Formal Methods and Functional Programming

Axiomatic Semantics

Peter Müller

Chair of Programming Methodology
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

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

- Step 2: The loop satisfies

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

- Step 3: The whole statement is partially correct

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

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 $P(T)$:

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

- We prove $P(T)$ 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 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 equality 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: WhF_{NS} and WhT_{NS}
- **Case 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:

$$\{\ \texttt{true}\ \}$$
```
  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 $\{\ \texttt{x} \mapsto 2, \texttt{y} \mapsto 0\ \}$
  - The final state will be $\{\ \texttt{x} \mapsto 1, \texttt{y} \mapsto 2\ \}$

- 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 (pre- and postconditions)
  - 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 $\{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\}$
  - $\vdash \{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\} \Leftrightarrow \exists T.\mathit{root}(T) = \{\, \mathbf{P} \,\}\ s\ \{\, \mathbf{Q} \,\}$

Axiomatic Semantics of IMP

- `skip` does not modify the state

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

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

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

  - 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 $s\,;s'$

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

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

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

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

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

  - The assertion $\underline{\mathbf{P}}$ 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
    $\{ \mathtt{x} = 4 \wedge \mathtt{y} = 5 \}\ \mathtt{skip}\ \{ \mathtt{y} = 5 \wedge \mathtt{x} = 4 \}$
    (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.)

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, $\mathcal{B}[\![\mathbf{P}]\!]\sigma = tt$ implies $\mathcal{B}[\![\mathbf{Q}]\!]\sigma = tt$"

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, $\mathcal{B}[\![\mathbf{P}]\!]\sigma = tt$ implies $\mathcal{B}[\![\mathbf{Q}]\!]\sigma = tt$"

- The rule of consequence allows semantic entailments in derivations

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

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

Derivation Trees: Example 1

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

$$\texttt{(z:=x; x:=y); y:=z}$$

- 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

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

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

- 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:

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

- 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

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

- 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

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

- 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)$*\ldots*(N$–$i+1)$	$N!$

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

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`

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

$\models$

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

Verification of Zune Example

```
{ ... }
  year := 1980;
  days := D;
  while     L(year) and 366 < days or
        not L(year) and 365 < days do
    if L(year) then days := days - 366
    else            days := days - 365
    end
    year := year + 1
  end
{ year-1980 ≤ D/365 }
```

Verification of Zune Example

```
{ ... }
  year := 1980;
  days := D;
  while     L(year) and 366 < days or
        not L(year) and 365 < days do
    if L(year) then days := days - 366
    else            days := days - 365
    end
    year := year + 1
  end
{ year-1980 ≤ D/365 }
```

- Determining the loop invariant: After i iterations, we have
  - year $= 1980 + i$
  - days $\leq$ D$-i \times 365$

Verification of Zune Example

