**High Integrity Computing (F23PS3)**

**High Integrity Software Development (F29HD2)**

# Program Specification

**Andrew Ireland**

**School of Mathematical and Computer Sciences**
**Heriot-Watt University**
**Edinburgh**



## Overview

- Specifying correctness of imperative programs.
- Logical basis for proving programs correct.

## Some Background

- Long history:
  - Goldstine & von Neumann 1947
  - Turing 1949
  - Floyd 1967
  - Hoare 1969
- Current applications & trends:
  - ESC/Java (HP Labs)
  - SPARK (Praxis)
  - SLAM (Microsoft Research)
  - Proof Carrying Code



## Imperative Program Correctness

- Correctness is a relative concept. Here we are concerned with establishing the correctness of imperative code with respect to some specification of **desired behaviour**.
- By desired behaviour we refer to how we expect the execution of a program to effect program state, *i.e.* the variables and their associated bindings.
- Our specifications correspond to logical assertions about program states, *e.g.*

$$\begin{array}{c} X = 2 \\ \wedge \\ Y = 3 \end{array} \Rightarrow \boxed{\text{Swap Program}} \Rightarrow \begin{array}{c} X = 3 \\ \wedge \\ Y = 2 \end{array}$$

**Before state** **After state**

## Correctness Specifications

**Partial correctness:**

- $\{P\}\,C\{Q\}$
- $P$ and $Q$ are propositions and $C$ is code.
- IF $P$ is true before $C$ is executed AND the execution of $C$ terminates THEN $Q$ is true after the execution terminates.

**Total correctness:**

- $[P]\ C\ [Q]$
- IF $P$ is true before $C$ is executed THEN the execution of $C$ will terminate and $Q$ will be true on termination.
- total correctness = partial correctness + termination



## A Toy Programming Language

$$
\begin{array}{lll}
\langle \textit{command} \rangle ::= & \langle \textit{variable} \rangle := \langle \textit{term} \rangle \mid \\
 & \langle \textit{command} \rangle;\quad \ldots ;\ \langle \textit{command} \rangle \mid \\
 & \textbf{if}\ \langle \textit{statement} \rangle\ \ \textbf{then}\ \langle \textit{command} \rangle\ \textbf{end if} \mid \\
 & \textbf{if}\ \langle \textit{statement} \rangle\ \ \textbf{then}\ \langle \textit{command} \rangle \\
 & \qquad\qquad\qquad\ \ \textbf{else}\ \ \langle \textit{command} \rangle\ \textbf{end if} \mid \\
 & \textbf{while}\ \langle \textit{statement} \rangle\ \ \textbf{loop}\ \langle \textit{command} \rangle\ \textbf{end loop}
\end{array}
$$

- Variables are represented by single upper-case letters.
- Terms denote integer values, *e.g.* `42`, `2+3`, `X+1`.
- Statements denote boolean values and can be built up using logical constants, *i.e.* *true* and *false*, terms and predicates, *e.g.* `=`, `/=`, `>=`, `<=`, *etc*, as well as the logical operators **and**, **or**, **not**.

## A Little Logic

- $P \wedge Q$: is true if $P$ and $Q$ are true.
- $P \vee Q$: is true if either $P$ or $Q$ are true, otherwise false.
- $P \rightarrow Q$: is true if whenever $P$ is true $Q$ is also true. If $P$ is false, then $P \rightarrow Q$ is true.
- $\neg P$: is true if $P$ is false, and false if $P$ is true.



## Some Examples

| | |
|---|---|
| $\{X = x \wedge Y = y\}$ | $[X \geq 0 \wedge even(X)]$ |
| `T:= X;` | **while** `X /= 0` **loop** |
| `X:= Y;` | `    X := X - 2;` |
| `Y:= T;` | **end loop** |
| $\{X = y \wedge Y = x\}$ | $[X = 0]$ |

- Note that $x$ and $y$ denote variables that are used to name the initial values of $X$ and $Y$ respectively.
- Such variables are called **auxiliary** or **ghost** variables as they do not appear within the code. Lower-case letters will be used to denote auxiliary variables.

