & \\ \text{case } n \text{ in } & \\ \text{Entries} \Rightarrow 1 \{\text{unique entry node}\} & \\ \text{Assignments} \Rightarrow & \\ \text{Tests} \Rightarrow & \\ \text{case } r \text{ in } & \\ \{a-\text{succ}\text{-}r(n)\} \Rightarrow & Cv(a-pred(n)) \ast \frac{p(a-pred(n))}{\text{Prob}(\text{test}(n) = \text{true})} & \\ \{a-\text{succ}\text{-}r(n)\} \Rightarrow & Cv(a-pred(n)) \ast \frac{1 - \text{Prob}(\text{test}(n) = \text{true})}{\text{esac}} & \\ \text{esac} & \\ \end{cases}$$

The main difficulty is to obtain the probability $\text{Prob}(\text{true})$ 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 Kir is clearly continuous (although it is not a complete morphism) since

$$\text{Kir}(r, Cv) = \left\{ \begin{array}{ll} \text{let } n \text{ be origin}(r) \text{ in } & \\ \text{case } n \text{ in } & \\ \text{Entries} \Rightarrow 1 \{\text{unique entry node}\} & \\ \text{Assignments} \Rightarrow & \\ \text{Tests} \Rightarrow & \\ \text{case } r \text{ in } & \\ \{a-\text{succ}\text{-}r(n)\} \Rightarrow & Cv(a-pred(n)) \ast \frac{p(a-pred(n))}{\text{Prob}(\text{true})} & \\ \{a-\text{succ}\text{-}r(n)\} \Rightarrow & Cv(a-pred(n)) \ast \frac{1 - \text{Prob}(\text{true})}{\text{esac}} & \\ \text{esac} & \\ \end{array} \right.$$
9.1.2 Generalization of Kleene's Ascending Sequence

When A-Cont satisfies the ascending chain condition one can choose $A^\text{int}$ to be Int and therefore the approximation sequence generalizes Kleene's sequence and the related methods.

9.1.3 Widening in Increasing Approximation Sequences

The definition of the approximate interpretation $A^\text{int}$ in 9.1.1 is global. We now indicate a way to construct $A^\text{int}$ by local modifications to Int.

Let $(q, r) \in \text{Arcs}^2$, we say that the context associated to $q$ is dependent on the context associated to $r$, if and only if:

\[ \exists C \in \text{A-Cont}, r \leq C \leq \text{A-Cont} \quad \text{such that} \quad \text{Int}(q, C) \neq \text{Int}(q, C/r) \]

(e.g. in a forward system of equations the context associated to $q$ may only depend on the contexts associated with the immediate predecessor arcs of $q$). In the system of equations $Cv = \text{Int}(Cv)$ we define a cycle to be a sequence $<q_0, \ldots, q_n>$ of arcs, such that $i \in [1, n], Cv(q_i) = \text{Int}(Cv(q_{i-1}))$ depends on $Cv(q_i)$ and $Cv(q_i)$ depends on $Cv(q_{i+1})$. (e.g. in a forward interpretation a cycle corresponds to a loop in the program).

In any infinite strictly increasing Kleene's sequence $Cv_1, \ldots, Cv_n, \ldots$, since Arcs is finite there is some arc $q$ for which the sequence $Cv(q_1), \ldots, Cv(q_n), \ldots$ never stabilizes. Therefore $q$ must belong to a cycle or the contexts associated to $q$ transitivity depend on the contexts associated to some other arc $r$ which itself belongs to a cycle. The sequence of contexts associated to any arc of that cycle never stabilizes. In order to avoid this phenomenon, we introduce:

- The binary operation $\triangledown$ called widening defined by:

\[ \triangledown : A-\text{Cont} \times A-\text{Cont} \rightarrow A-\text{Cont} \]

9.1.3.2 $\forall (C, C') \in A-\text{Cont}^2, C \circ C' \leq C \triangledown C'$

9.1.3.3 Every infinite sequence $s_0, s_1, s_2, \ldots$ of the form $s_0 = C_0, s_n = s_{n-1} \triangledown C_{n}$, where $C_0, \ldots, C_n$, arc arbitrary abstract contexts is not strictly increasing.

