A technique for deriving semantic information on computer programs

Published: 01/01/1973

Citation for published version (APA):
A technique for deriving semantic information on computer programs.

1. Let \( \pi \) be a computer program. We assume that it operates deterministically on a set \( \Omega \) of states. Semantically it maps a subset of \( \Omega \) into \( \Omega \); the states not belonging to that subset lead to nonterminating program execution. That is, the semantics are described by a partial mapping \( g \) of \( \Omega \) into \( \Omega \).

In a recent note [2] E.W. Dijkstra recommends to discuss the semantics by means of inverse images. The inverse image of \( g \) is the mapping \( g^* \) of \( \mathcal{P}(\Omega) \) into \( \mathcal{P}(\Omega) \) (\( \mathcal{P}(\Omega) \) is the set of all subsets of \( \Omega \)) described by

\[
g^*(S) = \{ \omega \in \Omega \mid g(\omega) \text{ is defined, and } g(\omega) \in S \}.
\]

Indeed, as we know from various branches of mathematics (e.g. topology, ergodic theory) \( f^* \) has nicer properties than itself. For example we have

\[
g^*(S_1 \cap S_2) = g^*(S_1) \cap g^*(S_2),
\]

but not necessarily \( g(S_1 \cap S_2) = g(S_1) \cap g(S_2) \).

For non-deterministic programs see section 22.

2. Another matter is that Dijkstra discusses predicates on \( \Omega \) rather than subsets of \( \Omega \). Technically this makes no difference, but the predicates are the more natural objects when discussing a program. Thus Dijkstra expresses everything in terms of the predicate transformer \( f^* \) that maps any post-condition onto the weakest pre-condition. That is to say, if \( Q \) is a
predicate on \( R \), then \( f\pi(Q) \) is the predicate described by
\[
(f\pi(Q))(w) = \text{"} g(w) \text{ is defined, and } Q(g(w)) \text{ is true.}" 
\]

Dijkstra discusses (i) empty statement, (ii) concatenation, (iii) binary selection, (iv) assignment, (v) recursion.

In the latter case it is not quite clear whether his simplified point of view will be efficient for deriving all required semantic information, in particular since his result requires the knowledge of \( f\pi(I) \).

3. In practical questions about programs, we usually do not bother about what the weakest pre-condition \( f\pi(Q) \) is exactly. The usual question is, if \( P \) and \( Q \) are given predicates, whether \( P \Rightarrow f\pi(Q) \) holds. That is, we loose something deliberately. And at intermediate stages we have to construct predicates which provide a loss we can afford and which are still sufficiently simple to manage. (This is a well known strategy in mathematics, e.g. when constructing strings of inequalities in analysis). Therefore we may wish to deal with \( P \Rightarrow f\pi(Q) \) as the basic notion.

4. Another matter is that it cannot be hoped that it always suffices to consider, at intermediate stages, a single \( Q \) only. On the other hand, the full knowledge of the mapping \( f\pi \) seems to be too much in many cases. We want to have a tool by which it can be specified what information on the semantics of pieces of program we want to keep track of.

5. We suggest the use of a proposition \( v(\pi,P,Q) \) we shall explain presently.
We consider pairs of predicates \( P, Q \), where \( P \) is a predicate on \( \Omega \) and \( Q \) a predicate on \( \Omega \times \Omega \). Actually the values of \( Q(\omega,\omega') \) at points where \( P(\omega) \) is false, will never play a role. We might as well consider \( Q \)'s that are undefined at points \((\omega,\omega')\) with \( P(\omega) \) false, but this would complicate many statements considerably.

Accordingly, we shall say that the pairs \( P, Q \) and \( P_1, Q_1 \) are equal if \( P = P_1 \) and \( Q(\omega,\omega') = Q_1(\omega,\omega') \) as long as \( P(\omega) \) is true. This definition of equality has to be used in order to maintain (in section 6) that "<" is a partial order relation.

For every program \( \pi \) and for every pair \( P, Q \) we consider the proposition \( v(\pi, P, Q) \). It has the following meaning:

"For every \( \omega \in \Omega \) for which \( P(\omega) \) is true, execution of the program \( \pi \) with initial state \( \omega \) leads to termination at a state \( \omega' \) for which \( Q(\omega,\omega') \) is true."

