Programmeren en Correctheid

Axiomatic Semantics

Michiel Helvensteijn (mhelvens@liacs.nl)

LIACS, Leiden University

CWI, Amsterdam

9 April 2010
Overview

March 26
Syntax Rules, Operational Semantics Rules, Proof Trees

April 9
Axiomatic Semantics Rules

April 16
Weakest Preconditions, Proof Outlines

May 7
Total Correctness Rules

May 21
Arrays
Last time, on P&C

- A syntax for sequential programs.
A syntax for sequential programs.

An operational semantics (transition system) for ‘running’ those programs from a starting state. A computation may terminate in a state or run forever. These semantics define what the code means.
We would also like to have a semantics for reasoning about program correctness.
We would also like to have a semantics for reasoning about program correctness.

We need:

- A logical language for making assertions about programs:
  - The program terminates.
  - If $x = 0$ then $y = z + 1$ throughout the rest of the execution of the program.
  - If and when the program terminates, then $x = y + z$. 
We would also like to have a semantics for reasoning about program correctness.

We need:

- A logical language for making assertions about programs:
  - The program terminates.
  - If $x = 0$ then $y = z + 1$ throughout the rest of the execution of the program.
  - If and when the program terminates, then $x = y + z$.

- A proof system, to prove that the program is correct with regard to those assertions.
Why?

- Documentation of programs and interfaces: Design by Contract (Meyer)
- Guidance of program design
- Proving the correctness of algorithms
- Proving the absence of bugs (array bounds, null dereference)
- Proof-carrying code
Introduction

Why?

- Documentation of programs and interfaces: Design by Contract (Meyer)
- Guidance of program design
- Proving the correctness of algorithms
- Proving the absence of bugs (array bounds, null dereference)
- Proof-carrying code

Why not testing?

Dijkstra: “Program testing can be used to show the presence of bugs, but never to show their absence!”
Example

Write a program that computes a number $y$ whose square is less than the input $x$. 

But what if $x = -4$? There would be no such $y$!
Example
Write a program that computes a number $y$ whose square is less than the input $x$.

Example (Formalized)
Write program $P$ such that:

$$P \ (\ y \times y < x \ )$$

But what if $x = -4$? There would be no such $y$!
Example
Write a program that computes a number $y$ whose square is less than the input $x$.

Example (Formalized)
Write program $P$ such that:

$$P \left( y \times y < x \right)$$

But what if $x = -4$? There would be no such $y$!
We also need to specify restrictions on the input state.
We also need to specify restrictions on the input state.

**Example (Corrected)**

Write a program that, given that the input $x$ is a positive number, computes a number $y$ whose square is less than $x$. 

**Program Correctness**

**Examples**
We also need to specify restrictions on the input state.

**Example (Corrected)**
Write a program that, given that the input $x$ is a positive number, computes a number $y$ whose square is less than $x$.

**Example (Corrected, Formalized)**
Write program $P$ such that:

$$
\langle x > 0 \rangle \ P \ \langle y \times y < x \rangle
$$
Partial Correctness

If the command $C$ terminates when it is executed in a state that satisfies $\phi$, then the resulting state will satisfy $\psi$:

$$\vdash_{\text{par}} (\phi) \triangleright C (\psi)$$

Program termination is not required for partial correctness.
Program Correctness
Partial and Total Correctness

Examples

\[ \vdash_{\text{par}} \left( y \leq x \right) z := x; z := z + 1 \left( y < z \right) \]
Program Correctness
Partial and Total Correctness

Examples

\[ \models_{\text{par}} (y \leq x) \quad z := x; \quad z := z + 1 \quad (y < z) \]

\[ \models_{\text{par}} (true) \quad \textbf{while} \quad true \quad \textbf{do} \quad \textbf{skip} \quad \textbf{od} \quad (false) \]
Program Correctness
Partial and Total Correctness