- The set $W$-arcs of widening arcs, which is one of the minimal sets of arcs such that any cycle $<q_0, \ldots, q_n >$ of the system of equations $Cv = \text{Int}(Cv)$ contains at least a widening arc:

\[ \forall n \in [1, n] \quad q_i \in (W-\text{arcs}) \quad (e.g. \text{in a forward interpretation on a reducible program graph, W-arcs may be chosen to be the set of exit arcs of the junction nodes which are interval headers. On irreducible graphs an arbitrary choice has to be made so that any loop of the program goes through a widening arc}).

- The approximate interpretation $A^\text{int} : A-\text{Cont} \times A-\text{Cont} \rightarrow A-\text{Cont}$ defined by:

\[ A^\text{int}(q, Cv) = \lambda(q, Cv), \quad \text{if} \quad Cv(q) = \text{Int}(q, Cv) \]

\[ \text{else} \quad \text{Int}(q, Cv) \]

As before, we define:

\[ A^\text{int} = \lambda(q, Cv) \cdot (\lambda(q, A^\text{int}(q, Cv))) \]

Now we have to show that this definition of $A^\text{int}$ satisfies the requirements 9.1.1.2 and 9.1.1.7.

Let us consider a sequence $S_0 = \overline{1}, \ldots, S_n = \overline{\text{Int}(q)} \ldots$. We show that this sequence is increasing that is to say:

\[ S_n \triangledown A^\text{int}(S_n) \quad \forall n \geq 0. \]

Trivially for $n = 0$, $S_0 = \overline{1} \triangledown A^\text{int}(S_0)$. For the induction step, suppose the result to be true for $n \leq m$. Let us prove that:

\[ S_{m+1} \triangledown A^\text{int}(S_{m+1}) \Rightarrow S_{m+1} \triangledown A^\text{int}(S_{m+1}), \forall q \in \text{Arcs}. \]

If $q \in W$-arcs, then $\triangledown A^\text{int}(q, S_{m+1}) = A^\text{int}(q, S_{m+1}) \triangledown A^\text{int}(q, S_{m+1}) = A^\text{int}(q, S_{m+1})$.

Finally $S_{m+1} \triangledown A^\text{int}(S_{m+1})$, Q.E.D.

An infinite sequence $S_0 = \overline{1}, \ldots, S_n = A^\text{int}(S_{n-1})$, cannot be strictly increasing since otherwise there would exist some widening arc for which the sequence $S_0(q), S_1(q), \ldots, S_n(q), \ldots$ would never stabilize thus contradicting 9.1.3.3.

We now introduce 9.1.1.7 that is to say that:

\[ \forall n \geq 0, A^\text{int}(S_n) \triangledown A^\text{int}(S_n) \]

implies

\[ S_n \triangledown A^\text{int}(S_n) \triangledown A^\text{int}(S_n) \]

\[ \Rightarrow (S_n \triangledown A^\text{int}(S_n))(q) \triangledown A^\text{int}(S_n(q)), \forall q \in \text{Arcs} \]

\[ \Rightarrow S_n(q) \triangledown A^\text{int}(q, S_n(q)), \forall q \in \text{Arcs} \]

If $q \in W$-arcs, we have $A^\text{int}(q, S_n) = S_n(q) \triangledown A^\text{int}(q, S_n) \triangledown A^\text{int}(q, S_n)$.

9.1.3.2. If now $q \not\in W$-arcs we must show:

\[ A^\text{int}(q, S_n(q)) \leq A^\text{int}(q, S_n(q)) \]

which is true, from 9.1.3.6, Q.E.D.

9.2 Examples: Bounds of Integer Variables

In a PASCAL program operating on arrays, the compiler should ensure that arrays are subscripts only by indices within bounds. For that purpose one can use the lattice $L_1$ of section 7. Let us take an obvious example:

246
The final context on each arc is marked by a star.

\[ C_0 = [1, 100] \]
\[ C_1 = C_0 + [1, 1] \]
\[ = [1, 100] + [1, 1] \]
\[ C_4 = [2, 101] \]
\[ C_2 = C_1 \]
\[ C_5 = C_2 \leq C_2 = [1, +\infty] \]
\[ C_6 = C_2 \cap [101, +\infty] \]
\[ = [1, +\infty] \cap [101, +\infty] \]
\[ C_7 = [101, +\infty] \]
\[ C_8 = [101, +\infty] \]
\[ exit, stop. \]
Let \( D\text{-int} : A\text{-Cont} \rightarrow A\text{-Cont} \) be such that:

9.3.2.1 \( \forall c \in A\text{-Cont} \),

\[
(\exists \int(C) \rightarrow (C \succeq D\text{-int}(C) \succeq \int(C))
\]

9.3.2.2 \( \forall c \in A\text{-Cont}, \) every infinite sequence \( C, D\text{-int}(C), \ldots, D\text{-int}^n(C), \ldots \) is not strictly decreasing.

