Formal Methods and Functional Programming

Axiomatic Semantics

Peter Müller

Chair of Programming Methodology
ETH Zurich

Program Correctness

- Semantics can be used to prove 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

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

3. 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 $\texttt{x} < 1$

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

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

- This specification expresses partial correctness in natural semantics

Correctness Proof

- We prove partial correctness in three steps
- Step 1: The body of the loop satisfies

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

- Step 2: The loop satisfies

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

- Step 3: The whole statement is partially correct

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

Proof: Step 1—Loop Body

- Since we have the transition $\langle \texttt{y:=y*x;x:=x-1}, \sigma\rangle \to \sigma''$, we can assume that there are transitions $\langle \texttt{y:=y*x}, \sigma\rangle \to \sigma'$ and $\langle \texttt{x:=x-1}, \sigma'\rangle \to \sigma''$ for some σ'
- We get $\sigma' = \sigma[\texttt{y} \mapsto \mathcal{A}[\![\texttt{y*x}]\!]\sigma]$ and $\sigma'' = \sigma'[\texttt{x} \mapsto \mathcal{A}[\![\texttt{x-1}]\!]\sigma']$, which imply $\sigma'' = \sigma[\texttt{y} \mapsto \sigma(\texttt{y}) \times \sigma(\texttt{x})][\texttt{x} \mapsto \sigma(\texttt{x})\texttt{-1}]$
- By $\sigma''(\texttt{x}) > 0$, we calculate

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

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

Proof: Step 2—Loop

- Step 2: The loop satisfies

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

- We prove this property by induction on the shape of the derivation tree
  - There are two cases for the last rule applied: $\textsc{WhF}_{NS}$ and $\textsc{WhT}_{NS}$
- Case $\textsc{WhF}_{NS}$
  - We have $\sigma(\texttt{x}) = 1$ and $\sigma = \sigma''$
  - Since $1 = 1!$, we get $\sigma(\texttt{y}) \times \sigma(\texttt{x})! = \sigma(\texttt{y}) = \sigma''(\texttt{y})$
  - We trivially get $\sigma''(\texttt{x}) = 1$ and $\sigma(\texttt{x}) > 0$

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

- From the rule of the natural semantics we get for some σ'''

  (1) $\langle \texttt{y:=y*x;x:=x-1}, \sigma\rangle \rightarrow \sigma'''$

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

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

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

- Combining these results yields
  $\sigma(\mathtt{y}) \times \sigma(\mathtt{x})! = \sigma''(\mathtt{y}) \wedge \sigma''(\mathtt{x}) = 1 \wedge \sigma(\mathtt{x}) > 0$

Proof: Step 3—Factorial Statement

- Step 3: The whole statement is partially correct

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

- From the natural semantics we get for some σ''

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

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

- 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 $\sigma''(\texttt{y}) \times \sigma''(\texttt{x})! = \sigma'(\texttt{y}) \land \sigma'(\texttt{x}) = 1 \land \sigma''(\texttt{x}) > 0$
- We conclude $1 \times \sigma(\texttt{x})! = \sigma'(\texttt{y}) \land \sigma(\texttt{x}) > 0$

Verification Example: Observations

- We can prove correctness of a program based on a formal semantics
  - The proof would also be possible with SOS and denotational 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
- Axiomatic Semantics describes essential properties of syntactic constructs
  - The choice of essential properties depends on what we want to prove

3. Axiomatic Semantics

Assertions

- Properties of programs are specified as assertions

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

where s is a statement and **P** and **Q** are predicates (boolean expressions)

- Terminology
  - Assertions are also called (Hoare) triples
  - **P** is called the precondition
  - **Q** is called the postcondition

Meaning of Assertions

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

  > If $\mathbf{P}$ evaluates to true in the initial state σ, and
  > if the execution of s from σ terminates in a state σ'
  > then $\mathbf{Q}$ 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 assertions

