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
ABSTRACT INTERPRETATION: A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS
OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

[Signature]
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 $\text{Bool} = \{\text{true}, \text{false}\}^0$ and some other primitive domains.

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

$$\text{Env} = \text{Ident}^0 \to \text{Values}$$

- We assume that the meaning of an expression $\text{expr} \in \text{Expr}$ in the environment $e \in \text{Env}$ is given by $\text{val} \equiv \text{expr} \| (e)$ so that:

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

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

$$C_q \in \text{Contexts} = \mathbb{2}^\text{Env}$$

$$C_q = \{ e | (\exists i \geq 0, i_{i_0} \in I\text{-states} | <q, e> = n\text{-state}^i(i_{i_0}) \}$$

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

$$C_v \in \text{Context-Vectors} = \text{Arcs}^0 \to \text{Contexts}$$

$$C_v = \lambda q . \{ e | (\exists i \geq 0, \exists i_{i_0} \in I\text{-states} | <q, e> = n\text{-state}^i(i_{i_0}) \}$$

According to the semantics of programs, the context $C_v(r)$ associated to arc $r$ is related to the

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:

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

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

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

$$\{ C_v, C_v \} \leq \{ C_v, C_v \} \iff \{ \forall r \in \text{Arcs}, C_v(r) \leq C_v(r) \}$$

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 $\tilde{S}$ to the system of equations $X = F\text{-cont}(X)$, ($C_v \leq \tilde{S}$). 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:

$$D_1 : C_v = \lambda q . \{ e | (\exists i \geq 0, \exists i_{i_0} \in I\text{-states} | <q, e> = n\text{-state}^i(i_{i_0}) \}$$

or

$$D_2 : C_v = \text{Y}_{\text{Context-Vectors}}(F\text{-cont})$$

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 $P$:

$$\{ \exists i \geq 0, \exists i_{i_0} \in I\text{-states} | <q, e> = n\text{-state}^i(i_{i_0}) \} \implies \{ e \in C_v(q) \}$$

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

$$\{ e \in C_v(q) \} \implies \{ \exists i \geq 0, \exists i_{i_0} \in I\text{-states} | <q, e> = n\text{-state}^i(i_{i_0}) \}$$

$C_v$ is merely a static summary of the possible exe-
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 = Arcs^0 \rightarrow A-Cont.

Whatever \((Cv', Cv'') \in A-Cont\) may be, we define:
\[ Cv' \sqsupset Cv'' = \lambda r. Cv'(r) \circ Cv''(r) \]
\[ Cv' \sqsubseteq Cv'' = \{ \forall r \in \text{Arcs}^0, Cv'(r) \leq Cv''(r) \} \]
\[ \sim = \lambda r. \simr \text{ and } \perp = \lambda r. \perpr \]

\(<A-Cont, \sqsupset, \sqsubseteq, \perp, \top>\) can be shown to be a complete lattice. The function:
\[ \text{Int} : \text{Arcs}^0 \times A-Cont \rightarrow A-Cont \]
defines the interpretation of basic instructions. If \(\{C(q)\} q \in \text{a-prod}(n)\) is the set of input contexts of node \(n\), then the output context on exit arc \(r\) of \(n\) (\(r \in \text{a-succ}(n)\)) is equal to Int(r, C).

\(\text{Int}\) is supposed to be order-preserving:
\[ \forall a \in \text{Arcs}, \forall (Cv', Cv'') \in A-Cont, \]
\[ \{Cv' \sqsubseteq Cv''\} \rightarrow (\text{Int}(a, Cv') \leq \text{Int}(a, Cv'')) \]

The local interpretation of elementary program constructs which is defined by \(\text{Int}\) is used to associate a system of equations with the program. We define:
\[ \text{Int} : A-Cont \rightarrow A-Cont | \text{Int}(Cv) = \lambda r. \text{Int}(r, Cv) \]

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

### 5.2 Typology of Abstract Interpretations

The restriction that "A-Cont" must be a complete
\[ (u, \rightarrow, \top) \]
\[ (v, \simr, \top) \]
\[ (u, \simr, \top) \]
\[ (n, \top, \top) \]

Examples:

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

### 5.3 Examples

#### 5.3.1 Static Semantics of Programs

The static semantics of programs we defined in section 4 is an abstract interpretation:
\[ I_{ss} = <\text{Contexts}, u, \leq, \text{Env}, \emptyset, \text{n-context}> \]

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

#### 5.3.2 Data Flow Analysis

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

"Available expressions" give a classical example. An expression is available on arc \(r\), if whenever control reaches \(r\), the value of the expression has
The determination of available expressions, back-dominators, intervals, etc., 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 graph, they are examples of backward systems of equations.

