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

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
### 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 $\text{Bool} = \{\text{true}, \text{false}\}$ 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 the environment $\epsilon \in \text{Env}$ is given by $\text{val} \sqsupset \text{expr} \sqsubseteq \epsilon$ so that:

  $$\text{val} : \text{Expr} \rightarrow [\text{Env} \rightarrow \text{Values}]$$

  In particular the projection $\text{val} \mid \text{Bexpr}$ of the function $\text{val}$ in domain Bexpr has the functionality:

  $$\text{val} \mid \text{Bexpr} : \text{Bexpr} \rightarrow [\text{Env} \rightarrow \text{Booll}]$$

  - The state set "States" consists of the set of all information configurations that can occur during computations:
    $$\text{States} = \text{Ares}^* \times \text{Env}.$$  
    A state $s \in \text{States}$ consists in a control state $\text{cs}(s)$ and an environment $\text{env}(s)$, such that:
    $$\forall s \in \text{States}, s = <\text{cs}(s), \text{env}(s)>.$$  

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

  - If $e \in \text{Env}, v \in \text{Values}, x \in \text{Ident}$ then $
    e_1[v/x] \equiv x, \text{cond}(y = x, v, e(y))$.  

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

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

    1. Let $n$ be $\text{end}(\text{cs}(s))$, $e$ be $\text{env}(s)$ within case $n$

    2. Assignments $\Rightarrow$

    3. Tests $\Rightarrow$

    4. Exits $\Rightarrow$

      Each partial function $f$ on a set $S$ is extended to a continuous total function on the corresponding domain $S^\mathbb{R}$ by $f(\bot) = 1, f(\top) = \top$ and $f(x) = 1$ if the partial function is undefined at $x$.

- Let $\bot_{\text{env}}$ be the bottom function on $\text{Env}$ such that $$(\forall \epsilon \in \text{Ident}^*, \bot_{\text{env}}(\epsilon) = \epsilon[\text{Values}])$$

- Let I-states be the subset of initial states:
  $$\text{I-states} = \{ <\text{a-succ}(n), \text{I-env}> | m \in \text{Entries} \}$$
A "computation sequence" with initial state $i_s \in \text{I-states}$ is the sequence:

$$s_n = n\text{-state}^n(i_s) \text{ for } n = 0, 1, \ldots$$

where $f^n$ is the identity function and $f^{n+1} = f \circ f^n$.

The initial to final state transition function:

$$n\text{-state}^\infty : \text{States} \rightarrow \text{States}$$

is the minimal fixpoint of the functional:

$$\lambda F. (n\text{-state} \circ F)$$

Therefore

Since the equation $Cv(r) = n\text{-context}(r, Cv)$ must be valid for each arc $r$, $Cv$ is a solution to the system of "forward" equations:

$$Cv = F\text{-cont}(Cv)$$

where

$$F\text{-cont} : \text{Context-Vectors} \rightarrow \text{Context-Vectors}$$

is defined by:

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

Context-Vectors is a complete lattice with union $\vee$ such that $Cv_1 \vee Cv_2 = \lambda r. (Cv_1(r) \vee Cv_2(r))$.

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

$$(Cv, \preceq Cv_2) \quad \rightarrow \quad \{ \forall r \in \text{Arca}, Cv_1(r) \preceq Cv_2(r) \}$$
This implies that A-Cont is in fact a complete lattice, but we need only one of the two join and meet operations. The set of context vectors is defined by $A\text{-}Cont = \text{Arcs}^0 \rightarrow A\text{-}Cont$.

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

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

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

$\sim \tau = \lambda r. \tau$ and $\perp = \lambda r. \perp$
The determination of available expressions, back-
dominators, intervals, ... requires a forward sys-
tem of equations. Some global flow problems, nota-
hly the live variables and very busy expressions
require propagating information backward through
the program graph, they are examples of backward
systems of equations.

6.5.5 Remarks

Our formal definition of abstract interpretations

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

\[ \{(\gamma(a, x) \in \text{Arcg} \times \text{A-Contr}, \gamma(\text{Int}(a, x)) \geq \text{Int}(a, \gamma(x))\} \]

6.5 and \( \{(\gamma(a, x) \in \text{Arcg} \times \text{C-Contr}, \text{Int}(a, \gamma(x)) \geq \alpha(\text{Int}(a, x))\} \)

These two hypothesis are in fact equivalent (lemma
1.7 in appendix 19). The following schema illus-

where \( n\text{-pred} \) defines Floyd[67]'s strongest post condition:

\[
\text{n-pred}(r, PV) = \text{let}(a \text{ be origin}(r), (p \text{ be n-pred(origin}(r))) \text{ within case } n \text{ in }
\]

