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 \(-1515 \times 17 \implies -(-) \times (+) \implies (-) \times (+) \implies (-)\), 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. \( -1515 + 17 \implies -(-) + (+) \implies (-) + (+) \implies (\#)\)). 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

Section 3 describes the syntax and mathematical semantics of a simple flowchart language, Scott and Strachey[71]. This mathematical semantics is used in section 4 to build a more abstract model of the semantics of programs, in that it ignores the sequencing of control flow. This model is taken to be the most concrete of the abstract interpretations of programs. Section 5 gives the formal definition of the abstract interpretations of a program.

---

* Attatchede Recherche au C.N.R.S., Laboratoire Associé n° 7.

** This work was supported by IRIA-SESOIRI under grants 75-035 and 76-160.

3. Syntax and Semantics of Programs

We will use finite flowcharts as a language independent representation of programs.

3.1 Syntax of a Program

A program is built from a set "Nodes". Each node has successor and predecessor nodes: \( n\text{-succ}, n\text{-pred} : \text{Nodes} \rightarrow \text{Nodes} \mid (m \in n\text{-succ}(n)) \implies (n \in n\text{-pred}(m)) \).

Hereafter, we note \(|S|\) the cardinality of a set \( S \). When \(|S| = 1\) so that \( S = \{x\} \) we sometimes use \( S \) to denote \( x \).

The node subsets "Entries", "Assignments", "Tests", "Junctions" and "Exits" partition the set Nodes.

- An entry node \((n \in \text{Entries})\) has no predecessors and one successor, \(((n\text{-pred}(n) = \emptyset) \text{ and } |n\text{-succ}(n)| = 1))\).
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 $S^*$ the complete lattice obtained from $S$ by adjoining $\{\bot, \top\}$ to it, and imposing the ordering $\frac{1}{2} \leq x \leq \frac{1}{2}$ for all $x \in S$.

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

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

$$Env = Ident^0 \rightarrow Values$$

We assume that the meaning of an expression $expr$ in the environment $e \in Env$ is given by $val \Xi \text{expr} \Pi e$ so that:

$$val : Expr \rightarrow [Env \rightarrow Values].$$

In particular the projection $val | Bexpr$ of the function $val$ in domain $Bexpr$ has the functionality:

$$val | Bexpr : Bexpr \rightarrow [Env \rightarrow Bool].$$

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

$$States = \text{Arts}^\times Env.$$

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

$$\forall s \in States, s = \langle cs(s), env(s) \rangle.$$

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

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

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

$$n-state : States \rightarrow States$$

$$n-state(s) = \begin{cases} 
\text{let } n \text{ be end}\langle cs(s) \rangle, e \text{ be env}(s) \text{ within} \\
\text{Assignments} \rightarrow \\
\text{Tests} \rightarrow \\
\text{Junctions} \rightarrow \\
\text{Exits} \rightarrow s 
\end{cases}.$$
This implies that $A\text{-}\text{Cont}$ 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 $A\text{-}\text{Cont} = \text{Arcs}^0 \rightarrow A\text{-}\text{Cont}$.

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

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

$$Cv' \bowtie Cv'' = \{ \forall r \in \text{Arcs}^0, \; Cv'(r) \leq Cv''(r) \}$$

$$\hat{\land} = \lambda r. \top \quad \text{and} \quad \hat{\lor} = \lambda r. \bot$$

$<A\text{-}\text{Cont}, \bowtie, \leq, \hat{\land}, \hat{\lor}>$ can be shown to be a complete lattice. The function:

$$\text{Int} : \text{Arcs}^0 \times A\text{-}\text{Cont} \rightarrow A\text{-}\text{Cont}$$

defines the corresponding operation.

\[ \text{Examples:} \]

