Formal Methods and Functional Programming

Axiomatic Semantics

Peter Müller

Programming Methodology Group
ETH Zurich

Program Correctness

- Formal semantics can be used to prove the correctness of a program
- Partial correctness expresses that if a program terminates then there will be a certain relationship between the initial and the final state
- Total correctness expresses that a program will terminate and there will be a certain relationship between the initial and the final state
  - The relationship is expressed by a formal specification

$$\text{total correctness} = \text{partial correctness} + \text{termination}$$

4. Axiomatic Semantics

Program Correctness: Example

- Consider the factorial statement

```
y := 1;
while not x = 1 do
  y := y * x;
  x := x - 1
end
```

- Specification:
  The final value of `y` is the factorial of the initial value of `x`

- The statement is partially correct
  - It does not terminate for $x < 1$

Formal Specification

- Specification:
  The final value of `y` is the factorial of the initial value of `x`

- We can express the specification formally based on a formal semantics

$$\forall \sigma, \sigma'.$$
$$\vdash \langle \texttt{y := 1;while not x = 1 do y := y * x;x := x - 1 end}, \sigma \rangle \rightarrow \sigma'$$
$$\Rightarrow \quad \sigma'(\texttt{y}) = \sigma(\texttt{x})! \wedge \sigma(\texttt{x}) > 0$$

- This specification expresses partial correctness using big-step semantics
  - We could have used small-step semantics to formulate the property, instead

Correctness Proof

- We prove partial correctness in three steps

- Step 1: The body of the loop satisfies

$$\forall \sigma, \sigma''. \vdash \langle \texttt{y := y*x;x := x-1}, \sigma \rangle \to \sigma'' \wedge \sigma''(\texttt{x}) > 0 \Rightarrow$$
$$\sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma''(\texttt{y}) \times \sigma''(\texttt{x})! \wedge \sigma(\texttt{x}) > 0$$

- Step 2: The loop satisfies

$$\forall \sigma, \sigma''. \vdash \langle \texttt{while not x = 1 do y := y*x;x := x-1 end}, \sigma \rangle \to \sigma'' \Rightarrow$$
$$\sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma''(\texttt{y}) \wedge \sigma''(\texttt{x}) = 1 \wedge \sigma(\texttt{x}) > 0$$

- Step 3: The whole statement is partially correct

$$\forall \sigma, \sigma'. \vdash \langle \texttt{y := 1;while not x = 1 do y := y*x;x := x-1 end}, \sigma \rangle \to \sigma' \Rightarrow$$
$$\sigma'(\texttt{y}) = \sigma(\texttt{x})! \wedge \sigma(\texttt{x}) > 0$$

Proof: Step 1—Loop Body

- Let σ and σ'' be arbitrary. To prove the implication, we assume the left-hand-side and prove the right.

- Since we have $\vdash \langle \texttt{y := y*x;x := x-1}, \sigma \rangle \to \sigma''$, we can assume that both $\vdash \langle \texttt{y := y*x}, \sigma \rangle \to \sigma'$ and $\vdash \langle \texttt{x := x-1}, \sigma' \rangle \to \sigma''$ for some σ' (the last rule applied in the original derivation must be $\textsc{Seq}_{NS}$)

- Since these two derivations must end in the $\textsc{Ass}_{NS}$ rule, we must have $\sigma' = \sigma[\texttt{y} \mapsto \mathcal{A}[\![\texttt{y*x}]\!]\sigma]$ and $\sigma'' = \sigma'[\texttt{x} \mapsto \mathcal{A}[\![\texttt{x-1}]\!]\sigma']$, which together imply that $\sigma'' = \sigma[\texttt{y} \mapsto \sigma(\texttt{y}) \times \sigma(\texttt{x})][\texttt{x} \mapsto \sigma(\texttt{x})\texttt{-1}]$

- By $\sigma''(\texttt{x}) > 0$, we calculate

$$\sigma''(\texttt{y}) \times \sigma''(\texttt{x})! =$$
$$\sigma(\texttt{y}) \times \sigma(\texttt{x}) \times (\sigma(\texttt{x})\texttt{-1})! = \sigma(\texttt{y}) \times \sigma(\texttt{x})!$$

- Finally, by $\sigma''(\texttt{x}) = \sigma(\texttt{x})\texttt{-1}$, we get $\sigma(\texttt{x}) > 0$

Proof: Step 2—Loop

- Step 2: The loop satisfies

$$\forall \sigma, \sigma''. \quad \vdash \langle \texttt{while not x = 1 do y := y*x;x := x-1 end}, \sigma \rangle \to \sigma'' \Rightarrow \\ \sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma''(\texttt{y}) \wedge \sigma''(\texttt{x}) = 1 \wedge \sigma(\texttt{x}) > 0$$