Assertions: Example

- Specification of the factorial statement by an assertion

$$\{\, \mathtt{true} \,\}$$
$$\mathtt{y:=1;while}\ \mathtt{not}\ \mathrm{x} = 1\ \mathtt{do}\ \mathtt{y:=y*x;x:=x-1}\ \mathtt{end}$$
$$\{\, \mathrm{y} = \mathrm{x}! \land \mathrm{x} > 0 \,\}$$

- In general, this assertion does not hold
  - Consider an initial state $\{\, \mathrm{x} \mapsto 2, \mathrm{y} \mapsto 0 \,\}$
  - The final state will be $\{\, \mathrm{x} \mapsto 1, \mathrm{y} \mapsto 2 \,\}$
- We have to express that `y` in the final state is the factorial of `x` in the initial state

Logical Variables

- Assertions can contain logical variables
  - Logical variables may occur only in 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 of the initial state for the final state

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

- States map logical variables to their values

Assertion Language

- Pre- and postconditions are predicates, that is, boolean expressions
  - It is common 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 for the extended expression language

- 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"

3. Axiomatic Semantics

Inference System

- We formalize the semantics of a programming language by describing the valid assertions
- This is done by an inference system
  - An inference system consists of a set of axioms and rules
  - The formulas of the inference system are assertions

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

- The inference system specifies an axiomatic semantics of the programming language

Axiomatic Semantics of IMP

- `skip` does not modify the state

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

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

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

  - 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

- The rules are axiom schemes

Axiomatic Semantics of IMP (cont'd)

- Sequential composition $s_1 ; s_2$

$$\textsc{Seq}_{Ax}\,\frac{\{\,\mathbf{P}\,\}\; s_1 \;\{\,\mathbf{Q}\,\} \qquad \{\,\mathbf{Q}\,\}\; s_2 \;\{\,\mathbf{R}\,\}}{\{\,\mathbf{P}\,\}\; s_1 ; s_2 \;\{\,\mathbf{R}\,\}}$$

- Conditional statement $\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}$

$$\textsc{If}_{Ax}\,\frac{\{\, b \wedge \mathbf{P}\,\}\; s_1 \;\{\,\mathbf{Q}\,\} \qquad \{\,\neg b \wedge \mathbf{P}\,\}\; s_2 \;\{\,\mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\; \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end} \;\{\,\mathbf{Q}\,\}}$$

Axiomatic Semantics of IMP (cont'd)

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

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

  - **P** is the loop invariant

- Rule of consequence

$$\textsc{Cons}_{Ax}\ \frac{\{\ \mathbf{P}'\ \}\ s\ \{\ \mathbf{Q}'\ \}}{\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}}\ \text{if } \mathbf{P} \Rightarrow \mathbf{P}' \text{ and } \mathbf{Q}' \Rightarrow \mathbf{Q}$$

  - We can strengthen preconditions
  - We can weaken postconditions
  - $\mathbf{P} \Rightarrow \mathbf{Q}$ is defined as:
    "For all states σ, $\mathcal{B}[[\mathbf{P}]]\sigma$ implies $\mathcal{B}[[\mathbf{Q}]]\sigma$"

Inference Trees

- Axioms and rules are used like in natural semantics or natural deduction
- Derivation trees are called inference trees since they show how to infer an assertion
  - The leaves are instances of axiom schemes
  - The internal nodes correspond to instances of rules
- A finite inference tree gives a proof of the assertion at its root
- To express that an assertion $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$ can be inferred, we write

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

Inference Trees: Example 1

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

$$\boxed{\texttt{z:=x; x:=y; y:=z}}$$

- We can build the following inference tree

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\ \mathbf{P}\ \}\ \texttt{z:=x}\ \{\ \mathtt{z} = X_0 \land \mathtt{y} = Y_0\ \}}}{\{\ \mathbf{P}\ \}\ \texttt{z:=x}\ \{\ \mathtt{y} = Y_0 \land \mathtt{z} = X_0\ \}} \quad \dfrac{}{\{\ \mathtt{y} = Y_0 \land \mathtt{z} = X_0\ \}\ \texttt{x:=y}\ \{\ \mathbf{Q'}\ \}}}{\{\ \mathbf{P}\ \}\ \texttt{z:=x; x:=y}\ \{\ \mathbf{Q'}\ \}} \quad \dfrac{}{\{\ \mathbf{Q'}\ \}\ \texttt{y:=z}\ \{\ \mathbf{Q}\ \}}}{\{\ \mathbf{P}\ \}\ \texttt{z:=x; x:=y; y:=z}\ \{\ \mathbf{Q}\ \}}$$

where we write:

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

Inference Trees: Example 2

- Consider the non-terminating loop

```
while true  do skip end
```

- We can build the following inference tree

$$
\textsc{Cons}_{Ax}\;\dfrac{\textsc{Wh}_{Ax}\;\dfrac{\textsc{Cons}_{Ax}\;\dfrac{\textsc{Skip}_{Ax}\;\dfrac{}{\{\,\texttt{true}\,\}\;\texttt{skip}\;\{\,\texttt{true}\,\}}}{\{\,\texttt{true}\wedge\texttt{true}\,\}\;\texttt{skip}\;\{\,\texttt{true}\,\}}}{\{\,\texttt{true}\,\}\;\texttt{while true do skip end}\;\{\,\neg\texttt{true}\wedge\texttt{true}\,\}}}{\{\,\texttt{true}\,\}\;\texttt{while true do skip end}\;\{\,\neg\texttt{true}\,\}}
$$

- This proof illustrates that we have partial correctness

Proof Outlines

- Inference 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 axioms as:

$$\begin{array}{l} \{\ \mathbf{P}\ \} \\ \quad \texttt{skip} \\ \{\ \mathbf{P}\ \} \end{array} \qquad\qquad \begin{array}{l} \{\ \mathbf{P}[x \mapsto e]\ \} \\ \quad x\texttt{:=}e \\ \{\ \mathbf{P}\ \} \end{array}$$

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

$$\begin{array}{l} \{\ \mathbf{P}\ \} \\ \quad s_1\,; \\ \{\ \mathbf{Q}\ \} \\ \quad s_2 \\ \{\ \mathbf{R}\ \} \end{array}$$

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

Proof Outlines: Notation (cont'd)

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

```
{ P }
  if b then
    { b ∧ P }
      s1
    { Q }
  else
    { ¬b ∧ P }
      s2
    { Q }
  end
{ Q }
```

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

```
{ P }
  while b do
    { b ∧ P }
      s
    { P }
  end
{ ¬b ∧ P }
```

Proof Outlines: Notation (cont'd)

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

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

  - We omit the implication when **P** and **P**′ or **Q** and **Q**′ are syntactically identical

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

Proof Outlines: Example

- Back to our swap-example:

```
z:=x; x:=y; y:=z
```

- Proof outline:

$$\{\, \mathtt{x} = X_0 \land \mathtt{y} = Y_0 \,\}$$
$$\Rightarrow$$
$$\{\, \mathtt{y} = Y_0 \land \mathtt{x} = X_0 \,\}$$
$$\boxed{\mathtt{z\ :=\ x;}}$$
$$\{\, \mathtt{y} = Y_0 \land \mathtt{z} = X_0 \,\}$$
$$\boxed{\mathtt{x\ :=\ y;}}$$
$$\{\, \mathtt{x} = Y_0 \land \mathtt{z} = X_0 \,\}$$
$$\boxed{\mathtt{y\ :=\ z}}$$
$$\{\, \mathtt{x} = Y_0 \land \mathtt{y} = X_0 \,\}$$

- Proof outlines are typically developed bottom-up

Verification of Factorial Statement

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

Verification of Factorial Statement

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

- Determining the loop invariant

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

Verification of Factorial Statement

$$
\begin{array}{l}
\{\, \mathtt{x} = N \,\} \\
\quad \mathtt{y{:=}1;while\ not\ x} = 1\ \mathtt{do\ y{:=}y{*}x;x{:=}x{-}1\ end} \\
\{\, \mathtt{y} = N! \wedge N > 0 \,\}
\end{array}
$$