6.3.5 Remarks

Our formal definition of abstract interpretations has the completeness property since the model ensures the existence of a particular solution to the system of equations and therefore defines at least some global property of the program. It must also have the consistency property, that is define only correct properties of programs.

One can distinguish between syntactic and semantic abstract interpretations of a program. Syntactic interpretations are proved to be correct by reference to the program syntax (e.g., the algorithm for finding available expressions is justified by reasoning on paths of the program graph). By contrast semantic abstract interpretations must be proved to be consistent with the formal semantics of the language (e.g., constant propagation).

6. Consistent Abstract Interpretations

An "abstract" interpretation $I = \langle A\text{-}Cont, \gamma, \tilde{x}, \tilde{T}, \tilde{I}, \tilde{Int} \rangle$ of a program is consistent with a "concrete" interpretation $I' = \langle C\text{-}Cont, \lambda, \tilde{x}, \tilde{T}, I, \tilde{Int} \rangle$ instead of the global hypothesis 6.0 we will use the following local hypothesis on the concrete and abstract interpretations of primitive language constructs:

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

6.5 and

\[
\begin{align*}
\{(\Psi(a, x) \in C\text{-}Cont, \\
\tilde{\text{Int}}(a, x) \geq \alpha(\text{Int}(a, x))
\end{align*}
\]

These two hypotheses are in fact equivalent (lemma 2.2 in appendix 12). The following schema illustrates 6.5, i.e. the idea of abstract simulation of concrete computations:

Suppose we want to compute the concrete output context $C_0$ (associated with arc a) resulting from concrete input contexts $\tilde{C}_I$, then $C_0 = \gamma(\text{Int}(a, \tilde{C}_I))$. 6.5 requires $\tilde{C}_I$ to contain at least $\tilde{C}_0$, that is $\tilde{C}_0 \subseteq \tilde{C}_I$. On the
where n-pred defines Floyd[67]'s strongest post condition:

\[ n\text{-}\text{pred}(r, Pv) = \]
\[ \text{let}(s \text{ be origin } r, \langle p \text{ be } a\text{-}\text{pred(origin } r)\rangle) \text{ within} \]
\[ \text{case } n \text{ in} \]
\[ \text{Entries} \Rightarrow \langle \forall x \in \text{Ident}, x = ^{i}\text{Values} \rangle \]
\[ \text{Junctions} \Rightarrow \langle Pv(q) \rangle \]
\[ \text{Tests} \Rightarrow \langle \text{case } r \text{ in} \]
\[ \langle a\text{-}\text{succ } t(n) \rangle \Rightarrow Pv(p) \text{ and} \]
\[ \langle a\text{-}\text{succ } f(n) \rangle \Rightarrow Pv(p) \text{ and} \]
\[ \text{not test}(n) \]
\[ \text{esac} \]
\[ \text{Assignments} \Rightarrow \]
\[ \text{let}(P \text{ be } Pv(p), \langle x \text{ be id}(n) \rangle, \]
\[ \langle e \text{ be expr}(n) \rangle) \text{ within} \]
\[ (\forall v \in \text{Values} | P[v/x] \text{ and } x = e[v/x]) \]
\[ \text{esac} \]

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

The deductive semantics is easily validated by proving that \( L \subseteq (\alpha, \gamma) L \)gs, where:

\[ \alpha : \text{Contexts} \rightarrow \text{Pred} \]
\[ = \lambda C. (\text{or } t \text{ and } (x = e(x))) \]
\[ \in C \text{ and } x \in \text{Ident} \]

\[ \gamma : \text{Pred} \rightarrow \text{Contexts} \]
\[ = \lambda P. \langle e | P[e(x)/x], x \in \text{Ident} \rangle \]

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

\[ [\alpha \vdash P, \gamma] \text{ and } [\alpha \vdash P, \gamma] \text{ is closed.} \]

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:

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

2 - reciprocally,

If \( I \equiv (\alpha_1, \gamma_1 I') \), let \( \equiv (\alpha_1) \) 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 \( \gamma_1 | \gamma_1(I') \) of \( \gamma_1 \)

on \( \gamma_1(I') \) is a bijection from the set \( \gamma_1(I') \) of representatives of the equivalence classes on \( I \).

