Conference Record
of the
FOURTH ACM SYMPOSIUM ON
PRINCIPLES OF 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 * 17 may be understood to denote computations on the...
### 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:
  
  $$\text{Env} = \text{Ident}^0 \times \text{Values}$$

  - We assume that the meaning of an expression $e$ in the environment $\epsilon \in \text{Env}$ is given by $\text{val} \upharpoonright \text{Expr} \downarrow (\epsilon)$ so that:
    
    $$\text{val} : \text{Expr} \rightarrow ([\text{Env}] \rightarrow \text{Values})$$

  In particular the projection $\text{val} \upharpoonright \text{Bexpr}$ of the function $\text{val}$ in domain $\text{Bexpr}$ has the functionality:
    
    $$\text{val} \upharpoonright \text{Bexpr} : \text{Bexpr} \rightarrow ([\text{Env}] \rightarrow \text{Booll})$$

  - The state set "States" consists of the set of all information configurations that can occur during computations:
    
    $$\text{States} = \text{Arcs}^0 \times \text{Env}$$

    A state $(s \in \text{States})$ consists in a control state $(\text{cs}(s))$ and an environment $(\text{env}(s))$, such that:
    
    $$\forall s \in \text{States}, s = (\text{cs}(s), \text{env}(s))$$

    - We use a continuous conditional function $\text{cond}(b, e_1, e_2)$ equal to $1$, $e_1$, $e_2$ or $\bot$ respectively as the value of $b$ is $1$, $\text{true}$, $\text{false}$ or $\bot$. We also use $\text{if } b \text{ then } e_1 \text{ else } e_2 \text{ fi}$ to denote $\text{cond}(b, e_1, e_2)$.

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

    - The state transition function defines for each state a next state (we consider deterministic programs):
      
      $$\text{n-state} : \text{States} \rightarrow \text{States}$$

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

      $$\text{let } n \text{ be } \text{end}(\text{cs}(s)), e \text{ be } \text{env}(s) \text{ within }$$

      $$\text{case } n \text{ in}$$

      $$\begin{align*}
      \text{Assignments} & \rightarrow \\
      \text{Tests} & \rightarrow
      \end{align*}$$

      $$\text{Junctions} =$$

      $$\text{Exe}$$

      $$\text{esac}$$

      (Each partial function $f$ on a set $S$ is extended to a continuous total function on the corresponding lattice $S^\mathbb{R}$ by $f(x) = f(\bot) = \top$ and $f(\top) = \bot$)
- A "computation sequence" with initial state $i_s \in I$-states is the sequence:
  \[ s_n = n\text{-state}^n(i_s) \] for $n = 0, 1, \ldots$
  where $f^0$ is the identity function and $f^{n+1} = f \circ f^n$.

- The initial to final state transition function:
  \[ n\text{-state}^\infty : \text{States} \to \text{States} \]
  is the minimal fixedpoint of the functional $\cdot^n$.

Since the equation $Cy(r) = n\text{-context}(r, Cy)$ must be valid for each arc, $Cy$ is a solution to the system of "forward" equations:
\[ Cy = F\text{-cont}(Cy) \]
where
\[ F\text{-cont} : \text{Context-Vectors} \to \text{Context-Vectors} \]
is defined by:
\[ F\text{-cont}(Cy) = \lambda r . n\text{-context}(r, Cy) \]
Context-Vectors is a complete lattice with union $\sqcup$ such that $Cy, \sqcup Cy_2 = \lambda r . (Cy(r) \sqcup Cy_2(r))$. 
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 = \( A \circ \circ \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarro
Instead of the global hypothesis 6.0 we will use the following local hypothesis on the concrete and abstract interpretations of primitive language constructs:

\[
\begin{align*}
\forall (a, \tilde{x}) \in \text{ArCs} \times \tilde{A} - \text{Cont}, \\
\gamma(\text{Int}(a, \tilde{x})) \geq \text{Int}(a, \gamma(\tilde{x}))
\end{align*}
\]

6.5 \hspace{1cm}
\[
\forall (a, x) \in \text{ArCs} \times \tilde{C} - \text{Cont,} \\
\text{Int}(a, \alpha(x)) \geq \alpha(\text{Int}(a, x))
\]