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:

\[
\{I \equiv (\beta I')\} \iff \{\beta \text{ is an isomorphism between the algebras } I \text{ and } I'\}
\]
A further abstraction may be:
\[ a((a, b)) = \text{if } a \neq b \text{ then } a \text{ else if } a \geq 0 \text{ then } + \text{ else } - \]
\[ t(r, y) = (n, m), \quad z(+) = [0, +\infty), \quad z(-) = [-\infty, 0), \quad z(\pm) = [-\infty, +\infty]. \]
The abstract contexts are then:

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

The abstract evaluation terminates iff Kleene's sequence is finite. This may be the case because A-Cont is finite (e.g. type checking in ALGOL 60, Naur[65]), or a finite subset only is to be considered for any particular program (e.g. type checking in ALGOL 68), or A-Cont may be of finite length \( m \) (the length of any strictly increasing chain is bounded by \( m \), Kildall[73], Wegbreit[75]) or A-Cont may satisfy the ascending chain condition (every strictly increasing chain is finite, although not bounded). A lattice may have infinite chains, although Int is chosen so that Kleene's sequences are finite. Finally an infinite Kleene's sequence may be directly truncated (to get a lower bound).
8.4 Example: Performance Analysis of Programs

The performance of programs may be analyzed by deriving for each program point the final value of an imaginary counter which is incremented each time control goes through that point.

Let \( A-\text{Cont} \) be the lattice \( \mathbb{R}^+ \) of positive real numbers augmented by the upper bound \( \infty \), with natural ordering \( \leq \). The abstract interpretation:

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

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

\[
\text{Kir}(r, Cv) = \\
\text{let } n \text{ be origin}(r) \text{ within} \\
\text{case } n \text{ in} \\
\text{Entries} \Rightarrow 1 \{ \text{unique entry node} \} \\
\text{Junctions} \cup \text{Assignments} \Rightarrow \{ \text{pred}(n) \} \\
\text{Tests} \Rightarrow \\
\text{case } r \text{ in} \\
\{ a \text{-succ}(r) \} \Rightarrow \text{Cv}(\text{pred}(n)) \ast \text{Prob}(\text{test}(n) = \text{true}) \\
\{ a \text{-succ}(r) \} \Rightarrow \text{Cv}(\text{pred}(n)) \ast \text{Prob}(\text{test}(n) = \text{true})
\]

9. Fixpoints Approximation Methods

When the extreme fixpoints of the system of equations established for an abstract interpretation \( \Gamma \) of a program \( P \) cannot be computed in finitely many steps, they can be approximated. A more abstract interpretation \( \tilde{\Gamma} (1 \leq \tilde{T}) \) may be used for that purpose (e.g. Tennenbaum [74]). It is often better to make approximations in \( \Gamma \), 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 \( \Gamma = \langle A-\text{Cont}, \circ, \leq, \tau, \text{Int} \rangle \) 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 \( \text{Cv} \leq \text{UB} \) implies \( \text{Cv} \leq \gamma(\text{UB}) \).

9.1.1 Increasing Approximation Sequence

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

\[ \forall n \geq 0, \tilde{C} = \tilde{\text{Int}}^n(\tilde{A}) \text{ and not}(\text{Int}(C) \leq C) \]
9.1.2 Generalization of Kleene's Ascending Sequence

When $A$-Cont satisfies the ascending chain condition one can choose $A\dash{\text{int}}$ to be $\text{Int}$ 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 $A\dash{\text{int}}$ in 9.1.1 is global. We now indicate a way to construct $A\dash{\text{int}}$ by local modifications to $\text{Int}$.

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

\[ \exists \alpha \in A\dash{\text{Cont}}, \beta \in A\dash{\text{Cont}} \mid \text{Int}(q, \alpha \sigma) \neq \text{Int}(q, \beta \sigma[C/r]) \]

(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 $i \in [1, n]$, and $Cv(q_{i+1})$ depends on $Cv(q_i)$, and $Cv(q_i)$ depends on $Cv(q_{i+1})$. (e.g. in a forward interpretation a cycle corresponds to a loop in the program).

In any infinite strictly increasing Kleene's sequence $Cv_1, \ldots, Cv_n, \ldots$ since $\text{Arcs}$ is finite there is some arc $q$ for which the sequence $Cv_1(q), \ldots, Cv_n(q), \ldots$ never stabilizes. Therefore $q$ must belong to a cycle or the context $\alpha$ associated to $q$.

