Semantics of Programming Languages

Axiomatic Semantics

Prof. Peter Müller

Software Component Technology

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology 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**

total correctness = partial correctness + 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

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

- 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

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

$$\sigma(\mathsf{y}) \times \sigma(\mathsf{x})! = \sigma''(\mathsf{y}) \times \sigma''(\mathsf{x})! \wedge \sigma(\mathsf{x}) > 0$$

- Step 2: The loop satisfies

$$\langle \mathtt{while}\ \mathtt{not}\ \mathsf{x} = 1\ \mathtt{do}\ \mathsf{y:=y} * \mathsf{x;x:=x} - 1\ \mathtt{end}, \sigma \rangle \rightarrow \sigma'' \Rightarrow$$

$$\sigma(\mathsf{y}) \times \sigma(\mathsf{x})! = \sigma''(\mathsf{y}) \wedge \sigma''(\mathsf{x}) = 1 \wedge \sigma(\mathsf{x}) > 0$$

- Step 3: The whole statement is partially correct

$$\langle \mathsf{y:=}1\mathtt{;while}\ \mathtt{not}\ \mathsf{x} = 1\ \mathtt{do}\ \mathsf{y:=y} * \mathsf{x;x:=x} - 1\ \mathtt{end}, \sigma \rangle \rightarrow \sigma' \Rightarrow$$

$$\sigma'(\mathsf{y}) = \sigma(\mathsf{x})! \wedge \sigma(\mathsf{x}) > 0$$

Proof: Step 1—Loop Body

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

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

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

Proof: Step 2—Loop

- Step 2: The loop satisfies

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

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

- Case 1: $\mathcal{B}[\![\texttt{not x} = 1]\!]\sigma = \mathit{ff}$
  - We have $\sigma(\mathsf{x}) = 1$ and $\sigma = \sigma''$
  - Since $1 = 1!$, we get $\sigma(\mathsf{y}) \times \sigma(\mathsf{x})! = \sigma(\mathsf{y}) = \sigma''(\mathsf{y})$
  - We trivially get $\sigma''(\mathsf{x}) = 1$ and $\sigma(\mathsf{x}) > 0$

Proof: Step 2—Loop (Case 2)

- Case 2: $\mathcal{B}[\![\texttt{not x} = 1]\!]\sigma = tt$
- From the rule of the natural semantics we get for some σ'''

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

  (2) $\langle \texttt{while not x} = 1 \texttt{ do y:=y} * \texttt{x;x:=x} - 1 \texttt{ end}, \sigma'''\rangle \to \sigma''$
- Applying the induction hypothesis to (2) yields
  $\sigma'''(\mathsf{y}) \times \sigma'''(\mathsf{x})! = \sigma''(\mathsf{y}) \wedge \sigma''(\mathsf{x}) = 1 \wedge \sigma'''(\mathsf{x}) > 0$
- By (1), $\sigma'''(\mathsf{x}) > 0$, and Proof Step 1, we get
  $\sigma(\mathsf{y}) \times \sigma(\mathsf{x})! = \sigma'''(\mathsf{y}) \times \sigma'''(\mathsf{x})! \wedge \sigma(\mathsf{x}) > 0$
- Combining these results yields
  $\sigma(\mathsf{y}) \times \sigma(\mathsf{x})! = \sigma''(\mathsf{y}) \wedge \sigma''(\mathsf{x}) = 1 \wedge \sigma(\mathsf{x}) > 0$

Proof: Step 3—Factorial Statement

- Step 3: The whole statement is partially correct

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

  $\sigma'(\texttt{y}) = \sigma(\texttt{x})! \wedge \sigma(\texttt{x}) > 0$

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

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

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

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

Verification Example: Observations

- We can prove correctness of a program based on a formal semantics
  - The proof would also be possible with 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

4. Axiomatic Semantics

4.1 Hoare Logic

4.1.1 Proofs of Program Correctness

4.1.2 Assertion Language

4.1.3 Inference System

4.1.4 Properties of the Semantics

4.1.5 Extensions

4.2 Soundness and Completeness

Assertions

- Properties of programs are specified as **assertions**

$$\{ \mathrm{P} \}\ s\ \{ \mathrm{Q} \}$$

where s is a statement and P and Q are predicates

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

Meaning of Assertions

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

> **If** P holds in the initial state σ, and
> **if** the execution of s from σ terminates in a state σ'
> **then** Q will hold 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