- We prove this property by induction on the shape of the derivation tree:

- Define $P(T)$:

$$P(T) \equiv \forall \sigma, \sigma''. \\ \quad \mathit{root}(T) \equiv \langle \texttt{while not x = 1 do y := y*x;x := x-1 end}, \sigma \rangle \to \sigma'' \Rightarrow \\ \quad \sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma''(\texttt{y}) \wedge \sigma''(\texttt{x}) = 1 \wedge \sigma(\texttt{x}) > 0$$

- We prove $P(T)$ for arbitrary T, with the induction hypothesis $\forall T' \sqsubset T.P(T')$ (i.e., we can assume the property for all subderivations of the derivation tree T).

Proof: Step 2—Loop (Case $\mathbf{WhF}_{NS}$)

- We prove $P(T)$ for arbitrary T, with the induction hypothesis $\forall T' \sqsubset T.P(T')$ (i.e., we can assume the property for all subderivations of the derivation tree T.
- Let σ, σ'' be arbitrary. We assume the property above for $root(T)$, and need to show that $\sigma(\mathtt{y}) \times \sigma(\mathtt{x})! = \sigma''(\mathtt{y}) \wedge \sigma''(\mathtt{x}) = 1 \wedge \sigma(\mathtt{x}) > 0$
- We consider the two possible cases for the last rule applied in T: $\textsc{WhF}_{NS}$ and $\textsc{WhT}_{NS}$
- **Case $\mathbf{WhF}_{NS}$:**
  - From the form of the rule, we must have $\sigma(\mathtt{x}) = 1$ and $\sigma = \sigma''$
  - Since $1 = 1!$, we get $\sigma(\mathtt{y}) \times \sigma(\mathtt{x})! = \sigma(\mathtt{y}) = \sigma''(\mathtt{y})$
  - We also immediately get $\sigma''(\mathtt{x}) = 1$ and $\sigma(\mathtt{x}) > 0$

Proof: Step 2—Loop (Case WhT_{NS})

- From the form of the rule, we know that two subderivations exist: for some σ''',
  (1) $\vdash \langle \texttt{y := y*x;x := x-1}, \sigma \rangle \to \sigma'''$
  (2) $\vdash \langle \texttt{while not x = 1 do y := y*x;x := x-1 end}, \sigma''' \rangle \to \sigma''$

- Applying the induction hypothesis to (2) yields
  $\sigma'''(\texttt{y}) \times \sigma'''(\texttt{x})! = \sigma''(\texttt{y}) \wedge \sigma''(\texttt{x}) = 1 \wedge \sigma'''(\texttt{x}) > 0$

- By (1), $\sigma'''(\texttt{x}) > 0$, and Proof Step 1, we get
  $\sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma'''(\texttt{y}) \times \sigma'''(\texttt{x})! \wedge \sigma(\texttt{x}) > 0$

- Combining these results yields, as required:
  $\sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma''(\texttt{y}) \wedge \sigma''(\texttt{x}) = 1 \wedge \sigma(\texttt{x}) > 0$

Proof: Step 3—Factorial Statement

- Step 3: The whole statement is partially correct:

  $$\forall \sigma, \sigma'. \vdash \langle \texttt{y := 1;while not x = 1 do y := y*x;x := x-1 end}, \sigma \rangle \to \sigma' \Rightarrow \sigma'(\texttt{y}) = \sigma(\texttt{x})! \wedge \sigma(\texttt{x}) > 0$$

- Let σ, σ' be arbitrary. The last rule applied in the assumed derivation must be $\textsc{Seq}_{NS}$.

- From the form of the rule, we get, for some σ'':

  (1) $\vdash \langle \texttt{y := 1}, \sigma \rangle \to \sigma''$

  (2) $\vdash \langle \texttt{while not x = 1 do y := y*x;x := x-1 end}, \sigma'' \rangle \to \sigma'$

- By (1), we get $\sigma'' = \sigma[\texttt{y} \mapsto 1]$ and, thus, $\sigma''(\texttt{x}) = \sigma(\texttt{x})$

- By (2), and Proof Step 2, we get
  $\sigma''(\texttt{y}) \times \sigma''(\texttt{x})! = \sigma'(\texttt{y}) \wedge \sigma'(\texttt{x}) = 1 \wedge \sigma''(\texttt{x}) > 0$

- We conclude, as required: $1 \times \sigma(\texttt{x})! = \sigma'(\texttt{y}) \wedge \sigma(\texttt{x}) > 0$

