Software Verification

Partial and Total Correctness

Ahmed Rezine

IDA, Linköpings Universitet

Spring 2021

Outline

Outline

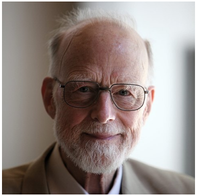
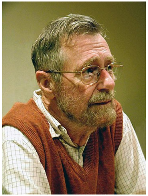
Function Specifications and Correctness

- Contract between the caller and the implementation. **Total Correctness** requires that:
  - if the pre-condition (`0 < x && 0 < y`) holds
  - then the implementation terminates,
  - after termination, the following post-condition holds (`x < sum && y < sum`)
- **Partial Correctness** does not require termination

```
method add(x: int, y: int) returns (sum: int)
   requires 0 < x && 0 < y
   ensures  x < sum && y < sum
{
   sum := x + y;
}
```

Already more than 50 years

- R. W. Floyd. Assigning meanings to programs. 1968.
- C. A. R. Hoare. An axiomatic basis for computer programming. 1969.
- E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. 1975.

Hoare Triples and Partial Correctness

- a Hoare triple $\{P\}$ *stmt* $\{R\}$ consists in:
  - a predicate pre-condition P
  - an program *stmt*,
  - a predicate post-condition R
- intuitively, $\{P\}$ *stmt* $\{R\}$ holds if whenever P holds and *stmt* is executed and terminates (**partial correctness**), then R holds after *stmt* terminates.
- examples:
  - $\{true\}\ x := y\ \{x = y\}$
  - $\{x = 1 \wedge y = 2\}\ x := y\ \{x = 2\}$
  - $\{x \geq 1\}\ y := 2\ \{x = 0 \vee y \leq 10\}$
  - $\{x \geq 1\}\ \text{if}(y = 2)\ \text{then}\ x := 0\ \{x \geq 0\}$
  - $\{false\}\ x := 1\ \{x = 2\}$
  - $\{true\}\ \texttt{abort}\ \{false\}$
  - $\{x = 10\}\ \texttt{skip}\ \{x = 10\}$

Hoare Triples and Partial Correctness

$$\frac{P' \implies P \quad \{P\}\ stmt\ \{Q\} \quad Q \implies Q'}{\{P'\}\ stmt\ \{Q'\}} \qquad \frac{\{P\}\ stmt\ \{Q\} \qquad \{Q\}\ stmt'\ \{R\}}{\{P\}\ stmt; stmt'\ \{R\}}$$

$$\frac{\{P \wedge B\}\ stmt\ \{Q\} \qquad \{P \wedge \neg B\}\ stmt'\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ stmt\ \textbf{else}\ stmt'\ \{Q\}} \qquad \frac{}{\{P[e/x]\}\ x := e\ \{P\}}$$

$$\frac{\{P \wedge B\}\ stmt\ \{P\}}{\{P\}\ (\textbf{while}\ (B)\ \{stmt\})\ \{P \wedge \neg B\}}$$

Weakest precondition

- if $\{P\}$ *stmt* $\{R\}$ and $P' \Rightarrow P$ for any P' s.t. $\{P'\}$ *stmt* $\{R\}$, then P is the **weakest precondition** of R wrt. *stmt*, written $\mathbf{wp}\,(stmt, R)$. It is unique.
- $\mathbf{wp}\,(stmt, R)$ transforms predicate R wrt. *stmt*. It is said to be a **predicate transformer**.
- $\mathbf{wp}\,(x := x + 1, x \geq 1) = (x \geq 0)$. Observe $(x \geq 5)$, $(x = 6)$, $(x \geq 0 \wedge y = 8)$ are all valid preconditions, but they are not weaker than $x \geq 0$.
- Intuitively $\mathbf{wp}\,(stmt, R)$ is the weakest predicate P for which $\{P\}$ *stmt* $\{R\}$ holds