- Determining the loop invariant

Iteration	0	1	2	i	N-1
x	N	N-1	N-2	N-i	1
y	1	N	$N*(N\text{-}1)$	$N*(N\text{-}1)*\ldots*(N\text{-}i+1)$	$N!$

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

Proof Outline for Factorial Statement

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

$\Rightarrow$

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

```
y := 1;
```

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

```
while not x = 1 do
```

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

$\Rightarrow$

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

```
y := y*x;
```

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

```
x := x-1
```

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

```
end
```

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

$\Rightarrow$

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

Verification of Zune Example

$\{ \dots \}$

```
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
```

$\{ \text{year} - 1980 \le D/365 \}$

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
  - $\texttt{year} = 1980 + i$
  - $\texttt{days} \leq \texttt{D-}i \times 365$

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: $\texttt{year-1980} \leq (D\texttt{-days})/365 \land 0 \leq \texttt{days}$

Proof Outline for Zune Example

$\{\ 0 \leq D\ \}$
$\Rightarrow$
$\{\ 1980\text{-}1980 \leq (D\text{-}D)/365 \wedge 0 \leq D\ \}$

```
year := 1980;
```

$\{\ \texttt{year}\text{-}1980 \leq (D\text{-}D)/365 \wedge 0 \leq D\ \}$

```
days := D;
```

$\{\ \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$

```
while L(year) and 366 < days or not L(year) and 365 < days do
```

$\{\ (L(\texttt{year}) \wedge 366 < \texttt{days} \vee \neg L(\texttt{year}) \wedge 365 < \texttt{days}) \wedge \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$
$\Rightarrow$
$\{\ \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge (L(\texttt{year}) \Rightarrow 366 < \texttt{days}) \wedge 365 < \texttt{days}\ \}$

```
if L(year) then
```

$\{\ L(\texttt{year}) \wedge \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge (L(\texttt{year}) \Rightarrow 366 < \texttt{days}) \wedge 365 < \texttt{days}\ \}$
$\Rightarrow$
$\{\ \texttt{year} + 1\text{-}1980 \leq (D\text{-}(\texttt{days}\text{-}366))/365 \wedge 0 \leq (\texttt{days}\text{-}366)\ \}$

```
days := days - 366
```

$\{\ \texttt{year} + 1\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$

```
else
```

$\{\ \neg L(\texttt{year}) \wedge \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge (L(\texttt{year}) \Rightarrow 366 < \texttt{days}) \wedge 365 < \texttt{days}\ \}$
$\Rightarrow$
$\{\ \texttt{year} + 1\text{-}1980 \leq (D\text{-}(\texttt{days}\text{-}365))/365 \wedge 0 \leq \texttt{days}\text{-}365\ \}$

```
days := days - 365;
```

$\{\ \texttt{year} + 1\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$

```
end;
```

$\{\ \texttt{year} + 1\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$

```
year := year + 1
```

$\{\ \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$

```
end
```

$\{\ \neg(L(\texttt{year}) \wedge 366 < \texttt{days} \vee \neg L(\texttt{year}) \wedge 365 < \texttt{days}) \wedge \texttt{year}\text{-}1980 \leq (D\text{-}\texttt{days})/365 \wedge 0 \leq \texttt{days}\ \}$
$\Rightarrow$
$\{\ \texttt{year}\text{-}1980 \leq D/365\ \}$

3. Axiomatic Semantics

Induction on the Shape of Inference Trees

- Properties of the axiomatic semantics are typically proved by induction on the shape of the inference tree
  - Analogous to induction on the shape of derivation trees in NS
  - Note: structural induction on the shape of the statement does not work because of the rule of consequence

- Induction on the shape of inference trees
  To prove a property $P(t)$ for all inference trees t, prove that $P(t)$ holds for an arbitrary inference tree t under the assumption that $P(t')$ holds for all proper sub-trees t' of t

