Software Verification

Bertrand Meyer

Carlo A. Furia

Lecture 2: Axiomatic semantics
Program Verification: the very idea

P: a program

max (a, b: INTEGER): INTEGER
  do
    if a > b then
      Result := a
    else
      Result := b
  end
end

S: a specification

require
ture
ensure
Result >= a
Result >= b

Does P ⊧ S hold?

The Program Verification problem:

- Given: a program P and a specification S
- Determine: if every execution of P, for every value of input parameters, satisfies S
What is a theory?

(Think of any mathematical example, e.g. elementary arithmetic)

A theory is a mathematical framework for proving properties about a certain object domain.

Such properties are called theorems.

Components of a theory:

- **Grammar** (e.g. BNF), defines well-formed formulae (WFF)
- **Axioms**: formulae asserted to be theorems
- **Inference rules**: ways to derive new theorems from previously obtained theorems, which can be applied mechanically
Soundness and completeness

How do we know that an axiomatic semantics (or logic) is “right”?

- **Sound**: every theorem (i.e., deduced property) is a true formula
- **Complete**: every true formula can be established as a theorem (i.e., by applying the inference rules).
- **Decidable**: there exists an effective (terminating) process to establish whether an arbitrary formula is a theorem.
Notation

Let $f$ be a well-formed formula

Then

$$\vdash f$$

expresses that $f$ is a theorem
An inference rule is written

\[
f_1, \ f_2, \ldots, \ f_n \quad \text{___________} \quad f_0
\]

It expresses that if \( f_1, \ f_2, \ldots, \ f_n \) are theorems, we may infer \( f_0 \) as another theorem.
Example inference rule

“Modus Ponens” (common to many theories):

\[ p, \quad p \Rightarrow q \]

\[ \quad \therefore q \]
How to obtain theorems

Theorems are obtained from the axioms by zero or more* applications of the inference rules.

*Finite of course
Example: a simple theory of integers

Grammar: Well-Formed Formulae are boolean expressions

- \( i_1 = i_2 \)
- \( i_1 < i_2 \)
- \( \neg b_1 \)
- \( b_1 \implies b_2 \)

where \( b_1 \) and \( b_2 \) are boolean expressions, \( i_1 \) and \( i_2 \) integer expressions

An integer expression is one of

- 0
- A variable \( n \)
- \( f' \) where \( f \) is an integer expression
  (represents “successor”
An axiom and axiom schema

\[ \vdash 0 < 0' \]

\[ \vdash f < g \Rightarrow f' < g' \]
An inference rule

\[ P(0), \quad P(f) \Rightarrow P(f') \]

\[ \therefore P(f) \]
Axiomatic semantics


Purpose:

- Describe the effect of programs through a theory of the underlying programming language, allowing proofs
The theories of interest

Grammar: a well-formed formula is a “Hoare triple”

\{P\} \quad A \quad \{Q\}

Informal meaning: \( A \), started in any state satisfying \( P \), will satisfy \( Q \) on termination
Software correctness (a quiz)

Consider

{\(P\)} A \{Q\}

Take this as a job ad in the classifieds

Should a lazy employment candidate hope for a weak or strong \(P\)? What about \(Q\)?

Two “special offers“:

1. \{\text{False}\} A \{...\}
2. \{...\} A \{\text{True}\}
Axiomatic semantics

“Hoare semantics” or “Hoare logic”: a theory describing the partial correctness of programs, plus termination rules
What is an assertion?

Predicate (boolean-valued function) on the set of computation states

**True**: Function that yields True for all states

**False**: Function that yields False for all states

**P implies Q**: means $\forall s: \text{State}, P(s) \Rightarrow Q(s)$

and so on for other boolean operators
Another view of assertions

We may equivalently view an assertion $P$ as a subset of the set of states (the subset where the assertion yields True):

True: Full State set

False: Empty subset

implies: subset (inclusion) relation

and: intersection or: union
extend(new: G; key: H)

-- Assuming there is no item of key key,
-- insert new with key; set inserted.

require
  key_not_present: not has(key)

ensure
  insertion_done: item(key) = new
  key_present: has(key)
  inserted: inserted
  one_more: count = old count + 1
The case of postconditions

Postconditions are often predicates on two states

Example (Eiffel, in a class \textit{COUNTER}): 

\begin{verbatim}
increment
  require
    count >= 0
  ...
  ensure
    count = old\text{count + 1}
\end{verbatim}
Partial vs total correctness