**Examples**

\[ \models_{\text{par}} (y \leq x) \quad z := x; \quad z := z + 1 \quad (y < z) \]

\[ \models_{\text{par}} (\text{true}) \quad \text{while } \text{true} \quad \text{do } \text{skip} \quad \text{od} \quad (\text{false}) \]

\[ \text{Fact } \equiv \quad y := 1; \quad z := 0; \]
\[ \quad \text{while } z \neq x \quad \text{do} \]
\[ \quad \quad z := z + 1; \]
\[ \quad \quad y := y \ast z \]
\[ \quad \text{od} \]

\[ \models_{\text{par}} (x \geq 0) \quad \text{Fact} \quad (y = x!) \]
Total Correctness

If the command $C$ is executed in a state that satisfies $\phi$, then $C$ is guaranteed to terminate and the resulting state will satisfy $\psi$:

$$\models_{\text{tot}} \left( \phi \right) \implies C \left( \psi \right)$$

Program termination is required for total correctness.
Examples

\[ \models_{\text{tot}} (y \leq x) z := x; z := z + 1 \quad (y < z) \]
## Program Correctness
### Partial and Total Correctness

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\models_{\text{tot}} (y \leq x) \quad z := x; \quad z := z + 1 \quad (y &lt; z)$</td>
</tr>
<tr>
<td>$\not\models_{\text{tot}} (true) \quad \textbf{while} \quad \text{true} \quad \textbf{do} \quad \textbf{skip} \quad \textbf{od} \quad (false)$</td>
</tr>
</tbody>
</table>
Program Correctness
Partial and Total Correctness

Examples

\[ \models_{\text{tot}} (y \leq x) \quad z := x; \quad z := z + 1 \quad (y < z) \]

\[ \not\models_{\text{tot}} (true) \quad \textbf{while} \quad true \quad \textbf{do} \quad \textbf{skip} \quad \textbf{od} \quad (false) \]

\[ \models_{\text{tot}} (false) \quad \textbf{while} \quad true \quad \textbf{do} \quad \textbf{skip} \quad \textbf{od} \quad (true) \]
Program Correctness
Partial and Total Correctness

Examples

\[ \models_{\text{tot}} \langle y \leq x \rangle \quad z := x; \quad z := z + 1 \quad \langle y < z \rangle \]

\[ \not\models_{\text{tot}} \langle \text{true} \rangle \quad \textbf{while} \quad \text{true} \quad \textbf{do} \quad \texttt{skip} \quad \textbf{od} \quad \langle \text{false} \rangle \]

\[ \models_{\text{tot}} \langle \text{false} \rangle \quad \textbf{while} \quad \text{true} \quad \textbf{do} \quad \texttt{skip} \quad \textbf{od} \quad \langle \text{true} \rangle \]

\[ \text{Fact} \equiv \quad y := 1; \quad z := 0; \]
\[ \quad \textbf{while} \quad z \neq x \quad \textbf{do} \]
\[ \quad \quad z := z + 1; \]
\[ \quad \quad y := y \times z \]
\[ \quad \textbf{od} \]

\[ \models_{\text{tot}} \langle x \geq 0 \rangle \quad \text{Fact} \quad \langle y = x! \rangle \]
Program Correctness

In Terms of Operational Semantics

Hoare Triple for Partial Correctness

\[ \models_{\text{par}} (\phi) \quad C \quad (\psi) \]

If \( \phi \) holds in a state \( \sigma \) and \( \langle C, \sigma \rangle \rightarrow \sigma' \) then \( \psi \) holds in \( \sigma' \).

Hoare Triple for Total Correctness

\[ \models_{\text{tot}} (\phi) \quad C \quad (\psi) \]

If \( \phi \) holds in a state \( \sigma \) then there exists a state \( \sigma' \) such that \( \langle C, \sigma \rangle \rightarrow \sigma' \) and \( \psi \) holds in \( \sigma' \).
Program Correctness
In Terms of Operational Semantics

Hoare Triple for Partial Correctness
$$\models_{\text{par}} (\phi) \ C \ (\psi)$$
If \( \phi \) holds in a state \( \sigma \) and \( \langle C, \sigma \rangle \rightarrow \sigma' \) then \( \psi \) holds in \( \sigma' \).

