# Summary
# Formal Methods

Fabian Hahn, Dino Wernli

October 4, 2009

## 1 Induction

### 1.1 Well-foundedness

A binary relation $\prec$ on a set A is **well-founded** iff there are no infinite descending chains $\ldots \prec a_i \prec \ldots \prec a_1 \prec a_0$.
The *well-founded* relations easiest to construct (and to handle) are those with a single smallest element $a_{min}$ such that the following holds:

$$\forall a \in A : a_{min} < a \Leftrightarrow a \neq a_{min} \tag{1}$$

### 1.2 Definition

Let $\prec$ be a *well-founded* relation on a set A. Let P be a property or a predicate to show. Then the following equivalence holds:

$$\begin{aligned}(\forall a \in A : ((\forall b \in A : b \prec a \rightarrow P(b)) \rightarrow P(a))) \\ \Leftrightarrow \quad \forall a \in A : P(a)\end{aligned} \tag{2}$$

Note that the inductive base case is implicitly defined by the fact that "false implies everything", e.g. the left side of the first implication arrow becomes false.

## 2 IMP Syntax

**Characters:**

$$\text{Letter} = a \mid b \mid \ldots \mid z \mid A \mid \ldots \mid Z \tag{3}$$
$$\text{Digit} = 0 \mid 1 \mid \ldots \mid 9 \tag{4}$$

**Tokens:**

$$\text{Ident} = \text{Letter}\{\text{Letter} \mid \text{Digit}\} \tag{5}$$
$$\text{Integer} = \text{Digit}\{\text{Digit}\} \tag{6}$$
$$\text{Var} = \text{Ident} \tag{7}$$

**Arithmetic expressions:**

$$\text{Aexp} = (\text{Aexp Op Aexp}) \mid \text{Var} \mid \text{Integer} \tag{8}$$
$$\text{Op} = + \mid - \mid * \mid / \mid \mathit{mod} \tag{9}$$

**Boolean expressions:**

$$\begin{aligned}\text{Bexp} = (\text{Bexp } \mathit{or} \text{ Bexp}) \mid (\text{Bexp } \mathit{and} \text{ Bexp}) \\ \mid \mathit{not} \text{ Bexp} \mid \text{Aexp RelOp Aexp}\end{aligned} \tag{10}$$
$$\text{RelOp} = \# \mid = \mid < \mid <= \mid > \mid >= \tag{11}$$

Note that the recursive base case in (10) is actually *Aexp RelOp Aexp*, which is defined recursively itself in (8).

**Statements:**

$$\begin{aligned}\text{Stm} = \mathit{skip} \mid \text{Var} := \text{Aexp} \mid \text{Stm; Stm} \\ \mid \mathit{if} \text{ Bexp } \mathit{then} \text{ Stm } \mathit{else} \text{ Stm } \mathit{end} \\ \mid \mathit{while} \text{ Bexp } \mathit{do} \text{ Stm } \mathit{end}\end{aligned} \tag{12}$$

## 3 Operational semantics

### 3.1 States

The semantic function State : Var → Val associates a **value** (usually from $\mathbb{Z}$) to each **variable** in a *statement*. Usually, a *state* $\sigma$ that maps $x_i$ to $v_i$ is represented as:

$$\sigma = \{x_1 \mapsto v_1, x_2 \mapsto v_2, \ldots, x_n \mapsto v_n\} \tag{13}$$

**Updating states:** The substitution $\sigma[y \mapsto v]$ overrides or adds $y$'s association in $\sigma$, or formally:

$$(\sigma[y \mapsto v])(x) = \begin{cases} v & \text{if } x = y \\ \sigma(x) & \text{if } x \neq y \end{cases} \tag{14}$$