Let us show now that under the hypothesis

\( I \leq (\alpha_1, \gamma_1 I') \) and \( I' \leq (\alpha_2, \gamma_2 I') \), \( \gamma_1 \) is bijective.

\( \alpha_1 | \gamma_1(I') \) and \( \alpha_2 | \gamma_2(I) \) are bijections, hence

\( \forall x' \in I' \), \exists x (unique) \in \gamma_1(I') \) such that

\( x' = (\alpha_1 | \gamma_1(1'))(x). \) Likewise, \( x \in \gamma_1(1') \)

\( \iff x \in I \iff \exists x'' \in \gamma_2(I) \mid x = (\alpha_2 | \gamma_2(1'))(x''). \)

Therefore, \( x' \in I' \iff \exists x'' \in \gamma_2(I) | \]
\[ x'' = (\alpha_1 | \gamma_1(1')) = (\alpha_2 | \gamma_2(1'))(x''). \] Thus
\( \alpha_1 \gamma_1(I') = (\alpha_2 | \gamma_2(1')) \) is a bijection between \( \gamma_2(I) \) and \( I' \).

Since \( (\alpha_2, \gamma_2(1'))^{-1} \) is a bijection
between \( I \) and \( \gamma_2(I) \), the composition

\( (\alpha_1 | \gamma_1(I')) \circ (\alpha_2 | \gamma_2(1)) \circ (\alpha_2 | \gamma_2(1'))^{-1} \)
A further abstraction may be:
\( \alpha([a, b]) = \text{if } a \leq b \text{ then } \text{elsiif } a > 0 \text{ then } + \text{ elsif } b \leq 0 \text{ then } - \text{ else } f \} \gamma(n) = [n, \infty], \gamma(+) = [0, +\infty], \gamma(-) = [-\infty, 0], \gamma(\#) = [-\infty, +\infty]. \)

The abstract contexts are then:

8. Abstract Evaluation of Programs

The system of equations:

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

resulting from an interpretation \( I = \langle A, \text{Cont}, \circ, \preceq, \tau, i, \text{Int} \rangle \) 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):

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

8.1 Correctness

If \( \text{Int} \) is supposed to be a complete morphism (i.e. infinitely distributive over \( \circ \)) then \( \text{Cv} \) 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 \( \text{Cv} \) 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, \( \text{Cv} \) is an approximation (7) of the least fixpoint (e.g. Knu and Ullman[75]).
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:

\[
I_{\text{p}} = \langle \mathbb{R}^+, \max, \leq, 0, \infty, \text{Kir} \rangle
\]

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

\[
\text{Kir}(r, C_{\text{v}}) = \begin{cases} \sum_{n \in \text{Entries}} \{\text{unique entry node}\} & \text{if } \forall v \in \text{ Juncs } \cup \text{ Assigns } \Rightarrow \text{ Kir}(p) \\ \text{Kir}(r, C_{\text{v}}) = \max \{a \in \text{pred}(n)\} & \text{if } \forall v \in \text{ Juncs } \cup \text{ Assigns } \Rightarrow \text{ Kir}(p) \\ \text{Kir}(r, C_{\text{v}}) = \max \{a \in \text{pred}(n)\} \times \text{Prob(test}(n) = \text{true}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Prob(test}(n) = \text{true}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Prob(test}(n) = \text{true}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Prob(test}(n) = \text{true}) \end{cases}
\]

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

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

\[
\text{Kir}(r, C_{\text{v}}) = \begin{cases} \sum_{n \in \text{Entries}} \{\text{unique entry node}\} & \text{if } \forall v \in \text{ Juncs } \cup \text{ Assigns } \Rightarrow \text{ Kir}(p) \\ \text{Kir}(r, C_{\text{v}}) = \max \{a \in \text{pred}(n)\} & \text{if } \forall v \in \text{ Juncs } \cup \text{ Assigns } \Rightarrow \text{ Kir}(p) \\ \text{Kir}(r, C_{\text{v}}) = \max \{a \in \text{pred}(n)\} \times \text{Prob(test}(n) = \text{true}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Prob(test}(n) = \text{true}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Prob(test}(n) = \text{true}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Kir}(r, C_{\text{v}}) \times \text{Prob(test}(n) = \text{true}) \end{cases}
\]

and

\[
\max_{n \in \text{Entries}} \{\text{unique entry node}\} = \sum_{n \in \text{Entries}} \{\text{unique entry node}\}
\]