These two hypotheses are in fact equivalent (Lemma 1.2 in Appendix 12). The following schema illustrates 6.5, i.e. the idea of abstract simulation of concrete computations:

```
\begin{align*}
\tilde{C}_1 & \rightarrow_{\tilde{C}} \tilde{C}_0 \\
\gamma & \rightarrow \gamma' \\
\tilde{C}_1 & \rightarrow_{\alpha} \tilde{C}_0 \\
\text{Int}(a, \tilde{C}_1) & \rightarrow \text{C}_0
\end{align*}
```

Suppose we want to compute the concrete output context \(C_0 \) (associated with arc \(a \)) resulting from concrete input contexts \( \tilde{C}_1 \). We can as well approximate this computation in the abstract universe, and get \( C'_0 = \gamma(\text{Int}(a, \alpha(\tilde{C}_1))) \). 6.5 requires \( C'_0 \) to contain at least \( C_0 \), that is \( C_0 \subseteq C'_0 \). On the contrary we do not require \( C'_0 \) to contain at most \( C_0 \), that is \( C'_0 \subseteq C_0 \) is not compulsory.

We will say that \( I \) is a refinement of \( \tilde{I} \), or that \( \tilde{I} \) is an abstraction of \( I \), denoted \( I \leq (\alpha, \gamma)\tilde{I} \), if and only if there exist \( \alpha \) and \( \gamma \) satisfying hypothesis 6.1 to 6.3.

Note that \( I \leq (\alpha, \gamma)\tilde{I} \) imposes a local consistency of the interpretations \( I \) and \( \tilde{I} \), at the level of primitive language constructs (6.5). Theorems T1 and T2 of Appendix 12 then prove 6.0 which defines the global consistency of \( I \) and \( \tilde{I} \) at the program level.

In particular if we take

\[ I_{SS} = \langle \text{Contexts, } \alpha, \gamma, \text{Env, } \emptyset, \alpha - \text{context} \rangle \]

any abstract interpretation \( \tilde{I} \) of \( P \), consistent with \( I_{SS} \), \( (I_{SS} \leq (\alpha, \gamma)\tilde{I}) \) is consistent with the semantics of \( P \), which implies:

\[ \forall q \in \text{ArCs, let } \tilde{C} \text{ be the result of } \tilde{I}, \]

\[ \exists n \in \emptyset, \exists i_i \in \text{I-states } | \langle q, i_i | = \text{n-state}^n(i_i) \rangle \]

As previously noticed, the abstract interpretations...
where \( n\text{-pred} \) defines Floyd\[67]\'s strongest post condition:

\[
\text{n-pred}(r, p_v) = \text{let}(s \text{ be origin}(r), \langle p \text{ be a-pred(origin}(r))\rangle \text{within case } n \text{ in } \begin{align*}
\text{Entries} & \quad \Rightarrow (x \in \text{Ident}, x = i_v) \text{Values} \\
\text{Junctions} & \quad \Rightarrow (p_v(x)) \\
\text{Tests} & \quad \text{case } r \in \begin{cases} 
(a, \text{succ}(t)) \Rightarrow p_v(p) \text{ and } test(n) \\
(a, \text{succ}(t)) \Rightarrow p_v(p) \text{ and } \neg test(n)
\end{cases}
\text{esac}
\text{Assignments} & \quad \begin{cases} 
(p \text{ be } p_v(p)), (x \text{ be } x) \text{within } \\
(e \text{ be } e(x)) \text{ within }
\end{cases}
\text{esac}
\]

The "invariants" of the program are defined by the least fixpoint of \( n\text{-pred} \) (least for ordering \( \subseteq \) \( \Rightarrow \)), so that an invariance implies any other correct assertion.

The deductive semantics is easily validated by proving that \( I_{GS} \leq (\alpha, \gamma) I_{GS} \) where:

\[
\alpha : \text{Contexts } \rightarrow \text{Pred} \\
= \lambda C. (\text{or } (x = e(x)) \text{ or } (x \in \text{Ident}))
\gamma : \text{Pred } \rightarrow \text{Contexts} \\
= \lambda p . \text{let } (x / e(x), x \in \text{Ident})
\]

The main point is to justify Hoare\[67]\'s proof rules by showing:

\[
\{\forall a \in \text{Arcs}, p_v \in \text{Pred} \} \quad (a(n\text{-context}(a, \gamma(p_v))) \Rightarrow \text{n-pred}(a, p_v))
\]

See Hoare and Lauer\[74,75], Ligler\[75]. In particular Ligler\[75] shows clearly that the proof can be done only when considering realizable Contexts and programs involving "clean" basic constructs (e.g. contexts excluding non-termination, errors, side-effects, sharing between identifiers, ...).