### 3.2 Arithmetic evaluation

The semantic function $\mathcal{A}$ : Aexp → State → Val maps an *arithmetic expression* $e$ and a *state* $\sigma$ to a value $\mathcal{A}[\![e]\!]\sigma$:

$$\mathcal{A}[\![x]\!]\sigma = \sigma(x) \tag{15}$$
$$\forall i \in \text{Val} : \mathcal{A}[\![i]\!]\sigma = i \tag{16}$$
$$\begin{aligned}\forall e_1, e_2 \in \text{Aexp } \forall op \in \text{Op} : \\ \mathcal{A}[\![e_1 \ op \ e_2]\!]\sigma = \mathcal{A}[\![e_1]\!]\sigma \ \overline{op} \ \mathcal{A}[\![e_2]\!]\sigma\end{aligned} \tag{17}$$

Note that $\overline{op}$ denotes the actual operation Val → Val → Val corresponding to *op*.

### 3.3 Boolean evaluation

The semantic function $\mathcal{B}$ : Bexp → State → Bool maps a *boolean expression* $b$ and a *state* $\sigma$ to a truth value $\mathcal{B}[\![b]\!]\sigma$ which is either *tt* (true) or *ff* (false):

$$\mathcal{B}[\![e_1 \ relop \ e_2]\!]\sigma = \mathcal{A}[\![e_1]\!]\sigma \ \overline{relop} \ \mathcal{A}[\![e_2]\!]\sigma \tag{18}$$
$$\mathcal{B}[\![b_1 \ and \ b_2]\!]\sigma = \mathcal{B}[\![b_1]\!]\sigma \ \&\& \ \mathcal{B}[\![b_2]\!]\sigma \tag{19}$$
$$\mathcal{B}[\![b_1 \ or \ b_2]\!]\sigma = \mathcal{B}[\![b_1]\!]\sigma \ || \ \mathcal{B}[\![b_2]\!]\sigma \tag{20}$$
$$\mathcal{B}[\![not \ b]\!]\sigma = \sim \mathcal{B}[\![b]\!]\sigma \tag{21}$$

Note that $\overline{relop}$ denotes the actual relation Val → Val → Bool corresponding to *relop* and that (18) is the base case of the recursive definition, although defined inductively itself in (17).

## 3.4 Free variables

Since the basic definition of IMP does not feature local *variables*, a recursive function FV : Stm → {Var} that maps all IMP *statements* to their set of contained *free variables* can be trivially defined.

## 3.5 Configurations

A *configuration* $\langle s, \sigma \rangle$ represents the overall condition during the execution of a program, and therefore consists of a remaining *statement* $s$ to execute and a *state* $\sigma$. If $s$ is empty, the *configuration* is simply written as $\sigma$ and is called **final**, otherwise **non-final**.

## 3.6 Natural semantics

*Natural semantics* or **big-step semantics** describe how the overall results of an execution are obtained.

### 3.6.1 Transitions

The transition relation $\rightarrow$ of *natural IMP semantics* completely executes the *statement* of a *configuration* and therefore always results in a *final configuration*.

### 3.6.2 Axioms

The following axioms can be used to prove the execution of a program using natural deduction.

**Skip:**

$$\frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} \tag{22}$$

**Assignment:**

$$\frac{}{\langle x := e, \sigma \rangle \rightarrow \sigma \left[ x \mapsto \mathcal{A}[\![e]\!]\sigma \right]} \tag{23}$$

**Sequential composition:**

$$\frac{\langle s_1, \sigma \rangle \rightarrow \sigma' \quad \langle s_2, \sigma' \rangle \rightarrow \sigma''}{\langle s_1; s_2, \sigma \rangle \rightarrow \sigma''} \tag{24}$$

**Conditionals:**

$$\frac{\langle s_1, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow \sigma'} \text{ if } \mathcal{B}[\![b]\!]\sigma = tt \tag{25}$$

$$\frac{\langle s_2, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma \rangle \rightarrow \sigma'} \text{ if } \mathcal{B}[\![b]\!]\sigma = ff \tag{26}$$