If this proposition is true, we say that \( (P, Q) \) gives semantic information about the program. We also say that \( (P, Q) \) is a predicate pair for \( \pi \).

6. The class of all pairs \( (P, Q) \) is partially ordered by a relation we write as <. We say that \( (P, Q) < (P', Q') \) if both (i) and (ii) are true:

(i) For all \( \omega \) we have \( P(\omega) \Rightarrow P'(\omega) \)

(ii) For all \( \omega, \omega' \) we have \( (P(\omega) \land Q'(\omega,\omega')) \Rightarrow Q(\omega,\omega') \)

It is easy to show that if \( \pi \) is a program, and \( (P, Q) < (P', Q') \), then \( v(\pi, P', Q') = v(\pi, P, Q) \). The philosophy is that \( v(\pi, P', Q') \) gives the same or more detailed information about at least the same cases as \( v(\pi, P, Q) \) did.

7. For every program \( \pi \) there is a maximal pair \( (P_0, Q_0) \). If \( g \) is the partial mapping produced by \( \pi \) then we take \( P_0(\omega) \) to be the proposition that \( g(\omega) \) is
defined (i.e. that execution of π with the initial state ω leads to
termination, and \( Q_0(\omega, \omega') \) is the proposition \( P_0(\omega) \Rightarrow \omega' = g(\omega) \). We have \( v(\pi, P, Q) \),
and for all pairs \((P, Q)\)

\[ (P, Q) < (P'_0, Q'_0) \iff v(\pi, P, Q). \]

8. The art of proving program correctness can be described as follows.
We want to establish for a given composite program that for a given \((P, Q)\) we
have \( v(\pi, P, Q) \). We do this by establishing suitable predicate pairs
for various parts of \( \pi \). There are several syntactic devices for composing
a program from its parts; for each of these devices we have to say how a
pair \((P, Q)\) for the composite program can be constructed when pairs for
the sub-programs are known. At any stage of the program correctness proof
we may deliberately lose information by replacing a pair by a simpler one
that is lower in the sense of the partial ordering.

9. Quite often it will be possible to select pairs \((P, Q)\) where \( Q(\omega, \omega') \)
depends on \( \omega' \) only. In such cases \( v(\pi, P, Q) \) is equivalent to the proposition
\( P \Rightarrow f_\pi(Q) \) discussed in section 3.

10. Without going into axiomatization, we display a few properties of the
function \( v \). In order to avoid further repetition, we list our notations here:

- \( \Omega \) is the set of states; \( \omega_1, \ldots \) denote elements of \( \Omega \); we use letters \( P, Q \)
for predicate pairs; we use "\(<\)" for the partial ordering (section 6);
- \( \pi_1, \ldots \) denote programs, \( T \) is the predicate that is true for all values of
its variables, and \( F \) is the one that is false for all values of its variables.
We use $\neg$ for negation, $\Rightarrow$ for implication, $\wedge$ for "and", $\vee$ for "or", $\equiv$ for logical equivalence. These symbols are used for propositions as well as for predicates.

11. Properties that do not depend on the structure of the program but only on the fact that the program provides a partial mapping.

(i) $v(\pi, P, F) = (P = F)$.
(ii) $v(\pi, F, Q)$ is true.
(iii) $v(\pi, P, Q \wedge Q_2) = v(\pi, P, Q) \wedge v(\pi, P, Q_2)$.
(iv) $v(\pi, P_1, P_2, Q) = (v(\pi, P_1, Q) \wedge v(\pi, P_2, Q_2))$.
(v) (stated already in section 6). If $(P, Q) < (P', Q')$ then $v(\pi, P', Q') \Rightarrow v(\pi, P, Q)$.

12. The empty statement. Let $\pi_0$ be the empty program. (Its partial mapping is the identity on $\Omega$). Then

$$v(\pi_0, P, Q) = \forall_{\omega} Q(\omega, \omega).$$

The maximal pair is the pair $P_0, Q_0$ with $P_0 = T$, $Q_0(\omega, \omega') = (\omega=\omega')$ for all $\omega, \omega'$.

