Hoare Logic and Verification Condition Generation

Continued

David Pereira José Proença Eduardo Tovar

FVOCA 2021/2022

Formal Verification of Critical Applications

CISTER – ISEP
Porto, Portugal

`https://cister-labs.github.io/fvoca2122`

The Simple Imperative Language

With Assertions and Specifications

Hoare Triples - review

Core concept

The atomic concept that we will be using in order to reason about the partial correctness of programs is that known as *Hoare Triple*. It is a specification of the form

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

that reads as follows: for all states satisfying the *precondition* P, if the program C executes and terminates in a state satisfying the *postcondition* Q, then the triple is valid.

A language for Hoare Triples

The syntax of the language that we will be using is an extension of the basic Imperative Language with logical assertions, plus the concept of **Hoare Triple**.

$$
\begin{array}{rcll}
x & \in & \text{Identifiers} & \\
n & \in & \mathbb{Z} & \\
B & ::= & \mathsf{true} \mid \mathsf{false} \mid B \,\&\&\, B \mid B \,||\, B \mid\, !B \mid E < E \mid E = E & \text{(boolean-expr)} \\
E & ::= & n \mid x \mid E + E \mid E * E \mid E - E & \text{(int-expr)} \\
C & ::= & \mathsf{skip} \mid C; C \mid x := E \mid \mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ C \mid \mathsf{while}\ B\ \mathsf{do}\ C & \text{(command)} \\
A & ::= & \mathsf{true} \mid \mathsf{false} \mid E < E \mid E = E \mid A \wedge A \mid A \vee A \mid \neg A \mid A \rightarrow A & \\
 & & \mid \forall x.A \mid \exists x.A & \text{(assertions)} \\
S & ::= & \{A\}\ C\ \{A\} & \text{(specification/Hoare Triple)}
\end{array}
$$

Hoare Logic

Proof Rules and Verification Conditions Generation

On Verification Conditions Generation

From proof trees to algorithms

We have stated that a proof tree must be constructed to perform verification using Hoare Logic. When we close all the trees via the application of one of the rules that represents an axiom (e.g., skip rule or assignment rule), then we can state that a valid proof is obtained. Still, it is a manual process, which we would really like to avoid.

Verification Conditions Generation

The process of constructing a proof tree can, fortunately, be replaced by an algorithm. This algorithm, called **Verification Conditions Generator**, or **VCGen** for short, applies a particular strategy for producing verification conditions that correspond to the side conditions of a particular derivation

Verification Conditions Generation

How do these VCGen work?

- A VCGen algorithm takes as input a Hoare Triple $\{P\}\ C\ \{Q\}$ and returns a set of first-order logic proof obligations.
- The proof obligations represent side conditions of the form $F_1 \rightarrow F_2$

Algorithm for weakest precondition generation

$$\begin{aligned}
\mathit{wprec}(\mathsf{skip}, Q) &= Q \\
\mathit{wprec}(x := E, Q) &= Q[x \mapsto E] \\
\mathit{wprec}(C_1; C_2, Q) &= \mathit{wprec}(C_1, \mathit{wprec}(C_2, Q)) \\
\mathit{wprec}(\mathsf{if}\ B\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, Q) &= (B \rightarrow \mathit{wprec}(C_1, Q)) \wedge (\neg B \rightarrow \mathit{wprec}(C_2, Q)) \\
\mathit{wprec}(\mathsf{while}\ B\ \mathsf{do}\ \{I\}\ C, Q) &= I
\end{aligned}$$

Algorithm for generating Verification Conditions

$$VC(\{P\}\,\text{skip}\{Q\}) = \{P \to Q\}$$

$$VC(\{P\}\,x := E\,\{Q\}) = \{P \to Q[x \mapsto E]\}$$

$$VC(\{P\}\,C_1; C_2\,\{Q\}) =$$
$$VC(\{P\}\,C_1\,\{wprec(C_2, Q)\}) \cup VC(\{wprec(C_2, Q)\}\,C_2\,\{Q\})$$

$$VC(\{P\}\,\text{if}\,B\,\text{then}\,C_1\,\text{else}\,C_2\,\{Q\}) =$$
$$VC(\{P \wedge B\}\,C_1\{Q\}) \cup VC(\{P \wedge \neg B\}\,C_2\,\{Q\})$$

$$VC(\{P\}\,\text{while}\,B\,\text{do}\,\{I\}\,C\,\{Q\}) =$$
$$\{P \to I, I \wedge \neg B \to Q\} \cup VC(\{P \wedge \neg B\}\,C\,\{Q\})$$

