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

Abstract program properties are modeled by a com-

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 $\bot \leq x \leq \top$, 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 $\text{expr} \in \text{Expr}$ in the environment $e \in \text{Env}$ is given by $\text{val}[e\text{expr}] (e)$ so that:
  $\text{val} : \text{Expr} \rightarrow [\text{Env} \rightarrow \text{Values}].$
- A "computation sequence" with initial state $i_0 \in I$-states is the sequence :
  \[s_n = n\text{-state}^n(i_0)\] for $n = 0, 1, \ldots$
  where $\text{id}$ is the identity function and $\text{id}^n f = f \circ \text{id}^{n-1} f$.

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

\[C_q \in \text{Contexts} = \sum_{e \in \text{Env}} \{ e \mid \exists n \geq 0, \exists i_0 < I\text{-states} \mid (q, e) = n\text{-state}^n(i_0)\}\]

The context vector $C_v$ associates a context to each of the program points of a program:

\[C_v : \text{Context-Vectors} \to \text{Arcs}^0 \to \text{Contexts}\]

\[C_v = \lambda q. \{ e \mid \exists n \geq 0, \exists i_0 < I\text{-states} \mid (q, e) = n\text{-state}^n(i_0)\}\]

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 $q$ which is a subarc of $r$. 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:

\[F\text{-cont}(C_v) = \lambda q. n\text{-context}(r, C_v)\]

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

$F\text{-cont}$ is order preserving for the ordering $\sqsubseteq$ of Context-Vectors which is defined by:

\[C_v_1 \sqsubseteq C_v_2 \iff \forall r \in \text{Arcs}, C_v_1(r) \subseteq C_v_2(r)\]

Hence it is known that $F\text{-cont}$ 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 $\sqsubseteq$ to the system of equations $X = F\text{-cont}(X)$, ($C_v \sqsubseteq X$). Tarski [55] shows that this property uniquely determines $C_v$ as the least fixpoint of $F\text{-cont}$. Thus $C_v$ can be equivalently defined by:

\[D_1 : C_v = \lambda q. \{ e \mid \exists n \geq 0, \exists i_0 < I\text{-states} \mid (q, e) = n\text{-state}^n(i_0)\}\]

or

\[D_2 : C_v = \text{Y } \lambda F. (F\text{-cont} \circ F)\]

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 $P$:

\[\{ e \mid \exists n \geq 0, \exists i_0 < I\text{-states} \mid (q, e) = n\text{-state}^n(i_0)\}\]

\[\Rightarrow \{ e \mid C_v(q) \}\]

(2) $C_v(q)$ contains only the environments $e$ which may be associated to $q$ during an execution of $P$:

\[\{ e \mid C_v(q) \}\Rightarrow \{ e \mid \exists n \geq 0, \exists i_0 < I\text{-states} \mid (q, e) = n\text{-state}^n(i_0)\}\]

$C_v$ is merely a static summary of the possible executions of the program. However, our definitions $D_1$
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
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:
\[
I \equiv (\beta)I' \iff \{ (I \leq (\beta, \beta^{-1})I') \text{ and } (I' \leq (\beta^{-1}, \beta)I) \}
\]

2 - reciprocally,
If \( I \leq (\alpha, \gamma_1)I' \), let \( \equiv (\alpha) \) be the equivalence relation defined on \( I \) (properly speaking, on the set of abstract contexts of \( I \)) by:
\[
\{ x \equiv (\alpha_1)y \} \iff \{ \alpha_1(x) = \alpha_1(y) \}
\]
\( \forall x' \in I' \), each equivalence class \( C_x = \{ x \in I | \alpha_1(x) = x' \} \) has a least upper bound which is \( \gamma_1(x') \). Hence the projection \( \alpha \mid \gamma_1(I') \) of \( \alpha \) on \( \gamma_1(I') \) is a bijection from the set \( \gamma_1(I') \) of representatives of the equivalence classes on \( I \).
A further abstraction may be:
\[ a((a, b)) = \begin{cases} \text{if } a = b \text{ then } a \text{ else if } a \geq b \text{ then } + \\
\text{else if } a \leq b \text{ then } - \end{cases} \]
\[ f(L, \gamma(x)) = \langle n, m \rangle. \]
\[ \gamma(-) = [0, +\infty], \gamma(0) = [-\infty, 0], \gamma(\pm) = [-\infty, +\infty]. \]

The abstract contexts are then:

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

\[ I_{CP} \]

\[ I_{RS} \]

\[ I_{CS} \]

\[ I_{IR} \]

\[ I_{I} \]

8. Abstract Evaluation of Programs

The system of equations:
\[ CV : \text{Int}(CV) \]
resulting from an interpretation \( I = \langle \text{A-Cont, } \leq, \\tau, I, \text{Int} \rangle \) 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):
\[ CV := (C := I; \text{ until } C = \text{Int}(C) \text{ do } C := \text{Int}(C) \text{ repeat } C) \]

8.1 Correctness

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

8.2 Termination

The abstract evaluation terminates if 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 (Sinzoff[75]) or heuristics (Katz and Manna[76]) may be used to pass to the limit, or approximate it, (Courot[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 \( \omega \), with natural ordering \( \leq \). The abstract interpretation:

\[
I_p = \langle \mathbb{R}^+, \text{max}, \leq, 0, \omega, \text{Kle}\rangle
\]

may be used to derive the mean values of the counters using fixed-point iteration of fixed-point methods.