**Loops:**

$$\frac{\langle s, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } s \text{ end}, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma''} \text{ if } \mathcal{B}[\![b]\!]\sigma = tt \tag{27}$$

$$\frac{}{\langle \text{while } b \text{ do } s \text{ end}, \sigma \rangle \rightarrow \sigma} \text{ if } \mathcal{B}[\![b]\!]\sigma = ff \tag{28}$$

### 3.6.3 Termination

The execution of a *statement* $s$ in *state* $\sigma$ either:

- **terminates** iff $\exists \sigma' : \langle s, \sigma \rangle \rightarrow \sigma'$
- **is stuck** iff there is no possible transition (e.g. due to incorrect syntax or looping)

### 3.6.4 Semantic equivalence

Two statements $s_1$ and $s_2$ are *semantically equivalent* iff the following holds for all *states* $\sigma, \sigma'$:

$$\langle s_1, \sigma \rangle \rightarrow \sigma' \Leftrightarrow \langle s_2, \sigma \rangle \rightarrow \sigma' \tag{29}$$

### 3.6.5 Extensions

**Abort:** This extension defines an "abort" *statement* with no associated axioms. This makes sure all *configurations* $\langle \text{abort}, \sigma \rangle$ are stuck and cannot continue. Note that with this definition, the statements "while true do skip end" and "abort" are *semantically equivalent*.

**Non-determinism:**

$$\frac{\langle s_1, \sigma \rangle \rightarrow \sigma'}{\langle s_1 \square s_2, \sigma \rangle \rightarrow \sigma'} \tag{30}$$

$$\frac{\langle s_2, \sigma \rangle \rightarrow \sigma'}{\langle s_1 \square s_2, \sigma \rangle \rightarrow \sigma'} \tag{31}$$

Note that *natural semantics* always choose the "right" branch of a non-deterministic choice and will supress looping if possible.

**Parallelism:** Cannot be expressed in *natural semantics* since it would require rules to interleave atomic execution steps.

## 3.7 Structural operational semantics

*Structural operational semantics* or **small-step semantics** describe the executions of individual steps in order.

### 3.7.1 Transitions

The transition relation $\rightarrow_1$ of *structural operational IMP semantics* executes one step of the *statement* of a *configuration* and may therefore result in a *non-final configuration*. The notation $\alpha \rightarrow_1^i \beta$ may be used to indicate that there were $i$ steps in the execution from *configuration* $\alpha$ to $\beta$, $\alpha \rightarrow_1^* \beta$ denotes a finite number of steps from $\alpha$ to $\beta$.

### 3.7.2 Axioms

The following axioms can be used to prove one execution step of a program using natural deduction.

**Skip:**

$$\frac{}{\langle \text{skip}, \sigma \rangle \rightarrow_1 \sigma} \tag{32}$$

**Assignment:**

$$\frac{}{\langle x := e, \sigma \rangle \rightarrow_1 \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]} \tag{33}$$

**Sequential composition:**

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \sigma'}{\langle s_1; s_2, \sigma \rangle \rightarrow_1 \langle s_2, \sigma' \rangle} \tag{34}$$

$$\frac{\langle s_1, \sigma \rangle \rightarrow_1 \langle s_1', \sigma' \rangle}{\langle s_1; s_2, \sigma \rangle \rightarrow_1 \langle s_1'; s_2, \sigma' \rangle} \tag{35}$$

**Conditionals:**

$$\frac{}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma\rangle \rightarrow_1 \langle s_1, \sigma\rangle} \text{ if } \mathcal{B}[\![b]\!]\sigma = tt \quad (36)$$

$$\frac{}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}, \sigma\rangle \rightarrow_1 \langle s_2, \sigma\rangle} \text{ if } \mathcal{B}[\![b]\!]\sigma = ff \quad (37)$$