$\{\ \mathit{true}\ \}$

$\texttt{y:=}1\texttt{;while not x} = 1\ \texttt{do y:=y} * \texttt{x;x:=x} - 1\ \texttt{end}$

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

- In general, this assertion does not hold
  - 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 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 can occur only in pre- and postconditions
  - Programs cannot access logical variables
- Logical variables can be used to save values of the initial state for the final state

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

$$\mathtt{y:=1;while\ not\ x} = 1\ \mathtt{do\ y:=y} * \mathtt{x;x:=x} - 1\ \mathtt{end}$$

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

- States map logical variables to their values
- The value of a logical variable can never change

Assertion Language

- Pre- and postconditions are predicates, that is functions State $\longrightarrow$ Bool
- Each boolean expression b defines a predicate $\mathcal{B}[\![b]\!]$
- If P, P_1, and P_2 are predicates, then we use the following notation for predicates

$P_1 \wedge P_2$	where $(P_1 \wedge P_2)\sigma \Leftrightarrow P_1(\sigma) \wedge P_2(\sigma)$
$P_1 \vee P_2$	where $(P_1 \vee P_2)\sigma \Leftrightarrow P_1(\sigma) \vee P_2(\sigma)$
$\neg P$	where $(\neg P)\sigma \Leftrightarrow \neg P(\sigma)$
$P[x \mapsto \mathcal{A}[\![e]\!]]$	where $(P[x \mapsto \mathcal{A}[\![e]\!]])\sigma \Leftrightarrow P(\sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma])$
$P_1 \Rightarrow P_2$	where $(P_1 \Rightarrow P_2)\sigma \Leftrightarrow P_1(\sigma) \Rightarrow P_2(\sigma)$

4. Axiomatic Semantics

4.1 Hoare Logic

Inference System

- We can formalize the semantics of a programming language by describing which assertions hold

- 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

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

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

Axiomatic Semantics of IMP

- `skip` does not modify the state

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

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

$$\{ \mathbf{P}[x \mapsto \mathcal{A}[\![e]\!]] \}\ x\texttt{:=}e\ \{ \mathbf{P} \}$$

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

- These rules are **axiom schemes**

Axiomatic Semantics of IMP (cont'd)

- Sequential composition $s_1 ; s_2$

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

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

$$\frac{\{\,\mathcal{B}[\![b]\!] \wedge \mathbf{P}\,\}\ s_1\ \{\,\mathbf{Q}\,\} \qquad \{\,\neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\,\}\ s_2\ \{\,\mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\ \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{end}\ \{\,\mathbf{Q}\,\}}$$

Axiomatic Semantics of IMP (cont'd)

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

$$\frac{\{\ \mathcal{B}[\![b]\!] \wedge \mathbf{P}\ \}\ s\ \{\ \mathbf{P}\ \}}{\{\ \mathbf{P}\ \}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\ \neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\ \}}$$

  - $\mathbf{P}$ is the **loop invariant**

- Rule of consequence

$$\frac{\{\ \mathbf{P}'\ \}\ s\ \{\ \mathbf{Q}'\ \}}{\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}}\ \text{if } \mathbf{P} \Rightarrow \mathbf{P}' \text{ and } \mathbf{Q}' \Rightarrow \mathbf{Q}$$

