Software Verification

Bertrand Meyer

Carlo A. Furia

Lecture 2: Axiomatic semantics

Program Verification: the very idea

P: a program

S: a specification

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

```
require
    true

ensure
    Result >= a
    Result >= b
```

Does $P \models S$ 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

Inference rule

An inference rule is written

$$\frac{f_1, \quad f_2, \ldots, \; f_n}{f_0}$$

It expresses that if $f_1, \quad f_2, \ldots \; f_n$ are theorems, we may infer f_0 as another theorem

Example inference rule

"Modus Ponens" (common to many theories):

$$\frac{p, \quad p \Rightarrow q}{q}$$

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

- i1 = i2
- i1 < i2
- ¬ b1
- b1 ⇒ b2

where b1 and b2 are boolean expressions, i1 and i2 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

$$\frac{P(0), \quad P(f) \Rightarrow P(f')}{P(f)}$$

Axiomatic semantics

Floyd (1967), Hoare (1969), Dijkstra (1978)

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"

Instructions

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

Assertions

Informal meaning: A, started in any state satisfying P, will satisfy Q on termination

Software correctness (a quiz)

Consider

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

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":

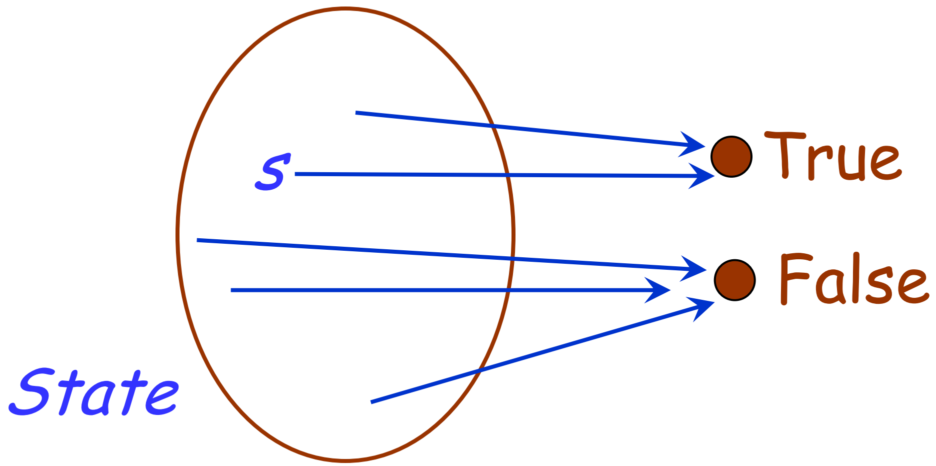
- 1. $\{False\}\quad A\quad \{...\}$
- 2. $\{...\}\quad A\quad \{True\}$

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

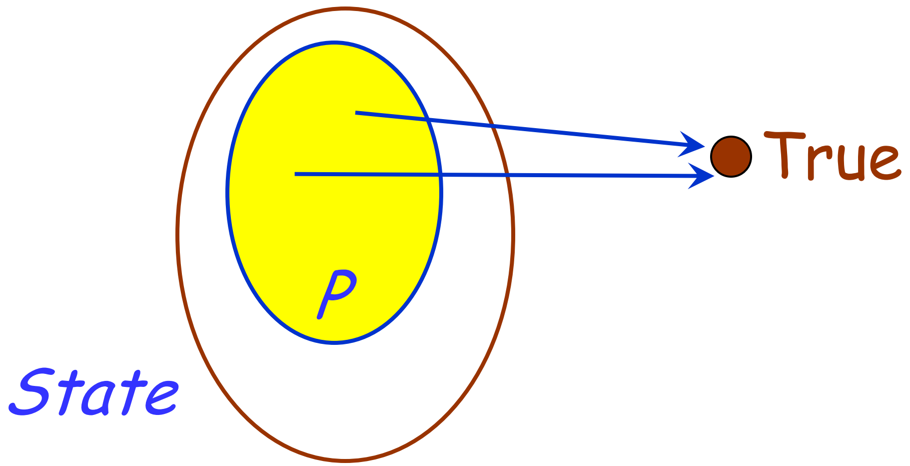
False: Function that yields False for all states

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

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

Application to a programming language: Eiffel

```
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 *COUNTER*):