Weakest precondition of assignments

- $\textbf{wp}\,(x := e, R) = R[e/x]$ replaces occurrences of x in R by e.
- examples:
  - $\textbf{wp}\,(x := 3, x = 5) = (x = 5)[x/3] = (3 = 5) = \mathit{false}$
  - $\textbf{wp}\,(x := 3, x \geq 0) = (x \geq 0)[x/3] = (3 \geq 0) = \mathit{true}$
  - $\textbf{wp}\,(x := y + 5, x \geq 0) = (x \geq 0)[x/y + 5] = (y + 5 \geq 0)$
  - $\textbf{wp}\,(x := 5 * y + 2 * z, x + y \geq 0) = (x + y \geq 0)[x/5 * y + 2 * z] = (6 * y + 2 * z \geq 0)$

Weakest precondition of sequences

- Assume a sequence of two instructions $stmt; stmt'$, for example $x := 2 * y; y := x + 3 * y;$
- the weakest precondition is given by:
  $\mathbf{wp}\,(stmt; stmt', R) = \mathbf{wp}\,(stmt, \mathbf{wp}\,(stmt', R))$,

$$
\begin{array}{ll}
 & \mathbf{wp}\,(x = 2 * y; y = x + 3 * y, y > 10) \\
= & \mathbf{wp}\,(x = 2 * y, \mathbf{wp}\,(y = x + 3 * y, y > 10)) \\
= & \mathbf{wp}\,(x = 2 * y, (y > 10)[y/x + 3 * y]) \\
= & \mathbf{wp}\,(x = 2 * y, x + 3 * y > 10) \\
= & (x + 3 * y > 10)[x/2 * y] \\
= & (2 * y + 3 * y > 10) \\
= & y > 2
\end{array}
$$

Weakest precondition of conditionals

- Assume a conditional (if(B) then *stmt* else *stmt′*), for example (if($x > y$) then $z := x$ else $z := y$)
- The weakest precondition is given by:

$$\left(\begin{array}{rl} & \mathbf{wp}\,((\text{if}(B) \text{ then } \mathit{stmt} \text{ else } \mathit{stmt}'), R) \\ = & (B \Rightarrow \mathbf{wp}\,(\mathit{stmt}, R)) \wedge (\neg B \Rightarrow \mathbf{wp}\,(\mathit{stmt}', R)) \end{array}\right)$$

- For example,

$$\begin{array}{rl} & \mathbf{wp}\,((\text{if}(x > y) \text{ then } z := x \text{ else } z := y), z \leq 10) \\ = & (x > y \Rightarrow \mathbf{wp}\,(z := x, z \leq 10)) \wedge (x \leq y \Rightarrow \mathbf{wp}\,(z := y, z \leq 10 \\ = & (x > y \Rightarrow x \leq 10) \wedge (x \leq y \Rightarrow y \leq 10) \end{array}$$

- More general:

$$\mathbf{wp}\left(\left(\begin{array}{llll} \textbf{if} & B_1 & \rightarrow & \mathit{stmt}_1 \\ \square & B_2 & \rightarrow & \mathit{stmt}_2 \\ \textbf{fi} & & & \end{array}\right), R\right)$$

$$=$$

$$(B_1 \vee B_2) \wedge (B_1 \Rightarrow \mathbf{wp}\,(\mathit{stmt}, R)) \wedge (B_2 \Rightarrow \mathbf{wp}\,(\mathit{stmt}', R))$$

Examples: weakest preconditions

Show:

1. $\mathbf{wp}\,(stmt; \texttt{skip}, P) = \mathbf{wp}\,(stmt, P)$
2. $stmt_a; stmt_b$ is equivalent to $stmt_c$ where:

$$\begin{array}{rllll} stmt_a = & \textbf{if} & B_1 & \rightarrow & stmt_1 \\ & \square & B_2 & \rightarrow & stmt_2 \\ & \textbf{fi} & & & \end{array}$$

$$\begin{array}{rllll} stmt_c = & \textbf{if} & B_1 & \rightarrow & stmt_1; stmt_b \\ & \square & B_2 & \rightarrow & stmt_2; stmt_b \\ & \textbf{fi} & & & \end{array}$$