```
{ ... }
  year := 1980;
  days := D;
  while     L(year) and 366 < days or
        not L(year) and 365 < days do
    if L(year) then days := days - 366
    else            days := days - 365
    end
    year := year + 1
  end
{ year-1980 ≤ D/365 }
```

- Determining the loop invariant: After i iterations, we have
  - $\texttt{year} = 1980 + i$
  - $\texttt{days} \leq \texttt{D} - i \times 365$
- Invariant: $\texttt{year}-1980 \leq (D-\texttt{days})/365 \wedge 0 \leq \texttt{days}$

Proof Outline for Zune Example

$\{\ 0 \le D\ \}$
$\models$
$\{\ 1980\text{-}1980 \le (D\text{-}D)/365 \land 0 \le D\ \}$
`year := 1980;`
$\{\ \texttt{year}\text{-}1980 \le (D\text{-}D)/365 \land 0 \le D\ \}$
`days := D;`
$\{\ \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
`while L(year) and 366 < days or not L(year) and 365 < days do`
$\{\ (L(\texttt{year}) \land 366 < \texttt{days} \lor \neg L(\texttt{year}) \land 365 < \texttt{days}) \land \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
$\models$
$\{\ \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land (L(\texttt{year}) \Rightarrow 366 < \texttt{days}) \land 365 < \texttt{days}\ \}$
`if L(year) then`
$\{\ L(\texttt{year}) \land \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land (L(\texttt{year}) \Rightarrow 366 < \texttt{days}) \land 365 < \texttt{days}\ \}$
$\models$
$\{\ \texttt{year} + 1\text{-}1980 \le (D\text{-}(\texttt{days}\text{-}366))/365 \land 0 \le (\texttt{days}\text{-}366)\ \}$
`days := days-366`
$\{\ \texttt{year} + 1\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
`else`
$\{\ \neg L(\texttt{year}) \land \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land (L(\texttt{year}) \Rightarrow 366 < \texttt{days}) \land 365 < \texttt{days}\ \}$
$\models$
$\{\ \texttt{year} + 1\text{-}1980 \le (D\text{-}(\texttt{days}\text{-}365))/365 \land 0 \le \texttt{days}\text{-}365\ \}$
`days := days-365;`
$\{\ \texttt{year} + 1\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
`end;`
$\{\ \texttt{year} + 1\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
`year := year + 1`
$\{\ \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
`end`
$\{\ \neg(L(\texttt{year}) \land 366 < \texttt{days} \lor \neg L(\texttt{year}) \land 365 < \texttt{days}) \land \texttt{year}\text{-}1980 \le (D\text{-}\texttt{days})/365 \land 0 \le \texttt{days}\ \}$
$\models$
$\{\ \texttt{year}\text{-}1980 \le D/365\ \}$

4. Axiomatic Semantics

Induction on the Shape of Derivation Trees

- Properties of the axiomatic semantics are typically proved by induction on the shape of the derivation tree
  - Same technique as used earlier with big-step semantics
  - Note: structural induction on IMP statements is often insufficient for reasoning about derivations, because of the rule of consequence

- Reminder: induction on the shape of derivation trees
  To prove a property $P(T)$ for all derivation trees T, prove that $P(T)$ holds for an arbitrary derivation tree T under the assumption (I.H.) that $P(T')$ holds for all sub-derivations T' of T

- Proofs by induction on the shape of derivation trees typically proceed by case distinction on the rule applied at the root of the arbitrary derivation tree T. In each case, one may assume that:
  - the condition of the rule is satisfied
  - there is a derivation tree T' for each premise of the last-rule in T
  - for each of these, $P(T')$ holds, since T' is a proper sub-tree of T

Proving Properties: Example

- We prove the following lemma for `skip` statements:

$$\forall \mathbf{P}, \mathbf{Q}.\ \vdash \{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}\ \Rightarrow\ \mathbf{P} \vDash \mathbf{Q}$$

  - If there exists an derivation tree for $\{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}$ then $\mathbf{P} \vDash \mathbf{Q}$

- We perform induction on the shape of the derivation tree for $\{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}$

- Thus, we need to prove $P(T)$ for some arbitrary T, with the induction hypothesis $\forall\, T' \sqsubset T.P(T')$, where:

$$P(T) \equiv \forall \mathbf{P}, \mathbf{Q}.\ \mathit{root}(T) = \{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}\ \Rightarrow\ \mathbf{P} \vDash \mathbf{Q}$$

- To prove $P(T)$, let $\mathbf{P}, \mathbf{Q}$ be arbitrary. We assume $\mathit{root}(T) = \{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}$ and need to show that $\mathbf{P} \vDash \mathbf{Q}$

- We consider the cases for the last rule applied in T; given that $\mathit{root}(T) = \{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}$, there are two cases: $\textsc{Skip}_{Ax}$ and $\textsc{Cons}_{Ax}$

Proving Properties: Example (cont'd)

- Case $\textsc{Skip}_{Ax}$:
  - From the form of the rule, we get $\mathbf{P} \equiv \mathbf{Q}$ and, thus, $\mathbf{P} \models \mathbf{Q}$ trivially (recall the definition of semantic entailment)

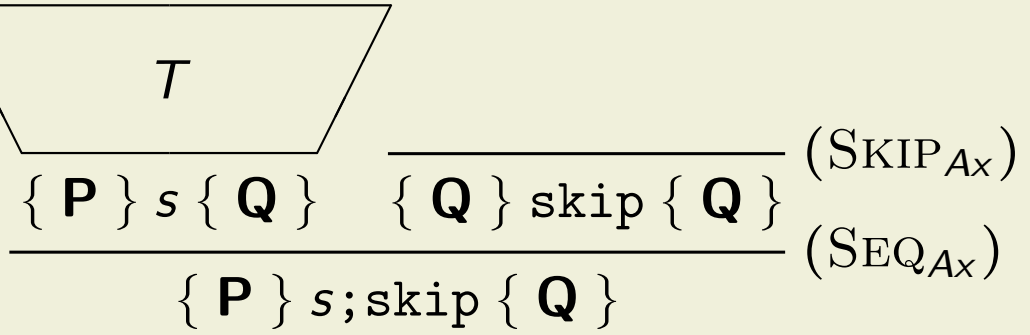
- Case $\textsc{Cons}_{Ax}$:
  - From the form of the rule, we know that, for some $\mathbf{P}'$ and $\mathbf{Q}'$:
    - There exists $T' \sqsubset T$ such that $\mathit{root}(T') = \{\ \mathbf{P}'\ \}\ \texttt{skip}\ \{\ \mathbf{Q}'\ \}$
    - $\mathbf{P} \models \mathbf{P}'$ (from the side-condition)
    - $\mathbf{Q}' \models \mathbf{Q}$ (from the side-condition)
  - Our induction hypothesis allows us to assume $P(T')$, and thus, we deduce $\mathbf{P}' \models \mathbf{Q}'$
  - Now we have $\mathbf{P} \models \mathbf{P}'$, $\mathbf{P}' \models \mathbf{Q}'$, and $\mathbf{Q}' \models \mathbf{Q}$ and, thus, $\mathbf{P} \models \mathbf{Q}$

Semantic Equivalence

Two statements s_1 and s_2 are provably equivalent if:

$$\forall \mathbf{P}, \mathbf{Q}.\ \vdash \{\ \mathbf{P}\ \}\ s_1\ \{\ \mathbf{Q}\ \} \ \Leftrightarrow\ \vdash \{\ \mathbf{P}\ \}\ s_2\ \{\ \mathbf{Q}\ \}$$

- Example: s and $s\texttt{;skip}$ are equivalent, for all statements s
- Let s be arbitrary. We prove the "⇒" direction (for arbitrary $\mathbf{P}, \mathbf{Q}$):
  - We assume that there is a derivation tree T with $\mathit{root}(T) = \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$, and need to show $\vdash \{\ \mathbf{P}\ \}\ s\texttt{;skip}\ \{\ \mathbf{Q}\ \}$
  - We extend that tree using the rule for `skip` and the rule for sequential composition:



Semantic Equivalence: Proof for "⇐" direction

- Note: to construct the required derivation tree for s, we decompose the assumed one for $s\,;\texttt{skip}$. This suggests a proof by induction on the structure of this derivation tree.