13. Concatenation. Let $\pi$ denote the program $\pi_1; \pi_2$. Assume that $v(\pi, P_1, Q_1)$ and $v(\pi_2, P_2, Q_2)$ are true. Let $P_3, Q_3$ be a third predicate pair. Assume

(i) $P_3 \Rightarrow P_1$
(ii) For all $\omega, \omega'$ we have

$$\left( P_3(\omega) \wedge Q_1(\omega, \omega') \right) \Rightarrow P_2(\omega').$$
(iii) For all \( \omega, \omega'' \) we have

\[
(P_3(\omega) \land \forall \omega', (Q_1(\omega, \omega') \Rightarrow Q_2(\omega', \omega''))) \Rightarrow Q_3(\omega, \omega'').
\]

Under these assumptions (i), (ii), (iii) we have \( v(\pi, P_3, Q_3) \).

The maximal pair \((P_3, Q_3)\) that satisfies these conditions is given by

\[
P_3(\omega) = P_1(\omega) \land \forall \omega', (Q_1(\omega, \omega') \Rightarrow P_2(\omega')) \land Q_3(\omega, \omega'') = \forall \omega', (Q_1(\omega, \omega') \Rightarrow Q_2(\omega', \omega'')).
\]

14. Binary selection. Let \( B \) be a predicate, let \( \pi_1, \pi_2 \) be programs, and let \( \pi \) be the program "if \( B \) then \( \pi_1 \) else \( \pi_2 \).

Assume that \( v(\pi, P_1, Q_1) \) and \( v(\pi_2, P_2, Q_2) \) are true. Moreover assume that

\[
P_3 \Rightarrow (P_1 \land B) \lor (P_2 \Rightarrow B)
\]

and that for all \( \omega, \omega' \)

\[
(P_3(\omega) \land ((B(\omega) \land Q_1(\omega, \omega')) \lor (\neg B(\omega) \land Q_2(\omega, \omega'))) \Rightarrow Q_3(\omega, \omega'))
\]

Then we have \( v(\pi, P_3, Q_3) \).

The maximal pair \( P_3, Q_3 \) that satisfies these relations is

\[
P_3 = (P_1 \land B) \lor (P_2 \land \neg B), \quad Q_3(\omega, \omega') = (B(\omega) \land Q_1(\omega, \omega')) \lor (\neg B(\omega) \land Q_2(\omega, \omega')).
\]

15. Assignment. An assignment \( \pi \) is a program whose partial mapping \( g \) is defined for all \( \omega \) that satisfy a predicate \( D \) (usually \( D = T \)); this \( g \) is explicitly given in the program as \( \omega := g(\omega) \).

Assuming that for all \( \omega \)

\[
P(\omega) \Rightarrow (D(\omega) \land Q(\omega, g(\omega)))
\]

we have \( v(\pi, P, Q) \).

The maximal pair is the one given by \( P = D \), \( Q(\omega, \omega') = (\omega' = g(\omega)) \).

16. Declaration. We consider two different sets of states. The one is \( \Omega \), the other is \( \Omega_1 = \Omega \times \Lambda \) (cartesian product), where \( \Lambda \) is some set.
In order to have something that can be used as a type in ordinary Algol, we take for \( \mathbb{A} \) the set of all integers. For \( \Omega \) we have a predicate pair \( P, Q \), and for \( \Omega_1 \) we have a predicate pair \( P_1, Q_1 \). Elements of \( \Omega_1 \) will be denoted as \( \omega, \lambda^\prime \) (where \( \omega \in \Omega, \lambda \in \mathbb{A} \)).

Let \( \pi \) be a program operating on \( \Omega_1 \), and let \( \pi \) be the program 

\[
\text{begin } \text{integer } \lambda; \pi \text{ end. Assume that } v(\pi_1, P_1, Q_1) \text{ is true, and that }
\]

\[
(i) \quad \forall \omega \in \Omega \ (P(\omega) \Rightarrow \forall \lambda \in \mathbb{A} \ P_1(\omega, \lambda^\prime)),
\]

\[
(ii) \quad \forall \omega \in \Omega \forall \omega \in \Omega \forall \lambda \in \mathbb{A} \forall \lambda^\prime \in \mathbb{A} \ ((P(\omega) \land Q_1(\omega, \lambda^\prime)) \Rightarrow Q(\omega, \lambda^\prime)).
\]

Then we have \( v(\pi, P, Q) \).