As before, we define:

\[ A\dash{\text{int}} = \lambda q. A\dash{\text{int}}(q, Cv) \]

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

Let us consider a sequence $S_0 = \ldots, S_n, \ldots$ where $\forall n \geq 0$.

We show that this sequence is increasing that is to say:

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

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

\[ S_{m+1} \geq A\dash{\text{int}}(S_{m+1}) \]

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

If $q \notin W\text{-arcs}$, then $A\dash{\text{int}}(q, S_{m+1}) = S_{m+1}(q) \geq \text{Int}(q, S_{m+1}) \geq S_{m+1}(q) \geq S_{m+1}(q)$.

If $q \notin W\text{-arcs}$, then

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

\[ \Rightarrow \text{Int}(q, S_{m+1}) \geq A\dash{\text{int}}(q, S_{m+1}) \]

Since $S_{m+1}(q) \geq S_{m+1}(q)$ and $\text{Int}$ is order preserving. Moreover from $q \notin W\text{-arcs}$ and 9.1.3.4 we get

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

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

An infinite sequence $S_0 = \ldots, S_n, S_{n+1}, \ldots$ cannot be strictly increasing since otherwise there
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. \(C^0 = \mathbf{[\cdot, \cdot]}\)
2. \(C^1 = \mathbf{[1, 1]}\)
3. \(C^2 = C^1 \cup C^4\)
4. \(C^3 = C^2 \cap \mathbf{[-\infty, 100]}\)
5. \(C^4 = C^3 + \mathbf{[1, 1]}\)
6. \(C^5 = C^4 \cap \mathbf{[101, +\infty]}\)

Assignment statements are treated using 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. \([0, 0] < [0, 1] < \ldots \leq \mathbf{[0, +\infty]}\) for the program \(x := 0\); while true do \(x := x+1\)), 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 \(\mathcal{V}\) of intervals by:

\(-\mathbf{[\cdot, \cdot]} = \text{null element of } \mathcal{V}
\)

\([-i, j] \mathcal{V} [k, l] = \begin{cases} \mathbf{[i, l]}, & \text{if } k < i \text{ then } -\infty \text{ else } i; \\ \mathbf{[j, +\infty]}, & \text{if } k > j \text{ then } +\infty \text{ else } j \end{cases}
\)

\(\mathcal{V}\) satisfies the requirements of 9.1.3. According to 9.1.3.4, the system of equations is modified by:

2. \(C^2 = C^2 \cup (C^1 \cup C^4)\)

The corresponding approximation sequence is:

* \(C^1 = \mathbf{[1, 1]}\) for \(i \in \{0, 3\}\)
* \(C^2 = C^2 \cup (C^1 \cup C^4)\)
  - \(= \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \cup \mathbf{[1, 1]} \) for \(i \in \{0, 3\}\)
* \(C^3 = C^2 \cap \mathbf{[-\infty, 100]}\)
  - \(\mathbf{[1, 1]} \cap \mathbf{[-\infty, 100]} \)
* \(C^4 = C^3 + \mathbf{[1, 1]}\)
  - \(\mathbf{[1, 1]} + \mathbf{[1, 1]} \)
* \(C^5 = C^4 \cap \mathbf{[101, +\infty]}\)
  - \(\mathbf{[2, 2]} \cap \mathbf{[101, +\infty]} \)
* \(C^6 = C^5 \cap \mathbf{[-\infty, 100]}\)
  - \(\mathbf{[1, 1]} \cap \mathbf{[-\infty, 100]} \)

The final context on each arc is marked by a star \(*\). Note that the results are approximate ones, (e.g. \(C^5\)).

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.

### 9.3 Finite Iterative and Decreasing Approximation of the Least Fixpoint Starting from an Upper Bound

The ascending approximation sequence leads to an upper bound \(S^m = \mathcal{Int}^m(\mathcal{V})\) of the least fixpoint. Moreover \(\mathcal{Int}(S^m) \preceq S^m\). Since \(\mathcal{Int}\) is order preserving, this implies that:

\(S^m \succeq \mathcal{Int}(S^m) \succeq \ldots \succeq \mathcal{Int}^m(S^m) \succeq \ldots \succeq \mathcal{CV}\).

If \(S^m\) is not a fixpoint of \(\mathcal{Int}\) and the above descending sequence is finite (e.g. the lattice A-Cont satisfies the descending chain condition), its limit is a better approximation of \(\mathcal{CV}\) than \(S^m\). When the sequence is infinite or slowly converging, one can among other solutions approximate its limit.