The truncated decreasing sequence \( S_0, \ldots, S_n, \ldots \) is recursively defined by:

9.3.2.3 \( S_0 = S_m \)

\[
S_{n+1} = \begin{cases} S_n & \text{if } (S_n \neq \int(S_n)) \text{ and } (S_n \neq D\text{-int}(S_n)) \\ S_n & \text{else} \end{cases}
\]

Let us now prove that the truncated decreasing sequence is a finite strictly decreasing chain whose terms are greater than \( \int(C) \) the least fixpoint of \( \int \).

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

\[
S_0 = S_m \succeq \int(S_0) \succeq \int(C)
\]

If \( p > 0 \) then \( S_p \neq \int(S_p) \), therefore \( S_p \succeq \int(S_p) \).

Then applying 9.3.2.1 we have:

\[
S_0 \succeq D\text{-int}(S_0) = S_1 \succeq \int(S_0) \succeq \int(C)
\]

But 9.3.2.3 implies \( S_0 \neq D\text{-int}(S_0) \), hence:

\[
S_1 > S_0 \succeq \int(S_0) \succeq \int(C)
\]

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

\[
S_{k-1} \succeq S_k \succeq \int(S_{k-1}) \succeq \int(C)
\]

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 \( D\text{-int} \) by local modifications to \( \int \):

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

9.3.4.2 \( \forall (C, C') \in A\text{-Cont}^2 \),

\[
[C \succeq C'] \Rightarrow (C \succeq C \Delta C' \succeq C')
\]

9.3.4.3 Every infinite sequence \( s_0, s_1, \ldots, s_n, \ldots \) 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 approximated interpretation

\( D\text{-int} : \text{Arcs}^\infty \times A\text{-Cont} \rightarrow A\text{-Cont} \) defined by:

9.3.4.4 \( D\text{-int} = \lambda(q, \int(C)) \) if \( q \in \text{W-arcs} \) then

\[
\int(C) \Delta \int(q, \int(C))
\]

else

\[
\int(q, \int(C))
\]

This definition of \( D\text{-int} \) trivially satisfies the requirement 9.3.2.1 since \( \forall \int(C) \in A\text{-Cont} \) with property \( C \succeq \int(C) \) implies \( \int(C) \succeq \int(q, \int(C)) \).

\( \forall q \in \text{W-arcs} \) if \( q \in \text{W-arcs} \) then 9.3.4.2 implies that \( C \succeq \int(C) \Delta \int(q, \int(C)) = D\text{-int}(q, \int(C)) \succeq \int(q, \int(C)) \).

Otherwise, if \( q \notin \text{W-arcs} \) then 9.3.4.2 implies that \( C \succeq \int(C) \Delta \int(q, \int(C)) = D\text{-int}(q, \int(C)) \).

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.
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:

\[
\begin{align*}
C_2 &= C_2 \land (C_1 \lor C_4) \\
&= [1, + \infty] \land ([1, 1] \lor [2, 101]) \\
&= [1, + \infty] \land [1, 101] \\
* C_2 &= [1, 101] \\
C_3 &= C_2 \cap [-\infty, 100] \\
* C_3 &= [1, 101] \cap [-\infty, 100] = [1, 100] \\
\text{stop on that path.} \\
C_5 &= C_2 \cap [101, + \infty] \\
* C_5 &= [1, 101] \cap [101, + \infty] = [101, 101] \\
\text{exit.}
\end{align*}
\]