  - We can **strengthen preconditions**
  - We can **weaken postconditions**

Inference Trees

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

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

Inference Trees: Example

- Consider the non-terminating loop

```
while true do skip end
```

- We can build the following inference tree

$$\dfrac{\dfrac{\{\ \mathit{true}\ \}\ \texttt{skip}\ \{\ \mathit{true}\ \}}{\{\ \mathit{true} \wedge \mathit{true}\ \}\ \texttt{skip}\ \{\ \mathit{true}\ \}}}{\{\ \mathit{true}\ \}\ \texttt{while true do skip end}\ \{\ \neg\mathit{true} \wedge \mathit{true}\ \}}$$
$$\{\ \mathit{true}\ \}\ \texttt{while true do skip end}\ \{\ \mathit{true}\ \}$$

  where we write *true* for $\mathcal{B}[\![\texttt{true}]\!]$

- This proof illustrates that we have **partial correctness**

Verification of Factorial Statement

$\{ \mathsf{x} = N \}$

$\mathtt{y:=1;while\ not\ x} = 1\ \mathtt{do\ y:=y} * \mathtt{x;x:=x} - 1\ \mathtt{end}$

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

- Determining the loop invariant

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

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

Verification (cont'd)

- We verify the factorial statement in three steps

1. The precondition and the assignment establish the loop invariant
2. The loop body preserves the loop invariant
3. The loop invariant and the negation of the loop condition imply the postcondition

- The proof can be written
  - as inference tree
  - as proof outline

4. Axiomatic Semantics

4.1 Hoare Logic

4.1.1 Proofs of Program Correctness

4.1.2 Assertion Language

4.1.3 Inference System

4.1.4 Properties of the Semantics

4.1.5 Extensions

4.2 Soundness and Completeness

Proving Properties

- We prove the lemma

$$\text{If} \vdash \{\, \mathrm{P} \,\}\ \texttt{skip}\ \{\, \mathrm{Q} \,\} \text{ then } \mathrm{P} \Rightarrow \mathrm{Q}$$

  by **induction on the shape of the inference tree**

- Induction base
  - $\{\, \mathrm{P} \,\}\ \texttt{skip}\ \{\, \mathrm{Q} \,\}$ is an instance of the `skip` axiom
  - We get $\mathrm{P} = \mathrm{Q}$ and, thus, $\mathrm{P} \Rightarrow \mathrm{Q}$
- Induction step
  - $\{\, \mathrm{P} \,\}\ \texttt{skip}\ \{\, \mathrm{Q} \,\}$ is infered by the rule of consequence
  - We can apply the induction hypothesis to $\{\, \mathrm{P}' \,\}\ \texttt{skip}\ \{\, \mathrm{Q}' \,\}$ to get $\mathrm{P}' \Rightarrow \mathrm{Q}'$
  - By $\mathrm{P} \Rightarrow \mathrm{P}'$ and $\mathrm{Q}' \Rightarrow \mathrm{Q}$, we get $\mathrm{P} \Rightarrow \mathrm{Q}$

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`;skip` and s are equivalent
- Proof
  - Part 1: "$\Leftarrow$" is trivial

$$\frac{\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \} \quad \{\ \mathbf{Q}\ \}\ \texttt{skip}\ \{\ \mathbf{Q}\ \}}{\{\ \mathbf{P}\ \}\ s\texttt{;skip}\ \{\ \mathbf{Q}\ \}}$$

  - Part 2 "$\Rightarrow$" runs by induction on the shape of the inference tree for $\{\ \mathbf{P}\ \}\ s\texttt{;skip}\ \{\ \mathbf{Q}\ \}$

Proof: Part 2

- The induction base is trivial
- The induction step has two interesting cases
- Case composition rule
  - We have $\vdash \{ \mathbf{P} \}\ s\ \{ \mathbf{R} \}$ and $\vdash \{ \mathbf{R} \}\ \texttt{skip}\ \{ \mathbf{Q} \}$ for some predicate $\mathbf{R}$
  - Applying the auxiliary lemma yields $\mathbf{R} \Rightarrow \mathbf{Q}$
  - By the rule of consequence, we get $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$
- Case rule of consequence
  - We have $\{ \mathbf{P}' \}\ s\texttt{;skip}\ \{ \mathbf{Q}' \}$ where $\mathbf{P} \Rightarrow \mathbf{P}'$ and $\mathbf{Q}' \Rightarrow \mathbf{Q}$
  - Applying the induction hypothesis yields $\{ \mathbf{P}' \}\ s\ \{ \mathbf{Q}' \}$
  - By the rule of consequence, we get $\{ \mathbf{P} \}\ s\ \{ \mathbf{Q} \}$