The maximal pair \( P, Q \) that satisfies these relations is given by

\[
P(\omega) = \forall \lambda \in \mathbb{A} \ P_1(\omega, \lambda^\prime),
\]

\[
Q(\omega, \lambda^\prime) = \exists \lambda^\prime \exists \lambda \forall \lambda \in \mathbb{A} \ Q_1(\omega, \lambda^\prime).
\]

17. Recursion. Just like Dijkstra [2] did, we restrict ourselves to a recursive procedure without local variables and without parameters. The body can be described as follows. Take a finite number of syntactic variables: \( \xi_1, \ldots, \xi_m \). The expression \( \tilde{I}(\xi_1, \ldots, \xi_m) \) stands for something that turns into a program if programs are substituted for \( \xi_1, \ldots, \xi_m \).

We now want to define the recursive procedure \( \Pi \) by describing its body as \( \tilde{I}(\Pi, \ldots, \Pi) \).

**Example.** Let \( \pi_1 \) and \( \pi_2 \) be programs, and let \( \tilde{I}(\xi_1, \xi_2) \) be

\[
\text{if } B \text{ then begin } \xi_1; \xi_2 \pi_1 \text{ end else } \pi_2.
\]

This leads to the procedure

\[
\text{procedure } \Pi; \text{ if } B \text{ then begin } \Pi; \Pi; \pi_1 \text{ end else } \pi_2;
\]

We return to the general case where \( \Pi \) is defined as \( \tilde{I}(\Pi, \ldots, \Pi) \). Let \( (P, Q), (P_0, Q_0), (P_1, Q_1), \ldots \) be predicate pairs. We assume that \( P_0 = F \), and that
for every integer $k > 0$ the following is true: For all programs $\pi_1, \ldots, \pi_m$ with $v(\pi_i, P_{k}, Q_{k})$ ($i = 1, \ldots, m$) we have

$$v(\vec{\pi}(\pi_1, \ldots, \pi_m), P_{k+1}, Q_{k+1}).$$

We also assume that for every pair $\omega, \omega'$ there is a number $k \geq 0$ such that

$$P(\omega) \Rightarrow P_k(\omega) \text{ and } ((P(\omega) \land Q_k(\omega, \omega')) \Rightarrow Q(\omega, \omega')).$$

Then we have $v(\vec{\pi}, P, Q)$.

18. As an example we treat the program "while $B$ do $\sigma$", where $B$ is a predicate and $\sigma$ is a program. It can be written as a recursive procedure (cf. [1]):

```plaintext
procedure $\vec{\pi}$; if $B$ then begin $\sigma$; $\vec{\pi}$ end else $\pi_0$;
```

(where $\pi_0$ is the empty statement). Now $\vec{\pi}$ is given by

$$\vec{\pi}(\epsilon) = \text{"if } B \text{ then begin } \tau; \epsilon \text{ end else } \pi_0".$$}

In order to apply the contents of the previous section, we have to know for what pairs $P, Q, P', Q'$ it is true that for all programs $\pi$ with $v(\pi, P, Q)$ we have $v(\vec{\pi}(\pi), P', Q')$.

The answer is that $(P', Q') < (P_\vec{\pi}, Q_\vec{\pi})$, where

$$P_{\vec{\pi}}(\omega) = (B(\omega) \Rightarrow (P_{\vec{\sigma}}(\omega) \lor \bigvee_{\omega'} (Q(\omega, \omega') \Rightarrow P(\omega))),$$

$$Q_{\vec{\pi}}(\omega, \omega'') = (B(\omega) \land \bigvee_{\omega'} (Q_{\vec{\sigma}}(\omega, \omega') \Rightarrow Q(\omega, \omega''))) \lor (\neg B(\omega) \land (\omega = \omega'')) .$$

Here $P_{\vec{\sigma}}, Q_{\vec{\sigma}}$ is the maximal pair for the program $\tau$.

19. In section 17 there is sentence "for all programs $\pi_1, \ldots, \pi_m$ with ..." This can be replaced by something that does not talk about all programs.
We can say: if we behave as if $\xi_1$ were a program for which $v(\xi_1, P, Q)$ holds, then we obtain, by application of our semantic rules, the truth of $v(\xi_1, \ldots, \xi_m, P_{k+1}, Q_{k+1})$. Needless to say, this requires a more serious discussion of those rules than we attempted here.

