Programmeren en Correctheid
Proof Outlines and Partial Correctness

Michiel Helvensteijn (mhelvens@liacs.nl)

LIACS, Leiden University
CWI, Amsterdam

16 April 2010
Introduction
Last Time

Last time, on P&C

- A language for assertions.
Last time, on P&C

- A language for assertions.
- A semantics for assertions, in terms of operational semantics.
Last time, on P&C

- A language for assertions.
- A semantics for assertions, in terms of operational semantics.
- An axiomatic semantics for symbolically deriving assertions and for proving correctness of Hoare triples.
Problem

- Those axiomatic proof trees are huge.
Those axiomatic proof trees are huge.
They repeat code in several places.
Introduction
And Now

Problem

- Those axiomatic proof trees are huge.
- They repeat code in several places.
- Building them is prone to human error.
Introduction

Problem

- Those axiomatic proof trees are huge.
- They repeat code in several places.
- Building them is prone to human error.
- It still requires creativity to know when and where to use the cons rule.
Introduction
And Now

Problem
- Those axiomatic proof trees are huge.
- They repeat code in several places.
- Building them is prone to human error.
- It still requires creativity to know when and where to use the \textit{cons} rule.

Solution
We will find a way to use the structure of the program to structure the proof of correctness.
To Do

- Define the concept of the **weakest precondition**.
Introduction

And Now

To Do

- Define the concept of the weakest precondition.
- Recognize (again) the while loop problem.
To Do

- Define the concept of the **weakest precondition**.
- Recognize (again) the *while loop* problem.
- Define a new proof system: **proof outlines**.
What does it mean for a condition to be weaker or stronger?
Weakest Precondition

Definition

What does it mean for a condition to be weaker or stronger?

There exists a refinement order $\preceq$ over all assertions:

$$(\phi \preceq \psi) \implies (\models \psi \Rightarrow \phi)$$

It is a complete partial order.
What does it mean for a condition to be weaker or stronger?

There exists a refinement order $\preceq$ over all assertions:

$$(\phi \preceq \psi) \implies (\models \psi \implies \phi)$$

It is a complete partial order.

True is smallest. It says nothing about a state. It is weak.
What does it mean for a condition to be weaker or stronger?

There exists a refinement order $\leq$ over all assertions:

$$(\phi \leq \psi) \implies (\models \psi \implies \phi)$$

It is a complete partial order.

true is smallest. It says nothing about a state. It is weak.

false is largest. No state can satisfy it. It is strong.
We would like to have a computable function $wp(C, \psi)$ which computes assertion $\phi$ such that:

- $\phi$ is a **precondition** of $\psi$ for $C$.
- $\phi$ is the **weakest** precondition.
We would like to have a computable function \( wp(C, \psi) \) which computes assertion \( \phi \) such that:

- \( \phi \) is a **precondition** of \( \psi \) for \( C \).
- \( \phi \) is the **weakest** precondition.

**Weakest Precondition (formalized)**

\( \phi = wp(C, \psi) \) is defined such that:

- \( \vdash_{\text{tot}} \langle \phi \rangle \subseteq C \langle \psi \rangle \)
- \( \forall \pi: (\vdash_{\text{tot}} \langle \pi \rangle \subseteq C \langle \psi \rangle) \implies (\vdash \pi \implies \phi) \)
The weakest liberal precondition is almost the same thing, but the set of states is expanded with those for which $C$ does not terminate.
Weakest Precondition

Definition

The weakest liberal precondition is almost the same thing, but the set of states is expanded with those for which $C$ does not terminate.