Hoare Triple for Total Correctness
$$\models_{\text{tot}} (\phi) \ C \ (\psi)$$
If \( \phi \) holds in a state \( \sigma \) then there exists a state \( \sigma' \) such that \( \langle C, \sigma \rangle \rightarrow \sigma' \) and \( \psi \) holds in \( \sigma' \).

We need to do the following:
- Give syntax for the language of assertions \((\phi, \psi)\).
- Say when an assertion \((\phi, \psi)\) holds in a state \((\sigma)\).
- Give semantics rules for reasoning directly with assertions.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z \in \mathbb{Z}$</td>
<td>${\ldots, -2, -1, 0, 1, 2, \ldots}$</td>
</tr>
<tr>
<td>$b \in \mathbb{B}$</td>
<td>${true, false}$</td>
</tr>
<tr>
<td>$x \in \mathbb{V}$</td>
<td>${a, b, \ldots, y, z}$</td>
</tr>
<tr>
<td>$v \in \mathbb{L}$</td>
<td>${i, j, k, \ldots}$</td>
</tr>
<tr>
<td>$A$</td>
<td>Extended Arithmetic expressions</td>
</tr>
<tr>
<td>$B$</td>
<td>Boolean expressions</td>
</tr>
<tr>
<td>$B, \phi, \psi$</td>
<td>Extended Boolean expressions</td>
</tr>
<tr>
<td>$C$</td>
<td>Commands</td>
</tr>
</tbody>
</table>
Extended Syntax

Extended Arithmetic Expressions

\[ A ::= z \mid x \mid v \mid A + A \mid A \times A \]
Extended Syntax

Extended Arithmetic Expressions

\[ A ::= z \mid x \mid v \mid A + A \mid A * A \]

Extended Boolean Expressions (Assertions)

\[ B ::= \text{true} \mid \text{false} \mid A < A \mid B \land B \mid \neg B \mid \forall v : B \]
### Extended Syntax

#### Program Variables and Logical Variables

#### Bound, Free and Substituted Variables

#### The Meaning of Assertions

### Extended Arithmetic Expressions

$$A ::= z \mid x \mid v \mid A + A \mid A \times A$$

### Extended Boolean Expressions (Assertions)

$$B ::= \text{true} \mid \text{false} \mid A < A \mid B \land B \mid \neg B \mid \forall v : B$$

- Derived operator: $$\exists v : A \equiv \neg \forall v : \neg A$$
Extended Syntax
Program Variables and Logical Variables
Bound, Free and Substituted Variables
The Meaning of Assertions

Extended Arithmetic Expressions

\[ A ::= z \ | \ x \ | \ v \ | \ A + A \ | \ A * A \]

Extended Boolean Expressions (Assertions)

\[ B ::= \text{true} \ | \ \text{false} \ | \ A < A \ | \ B \land B \ | \neg B \ | \forall v : B \]

- Derived operator: \( \exists v : A \equiv \neg \forall v : \neg A \)

The same equivalence rules and derived operations apply as before, but we now add logical variables to the mix.
Why program variables in our assertion language?

To express properties of a program state as basic assertions.
Why program variables in our assertion language?

To express properties of a program state as basic assertions.

Examples

\[ x = 5 \]

i.e. "The value of \( x \) is 5."
Why program variables in our assertion language?

To express properties of a program state as basic assertions.

Examples

\[ x = 5 \]

i.e. "The value of \( x \) is 5."

\[ x = 5 \implies y + 1 = x \times (y - x) \]

i.e. "If the value of \( x \) is 5 then the value of \( y + 1 \) is \( x \) times \( y - x \)."
Why logical variables in our assertion language?