**Loops:**

$$\frac{}{\langle \text{while } b \text{ do } s \text{ end}, \sigma\rangle \rightarrow_1 \langle s;\ \text{while } b \text{ do } s \text{ end}, \sigma\rangle} \quad (38)$$
$$\text{if } \mathcal{B}[\![b]\!]\sigma = tt$$

$$\frac{}{\langle \text{while } b \text{ do } s \text{ end}, \sigma\rangle \rightarrow_1 \sigma} \text{ if } \mathcal{B}[\![b]\!]\sigma = ff \quad (39)$$

#### 3.7.3 Termination

The execution of a *statement* $s$ in *state* $\sigma$ either:

- **terminates** iff $\exists \sigma' : \langle s, \sigma'\rangle \rightarrow_1^* \sigma'$
- **loops** iff there is an infinite derivation sequence starting with $\langle s, \sigma\rangle$
- **is stuck** iff there is no possible transition (e.g. due to incorrect syntax)

#### 3.7.4 Semantic equivalence

Two statements $s_1$ and $s_2$ are *semantically equivalent* iff the following two properties hold for all *states* $\sigma$:

- For all *stuck* or *terminal configurations* $\gamma$:

$$\langle s_1, \sigma\rangle \rightarrow_1^* \gamma \Leftrightarrow \langle s_2, \sigma\rangle \rightarrow_1^* \gamma \quad (40)$$

- There exists an infinite derivation tree from $\langle s_1, \sigma\rangle$ iff there exists one from $\langle s_2, \sigma\rangle$ as well.

#### 3.7.5 Extensions

**Abort:** This extension defines an "abort" *statement* with no associated axioms. This makes sure all *configurations* $\langle \text{abort}, \sigma\rangle$ are stuck and cannot continue. Note that with this definition, the statements "while true do skip end" and "abort" are not *semantically equivalent* since the first one loops and the second one is stuck!

**Non-determinism:**

$$\frac{}{\langle s_1 \square s_2, \sigma\rangle \rightarrow_1 \langle s_1, \sigma\rangle} \quad (41)$$

$$\frac{}{\langle s_1 \square s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma\rangle} \quad (42)$$

Note that *structural operational semantics* may choose the "wrong" branch of a non-deterministic choice and may loop or get stuck.

**Parallelism:**

$$\frac{\langle s_1, \sigma\rangle \rightarrow_1 \sigma'}{\langle s_1 \text{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_2, \sigma'\rangle} \quad (43)$$

$$\frac{\langle s_1, \sigma\rangle \rightarrow_1 \langle s_1', \sigma'\rangle}{\langle s_1 \text{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_1' \text{ par } s_2, \sigma'\rangle} \quad (44)$$

$$\frac{\langle s_2, \sigma\rangle \rightarrow_1 \sigma'}{\langle s_1 \text{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_1, \sigma'\rangle} \quad (45)$$

$$\frac{\langle s_2, \sigma\rangle \rightarrow_1 \langle s_2', \sigma'\rangle}{\langle s_1 \text{ par } s_2, \sigma\rangle \rightarrow_1 \langle s_1 \text{ par } s_2', \sigma'\rangle} \quad (46)$$

### 3.8 Determinism and Equivalence

The following two properties hold:

- Both *natural semantics* and *structural operational semantics* without extensions are deterministic.
- For every derivation that reaches a *final configuration*, *natural semantics* ($\rightarrow$) and *structural operational semantics* ($\rightarrow_1^*$) are equivalent.

## 4 Axiomatic semantics

### 4.1 Types of correctness

Semantics can be used to prove two different types of correctness:

- **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.