Verification Example: Observations

- We can prove correctness of a program based on a formal semantics
  - The proof would also be possible with the small-step semantics, but even more complicated
- Proofs are too detailed to be practical
  - We have to consider how whole states are modified
  - We would like to focus on certain properties of states
  - We need to manually decompose the proof into suitable parts
  - For each loop, we need to formulate a separate induction on derivations
- Axiomatic Semantics provides a way of constructing these proofs conveniently
  - Proofs can focus only on essential properties of interest
  - Decomposing the program into smaller parts happens naturally
  - The induction for reasoning about loops is "built" into the semantic rule for loops

4. Axiomatic Semantics

Hoare Triples

- Properties of programs are specified as Hoare triples

$$\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$$

  where s is a statement and **P** and **Q** are assertions (about the state)

- Terminology
  - The assertion **P** is called the precondition of a triple $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
  - The assertion **Q** is called the postcondition of a triple $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
  - Assertions are boolean expressions, with some additional features (explained shortly). We use **P**, **Q**, **R** as meta-variables over assertions.

Meaning of Hoare Triples

- The informal meaning of $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$ is

  If **P** evaluates to true in an initial state σ, and
  if the execution of s from σ terminates in a state σ'
  then **Q** will evaluate to true in σ'

- This meaning describes partial correctness, that is, termination is not an essential property

- It is also possible to assign different meanings to Hoare Triples
  - Total correctness interpretation (see later)

Hoare Triples: Example

- Attempted specification of the factorial statement:

$$\{\ \texttt{true}\ \}$$
$$\texttt{y := 1;while not x = 1 do y := y*x;x := x-1 end}$$
$$\{\ \texttt{y} = \texttt{x}! \wedge \texttt{x} > 0\ \}$$

- In general, this Hoare triple is not correct. For example:
  - Consider an initial state $\sigma_{zero}[\texttt{x} \mapsto 2][\texttt{y} \mapsto 0]$
  - The final state will be $\sigma_{zero}[\texttt{x} \mapsto 1][\texttt{y} \mapsto 2]$

- We need to express that y in the final state is the factorial of x in the initial state
  - We need a way for assertions to describe properties not just of the current state, but also of the initial state

Logical Variables

- We allow assertions to contain logical variables
  - Logical variables may occur only in assertions
  - Logical variables are not program variables and may, thus, not be accessed in programs; in particular, they are never assigned to
- Logical variables can be used to "save" values in the initial state, so that they can be referred to later

$\{ \texttt{x} = N \}$
```
  y := 1;while not x = 1 do y := y*x;x := x-1 end
```
$\{ \texttt{y} = N! \wedge N > 0 \}$

- States map logical variables (and program variables) to their values

Assertion Language

- Pre- and postconditions are assertions, that is, boolean expressions plus logical variables
  - In particular, we will use program boolean expressions b as assertions
  - It is common in practice to use a richer set of expressions for assertions, for instance, to include quantification
  - We will use additional expressions when it is convenient (e.g., `x!`)
  - We assume that the substitution lemma from the exercises still holds when we use an extended assertion language:

$$\mathcal{B}[\![\mathbf{P}[x \mapsto e]]\!]\sigma = \mathcal{B}[\![\mathbf{P}]\!]\sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$$

- We will use the following convenient notations
  - "$P_1 \wedge P_2$" for "P_1 `and` P_2"
  - "$P_1 \vee P_2$" for "P_1 `or` P_2"
  - "$\neg P$" for "`not` P"

4. Axiomatic Semantics

Axiomatic Semantics: Derivation System

- We formalize an axiomatic semantics of IMP by describing the valid Hoare triples

- This is done by a derivation system
  - The derivation rules specify which triples can be derived for each statement
  - The premises and conclusions of the derivation rules are Hoare triples

$$\{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\}$$

- Derivation trees (using the rules presented next) are defined as before

- Similarly to the other derivation systems we have studied, we write $\vdash \{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\}$ if and only if there exists a (finite) derivation tree ending in $\{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\}$
  - $\vdash \{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\} \Leftrightarrow \exists T.\mathit{root}(T) \equiv \{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\}$

Axiomatic Semantics of IMP

- `skip` does not modify the state

$$\frac{}{\{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{P}\ \}}\ (\textsc{Skip}_{Ax})$$

- x := e assigns the value of e to variable x

$$\frac{}{\{\ \mathbf{P}[x \mapsto e]\ \}\ x\ :=\ e\ \{\ \mathbf{P}\ \}}\ (\textsc{Ass}_{Ax})$$

  - Let σ be the initial state
  - Precondition: $\mathcal{B}[\![\mathbf{P}[x \mapsto e]]\!]\sigma$, which is equivalent to $\mathcal{B}[\![\mathbf{P}]\!]\sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$ (substitution lemma)
  - Final state: $\sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$
  - Consequently, $\mathcal{B}[\![\mathbf{P}]\!]$ holds in the final state

- Reminder: these are rule schemes (they can be instantiated by replacing meta-variables)