20. **Monotonicity rules.** Assume that a program $\pi$ contains a sub-program $\pi_1$, and that, by our semantic rules, we have derived $v(\pi, P, Q)$ using about $\pi_1$ nothing but $v(\pi_1, P_1, Q)$. Compare this with the situation where we start from $v(\pi_1, P_1', Q_1')$, with a pair $(P_1', Q_1') > (P_1, Q_1)$. Then our rules produce the truth of some $v(\pi, P', Q')$ with a pair $(P', Q') > (P, Q)$.

This is in accordance with the interpretation of the predicate pairs as information. If $v(\pi, P, Q)$ is true, then $(P, Q)$ gives semantic information about $\pi$. If $(P', Q') > (P, Q)$, and $v(\pi, P', Q')$ is still true, then $(P', Q')$ gives at least the same amount of information. The maximal amount is the maximal pair (see section 7). It gives all semantic information there is, viz. the full knowledge of the partial mapping.

21. We excluded non-deterministic programs in the previous sections; yet the technique of predicate pairs may apply to them, provided we omit the statements about maximal pairs made in the previous sections.

As an example we take for $\Omega$ the set of reals. The following program operates on $\Omega$:

```
begin integer x; x := x * x; w := x end
```

If we take $P = T$, $Q(w, w') = (w' > -2)$, then $v(\pi, P, Q)$ is true in the sense explained at the end of section 5.

Non-deterministic programs may have little practical value, unless one wants to include in this category the kind of numerical subroutines for which no complete information about rounding-off errors is available.
22. Often we have the situation that in a predicate pair $P_1, Q_1$ the $Q_1$ does not depend on its first variable (i.e. $Q_1$ depends on $\omega'$ only; cf. the end of section 9). The general case can be reduced to this one: If $P$ and $Q$ are given, and if $P_1, Q_1$ are defined by
\[
P_1(\omega) = (\omega = t \land P(t)),
\]
\[
Q_1(\omega, \omega') = Q(t, \omega),
\]
then we have
\[
v(\pi, P, Q) = \forall \tau v(\pi, P_1, Q_1).
\]
It is, however, questionable whether such a reduction is practical.

23. The technique of expressing semantic information by means of pairs is suggested to be quite suitable for the presentation of program correctness by means of an AUTOMATH text. There are various possibilities to do this (in AUTOMATH or in a related language) such that one and the same book contains both the syntactics and the semantics of the program.

We indicate one such possibility here in AUT-QE. For details about the AUTOMATH languages we refer to [1]. For better readability we write \texttt{prop} instead of \texttt{type} if the interpretation is a proposition.

For any type $\Omega$ we assume the existence of a type "program($\Omega$)". We intend to let the objects in this type be programs acting on $\Omega$.
\[
\Omega := \text{--------- type}
\]
\[
\text{program} := \text{PN type}
\]

By means of axioms we introduce primitive programa and composition rules for programs. For example, we describe a partial mapping $g$ with domain $D$ and we postulate the assignment $\omega := g(\omega)$:
For concatenation and if-then-else we write

\[ \pi_1 := \text{program}(\Omega) \]
\[ \pi_2 := \text{program}(\Omega) \]
\[ \text{concatenation} := \text{PN program}(\Omega) \]
\[ B := [\omega, \Omega] \text{prop} \]
\[ \text{ifthenelse} := \text{PN program}(\Omega) \]

So the programs "\(\pi_1;\pi_2\)" and "if \(B\) then \(\pi_1\) else \(\pi_2\)" are denoted by "concatenation \((\pi_1,\pi_2)\)" and "ifthenelse \((\pi_1,\pi_2,B)\)". Similarly we can introduce "while \(B\) do \(\pi\)".

For recursive programs we do the following. We introduce a mapping \(\Psi\) (cf. section 17) that sends programs \(\pi\) into \(\Psi(\pi)\). Now the recursive procedure \(\Pi := \Psi(\Pi)\) is introduced as fixp(\(\Psi\)):

\[ \Psi := [\pi, \text{program}(\Omega)] \text{ program}(\Omega) \]
\[ \text{fixp} := \text{PN program}(\Omega) \]

Having described program syntax this way, we turn to semantics. We introduce \(v(\pi;P,Q)\) as a primitive:
Next we can describe semantic rules (like those of sections 12-18) as mathematical theorems or axioms. This can of course be quite complicated.

References.