Weakest Liberal Precondition (formalized)

$\phi = wlp(C, \psi)$ is defined such that:

1. $\models_{\text{par}} (|\phi|) \parallel C (|\psi|)$
2. $\forall \pi : (\models_{\text{par}} (|\pi|) \parallel C (|\psi|)) \implies (\models \pi \Rightarrow \phi)$
For each $C$ and $\psi$, there is only one $wp(C, \psi)$ (modulo equivalence). Because, if there were two: $\phi_1$ and $\phi_2$ . . .

\[
\frac{\{} \phi_1 \{} \ C \ \{ \psi \} \ \{ \phi_2 \} \ C \ \{ \psi \} }{\{} \phi_1 \lor \phi_2 \} \ C \ \{ \psi \} }
\]

so $\phi_1 \lor \phi_2$ is also a precondition, and is also at least as weak as both $\phi_1$ and $\phi_2$. So either $\phi_1 \equiv \phi_2$ or $\phi_1 \lor \phi_2$ is weaker.
The definition of the weakest precondition follows the axiomatic semantics rules.
The definition of the weakest precondition follows the axiomatic semantics rules.

\[
wp(\text{skip}, \phi) = \phi
\]
The definition of the weakest precondition follows the axiomatic semantics rules.

\[
\boxed{
\begin{align*}
\wp(\langle \phi \rangle \text{skip}, \psi) &= \psi
\end{align*}
}\]

\boxed{\text{Skip}}

\[
\langle \phi \rangle \text{skip} \langle \phi \rangle = \text{skip}
\]
Weakest Precondition

Assignment

\[
\frac{\phi[A/x]}{x := A \vdash \phi} \quad \text{ass}
\]
Weakest Precondition

The Rules

Assignment

\[
\frac{\langle \phi[A/x] \rangle}{x := A \langle \phi \rangle} \quad \text{ass}
\]

\[
\text{wp}(x := A, \psi) = \psi[A/x]
\]
Sequential Composition

\[
\begin{array}{c}
\phi \quad C_1 \pi \\
\phi \\
\phi \quad C_1; C_2 \psi
\end{array}
\]

seq
Sequential Composition

\[
\begin{array}{c}
(\phi) \quad C_1 \quad (\pi) \\
(\pi) \quad C_2 \quad (\psi)
\end{array}
\quad \frac{\text{seq}}{(\phi) \quad C_1; \ C_2 \quad (\psi)}
\]

\[
\text{wp}(C_1; \ C_2, \psi) = \text{wp}(C_1, \text{wp}(C_2, \psi))
\]
Weakest Precondition

The Rules

Conditional

\[
\frac{(\phi \land B) \quad C_1 \quad (\psi) \quad (\phi \land \neg B) \quad C_2 \quad (\psi)}{\phi \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \quad (\psi)}
\]

\text{if}
Weakest Precondition

The Rules

Conditional

\[
\frac{(\phi \land B) \quad C_1 \quad (\psi)}{(\phi) \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \quad (\psi)} \quad \text{if} \]

but also

Conditional

\[
\frac{(\phi_1) \quad C_1 \quad (\psi) \quad (\phi_2) \quad C_2 \quad (\psi)}{(B \Rightarrow \phi_1) \land (\neg B \Rightarrow \phi_2) \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \quad (\psi)} \quad \text{if2}
\]
Weakest Precondition
The Rules

Conditional

\[
\frac{(\phi \land B) \quad C_1 \quad (\psi) \quad (\phi \land \neg B) \quad C_2 \quad (\psi)}{(\phi) \text{ if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \ (\psi)} \quad \text{if}
\]

but also

Conditional

\[
\frac{(\phi_1) \quad C_1 \quad (\psi) \quad (\phi_2) \quad C_2 \quad (\psi)}{(B \Rightarrow \phi_1) \land (\neg B \Rightarrow \phi_2) \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \ (\psi)} \quad \text{if2}
\]

\[
wp(\text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi}, \psi) = (B \Rightarrow wp(C_1, \psi)) \land (\neg B \Rightarrow wp(C_2, \psi))
\]
Weakest Precondition
The Rules

Loop

\[
\frac{(\phi \land B) \Rightarrow C (\phi)}{(\phi) \quad \text{while } B \text{ do } C \text{ od} (\phi \land \neg B)} \quad \text{while}
\]
Weakest Precondition
The Rules