Axiomatic Semantics of IMP (cont'd)

- Sequential composition $s;s'$

$$\frac{\{\,\mathbf{P}\,\}\,s\,\{\,\mathbf{Q}\,\} \quad \{\,\mathbf{Q}\,\}\,s'\,\{\,\mathbf{R}\,\}}{\{\,\mathbf{P}\,\}\,s;s'\,\{\,\mathbf{R}\,\}}\,(\textsc{Seq}_{Ax})$$

- Conditional statement `if` b `then` s_1 `else` s_2 `end`

$$\frac{\{\,b \wedge \mathbf{P}\,\}\,s\,\{\,\mathbf{Q}\,\} \quad \{\,\neg b \wedge \mathbf{P}\,\}\,s'\,\{\,\mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\,\texttt{if}\ b\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{end}\,\{\,\mathbf{Q}\,\}}\,(\textsc{If}_{Ax})$$

- Loop statement `while` b `do` s `end`

$$\frac{\{\,b \wedge \mathbf{P}\,\}\,s\,\{\,\mathbf{P}\,\}}{\{\,\mathbf{P}\,\}\,\texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\,\{\,\neg b \wedge \mathbf{P}\,\}}\,(\textsc{Wh}_{Ax})$$

  - The assertion **P** is the loop invariant

Axiomatic Semantics of IMP (cont'd)

- The rules so far manipulate assertions syntactically
  - For example, so far, we cannot derive the triple
    $\{ \texttt{x} = 4 \wedge \texttt{y} = 5 \}\ \texttt{skip}\ \{ \texttt{y} = 5 \wedge \texttt{x} = 4 \}$
    (assertions are not identical, so this is not an instance of the $\textsc{Skip}_{Ax}$ rule)
  - During proofs, we often need to perform semantic reasoning (e.g., applying mathematical properties of factorial, arithmetic rules, etc.)

Axiomatic Semantics of IMP (cont'd)

- The rules so far manipulate assertions syntactically
  - For example, so far, we cannot derive the triple
    $\{ \mathtt{x} = 4 \wedge \mathtt{y} = 5 \}\ \mathtt{skip}\ \{ \mathtt{y} = 5 \wedge \mathtt{x} = 4 \}$
    (assertions are not identical, so this is not an instance of the $\textsc{Skip}_{Ax}$ rule)
  - During proofs, we often need to perform semantic reasoning (e.g., applying mathematical properties of factorial, arithmetic rules, etc.)

- Semantic entailment expresses these reasoning steps:
  we write $\mathbf{P} \models \mathbf{Q}$ iff "for all states σ, $\mathcal{B}[\![P]\!]\sigma = tt$ implies $\mathcal{B}[\![Q]\!]\sigma = tt$"

Axiomatic Semantics of IMP (cont'd)

- The rules so far manipulate assertions syntactically
  - For example, so far, we cannot derive the triple
    $\{\ \mathtt{x} = 4 \wedge \mathtt{y} = 5\ \}\ \mathtt{skip}\ \{\ \mathtt{y} = 5 \wedge \mathtt{x} = 4\ \}$
    (assertions are not identical, so this is not an instance of the $\textsc{Skip}_{Ax}$ rule)
  - During proofs, we often need to perform semantic reasoning (e.g., applying mathematical properties of factorial, arithmetic rules, etc.)

- Semantic entailment expresses these reasoning steps:
  we write $\mathbf{P} \vDash \mathbf{Q}$ iff "for all states σ, $\mathcal{B}[\![P]\!]\sigma = tt$ implies $\mathcal{B}[\![Q]\!]\sigma = tt$"

- The rule of consequence allows semantic entailments in derivations

$$\frac{\{\ \mathbf{P'}\ \}\ s\ \{\ \mathbf{Q'}\ \}}{\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}}\ (\textsc{Cons}_{Ax}) \qquad \textit{if } \mathbf{P} \vDash \mathbf{P'} \textit{ and } \mathbf{Q'} \vDash \mathbf{Q}$$

  - We can strengthen preconditions ($\mathbf{P}$ cannot be weaker than $\mathbf{P'}$)
  - We can weaken postconditions ($\mathbf{Q}$ cannot be stronger than $\mathbf{Q'}$)

Derivation Trees: Example 1

- Prove that the following statement swaps the values in the variables `x` and `y`

```
(z:=x; x:=y); y:=z
```

