Théorie des Langages de programmation: Axiomatic Semantics of While

Giovanni Coray - Vincenzo Pallotta

May 1, 2000

Abstract

In this chapter we turn to the business of systematic verification of programs in **While**. The Hoare rules for showing the partial correctness of programs are introduced and shown sound. This involves extending the boolean expressions to a rich language of assertions about program states. The chapter concludes with an example of verification conducted within the framework of Hoare rules.

References:

- Chapter 6 of "The Formal Semantics of Programming Languages: An Introduction" - Glynn Winskel, MIT Press.
- Chapter 6 of "Semantics with Applications: A formal introduction" - H.R. Nielson & F. Nielson, Wiley.
- "Verification of Sequential and Concurrent Programs" - K. Apt & E.R. Olderog, Springer.

Formal methods for program verification

Problem: Prove that a program is correct with respect to a given specification

Solutions:

- Using Operational Semantics:
  - reasoning on possible (infinite) computations
  - hard for deterministic programs
  - untractable for non-deterministic programs
- Using Axiomatic Semantics
  - use a meta-language based on first order predicate logic to express "assertions"
  - use a formal system do proof assertions
  - if proofs are constructive then it is possible to "derive" programs from assertions (specifications).

History of Axiomatic Semantics

Turing [1949]: formal proof of correctness of Turing Machines

Floyd [1967]: programs correctness reasoning on flows diagrams

Hoare [1969]: extension of Floyd's logic to structured programming

Dijkstra [1976]: verification and synthesis of programs

Cook [1978]: completeness of Hoare's calculus

Hoare's Logic

Hoare's logic is based on the concept of Hoare's triple:

$$\models \{p\}S\{q\}$$

where S is an instruction (e.g. a program) and p, q are assertions. There are two possible interpretation of Hoare's triples

- using operational semantics:

  Partial correctness: $\models_{par} \{p\}S\{q\}$iff

  $$\forall \sigma \in \Sigma.(\sigma \models p \wedge \langle S, \sigma \rangle \to^* \sigma') \Rightarrow (\sigma' \models q)$$

  Total correctness: $\models_{tot} \{p\}S\{q\}$ iff

  $$\forall \sigma \in \Sigma.(\sigma \models p) \Rightarrow (\langle S, \sigma \rangle \to^* \sigma' \wedge \sigma' \models q)$$

- using denotational semantics:

  Partial correctness: $\models_{par} \{p\}S\{q\}$iff

  $$\forall \sigma \in \Sigma.(\sigma \models p \wedge \mathcal{C}[\![S]\!]\sigma \neq \perp) \Rightarrow \mathcal{C}[\![S]\!]\sigma \models q$$

  Total correctness: $\models_{tot} \{p\}S\{q\}$ iff

  $$\forall \sigma \in \Sigma.\sigma \models p \Rightarrow (\mathcal{C}[\![S]\!]\sigma \neq \perp \wedge \mathcal{C}[\![S]\!]\sigma \models q$$

Backgrounds (1)

Types:

- Basic types: $Int, Bool$
- Higher types: $T_1 \times \ldots \times T_n \rightarrow T$ where $T_1, \ldots, T_n, T$ are basic types.

Variables: simple variables and arrays (higher type variables).

Constants:

- basic type constans
- higher type constants (relations $T_1 \times \ldots \times T_n \rightarrow Bool$ and functions $T_1 \times \ldots \times T_n \rightarrow T$)
- $|\,| : Int \rightarrow Int$,
  $+, -, *, min, max, div, mod : Int \times Int \rightarrow Int$
- $=_{Int}, <, divides : Int \times Int \rightarrow Bool$,
  $\neg : Bool \rightarrow Bool$,
  $=_{Bool}, \wedge, \vee, \rightarrow, \leftrightarrow : Bool \times Bool \rightarrow Bool$

Backgrounds (2)

Expressions: simple variable, basic type constants and:

- if $s_1, \ldots, s_n$ are expressions of type $T_1, \ldots T_n$ respectively and op is a constant of higher type $T_1 \times \cdots \times T_n \rightarrow T$ then $op(s_1, \ldots, s_n)$ is an expression of type T and,
- if a is an array of type $T_1 \times \cdots \times T_n \rightarrow T$ then $a[s_1, \ldots, s_n]$ is an expression of type T
- if B is a boolean expression and s_1, s_2 are expression of type T then $Cond(B, s_1, s_2)$ is an expression of type T.