All the other axiomatic semantics rules could be easily rewritten to find the weakest precondition. The while-loop is going to be a bit more difficult.
Weakest Precondition
The Rules

Weakest precondition function for loops

There is, in general, no computable function. But we can approach one. Recall:

\[
\text{while } B \text{ do } C \text{ od } \equiv \text{ if } B \text{ then } C ; \text{ while } B \text{ do } C \text{ od fi}
\]
Weakest Precondition

The Rules

Weakest precondition function for loops

There is, in general, no computable function. But we can approach one. Recall:

\[ \text{while } B \text{ do } C \text{ od} \equiv \text{if } B \text{ then } C; \text{ while } B \text{ do } C \text{ od fi} \]

\[ W = \text{while } B \text{ do } C \text{ od} \quad W_{wp} = wp(W, \psi) \]

\[ W_{wp} = wp(\text{if } B \text{ then } C; W \text{ fi}, \psi) = (B \Rightarrow wp(C, W_{wp})) \land (\neg B \Rightarrow \psi) \]
### Weakest Precondition

#### The Rules

**Weakest precondition function for loops**

There is, in general, no computable function. But we can approach one. Recall:

\[
\text{while } B \text { do } C \text { od} \equiv \text{if } B \text{ then } C ; \text{ while } B \text{ do } C \text{ od fi}
\]

\[
W = \text{while } B \text{ do } C \text{ od} \quad W_{wp} = \text{wp}(W, \psi)
\]

\[
W_{wp} = \text{wp} (\text{if } B \text{ then } C ; W \text{ fi}, \psi) = (B \Rightarrow \text{wp}(C, W_{wp})) \land (\neg B \Rightarrow \psi)
\]

This is a recursive equation, and not of much use to us, unless we know some upper limit on the number of iterations.
For each command in our language, we have:

\[
\begin{align*}
\text{wp}(C, \text{true}) &= \text{true} \\
\text{wp}(C, \psi) \implies \text{wp}(C, \psi') \\
\text{wp}(C, \psi \land \psi') &= \text{wp}(C, \psi) \land \text{wp}(C, \psi') \\
\text{wp}(C, \psi \lor \psi') &= \text{wp}(C, \psi) \lor \text{wp}(C, \psi') \\
\text{wp}(C, \text{false}) &= \text{all states in which } C \text{ does not terminate.}
\end{align*}
\]
For each command in our language, we have:

- \( \text{wp}(C, \text{true}) = \text{true} \)
For each command in our language, we have:

- \( \text{wp}(C, \text{true}) = \text{true} \)
- if \( \psi \Rightarrow \psi' \) then \( \text{wp}(C, \psi) \Rightarrow \text{wp}(C, \psi') \)
For each command in our language, we have:

- $\text{wp}(C, \text{true}) = \text{true}$
- if $\psi \Rightarrow \psi'$ then $\text{wp}(C, \psi) \Rightarrow \text{wp}(C, \psi')$
- $\text{wp}(C, \psi \land \psi') = \text{wp}(C, \psi) \land \text{wp}(C, \psi')$
For each command in our language, we have:

- $wp(C, \text{true}) = \text{true}$
- if $\psi \Rightarrow \psi'$ then $wp(C, \psi) \Rightarrow wp(C, \psi')$
- $wp(C, \psi \land \psi') = wp(C, \psi) \land wp(C, \psi')$
- $wp(C, \psi \lor \psi') = wp(C, \psi) \lor wp(C, \psi')$
Weakest Precondition
General Properties

For each command in our language, we have:

- \( \text{wp}(C, \text{true}) = \text{true} \)
- If \( \psi \Rightarrow \psi' \) then \( \text{wp}(C, \psi) \Rightarrow \text{wp}(C, \psi') \)
- \( \text{wp}(C, \psi \land \psi') = \text{wp}(C, \psi) \land \text{wp}(C, \psi') \)
- \( \text{wp}(C, \psi \lor \psi') = \text{wp}(C, \psi) \lor \text{wp}(C, \psi') \)
- \( \text{wp}(C, \text{false}) \) represents all states in which \( C \) does not terminate.
Proof Outlines: the idea

For the program $P = C_1; C_2; \ldots; C_n$, we want to prove:

$$\vdash_{\text{par}} \left( \phi_0 \right) \quad P \quad \left( \phi_n \right)$$
Proof Outlines: the idea

For the program $P = C_1; C_2; \ldots; C_n$, we want to prove:

$$\vdash_{\text{par}} (\phi_0) \quad P \quad (\phi_n)$$

We can split the problem up into smaller problems if we can find the assertions $\phi_i$ such that:

$$\forall 0 \leq i < n : \quad \vdash_{\text{par}} (\phi_i) \quad C_{i+1} \quad (\phi_{i+1})$$
We need to define a calculus for presenting a proof of $\vdash_{par} \left( \phi_0 \right) \parallel \left( \phi_n \right)$ by interleaving assertions with code.
We need to define a calculus for presenting a proof of
\( \vdash_{\text{par}} (\phi_0 \ P \ \phi_n) \) by interleaving assertions with code.

**General Proof Outline**

\[
\begin{align*}
\phi_0 \\
C_1 \\
(\phi_1) & \quad \text{(justification)} \\
C_2 \\
(\phi_2) & \quad \text{(justification)} \\
& \quad \vdots \\
C_n & \\
(\phi_n) & \quad \text{(justification)}
\end{align*}
\]
We need to define a calculus for presenting a proof of \( \vdash_{\text{par}} (\phi_0) \quad P \quad (\phi_n) \) by interleaving assertions with code.

**General Proof Outline**

\[
(\phi_0) \quad C_1; \quad (\phi_1) \quad \text{(justification)} \\
C_2; \quad (\phi_2) \quad \text{(justification)} \\
\vdots \quad C_n \\
(\phi_n) \quad \text{(justification)}
\]

Sequential composition is implicit!
Proof Outlines
The Rules

How to find the assertions?

For each command in the language, we define the shape of the proof outline based mostly on weakest preconditions.
Proof Outlines
The Rules

How to find the assertions?

For each command in the language, we define the shape of the proof outline based mostly on weakest preconditions.

\[
\begin{align*}
(\phi) \quad \text{skip} \quad (\phi)
\end{align*}
\]
How to find the assertions?

For each command in the language, we define the shape of the proof outline based mostly on weakest preconditions.

\[
\begin{align*}
&\phi \quad \text{skip} \quad \phi \\
&\phi[A/x] \quad x := A \quad \phi
\end{align*}
\]
How to find the assertions?

For each command in the language, we define the shape of the proof outline based mostly on weakest preconditions.

\[
\begin{align*}
\frac{}{\langle \phi \rangle \text{skip} \langle \phi \rangle} & \quad \text{skip} \\
\frac{}{\langle \phi[A/x] \rangle \text{\(x := A\)} \langle \phi \rangle} & \quad \text{ass} \\
\frac{\vdash \phi \Rightarrow \psi}{\langle \phi \rangle \langle \psi \rangle} & \quad \text{implied}
\end{align*}
\]
Example

To prove: $\vdash_{\text{par}} (\parallel y = 5 \parallel) \ x := y + 1 \ (\parallel x = 6 \parallel)$
Proof Outlines

The Rules

Example

To prove: \( \vdash_{\text{par}} ((y = 5) \quad x := y + 1) \quad (x = 6) \)

\( (y = 5) \)

\( x := y + 1 \)

\( (x = 6) \)
Example

To prove: \( \vdash_{\text{par}} (\| y = 5 \|) \ x := y + 1 \ (\| x = 6 \|) \)

\[
\begin{align*}
(\| y = 5 \|) \\
(\| y + 1 = 6 \|) \\
x := y + 1 \\
(\| x = 6 \|)
\end{align*}
\]
Example