- We can build the following derivation tree

$$\dfrac{\dfrac{\dfrac{}{\{\ \mathbf{P}\ \}\ \texttt{z:=x}\ \{\ \texttt{z} = X_0 \wedge \texttt{y} = Y_0\ \}}\ (\textsc{Ass}_{Ax})}{\{\ \mathbf{P}\ \}\ \texttt{z:=x}\ \{\ \texttt{y} = Y_0 \wedge \texttt{z} = X_0\ \}}\ (\textsc{Cons}_{Ax})^{*} \qquad \dfrac{}{\{\ \texttt{y} = Y_0 \wedge \texttt{z} = X_0\ \}\ \texttt{x:=y}\ \{\ \mathbf{Q'}\ \}}\ (\textsc{Ass}_{Ax})}{\{\ \mathbf{P}\ \}\ \texttt{z:=x; x:=y}\ \{\ \mathbf{Q'}\ \}}\ (\textsc{Seq}_{Ax}) \qquad \dfrac{}{\{\ \mathbf{Q'}\ \}\ \texttt{y:=z}\ \{\ \mathbf{Q}\ \}}\ (\textsc{Ass}_{Ax})}{\{\ \mathbf{P}\ \}\ \texttt{(z:=x; x:=y); y:=z}\ \{\ \mathbf{Q}\ \}}\ (\textsc{Seq}_{Ax})$$

* *since* $\mathbf{P} \vDash \mathbf{P}$ *and* $\texttt{z} = X_0 \wedge \texttt{y} = Y_0 \vDash \texttt{y} = Y_0 \wedge \texttt{z} = X_0$

where we write:

- $\mathbf{P}$ for $\texttt{x} = X_0 \wedge \texttt{y} = Y_0$
- $\mathbf{Q}$ for $\texttt{x} = Y_0 \wedge \texttt{y} = X_0$
- $\mathbf{Q'}$ for $\texttt{x} = Y_0 \wedge \texttt{z} = X_0$

Derivation Trees: Example 2

- Consider the non-terminating loop

```
while true do skip end
```

- We can build the following derivation tree

$$\dfrac{\dfrac{\dfrac{}{\{\ \texttt{true}\ \}\ \texttt{skip}\ \{\ \texttt{true}\ \}}\ (\textsc{Skip}_{Ax})}{\{\ \texttt{true} \wedge \texttt{true}\ \}\ \texttt{skip}\ \{\ \texttt{true}\ \}}\ (\textsc{Cons}_{Ax})^1}{\{\ \texttt{true}\ \}\ \texttt{while true do skip end}\ \{\ \neg\texttt{true} \wedge \texttt{true}\ \}}\ (\textsc{Wh}_{Ax})}{\{\ \texttt{true}\ \}\ \texttt{while true do skip end}\ \{\ \neg\texttt{true}\ \}}\ (\textsc{Cons}_{Ax})^2$$

[1] $\texttt{true} \wedge \texttt{true} \vDash \texttt{true}$
[2] $\neg\texttt{true} \wedge \texttt{true} \vDash \neg\texttt{true}$

- This proof illustrates that we have partial correctness

Proof Outlines

- Derivation trees tend to get very large and are, thus, inconvenient to write
  - Most statements are written many times
  - Many assertions are written many times

- An alternative is to group the assertions around the program text

- We write assertions before and after each statement to indicate which properties hold in the states before and after the execution of this statement

Proof Outlines: Notation

- We write instances of the $\textsc{Skip}_{Ax}$ and $\textsc{Ass}_{Ax}$ rules as:

$$\{\ \mathbf{P}\ \}\quad \texttt{skip}\quad \{\ \mathbf{P}\ \}$$

$$\{\ \mathbf{P}[x \mapsto e]\ \}\quad x := e\quad \{\ \mathbf{P}\ \}$$

- We write an instance of the rule for sequential composition as:

$$\{\ \mathbf{P}\ \}\quad s_1\,;\quad \{\ \mathbf{Q}\ \}\quad s_2\quad \{\ \mathbf{R}\ \}$$

  - This expresses $\vdash \{\ \mathbf{P}\ \}\ s_1\ \{\ \mathbf{Q}\ \}$, $\vdash \{\ \mathbf{Q}\ \}\ s_2\ \{\ \mathbf{R}\ \}$, and $\vdash \{\ \mathbf{P}\ \}\ s_1\,;s_2\ \{\ \mathbf{R}\ \}$
  - Note: we write each statement and the intermediate assertion $\mathbf{Q}$ once

Proof Outlines: Notation (cont'd)

- We write an instance of the rule for conditional statements as:

$$
\begin{array}{l}
\{\ \mathbf{P}\ \} \\
\quad \texttt{if } b \texttt{ then} \\
\qquad \{\ b \wedge \mathbf{P}\ \} \\
\qquad\quad s_1 \\
\qquad \{\ \mathbf{Q}\ \} \\
\quad \texttt{else} \\
\qquad \{\ \neg b \wedge \mathbf{P}\ \} \\
\qquad\quad s_2 \\
\qquad \{\ \mathbf{Q}\ \} \\
\quad \texttt{end} \\
\{\ \mathbf{Q}\ \}
\end{array}
$$