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 given by the minimal solution to the equation \( n = n + 1 \) which is limit of \( 0 + 1 + 1 + 1 + \ldots \)

- Let \( P \) be the program "begin while T do L end". The number \( n \) of times the expression \( T \) is tested is given by the minimal solution to the equation \( n = 1 + q + q + q + \ldots + q \infty + \ldots \) which is an infinite series. Its sum is \( \frac{1}{1-q} \).

This abstract interpretation leads to a system of linear equations. Kleene’s sequence corresponds to the Jacobi’s iterative method (for numerical coefficients).

9. Fixpoints Approximation Methods

When the extreme fixpoints of the system of equations established for an abstract interpretation \( I \) of a program \( P \) cannot be computed in finitely many steps, they can be approximated. A more abstract interpretation \( \bar{I} (1 \leq \bar{I}) \) may be used for that purpose (e.g. Temenahaufna[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 \( I = \langle A-\text{Cont}, \leq, \leq, 1, \text{Int} \rangle \) be an interpretation of \( P \). When the least fixpoint \( C_{\text{v}} \) of \( \text{Int} \) is unreachable, we look for an upper bound \( U_{\text{v}} \) of \( C_{\text{v}} \), since according to the correctness requirement 6.0, \( C_{\text{v}} \leq Y(U_{\text{v}}) \) and \( Y(U_{\text{v}}) \leq U_{\text{v}} \implies C_{\text{v}} \leq Y(U_{\text{v}}) \).

9.1.1 Increasing Approximation Sequence

Let \( A-\text{Int} : A-\text{Cont} \to A-\text{Cont} \) be such that:

\[
A-\text{Int}(C) = \langle C \cup \text{Int}(C) \rangle \cup \text{Int}(C)
\]

9.1.2 Every infinite sequence \( C_{\text{v}}, A-\text{Int}(C_{\text{v}}), \ldots \), \( A-\text{Int}(\text{Int}(C_{\text{v}})), \ldots \) is not strictly increasing. The approximation sequence \( S_0, S_1, \ldots, S_n, \ldots \) is recursively defined by:

\[
S_0 = \lambda \quad S_{n+1} = \begin{cases} \text{not} \langle \text{Int}(S_n) \rangle & \iff S_n \leq S_n \quad \text{then} \quad A-\text{Int}(S_n) \\ \text{else} \quad S_n \quad \text{end case} \end{cases}
\]

We now prove that \( m \) finite such that:

\[
S_0 \sim S_1 \sim \ldots \sim S_m = S_{m+1} = \ldots
\]

Let \( m \) be the least natural number (eventually infinite) such that \( S_m = S_{m+1} \). \( \forall k \in [0, m) \), we know from 9.1.1.3 that \( \text{not} \langle \text{Int}(S_k) \rangle \leq S_k \). Whence by definition of the ordering \( \leq \), \( S_k \neq \text{Int}(S_k) \forall k \neq S_k \).

Since \( S_k \leq \text{Int}(S_k) \leq S_k \) is always true, we can state that \( S_k \leq \text{Int}(S_k) \leq S_k \). Besides \( \text{not} \langle \text{Int}(S_k) \rangle \leq S_k \) and 9.1.1.1 imply:

\[
S_{k+1} = A-\text{Int}(S_k) \leq \text{Int}(S_k) \leq S_k
\]

and therefore we conclude \( S_k \leq S_k \leq S_k \). \( \forall k \in [1, m] \). Moreover 9.1.1.2 implies that \( m \) is finite. Q.E.D.