From the Proof Rules into the Weakest Precondition & VC algorithms

From Program-Proof Rules to algorithms

Proof rule for skip

The skip command does not change the state of the program. Hence, the precondition must imply the postcondition.

$$\frac{}{\{P\}\ \mathsf{skip}\ \{Q\}}\ \text{if } P \to Q$$

WPrec and VC rules

From the algorithmic point of view we have:

$$\begin{aligned} \mathit{wprec}(\mathsf{skip}, Q) &= Q \\ VC(\{P\}\,\mathsf{skip}\{Q\}) &= \{P \to Q\} \end{aligned}$$

From Program-Proof Rules to algorithms

The case of the assignment command

The assignment rule states a change in a value assigned to a variable. Hence, the precondition P must imply that substitution.

$$\frac{}{\{P\}\ x := E\ \{Q\}}\ \text{if } P \to Q[x \mapsto E]$$

WPrec and VC rules

From the algorithmic point of view we have:

$$\begin{aligned} wprec(x := E, Q) &= Q[x \mapsto E] \\ VC(\{P\}\, x := E\, \{Q\}) &= \{P \to Q[x \mapsto E]\} \end{aligned}$$

The Program-Proof System Rules

The case fo while loops

For sequences of commands, the goal is to find the intermediate condition R between the two commands.

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

WPrec and VC rules

From the algorithmic point of view we have:

$$\begin{array}{rcl} wprec(C_1; C_2, Q) & = & wprec(C_1, wprec(C_2, Q)) \\ VC(\{P\}\, C_1; C_2\, \{Q\}) & = & VC(\{P\}\, C_1\, \{wprec(C_2, Q)\}) \cup VC(\{wprec(C_2, Q)\}\, C_2\, \{Q\}) \end{array}$$

The Program-Proof System Rules

The case of conditionals

In the case of conditionals, one must prove that independently of the value of the Boolean B that serves as a guarda for the conditional, the postcondition holds.

$$\frac{\{B \wedge P\}\, C_1\, \{Q\} \qquad \{\neg B \wedge P\}\, C_2\, \{Q\}}{\{P\}\, \textsf{if}\, B\, \textsf{then}\, C_1\, \textsf{else}\, C_2\, \{Q\}}$$

WPrec and VC rules

From the algorithmic point of view we have:

$$\begin{aligned} wprec(\textsf{if}\, B\, \textsf{then}\, C_1\, \textsf{else}\, C_2, Q) &= (B \rightarrow wprec(C_1, Q)) \wedge (\neg B \rightarrow wprec(C_2, Q)) \\ VC(\{P\}\, \textsf{if}\, B\, \textsf{then}\, C_1\, \textsf{else}\, C_2\, \{Q\}) &= VC(\{P \wedge B\}\, C_1 \{Q\}) \cup VC(\{P \wedge \neg B\}\, C_2\, \{Q\}) \end{aligned}$$

Extending the annotation to while loops

The case fo while loops

The new rule for while loops now considers the invariant as an annotation:

$$\frac{\{B \wedge I\}\, C\, \{I\}}{\{P\}\, \textsf{while}\ B\ \textsf{do}\ \{I\}\ C\ \{Q\}}\ \text{if } P \rightarrow I \text{ and } I \wedge \neg B \rightarrow Q$$