Induction on Inference Trees

1. **Induction base**: Prove that the property holds for all the simple derivation trees by showing that it holds for the **axioms** of the inference system
2. **Induction step**: Prove that the property holds for all composite inference trees:
   - **Induction hypothesis**: For each **rule**, assume that the property holds for its premises
   - Prove that it also holds for the conclusion, provided that the conditions of the rule are satisfied

Induction on derivations is a special case of **well-founded induction** (derivations are finite)

4. Axiomatic Semantics

4.1 Hoare Logic

4.1.1 Proofs of Program Correctness

4.1.2 Assertion Language

4.1.3 Inference System

4.1.4 Properties of the Semantics

4.1.5 Extensions

4.2 Soundness and Completeness

Total Correctness

- The meaning of $\{ \mathrm{P} \}\ s\ \{ \Downarrow \mathrm{Q} \}$ is

> **If** P holds in the initial state σ
> **then** the execution of s from σ terminates
> **and** Q will hold in the final state

- This meaning describes total correctness, that is, termination is required

- All rules except the rule for loops are straightforward

Loop Variants

- Termination is proved using **loop variants**
- A loop variant is a function from a state to a well-founded set, for instance, $\mathbb{N}$
- Each iteration decreases the value of the loop variant
- The loop has to terminate when the minimum of the set is reached
  - Standard loop variant yields number of iterations
- Example

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

  - Possible loop variant v : State $\rightarrow \mathbb{N}$ where $v(\sigma) = \sigma(\mathsf{x})$

While Rule

- We encode the loop invariant by a parameterized family of predicates $\mathbf{V}(Z)$
  - Idea: $\mathbf{V}(Z)\sigma \Leftrightarrow v(\sigma) = Z$
- For simplicity, we require that each iteration decreases the loop variant by 1
- We have to make sure that the loop variant yields a natural number before and after each loop iteration

$$\frac{\{\ \mathcal{B}[\![b]\!] \wedge \mathbf{P} \wedge \mathbf{V}(Z+1)\ \}\ s\ \{\ \Downarrow \mathbf{P} \wedge \mathbf{V}(Z)\ \}}{\{\ \mathbf{P} \wedge \exists Z : \mathbf{V}(Z)\ \}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\ \Downarrow \neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\ \}}$$

where $Z \in \mathbb{N}$

While Rule (cont'd)

- Why do we need the precondition $\exists Z : \mathbf{V}(Z)$?

$$\frac{\{\ \mathcal{B}[\![b]\!] \wedge \mathbf{P} \wedge \mathbf{V}(Z+1)\ \}\ s\ \{\ \Downarrow \mathbf{P} \wedge \mathbf{V}(Z)\ \}}{\{\ \mathbf{P}\ \}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\ \Downarrow \neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\ \}}$$

where $Z \in \mathbb{N}$

- With $\mathbf{V}(Z) \equiv \mathsf{x} = Z$, we can derive

$$\frac{\dfrac{\{\ \mathsf{x} - 1 = Z\ \}\ \mathsf{x}\texttt{:=}\mathsf{x} - 1\ \{\ \Downarrow \mathsf{x} = Z\ \}}{\{\ \mathsf{x}\#0 \wedge \mathsf{x} = Z + 1\ \}\ \mathsf{x}\texttt{:=}\mathsf{x} - 1\ \{\ \Downarrow \mathsf{x} = Z\ \}}}{\{\ \textit{true}\ \}\ \texttt{while}\ \mathsf{x}\#0\ \texttt{do}\ \mathsf{x}\texttt{:=}\mathsf{x} - 1\ \texttt{end}\ \{\ \Downarrow \mathsf{x} = 0\ \}}$$

- This derivation is not **sound**
- We cannot prove $\exists Z \in \mathbb{N} : \mathbf{V}(Z)$ for $\mathsf{x} < 0$

Total Correctness of Factorial

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

$$\texttt{y:=1;while not x} = 1 \texttt{ do y:=y} * \texttt{x;x:=x} - 1 \texttt{ end}$$