Kildeall[73] uses $(n, \rightarrow, \uparrow)$, Wegbreit[75] uses
The determination of available expressions, back-dominators, intervals, ... requires a forward system 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:

\[
\text{n-pred}(r, PV) = \begin{cases} 
\text{let}(x \text{ be origin}(r), (p \text{ be n-pred}(\text{origin}(r)))) \text{within case } n \text{ in } \\
\text{Entries} \implies (\forall x \in \text{Ident}, x = i \text{Values}) \\
\text{Junctions} \implies \text{or} (\text{P} \text{v}(q)) \\
\text{Tests} \implies \text{case } r \text{ in } \\
\{a = \text{succ}(t)(n) \implies \text{P} \text{v}(p) \text{ and test}(n)\} \\
\{a = \text{succ}(t)(n) \implies \text{P} \text{v}(p) \text{ and not test}(n)\} \\
\text{esac} \\
\text{Assignments} \implies \\
\begin{cases} 
(P \text{ be P} \text{v}(p)), (x \text{ be id}(n)), \\
(\text{e be expr}(n)) \text{ within } \\
(\forall x \text{ Values} \in [\text{P} \text{v} / x] \text{ and } x = e[x]) \\
\text{esac}
\end{cases}
\]

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

The deductive semantics is easily validated by proving that \( \text{I}_{DS} \subseteq (\alpha, \gamma) \text{I}_{DG} \) where:

\[
\alpha : \text{Contexts} \rightarrow \text{Pred} \\
\gamma : \text{Pred} \rightarrow \text{Contexts}
\]

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

\[
\{\forall x \in \text{Arcs}, \forall y \in \text{Pred}, \\
\alpha(\text{context}(a, y(PV))) \rightarrow \text{n-pred}(a, PV)\}
\]

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

Once \( \text{I}_{DS} \subseteq (\alpha, \beta) \text{I}_{DG} \) has been proved, we know that the deductive semantics gives a valid proof technique, which will never permit a false theorem to be deduced:

\[
\{\forall q \in \text{Arcs}, \forall y \in \text{Pred}, \\
\alpha(\text{context}(a, y(PV))) \rightarrow \text{n-pred}(a, PV)\} \\
\{\forall q \in \text{Arcs}, \forall y \in \text{Pred}, \\
\alpha(\text{context}(a, y(PV))) \rightarrow \text{n-pred}(a, PV)\} \\
\{\forall q \in \text{Arcs}, \forall y \in \text{Pred}, \\
\alpha(\text{context}(a, y(PV))) \rightarrow \text{n-pred}(a, PV)\}
\]

7. The Lattice of Abstract Interpretations

The relation \( \equiv \) 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: \((I \leq (a, \gamma_1) I') \text{ and } (I' \leq (a, \gamma_2) I'') \) imply \( I \leq (a, \gamma_1 \circ a, \gamma_2 \circ \gamma_1) I''\).

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:

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

The proof gives some insight in the abstraction process:

\[
1 \cdot \{I \equiv (\beta) I'\} \rightarrow \{I \leq (\beta, \beta^{-1}) I'\} \text{ and } \{I' \leq (\beta^{-1}, \beta) I\}\}
\]

2 - reciprocallly,

If \( I \leq (\alpha, \gamma_1) I' \text{, then } \{\alpha, \gamma_1\} \text{ be the equivalence relation defined on } I \text{ (properly speaking, on the set of abstract contexts of } I) \text{ by:}

\[
\{x \equiv (\alpha)(y)\} \iff \{\alpha(x) = \alpha(y)\}
\]

\( \forall x' \in I' \), each equivalence class \( \gamma_1 \text{ of } \alpha_1 \text{ on } \gamma_1(I') \text{ is a bijection from the set } \gamma_1(I') \text{ of representatives of the equivalence classes on } I \).

Let us show now that under the hypothesis \( I \leq (\alpha_1, \gamma_1) I' \text{ and } I' \leq (\alpha_2, \gamma_2) I' \text{, } \alpha_1 \text{ is bijective:}

\[
\alpha_1 \mid \gamma_1(I') \text{ and } \alpha_2 \mid \gamma_2(I') \text{ are bijections, hence } \\
\forall x' \in I' \text{, } \forall x \in I \text{ such that } x' = (\alpha_1 \mid \gamma_1(I'))(x). \text{ Likewise, } \forall x \in I \text{ such that } x' = (\alpha_2 \mid \gamma_2(I'))(x). \text{ Therefore, } x' \equiv (\alpha_1 \mid \gamma_1(I'))(x). \text{ Thus } \\
(\alpha_1 \mid \gamma_1(I'))(x) \equiv (\alpha_2 \mid \gamma_2(I'))(x). \text{ Hence } \gamma_1(I') \equiv \gamma_2(I') \text{ is a bijection between } I \text{ and } I'. \text{ Since } \gamma_2(I')^{-1} \text{ is a bijection between } I \text{ and } I_2 \text{, the composition } \\
(\alpha_1 \mid \gamma_1(I')) \circ (\alpha_2 \mid \gamma_2(I')) \circ (\alpha_2 \mid \gamma_2(I'))^{-1} = (\alpha_1 \mid \gamma_1(I')) \text{ is a bijection between } I \text{ and } I'. \text{ Hence } \alpha_1 \text{ is a bijection between } I \text{ and } I' \text{ which is trivially an algebraic isomorphism. } \}
\]

Let \( I \) be the set of abstract interpretations of a program, if equivalent interpretations are identified, the quasi-ordering \( \equiv \) becomes a partial ordering.

