Hoare Logic

`http://www.cl.cam.ac.uk/~mjcg/HoareLogic.html`

- **Program specification using Hoare notation**
- **Axioms and rules of Hoare Logic**
- **Soundness and completeness**
- **Mechanised program verification**
- **Pointers, the frame problem and separation logic**

A Little Programming Language ✓

Expressions:

$E ::= \quad N \mid V \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid \ldots$

Boolean expressions:

$B ::= \quad \mathtt{T} \mid \mathtt{F} \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \ldots$

Commands:

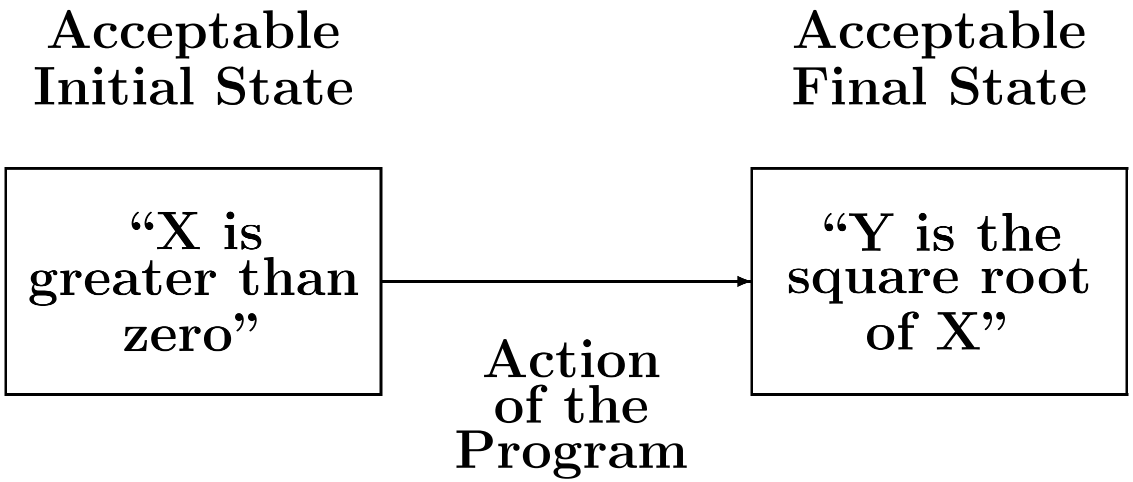
$C ::= \; V \; \mathtt{:=} \; E$

$\quad \mid \quad C_1 \; \mathtt{;} \; C_2$

$\quad \mid \quad \mathtt{IF} \; B \; \mathtt{THEN} \; C_1 \; \mathtt{ELSE} \; C_2$

$\quad \mid \quad \mathtt{WHILE} \; B \; \mathtt{DO} \; C$

Specification of Imperative Programs



Hoare's notation ✓

- **C.A.R. Hoare introduced the following notation called a** *partial correctness specification* **for specifying what a program does:**

  $$\{P\}\ C\ \{Q\}$$

  where:

  - C **is a command**
  - P **and** Q **are conditions on the program variables used in** C

- **Conditions on program variables will be written using standard mathematical notations together with** *logical operators* **like:**

  - $\wedge$ **('and'),** $\vee$ **('or'),** $\neg$ **('not'),** $\Rightarrow$ **('implies')**

- **Hoare's original notation was** $P\ \{C\}\ Q$ **not** $\{P\}\ C\ \{Q\}$**, but the latter form is now more widely used**