Structure: A structure is a pair $\mathcal{S} = (\mathcal{D}, \mathcal{I})$ where:

- $\mathcal{D} = \bigcup_{T\, is\, a\, type} \mathcal{D}_\mathcal{T} \wedge \forall T_1, T_2 : T_1 \neq T_2 \Rightarrow T_1 \cap T_2 = \emptyset$
- $\mathcal{I}$ is an *interpretation* of constants which assigns to each constant $c : T$ a value $\mathcal{I}(c) \in \mathcal{D}_\mathcal{T}$.
  Observe that for an higher types constants op, $\mathcal{I}(op)$ denotes a function.

States: a proper state is a total mapping from variables to values of corresponding domain type. The set of all proper states is denoted by Σ.

Backgrounds (3)

Semantics of expressions: The semantics of an expression s of type T in the structure $\mathcal{S}$ is a mapping: $\mathcal{S}[\![s]\!] : \Sigma \rightarrow \mathcal{D}_\mathcal{T}$ inductively defined by:

- if s is a simple variable then $\mathcal{S}[\![s]\!]\sigma = \sigma(s)$
- if s is a constant of basic type then $\mathcal{S}[\![s]\!]\sigma = \mathcal{I}(s)$
- if $s \equiv op(s_1, \ldots, s_n)$ is a constant of higher type then $\mathcal{S}[\![s]\!]\sigma = \mathcal{I}(op)(\mathcal{S}[\![s]\!]_1\sigma, \ldots, \mathcal{S}[\![s]\!]_n\sigma)$
- if $s \equiv a(s_1, \ldots, s_n)$ for some array a then $\mathcal{S}[\![s]\!]\sigma = \sigma(a)[\mathcal{S}[\![s]\!]_1\sigma, \ldots, \mathcal{S}[\![s]\!]_n\sigma]$
- if $s \equiv Cond(B, s_1, s_2)$ for some boolean expression B then

$$\mathcal{S}[\![s]\!]\sigma = \begin{cases} \mathcal{S}[\![s]\!]_1\sigma & if\ \mathcal{S}[\![B]\!]\sigma = \mathbf{tt} \\ \mathcal{S}[\![s]\!]_2\sigma & if\ \mathcal{S}[\![B]\!]\sigma = \mathbf{ff} \end{cases}$$

- if $s \equiv (s_1)$ then $\mathcal{S}[\![s]\!]\sigma = \mathcal{S}[\![s]\!]_1\sigma$

Backgrounds (4)

Update of a state:

- if u is a simple variable then

$$\sigma[u \mapsto d](v) = \begin{cases} d & if\ u \equiv v \\ \sigma(v) & otherwise \end{cases}$$

- if $u \equiv a[t_1, \ldots t_n]$ then
  $\sigma[u \mapsto d](v) = \sigma(v)$ if $a \not\equiv v$
  $\sigma[u \mapsto d](a) =$
  $\begin{cases} \lambda d_1, \ldots, d_n.d & if\ d_i = \sigma(t_i),\ i \in \{1, \ldots, n\} \\ \lambda d_1, \ldots, d_n.\sigma(a) & otherwise \end{cases}$

Example: Let a be an array of integers and suppose that $\sigma(x) = 3$ then for all simple variables y:

$$\begin{aligned} \sigma[a[x+1] \mapsto 2](y) &= \sigma(y) \\ \sigma[a[x+1] \mapsto 2](a)(4) &= 2 \end{aligned}$$

Backgrounds (5)

Assertion Language It is an extension of boolean expressions syntactic category defined inductively as:

- every Boolean expression is an assertion
- if p, q are assertions, then $\neg p, (p \vee q), (p \wedge q), (p \rightarrow q)$ and $(p \leftrightarrow q)$ are also assertions
- if x is a simple variable and p an assertion, then $\exists x : p$ and $\forall x : p$ are also assertions
- nothing else is an assertion.