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

Proving Properties: Example

- We prove the lemma

$$\boxed{\text{If} \vdash \{\ \mathbf{P}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}\ \text{then}\ \mathbf{P} \Rightarrow \mathbf{Q}}$$

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

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

- Out of the six rules of the axiomatic semantics, only $\textsc{Skip}_{Ax}$ and $\textsc{Cons}_{Ax}$ are possibly at the root of this inference tree

Proving Properties: Example (cont'd)

- Case $\textsc{Skip}_{Ax}$:
  - We get $\mathbf{P} = \mathbf{Q}$ and, thus, $\mathbf{P} \Rightarrow \mathbf{Q}$

- Case $\textsc{Cons}_{Ax}$:
  - From $\textsc{Cons}_{Ax}$, we know that there exists an inference tree for $\{\, \mathbf{P'} \,\}\ \texttt{skip}\ \{\, \mathbf{Q'} \,\}$, where $\mathbf{P} \Rightarrow \mathbf{P'}$ and $\mathbf{Q'} \Rightarrow \mathbf{Q}$
  - By applying the induction hypothesis to $\{\, \mathbf{P'} \,\}\ \texttt{skip}\ \{\, \mathbf{Q'} \,\}$, we get $\mathbf{P'} \Rightarrow \mathbf{Q'}$
  - Now we have $\mathbf{P} \Rightarrow \mathbf{P'}$, $\mathbf{P'} \Rightarrow \mathbf{Q'}$, and $\mathbf{Q'} \Rightarrow \mathbf{Q}$ and, thus, $\mathbf{P} \Rightarrow \mathbf{Q}$

Semantic Equivalence

Two statements s_1 and s_2 are provably equivalent if for all preconditions $\mathbf{P}$ and postconditions $\mathbf{Q}$ we have

$$\vdash \{\ \mathbf{P}\ \}\ s_1\ \{\ \mathbf{Q}\ \} \text{if and only if} \vdash \{\ \mathbf{P}\ \}\ s_2\ \{\ \mathbf{Q}\ \}$$

- Example: s and s;`skip` are equivalent
- Proof for "$\Rightarrow$"
  - We know there is an inference tree for $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$
  - We extend that tree using the `skip`-axiom and the rule for sequential composition:

$$\textsc{Seq}_{Ax}\ \dfrac{\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \} \qquad \textsc{Skip}_{Ax}\ \dfrac{}{\{\ \mathbf{Q}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}}}{\{\ \mathbf{P}\ \}\ s;\texttt{skip}\ \{\ \mathbf{Q}\ \}}$$

Semantic Equivalence: Proof for "$\Leftarrow$"

- The proof runs by induction on the shape of the inference tree
  - Only two rules may produce $\{\ \mathbf{P}\ \}\ s;\mathtt{skip}\ \{\ \mathbf{Q}\ \}$: $\textsc{Seq}_{Ax}$ and $\textsc{Cons}_{Ax}$
- Case $\textsc{Seq}_{Ax}$
  - We know there are inference trees for $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{R}\ \}$ and $\{\ \mathbf{R}\ \}\ \mathtt{skip}\ \{\ \mathbf{Q}\ \}$ for some predicate $\mathbf{R}$
  - Applying the auxiliary lemma to $\vdash \{\ \mathbf{R}\ \}\ \mathtt{skip}\ \{\ \mathbf{Q}\ \}$ yields $\mathbf{R} \Rightarrow \mathbf{Q}$
  - We extend the inference tree for $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{R}\ \}$ using $\textsc{Cons}_{Ax}$ to obtain $\vdash \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$
- Case $\textsc{Cons}_{Ax}$
  - We know that there exists an inference tree for $\{\ \mathbf{P}'\ \}\ s;\mathtt{skip}\ \{\ \mathbf{Q}'\ \}$ where $\mathbf{P} \Rightarrow \mathbf{P}'$ and $\mathbf{Q}' \Rightarrow \mathbf{Q}$
  - By applying the induction hypothesis to $\vdash \{\ \mathbf{P}'\ \}\ s;\mathtt{skip}\ \{\ \mathbf{Q}'\ \}$, we know there is an inference tree for $\{\ \mathbf{P}'\ \}\ s\ \{\ \mathbf{Q}'\ \}$
  - We extend the tree for $\{\ \mathbf{P}'\ \}\ s\ \{\ \mathbf{Q}'\ \}$ using $\textsc{Cons}_{Ax}$ to obtain a tree for $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$

