Proofs of Correctness: An Introduction to Axiomatic Verification

CEN 5035
Software Engineering

Prepared by
Stephen M. Thebaut, Ph.D.
University of Florida
Important info for students:

- “Intro to Proofs of Correctness” is an elementary introduction to the verification material covered in CEN 4072/6070, Software Testing & Verification.
- Therefore, if you have already taken CEN 4072/6070, you will NOT be tested on this material in Exam 2.
- Instead, you will be tested on Sommerville Chaps 16 and 25 (“Software reuse” and “Configuration management”), which will NOT be covered in class.
Outline

• Introduction
• Weak correctness predicate
• Assignment statements
• Sequencing
• Selection statements
• Iteration
Introduction

• *What is Axiomatic Verification?*

A formal method of reasoning about the **functional** correctness of a **structured, sequential program** by tracing its state changes from an initial (i.e., pre-) condition to a final (i.e., post-) condition according to a set of self-evident rules (i.e., **axioms**).
What is its primary goal?

To provide a means for “proving” (or “disproving”) the functional correctness of a sequential program with respect to its (formal) specification.
• What are the benefits of studying axiomatic verification?
  - Understanding its limitations.
  - Deeper insights into programming and program structures.
  - Criteria for judging both programs and programming languages.
  - The ability to formally verify small (or parts of large) sequential programs.
Bottom line: even if you never attempt to “prove” a program correct outside this course, the study of formal verification should change the way you write and read programs.
Weak Correctness Predicate

- To prove that program S is (weakly) correct with respect to pre-condition P and post-condition Q, it is sufficient to show: \{P\} S \{Q\}.
- Interpretation of \{P\} S \{Q\}: "if the input (initial state) satisfies pre-condition P and (if) program S executes and terminates, then the output (final state) must satisfy post-condition Q."
Note that \( \{P\} S \{Q\} \) is really just a “double conditional” of the form:

\[(A \land B) \Rightarrow C\]

where \( A \) is “\( P \) holds before executing \( S \)”, \( B \) is “\( S \) terminates”, and \( C \) is “\( Q \) holds after executing \( S \)”. 

Therefore, what is the one and only case (in terms of the values of \( A, B, \) and \( C \)) for which \( \{P\} S \{Q\} \) is \textit{false}?
Weak Correctness Predicate (cont’d)

• Thus, \{P\} S \{Q\} is true unless Q could be false if S terminates, given that P held before S executes.

• What are the truth values of the following assertions?

(1) \{x=1\} y := x+1 \{y>0\}
Weak Correctness Predicate (cont’d)

• Thus, \{P\} S \{Q\} is *true* unless Q *could be false* if S terminates, given that P held before S executes.

• What are the truth values of the following assertions?

(2) \{x>0\} x := x-1 \{x>0\}
Weak Correctness Predicate (cont’d)

- Thus, \{P\} S \{Q\} is true unless Q could be false if S terminates, given that P held before S executes.

- What are the truth values of the following assertions?

  (3) \{1=2\} k := 5 \{k<0\}
Weak Correctness Predicate (cont’d)

• Thus, $\{P\} \ S \ \{Q\}$ is true unless $Q$ could be false if $S$ terminates, given that $P$ held before $S$ executes.

• What are the truth values of the following assertions?

(4) $\{\text{true}\}$ while $x \not< 5$ do $x := x - 1$ $\{x=5\}$
    
    (Hint: When will $S$ terminate?)
Weak Correctness Predicate (cont’d)

- We now consider techniques for proving that such assertions hold for structured programs comprised of assignment statements, if-then (-else) statements, and while loops.

  (Why these particular constructs?)
Reasoning about Assignment Statements

• For each of the following pre-conditions, P, and assignment statements, S, identify a "strong" post-condition, Q, such that \{P\} S \{Q\} would hold.

• A "strong" post-condition captures all after-execution state information of interest.