Semantics of assertions: Given a state σ and an assertion p then $\sigma \models$ if and only if:

- for Boolean expression if $p = B$ then B: σ iff $\mathcal{B}[\![B]\!]\sigma = \mathbf{tt}$
- for logical connectives: as usual
- for universal quantifier applied to a simple variable x of type T: $\sigma \models \forall x : p$ iff $\sigma[x \mapsto d] \models p$ for all data values from $\mathcal{D}_{\mathcal{T}}$
- for existential quantifier applied to a simple variable x of type T: $\sigma \models \exists x : p$ iff $\sigma[x \mapsto d] \models p$ for some data values from $\mathcal{D}_{\mathcal{T}}$

Meaning of assertion:
$[\![p]\!] = \{\sigma \mid \sigma$ is a proper state and $\sigma \models p\}$.

Substitutions (1)

Let be s an expression

- if $s \equiv x$ for some simple variable x then
$$s[u \mapsto t] \equiv \begin{cases} t & if\ s \equiv u \\ s & otherwise \end{cases}$$
- if $s \equiv c$ for some constant c then $s[u \mapsto t] \equiv s$
- if $s \equiv op(s_1, \ldots, s_n)$ for some constant operation $op : T_1 \times \cdots \times T_n \rightarrow T$ then $s[u \mapsto t] \equiv op(s_1[u \mapsto t], \ldots, s_n[u \mapsto t])$
- if $s \equiv a[s_1, \ldots, s_n]$ for some array a and u is a simple variable or a subscripted variable $b[t_1, \ldots, t_k]$ where $a \not\equiv b$ then $s[u \mapsto t] \equiv a[s_1[u \mapsto t], \ldots, s_n[u \mapsto t]]$
- if $s \equiv a[s_1, \ldots, s_n]$ for some array a and $u \equiv a[t_1, \ldots, t_n]$ then
$$s[u \mapsto t] \equiv Cond(\textstyle\bigwedge_{i=1}^{n} s_i[u \mapsto t] = t_i,\ t,\ a[s_1[u \mapsto t], \ldots, s_n[u \mapsto t]])$$
- if $s \equiv Cond(B, s_1, s_2)$ then $s[u \mapsto t] \equiv Cond(B[u \mapsto t], s_1[u \mapsto t], s_2[u \mapsto t])$

Substitutions (2)

Lemma. *Identical Substitutions*
For all expressions s and t, all simple variables x and all subscripted variables $a[t_1, \ldots, t_n]$:

1. $s[x \mapsto t] \equiv s$ *if s does not contains x*
2. $s[a[t_1, \ldots t_n] \mapsto t] \equiv s$ *if s does not contain a.*

Substitutions (3)

Example: Suppose that a and b are arrays of type $Int \rightarrow Int$ and x is an integer variable, then:

$$\begin{array}{ll} & a[b[x]][b[1] \mapsto 2] \\ \equiv & \\ & a[b[x][b[1] \mapsto 2]] \\ \equiv & \\ & a[Cond(x[b[1] \mapsto 2] = 1, 2, b[x[b[1] \mapsto 2]]) \\ \equiv & \\ & a[Cond(x = 1, 2, b[x])] \end{array}$$

Substitutions can be easily extended to assertions (exercise).

Example: When dealing with quantified assertions it can be necessary to introduce "fresh" variables to avoid possible clashes with free occurrences in the term to be substituded as for example:

$$\begin{array}{ll} & (\exists x : z = 2 * x)[z \mapsto x + 1] \\ \equiv & \exists y : z = 2 * x[x \mapsto y][z \mapsto x + 1] \\ \equiv & \exists y : x + 1 = 2 * y \end{array}$$

Substitutions (4)

Lemma. *Substitution*

For all expressions s and t, all assertions p, all simple or subscripted variables u of the same type as t and all proper states $\sigma \neq \perp$ then:

1. $\sigma(s[u \mapsto t]) = \sigma[u \mapsto \sigma(t)](s)$

2. $\sigma \models p[u \mapsto t]$ *iff* $\sigma[u \mapsto \sigma(t)] \models p$.