Once \( I_{GS} \leq (\alpha, \beta) I_{GS} \) has been proved, we know that the deductive semantics gives a valid proof technique, which will never permit a false theorem to be deduced:

\[
\forall q \in \text{Arcs}, \forall p_v \in \text{Pred} \\
\{n \geq 0, \exists i_s \in I\text{-states} \mid q, e = n\text{-state}(i_s) \} \\
\Rightarrow (p_v(q) \Rightarrow a(e))
\]

7. The Lattice of Abstract Interpretations

The relation \( S \) comparing the levels of abstraction of two interpretations is a quasi-ordering since it is:

reflexive: \( (I \leq (1, 1)) \) where \( \gamma = \lambda x. x \) is the identity function,

transitive: \( (I \leq (\alpha_1, \gamma_1)(I') \) and \( (I' \leq (\alpha_2, \gamma_2)(I'') \) imply \( I \leq (\alpha_1 \circ \alpha_2, \gamma_2 \circ \gamma_1)(I'') \).
A further abstraction may be:
\[
\alpha((a, b)) = \begin{cases} 
\text{if } a \neq b \text{ then } a \text{ else } 0 
\end{cases} + \begin{cases} 
\text{else } \text{if } a > 0 \text{ then } + 
\text{else } \text{if } 1 < n \text{ then } n 
\end{cases},
\]
\[\gamma(+) = [0, +\infty], \gamma(-) = [-\infty, 0], \gamma(\cdot) = [-\infty, +\infty].\]
The abstract contexts are then:

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

\[I_{CP} = \text{abstracted representation}\]
\[I_{RS} = \text{abstracted representation}\]

\[I_{CS} = \text{abstracted representation}\]

\[I_{I} = \text{abstracted representation}\]

8. Abstract Evaluation of Programs

The system of equations:
\[CV : \text{Int}(CV)\]
resulting from an interpretation \[I = \{A\text{-Cont}, \leq, T, i, \text{Int}\}\] 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 (L.4 of Appendix 12):
\[CV := (C := 1; \text{until } C = \text{Int}(C) \text{ do } C := \text{Int}(C) \text{ repeat } C)\]

8.1 Correctness

If \(\text{Int}\) is supposed to be a complete morphism (i.e. infinitely distributive over \(\cdot\)) then \(CV\) is the least fixpoint of \(\text{Int}\), (e.g. Kildall[73]), 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 \(CV\) 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 isotone function is continuous). Finally, if \(\text{Int}\) is only supposed to be isotone, \(CV\) is an approximation (\(\approx\)) of the least fixpoint (e.g. Knu and Ullman[75]).

8.2 Termination

The abstract evaluation terminates iff Kleene's sequence is finite. This may be the case because \(A\text{-Cont}\) is finite (e.g. type checking in ALGOL 60, Nauf[68]), or a finite subset only is to be considered for any particular program (e.g. type checking in ALGOL 68), or \(A\text{-Cont}\) may be of finite length \(m\) (the length of any strictly increasing chain is bounded by \(m\), Kildall[73], Wegbreit[75]) or \(A\text{-Cont}\) 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 [73]) or heuristics (Katz and Manol[76]) may be used to pass to the limit, or approximate it, (Cousot[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 \( A^\text{-Cont} \) be the lattice \( \mathbb{R}^+ \) of positive real numbers augmented by the upper bound \( \infty \), with natural ordering \( \leq \). The abstract interpretation:

\[
\mathcal{I}_p = \langle \mathbb{R}^+, \max, \leq, 0, \infty, \text{Kir} \rangle
\]

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

\[
\text{Kir}(r, Cv) =
\begin{align*}
& \text{let } n \text{ be } \text{origin}(r) \text{ within } \\
& \text{case } n \text{ in } \\
& \quad \text{Entries } \Rightarrow 1 \text{ (unique entry node)} \\
& \quad \text{Junctions } \cup \text{Assignments } \Rightarrow \sum_{\pi} \text{Kir}(\pi) \\
& \quad \text{Tests } \Rightarrow \\
& \quad \text{case } r \text{ in } \\
& \quad \quad \{a = \text{suc}(r(n)) \} \Rightarrow \text{Kir}(a = \text{suc}(r(n)) \ast \\
& \quad \quad \text{Prob}(\text{test}(n) = \text{true})
\end{align*}
\]

9. Fixpoint 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 \( I' (1 \leq I) \) may be used for that purpose (e.g. Temenbaum[74]). 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 \( I = \langle A^\text{-Cont}, \circ, \leq, 1, \tau, \text{Int} \rangle \) 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 if \( \text{Ub} \leq \gamma(Cv) \) and \( Cv \leq \text{Ub} \) implies \( Cv \leq \gamma(\text{Ub}) \).