Let \( C_{\text{v}} \) be the least fixpoint of \( \text{Int} \), it is the greatest lower bound of the set of \( X \in A-\text{Cont} \) such that \( \text{Int}(X) \leq X \) (Tarski[55]) hence:

\[
\forall X \in A-\text{Cont}, \langle \text{Int}(X) \leq X \rangle \iff (C_{\text{v}} \leq X)
\]

Since \( S_m = S_m \) we have \( \text{Int}(S_m) \leq S_m \) and therefore \( C_{\text{v}} \leq S_m \). \( S_m \) is a correct approximation of \( C_{\text{v}} \).
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 \( \langle q, r \rangle \in \text{Arcs}^2 \), we say that the context associated to \( q \) is dependent on the context associated to \( r \), if and only if:

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

(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 \( q_1, \ldots, q_n \) of arcs, such that \( \{q_i \mid 1 \leq i \leq n \} \) depends on \( Cv(q_1) \) and \( Cv(q_n) \) depends on \( Cw(q_1) \). (e.g. in a forward interpretation a cycle corresponds to a loop in the program).

In any infinite strictly increasing Kleene sequence \( C_{q_1}, \ldots, C_{q_n} \) (\( \text{Arcs} \) is finite there is some arc \( q \) for which the sequence \( Cw(q), Cw(q), \ldots, Cw(q) \) 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 \( \triangledown \) called widening defined by:

9.1.3.1 \( \triangledown : A-Cont \times A-Cont \rightarrow A-Cont \)

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

9.1.3.3 Every infinite sequence \( s_0, \ldots, s_n, \ldots \) of the form \( s_0 = C_0, \ldots, s_n = C_n \triangledown C_{n+1}, \ldots \) (where \( C_0, \ldots, C_n \) are arbitrary abstract contexts) is not strictly increasing.

- The set \( \text{W-arcs} \) 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 \( Cv = \text{Int}(Cv) \) contains at least a widening arc : \( \{q_i \mid 1 \leq i \leq n \} \in \text{W-arcs} \). (e.g. in a forward interpretation on a reducible program graph, \( \text{W-arcs} \) 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:

9.1.3.4 \( A-\text{Int} = \lambda (q, Cv). \text{Int}(q, Cv) \triangledown \text{Int}(q, Cv) \) if \( q \in \text{W-arcs} \),

else \( \text{Int}(q, Cv) \) if \( \not{\triangledown} \). \)

As before, we define:

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

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 \( S_0 = \ldots, S_n, S_{n+1}, \ldots \). We show that this sequence is increasing that is to say:

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

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

\[ S_{m+1} = A-\text{Int}(S_{m+1}) \]

\[ S_{n+1}(q) \triangledown A-\text{Int}(q, S_{m+1}), \forall q \in \text{Arcs}. \]

If \( q \in \text{W-arcs}, \) then \( A-\text{Int}(q, S_{m+1}) \)

\[ A-\text{Int}(q, S_{m+1}) \triangledown A-\text{Int}(q, S_{m+1}) \]

\[ S_{m+1}(q) \triangledown A-\text{Int}(q, S_{m+1}) \]

If \( q \notin \text{W-arcs}, \) then \( A-\text{Int}(q, S_{m+1}) \)

\[ A-\text{Int}(q, S_{m+1}) \]

\[ S_{m+1}(q) \triangledown A-\text{Int}(q, S_{m+1}) \]

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

An infinite sequence \( S_0 = \ldots, S_n, \ldots \) cannot be strictly increasing since otherwise there would exist some widening arc \( q \) for which the sequence \( S_q, S_q, \ldots, S_q \) would never stabilize thus contradicting 9.1.3.3.

We now prove 9.1.1.1 that is to say that:

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

implies

\[ S_n \triangledown A-\text{Int}(S_n) \]

\[ \Rightarrow (S_{n+1}(S_{n+1}) \triangledown A-\text{Int}(S_{n+1})(q), \forall q \in \text{Arcs} \]

\[ \Rightarrow S_{n+1}(q) \triangledown A-\text{Int}(q, S_{n+1}), \text{(see 9.1.3.5)} \]

If \( q \notin \text{W-arcs}, \) we have \( A-\text{Int}(q, S_{n+1}) \)

\[ S_{n+1}(q) \triangledown A-\text{Int}(q, S_{n+1}) \triangledown A-\text{Int}(q, S_{n+1}) \]

\[ S_{n+1}(q) \triangledown A-\text{Int}(q, S_{n+1}) \]

\[ S_{n+1}(q) \triangledown A-\text{Int}(q, S_{n+1}) \]

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 subscripted only by indices within bounds. For that purpose one can use the lattice \( L \) of section 7. Let us take an obvious example:

246
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:

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

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. \([[0, 0] < [0, 1] < \ldots < [0, +\infty]\) for the program \(x := 0\); while \(x := x+1\)), we must use an approximation sequence. Hence the results will be somewhat inaccurate but runtime subscript tests may be inserted in the absence of certainty.