#### 9.3.1 Decreasing Approximation Sequence

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

\(\mathcal{Int}^{n+1}(S^m) \succeq \mathcal{Int}^n(S^m) \succeq \mathcal{CV}\)

In order to accelerate the convergence, we should find a descending sequence \(S^m, S^{m-1}, \ldots, S^0\) such that \(\mathcal{Int}^n(S^m) \succeq S^n \succeq \mathcal{CV}\). But not knowing \(\mathcal{CV}\), this characterization is very weak since \(S^n\) could be chosen incorrectly that is to say less than \(\mathcal{CV}\) or non comparable with \(\mathcal{CV}\). The fact that \(\mathcal{CV}\) is the greatest lower bound of the set of \(X \in A-Cont\) such that \(\mathcal{Int}(X) \preceq X\) gives a correctness criterion for the choice of \(D\) when \(\mathcal{CV}\) is unknown, we must have:

\(\mathcal{Int}^n(S^m) \succeq D \succeq \mathcal{Int}(D)\)

On the contrary to 9.1.1, this characterization does not provide an efficient construction of \(D\).

#### 9.3.2 Terminated Decreasing Sequence

In front of these difficulties we will enforce convergence by choosing \(D\) such that:

\(\exists n > 0 \quad \mathcal{Int}(S^m) \succeq D \succeq \mathcal{Int}^n(S^m)\)
(However, we will not artificially truncate the decreasing sequence by imposing an arbitrary upper bound on $n$).

Let $D$ be such that:

9.3.2.1 $\forall C \in A$, $C \succeq \overline{\text{Int}}(C) \implies \{ C \succeq D \implies \overline{\text{Int}}(C) \}$

9.3.2.2 $\forall C \in A$, every infinite sequence $C$, $D = \overline{\text{Int}}(C)$, ... is not strictly decreasing.

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

9.3.2.3 $S_0' = S$ 

$S_{n+1}' = \text{if } (S_n' \neq \overline{\text{Int}}(S_n')) \text{ and } (S_n' \neq D \overline{\text{Int}}(S_n'))$

The limit of the descending sequence $S_0' = \overline{\text{Int}}(\overline{\text{Int}}(\overline{\text{Int}}(...)))$ is an upper bound of the greatest fixpoint of $\overline{\text{Int}}$.

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 $D$ by local modifications to $\overline{\text{Int}}$:

9.3.4.1 $\Delta : A \times A \rightarrow A$

9.3.4.2 $\forall C, C' \in A$, $\Delta (C, C') \iff (C \succeq C \land C \succeq C')$

9.3.4.3 Every infinite sequence $s_0, ..., s_n, ...$ of the form $s_0 = C_0$, $s_i = s_{i-1} \Delta C_i$ for $i > 0$. 

\[ s \]
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:
a fixpoint $fp$ which is not the fixpoint $fp$ or gfp of interest. None of these methods can improve $fp$ to reach $fp$ or gfp, therefore a "fixpoint improvement method" is necessary. It is our feeling that such a method could be designed only when considering that $A$-cont possesses a richer structure (i.e. for particular applications).

Furthermore, in the AAS, TDS, DAS, TAS sequences the term of rank $n$ is computed only as a function of the term of rank $n-1$, hence these are "separate steps" methods. One can as well imagine to use "bound steps" methods, where the term of rank $n$ is computed as a function of the terms of rank $n-1, n-2, \ldots, n-k$. In this last case the Kleene's sequences may be used to compute the first $k$ terms. After $k$ steps more informations about the program would be available to heuristically accelerate the convergence so that the definition of $A$-int and $B$-int could be more refined.

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

II. References


Cousot[76']. Static determination of dynamic properties of generalized type unions. Submitted for publication. (Sept.)


Schwartz[75]. Automatic data structure choice in a

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

\leq \Rightarrow \{\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 \(<L, u, \leq, \top, \bot>\)
(T1): \( H_1, H_1, H_2, H_3 \) imply that the greatest fix-
points \( g \) and \( \bar{g} \) of \( F \) and \( \bar{F} \) are related by:
\[
\{ \alpha(g) \leq \bar{g} \} \text{ and } \{ g \leq \gamma(\bar{g}) \}
\]
Proof:
The existence of \( g \) and \( \bar{g} \) is stated by (L1).

<table>
<thead>
<tr>
<th>g \uparrow \alpha(g) \downarrow \alpha(g)</th>
<th>\text{trivially}</th>
</tr>
</thead>
<tbody>
<tr>
<td>g \equiv u(F(g)) \geq u(g)</td>
<td>since ( \bar{g} = F(g) )</td>
</tr>
</tbody>
</table>