- We define
  $P(T) \;\equiv\; \forall \mathbf{P}, \mathbf{Q}.\ \mathit{root}(T) = \{\ \mathbf{P}\ \}\ s\,;\texttt{skip}\ \{\ \mathbf{Q}\ \} \Rightarrow \vdash \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$
- We prove $\forall T.P(T)$ by induction on the shape of the derivation tree; so we need to show $P(T)$ for arbitrary T, with the I.H. $\forall T' \sqsubset T.P(T')$

- Let $\mathbf{P}, \mathbf{Q}$ be arbitrary. We assume $\mathit{root}(T) = \{\ \mathbf{P}\ \}\ s\,;\texttt{skip}\ \{\ \mathbf{Q}\ \}$ and need to show $\vdash \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$

- As usual, we consider the cases for the last rule applied in the derivation tree T:
  - Only two rules may produce $\{\ \mathbf{P}\ \}\ s\,;\texttt{skip}\ \{\ \mathbf{Q}\ \}$: $\textsc{Seq}_{Ax}$ and $\textsc{Cons}_{Ax}$

Semantic Equivalence: Proof for "⇐" direction

- Case $\textsc{Seq}_{Ax}$
  - We know there are derivation trees for the premises; i.e., we have $\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{R} \}$ and $\vdash \{ \mathbf{R} \}\ \texttt{skip}\ \{ \mathbf{Q} \}$ for some assertion $\mathbf{R}$
  - Applying the lemma from slide 193 to $\vdash \{ \mathbf{R} \}\ \texttt{skip}\ \{ \mathbf{Q} \}$ lets us deduce $\mathbf{R} \models \mathbf{Q}$
  - We extend the derivation tree for $\{ \mathbf{P} \}\ s\ \{ \mathbf{R} \}$ using $\textsc{Cons}_{Ax}$ to obtain $\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- Case $\textsc{Cons}_{Ax}$
  - From the form of the rule, we know that there exists a derivation tree $T' \sqsubset T$ with $\mathit{root}(T') = \{ \mathbf{P}' \}\ s\,;\texttt{skip}\ \{ \mathbf{Q}' \}$ for some $\mathbf{P}'$ and $\mathbf{Q}'$ such that $\mathbf{P} \models \mathbf{P}'$ and $\mathbf{Q}' \models \mathbf{Q}$
  - By applying the induction hypothesis to T', we deduce that there is a derivation tree for $\{ \mathbf{P}' \}\ s\ \{ \mathbf{Q}' \}$
  - We extend the tree for $\{ \mathbf{P}' \}\ s\ \{ \mathbf{Q}' \}$ using $\textsc{Cons}_{Ax}$ (using $\mathbf{P} \models \mathbf{P}'$ and $\mathbf{Q}' \models \mathbf{Q}$) to obtain a derivation tree for $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$

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:

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

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

- 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: $\underline{\mathbf{P}} \equiv \mathtt{x} > 0 \wedge \mathtt{y}*\mathtt{x}! = N!$
- 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

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

$\models$

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

`y := 1;`

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

`while not x = 1 do`*

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

$\models$

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

`y := y*x;`

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

`x := x-1`

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

`end`

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

$\models$

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

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

Zune Bug Revisited