In particular, we can restrict \( I \) to be set of interpretations which abstract \( \equiv \) \( I \) is then a lattice (with ordering \( \subseteq \)) which is isomorphic with a subset of the lattice of equivalence relations on Contexts.

Example:

Let \( P \) be a program with a single integer variable, (the generalization is obvious). Environments will be integers (the value of the variable). Contexts are sets of integers (the set of values at some program point).

A context \( S \) may be abstracted by a closed interval \( \alpha(S) = [\min(S), \max(S)] \). When \( S \) is infinite the bounds will eventually be \( \alpha(S) = [\neg \infty, \infty] \). The abstract contexts are then, (Courat[76]):

\[
243
\]
$I = \left[ -2, 2 \right] \\
\left[ -2, 1 \right] \left[ -1, 2 \right] \\
\left[ -2, 0 \right] \left[ -1, 1 \right] \left[ 0, 2 \right] \\

\begin{tikzpicture}[auto]
    \node (I) {$I_I$};
    \node (CP) [above of=I] {$I_{CP}$};
    \node (CS) [right of=CP] {$I_{CS}$};
    \node (I1) [right of=CS] {$I_1$};
    \node (I2) [right of=I1] {$I_{SS}$};
    \path[->]
    (I) edge (I1)
    (I1) edge (I2)
    (I1) edge (CP)
    (I2) edge (CS)
    (CS) edge (I RS)
    (I RS) edge (I R)
    (I R) edge (I)
    (I R) edge (CP)
\end{tikzpicture}
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 $\text{A-Cont}$ be the lattice $\mathbb{R}^+$ of positive real numbers augmented by the upper bound $\infty$, with natural ordering $\leq$. The abstract interpretation:

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

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

$$\text{Kir}(r, CV) = \frac{1}{\text{section}(r)} \sum_{p \in \text{pred}(n)} \text{CV}(p)$$

Tests $\Rightarrow$

- case r in
  - \{a-succ-(n)\} $\Rightarrow$ CV(a-pred(n)) $\times$ \{Prob(test(n) = true)\}
  - \{a-succ-(n)\} $\Rightarrow$ CV(a-pred(n)) $\times$ (1-Prob(test(n) = true))

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

$$\text{CV}_0 \leq \text{CV}_1 \leq \ldots \leq \text{CV}_n \leq \ldots$$

then

$$\max \sum_{i=0}^{\infty} p \times a \text{- pred}(n) \times (\text{CV}_i(p))$$

and

$$\max \left( n \times q \right) = \left( \max \left( n \right) \right) \times q.$$

The least fixpoint of $\text{Kir}$ is the limit of Kleene's sequence (the length of the sequence is in general infinite):

- Let $P$ be the program "begin $L$ : go to $L$ end".
  - The number $n$ of iterations in the loop is ni-

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. An abstract interpretation $I_i$ ($1 \leq i$) may be used for that purpose (e.g., Temenbaum[74]). It is often better to make approximations in $I_i$ 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 $I = \langle \text{A-Cont}, \leq, 1, \tau, \text{Int} \rangle$ be an interpretation of $P$. When the least fixpoint $\text{CV}$ of $\text{Int}$ is unreachable, we look for an upper bound $\text{UB}$ of $\text{CV}$, since according to the correctness requirement 6.0, $\text{CV} \leq \gamma(\text{CV})$ and $\text{CV} \leq \gamma(\text{UB})$ implies $\text{CV} \leq \gamma(\text{UB})$.