### 4.2 Hoare logic

#### 4.2.1 Predicates

A *predicate* is a function State $\rightarrow$ Bool usually defined by a *boolean expression* $b$'s *evaluation* $\mathcal{B}[\![b]\!](\sigma)$.
Let $\mathbf{P}$, $\mathbf{P}_1$ and $\mathbf{P}_2$ be *predicates*. Then the following rules can be used to combine them:

$$(\mathbf{P}_1 \wedge \mathbf{P}_2)(\sigma) := \mathbf{P}_1(\sigma) \wedge \mathbf{P}_2(\sigma) \quad (47)$$

$$(\mathbf{P}_1 \vee \mathbf{P}_2)(\sigma) := \mathbf{P}_1(\sigma) \vee \mathbf{P}_2(\sigma) \quad (48)$$

$$(\neg \mathbf{P})(\sigma) := \neg \mathbf{P}(\sigma) \quad (49)$$

$$(\mathbf{P}\,[x \mapsto \mathcal{A}[\![e]\!]])\,(\sigma) := \mathbf{P}\,(\sigma\,[x \mapsto \mathcal{A}[\![e]\!]\sigma]) \quad (50)$$

$$(\mathbf{P}_1 \Rightarrow \mathbf{P}_2)\,(\sigma) := \mathbf{P}_1(\sigma) \Rightarrow \mathbf{P}_2(\sigma) \quad (51)$$

#### 4.2.2 Assertions

Let $s$ be a *statement* and $\mathbf{P}$ and $\mathbf{Q}$ predicates. Then $\{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$ is called an *assertion* or a **Hoare triple**. The predicate $\mathbf{P}$ is called **precondition** and $Q$ is called **postcondition**.
The meaning of $\{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$ is that if $P$ holds in an initial *state* $\sigma$ and if the execution of $s$ terminates in a *state* $\sigma'$, then $Q$ will hold in $\sigma'$. Note that this meaning describes *partial correctness*.
If we can prove a *Hoare triple* $\{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$, we write $\vdash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$.

**Weakest precondition:** The *weakest precondition* $wp(s, \mathbf{Q})$ for a *statement* $s$ and a *postcondition* $\mathbf{Q}$ is the weakest predicate that still guarantees $\mathbf{Q}$ and program termination after the execution of $s$:

$$wp(s, \mathbf{Q})\sigma = tt \Leftrightarrow \exists \sigma' : (\langle s, \sigma\rangle \rightarrow \sigma') \wedge \mathbf{Q}(\sigma') \quad (52)$$

The *weakest precondition* that doesn't guarantee termination is called **liberal** and is defined as follows:

$$wlp(s, \mathbf{Q})\sigma = tt \Leftrightarrow \forall \sigma' : (\langle s, \sigma\rangle \rightarrow \sigma') \Rightarrow \mathbf{Q}(\sigma') \quad (53)$$

#### 4.2.3 Logical variables

In order to be able to refer to *variable* mappings from the initial *state* in a *postcondition*, *assertions* may contain *logical variables*, usually denoted by capital letters. They can only occur in *pre-* and *postconditions* and may not be modified by the program.

#### 4.2.4 Inference axioms

The following axioms can be used to prove *partial correctness* of a program:

**Skip:**

$$\frac{}{\{\mathbf{P}\}\,\text{skip}\,\{\mathbf{P}\}} \tag{54}$$

**Assignment:**

$$\frac{}{\{\mathbf{P}\,[x \mapsto \mathcal{A}[\![e]\!]]\}\,x := e\,\{\mathbf{P}\}} \tag{55}$$

**Sequential composition:**

$$\frac{\{\mathbf{P}\}\,s_1\,\{\mathbf{R}\} \quad \{\mathbf{R}\}\,s_2\,\{\mathbf{Q}\}}{\{\mathbf{P}\}\,s_1; s_2\,\{\mathbf{Q}\}} \tag{56}$$

**Conditionals:**

$$\frac{\{\mathcal{B}[\![b]\!] \wedge \mathbf{P}\}\,s_1\,\{\mathbf{Q}\} \quad \{\neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\}\,s_2\,\{\mathbf{Q}\}}{\{\mathbf{P}\}\,\text{if } b \text{ then } s_1 \text{ else } s_2 \text{ end}\,\{\mathbf{Q}\}} \tag{57}$$

**Loops:**

$$\frac{\{\mathcal{B}[\![b]\!] \wedge \mathbf{P}\}\,s\,\{\mathbf{P}\}}{\{\mathbf{P}\}\,\text{while } b \text{ do } s \text{ end}\,\{\neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\}} \tag{58}$$

The *predicate* $\mathbf{P}$ in this axiom is called loop invariant.