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 \( \mathbb{A} \text{-} \mathbb{C} \text{ont} \) may be partitioned as follows:

\[
\begin{align*}
X \text{ and } \overline{\text{Int}}(X) \\
\text{non comparable} \\
X \leq \overline{\text{Int}}(X), X \geq \overline{\text{Int}}(X) \\
\overline{\text{Int}}(X) \\
\end{align*}
\]

\( \overline{\text{Int}} \) is continuous and has finite bounds. When \( \overline{\text{Int}} \) is infinite we have proposed to use an ascending approximation sequence (AAS) to approximate \( \overline{\text{Int}} \). Its limit may be some fixpoint \( \overline{\text{Int}} \), or some \( S_m \) such that \( S_m \geq \overline{\text{Int}}(S_m) \) and \( S_m \geq \overline{\text{Int}} \).

When \( X \geq Y \) we have noted \( X \rightarrow Y \).

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 \( \{X \mid X \geq \overline{\text{Int}}(X)\} \), so that their limit \( S_p \) is greater than \( \overline{\text{Int}} \):

\[
\begin{align*}
\overline{\text{Int}}(S_m) \\
\overline{\text{Int}} \\
\overline{\text{Int}}(X) \\
\end{align*}
\]

It is clear that the ascending approximation sequence AAS when starting from \( 1 \rightarrow \overline{\text{Int}} \) leads to an upper bound of the least fixpoint \( \overline{\text{Int}} \), and the truncated descending sequence TDS when starting from \( 1 \rightarrow \overline{\text{Int}} \) leads to an upper bound of the greatest fixpoint \( \overline{\text{Int}} \). Hence the AAS and TDS methods are not dual, therefore when considering their duals DAS and TAS we get a means to surround both extreme fixpoints of \( \overline{\text{Int}} \):

\[
\begin{align*}
\text{AAS} \\
\text{TDS} \\
\text{DAS} \\
\text{TAS} \\
\end{align*}
\]

249
Any of the AAS, TDS, DAS, TAS methods may yield a fixpoint $fp$ which is not the fixpoint $fp$ or $gfp$ of interest. None of these methods can improve $fp$ to reach $fp$ or $gfp$, therefore a "fixpoint improvement method" is necessary. It is our feeling that such a method could be designed only when considering that A-cont 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 $n-1, n-2, \ldots, n-k$. 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 P-int could be more refined.

Finally, going deeply into the comparision 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 Renvoise[76], Schwartz[75], Ullman[75], Wegbreit[75], \ldots), type discovery (Cousot[76'], Sintzoff[72], 

\( \text{\textit{Acknowledgements}} \)

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.


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[73]. Verifications of assertions using loc

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

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

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

(L1): The fixpoints of \( F \) form a non-empty complete lattice with supremum \( g \), infimum \( \ell \) such that:
\( g = \cup \{ x \mid (x \in L) \land (x \leq F(x)) \} \)
\( \ell = \cap \{ x \mid (x \in L) \land (F(x) \leq x) \} \)

(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 \( \ell \) the greatest and least
(T1): \(H_1, H_1, H_2, H_3\) imply that the greatest fixpoints \(g\) and \(\overline{g}\) of \(F\) and \(\overline{F}\) are related by:

\[
(\alpha(g) \preceq \overline{g}) \text{ and } (g \preceq \gamma(\overline{g}))
\]

Proof:
The existence of \(g\) and \(\overline{g}\) is stated by (L1).

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

Q.E.D.

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

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

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

According to Scott[7] a subset \(X \subseteq L\) is called directed if every finite subset of \(X\) has an upper bound (in the sense of \(\preceq\)) belonging to \(X\). An obvious example of a directed subset is a non-empty ascending chain. A function \(f: D \rightarrow D\) is called continuous if whenever \(X \subseteq L\) is directed, then \(f(\bigcup\{x \mid x \in X\}) = \bigcup\{f(x) \mid x \in X\}\).

(H4): Let \(F\) be a continuous function from the complete semi-lattice \(<L, \preceq, \tau, \iota>\) in itself.

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

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

(L4): H4(H4') implies that \(F\) (\(\overline{F}\)) has a least fixpoint \(\ell(\overline{F})\) (\(\overline{\ell}(F)\)) which is the limit \(\ell = F^\omega(\ell)\) of the Kleene's sequence \(\ell \preceq F(\ell) \preceq \ldots \preceq F^n(\ell) \preceq \ldots\)

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