3. Axiomatic Semantics

Total Correctness

- 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

- All rules except the rule for loops are analogous

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 iteration decreases the value of the loop variant
- The loop has to terminate when a minimal value of the well-founded set is reached
- Example

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

  - Possible loop variant `x`

While Rule for Total Correctness

- For simplicity, we consider loop variants that evaluate to values in $\mathbb{N}$
  - We use arithmetic expressions of IMP as loop variants
  - We prove explicitly that $\mathcal{A}[\![e]\!] \in \mathbb{N}$ before each iteration
  - Intuition: loop variant gives an upper bound on the number of iterations
- Rule:

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

  where Z is a fresh logical variable

- Other well-founded sets are also possible and useful

Total Correctness of Factorial

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

- Invariant: $\mathbf{P} \equiv \mathtt{x} > 0 \wedge \mathtt{y*x!} = N!$
- Variant: $\mathtt{x}$
- Side condition: $\mathtt{x} \neq 1 \wedge \mathtt{x} > 0 \wedge \mathtt{y*x!} = N! \Rightarrow 0 \leq \mathtt{x}$

Proof Outline for Factorial Statement

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

$\Rightarrow$

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

$\boxed{\texttt{y := 1;}}$

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

$\boxed{\texttt{while not x = 1 do}}$

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

$\Rightarrow$

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

$\boxed{\texttt{y := y*x;}}$

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

$\boxed{\texttt{x := x-1}}$

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

$\boxed{\texttt{end}}$

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

$\Rightarrow$

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

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: $\mathbf{P} \equiv \texttt{true}$
- Variant: $\texttt{days}$
- Side condition: $365 < \texttt{days} \wedge \texttt{true} \Rightarrow 0 \leq \texttt{days}$

(Failing) Proof Attempt for Zune Bug

$\{\ \mathtt{true}\ \}$

$\boxed{\mathtt{while\ 365\ <\ days\ do}}$

$\{\ 365 < \mathtt{days} \land \mathtt{days} = Z\ \}$

$\boxed{\mathtt{if}\ \ L(\mathtt{year})\ \ \mathtt{then}}$

$\{\ L(\mathtt{year}) \land 365 < \mathtt{days} \land \mathtt{days} = Z\ \}$

$\boxed{\mathtt{if\ 366\ <\ days\ then}}$

$\{\ 366 < \mathtt{days} \land L(\mathtt{year}) \land 365 < \mathtt{days} \land \mathtt{days} = Z\ \}$

$\Rightarrow$

$\{\ \mathtt{days}\text{-}366 < Z\ \}$

$\boxed{\mathtt{days\ :=\ days\ -\ 366;year\ :=\ year\ +\ 1}}$

$\{\ \Downarrow \mathtt{days} < Z\ \}$

$\boxed{\mathtt{else}}$

$\{\ \neg(366 < \mathtt{days}) \land L(\mathtt{year}) \land 365 < \mathtt{days} \land \mathtt{days} = Z\ \}$

$\Rightarrow$

$\{\ \mathtt{days} < Z\ \}$

$\boxed{\mathtt{skip}}$

$\{\ \Downarrow \mathtt{days} < Z\ \}$

$\boxed{\mathtt{end}}$

$\{\ \Downarrow \mathtt{days} < Z\ \}$

$\boxed{\mathtt{else}}$

$\{\ \neg L(\mathtt{year}) \land 365 < \mathtt{days} \land \mathtt{days} = Z\ \}$