9. Fixpoints Approximation Methods

When the extreme fixpoints of the system of equations associated with an abstract interpretation \( I \) of a program \( P \) cannot be computed in finitely many steps, they can be approximated. A more abstract interpretation \( I_1 (1 \leq 1) \) may be used for that purpose (e.g., Tennenbaum[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
9.1.2 Generalization of Kleene's Ascending Sequence

When A-Cont satisfies the ascending chain condition one can construct 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:

\[\{c \in \text{A-Cont}, c \in \text{A-Cont} \mid \text{Int}(q, C, C) \neq \text{Int}(q, C, C/r)\}\]

\((\text{e.g., in a forward system of equations the context associated to } q \text{ may only depend on the contexts associated with the immediate predecessor arcs of } q).\)

In the system of equations \(C q = \text{Int}(C q)\) we define a cycle to be a sequence \(q_1, ..., q_n\) of arcs, such that \(q_1 \in \{1, m\}\), \(C q_1(q_2)\) depends on \(C q_1(q_2)\) and \(C q_2(q_1)\) depends on \(C q_1(q_2)\). \(\text{(e.g., in a forward interpretation a cycle corresponds to a loop in the program.)}\)

In any infinite strictly increasing Kleene's sequence \(C q_1, ..., C q_n\), since Arcs is finite there is some arc \(q\) for which the sequence \(C q(q), ..., C q_n(q)\), never stabilizes. Therefore \(q\) must belong to a cycle or the contexts associated to \(q\) transitively 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 \(\vee\) called widening defined by:

9.1.1.1 \(\forall : \text{A-Cont} \times \text{A-Cont} \rightarrow \text{A-Cont}\)

9.1.1.2 \(\forall(C, C') \in \text{A-Cont}^2, C \circ C' \subseteq C \vee C'\)

9.1.1.3 Every infinite sequence \(S_n, ..., S_0\) of the form \(S_n = C q_0, ..., S_0 = C q_n\), \(\text{where } C q_0, ..., C q_n\), \(\text{are arbitrary contexts) is not strictly increasing.}\)

- The set \(W\)-arcs of widening arcs, which is one of the minimal sets of arcs such that any cycle \(q_1, ..., q_n\) of the system of equations \(C q = \text{Int}(C q)\) contains at least a widening arc: \(\exists i \in \{1, n\} \mid q_i \in W\)-arcs. \(\text{(e.g., in a forward interpretation on a reducible program graph, W-arcs may be chosen to be the set of exit arcs of the junction nodes which are interval headers. On irreducible graphs an arbitrary choice has to be made so that any loop of the program goes through a widening arc.)}\)

- The approximate interpretation A-\text{Int} = A-\text{Cont} defined by:

9.1.3.4 A-\text{Int} = \lambda(q, C q). \text{if } q \in W\text{-arcs then } C q(q) \vee \text{Int}(q, C q) \text{ else } \text{Int}(q, C q)\)}

As before, we define:

9.1.3.5 A-\text{Int} = \lambda q. A-\text{Int}(q, C q)\)

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 \(\langle S_n, q \rangle = \hat{S}_n, ..., S_n, \text{A-Int}(q)\) be a widening sequence. We show that this sequence is increasing that is to say:

9.1.3.6 \(S_n \subseteq A-\text{Int}(S_n), \forall n \geq 0.\)

Trivially for \(n = 0\), \(S_0 = \hat{S}_n \subseteq A-\text{Int}(S_0)\). For the induction step, suppose the result to be true for \(n \leq m\). Let us prove that:

\(S_m \subseteq A-\text{Int}(S_m)\)

\(\iff\) \(S_{m+1} \subseteq A-\text{Int}(S_{m+1})\) \(\forall q \in \text{Arcs}\).

If \(q \in W\text{-arcs},\) then:

\(A-\text{Int}(q, S_{m+1}) = S_{m+1}(q) \vee \text{Int}(q, S_{m+1}) \supseteq S_{m+1}(q) \supseteq S_m(q).\)

If \(q \notin W\text{-arcs},\) then:

\(A-\text{Int}(q, S_{m+1}) = \text{Int}(q, S_{m+1}) \supseteq \text{Int}(q, S_{m+1}) \subseteq A-\text{Int}(q, S_{m+1}) \subseteq \text{Int}(q, S_{m+1}).\)

Finally \(S_{m+1} \subseteq A-\text{Int}(S_{m+1})\), Q.E.D.