WPrec and VC rules

From the algorithmic point of view we have:

$$\begin{aligned} wprec(\textsf{while}\, B\, \textsf{do}\, \{I\}\, C, Q) &= I \\ VC(\{P\}\, \textsf{while}\, B\, \textsf{do}\, \{I\}\, C\, \{Q\}) &= \{P \rightarrow I, I \wedge \neg B \rightarrow Q\} \cup VC(\{P \wedge \neg B\}\, C\, \{Q\}) \end{aligned}$$

Improving the VCGen

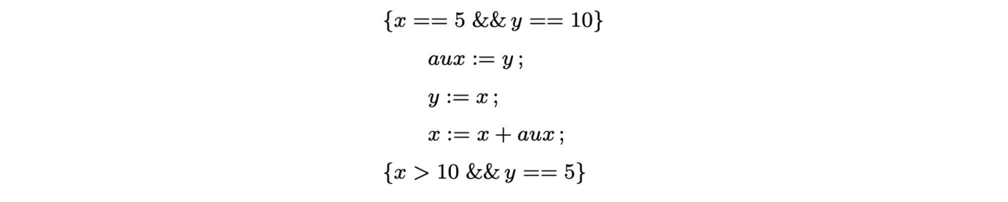
Improved Verification Condition Generation

$$
\begin{aligned}
VC(\textsf{skip}, Q) &= \emptyset \\
VC(x := e, Q) &= \emptyset \\
VC(C_1; C_2, Q) &= VC(C_1, wprec(C_2, Q)) \cup VC(C_2, Q) \\
VC(\textsf{if}\ B\ \textsf{then}\ C_1\ \textsf{else}\ C_2, Q) &= VC(C_1, Q) \cup VC(C_2, Q) \\
VC(\textsf{while}\ B\ \textsf{do}\ \{I\}\ C, Q) &= \{(I \wedge B) \rightarrow wprec(C, I)\} \cup VC(C, I) \\
&\quad \{(I \wedge \neg B) \rightarrow Q\}
\end{aligned}
$$

$$
VCG(\{P\}\ C\ \{Q\}) = \{P \rightarrow wprec(C, Q)\} \cup VC(C, Q)
$$

Propagation of annotations in code

Consider the following code:



$$\{x == 5 \;\&\&\; y == 10\}$$
$$aux := y\,;$$
$$y := x\,;$$
$$x := x + aux\,;$$
$$\{x > 10 \;\&\&\; y == 5\}$$

Let us run the first *wprec* function on this. The actual call to the function will be

$$\mathit{wprec}(\mathit{aux} := y; y := x; x := x + \mathit{aux}, \{x > 10 \wedge y == 5\})$$

Running example - usage of wprec

$$
\begin{aligned}
\mathbf{wprec}(aux := y; y := x; x := x + aux, \{x > 10 \land y == 5\}) &= \\
\mathbf{wprec}(aux := y, \mathbf{wprec}(y := x; x := x + aux, \{x > 10 \land y == 5\})) &= \\
\mathbf{wprec}(aux := y, \mathbf{wprec}(y := x, \mathbf{wprec}(x := x + aux, \{x > 10 \land y == 5\}))) &= \\
\mathbf{wprec}(aux := y, \mathbf{wprec}(y := x, \{x > 10 \land y == 5\}[x \mapsto x + aux])) &= \\
\mathbf{wprec}(aux := y, \mathbf{wprec}(y := x, \{x + aux > 10 \land y == 5\})) &= \\
\mathbf{wprec}(aux := y, \{x + aux > 10 \land y == 5\}[y \mapsto x]) &= \\
\mathbf{wprec}(aux := y, \{x + aux > 10 \land x == 5\}) &= \\
\{x + aux > 10 \land x == 5\}[aux \mapsto y] &= \\
\{x + y > 10 \land x == 5\} &
\end{aligned}
$$