• We won’t bother with propositions such as X=X' (“the final value of X is the same as the initial value of X”) for the time being.
Reasoning about Assignment Statements (cont’d)

<table>
<thead>
<tr>
<th>{P}</th>
<th>S</th>
<th>{Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{J=6}</td>
<td>K := 3</td>
<td></td>
</tr>
<tr>
<td>{J=6}</td>
<td>J := J+2</td>
<td></td>
</tr>
<tr>
<td>{A&lt;B}</td>
<td>Min := A</td>
<td></td>
</tr>
<tr>
<td>{X&lt;0}</td>
<td>Y := -X</td>
<td></td>
</tr>
</tbody>
</table>
Reasoning about Assignment Statements (cont’d)

• For each of the following post-conditions, Q, and assignment statements, S, identify a “weak” pre-condition, P, such that \{P\} S \{Q\} would hold.

(A “weak” pre-condition reflects only what needs to be true before.)
Reasoning about Assignment Statements (cont’d)

\[\{P\} \quad S \quad \{Q\}\]

\[
\begin{align*}
I & := 4 & \{J=7 \land I=4\} \\
I & := 4 & \{I=4\} \\
I & := 4 & \{I=17\} \\
Y & := X+3 & \{Y=10\}
\end{align*}
\]
Reasoning about Sequencing

- In general: if you know \{P\} S_1 \{R\} and you know \{R\} S_2 \{Q\} then you know \{P\} S_1 ; S_2 \{Q\}.

  (So, to prove \{P\} S_1 ; S_2 \{Q\}, find \{R\}.)
Example 1

- Prove the assertion:

\[
\{A=5\} \quad B := A + 2; \quad C := B - A; \quad D := A - C \quad \{A=5 \land D=3\}
\]
Reasoning about If_then_else Statements

• Consider the assertion:
  \{P\} if b then S₁ else S₂ \{Q\}

• What are the necessary conditions for this assertion to hold?
Necessary Conditions: If_then_else
Reasoning about If_then Statements

- Consider the assertion:

  \{P\} \text{ if } b \text{ then } S \{Q\}

- What are the necessary conditions for this assertion to hold?

- What are the \textit{necessary} conditions for this assertion to hold?
Necessary Conditions: If_then
Example 2

- Prove the assertion:

\{Z=B\} \text{ if } A>B \text{ then } Z := A \ \{Z=\text{Max}(A,B)\}
Proof Rules

- Before proceeding to while loops, let’s capture our previous reasoning about sequencing and selection statements in appropriate *rules of inference* (ROI).

ROI for Sequencing:

\[
\begin{align*}
\{P\} S_1 & \{R\}, \{R\} S_2 \{Q\} \\
\{P\} S_1; S_2 \{Q\}
\end{align*}
\]
Proof Rules (cont’d)

ROI for \texttt{if\_then\_else} statement:

\[
\begin{align*}
\{P \land b \} & \iff S_1 \{Q\}, \{P \land \neg b\} \iff S_2 \{Q\} \\
\{P\} & \text{if } b \text{ then } S_1 \text{ else } S_2 \{Q\}
\end{align*}
\]

ROI for \texttt{if\_then} statement:

\[
\begin{align*}
\{P \land b \} & \iff S \{Q\}, (P \land \neg b) \Rightarrow Q \\
\{P\} & \text{if } b \text{ then } S \{Q\}
\end{align*}
\]
Reasoning about Iteration

- Consider the assertion: \( \{P\} \text{ while } b \text{ do } S \{Q\} \)

- What are the necessary conditions for this assertion to hold?