Meaning of Hoare's Notation

- $\{P\}\ C\ \{Q\}$ **is true if**
  - **whenever** C **is executed in a state satisfying** P
  - **and** *if* **the execution of** C **terminates**
  - **then the state in which** C **terminates satisfies** Q
- **Example:** $\{\mathtt{X} = 1\}\ \mathtt{X:=X+1}\ \{\mathtt{X} = 2\}$
  - P **is the condition that the value of** `X` **is** `1`
  - Q **is the condition that the value of** `X` **is** `2`
  - C **is the assignment command** `X:=X+1`
    - **i.e. '**`X` **becomes** `X+1`**'**
- $\{\mathtt{X} = 1\}\ \mathtt{X:=X+1}\ \{\mathtt{X} = 2\}$ **is true**
- $\{\mathtt{X} = 1\}\ \mathtt{X:=X+1}\ \{\mathtt{X} = 3\}$ **is false**

Hoare Logic and Verification Conditions ✓

- **Hoare Logic is a deductive proof system for Hoare triples $\{P\}\ C\ \{Q\}$**

- **Can use Hoare Logic directly to verify programs**
  - **original proposal by Hoare**
  - **tedious and error prone**
  - **impractical for large programs**

- **Can 'compile' proving $\{P\}\ C\ \{Q\}$ to verification conditions**
  - **more natural**
  - **basis for computer assisted verification**

- **Proof of verification conditions equivalent to proof with Hoare Logic**
  - **Hoare Logic can be used to explain verification conditions**

Partial Correctness Specification

- **An expression $\{P\}\ C\ \{Q\}$ is called a *partial correctness specification***
  - **P is called its *precondition***
  - **Q its *postcondition***
- **$\{P\}\ C\ \{Q\}$ is true if**
  - **whenever C is executed in a state satisfying P**
  - **and *if* the execution of C terminates**
  - **then the state in which C's execution terminates satisfies Q**
- **These specifications are 'partial' because for $\{P\}\ C\ \{Q\}$ to be true it is *not* necessary for the execution of C to terminate when started in a state satisfying P**
- **It is only required that *if* the execution terminates, *then* Q holds**
- $\{\mathtt{X}=1\}$ `WHILE T DO X:=X` $\{\mathtt{Y}=2\}$ – **this specification is true!**

Total Correctness Specification

- **A stronger kind of specification is a** *total correctness specification*
  - **there is no standard notation for such specifications**
  - **we shall use** $[P]\ C\ [Q]$
- **A total correctness specification** $[P]\ C\ [Q]$ **is true if and only if**
  - **whenever** C **is executed in a state satisfying** P **the execution of** C **terminates**
  - **after** C **terminates** Q **holds**
- $[\mathtt{X} = 1]$ `Y:=X; WHILE T DO X:=X` $[\mathtt{Y} = 1]$
  - **this says that the execution of** `Y:=X;WHILE T DO X:=X` **terminates when started in a state satisfying** $\mathtt{X} = 1$
  - **after which** $\mathtt{Y} = 1$ **will hold**
  - **this is clearly false**

Total Correctness

- **Informally:**

  Total correctness = Termination + Partial correctness

- **Total correctness is the ultimate goal**
  - **usually easier to show partial correctness and termination separately**

- **Termination is usually straightforward to show, but there are examples where it is not: no one knows whether the program below terminates for all values of `X`**

```
WHILE X>1 DO
  IF ODD(X) THEN X := (3×X)+1 ELSE X := X DIV 2
```

  - `X DIV 2` **evaluates to the result of rounding down** `X/2` **to a whole number**
  - **the Collatz conjecture is that this terminates with** `X=1`

- **Microsoft's T2 tool proves systems code terminates**

Auxiliary Variables ✓

- $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X; X:=Y; Y:=R` $\{$`X=y` $\wedge$ `Y=x`$\}$

  - this says that *if* the execution of

    `R:=X; X:=Y; Y:=R`

    terminates (which it does)

  - *then* the values of `X` and `Y` are exchanged

- The variables `x` and `y`, which don't occur in the command and are used to name the initial values of program variables `X` and `Y`

- They are called *auxiliary* variables or *ghost* variables

- Informal convention:

  - program variable are upper case

  - auxiliary variable are lower case

Floyd-Hoare Logic

✓

- **To construct formal proofs of partial correctness specifications,** *axioms* **and** *rules of inference are needed*