Running example - usage of VC

$$\textbf{VC}(\{x = 5 \wedge y = 10\}\ aux := y; y := x; x := x + aux, \{x > 10 \wedge y == 5\}) =$$

$$(1)\,\textbf{VC}(\{x = 5 \wedge y = 10\}\ aux := y\ \{\textbf{wprec}(y := x; x := x + aux, \{x > 10 \wedge y == 5\})\})$$

$$\cup$$

$$(2)\,\textbf{VC}(\{\textbf{wprec}(y := x; x := x + aux, \{x > 10 \wedge y == 5\})\}$$

$$y := x; x := x + aux\ \{x > 10 \wedge y == 5\}) =$$

Running example - usage of VC (1)

$$(1)\ \mathbf{VC}(\{x = 5 \land y = 10\}\ \mathit{aux} := y\ \{\mathbf{wprec}(y := x; x := x + \mathit{aux}, \{x > 10 \land y = 5\})\})$$

$$=$$

$$\mathbf{VC}(\{x = 5 \land y = 10\}\ \mathit{aux} := y\ \{\mathbf{wprec}(y := x, \mathbf{wprec}(x := x + \mathit{aux}, \{x > 10 \land y = 5\}))\})$$

$$=$$

$$\mathbf{VC}(\{x = 5 \land y = 10\}\ \mathit{aux} := y\ \{\mathbf{wprec}(y := x, \{x > 10 \land y = 5\}[x \mapsto x + \mathit{aux}])\})$$

$$=$$

$$\mathbf{VC}(\{x = 5 \land y = 10\}\ \mathit{aux} := y\ \{x > 10 \land y = 5\}[x \mapsto x + \mathit{aux}][y \mapsto x])\})$$

$$=$$

$$\mathbf{VC}(\{x = 5 \land y = 10\}\ \mathit{aux} := y\ \{x + \mathit{aux} > 10 \land x = 5\})$$

$$=$$

$$\{(x = 5 \land y = 10) \rightarrow (x + \mathit{aux} > 10 \land x = 5)[\mathit{aux} \rightarrow y]\}$$

$$=$$

$$\{(x = 5 \land y = 10) \rightarrow (x + y > 10 \land x = 5)\}$$

Running example - usage of VC (2)

$$(2)\ \mathbf{VC}(\{\mathbf{wprec}(y := x; x := x + \mathit{aux}, \{x > 10 \land y = 5\})\}\ y := x; x := x + \mathit{aux}\ \{x > 10 \land y = 5\})$$

$$=$$

$$\mathbf{VC}(\{\mathbf{wprec}(y := x, \mathbf{wprec}(x := x + \mathit{aux}, \{x > 10 \land y = 5\}))\}\ y := x; x := x + \mathit{aux}\ \{x > 10 \land y = 5\})$$

$$=$$

$$\mathbf{VC}(\{\mathbf{wprec}(y := x, \{x > 10 \land y == 5\})[x \mapsto x + \mathit{aux}]\}\ y := x; x := x + \mathit{aux}\ \{x > 10 \land y = 5\})$$

$$=$$

$$\mathbf{VC}(\{x > 10 \land y = 5\}[x \mapsto x + \mathit{aux}][y \mapsto x]\ y := x; x := x + \mathit{aux}\ \{x > 10 \land y == 5\})$$

$$=$$

$$\mathbf{VC}(\{x + \mathit{aux} > 10 \land y = 5\}[y \mapsto x]\ y := x; x := x + \mathit{aux}\ \{x > 10 \land y = 5\})$$

$$=$$

$$\mathbf{VC}(\{x + \mathit{aux} > 10 \land x = 5\}\ y := x; x := x + \mathit{aux}\ \{x > 10 \land y = 5\})$$

$$=$$

Running example - usage of VC (2)

$$\mathbf{VC}(\{x + aux > 10 \wedge x = 5\}\ y := x\ \{\mathbf{wprec}(x := x + aux, \{x > 10 \wedge y = 5\})\})$$

$$\cup$$

$$\mathbf{VC}(\{\mathbf{wprec}(x := x + aux, \{x > 10 \wedge y = 5\})\}\ x := x + aux\ \{x > 10 \wedge y = 5\}) =$$

$$\mathbf{VC}(\{x + aux > 10 \wedge x = 5\}\ y := x\{x > 10 \wedge y = 5\}[x \rightarrow x + aux])\})$$

$$\cup$$

$$\mathbf{VC}(\{x > 10 \wedge y = 5\}[x \rightarrow x + aux])\}\ x := x + aux\ \{x > 10 \wedge y = 5\}) =$$

$$\mathbf{VC}(\{x + aux > 10 \wedge x = 5\}\ y := x\ \{x + aux > 10 \wedge y = 5\})$$

$$\cup$$

$$\mathbf{VC}(\{x + aux > 10 \wedge y = 5\}\ x := x + aux\ \{x > 10 \wedge y = 5\}) =$$