9.1.1 Increasing Approximation Sequence

Let $\overline{\text{Int}}: \overline{\text{A-Cont}} \to \overline{\text{A-Cont}}$ be such that:

9.1.1.1 $(\forall n \geq 0, C = \overline{\text{Int}}(n)$ and not($\overline{\text{Int}}(C) \subseteq C)$)

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

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

9.1.1.3 $S_0 = I$

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

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

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

Let $m$ be the least natural number (eventually infinite) such that $S_m = S_{m+1}$. V€ $k \in [0, m]$, we know from 9.1.1.3 that not($\overline{\text{Int}}(S_k) \subseteq S_k$). Whence by definition of the ordering $\subseteq$, $S_k \not\subseteq \overline{\text{Int}}(S_k) \subseteq S_k$.

Since $S_k \subseteq \overline{\text{Int}}(S_k) \subseteq S_k$ is always true, we can state
9.1.2 Generalization of Kleene's Ascending Sequence

When A-Cont satisfies the ascending chain condition one can choose A-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-Int in 9.1.1 is global. We now indicate a way to construct A-Int by local modifications to Int.

Let \((q, r) \in Arcs\), we say that the context associated to \(q\) is dependent on the context associated to \(r\), if and only if:

\[ \exists \psi \in \text{Arcs}, \psi \subset (q, r) \]

In a forward system of equations the context associated to \(q\) may only depend on the context associated with the immediate predecessor arc of \(q\). In the system of equations \(C[q] = \text{Int}(C[q])\) we define a cycle to be a sequence \(<q_1, \ldots, q_n>\) of arcs, such that \(q_i \in [1, n]\) and \(C[q_{i+1}] = \text{Int}(C[q_i])\) depends on \(C[q_{i+1}]\) and \(C[q_i]\) depends on \(C[q_{i+1}]\). (e.g. in a forward interpretation a cycle corresponds to a loop in the program).

In any infinite strictly increasing Kleene's sequence \(C_{q_1}, \ldots, C_{q_n}\) since Arcs is finite there is some arc \(q\) for which the sequence \(C_{q_1}(q), \ldots, C_{q_n}(q)\), ... 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:

9.1.3.1 \(\forall : A-\text{Cont} \times A-\text{Cont} \to A-\text{Cont}\)

9.1.3.2 \(\forall(C, C') \subset A-\text{Cont}, C \circ C' \subset C \vee C'\)

9.1.3.3 Every infinite sequence \(S_q, \ldots, S_n\) of the form \(S_q = C[q], \ldots, S_n = C[q_n] \vee C[q_{n+1}], \ldots\) (where \(C[q], \ldots, C[q_n]\) 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 \(<q_1, \ldots, q_n>\) of the system of equations \(C[q] = \text{Int}(C[q])\) contains at least a widening arc: \(W \subset [1, n]\) \(q_i \in W\)-arcs. (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 internal 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 \(A-\text{Int} : A-\text{Cont} \times A-\text{Cont} \to A-\text{Cont}\) defined by:

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

As before, we define:

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

Now we have to show that this definition of \(\hat{A}-\text{Int}\) satisfies the requirements 9.1.1.2 and 9.1.1.7.

Let us consider a sequence \(s = \hat{S}_n = \hat{S}_{n+1} = \hat{S}_m, \ldots\) We show that this sequence is increasing that is to say:

9.1.3.6 \(\hat{S}_n = \hat{A}-\text{Int}(\hat{S}_n), \forall n \geq 0,\)

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

\(\hat{S}_{m+1} = \hat{A}-\text{Int}(\hat{S}_{m+1})\)

\(\Rightarrow \hat{S}_{m+1} = \hat{A}-\text{Int}(\hat{S}_{m+1})\), \(\forall q \in \text{Arcs}\).

If \(q \in W\)-arcs, then \(\hat{A}-\text{Int}(q, \hat{S}_{m+1}) = \hat{S}_{m+1}(q) \vee \text{Int}(q, \hat{S}_{m+1}) \geq \hat{S}_{m+1}(q) = \hat{S}_{m+1}(q).

If \(q \notin W\)-arcs, then \(\hat{A}-\text{Int}(q, \hat{S}_{m+1}) = \hat{A}-\text{Int}(q, \hat{S}_{m}) = \hat{A}-\text{Int}(q, \hat{S}_{m+1}) \leq \hat{A}-\text{Int}(q, \hat{S}_{m+1})

\(\Rightarrow \hat{S}_{m+1} = \hat{A}-\text{Int}(\hat{S}_{m+1})\), \(Q.E.D\).