$\Rightarrow$

$\{\ \mathtt{days}\text{-}365 < Z\ \}$

$\boxed{\mathtt{days\ :=\ days\ -\ 365;\ year\ :=\ year\ +\ 1}}$

$\{\ \Downarrow \mathtt{days} < Z\ \}$

$\boxed{\mathtt{end}}$

$\{\ \Downarrow \mathtt{days} < Z\ \}$

$\boxed{\mathtt{end}}$

$\{\ \Downarrow \neg(365 < \mathtt{days})\ \}$

$\Rightarrow$

$\{\ \Downarrow \mathtt{true}\ \}$

3. Axiomatic Semantics

3.1 Hoare Logic

3.2 Soundness and Completeness

Motivation

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

Unsoundness: Example

$$\mathrm{W{\scriptstyle HU}}_{Ax}\ \frac{\{\, b \wedge \mathbf{P} \wedge e = Z \,\}\ s\ \{ \Downarrow \mathbf{P} \wedge e < Z \,\}}{\{\, \mathbf{P} \wedge 0 \leq e \,\}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{ \Downarrow \neg b \wedge \mathbf{P} \,\}}$$

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

$$\mathrm{C{\scriptstyle ONS}}\ \dfrac{\mathrm{W{\scriptstyle HU}}\ \dfrac{\mathrm{C{\scriptstyle ONS}}\ \dfrac{\mathrm{A{\scriptstyle SS}}\ \dfrac{}{\{\, \texttt{true} \wedge \texttt{x-1} < Z \,\}\ \texttt{x:=x-1}\ \{ \Downarrow \texttt{true} \wedge \texttt{x} < Z \,\}}}{\{\, \texttt{true} \wedge \texttt{true} \wedge \texttt{x} = Z \,\}\ \texttt{x:=x-1}\ \{ \Downarrow \texttt{true} \wedge \texttt{x} < Z \,\}}}{\{\, \texttt{true} \wedge 0 \leq \texttt{x} \,\}\ \texttt{while true do x:=x-1 end}\ \{ \Downarrow \neg\texttt{true} \wedge \texttt{true} \,\}}}{\{\, 0 \leq \texttt{x} \,\}\ \texttt{while true do x:=x-1 end}\ \{ \Downarrow \texttt{true} \,\}}$$

- 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

$$\textsc{Whi}_{Ax} \frac{\{\, b \wedge \mathbf{P} \wedge e = Z \,\}\; s \;\{\, \Downarrow \mathbf{P} \wedge e < Z \,\}}{\{\, \mathbf{P} \,\}\; \texttt{while}\; b\; \texttt{do}\; s\; \texttt{end}\; \{\, \Downarrow \neg b \wedge \mathbf{P} \,\}} \quad \text{if } \mathbf{P} \Rightarrow 0 \le e$$

- 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

The partial correctness assertion $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$ is valid—written as $\models \{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$— iff

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

- 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

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

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

Soundness Proof

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

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

- The proof runs by induction on the shape of the inference tree for $\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- 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

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

- In the following, we consider partial correctness

Completeness Proof

- We prove $\models \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \} \Rightarrow \vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- It suffices to infer $\vdash \{ \mathit{wlp}(s, \mathbf{Q}) \}\ s\ \{ \mathbf{Q} \}$
  - By $\models \{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$, the *wlp*-lemma implies $\mathbf{P} \Rightarrow \mathit{wlp}(s, \mathbf{Q})$

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

- We prove $\vdash \{ \mathit{wlp}(s, \mathbf{Q}) \}\ s\ \{ \mathbf{Q} \}$ by structural induction on s
- See the Nielson & Nielson book for the full proof

Summary: Axiomatic Semantics

- Axiomatic semantics
  - expresses specific properties of the effect of executing a program
  - Some aspects of the computation may be ignored
- Axiomatic semantics is used to verify programs
  - Partial correctness
  - Total correctness
  - Other properties, e.g., resource consumption
- The inference system for partial correctness of IMP programs is sound and complete