8.1.1 Increasing Approximation Sequence
9.1.2 Generalization of Kleene's Ascending Sequence

As before, we define:

\[ A^{-1} \text{Int} = \lambda y \left( A^{-1} \text{Int}(d, c, y) \right) \]
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:

\[
\begin{align*}
(0) & \quad C0 = \{x \geq 1\} \\
(1) & \quad C1 = \{x = 1\} \\
(2) & \quad C2 = C1 \cup C4 \\
(3) & \quad C3 = C2 \cap \{x \geq 100\} \\
(4) & \quad C4 = C3 \times \{x = 1\} \\
(5) & \quad C5 = C2 \cap \{x \geq 101, \infty\}
\end{align*}
\]

Assignment statements are treated using an interval arithmetic (e.g. \([i, j] + [k, l] = [i+k, j+l]\) naturally extended to include the case of the empty interval). Similarly tests are treated using an "interval logic". Since there exist infinite Kleene's sequences (e.g. \([1, \infty) \times [0, 0] \subset [0, 1] \times ... \times [0, 1] \times \{x \geq 100\}\), we must use an approximation sequence. Hence the results will be somewhat inaccurate but runtime subtyping checks may be inserted.

\* \(C3 = \{1, 100\}\)  
\* \(C4 = C3 \times \{1, 1\}\)  
\* \(C5 = C2 \cap \{x \geq 101, \infty\}\)  

Note: \(C1 \cup C4 = \{1, 101\} \leq C2 = \{1, \infty\}\) stop on that path.  
\(C5 = C2 \cap \{x \geq 101, \infty\}\)  
\(= \{1, \infty\} \cap \{101, \infty\}\)  
\* \(C5 = \{101, \infty\}\) exit, stop.

The final context on each arc is marked by a star \(\ast\). Note that the results are approximate ones, (e.g. \(C5\)).

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 \(S_m = A^{-\text{Int}^m}(x)\) of the least fixpoint \(C_f\) of \(\text{Int}\). Moreover \(\text{Int}^n(S_m) \leq S_m\). Since \(\text{Int}\) is order preserving, this implies that:

\[
S_0 \leq \text{Int}(S_1) \leq ... \leq \text{Int}^n(S_m) \leq \text{Int}^m(S_m) \leq \text{Im}. 
\]

If \(S_m\) is not a fixpoint of \(\text{Int}\) and the above descending sequence is finite (e.g. the lattice A-Cont satisfies the descending chain condition) its limit is a better approximation of \(C_f\) than \(S_m\).
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound. The limit of the descending sequence \( S'_n = \tilde{t}, \ldots \), \( S'_n = \operatorname{Int}^n(i) \), \ldots \) is an upper bound of the greatest \( \lim S'_n \).

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

9.3.2.1 \( \forall c \in \mathcal{A-Cont}, \ operatorname{Int}(C) \Rightarrow (C \geq \operatorname{D-Int}(C)) \geq \operatorname{Int}(C) \)

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

The truncated decreasing sequence \( S'_0, \ldots, S'_n, \ldots \) is recursively defined by:

9.3.2.3 \( S'_0 = S_m \)

\[ S'_n = \begin{cases} S'_{n+1} & \text{if } (S'_n \not\in \operatorname{Int}(S'_n)) \text{ and } (S'_n \not\in \operatorname{D-Int}(S'_n)) \\ S'_n & \text{otherwise} \end{cases} \]

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 \( \operatorname{D-Int} \) by local modifications to \( \operatorname{Int} \):

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

9.3.4.2 \( \forall (C, C') \in \mathcal{A-Cont}^2, \) \( (C \geq C') \Rightarrow (C \geq C \Delta C' \geq C') \)