- We write an instance of the rule for loops as:

$$
\begin{array}{l}
\{\ \mathbf{P}\ \} \\
\quad \texttt{while } b \texttt{ do} \\
\qquad \{\ b \wedge \mathbf{P}\ \} \\
\qquad\quad s \\
\qquad \{\ \mathbf{P}\ \} \\
\quad \texttt{end} \\
\{\ \neg b \wedge \mathbf{P}\ \}
\end{array}
$$

Proof Outlines: Notation (cont'd)

- We write an instance of the rule of consequence as:

$$
\begin{array}{l}
\{\, \mathbf{P} \,\} \\
\models \\
\{\, \mathbf{P}' \,\} \\
\quad s \\
\{\, \mathbf{Q}' \,\} \\
\models \\
\{\, \mathbf{Q} \,\}
\end{array}
$$

  - We omit the entailment step when $\mathbf{P}$ and $\mathbf{P}'$ or $\mathbf{Q}$ and $\mathbf{Q}'$ are syntactically identical. For example, we may also write:

$$
\begin{array}{l}
\{\, \mathbf{P} \,\} \\
\quad s \\
\{\, \mathbf{Q}' \,\} \\
\models \\
\{\, \mathbf{Q} \,\}
\end{array}
\qquad\qquad
\begin{array}{l}
\{\, \mathbf{P} \,\} \\
\models \\
\{\, \mathbf{P}' \,\} \\
\quad s \\
\{\, \mathbf{Q} \,\}
\end{array}
$$

Proof Outlines: Example

- Back to our swap-example:

```
(z:=x; x:=y); y:=z
```

- Proof outline:

$$\{\ \mathtt{x} = X_0 \wedge \mathtt{y} = Y_0\ \}$$
$$\models$$
$$\{\ \mathtt{y} = Y_0 \wedge \mathtt{x} = X_0\ \}$$
```
z := x;
```
$$\{\ \mathtt{y} = Y_0 \wedge \mathtt{z} = X_0\ \}$$
```
x := y;
```
$$\{\ \mathtt{x} = Y_0 \wedge \mathtt{z} = X_0\ \}$$
```
y := z
```
$$\{\ \mathtt{x} = Y_0 \wedge \mathtt{y} = X_0\ \}$$

- Proof outlines are often best developed bottom-up

Verification of Factorial Statement

$\{ \texttt{x} = N \}$

```
  y := 1;while not x = 1 do y := y*x;x := x-1 end
```

$\{ \texttt{y} = N! \wedge N > 0 \}$

Verification of Factorial Statement

$\{ \texttt{x} = N \}$
```
  y := 1;while not x = 1 do y := y*x;x := x-1 end
```
$\{ \texttt{y} = N! \wedge N > 0 \}$

- Determining the loop invariant

Iteration	0	1	2	i	N–1
x	N	N–1	N–2	N–i	1
y	1	N	$N*(N$–1)	$N*(N$–1)$*\ldots*(N$–$i+1)$	$N!$

Verification of Factorial Statement

$\{ \mathtt{x} = N \}$
`y := 1;while not x = 1 do y := y*x;x := x-1 end`
$\{ \mathtt{y} = N! \wedge N > 0 \}$

- Determining the loop invariant

Iteration	0	1	2	i	N–1
x	N	N–1	N–2	N–i	1
y	1	N	$N*(N$–1)	$N*(N$–1)$*\ldots*(N$–$i+1)$	$N!$

- Invariant: $\mathtt{x} > 0 \Rightarrow \mathtt{y}*\mathtt{x}! = N! \wedge N \geq \mathtt{x}$

Proof Outline for Factorial Statement

$\{\ \mathtt{x} = N\ \}$

$\models$

$\{\ \mathtt{x} > 0 \Rightarrow 1*\mathtt{x}! = N! \wedge N \geq \mathtt{x}\ \}$

`y := 1;`

$\{\ \mathtt{x} > 0 \Rightarrow \mathtt{y}*\mathtt{x}! = N! \wedge N \geq \mathtt{x}\ \}$

`while not x = 1 do`

$\{\ \mathtt{x} \neq 1 \wedge (\mathtt{x} > 0 \Rightarrow \mathtt{y}*\mathtt{x}! = N! \wedge N \geq \mathtt{x})\ \}$

$\models$

$\{\ \mathtt{x}-1 > 0 \Rightarrow \mathtt{y}*\mathtt{x}*(\mathtt{x}-1)! = N! \wedge N \geq \mathtt{x}-1\ \}$

`y := y*x;`

$\{\ \mathtt{x}-1 > 0 \Rightarrow \mathtt{y}*(\mathtt{x}-1)! = N! \wedge N \geq \mathtt{x}-1\ \}$