An infinite sequence \(S_0 = \hat{S}_n, ..., S_n = A-\text{Int}(S_n)\) cannot be strictly increasing since otherwise there would exist some widening arc \(q\) for which the sequence \(S_0(q), ..., S_n(q)\) would never stabilize thus contradicting 9.1.3.3.

We now prove 9.1.1.7 that is to say that:

\(\forall n \geq 0, S_n = A-\text{Int}(S_n)\)

implies:

\(\forall n \geq 0, S_n = A-\text{Int}(S_n)\)

\(\iff\) \(\langle S_n = \hat{S}_n, ..., S_{n+1} = A-\text{Int}(S_n), \forall q \in \text{Arcs}\rangle\)

\(\iff\) \(S_{n+1}(q) = \text{Int}(q, S_{n+1}) \subseteq A-\text{Int}(q, S_{n+1})\) \(\text{(see 9.1.3.5)}\)

If \(q \in W\text{-arcs},\) we have:

\(A-\text{Int}(q, S_n) = S_n(q) \vee \text{Int}(q, S_n) \supseteq S_n(q) \supseteq S_n(q) \supseteq \text{Int}(q, S_n)\) by 9.1.3.2. If now \(q \notin W\text{-arcs}\) we must show:

\(S_n(q) = \text{Int}(q, S_n) \supseteq \text{Int}(q, S_n)\)

\(\iff\) \(S_n(q) = \text{Int}(q, S_n) \supseteq \text{Int}(q, S_n)\) \(\iff\) \(S_n(q) = \text{Int}(q, S_n)\) \(\iff\) \(S_n(q) = \text{Int}(q, S_n)\) \(\text{by 9.1.3.4 which is true, from 9.1.3.6, Q.E.D.}\)

9.2 Example: Bounds of Integer Variables

In a PASCAL program operating on arrays, the compiler should ensure that arrays are subscripted only by indices within bounds. For that purpose one can use the lattice \(L_n\) of section 7. Let us take an obvious example.
Let us note \([a, b]\) where \(a \leq x \leq b\) the predicate \(a \leq x \leq b\). The system of equations corresponding to the example is:

1. \(C0 = \emptyset\)
2. \(C1 = [1, 1]\)
3. \(C2 = C1 \cup C4\)
4. \(C3 = [1, 100]\)
5. \(C4 = C3 \cup [1, 1]\)
   \(= [1, 100] \cup [1, 1]\)
6. \(C5 = [2, 101]\)

Note: \(C1 \cup C4 = [1, 101] \leq C2 = [1, +\infty]\)
stop on that path.
\(C5 = C2 \cap [101, +\infty]\) 
\(= [1, +\infty] \cap [101, +\infty]\)
\(C5 = [101, +\infty]\)
exit, stop.

The final context on each arc is marked by a star \(*\). 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.
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound on n).

Let $D\text{-int} : A\text{-cont} \rightarrow A\text{-cont}$ be such that:

9.3.2.1 \{∀C ∈ A\text{-cont}:
\[ (C ≥ \text{Int}(C)) \implies (C ≥ D\text{-int}(C)) ≥ \text{Int}(C) \]

9.3.2.2 \{∀C ∈ A\text{-cont}, every infinite sequence C, D\text{-int}(C), ..., D\text{-int}^n(C), ... is not strictly decreasing.

The truncated decreasing sequence $S_0, S_1, ..., S_n, ...$ is recursively defined by:

9.3.2.3 $S_0 = S, S_{n+1} = \begin{cases} S_n' & (S_n' \neq \text{Int}(S_n')) \text{ and } (S_n' \neq D\text{-int}(S_n')) \\ S_n & \text{otherwise} \end{cases}$

Let us now prove that the truncated decreasing sequence is a finite strictly decreasing chain which terms are greater than $CV$ 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:

9.3.4.1 $S_0 = S ≥ \text{Int}(S_0') ≥ CV$

If $p > 0$ then $S'_0 \neq \text{Int}(S'_0)$, therefore $S'_0 ≥ \text{Int}(S'_0)$. Then applying 9.3.2.1 we have:

$S'_0 ≥ D\text{-int}(S'_0) = S'_1 ≥ \text{Int}(S'_0) ≥ CV$

But 9.3.2.3 implies $S'_0 \neq \text{Int}(S'_0)$, hence:

$S'_0 > S'_1 ≥ \text{Int}(S'_0) ≥ CV$

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

$S_{k-1}' = S_k' ≥ \text{Int}(S_{k-1}') ≥ CV$

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

$\text{Int}(S_{k+1}') ≥ \text{Int}(S_k') ≥ \text{Int}(S_{k-1}') ≥ \text{Int}(CV)$