To prove: \( \vdash_{\text{par}} \left( \| y = 5 \| \right) x := y + 1 \left( \| x = 6 \| \right) \)

\( \| y = 5 \| \)
\( \| y + 1 = 6 \| \) implied
\( x := y + 1 \)
\( \| x = 6 \| \) assignment
Example

To prove: $\vdash_{\text{par}} (\mid y = 5 \mid) \quad x := y + 1 \quad (\mid x = 6 \mid)$

$\mid y = 5 \mid$

$\mid y + 1 = 6 \mid$  implied

$x := y + 1$

$\mid x = 6 \mid$  assignment

Left to prove: $\vdash y = 5 \Rightarrow y + 1 = 6$
Proof Outlines

The Rules

\[
\frac{(\phi) \ C_1 \ (\pi) \ (\pi) \ C_2 \ (\psi)}{
(\phi) \ C_1; \ (\pi) \ C_2 \ (\psi)\} \quad \text{seq}
\]
Proof Outlines

The Rules

\[
\frac{\{ \phi \} \ C_1 \ (\pi) \ (\pi) \ C_2 \ (\psi)}{\{ \phi \} \ C_1 ; \ (\pi) \ C_2 \ (\psi)} \quad \text{seq}
\]

\[
\frac{\{ \phi_1 \} \ C_1 \ (\psi) \ (\phi_2) \ C_2 \ (\psi)}{\{ (B \Rightarrow \phi_1) \land (\neg B \Rightarrow \phi_2) \} \ \text{if} \ B \ \text{then} \ \{ \phi_1 \} \ C_1 \ (\psi) \ \text{else} \ (\phi_2) \ C_2 \ (\psi) \ \text{fi} \ (\psi)} \quad \text{if2}
\]
Example

\[ \neg \text{par} (\{ \text{true} \} \; z := x; \; z := z + y; \; u := z \; (\; u = x + y \; )) \]
Example

\[ \vdash_{\text{par}} (\text{ltrue}) \quad z := x; \quad z := z + y; \quad u := z \quad (u = x + y) \]

\[ (\text{ltrue}) \]

\[ z := x; \]

\[ z := z + y; \]

\[ u := z \]

\[ (u = x + y) \]
The Idea

Proof Outlines

The Rules

Example

\[ \vdash_{\text{par}} (\text{true}) \quad z := x; \quad z := z + y; \quad u := z \quad (u = x + y) \]

\[ (\text{true}) \]

\[ z := x; \]

\[ z := z + y; \]

\[ (z = x + y) \]

\[ u := z \]

\[ (u = x + y) \]
Proof Outlines
The Rules

Example

\[ \vdash_{\text{par}} (\text{true}) \quad \begin{array}{l}
z := x; \\
z := z + y; \\
u := z \\
\end{array} \quad (u = x + y) \]

\[ (\text{true}) \]

\[ \begin{array}{l}
z := x; \\
(\text{true}) \\
z := z + y; \\
(\text{true}) \\
u := z \\
(\text{true}) \\
u = x + y \]

Programmeren en Correctheid
Example

\[ \vdash \text{par (} \text{true}) z := x; \ z := z + y; \ u := z (u = x + y) \]

\[
\text{\{true\)} \\
\text{\{x + y = x + y\}} \\
z := x; \\
\text{\{z + y = x + y\}} \\
z := z + y; \\
\text{\{z = x + y\}} \\
u := z \\
\text{\{u = x + y\}}
\]
Proof Outlines

The Rules

Example

\[ \vdash_{\text{par}} (\text{true}) \quad z := x; \quad z := z + y; \quad u := z \quad (u = x + y) \]

\[ \begin{align*}
& (\text{true}) \\
& (x + y = x + y) \quad \text{implied} \\
& z := x; \\
& (z + y = x + y) \quad \text{assignment} \\
& z := z + y; \\
& (z = x + y) \quad \text{assignment} \\
& u := z \\
& (u = x + y) \quad \text{assignment}
\]
Proof Outlines
The Rules