```
increment
    require
        count >= 0
    ...
    ensure
        count = old count + 1
```

Partial vs total correctness

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

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

Elementary mathematics

Assume we want to prove, on integers

$\{x > 0\}$ A $\{y \geq 0\}$ [1]

but have actually proved

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

We need properties from other theories, e.g. arithmetic

"EM": Elementary Mathematics

The mark [EM] will denote results from other theories, taken (in this discussion) without proof

Example:

$y = z \wedge 2$ **implies** $y \geq 0$ [EM]

Rule of consequence

$$\frac{\{P\}\ A\ \{Q\},\quad P' \textbf{ implies } P,\quad Q \textbf{ implies } Q'}{\{P'\}\ \ A\ \ \{Q'\}}$$

Example: $\{x > 0\}\ y := x + 2\ \{y > 0\}$

Rule of conjunction

$$\frac{\{P\}\ A\ \{Q\},\quad \{P\}\ A\ \{R\}}{\{P\}\quad A\quad \{Q \textbf{ and } R\}}$$

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

Skip

$\{P\}$ **skip** $\{P\}$

Abort

{False} **abort** {P}

Sequential composition

$$\frac{\{P\}\ A\ \{Q\}, \qquad \{Q\}\ B\ \{R\}}{\{P\}\quad A\ ;\ B\quad \{R\}}$$

Example:

$\{x > 0\}\ x := x + 3\ ;\ x := x + 1\ \{x > 4\}$

Assignment axiom (schema)

$$\{P\ [e\ /\ x]\} \qquad x := e \qquad \{P\}$$

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

Substitution

$x\,[x/x] =$

$x\,[y/x] =$

$x\,[x/y] =$

$x\,[z/y] =$

$3 * x + 1\,[y/x] =$

Applying the assignment axiom

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

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

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

$\{x + 1 > 0\}\ x := x + 1\ \{x > 0\}$

Limits to the assignment axiom

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

$$\frac{\{P\}\ A\ \{Q\},\quad FV(R) \cap modifies(A) = \varnothing}{\{P \text{ and } R\}\quad A\quad \{Q \text{ and } R\}}$$

FV(F) = variables free in formula F

modifies(A) = variables assigned to in code A

"Whatever A doesn't modify stays the same"

The rule of constancy: examples

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

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

$\{ y = 3 \}\ x := sqrt(y)\ \{ y = 3 \}$

$\{ a[3] = 0 \}\ a[i] := 2\ \{ a[3] = 0 \}$

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

The frame rule: examples and caveats

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

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

$\{ y = 3 \}\ x := sqrt(y)\ \{ y = 3 \}$

Only if sqrt doesn't have side effects on y!

$\{ a[3] = 0 \}\ a[i] := 2\ \{ a[3] = 0 \}$

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:

$\{ 3 = i \text{ or } (3 \neq i \text{ and } a[3] = 2) \}$

$a[i] := 2$

$\{ a[3] = 2 \}$

Conditional rule

$$\frac{\{P \textbf{ and } c\}\ A\ \{Q\}, \qquad \{P \textbf{ and not } c\}\ B\ \{Q\}}{\{P\}\quad \textbf{if } c \textbf{ then } A \textbf{ else } B \textbf{ end } \{Q\}}$$

Example:

{y > 0}

if x > 0 then y := y + x else y := y - x

{y > 0}

Conditional rule: example proof

Prove:

{ $m, n, x, y > 0$ **and** $x \neq y$ **and** $gcd(x, y) = gcd(m, n)$ }

```
if x > y then
    x := x - y
else
    y := y - x
end
```

{ $m, n, x, y > 0$ **and** $gcd(x, y) = gcd(m, n)$ }

Loop rule (partial correctness)

$$\frac{\{P\}\ A\ \{I\} \qquad \{I \textbf{ and not } c\}\ B\ \{I\}}{\{P\}\ \textbf{from}\ A\ \textbf{until}\ c\ \textbf{loop}\ B\ \textbf{end}\ \{I \textbf{ and } c\}}$$

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

{I **and not** c} B {I} proves consecution (or inductiveness): the invariant is preserved by an arbitrary iteration of the loop

Loop rule (partial correctness, variant)

$$\frac{\{P\}\ A\ \{I\},\quad \{I \textbf{ and not } c\}\ B\ \{I\},\quad \{(I \textbf{ and } c) \textbf{ implies } Q\}}{\{P\}\ \textbf{from}\ A\ \textbf{until}\ c\ \textbf{loop}\ B\ \textbf{end}\ \{Q\}}$$

Example:

```
{y > 3 and n > 0}
        from i := 0 until i = n loop
                i := i + 1
                y := y + 1
        end
{y > 3 + n}
```

Loop termination

Must show there is a variant:

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

1. $\{P\}\ A\ \{v \geq 0\}$

2. $\forall\ v0 > 0$:

   $\{v = v0$ **and not** $c\}\ B\ \{v < v0$ **and** $v \geq 0\}$

You can reuse an invariant to prove 1 and 2.

Loop termination: example

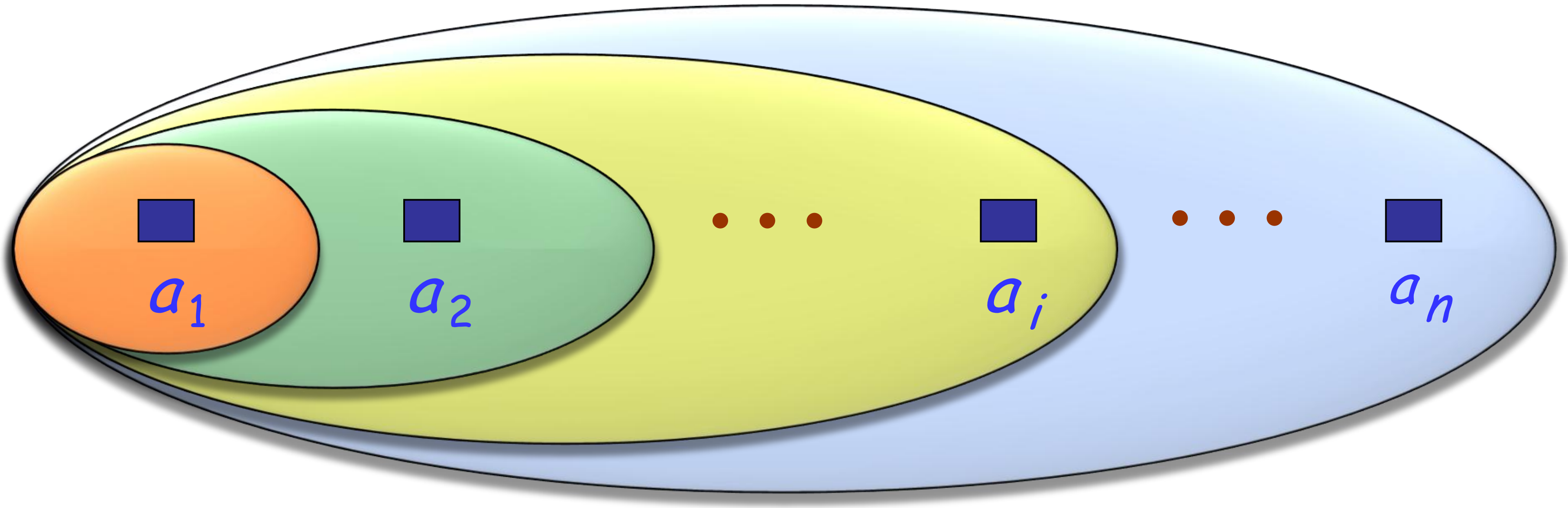
```
{y > 3 and n > 0}
        from i := 0 until i = n loop
                i := i + 1
                y := y + 1
        variant
                ??
        end
{y > 3 + n}
```

Computing the maximum of an array

```
from
        i := 0 ; Result := a [1]
until

        i = a.upper
loop
        i := i + 1
        Result := max (Result , a [i])
end
```

Loop as approximation strategy



Loop body:

$i := i + 1$
Result := max (**Result** , $a[i]$)

Result = a_1 = $Max\ (a_1 .. a_1)$

Result = $Max\ (a_1 .. a_2)$

Result = $Max\ (a_1 .. a_i)$

The loop invariant

Result = $Max\ (a_1 .. a_n)$