To express properties that quantify over all integers:

$$\exists i : x = i \times i$$

i.e. "The value of $x$ is the square of an integer."
## Assertions

### Program Variables and Logical Variables

**Why logical variables in our assertion language?**

1. To express properties that quantify over all integers:
   \[ \exists i : x = i \times i \]  
   i.e. "The value of \( x \) is the square of an integer."

2. To remember the value of a program variable that was destroyed by a computation:

   ```plaintext
   Fact2 \equiv y := 1; \textbf{while} x \neq 0 \textbf{do}
   y := y \times x;
   x := x - 1
   \textbf{od}
   \)
   \]  
   \[ \not \vdash_{\text{par}} (x \geq 0) \textbf{Fact2} (y = x!)
   \[ \vdash_{\text{par}} (x = i \land x \geq 0) \textbf{Fact2} (y = i!)
   \]
Bound variable

A logical variable occurrence is bound in an assertion if it refers to a quantified variable:

$$\exists i : x = i \times y$$
Bound variable

A logical variable occurrence is **bound** in an assertion if it refers to a quantified variable:

\[ \exists i : x = i \times y \]

Free variable

A logical variable occurrence is **free** in an assertion if it is not bound:

\[ i < 42 \wedge \forall i : j + i = 3 \]
Variable Substitution

For an assertion $\phi$, logical variable $v$ and extended arithmetic expression $A$, we define $\phi[A/v]$ as the assertion $\phi$ in which all free occurrences of $v$ are substituted by $A$. 
Variable Substitution

For an assertion $\phi$, logical variable $v$ and extended arithmetic expression $A$, we define $\phi[A/v]$ as the assertion $\phi$ in which all free occurrences of $v$ are substituted by $A$.

Inductive definition for extended arithmetic expressions

\[
\begin{align*}
  z[A/v] & = z \\
  x[A/v] & = x \\
  v_1[A/v] & = \begin{cases} (A) & \text{if } v = v_1 \\ v_1 & \text{if } v \neq v_1 \end{cases} \\
  (A_1 + A_2)[A/v] & = (A_1[A/v] + A_2[A/v]) \\
  (A_1 \times A_2)[A/v] & = (A_1[A/v] \times A_2[A/v])
\end{align*}
\]
Inductive definition for extended boolean expressions

\[
\begin{align*}
\text{true}[A/v] &= \text{true} \\
\text{false}[A/v] &= \text{false} \\
(\neg \phi)[A/v] &= \neg(\phi[A/v]) \\
(\phi_1 \land \phi_2)[A/v] &= (\phi_1[A/v] \land \phi_2[A/v]) \\
(A_1 < A_2)[A/v] &= (A_1[A/v] < A_2[A/v]) \\
(\forall v_1 : \phi)[A/v] &= \begin{cases} 
(\forall v_1 : \phi) & \text{if } v = v_1 \\
(\forall v_1 : \phi[A/v]) & \text{if } v \neq v_1
\end{cases}
\end{align*}
\]
Inductive definition for extended boolean expressions

\[
\begin{align*}
\text{true}[A/v] &= \text{true} \\
\text{false}[A/v] &= \text{false} \\
(\neg \phi)[A/v] &= \neg (\phi[A/v]) \\
(\phi_1 \land \phi_2)[A/v] &= (\phi_1[A/v] \land \phi_2[A/v]) \\
(A_1 < A_2)[A/v] &= (A_1[A/v] < A_2[A/v]) \\
(\forall v_1 : \phi)[A/v] &= \begin{cases} 
(\forall v_1 : \phi) & \text{if } v = v_1 \\
(\forall v_1 : \phi[A/v]) & \text{if } v \neq v_1
\end{cases}
\end{align*}
\]

A picture is worth...

\[
(\ldots i \ldots i \ldots i \ldots i \ldots)[A/i] = (\ldots (A) \ldots (A) \ldots \ldots (A) \ldots)
\]

(if all those \(i\) occur free)
Problem

We say “ϕ holds in a state σ”, but that may depend on the values of the free logical variables in ϕ.
Problem
We say “$\phi$ holds in a state $\sigma$”, but that may depend on the values of the free logical variables in $\phi$.