An infinite sequence \(S_q = S_{q_1}, \ldots, S_{q_n} = \hat{A}-\text{Int}(S_{q_n})\), cannot be strictly increasing since otherwise there would exist some widening arc \(q\) for which the sequence \(S(q), \ldots, S_{q_n}\) would never stabilize thus contradicting 9.1.3.3.

We now prove 9.1.1.1 that is to say that:

\(\forall n \geq 0, \hat{S}_n = \hat{A}-\text{Int}(\hat{S}_n)\)

implying:

\(\hat{S}_n = \hat{A}-\text{Int}(\hat{S}_n)\)

\(\Rightarrow \hat{S}_n = \hat{A}-\text{Int}(\hat{S}_n)\), \(\forall q \in \text{Arcs}\)

\(\Rightarrow \hat{S}_n = \hat{A}-\text{Int}(\hat{S}_n)\)

If \(q \in W\)-arcs, we have \(\hat{A}-\text{Int}(q, \hat{S}_n) = \hat{S}_n(q) \vee \text{Int}(q, \hat{S}_n) \geq \hat{S}_n(q) = \hat{S}_n(q).

9.1.3.2. If now \(q \notin W\)-arcs we must show:

\(\hat{S}_n(q) = \hat{A}-\text{Int}(\hat{S}_n(q)) \leq \hat{A}-\text{Int}(\hat{S}_n(q))\)

\(\Rightarrow \hat{S}_n(q) = \hat{A}-\text{Int}(\hat{S}_n(q))\)

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 \(T^N\) of section 7. Let us take an obvious example:
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:

\[
\begin{align*}
C0 & : x := 1 \\
C1 & : C2 \rightarrow C3 \\
C2 & : x \leq 100 \\
C3 & : C0, \text{true} \\
C4 & : C0, \text{false} \\
C5 & : x := x + 1
\end{align*}
\]

\[
\begin{align*}
* & \quad C3 = [1, 100] \\
C4 = C3 + [1, 1] & \quad = [1, 100] + [1, 1] \\
* & \quad C4 = [2, 101] \\
& \quad \text{Note: } C1 \cup C4 = [1, 101] \leq C2 = [1, +\infty] \\
& \quad \text{stop on that path.} \\
C5 = C2 \cap [101, +\infty] & \quad = [1, +\infty] \cap [101, +\infty] \\
* & \quad C5 = [101, +\infty] \\
& \quad \text{exit, stop.}
\end{align*}
\]

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 infinites.
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound on n).

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

9.3.2.1 \( \forall C \in \text{A-Cont} \)
\[ (C \preceq \text{Int}(C)) \implies (C \succeq \mathcal{D}\text{-}\text{int}(C) \preceq \text{Int}(C)) \]

9.3.2.2 \( \forall C \in \text{A-Cont}, \) every infinite sequence \( C, \mathcal{D}\text{-}\text{int}(C), \ldots, \mathcal{D}\text{-}\text{int}^n(C), \ldots \) is strictly decreasing.

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

9.3.2.3 \( S'_0 = S_m \)
\[ S'_{n+1} = \begin{cases} S'_n & \text{if } (S'_n \neq \text{Int}(S'_n)) \text{ and } (S'_n \neq \mathcal{D}\text{-}\text{int}(S'_n)) \\ \mathcal{D}\text{-}\text{int}(S'_n) & \text{else} \end{cases} \]

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

Let \( p \) be the least natural number (eventually)

The limit of the descending sequence \( S'_0 = \tilde{t}, \ldots, S'_n = \mathcal{D}\text{-}\text{int}^p(t), \ldots \) 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 \( \mathcal{D}\text{-}\text{int} \) by local modifications to \( \text{int} \):

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

9.3.4.2 \( \forall (C, C') \in \text{A-Cont}^2, \) \[ (C \succeq C') \implies (C \succeq C \Delta C' \succeq C') \]

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 approximated interpretation

\( \mathcal{D}\text{-}\text{int} : \text{Arcs}^3 \times \text{A-Cont} \rightarrow \text{A-Cont} \) is defined by:

9.3.4.4 \( \mathcal{D}\text{-}\text{int} = \lambda (q, C_v). \) if \( q \in \mathcal{W}\text{-}\text{arc} \) then
\[ C_v(q) \triangle \text{Int}(q, C_v) \]
else
\[ \text{Int}(q, C_v) \]
fi
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:

\[
\begin{align*}
C_2 &= C_2 \triangle (C_1 \cup C_4) \\
&= [1, + \infty) \triangle ([1, 1] \cup [2, 10]) \\
&= [1, + \infty) \triangle [1, 101] \\
C_3 &= C_2 \cap [1, 101] \\
C_4 &= C_2 \cap [1, 101] \\
C_5 &= C_2 \cap [1, 101] \cap [101, + \infty] \\
C_6 &= C_2 \cap [1, 101] \cap [101, + \infty]
\end{align*}
\]

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 \( \text{Cont} \) may be partitioned as follows:

\[
\begin{align*}
X \text{ and } \text{Inf}(X) \\
\text{non comparable}
\end{align*}
\]

\[
\begin{align*}
X \preceq \text{Inf}(X) \\
X \preceq \text{Inf}(X)
\end{align*}
\]

\[
\begin{align*}
X = \text{Inf}(X) \\
\text{AKS} \rightarrow \text{fp}, \text{gfp} \\
\text{DRS} \rightarrow \text{fp}
\end{align*}
\]

\( \text{fp} \) and \( \text{gfp} \) are the least and greatest fixpoints of \( \text{Inf} \). The ascending (AKS) and descending (DRS) Kleene's sequences converge toward \( \text{fp} \) and \( \text{gfp} \) respectively. These limits are reached when \( \text{Inf} \) is continuous. When AKS is infinite we have proposed to use an ascending approximation sequence (AAS) to approximate \( \text{fp} \). Its limit may be some fixpoint \( \text{fp} \), or some \( S_m \) such that \( S_m \gg \text{Inf}(S_m) \) and \( S_m \gg \text{fp} \).

When \( X \preceq Y \) we have noted \( X \rightarrow \text{fp} \rightarrow Y \).

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 \( \{X \mid X \preceq \text{Inf}(X)\} \), so that their limit \( S' \) is greater than \( \text{fp} \):

\[
\begin{align*}
\text{fp} \rightarrow \text{gfp} \\
\text{TDS} \rightarrow \text{TDS}
\end{align*}
\]

It is clear that the ascending approximation sequence AAS when starting from \( \text{fp} \) leads to an upper bound of the least fixpoint \( \text{fp} \) of the \( \text{Inf} \), and the truncated descending sequence TDS when starting from \( \text{fp} \) leads to an upper bound of the greatest fixpoint \( \text{gfp} \). 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 \( \text{Inf} \):

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

249
Any of the AAS, TDS, DAS, TAS methods may yields 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 $A$-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 terms 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 $D$-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 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], 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 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.

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 Ulman[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.

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

\[ \iff \forall (x, y) \in L^2, \{ f(x \cup y) \geq f(x) \cup f(y) \} \]
(T1): $H_1$, $H_1$, $H_2$, $H_3$ imply that the greatest fixpoints $\alpha$ of $F$ and $\overline{F}$ are related by:

$$(\alpha(g) \preceq \overline{g}) \text{ and } (g, g) \gamma(\overline{g})$$

Proof:

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

$\overline{g} \preceq \alpha(g)$

$\overline{g} \preceq \alpha(F(g)) \preceq \alpha(g)$

$\overline{g} \preceq \overline{\alpha(g)} \preceq \alpha(g)$

$\overline{g} \preceq \alpha(g)$

$L3$

$\gamma(\overline{g}) \preceq \gamma(\alpha(g))$

$H2.4$

$\gamma(\overline{g}) \preceq g$

$H2.6$, $\preceq$ transitive.

Q.E.D.

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

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

$$(\gamma(\underline{g}) \preceq \underline{g}) \text{ and } (\underline{g}, \underline{g}) \preceq \gamma(\overline{g})$$

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 $\preceq$) 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 $f(\cup \{ x \mid x \in X \}) = \cup \{ f(x) \mid x \in X \}$.

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

(H4): Let $\overline{F}$ be a continuous function from the complete semi-lattice $\langle L, \preceq, \preceq, \tau, \iota \rangle$ 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 $\underline{g}(\overline{g})$ which is the limit $\cup F^i(\iota)$ of the Kleene's sequence $\iota \preceq F(\iota) \preceq \ldots \preceq F^n(\iota) \preceq \ldots$

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