```
//-------------------------
// Split total days since
// Jan. 01, ORIGINYEAR
// into year, month and day
//-------------------------
BOOL ConvertDays(UINT32 days, ...) {
  int year = ORIGINYEAR; /* =1980 */

  while (365 < days) {
    if (IsLeapYear(year)) {
      if (366 < days) {
        days -= 366; year += 1;
      }
    } else {
      days -= 365; year += 1;
    }
  }
  ... }
```

- Invariant: $\underline{\mathbf{P}} \equiv \texttt{true}$
- Variant: days
- Side condition: $365 < \texttt{days} \wedge \texttt{true} \models 0 \leq \texttt{days}$
- Because of the termination bug in the original code, a total correctness proof will fail

(Failing) Proof Attempt for Zune Bug

$\{\ \texttt{true}\ \}$
$\quad\boxed{\texttt{while } 365 < \texttt{days do}}$
$\quad\{\ 365 < \texttt{days} \wedge \texttt{days} = Z\ \}$
$\quad\quad\boxed{\texttt{if } L(\texttt{year}) \texttt{ then}}$
$\quad\quad\{\ L(\texttt{year}) \wedge 365 < \texttt{days} \wedge \texttt{days} = Z\ \}$
$\quad\quad\quad\boxed{\texttt{if } 366 < \texttt{days then}}$
$\quad\quad\quad\{\ 366 < \texttt{days} \wedge L(\texttt{year}) \wedge 365 < \texttt{days} \wedge \texttt{days} = Z\ \}$
$\quad\quad\quad\models$
$\quad\quad\quad\{\ \texttt{days-366} < Z\ \}$
$\quad\quad\quad\quad\boxed{\texttt{days} := \texttt{days} - 366; \texttt{year} := \texttt{year} + 1}$
$\quad\quad\quad\{\ \Downarrow \texttt{days} < Z\ \}$
$\quad\quad\quad\boxed{\texttt{else}}$
$\quad\quad\quad\{\ \neg(366 < \texttt{days}) \wedge L(\texttt{year}) \wedge 365 < \texttt{days} \wedge \texttt{days} = Z\ \}$
$\quad\quad\quad\not\models$
$\quad\quad\quad\{\ \texttt{days} < Z\ \}$
$\quad\quad\quad\quad\boxed{\texttt{skip}}$
$\quad\quad\quad\{\ \Downarrow \texttt{days} < Z\ \}$
$\quad\quad\quad\boxed{\texttt{end}}$
$\quad\quad\{\ \Downarrow \texttt{days} < Z\ \}$
$\quad\quad\boxed{\texttt{else}}$
$\quad\quad\{\ \neg L(\texttt{year}) \wedge 365 < \texttt{days} \wedge \texttt{days} = Z\ \}$
$\quad\quad\models$
$\quad\quad\{\ \texttt{days-365} < Z\ \}$
$\quad\quad\quad\boxed{\texttt{days} := \texttt{days} - 365; \texttt{year} := \texttt{year} + 1}$
$\quad\quad\{\ \Downarrow \texttt{days} < Z\ \}$
$\quad\quad\boxed{\texttt{end}}$
$\quad\{\ \Downarrow \texttt{days} < Z\ \}$
$\quad\boxed{\texttt{end}}$
$\{\ \Downarrow \neg(365 < \texttt{days})\ \}$
$\models$
$\{\ \Downarrow \texttt{true}\ \}$

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

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

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

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

- 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

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

- With this rule, we cannot prove that the following loop always terminates

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

  - The loop variant is `x`
  - The strongest possible loop invariant is `true` (because we want to show termination for all initial states)
  - This loop invariant is not strong enough to show the side condition

Soundness and Completeness

- Soundess 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:

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

  - 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 $\mathbf{P}$ holds, if the execution terminates, then $\mathbf{Q}$ 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

Soundess and Completeness Theorem (Partial Correctness):

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

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

Soundness Proof

- We prove $\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \} \Rightarrow \vDash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- That is, we have to show

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

- The proof runs by induction on the shape of the derivation tree for $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- See the Nielson & Nielson book for the full proof

Weakest (Liberal) Preconditions

- The weakest precondition of a statement s and a postcondition $\mathbf{Q}$ is the weakest predicate that has to hold in the initial state of an execution of s to guarantee that $\mathbf{Q}$ holds in the final state
  - The weakest precondition $wp(s, \mathbf{Q})$ guarantees termination
  - The weakest liberal precondition $wlp(s, \mathbf{Q})$ does not guarantee termination

$$\begin{aligned} \mathcal{B}[\![wp(s, \mathbf{Q})]\!]\sigma = tt &\Leftrightarrow \exists \sigma'.(\langle s, \sigma\rangle \to \sigma' \wedge \mathcal{B}[\![\mathbf{Q}]\!]\sigma') \\ \mathcal{B}[\![wlp(s, \mathbf{Q})]\!]\sigma = tt &\Leftrightarrow \forall \sigma'.(\langle s, \sigma\rangle \to \sigma' \Rightarrow \mathcal{B}[\![\mathbf{Q}]\!]\sigma') \end{aligned}$$

- In the following, we consider partial correctness

Weakest Preconditions Lemma

Lemma: there exists a function *wlp* from pairs of statements and assertions to assertions, such that, for every statement s and predicate $\mathbf{Q}$, we have:

1. $\models \{\ \mathit{wlp}(s, \mathbf{Q})\ \}\ s\ \{\ \mathbf{Q}\ \}$
2. $\models \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \} \Rightarrow (\mathbf{P} \models \mathit{wlp}(s, \mathbf{Q}))$

- Note that we require a semantic notion of being "weakest", here, defined in terms of $\models$
- See the Nielson & Nielson book for the definition of *wlp* which satisfies this property (and the proof of this lemma)

Completeness Proof

- We need to prove $\models \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \} \Rightarrow \vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- It suffices to infer $\vdash \{ \mathit{wlp}(s, \mathbf{Q}) \}\ s\ \{ \mathbf{Q} \}$, because:
  - By $\models \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$, the *wlp*-lemma on the previous slide implies $\mathbf{P} \models \mathit{wlp}(s, \mathbf{Q})$
  - Therefore, if we can show $\vdash \{ \mathit{wlp}(s, \mathbf{Q}) \}\ s\ \{ \mathbf{Q} \}$, we can conclude $\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$ using the rule of consequence:

$$\frac{\{ \mathit{wlp}(s, \mathbf{Q}) \}\ s\ \{ \mathbf{Q} \}}{\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}}\ (\textsc{Cons}_{Ax})$$

- The property $\vdash \{ \mathit{wlp}(s, \mathbf{Q}) \}\ s\ \{ \mathbf{Q} \}$ is proved by structural induction on s (see the Nielson & Nielson book for the definitions and full proof)

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