3. find the weakest P s.t. $\{P\}\ stmt\ \{x = 1\}$ where:

$$\begin{array}{rllll} stmt = & \textbf{if} & T & \rightarrow & x := 1 \\ & \square & T & \rightarrow & x := -1 \\ & \textbf{fi} & & & \end{array}$$

Hoare Triples for Loops, Partial Correctness

- In order to establish $\{P\}$ $(\texttt{while}(B)\texttt{do}\{stmt\})$ $\{R\}$, you will need to find an invariant *Inv* such that:
  - $P \Rightarrow Inv$
  - $\{Inv \wedge B\}$ $stmt$ $\{Inv\}$
  - $(Inv \wedge \neg B) \Rightarrow R$
- For example $\{i = j = 0\}$ $(\texttt{while}(i < 10)\texttt{do}\{i := i{+}1; j := j{+}1\})$ $\{j = 10\}$, we need to find *Inv* such that:
  - $(i = j = 0) \Rightarrow Inv$
  - $\{Inv \wedge (i < 10)\}$ $i := i + 1; j := j + 1$ $\{Inv\}$
  - $(Inv \wedge i \geq 10) \Rightarrow j = 10$

Examples: Invariants

Show:

$$
\begin{array}{l}
\{Q : a \geq 0 \wedge b \geq 0\} \\
z := 0; x := a; y := b \\
\{P : (x \geq 0) \wedge (z + x * y = a * b)\} \\
\quad \textbf{do} \quad x \geq 1 \quad \rightarrow \quad \begin{array}{llll}
\textbf{if} & odd(x) & \rightarrow & z := z + y \\
\square & even(x) & \rightarrow & \texttt{skip} \\
\textbf{fi}; & & & \\
x := x/2; & & & \\
y := 2 * y & & &
\end{array} \\
\quad \textbf{od} \\
\{R : z = a * b\}
\end{array}
$$

Hoare Triples for Loops, Total Correctness

- $\{P\}$ $(\texttt{while}(B)\texttt{do}\{stmt\})$ $\{R\}$
- Partial correctness: if we start from P and $(\texttt{while}(B)\texttt{do}\{stmt\})$ terminates, then R terminates.
  - $P \Rightarrow Inv$
  - $\{Inv \wedge B\}$ $stmt$ $\{Inv\}$
  - $(Inv \wedge \neg B) \Rightarrow R$
- Total correctness: the loop does terminate: find a **variant function** v such that:
  - $(Inv \wedge B) \Rightarrow (v > 0)$
  - $\{Inv \wedge B \wedge v = v_0\}$ $stmt$ $\{v < v_0\}$
- For example $(\texttt{while}(i < 10)\texttt{do}\{i := i + 1; j := j + 1\})$ can be shown to terminate with $v = (10 - i)$ and $Inv = (i \leq 10)$

Examples: Termination

Show:

$$
\begin{array}{l}
\{Q : a \geq 0 \wedge b \geq 0\} \\
z := 0; x := a; y := b \\
\{invP : (x \geq 0) \wedge (z + x * y = a * b)\} \\
\{boundt : x\} \\
\begin{array}{lllllll}
\textbf{do} & x \geq 1 & \rightarrow & \textbf{if} & odd(x) & \rightarrow & z := z + y \\
 & & & \Box & even(x) & \rightarrow & \texttt{skip} \\
 & & & \textbf{fi}; & & & \\
 & & & x := x/2; & & & \\
 & & & y := 2 * y & & & \\
\textbf{od} & & & & & &
\end{array} \\
\{R : z = a * b\}
\end{array}
$$

Outline

Further readings

A. R. Bradley and Z. Manna.
(chap 5-6) The calculus of computation: decision procedures with applications to verification.
Springer Science & Business Media, 2007.

C. A. R. Hoare.
An axiomatic basis for computer programming.
Communications of the ACM, 12(10):576–580, 1969.

K. R. M. Leino.
Dafny: An automatic program verifier for functional correctness.
In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 348–370. Springer, 2010.