**Consequence:**

$$\frac{\{\mathbf{P'}\}\,s\,\{\mathbf{Q'}\}}{\{\mathbf{P}\}\,s\,\{\mathbf{Q}\}} \text{ if } \mathbf{P} \Rightarrow \mathbf{P'} \text{ and } \mathbf{Q'} \Rightarrow \mathbf{Q} \tag{59}$$

In other terms, to prove $\{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$, we may strengthen the *precondition* $\mathbf{P}$ and weaken the *postcondition* $\mathbf{Q}$.

#### 4.2.5 Semantic equivalence

Two *statements* $s_1$ and $s_2$ are **provably equivalent** iff the following holds for all *preconditions* $\mathbf{P}$ and *postconditions* $\mathbf{Q}$:

$$\vdash \{\mathbf{P}\}\,s_1\,\{\mathbf{Q}\} \quad \Leftrightarrow \quad \vdash \{\mathbf{P}\}\,s_2\,\{\mathbf{Q}\} \tag{60}$$

#### 4.2.6 Extension: Total correctness

This extension defines the triple $\{\mathbf{P}\}\,s\,\{\Downarrow \mathbf{Q}\}$ of which the meaning is that if $\mathbf{P}$ holds in an initial *state* $\sigma$ then the execution of $s$ from $\sigma$ terminates in $\sigma'$ and $\mathbf{Q}$ will hold in *state* $\sigma'$. Note that this meaning describes *total correctness*.
All *inference axioms* except for the while loop are straightforward to define. For the while loop, the notion of a **loop variant** is needed, which is defined as a function from a *state* to a *well-founded* set, e.g. $\mathbb{N}$. The idea is now to show that the *variant* holds at the beginning of the execution and decreses with each loop iteration. A *variant* $v$ is usually encoded into the parameterized *predicate* $\mathbf{V}(Z) := v(\sigma) = Z$. Using this notation, an axiom for the *well-founded* set $\mathbb{N}$ can be defined as follows:

$$\frac{\{\mathcal{B}[\![b]\!] \wedge \mathbf{P} \wedge \mathbf{V}(Z+1)\}\,s\,\{\Downarrow \mathbf{P} \wedge \mathbf{V}(Z)\}}{\{\mathbf{P}\}\,\text{while } b \text{ do } s \text{ end}\,\{\Downarrow \neg\mathcal{B}[\![b]\!] \wedge \mathbf{P}\}} \text{ if } \mathbf{P} \Rightarrow \exists Z : \mathbf{V}(Z) \tag{61}$$

#### 4.2.7 Derivability

If a theorem $\mathbf{T}$ can be proven in a logic system, we write $\vdash \mathbf{T}$. $\mathbf{T}$ is then called *derivable* in the system.
If a *hoare triple* $\{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$ is provable, we therefore write $\vdash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\}$.

#### 4.2.8 Validity

If a theorem $\mathbf{T}$ is actually true, we write $\vDash \mathbf{T}$. $\mathbf{T}$ is then called *valid*.
In *hoare logic*, this can be expressed as:

$$\vDash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\} \Leftrightarrow \forall\sigma\forall\sigma' : \mathbf{P}(\sigma) \wedge (\langle s, \sigma\rangle \rightarrow \sigma') \Rightarrow \mathbf{Q}(\sigma') \tag{62}$$

#### 4.2.9 Soundness

A logic system is called *sound* if everything that can be proven actually holds. It therefore doesn't make any sense do design an *unsound* logic system.
In *hoare logic*, *soundness* can be expressed as:

$$\vdash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\} \quad \Rightarrow \quad \vDash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\} \tag{63}$$

*Hoare logic* for *IMP natural semantics* without extensions can be proven to be *sound*.