Solution
We use interpretations of logical variables. Remember that $\sigma : \forall \rightarrow \mathbb{Z}$. Now we also define the function $I : \mathbb{L} \rightarrow \mathbb{Z}$. 
### Assertions

#### The Meaning of Assertions

**Problem**

We say “\( \varphi \) holds in a state \( \sigma \)”, but that may depend on the values of the free logical variables in \( \varphi \).

**Solution**

We use interpretations of logical variables. Remember that \( \sigma : \mathcal{V} \rightarrow \mathbb{Z} \). Now we also define the function \( I : \mathcal{L} \rightarrow \mathbb{Z} \).

We now say: “\( \varphi \) holds in a state \( \sigma \) under interpretation \( I \)”. (This is only for the moment; we will get rid of \( I \) later.)
Problem

We say “\( \phi \) holds in a state \( \sigma \)”, but that may depend on the values of the free logical variables in \( \phi \).

Solution

We use interpretations of logical variables. Remember that \( \sigma : V \rightarrow \mathbb{Z} \). Now we also define the function \( I : L \rightarrow \mathbb{Z} \).

We now say: “\( \phi \) holds in a state \( \sigma \) under interpretation \( I \)”.

(This is only for the moment; we will get rid of \( I \) later.)

Example

\( i < x \) holds in the state \( \{ x \mapsto 3 \} \) under all interpretations \( I \) such that \( I(i) < 3 \).
The meaning of extended arithmetic expressions

Given a state $\sigma$ and an interpretation $I$, we define the meaning of an expression $A$ as $[A]_\sigma^I$, inductively given by:

- $[z]_\sigma^I = z$
- $[x]_\sigma^I = \sigma(x)$
- $[v]_\sigma^I = I(v)$
- $[A_1 + A_2]_\sigma^I = [A_1]_\sigma^I + [A_2]_\sigma^I$
- $[A_1 \times A_2]_\sigma^I = [A_1]_\sigma^I \times [A_2]_\sigma^I$
The meaning of extended boolean expressions (assertions)

Given a state $\sigma$ and an interpretation $I$, we define $\sigma, I \models \phi$ inductively by:

- $\sigma, I \models true$
- $\sigma, I \not\models false$
- $\sigma, I \models A_1 < A_2 \iff \llbracket A_1 \rrbracket'_\sigma < \llbracket A_2 \rrbracket'_\sigma$
- $\sigma, I \models \phi_1 \land \phi_2 \iff \sigma, I \models \phi_1$ and $\sigma, I \models \phi_2$
- $\sigma, I \models \neg \phi \iff \text{not } \sigma, I \models \phi$
- $\sigma, I \models \forall v : \phi_1 \iff \sigma, I[z/v] \models \phi_1$ for all $z \in \mathbb{Z}$
### Assertions

#### The Meaning of Assertions

**Hoare Triple for Partial Correctness**

\[
\forall \sigma : (\sigma, I \models \phi \land [C, \sigma] \rightarrow \sigma') \Rightarrow (\sigma', I \models \psi)
\]

**Hoare Triple for Total Correctness**

\[
\forall \sigma : (\sigma, I \models \phi) \Rightarrow \exists \sigma' : ([C, \sigma] \rightarrow \sigma' \land \sigma', I \models \psi)
\]
Now to get rid of \( I \)

To give an absolute meaning to

\[
\begin{align*}
(i < x) & \quad x := x + 3 \quad (i < x)
\end{align*}
\]

we have to quantify over all possible interpretations \( I \).

\[
\models_{\text{par}} (i < x) \quad x := x + 3 \quad (i < x) \quad \equiv \quad \forall I : I \models_{\text{par}} (i < x) \quad x := x + 3 \quad (i < x)
\]

\[
\models_{\text{tot}} (i < x) \quad x := x + 3 \quad (i < x) \quad \equiv \quad \forall I : I \models_{\text{tot}} (i < x) \quad x := x + 3 \quad (i < x)
\]
So, we should now know exactly what these mean:

\[
\models_{\text{par}} \left\{ \phi \right\} C \left\{ \psi \right\} \quad \models_{\text{tot}} \left\{ \phi \right\} C \left\{ \psi \right\}
\]
So, we should now know exactly what these mean:

\[ \vdash_{\text{par}} \phi \quad C \quad \psi \quad \vdash_{\text{tot}} \phi \quad C \quad \psi \]

But they are defined in terms of operational semantics! We cannot effectively use such a low level of reasoning to prove the validity of Hoare triples. (At least it won’t be fun.)
So, we should now know exactly what these mean:

\[ \models_{\text{par}} \left[ \phi \right] \ C \left[ \psi \right] \quad \models_{\text{tot}} \left[ \phi \right] \ C \left[ \psi \right] \]

But they are defined in terms of operational semantics! We cannot effectively use such a low level of reasoning to prove the validity of Hoare triples. (At least it won’t be fun.)

So we define a proof system, so we can derive valid assertions from other valid assertions symbolically, and thus prove program correctness.
Axiomatic Semantics
Soundness and Completeness

⊨ is used to indicate that which is true.
⊨ is used to indicate that which is true.

So, let’s assume we have that proof system we want. ⊢ is used to indicate that which we can prove with the system.
Axiomatic Semantics

Soundness and Completeness

⊨ is used to indicate that which is true.

⊢ is used to indicate that which we can prove with the system.

So, let’s assume we have that proof system we want.

Soundness

A proof system is sound iff everything we can prove with it, is in fact true. (This is a very important property!)
Axiomatic Semantics
Soundness and Completeness

⊢ is used to indicate that which we can prove with the system.

So, let’s assume we have that proof system we want.

⊨ is used to indicate that which is true.

Soundness

⊢ \Phi \iff \vdash \Phi

A proof system is sound iff everything we can prove with it, is in fact true. (This is a very important property!)

Completeness

⊢ \Phi \iff \vdash \Phi

A proof system is complete iff everything that is true, can be proved with the system.
The system we are about to define is sound. That is:

\[
\begin{align*}
\vdash_{\text{par}} \phi & \quad \mathcal{C} \quad \psi \quad \implies \quad \vdash_{\text{par}} \phi \quad \mathcal{C} \quad \psi \\
\vdash_{\text{tot}} \phi & \quad \mathcal{C} \quad \psi \quad \implies \quad \vdash_{\text{tot}} \phi \quad \mathcal{C} \quad \psi
\end{align*}
\]

So, everything that’s provable is valid.
The system we are about to define is sound. That is:
\[
\begin{align*}
\vdash_{\text{par}} (| \phi |) C (| \psi |) & \implies \vdash_{\text{par}} (| \phi |) C (| \psi |) \\
\vdash_{\text{tot}} (| \phi |) C (| \psi |) & \implies \vdash_{\text{tot}} (| \phi |) C (| \psi |)
\end{align*}
\]
So, everything that’s provable is valid.

The system is also complete:
\[
\begin{align*}
\vdash_{\text{par}} (| \phi |) C (| \psi |) & \iff \vdash_{\text{par}} (| \phi |) C (| \psi |) \\
\vdash_{\text{tot}} (| \phi |) C (| \psi |) & \iff \vdash_{\text{tot}} (| \phi |) C (| \psi |)
\end{align*}
\]
but only if the underlying assertion language is expressive enough \((\models \phi \implies \vdash \phi)\). We call that \textbf{relative completeness}. 
Axiomatic Semantics
Proof Rules for Partial Correctness