Running example - usage of VC (2)

$$\{x + aux > 10 \land y = 5 \rightarrow (x + aux > 10 \land y = 5)[y \mapsto x]\}$$

$$\cup$$

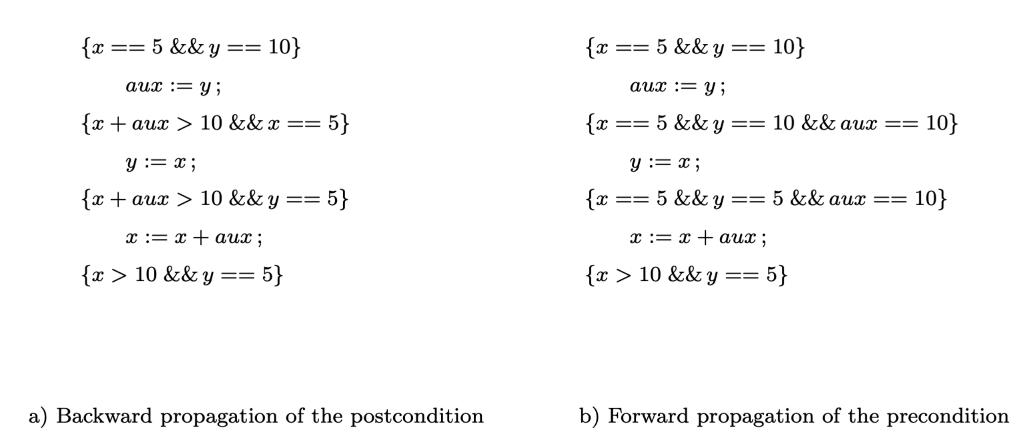
$$\{x + aux > 10 \land y = 5 \rightarrow (x > 10 \land y = 5)[x \mapsto x + aux]\} =$$

$$\{x + aux > 10 \land y = 5 \rightarrow (x + aux > 10 \land x = 5)\}$$

$$\cup$$

$$\{x + aux > 10 \land y = 5 \rightarrow (x + aux > 10 \land y = 5)\} =$$

Results from propagation



$\{x == 5 \;\&\&\; y == 10\}$

$\quad aux := y\,;$

$\{x + aux > 10 \;\&\&\; x == 5\}$

$\quad y := x\,;$

$\{x + aux > 10 \;\&\&\; y == 5\}$

$\quad x := x + aux\,;$

$\{x > 10 \;\&\&\; y == 5\}$

a) Backward propagation of the postcondition

$\{x == 5 \;\&\&\; y == 10\}$

$\quad aux := y\,;$

$\{x == 5 \;\&\&\; y == 10 \;\&\&\; aux == 10\}$

$\quad y := x\,;$

$\{x == 5 \;\&\&\; y == 5 \;\&\&\; aux == 10\}$

$\quad x := x + aux\,;$

$\{x > 10 \;\&\&\; y == 5\}$

b) Forward propagation of the precondition

Weakest Preconditions

Weakest Preconditions

A new look at back propagation of annotations

- we have seen that back propagating annotations from postconditions works
- it can be realised as an algorithm
- such algorithm generates the so-called **weakest preconditions** of an annotated code starting from its postcondition, that is, the weakes conditions that ensure that a program C satisfies the postcondition Q if it terminates, that is

$$\{\mathrm{wprec}(C, Q)\}\, C\, \{Q\}$$

- if we are able to prove that a precondition $p \to \mathrm{wprec}(C, Q)$ holds, then we can use this weakest precondition generation algorithm to help building the proof tree!