By transitivity $S_k' ≥ \text{Int}(S_k')$ and since 9.3.2.3 implies $S_k' \neq \text{Int}(S_k')$ we have from 9.3.2.1:

$S_k' ≥ D\text{-int}(S_k') = S_{k+1}' ≥ \text{Int}(S_k')$

Since 9.3.2.3 implies $S_k' \neq D\text{-int}(S_k')$ we have:

$S_k' ≥ S_{k+1}' ≥ \text{Int}(S_k') ≥ CV$

By recurrence on $k$ the result is true for $k ≤ p$.

Moreover, 9.3.2.2 implies that $n$ is finite.

The limit of the descending sequence $S_n' = t, ..., S_p' = D\text{-int}(S_1'), ...,$ is an upper bound of the greatest fixpoint of $\text{Int}$.

9.3.4.4 $D\text{-int} = \lambda(q, Cv). \begin{cases} \text{Int}(q) & \text{if } q ∈ W\text{-arcs} \text{ and } Cv(q) ≥ Int(q, Cv) \\ \text{else } \text{Int}(q, Cv) \end{cases}$

This definition of $D\text{-int}$ trivially satisfies the requirement 9.3.2.1 since $∀q ∈ W\text{-arcs with property } Cv(q) ≥ \text{Int}(q, Cv)$ implies $\text{Int}(q, Cv) ≥ \text{Int}(q, Cv)$. If $q ∈ W\text{-arcs}$ then 9.3.4.4 implies that $Cv(q) ≥ \text{Int}(q, Cv) ≥ D\text{-int}(q, Cv) ≥ \text{Int}(q, Cv)$. Otherwise, if $q ∉ W\text{-arcs} \text{ and } Cv(q) ≥ \text{Int}(q, Cv) = D\text{-int}(q, Cv)$. Hence $Cv ≥ D\text{-int}(Cv) ≥ \text{Int}(Cv)$.

The proof of termination (requirement 9.3.2.2) is very similar to the one outlined for $A\text{-cont}$ 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:

1. $C_1 = [1, 1]$
2. $C_2 = C_1 ∨ C_4$
3. $C_3 = C_2 ∧ [-∞, 100]$
4. $C_4 = C_3 + [1, 1]$
5. $C_5 = C_2 ∧ [101, ∞]$

The ascending approximation sequence led to the approximate solution:

$$0 \subseteq D\text{-int}[C_5] = [1, 1]$$
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) \( C_2 = C_2 \triangle (C_1 \cup C_4) \)

The descending approximation sequence is:

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

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

When \( X \succeq Y \) we have noted \( X \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow Y \).

The truncated descending sequence TDS is fundamentally different from AAS, since it ensures that the successive approximations starting from \( S_0 \) remain
Any of the AAS, TDS, DAS, TAS methods may yield a fixpoint \( fp \) which is not the fixpoint \( ffp \) or \( gfp \) of interest. None of these methods can improve \( fp \) to reach \( ffp \) 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 \( A\text{-int} \) and \( B\text{-int} \) 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], Morel and Renvoise[76], Schwartz[75], Ullman[75], Wegbreit[75], ...), type discovery (Cousot[76], Sintzoff[72], Tenenbaum[74], ...), program testing (Henderson [75], ...) symbolic evaluation of programs (Heivitt 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], 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 built 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.

Acknowledgments

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).


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

(H1): Let \( F \) be an order-preserving function from the complete semi-lattice \( \langle L, \cup, \leq, \top, 1 \rangle \) in itself.

(H1): Let \( \overline{F} \) be an order-preserving function from the complete semi-lattice \( \langle L, \cup, \leq, \top, 1 \rangle \) in itself.

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

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

We note \( \gamma \) and \( \delta \) the greatest and least fixpoints of \( \overline{F} \).

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

\[(H2.1) \quad \alpha : L \rightarrow L \]
\[(H2.2) \quad \gamma : L \rightarrow L \]
\[(H2.3) \quad \alpha \text{ is order preserving} \]
\[(H2.4) \quad \gamma \text{ is order preserving} \]
\[(H2.5) \quad \forall x \in L, \ x = \alpha(\gamma(x)) \]
\[(H2.6) \quad \forall x \in L, \ x \leq \gamma(\alpha(x)) \]

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

(H3.2): (H1), (H2), (H3.1) and \( \forall x \in L, \overline{F}(\alpha(x)) = \alpha(F(x)) \)
(T1): \(H_1, H_1, H_2, H_3\) imply that the greatest fix-points \(g\) and \(\overline{g}\) of \(F\) and \(\overline{F}\) are related by:

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

Proof:

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

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

Q.E.D.