Example

\[ \parallel \text{true} \parallel \underbrace{z := x; \ z := z + y; \ u := z}_{(u = x + y)} \]

\[ \parallel \text{true} \parallel \]
\[ \parallel x + y = x + y \parallel \quad \text{implied} \]
\[ z := x; \]
\[ \parallel z + y = x + y \parallel \quad \text{assignment} \]
\[ z := z + y; \]
\[ \parallel z = x + y \parallel \quad \text{assignment} \]
\[ u := z \]
\[ \parallel u = x + y \parallel \quad \text{assignment} \]

Left to prove:  \[ \parallel \text{true} \parallel \Rightarrow x + y = x + y \]
The Idea

The Rules

Example

\[ \vdash_{\text{par}} (\text{true}) \quad a := x + 1; \quad \textbf{if} \ a = 1 \ \textbf{then} \ y := 1 \ \textbf{else} \ y := a \ \textbf{fi} \quad (y = x + 1) \]
We need to discover an invariant $I$:

- $I$ needs to hold before and after the loop.
- $I$ needs to hold before and after each iteration $C$.
- $I$ doesn’t need to hold during the execution of $C$.
- $I$ needs to be strong enough.
- $I$ needs to be weak enough.

Finding $I$ cannot be done mechanically! It requires creativity.
Proof Outlines

The Rules

\[
\begin{align*}
(\langle I \land B \rangle \; C \; \langle I \rangle) \\
\langle I \rangle \; \text{while} \; B \; \text{do} \; (\langle I \land B \rangle \; C \; \langle I \rangle) \; \text{od} \; (\langle I \land \neg B \rangle)
\end{align*}
\]  

while

We need to discover an invariant \( I \):

- \( I \) needs to hold before and after the loop.
- \( I \) needs to hold before and after each iteration \( C \).
- \( I \) doesn’t need to hold during the execution of \( C \).
- \( I \) needs to be strong enough.
- \( I \) needs to be weak enough.
Proof Outlines
The Rules

\[
\frac{(I \land B) \ \ C \ \ (I)}{(I) \ \ \text{while} \ B \ \ \text{do} \ (I \land B) \ \ C \ \ (I) \ \ \text{od} \ (I \land \neg B)} \quad \text{while}
\]

We need to discover an invariant \( I \):

- \( I \) needs to hold before and after the loop.
- \( I \) needs to hold before and after each iteration \( C \).
- \( I \) doesn’t need to hold during the execution of \( C \).
- \( I \) needs to be strong enough.
- \( I \) needs to be weak enough.

Finding \( I \) cannot be done mechanically! It requires creativity.
It is useful to consider the relations between variables that the loop body maintains. You need to capture enough of them to find a useful loop invariant.
It is useful to consider the relations between variables that the loop body maintains. You need to capture enough of them to find a useful loop invariant.

Example

\[ W = \text{while } x > 0 \text{ do } y := x \times y; \ x := x - 1 \text{ od.} \]

To prove: \( \vdash_{\text{par}} (x_0 = x \land x_0 \geq 0 \land y = 1) \ W (y = x_0!) \)
Proof trees are bulky and tedious to construct.
Conclusion

Summary

- Proof trees are bulky and tedious to construct.
- We define the weakest (liberal) precondition.
Proof trees are bulky and tedious to construct.
We define the weakest (liberal) precondition.
We define a new, more streamlined, proof system: the proof outline.
Conclusion

Summary

- Proof trees are bulky and tedious to construct.
- We define the weakest (liberal) precondition.
- We define a new, more streamlined, proof system: the proof outline.

Likely exam assignment

“Prove Hoare triple $\vdash_{\text{par}} (\phi) \ C (\psi)$ using a proof outline.”