Let us define the widening \(\triangledown\) of intervals by:

\[\triangledown [i, j] = \triangledown [k, l] = \begin{cases} \triangledown [i, j] & \text{if } k < i \text{ then } -\infty \text{ else } i, \\ \triangledown [k, l] & \text{if } k > j \text{ then } +\infty \text{ else } l \end{cases} \]

\(\triangledown\) satisfies the requirements of 9.1.3. According to 9.1.3.4 the system of equations is modified by:

7. \(C2 = C2 \triangledown (C1 \cup C4)\)

The corresponding approximation sequence is:

\[\begin{align*}
C3 & = [1, 100] \\
C4 & = C3 \triangledown [1, 1] \\
C5 & = C2 \cap [101, +\infty] \\
\end{align*}\]

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 \(S_m = \text{A-Inf}^m(T)\) of the least fixpoint \(CV\) of \(T\). However, \(\text{A-Inf}^m(S_m) \not< S_m\). Since \(\text{A-Inf}\) is order preserving, this implies that:

\[S_m \not< \text{A-Inf}(s_m) \not< \ldots \not< \text{A-Inf}^m(s_m) \not< \ldots \not< CV.\]

If \(S_m\) is not a fixpoint of \(\text{A-Inf}\), and the ascending sequence is finite (e.g., the lattice \(A\)-Cont satisfies the ascending chain condition), its limit is a better approximation of \(CV\) than \(S_m\).

When the sequence is infinite or slowly converging, one can among other solutions approximate its limit.

8.3.1 Decreasing Approximation Sequence

At step \(n\) in the descending sequence, we have:

\[\text{A-Inf}^n(S_m) \not> \text{A-Inf}^m(S_m) \not> CV.\]

In order to accelerate the convergence, we should find an approximation \(D\) such that \(\text{A-Inf}^n(S_m) \not> D \not> CV.\) But not knowing \(CV\), this characteristic function \(D\) cannot be chosen.
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound on $n$).

The limit of the descending sequence $S'_0 = \tilde{r}, \ldots, S'_n = \hat{\delta}^{-1}\text{Int}(\tilde{r}), \ldots$ is an upper bound of the greatest fixpoint of $\text{Int}$.
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) \( C_2 = C_2 \land (C_1 \cup C_4) \)

The descending approximation sequence is:
Any of the AAS, TDS, DAS, TAS methods may yields a fixpoint \( fp \) which is not the fixpoint \( \xi fp \) or


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


\{\forall (x, y) \in L^2, (x \leq y) \Rightarrow (f(x) \leq f(y))\}

\Rightarrow \{\forall (x, y) \in L^2, (f(x \lor y) \geq f(x) \lor f(y))\}

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

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

(L1): The fixpoints of \( F \) form a non-empty complete lattice with supremum \( g \), infimum \( \lambda \) such that:

\[ g = \bigvee \{x | (x \in L) \land (x \leq F(x))\} \]

\[ \lambda = \bigwedge \{x | (x \in L) \land (F(x) \leq x)\} \]

(This result is proved in Tarski[55], pp.286-287.)
(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)) & \text{since } \overline{g} = F(g) \\
\overline{g} &\preceq \alpha(F(\alpha(g))) & \text{H3.1, \(\cup\) isotone, \(\preceq\) transitive} \\
\overline{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 \(g, \overline{g}, \preceq, \preceq, \dots, F, \overline{F}, \alpha, \gamma, H3.1, H2.4, H2.6\) respectively by \(\overline{g}, \overline{\ell}, \preceq, \preceq, F, \overline{F}, \alpha, \gamma, H3.2, H2.3, H2.5\) 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 \(f: D \to D\) is called continuous if whenever \(X \subseteq L\) is directed, then \(f(\{x \mid x \in X\}) = \cup\{f(x) \mid x \in X\}\).

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

(H4): Let \(\overline{F}\) be a continuous function from the complete semi-lattice \(\langle L, \cup, \preceq, \tau, \iota \rangle\) 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 \(\ell(\overline{\ell})\) which is the limit \(\lim_{i=0}^{\infty} F^i(L)\) of the Kleene's sequence \(1 \leq F(1) \leq \ldots \leq F^n(1) \leq \ldots\)

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