\{P\} \ A \ \{Q\}

Total correctness:
- A, started in any state satisfying \(P\), will terminate in a state satisfying \(Q\)

Partial correctness:
- A, started in any state satisfying \(P\), will, if it terminates, yield a state satisfying \(Q\)
Assume we want to prove, on integers

\[ \{x > 0\} \land \{y \geq 0\} \]  \[\text{[1]}\]

but have actually proved

\[ \{x > 0\} \land \{y = z^2\} \]  \[\text{[2]}\]

We need properties from other theories, e.g. arithmetic
The mark [EM] will denote results from other theories, taken (in this discussion) without proof

Example:

\[ y = z^2 \quad \text{implies} \quad y \geq 0 \quad \text{[EM]} \]
Rule of consequence

\[ \{P\} \quad A \quad \{Q\}, \quad P' \text{ implies } P, \quad Q \text{ implies } Q' \]

\[ \{P'\} \quad A \quad \{Q'\} \]

Example: \( \{x > 0\} \quad y := x + 2 \quad \{y > 0\} \)
Rule of conjunction

\[
\{P\} \ A \ \{Q\}, \quad \{P\} \ A \ \{R\}
\]

\[
\text{___________}
\]

\[
\{P\} \ A \ \{Q \ \text{and} \ R\}
\]

Example: \{True\} x := 3 \{x > 1 \text{ and } x > 2\}
Axiomatic semantics for a programming language

Example language: Graal (from *Introduction to the theory of Programming Languages*)

Scheme: give an axiom or inference rule for every language construct
Abort

\{\text{False}\} \text{ abort } \{P\}
Sequential composition

\[
\{P\} A \{Q\}, \quad \{Q\} B \{R\}
\]

\[
\{P\} \quad A ; B \quad \{R\}
\]

Example:

\[
\{x > 0\} \quad x := x + 3 ; x := x + 1 \{x > 4\}
\]
Assignment axiom (schema)

\{P [e / x]\} \quad x := e \quad \{P\}

\textit{P [e/x]} is the expression obtained from \( P \) by replacing (substituting) every occurrence of \( x \) by \( e \).
Substitution

\[ x \ [x/x] = \]
\[ x \ [y/x] = \]
\[ x \ [x/y] = \]
\[ x \ [z/y] = \]
\[ 3 \cdot x + 1 \ [y/x] = \]
Applying the assignment axiom

\{y > z - 2\} \ x := x + 1 \ \{y > z - 2\}

\{2 + 2 = 5\} \ x := x + 1 \ \{2 + 2 = 5\}

\{y > 0\} \ x := y \ \{x > 0\}

\{x + 1 > 0\} \ x := x + 1 \ \{x > 0\}
No side effects in expressions!

```
asking_for_trouble (x: in out INTEGER): INTEGER
  do
    x := x + 1;
    global := global + 1;
  Result := 0
end
```

Do the following hold?

```
{global = 0}  u := asking_for_trouble (a)  {global = 0}
{a = 0}       u := asking_for_trouble (a)  {a = 0}
```
The rule of constancy

\{P\} A \{Q\}, \ FV(R) \cap \text{modifies}(A) = \emptyset

\{P \text{ and } R\} \quad A \quad \{Q \text{ and } R\}

\text{FV}(F) = \text{variables free in formula } F
\text{modifies}(A) = \text{variables assigned to in code } A

“Whatever }A\text{ doesn’t modify stays the same”
The rule of constancy: examples

\{ y = 3 \} x := x + 1 \{ y = 3 \}

\{ \forall y \neq 0: y^2 > 0 \} y := y + 1 \{ \forall y \neq 0: y^2 > 0 \}

\{ y = 3 \} x := \sqrt{y} \{ y = 3 \}


\{ bob.age = 65 \} tony.age := 78 \{ bob.age = 65 \}
The frame rule: examples and caveats

\{ y = 3 \} x := x + 1 \{ y = 3 \}

\{ \forall y \neq 0: y^2 > 0 \} y := y + 1 \{ \forall y \neq 0: y^2 > 0 \}

\{ y = 3 \} x := \sqrt{y} \{ y = 3 \}

Only if \sqrt{\cdot} doesn't have side effects on y!