- **This is what Floyd-Hoare logic provides**
  - **the formulation of the deductive system is due to Hoare**
  - **some of the underlying ideas originated with Floyd**

- **A proof in Floyd-Hoare logic is a sequence of lines, each of which is either an** *axiom* **of the logic or follows from earlier lines by a** *rule of inference* **of the logic**
  - **proofs can also be trees, if you prefer**

- **A formal proof makes explicit what axioms and rules of inference are used to arrive at a conclusion**

Judgements

- **Three kinds of things that could be true or false:**
  - **statements of mathematics, e.g.** $(\mathtt{X}+1)^2 = \mathtt{X}^2 + 2 \times \mathtt{X} + 1$
  - **partial correctness specifications** $\{P\}\ C\ \{Q\}$
  - **total correctness specifications** $[P]\ C\ [Q]$

- **These three kinds of things are examples of** *judgements*
  - **a logical system gives rules for proving judgements**
  - **Floyd-Hoare logic provides rules for proving partial correctness specifications**
  - **the laws of arithmetic provide ways of proving statements about integers**

- $\vdash S$ **means statement** S **can be proved**
  - **how to prove predicate calculus statements assumed known**
  - **this course covers axioms and rules for proving** *program correctness statements*

Reminder of our little programming language

- **The proof rules that follow constitute an** *axiomatic semantics* **of our programming language**

Expressions

$$E \ ::= \ N \mid V \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid \ldots$$

Boolean expressions

$$B \ ::= \ \texttt{T} \mid \texttt{F} \mid E_1 {=} E_2 \mid E_1 \leq E_2 \mid \ldots$$

Commands

$C \ ::= \ V$ `:=` E	**Assignments**
$\mid \ C_1$ `;` C_2	**Sequences**
$\mid$ `IF` B `THEN` C_1 `ELSE` C_2	**Conditionals**
$\mid$ `WHILE` B `DO` C	**WHILE-commands**

Substitution Notation

- $Q\,\texttt{[}E/V\texttt{]}$ **is the result of replacing all occurrences of V in Q by E**
  - **read $Q\,\texttt{[}E/V\texttt{]}$ as 'Q with E for V'**
  - **for example:** $(\texttt{X}+1 > \texttt{X})\texttt{[Y+Z/X]} \;=\; ((\texttt{Y+Z})+1 > \texttt{Y+Z})$
  - **ignoring issues with bound variables for now (e.g. variable capture)**
- **Same notation for substituting into terms, e.g.** $E_1\,\texttt{[}E_2/V\texttt{]}$
- **Think of this notation as the 'cancellation law'**

$$V\,\texttt{[}E/V\texttt{]} \;=\; E$$

which is analogous to the cancellation property of fractions

$$v \times (e/v) \;=\; e$$

- **Note that $Q\,\texttt{[}x/V\texttt{]}$ doesn't contain V (if $V \neq x$)**

The Assignment Axiom (Hoare)

- **Syntax:** V `:=` E
- **Semantics: value of V in final state is value of E in initial state**
- **Example:** `X:=X+1` **(adds one to the value of the variable** `X`**)**

The Assignment Axiom
$\vdash\ \{Q[E/V]\}\ V\texttt{:=}E\ \{Q\}$
Where V is any variable, E is any expression, Q is any statement.

- **Instances of the assignment axiom are**
  - $\vdash\ \{\mathtt{E} = \mathtt{x}\}\ \mathtt{V} := \mathtt{E}\ \{\mathtt{V} = \mathtt{x}\}$
  - $\vdash\ \{\mathtt{Y} = 2\}\ \mathtt{X} := 2\ \{\mathtt{Y} = \mathtt{X}\}$
  - $\vdash\ \{\mathtt{X} + 1 = \mathtt{n} + 1\}\ \mathtt{X} := \mathtt{X} + 1\ \{\mathtt{X} = \mathtt{n} + 1\}$
  - $\vdash\ \{E = E\}\ \mathtt{X} := E\ \{\mathtt{X} = E\}$ **(if** `X` **does not occur in E)**