Example Let a, b arrays of type $Int \to Int$, x an integer variable and σ a proper state such that $\sigma(x) = 1$ and $\sigma(a)(1) = 2$. Then

$$\begin{aligned}
& a[b[1] \mapsto 2](a[b[x]]) \\
= \; & \{Substitution\, lemma\} \\
& \sigma(a[b[x]][b[1] \mapsto 2]) \\
= \; & \{previous\, example\} \\
& \sigma(a[Cond(x = 1, 2, b[x])]) \\
= \; & \{definition\, of\, \sigma\} \\
& \sigma(a)(\sigma(Cond(x = 1, 2, b[x])) \\
= \; & \{\sigma(x) = 1,\; def.\, of\; \sigma(s)\} \\
& \sigma(a)(\sigma(2)) \\
= \; & \sigma(a[2]).
\end{aligned}$$

The Hoare's Calculus

The proof system PD for partial correctness is given by the following rules:

Skip $\vdash_{PD} \{p\}\mathbf{skip}\{p\}$

Assignement $\vdash_{PD} \{p[x \mapsto t]\}x := t\{p\}$

Composition

$$\frac{\vdash_{PD} \{p\}S_1\{r\},\ \vdash_{PD} \{r\}S_2\{q\}}{\vdash_{PD} \{p\}S_1; S_2\{q\}}$$

Conditional

$$\frac{\vdash_{PD} \{p \wedge B\}S_1\{q\},\ \vdash_{PD} \{p \wedge \neg B\}S_2\{q\}}{\vdash_{PD} \{p\}\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\{q\}}$$

While

$$\frac{\vdash_{PD} \{p \wedge B\}S\{p\}}{\vdash_{PD} \{p\}\mathbf{while}\ B\ \mathbf{do}(S)\{p \wedge \neg B\}}$$

Consequence

$$\frac{\models p \to p_1,\ \vdash_{PD} \{p_1\}S\{q_1\},\ \models q_1 \to q}{\vdash_{PD} \{p\}S\{q\}}$$

Theorem. *Soundess and (relative) Completeness*

$$\models_{par} \{p\}S\{q\}\ \mathbf{iff}\ \vdash_{PD} \{p\}S\{q\}$$

Observe that in the consequence rule we assume validity of assertions thus avoiding the Gödel incompleteness theorem!

Total Correctness

The proof system for total correctness TD is obtained from PD substituting the rule for **while**:

While

$$\frac{\begin{array}{l}\vdash_{TD} \{p \wedge B\}S\{p\} \\ \vdash_{TD} \{p \wedge B \wedge t = z\}S\{t < z\} \\ \models p \to (t \geq 0)\end{array}}{\vdash_{TD} \{p\}\mathbf{while}\, B\, \mathbf{do}\,(S)\{p \wedge \neg B\}}$$

where t is an integer expression and z is an integer variable that does not appear in p, B, t, S. Expression t is called *bound function* of the loop **while** B **do** (S).

Theorem. *Soundess and (relative) Completeness*

$$\models_{tot} \{p\}S\{q\}\ \mathbf{iff}\ \vdash_{TD} \{p\}S\{q\}$$

Observe that in the consequence rule we assume validity of assertions thus avoiding the Gödel incompleteness theorem!

Example: the division algorithm

Let $DIV \equiv INIT; S_0$ where:

$$\begin{aligned}
INIT &\equiv quo := 0; rem := x \\
S_0 &\equiv \mathbf{while}\, rem \geq y\, \mathbf{do}\, LOOP \\
LOOP &\equiv (rem := rem - y; quo := quo + 1).
\end{aligned}$$

The program DIV is given entirely:

$$\begin{aligned}
DIV \equiv\ & quo := 0; rem := x; \\
& \mathbf{while}\, rem \geq y\, \mathbf{do} \\
& (rem := rem - y; \\
& quo := quo + 1)
\end{aligned}$$

Show that

$$\vdash_{PD} \{x \geq 0 \wedge y \geq 0\}DIV\{quo{*}y{+}rem = x \wedge 0 \leq rem < y\}$$