There are derivation rules for each command in the programming language.

\[
\frac{\phi \vdash \text{skip} \vdash \phi}{\phi}
\]
There are derivation rules for each command in the programming language.

\[ \frac{\phi}{\langle \phi \rangle \text{skip} \langle \phi \rangle} \text{skip} \]

\[ \frac{\phi[A/x]}{\langle \phi \rangle \text{ass} \langle \phi \rangle} \text{ass} \]
There are derivation rules for each command in the programming language.

\[
\begin{align*}
\frac{\phi}{\text{skip} \quad \phi} & \quad \text{skip} \\
\frac{\phi[A/x]}{x := A \quad \phi} & \quad \text{ass} \\
\frac{\phi \quad \pi \quad \psi \quad \phi \quad \pi \quad \psi}{C_1 \quad C_2 \quad \psi} & \quad \text{seq}
\end{align*}
\]
Axiomatic Semantics

Proof Rules for Partial Correctness

\[
\begin{array}{c}
(\phi \land B) \quad C_1 \quad (\psi) \\
(\phi \land \neg B) \quad C_2 \quad (\psi) \\
\end{array}
\]

\[
\frac{\begin{array}{c}
(\phi) \\
\text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \quad (\psi)
\end{array}}{\text{if}}
\]
Axiomatic Semantics
Proof Rules for Partial Correctness

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

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

Proof Rules for Partial Correctness

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

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

\[
\frac{\phi \Rightarrow \phi' \quad (\phi') \quad C \quad (\psi') \quad \psi' \Rightarrow \psi}{(\phi) \quad C \quad (\psi)} \quad \text{cons}
\]
Assignment

Prove: $\vdash_{\text{par}} (\text{true}) \ x := 1 \ (x = 1)$. 
Axiomatic Semantics

Examples

Assignment

Prove: \( \vdash_{\text{par}} (\text{true}) \ x := 1 \ (x = 1) \).

Ok, this one is still relatively easy. (Right?)

\[
\vdash \text{true} \Rightarrow 1 = 1 \quad \frac{1 = 1}{\text{ass}} \quad \frac{x := 1}{\text{cons}} \quad \frac{(x = 1)}{(\text{true}) \ x := 1 \ (x = 1)}
\]
Assignment (generalized)

Prove: \( \vdash_{\text{par}} \langle \text{true} \rangle \ x := A \ (x = A) \) where \( x \) does not appear in \( A \).
Axiomatic Semantics

Examples

Assignment (generalized)

Prove: $\vdash_{\text{par}} (\mid \text{true} \mid) \ x := A \ (x = A)$ where $x$ does not appear in $A$.

Because $x$ does not appear in $A$, we have:

$$(x = A)[A/x] = (x[A/x] = A[A/x]) = (A = A)$$

$$\vdash \text{true} \Rightarrow A = A \quad \frac{(A = A) \ x := A \ (x = A)}{(\mid \text{true} \mid) \ x := A \ (x = A)} \quad \text{ass}$$

$$\frac{(A = A) \ x := A \ (x = A)}{(\mid \text{true} \mid) \ x := A \ (x = A)} \quad \text{cons}$$
Axiomatic Semantics

Examples

Conditional

Prove:

\[ \text{par} (| \text{true} | \text{if } y < 1 \text{ then } x := 1 \text{ else } x := y \text{ fi } (| x > 0 |). \]
Axiomatic Semantics
Examples

While

Prove: \( \vdash_{\text{par}} (0 \leq x) \textbf{while } x > 0 \textbf{ do } x := x - 1 \textbf{ od } (x = 0) \).
Axiomatic Semantics

Examples

While (again)

Prove:

\[ \vdash_{\text{par}} (x \leq 0) \quad \text{while } x \leq 5 \quad \text{do } x := x + 1 \quad \text{od } (x = 6). \]
Axiomatic Semantics

Auxiliary Rules

These rules can be derived from the previous ones.

\((\phi)\ C\ (\phi)\) \quad \text{If } \phi \text{ and } C \text{ have no variables in common.}
These rules can be derived from the previous ones.

\[
\begin{align*}