The Backwards Fallacy ✓

- **Many people feel the assignment axiom is 'backwards'**

- **One common erroneous intuition is that it should be**

$$\vdash\ \{P\}\ V\texttt{:=}E\ \{P\texttt{[}V/E\texttt{]}\}$$

  - **where $P\texttt{[}V/E\texttt{]}$ denotes the result of substituting V for E in P**
  - **this has the false consequence $\vdash\ \{\texttt{X=0}\}\ \texttt{X:=1}\ \{\texttt{X=0}\}$**
    (since `(X=0)[X/1]` is equal to `(X=0)` as `1` doesn't occur in `(X=0)`)

- **Another erroneous intuition is that it should be**

$$\vdash\ \{P\}\ V\texttt{:=}E\ \{P\texttt{[}E/V\texttt{]}\}$$

  - **this has the false consequence $\vdash\ \{\texttt{X=0}\}\ \texttt{X:=1}\ \{\texttt{1=0}\}$**
    (which follows by taking P to be `X=0`, V to be `X` and E to be `1`)

Validity

✓

- **Important to establish the validity of axioms and rules**

- **Later will give a *formal semantics* of our little programming language**

  - **then *prove* axioms and rules of inference of Floyd-Hoare logic are sound**

  - **this will only increase our confidence in the axioms and rules to the extent that we believe the correctness of the formal semantics!**

- **The Assignment Axiom is not valid for 'real' programming languages**

  - **In an early PhD on Hoare Logic G. Ligler showed that the assignment axiom can fail to hold in six different ways for the language Algol 60**

Expressions with Side-effects

✓

- **The validity of the assignment axiom depends on expressions not having side effects**

- **Suppose that our language were extended so that it contained the 'block expression'**

```
BEGIN Y:=1; 2 END
```

  - **this expression has value `2`, but its evaluation also 'side effects' the variable `Y` by storing `1` in it**

- **If the assignment axiom applied to block expressions, then it could be used to deduce**

$$\vdash \ \{\texttt{Y=0}\}\ \texttt{X:=BEGIN Y:=1; 2 END}\ \{\texttt{Y=0}\}$$

  - **since `(Y=0)[E/X] = (Y=0)` (because `X` does not occur in `(Y=0)`)**

  - **this is clearly false; after the assignment `Y` will have the value `1`**

A Forwards Assignment Axiom (Floyd)

- **This is the original semantics of assignment due to Floyd**

$$\vdash\ \{P\}\ V\texttt{:=}E\ \{\exists v.\ V = E[v/V]\ \wedge\ P[v/V]\}$$

  - **where** v **is a new variable (i.e. doesn't equal** V **or occur in** P **or** E**)**

- **Example instance**

$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\exists v.\ \texttt{X} = \texttt{X+1}[v/\texttt{X}]\ \wedge\ \texttt{X=1}[v/\texttt{X}]\}$$

- **Simplifying the postcondition**

$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\exists v.\ \texttt{X} = \texttt{X+1}[v/\texttt{X}]\ \wedge\ \texttt{X=1}[v/\texttt{X}]\}$$
$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\exists v.\ \texttt{X} = v + \texttt{1}\ \wedge\ v = \texttt{1}\}$$
$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\exists v.\ \texttt{X} = \texttt{1} + \texttt{1}\ \wedge\ v = \texttt{1}\}$$
$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\texttt{X} = \texttt{1} + \texttt{1}\ \wedge\ \exists v.\ v = \texttt{1}\}$$
$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\texttt{X} = \texttt{2}\ \wedge\ \texttt{T}\}$$
$$\vdash\ \{\texttt{X=1}\}\ \texttt{X:=X+1}\ \{\texttt{X} = \texttt{2}\}$$

- **Forwards Axiom equivalent to standard one but harder to use**