9.3.4.3 Every infinite sequence \( s_0, s_1, s_2, \ldots \) of the form \( s_0 = C_0, s_1 = s_0 \Delta C_1, \ldots \)
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:

\[(2) \quad C_2 = C_2 \land (C_1 \cup C_4)\]

The descending approximation sequence is:

- \[C_2 = C_2 \land (C_1 \cup C_4)\]
- \n\n\[= [1, +\infty) \land ([1, 1] \cup [2, 101])\]
- \n\n\[= [1, +\infty) \land [1, 101]\]
- \n\n\[\ast \quad C_2 = [1, 101]\]
- \n\n\[C_3 = C_2 \cap [-\infty, 100]\]
- \n\n\[\ast \quad C_3 = [1, 101] \cap [-\infty, 100] = [1, 100]\]
- \n\n\[\text{stop on that path.}\]
- \n\n\[C_5 = C_2 \cap [101, +\infty]\]
- \n\n\[\ast \quad C_5 = [1, 101] \cap [101, +\infty] = [101, 101]\]
- \n\n\[\text{exit.}\]

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 \(\mathbb{A}_{	ext{cont}}\) may be partitioned as follows:

- \(X \land \text{Int}(X)\)
- \(\text{non comparable}\)
- \(X \lor \text{Int}(X)\)
- \(X \lor \text{Int}(X)\)
- \(X = \text{Int}(X)\)
- \(\text{AKS} \quad \text{lfp} \quad \text{gfp} \quad \text{DKS}\)

When \(\tilde{X} \geq Y\) we have noted \(X \longrightarrow \vdots \longrightarrow Y\).

The truncated descending sequence TDS is fundamentally different from AAS, since it ensures that the successive approximations starting from \(S_0\) remain in the partition \(\{X | X \geq \text{Int}(X)\}\), so that their limit \(S^*\) is greater than \(\text{lfp}\):

\[\text{TDS} \quad \text{Int}(X) \quad \text{TDS} \quad \text{TDS}\]

It is clear that the ascending approximation sequence AAS when starting from 1 leads to an upper bound of the least fixpoint \(\text{lfp}\) of \(\text{Int}\), and the truncated descending sequence TDS when starting.
Any of the AAS, TDS, DAS, TAS methods may yield a fixpoint \( fp \) which is not the fixpoint \( \tilde{fp} \) or \( gfp \) of interest. None of these methods can improve \( fp \) to reach \( \tilde{fp} \) 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-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 \( n - 1, n - 2, \ldots, n - k \). 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 B-int could be more refined.

Finally, going deeply into the comparism 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], Morei and Renoise[76], Schwartz[75], Ullman[75], Wegbreit[75], ...), type discovery (Couso[76], Sintozoff[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], Ligler[75], Manna and Shamir[75], ...), verification of program correctness (Floyd[67], Park[69], Sintozoff[75], ...), discovery of inductive invariants (Katz and Manna[76], ...), proofs of program termination (Sintozoff [76], ...), program transformation (Sintozoff [76], ...).

Acknowledgements

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

11. References


Couso[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.


(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).

- \( g \preceq \alpha(g) \)
- \( \overline{g} \preceq \alpha(g) \) trivially
- \( g \preceq \alpha(F(g)) \) since \( g = F(g) \)
- \( \overline{g} \preceq \alpha(\overline{F}(g)) \) \( H3.1 \), \( \cup \) isotone, \( \preceq \) transitive
- \( g \preceq \overline{\alpha(g)} \) \( L3 \)
- \( \gamma(\overline{g}) \preceq \gamma(\alpha(g)) \) \( H2.4 \)
- \( \gamma(\overline{g}) \preceq \overline{g} \) \( H2.6 \), \( \preceq \) transitive.

Q.E.D.

Replacing \( <g, \overline{g}, \preceq, \leq, \geq, F, \overline{F}, \alpha, \gamma, H3.1, H2.4, H2.6> \) respectively by \( <\overline{g}, \preceq, \alpha, \gamma, H3.1, H2.4, H2.6> \) 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:

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

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 \( f : D \rightarrow D \) 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 \( <L, \cup, \preceq, \tau, I> \) in itself.

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

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

(L4): \( H_4(H4) \) implies that \( F, \overline{F} \) has a least fixpoint \( \overline{\ell}(\overline{\ell}) \) which is the limit \( \cup \equiv F^n(\overline{\ell}) \) of the Kleene's sequence \( I \leq F(I) \leq \cdots \leq F^n(I) \)

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