\frac{\phi}{C \phi} & \quad \text{If } \phi \text{ and } C \text{ have no variables in common.} \\
\frac{\phi}{x := A \ (\exists i : \phi[i/x] \land x = A[i/x])} 
\end{align*}
\]
Axiomatic Semantics

Auxiliary Rules

These rules can be derived from the previous ones.

\[
\frac{\phi \quad C \quad \phi}{\exists i : \phi[i/x] \land x = A[i/x]} \quad \text{If } \phi \text{ and } C \text{ have no variables in common.}
\]

\[
\frac{\phi \quad x := A \quad \exists i : \phi[i/x] \land x = A[i/x]}{\phi}
\]

\[
\frac{\phi_1 \quad C_1 \quad \psi \quad \phi_2 \quad C_2 \quad \psi}{(B \Rightarrow \phi_1) \land (\neg B \Rightarrow \phi_2) \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \quad \psi}
\]
These rules can be derived from the previous ones.

\[
\begin{align*}
\phi \quad C \quad \phi \\
\phi \quad x := A \quad \exists i : \phi[i/x] \land x = A[i/x]
\end{align*}
\]

\[
\begin{align*}
\phi_1 \quad C_1 \quad \psi \quad \phi_2 \quad C_2 \quad \psi \\
(B \Rightarrow \phi_1) \land (\neg B \Rightarrow \phi_2) \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} \quad \psi
\end{align*}
\]

\[
\begin{align*}
\phi_1 \quad C \quad \psi \quad \phi_2 \quad C \quad \psi \\
\phi_1 \lor \phi_2 \quad C \quad \psi
\end{align*}
\]
Axiomatic Semantics
Auxiliary Rules

These rules can be derived from the previous ones.

\[ \phi \quad C \quad \phi \quad \text{If } \phi \text{ and } C \text{ have no variables in common.} \]

\[ \phi \quad x := A \quad \exists i : \phi[i/x] \land x = A[i/x] \]

\[ \phi_1 \quad C_1 \quad \psi \quad \phi_2 \quad C_2 \quad \psi \]

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

\[ \phi_1 \quad C \quad \psi \quad \phi_2 \quad C \quad \psi \]

\[ \phi_1 \lor \phi_2 \quad C \quad \psi \]

\[ \phi_1 \land \phi_2 \quad C \quad \psi_1 \land \psi_2 \]
We recognize the need for program specification and verification.

Summary

- We recognize the need for program specification and verification.
We recognize the need for program specification and verification.

We define a syntax and semantics for assertions.

We define a proof system (Axiomatic Semantics) in order to verify Hoare triples.

We define several concepts: free/bound variables, substitution, interpretation, soundness, completeness.

Likely exam assignment

"Prove Hoare triple \(\vdash\) par [L] [\(\phi\) [M] [C] [L] [\(\psi\) [M] [C]] using a proof tree."
We recognize the need for program specification and verification.

We define a syntax and semantics for assertions.

We define a proof system (Axiomatic Semantics) in order to verify Hoare triples.
Conclusion

Summary

- We recognize the need for program specification and verification.
- We define a syntax and semantics for assertions.
- We define a proof system (Axiomatic Semantics) in order to verify Hoare triples.
- We define several concepts: free/bound variables, substitution, interpretation, soundness, completeness.
Conclusion

Summary

- We recognize the need for program specification and verification.
- We define a syntax and semantics for assertions.
- We define a proof system (Axiomatic Semantics) in order to verify Hoare triples.
- We define several concepts: free/bound variables, substitution, interpretation, soundness, completeness.

Likely exam assignment

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