#### 4.2.10 Completeness

A logic system is called *complete* if everything that holds can be proven in the system. It can make sense so have an *incomplete* system, for example if only a subset of holding theorems need to be proven in practice.
In *hoare logic*, *completeness* can be expressed as:

$$\vDash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\} \quad \Rightarrow \quad \vdash \{\mathbf{P}\}\,s\,\{\mathbf{Q}\} \tag{64}$$

*Hoare logic* for *IMP natural semantics* without extensions can be proven to be *complete*.

# 5 Linear temporal logic

## 5.1 Infinite sequences

Let $S$ be a finite set. Then $S^\omega$ is the set of *infinite sequences* consisting of elements of $S$. Note that the set $S^\omega$ is also infinite itself.
We write $s_i$ to denote the $i$-th element of of the *inifinite sequence* $s \in S^\omega$.

## 5.2 Transition systems

A *transition system* is a triple $(\Gamma, \gamma_I, \rightarrow)$, where:

- $\Gamma$ is a finite set of *configurations*
- $\gamma_I$ is an initial *configuration*
- $\rightarrow \subseteq \Gamma \times \Gamma$ is a transition relation

## 5.3 Computations

Let $TS = (\Gamma, \gamma_I, \rightarrow)$ be a *transition system*. Then the *infinite sequence* $\gamma \in \Gamma^\omega$ of *configurations* is called a *computation* of $TS$ if the following two properties hold:

$$\gamma_0 = \gamma_I \tag{65}$$

$$\forall i \geq 0 : \gamma_i \rightarrow \gamma_{i+1} \tag{66}$$

We write $\mathcal{C}(TS)$ to denote the set of all *computations* of $TS$.

## 5.4 Linear-time properties

Let $TS = (\Gamma, \gamma_I, \rightarrow)$ be a *transition system*. A *linear-time property* $P$ over $\Gamma$ is a subset of $\Gamma^\omega$, i.e. $P \subseteq \Gamma^\omega$.
We say that $TS$ **satisfies** $P$, or formally $TS \vDash P$, iff $\mathcal{C}(TS) \subseteq P$ i.e. if all *computations* of the *transition system* are part of the *linear-time property*. All *computations* $\gamma \in P$ are called **admissable**.

## 5.5 Atomic propositions

A set of *atomic propositions* $AP$ can be seen as a set of flags or labels to be mapped to *configurations* of a *transition system* $TS = (\Gamma, \gamma_I, \rightarrow)$ by means of a **labeling function** $L : \Gamma \rightarrow \mathcal{P}(AP)$.
The *atomic propositions* $L(\gamma_i)$ of a *configuration* $\gamma_i$ are called its **abstract state**.

## 5.6 Traces

A *trace* $t$ is an abstraction of a *computation* $\gamma$ using a *labeling function* $L$ to encode its *configurations* $\gamma_i$:

$$t = L(\gamma_0)L(\gamma_1)L(\gamma_2)\dots \tag{67}$$

As such, a *trace* can be seen as an infinite sequence from the set $\mathcal{P}(AP)^\omega$. For a *transition system* $TS$, we write $\mathcal{T}(TS)$ to denote the set of all its *traces*. For a *linear-time property* $P$, $L(P)$ denotes the set of all *traces* for the *computations* $\gamma \in P$. We write $t_{\geq k}$ to denote the *trace* starting from the $k$-th *configuration* of $t$.