`x := x-1`

$\{\ \mathtt{x} > 0 \Rightarrow \mathtt{y}*\mathtt{x}! = N! \wedge N \geq \mathtt{x}\ \}$

`end`

$\{\ \mathtt{x} = 1 \wedge (\mathtt{x} > 0 \Rightarrow \mathtt{y}*\mathtt{x}! = N! \wedge N \geq \mathtt{x})\ \}$

$\models$

$\{\ \mathtt{y} = N! \wedge N > 0\ \}$

4. Axiomatic Semantics

Total Correctness

- We introduce an alternative form of Hoare triple $\{ \mathbf{P} \}\ s\ \{ \Downarrow \mathbf{Q} \}$
- The informal meaning of $\{ \mathbf{P} \}\ s\ \{ \Downarrow \mathbf{Q} \}$ is

  If **P** evaluates to true in the initial state σ
  then the execution of s from σ terminates
  and **Q** will evaluate to true in the final state

- This meaning describes total correctness, that is, termination is required
- We do not mix these triples with those of partial correctness; the two form two separate axiomatic semantics (and corresponding derivation systems)
- However, all total correctness derivation rules are analogous to those for partial correctness, except for the rule for loops

Loop Variants

- Termination is proved using loop variants
- A loop variant is an expression that evaluates to a value in a well-founded set (for instance, $\mathbb{N}$) before each iteration
- Each loop iteration must decrease the value of the loop variant
- The loop has to terminate when a minimal value of the well-founded set is reached (or earlier than this)
- For example:

```
x := 5;
while x # 0 do x := x - 1 end
```

  - `x` is a possible loop variant for this loop

While Rule for Total Correctness

- For simplicity, we consider loop variants that evaluate to values in $\mathbb{N}$
  - We use arithmetic expressions e of IMP to represent loop variants
  - We prove explicitly that the value of e will be non-negative before each loop iteration
  - Intuition: a correct loop variant provides an upper bound on the number of loop iterations
- Total correctness derivation rule for loops:

$$\frac{\{\ b \wedge \mathbf{P} \wedge e = Z\ \}\ s\ \{\ \Downarrow \mathbf{P} \wedge e < Z\ \}}{\{\ \mathbf{P}\ \}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\ \Downarrow \neg b \wedge \mathbf{P}\ \}}\ (\textsc{WhTot}_{Ax})\ \text{if}\ b \wedge \mathbf{P} \vDash 0 \leq e$$

  where Z is a fresh logical variable (not used in $\mathbf{P}$)

- Note: in practice, other well-founded sets and orderings can be useful

Total Correctness of Factorial

$$\{\, \mathtt{x} = N \ \wedge \mathtt{x} > 0 \,\}$$
```
  y := 1;while not x = 1 do y := y*x;x := x-1 end
```
$$\{\, \Downarrow \mathtt{y} = N! \,\}$$

- Invariant: $\mathbf{P} \equiv \mathtt{x} > 0 \wedge N > 0 \ \wedge \ \mathtt{y}*\mathtt{x}! = N!$
- Variant: $\mathtt{x}$
- In the proof outline (on the next slide) we explicitly note the side-condition for the while-rule, concerning the loop variant

Proof Outline for Factorial Statement

$\{\ \mathtt{x} = N \wedge \mathtt{x} > 0\ \}$

$\models$

$\{\ \mathtt{x} > 0 \wedge N > 0 \wedge 1*\mathtt{x}! = N!\ \}$

`y := 1;`

$\{\ \mathtt{x} > 0 \wedge N > 0 \wedge \mathtt{y}*\mathtt{x}! = N!\ \}$

`while not x = 1 do`*

$\{\ \mathtt{x} \neq 1 \wedge \mathtt{x} > 0 \wedge N > 0 \wedge \mathtt{y}*\mathtt{x}! = N! \wedge \mathtt{x} = Z\ \}$

$\models$

$\{\ \mathtt{x}{-}1 > 0 \wedge N > 0 \wedge (\mathtt{y}*\mathtt{x})*(\mathtt{x}{-}1)! = N! \wedge \mathtt{x}{-}1 < Z\ \}$

`y := y*x;`

$\{\ \mathtt{x}{-}1 > 0 \wedge N > 0 \wedge \mathtt{y}*(\mathtt{x}{-}1)! = N! \wedge \mathtt{x}{-}1 < Z\ \}$

`x := x-1`

$\{\ \Downarrow \mathtt{x} > 0 \wedge N > 0 \wedge \mathtt{y}*\mathtt{x}! = N! \wedge \mathtt{x} < Z\ \}$