- What are the necessary conditions for this assertion to hold?
Consider a Loop “Invariant” - I

Suppose I holds initially...

is preserved by S...

and implies Q when and if the loop finally terminates...

then the assertion would hold!
Sufficient Conditions: while_do

• Thus, a ROI for the `while_do` statement is:

\[
P \Rightarrow I, \ (I \land b) \ S \ {I}, \ (I \land \neg b) \Rightarrow Q \]

\[
{\{P\}} \ \text{while} \ b \ \text{do} \ S \ {\{Q\}}
\]

where the three antecedents are sometimes given the names *initialization*, *preservation*, and *finalization*, respectively.
Example 3

Use the invariant I: Z=XJ to prove:

\{\text{true}\}  \quad \text{Initialization: } P \Rightarrow I

Z := X
J := 1
while J<>Y do
  Z := Z+X
  J := J+1
end_while

\{Z=XY\}  \quad \text{Preservation: } \{I \land b\} \ S \ \{I\}

Finalization: (I \land \neg b) \Rightarrow Q
Example 3

Use the invariant $I: Z=XJ$ to prove:

$\{\text{true}\}$

\[
\begin{align*}
Z & := X \\
J & := 1 \\
\text{while } J<>Y \text{ do} \\
Z & := Z+X \\
J & := J+1 \\
\text{end\_while}
\end{align*}
\]

$\{Z=XY\}$

**Initialization:** $P \Rightarrow I$

What is “$P$”? 

$(Z=X \land J=1)$

Does $(Z=X \land J=1) \Rightarrow Z=XJ$? 

Yep!
Example 3

Use the invariant $I: Z=XJ$ to prove:

$$\begin{align*}
\{\text{true}\} & \quad \text{Initialization: } P \Rightarrow I \checkmark \\
Z := X & \quad \text{Preservation: } \{I \land b\} S \{I\} \\
J := 1 & \quad \{Z=XJ \land J\neq Y\}
\end{align*}$$

while $J \not< \not> Y$ do

$$\begin{align*}
Z := Z+X & \quad Z := Z+X \\
J := J+1 & \quad \{Z=X(J+1) \land J\neq Y\}
\end{align*}$$

end_while

$$\begin{align*}
Z := X(J-1)+1 & \quad J := J+1 \\
{Z=X((J-1)+1) \land J-1\neq Y} & \Rightarrow Z=XJ
\end{align*}$$
Example 3

Use the invariant $I: Z=XJ$ to prove:

$$\{true\}$$

$Z := X$
$J := 1$

while $J<>Y$ do
  $Z := Z+X$
  $J := J+1$
end_while

$$\{Z=XY\}$$

**Initialization:** $P \Rightarrow I \checkmark$

**Preservation:** $\{I \land b\} \downarrow \{I\} \checkmark$

**Finalization:** $(I \land \neg b) \Rightarrow Q$

Does $(Z=XJ \land J=Y) \Rightarrow Z=XY$?

Yep!
Example 3

Use the invariant $I: Z=XJ$ to prove:

{true} \\
\begin{align*}
Z &:= X \\
J &:= 1 \\
\text{while } J &\not<\not> Y \text{ do} \\
&\quad Z := Z+X \\
&\quad J := J+1 \\
\text{end}_\text{while}
\end{align*}

{Z=XY}

Initialization: $P \Rightarrow I$ √

Preservation: $\{I \land b\} \subseteq \{I\}$ √

Finalization: $(I \land \neg b) \Rightarrow Q$ √
Exercise

- See **WHILE LOOP VERIFICATION EXERCISE** on course website
Some Limitations of Formal Verification

• Difficulties can arise when dealing with:
  – parameters
  – pointers
  – synthesis of invariants
  – decidability of verification conditions
  – concurrency
Some Limitations of Formal Verification (cont’d)

• In addition, a formal specification:
  – may be expensive to produce
  – may be incorrect and/or incomplete
  – normally reflects *functional* requirements only

• Will the proof process be manual or automatic? Who will prove the proof?
That’s all, folks, but if you like formal verification...

- Take CEN 6070, Software Testing & Verification and learn about:
  - deriving invariants using the Invariant Status Theorem,
  - proving termination using the Method of Well-Founded Sets,
  - Predicate transforms (“weakest pre-conditions”)
  - function-theoretic verification (prove the correctness of loops without invariants!)
  - and MUCH more!
Proofs of Correctness: An Introduction to Axiomatic Verification

CEN 5035
Software Engineering

Prepared by
Stephen M. Thebaut, Ph.D.
University of Florida