## 5.7 Safety properties

In a *transition system* $TS$ with *labeling function* $L$, a *linear-time property* $P$ is called a *safety property* iff for all *traces* $t \notin L(P)$ there exists a finite prefix $\hat{t}$ of $t$ such that for every trace $t'$ with that prefix $\hat{t}$, $t' \notin L(P)$ holds. $\hat{t}$ is called **bad prefix**.
Intuitively, this means that *safety properties* are violated in finite time and cannot be repaired.
A *safety property* is called **regular** if its *bad prefixes* can be described by a regular language.

## 5.8 Liveness property

In a *transition system* $TS$ with *labeling function* $L$, a *linear-time property* $P$ is called a *liveness property* iff every finite sequence $\hat{t} \in \mathcal{P}(AP)^*$ is a prefix of a *trace* $t \in L(P)$.
Intuitively, this means that *liveness properties* are violated in infinite time and do not rule out any prefix.

## 5.9 LTL formulas

Let $TS$ be a *transition system* with *labeling function* $L$.

**Syntax:** An *LTL formula* $\phi$ can be composed as follows:

$$\phi = p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \cup \phi \mid \bigcirc \phi \quad \text{where } p \in AP \tag{68}$$

**Semantics:** For a *trace* $t \in \mathcal{T}(TS)$, $t \vDash \phi$ expresses that $t$ **satisfies** an *LTL formula* $\phi$:

$$t \vDash p \quad \text{iff } p \in t_0 \tag{69}$$

$$t \vDash \neg\phi \quad \text{iff } \neg(t \vDash \phi) \tag{70}$$

$$t \vDash \phi \wedge \psi \text{ iff } (t \vDash \phi) \wedge (t \vDash \psi) \tag{71}$$

$$t \vDash \phi \cup \psi \text{ iff } \exists\, k \geq 0 \colon (t_{\geq k} \vDash \psi) \wedge \forall\, 0 \leq j < k \colon t_{\geq j} \vDash \phi \tag{72}$$

$$t \vDash \bigcirc\phi \quad \text{iff } t_{\geq 1} \vDash \phi \tag{73}$$

Intuitively, $\phi \cup \psi$ means that the formula $\phi$ holds until $\psi$ holds. $\vDash \bigcirc\phi$ means that in the next *configuration*, $\phi$ will hold.
From the above definition, more operators can be derived:

- $true, false, \vee, \Rightarrow$: as usual
- $\Diamond\phi \equiv true \cup \phi$: eventually, $\phi$ will hold
- $\Box\phi \equiv \neg\Diamond\neg\phi$: generally, $\phi$ holds

**Patterns:** The following formula patterns are frequently used:

- **Strong invariant**: $\Box\phi$
  $\phi$ does always hold (*safety property*)
- **Monotone invariant**: $\Box(\phi \rightarrow \Box\phi)$
  Once $\phi$ holds, it will always hold (*safety property*)
- **Establishing invariant**: $\Diamond\Box\phi$
  At some point, $\phi$ will start to always hold (*liveness property*)
- **Responsiveness**: $(\phi \Rightarrow \Diamond\psi)$
  Once $\phi$ holds, $\psi$ will hold at some point in the future (*liveness property*)
- **Fairness**: $\Box\Diamond\phi$
  $\phi$ will hold infinitely often (*liveness property*)

## 5.10 LTL model checking of regular safety properties

To prove a *regular safety property* $P$ of a *transition system* $TS$ with *labeling function* $L$, follow these steps:

1. Construct a finite automaton $FA_{TS}$ that accepts all finite prefixes of $TS$
2. Construct a finite automaton $FA_{\overline{P}}$ that accepts all bad prefixes of $P$
3. Construct a finite automaton $FA_\cap$ that accepts the language $\mathcal{L}(FA_{TS}) \cap \mathcal{L}(FA_{\overline{P}})$
4. Search the possible transitions of $FA_\cap$ from its initial state for an accepting state. If no accepting state can be reached, $P$ holds

## 5.11 Büchi automata

A *Büchi automaton* is a finite automaton with the difference that it accepts infinite words if they pass infinitely often through an accepting state. They can be used to prove *liveness properties*.