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

- Invariant: $\mathbf{P} \equiv \mathsf{x} > 0 \wedge \mathsf{y} \times \mathsf{x}! = N!$
- Variant: $\mathbf{V}(Z) \equiv \mathsf{x} = Z$

- We verify the factorial statement and give a proof outline

Non-Recursive Procedures

$$\begin{array}{rl} \text{Stm} & = \ldots \\ & \mid \text{'}\texttt{proc}\text{'}\ p\ \text{'}\texttt{is}\text{'}\ s\ \text{'}\texttt{end}\text{'} \\ & \mid \text{'}\texttt{call}\text{'}\ p \end{array}$$

- For simplicity, we require that
  - Procedures have no parameters
  - Procedures cannot be hidden (unique procedure names)

$$\frac{\{\,\mathbf{P}\,\}\ s\ \{\,\mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\ \texttt{call}\ p\ \{\,\mathbf{Q}\,\}} \qquad \frac{\{\,\mathbf{P}\,\}\ s\ \{\,\Downarrow \mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\ \texttt{call}\ p\ \{\,\Downarrow \mathbf{Q}\,\}}$$

where p is defined by $\texttt{proc}\ p\ \texttt{is}\ s\ \texttt{end}$

Recursive Procedures

- We use the same procedures as before, but allow them to be recursive
- The Hoare rule does not work for recursive procedures

```
proc p is
  if x > 0 then
    x:=x-1; call p
  end
end;
x := 5;
call p
```

$$\frac{\frac{\{\,\mathbf{P}\,\}\ \ldots \texttt{call}\ p \ldots\ \{\,\mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\ \ldots \texttt{call}\ p \ldots\ \{\,\mathbf{Q}\,\}}}{\{\,\mathbf{P}\,\}\,\texttt{call}\ p\,\{\,\mathbf{Q}\,\}}$$

Assumptions

- To prove an assertion for the body of a procedure, we may **assume** that the assertion holds for recursive calls

$$\frac{\{\,\mathbf{P}\,\}\,\texttt{call}\;p\,\{\,\mathbf{Q}\,\} \vdash \{\,\mathbf{P}\,\}\;s\;\{\,\mathbf{Q}\,\}}{\{\,\mathbf{P}\,\}\,\texttt{call}\;p\,\{\,\mathbf{Q}\,\}}$$

where p is defined by $\texttt{proc}\;p\;\texttt{is}\;s\;\texttt{end}$

- A full treatment of assumptions requires several additional rules for
  - Adapting assumptions
  - Introducing and eliminating assumptions (mutually recursive methods)

Example

```
proc fac is
  if x = 1
    then  skip
    else  y := x * y;  x := x - 1; call fac
  end
end;
y := 1;
call fac
```

- We prove
  1. $\{ \mathsf{x} > 0 \wedge N = \mathsf{y} \times \mathsf{x}! \}$ `call fac` $\{ \mathsf{y} = N \} \vdash$
     $\{ \mathsf{x} > 0 \wedge N = \mathsf{y} \times \mathsf{x}! \}$ *body*(`fac`) $\{ \mathsf{y} = N \}$
  2. $\{ \mathsf{x} > 0 \wedge N = \mathsf{x}! \}$ `y:=`1`;call fac` $\{ \mathsf{y} = N \}$

Total Correctness

- Idea: Like loop variants, we use a function that decreases with each recursive call
- If we assume that the recursive call terminates after Z recursions, then the procedure body will terminate after $Z + 1$ recursions

$$\frac{\{\, \mathbf{P} \wedge \mathbf{V}(Z) \,\}\ \texttt{call}\ p\ \{\, \Downarrow \mathbf{Q} \,\} \vdash \{\, \mathbf{P} \wedge \mathbf{V}(Z+1) \,\}\ s\ \{\, \Downarrow \mathbf{Q} \,\}}{\{\, \mathbf{P} \wedge \exists Z : \mathbf{V}(Z) \,\}\ \texttt{call}\ p\ \{\, \Downarrow \mathbf{Q} \,\}}$$

where $Z \in \mathbb{N}$ and p is defined by `proc` p `is` s `end`

- For procedure `fac`, we could use $\mathbf{V}(Z) \equiv \mathsf{x} = Z$