`end`

$\{\ \Downarrow \neg\neg \mathtt{x} = 1 \wedge \mathtt{x} > 0 \wedge N > 0 \wedge \mathtt{y}*\mathtt{x}! = N!\ \}$

$\models$

$\{\ \Downarrow \mathtt{y} = N!\ \}$

*$\mathtt{x} \neq 1 \wedge \mathtt{x} > 0 \wedge N > 0 \wedge \mathtt{y}*\mathtt{x}! = N! \ \models\ 0 \leq \mathtt{x}$

4. Axiomatic Semantics

Motivation

- Developing an axiomatic semantics is difficult
- Soundness:
  If a property can be proved then it does indeed hold
  - An unsound derivation system is useless
- Completeness:
  If a property does hold then it can be proved
  - With an incomplete derivation system, a program might be correct, but we cannot prove it

Unsoundness: Example

$$\frac{\{\ b \wedge \mathbf{P} \wedge e = Z\ \}\ s\ \{\ \Downarrow \mathbf{P} \wedge e < Z\ \}}{\{\ \mathbf{P} \wedge 0 \leq e\ \}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\ \Downarrow \neg b \wedge \mathbf{P}\ \}}\ (\textsc{WhU}_{Ax})$$

- With $e \equiv \texttt{x}$, we can derive:

$$\dfrac{\dfrac{\dfrac{}{\{\ \texttt{true} \wedge \texttt{x-1} < Z\ \}\ \texttt{x := x-1}\ \{\ \Downarrow \texttt{true} \wedge \texttt{x} < Z\ \}}\ (\textsc{Ass}_{Ax})}{\{\ \texttt{true} \wedge \texttt{true} \wedge \texttt{x} = Z\ \}\ \texttt{x := x-1}\ \{\ \Downarrow \texttt{true} \wedge \texttt{x} < Z\ \}}\ (\textsc{Cons}_{Ax})}{\{\ \texttt{true} \wedge 0 \leq \texttt{x}\ \}\ \texttt{while true do x := x-1 end}\ \{\ \Downarrow \neg\texttt{true} \wedge \texttt{true}\ \}}\ (\textsc{WhU}_{Ax})}{\{\ 0 \leq \texttt{x}\ \}\ \texttt{while true do x := x-1 end}\ \{\ \Downarrow \texttt{true}\ \}}\ (\textsc{Cons}_{Ax})$$

- This derivation is not sound (the derived triple does not hold)

- The rule does not ensure that the loop variant is non-negative before each loop iteration

Incompleteness: Example

$$\frac{\{\, b \wedge \mathbf{P} \wedge e = Z \,\}\ s\ \{\, \Downarrow \mathbf{P} \wedge e < Z \,\}}{\{\, \mathbf{P} \,\}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\, \Downarrow \neg b \wedge \mathbf{P} \,\}}\ (\textsc{WhI}_{Ax}) \quad \text{if } \mathbf{P} \vDash 0 \leq e$$

- With this rule, we cannot prove termination for the following loop

```
while 0 < x do
  x := x - 1
end
```

  - The strongest possible loop invariant is `true` (because we want to show termination for all initial states)
  - For the natural loop variant, `x`, the loop invariant is not strong enough to show the side condition
  - The loop variant required to prove termination is $|\,\texttt{x}\,|$, which cannot be expressed as IMP expression
  - Other examples require lexicographic ordering (e.g., Ackermann function)

Soundness and Completeness

- Soundness and completeness can be proved w.r.t. an operational semantics (here, big-step semantics)

  The partial correctness triple $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$ is valid, written as $\models \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$, iff:

  $$\forall \sigma, \sigma'.\ \mathcal{B}[\![P]\!]\sigma = tt \wedge \vdash \langle s, \sigma\rangle \to \sigma' \Rightarrow \mathcal{B}[\![Q]\!]\sigma' = tt$$

  - This is the intuitive interpretation of triples: $\models \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$ is true if, whenever we start execution of s from a state in which **P** holds, if the execution terminates, then **Q** will hold in the final state
  - Conversely, recall that $\vdash \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$ is defined purely in terms of the derivation rules of the axiomatic semantics

- Soundness: $\vdash \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \} \Rightarrow \models \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$
- Completeness: $\models \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \} \Rightarrow \vdash \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$

Theorem

Soundness and Completeness Theorem (Partial Correctness):

For all partial correctness triples $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$ of IMP we have

$$\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \} \Leftrightarrow \models \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$$

Summary: Axiomatic Semantics

- Axiomatic semantics
  - is concerned with specific properties of the effect of executing programs
  - allows for succinct proofs about program properties
- Axiomatic semantics is used to verify programs
  - Partial correctness
  - Total correctness
  - Other properties, e.g., resource consumption
- The derivation system for partial correctness of IMP programs is sound and complete