## Which are Correct?

1. $\{X = x \wedge Y = y\}$
```
X := Y;
Y := X
```
$\{X = y \wedge Y = x\}$

2. $\{true\}$
```
if even(X) then X := X + 1 end if
```
$\{odd(X)\}$

3. $[true]$
```
while X /= 0 loop
      X := X - 2
end loop
```
$[X = 0]$



## Formalizing Understandings

- In order to reason about the correctness of a program with respect to a pre- and postcondition, *i.e.* a specification, we require some axioms and rules of inference. In other words, a theory which expresses the semantics of our programming language.
- We begin informally with the assignment command. Consider the following specifications:

$$\{X > 4\}\, X := X + 1\{X > 5\}$$

and

$$\{X > 4\}\, X := X + 1\{X > 6\}$$

- Do you judge these to be valid or invalid specification?
- Can you explain (formalize) your reasons?

## An Informal Explanation of :=

- Consider you are told that the following is correct:

$$\{P\}\, X := 4\{Y > X\}$$

What should $P$ be?

- From your understanding of assignment you might guess that $P$ is $y > 4$, *i.e.*

$$\{Y > 4\}\, X := 4\{Y > X\}$$

This is actually the **most general** or **weakest** statement we can make.

- Note that my definition of $P$ can be obtained from $Q$ by **mechanically** replacing all the occurrences of $X$ by 4. By generalizing this observation we obtain an axiom for the assignment command:

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



## The Assignment Axiom

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

- Where $V$ is an arbitrary variable, $E$ is an expression and $P$ is a statement. Note that $P[E/V]$ denotes the result of substituting $E$ for all occurrences of $V$ in $P$.
- Some example instances of the assignment axiom:

$$\{Y = 2\}\, X := 2\{Y = X\}$$
$$\{X + 1 = N + 1\}\, X := X + 1\{X = N + 1\}$$
$$\{A = B\}\, X := A\{X = B\}$$

## The Sequence Rule

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}$$

Given:

$\{X = m \land Y = n\}$
```
R:=X; X:=Y
```
$\{R = m \land X = n\}$

$\{R = m \land X = n\}$
```
Y:=R
```
$\{Y = m \land X = n\}$

we can use the Sequence-rule to get:

$\{X = m \land Y = n\}$
```
R:=X; X:=Y; Y:=R
```
$\{Y = m \land X = n\}$



## Conditional Rules

- **If-Then rule:**

$$\frac{\{P \land S\}\ C\ \{Q\} \qquad P \land \neg S \rightarrow Q}{\{P\}\ \textbf{if}\ S\ \textbf{then}\ C\ \textbf{end if}\ \{Q\}}$$

- **If-Then-Else rule:**

$$\frac{\{P \land S\}\ C_1\ \{Q\} \qquad \{P \land \neg S\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ S\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{end if}\ \{Q\}}$$

## The While Rule

$$\frac{\{P \rightarrow R\} \quad \{R \wedge S\}\ C\ \{R\} \quad \{\neg S \wedge R \rightarrow Q\}}{\{P\}\ \textbf{while}\ \ S\ \ \textbf{loop}\ \{R\}\ C\ \textbf{end loop}\ \{Q\}}$$

- Note that the assertion $R$ appears as both a precondition and a postcondition to the body of the loop, *i.e.* it is *true* before and after each iteration of a loop. Such an assertion is called the **loop invariant**.
- A loop has many invariants. However, the selection of an "appropriate" invariant is crucial to the successful construction of a verification proof.



## Logical Rules

- Precondition strengthening:

$$\frac{\vdash P \rightarrow P' \qquad \vdash \{P'\}\ C\ \{Q\}}{\vdash \{P\}\ C\ \{Q\}}$$

- Postcondition weakening:

$$\frac{\vdash \{P\}\ C\ \{Q'\} \qquad \vdash Q' \rightarrow Q}{\vdash \{P\}\ C\ \{Q\}}$$

## Learning Outcomes

- Understand assertion based program specification.
- Understand the distinction between partial and total correctness.
- Understand how the *meaning* of program constructs can be defined in terms of rules and axioms.



## Recommended Reading

- "High Integrity Software: The SPARK Approach to Safety and Security" Barnes, J. Addison-Wesley 2003 [ **chapters 3 & 11** ]
- "Assigning meanings to programs", Floyd, R.W. Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics 19, 1967.
- "Programming Language Theory and its Implementation", M.J.C.Gordon, Prentice Hall, 1988.
- "An axiomatic basis for computer programming", Hoare, C.A.R. Communications of the ACM, 12, 1969.