High Integrity Computing (F23PS2)

High Integrity Software Development (F29HD2)

Verification Condition Generation

Andrew Ireland

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Overview

- A goal oriented approach to assertion based program proof.
- Verification condition generation.

Goal Oriented Program Proof

1. An annotated program C is developed where the annotations are assertions (logical formulae) that express desired conditions at various intermediate points.
2. A set of logical formulae, or **verification conditions** (VCs) is generated from the annotated program.
3. The verification conditions are proved by a theorem proving system.

Annotations

In order for VC generation to be automatic requires that the program is properly annotated, *i.e.* the program contains enough assertions. A program is properly annotated if there exists an assertion:

- Before each program command C_i (where $i > 1$) in a sequence $C_1; C_2; \ldots; C_n$ which is **not** an assignment command.
- After the reserved word **loop** within the **while** loop construct, *i.e.* the loop invariant.

Note that the generation of loop invariants is undecidable in general and remains a major bottle-neck in terms of fully automating the process of verifying programs.

Quotient-Remainder Example

$\{true\}$

```
R:= X;
Q:= 0;
```

$\{R = X \wedge Q = 0\}$

while `Y<=R` **loop**

  $\{X = R + (Y * Q)\}$ $\longleftarrow$ **loop invariant**

```
    R:= R - Y;
    Q:= Q + 1;
```

end loop;

$\{X = R + (Y * Q) \wedge R < Y\}$

A **loop invariant** is an assertion that is true before and after each execution of the loop body.

Quotient-Remainder VCs

$$\begin{array}{rcl} true & \rightarrow & (X = X \wedge 0 = 0) \\ (R = X \wedge Q = 0) & \rightarrow & (X = R + (Y * Q)) \\ (X = R + (Y * Q)) \wedge \neg(Y \leq R) & \rightarrow & (X = R + (Y * Q) \wedge R < Y) \\ (X = R + (Y * Q)) \wedge Y \leq R & \rightarrow & (X = (R - Y) + (Y * (Q + 1))) \end{array}$$

Note that VCs are logical formulae — they contain no references to programming constructs.

Generating VCs for Assignment Commands

The VC generated by

$$\{P\}V := E\{Q\}$$

is

$$P \to Q[E/V]$$

where $Q[E/V]$ denotes the expression constructed by replacing all occurrences of V by E within Q.

Example: The VC generated by

$$\{X = 0\}X := X + 1\{X = 1\}$$

is

$$(X = 0) \to (X + 1) = 1$$

Generating VCs for if-then Conditionals

The VCs generated by

$$\{P\}\textbf{if}\ S\ \textbf{then}\ C\,\textbf{end if}\{Q\}$$

are

1. $(P \wedge \neg S) \to Q$
2. the VCs generated from:

$$\{P \wedge S\}C\{Q\}$$

Generating VCs for if-then-else Conditionals

The VCs generated by

$$\{P\}\textbf{if}\ S\ \textbf{then}\ C_1\ \textbf{else}\ C_2\,\textbf{end if}\{Q\}$$

are

1. the VCs generated from:

$$\{P \wedge S\}C_1\{Q\}$$

2. the VCs generated from:

$$\{P \wedge \neg S\}C_2\{Q\}$$

Generating VCs for Sequences

1. The VCs generated by

$$\{P\}C_1; C_2; \ldots; C_{n-1}; \{R\}C_n\{Q\}$$

(where C_n is **not** an assignment) are:

(a) the VCs generated from:

$$\{P\}C_1; C_2; \ldots; C_{n-1}; \{R\}$$

(b) the VCs generated from:

$$\{R\}C_n\{Q\}$$

2. The VCs generated by

$$\{P\}C_1; C_2; \ldots; C_{n-1}; V := E\{Q\}$$

are the VCs generated from:

$$\{P\}C_1; C_2; \ldots; C_{n-1}; \{Q[E/V]\}$$

Generating VCs for while-loop Commands

The VCs generated by

$$\{P\}\textbf{while } S \textbf{ loop } \{R\}\, C\, \textbf{end loop}\{Q\}$$

are

1. $P \rightarrow R$
2. $R \wedge \neg S \rightarrow Q$
3. the VCs generated from:

$$\{R \wedge S\}C\{R\}$$

Quotient-Remainder Revisited

(1) $\{true\}$

```
R:= X;
Q:= 0;
```

$\{R = X \wedge Q = 0\}$

while `Y<=R` **loop**

  $\{X = R + (Y * Q)\}$

```
    R:= R - Y;
    Q:= Q + 1;
```

end loop;

$\{X = R + (Y * Q) \wedge R < Y\}$

Apply sequence generation to (1) giving ...

Quotient-Remainder Revisited

(2) $\{true\}$

```
R:= X;
Q:= 0;
```

$\{R = X \land Q = 0\}$

(3) $\{R = X \land Q = 0\}$

while `Y<=R` **loop**

$\{X = R + (Y * Q)\}$

```
R:= R - Y;
Q:= Q + 1;
```

end loop;

$\{X = R + (Y * Q) \land R < Y\}$

Quotient-Remainder Revisited

(2) $\{true\}$

```
R:= X;
Q:= 0;
```

$\{R = X \land Q = 0\}$

Apply sequence generation to (2):

(4) $\{true\}$

```
R:= X;
```

$\{R = X \land 0 = 0\}$

Apply assignment generation to (4) giving VC1:

$$true \rightarrow (X = X \land 0 = 0)$$

Quotient-Remainder Revisited

```
(3)        {R = X ∧ Q = 0}
           while Y<=R  loop
                 {X = R + (Y * Q)}
                 R:= R - Y;
                 Q:= Q + 1;
           end loop;
           {X = R + (Y * Q) ∧ R < Y}
```

Apply **while** generation to (3) giving rise to VC2 and VC3:

$$(R = X \wedge Q = 0) \quad \rightarrow \quad (X = R + (Y * Q))$$

$$(X = R + (Y * Q)) \wedge \neg(Y \leq R) \quad \rightarrow \quad (X = R + (Y * Q) \wedge R < Y)$$

and the VCs derived from (5) ...

Quotient-Remainder Revisited

```
(5)        {X = R + (Y * Q) ∧ Y ≤ R}
           R:= R - Y;
           Q:= Q + 1;
           {X = R + (Y * Q)}
```

Apply sequence generation to (5):

```
(6)        {X = R + (Y * Q) ∧ Y ≤ R}
           R:= R - Y;
           {X = R + (Y * (Q + 1))}
```

Apply assignment generation to (6) giving rise to VC4:

$$(X = R + (Y * Q)) \wedge Y \leq R) \rightarrow (X = (R - Y) + (Y * (Q + 1)))$$

[**next lecture addresses how-to-prove VCs**]

Learning Outcomes

- Understand the relationship between verification conditions and program proof.
- Be able to derive verification conditions for a simple imperative programming language.

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.