Only if i \neq 3!

\{ bob.age = 65 \} tony.age := 78 \{ bob.age = 65 \}

Only if bob \neq tony, i.e., they are not aliases!
The assignment axiom for arrays

\[
\{ P \ [ \text{if } k = i \text{ then } e \text{ else } a[k] / a[k] ] \} \quad a[i] := e \quad \{ P \}
\]

Example:

\[
\{ 3 = i \text{ or } (3 \neq i \text{ and } a[3] = 2) \}
\quad a[i] := 2
\quad \{ a[3] = 2 \}
\]
Conditional rule

\[
\begin{align*}
\{ \text{P and } c \} & \text{ A } \{ \text{Q} \}, & \{ \text{P and not } c \} & \text{ B } \{ \text{Q} \} \\
\hline
\{ \text{P} \} & \text{ if } c \text{ then } \text{A} & \text{ else } & \text{B} & \text{ end } \{ \text{Q} \}
\end{align*}
\]

Example:

\[
\{ y > 0 \}
\]

\[
\text{if } x > 0 \text{ then } y := y + x \text{ else } y := y - x
\]

\[
\{ y > 0 \}
\]
Conditional rule: example proof

Prove:

\[
\{ m, n, x, y > 0 \text{ and } x \neq y \text{ and } \gcd(x, y) = \gcd(m, n) \}\]

if \( x > y \) then
  \( x := x - y \)
else
  \( y := y - x \)
end

\[
\{ m, n, x, y > 0 \text{ and } \gcd(x, y) = \gcd(m, n) \}\]
Loop rule (partial correctness)

\[
\{P\} \quad A \quad \{I\} \quad \{I \text{ and not } c\} \quad B \quad \{I\}
\]

\[
\{P\} \quad \text{from} \quad A \quad \text{until} \quad c \quad \text{loop} \quad B \quad \text{end} \quad \{I \text{ and } c\}
\]

\{P\} \ A \ \{I\} \text{ proves initiation: the invariant holds initially}

\{I \text{ and not } c\} \ B \ \{I\} \text{ proves consecution (or inductiveness): the invariant is preserved by an arbitrary iteration of the loop}
Loop rule (partial correctness, variant)

\[
\{P\} \ A \ {\{I\}}, \ \{I \text{ and not } c\} \ B \ {\{I\}}, \ \{(I \text{ and } c) \implies Q\}
\]

\[
\{P\} \ \text{from } A \text{ until } c \text{ loop } B \text{ end} \ \{Q\}
\]

Example:
\[
\{y > 3 \text{ and } n > 0\}
\]

\[
\text{from } i := 0 \text{ until } i = n \text{ loop}
\]
\[
i := i + 1
\]
\[
y := y + 1
\]
\[
\text{end}
\]

\[
\{y > 3 + n\}
\]
Loop termination

Must show there is a variant:

Expression \( v \) of type \texttt{INTEGER} such that
(for a loop \texttt{from A until c loop B end} with precondition \( P \)):

1. \( \{P\} \ A \ \{v \geq 0\} \)

2. \( \forall v_0 > 0:\)
   \( \{v = v_0 \text{ and not c}\} \ B \ \{v < v_0 \text{ and } v \geq 0\} \)

You can reuse an invariant to prove 1 and 2.
Loop termination: example

\{ y > 3 \text{ and } n > 0 \}

from i := 0 until i = n loop
  i := i + 1
  y := y + 1
variant
  ??
end

\{ y > 3 + n \}
Computing the maximum of an array

\[
\begin{align*}
\text{from} & \quad i := 0 \; ; \; \text{Result} := a[1] \\
\text{until} & \quad i = a.\text{upper} \\
\text{loop} & \quad i := i + 1 \\
\text{end} & \quad \text{Result} := \max (\text{Result} , a[i])
\end{align*}
\]
Loop as approximation strategy

\[
\begin{align*}
\text{Result} &= a_1 &= \text{Max} (a_1 \ldots a_1) \\
\text{Result} &= \text{Max} (a_1 \ldots a_2) \\
\text{Result} &= \text{Max} (a_1 \ldots a_i) \\
\text{Result} &= \text{Max} (a_1 \ldots a_n)
\end{align*}
\]

Loop body:

\[
i := i + 1
\]

\[
\text{Result} := \text{max}(\text{Result}, a[i])
\]