Specification and Verification with Higher-Order Logic
Vorlesung SS 2014

Prof. Dr. A. Poetzsch-Heffter
AG Softwaretechnik
TU Kaiserslautern

© Arnd Poetzsch-Heffter et al. TU Kaiserslautern
Chapter 0

Preliminaries
Overview of Chapter

0. Preliminaries
0.1 Organisation
0.2 Course Overview
Section 0.1

Organisation
Contact

- Arnd Poetzsch-Heffter
- Peter Zeller
- Christoph Feller
- Information about course: [http://softech.informatik.uni-kl.de/](http://softech.informatik.uni-kl.de/)
- Wiki for the course and Isabelle/HOL: [http://svhol.pbmichel.de/](http://svhol.pbmichel.de/)
0. Preliminaries

0.1 Organisation

Dates, Time, and Location

- 3C + 3R (8 ECTS-LP)
- Monday, 11:45-13:15, room 48-462 (Lecture)
- Wednesday, 11:45-13:15, room 32-411 (Exercises)
- Thursday, 11:45-13:15, room 48-462/32-411 (Lecture/Exercises)

Exams

- Oral
- Topics: content of lecture and exercises
- Dates: after lecture period; dates will be announced

© Arnd Poetzsch-Heffter et al.
TU Kaiserslautern
Literature

Further reading


Further reading (2)


Further reading (3)


Further reading (4)


Acknowledgements

- Dr. Jens Brandt for designing several of the slides
- Prof. Madlener for designing further parts of this course material
- Prof. Basin, Dr. Brucker, Dr. Smaus, Prof. Wolff, and the MMISS-project for the slides on CSMR
- Prof. Nipkow for the slides on Isabelle/HOL.
- Isabelle/HOL community for providing tools and theories
0. Preliminaries 0.2 Course Overview

Section 0.2

Course Overview
 Topics and learning objectives

• Functional programming and modeling of software systems
• Higher-order logic
• Formal verification in Isabelle/HOL (and other theorem provers)
• Verification of algorithms
• Modeling and verification of transition systems
• Specification of programming languages
• Program logics and program verification
• Beyond interactive theorem proving
Course structure

1. Introduction
2. Functional programming and modeling
3. Foundations of higher-order logic
4. A proof system for higher-order logic
5. Verifying functions
6. Inductive definitions and fixed points
7. Programming language semantics
8. Program verification
Chapter 1

Introduction
Overview of Chapter

1. Introduction

1.1 Language: Syntax and Semantics
   Syntax
   Semantics

1.2 Proof Systems/Logical Calculi
   Hilbert Calculus
   Natural Deduction

1.3 Specification and Verification in Software Engineering

1.4 Summary
Goals of introduction

- Motivation for the topics
- Terminology: Specification, verification, logic
- Relation to other courses
- Review/introduce basic concepts in logic:
  1. Language: Syntax and semantics
  2. Proof systems
     2.1 Hilbert style proof systems
     2.2 Proof system for natural deduction
Section 1.1

Language: Syntax and Semantics
Subsection 1.1.1

Syntax
Aspects of syntax

- *used to designate things and express facts*
- syntax of terms and formulas: constructed from variables and function symbols
- function symbols map a tuple of terms to another term
- constant symbols: no arguments
  - constant can be seen as functions with zero arguments
- predicate symbols are considered as boolean functions
- sets of variables
Syntax (2)

Example (Natural Numbers)

- constant symbol: 0
- function symbol suc : \( \mathbb{N} \to \mathbb{N} \)
- function symbol plus : \( \mathbb{N} \times \mathbb{N} \to \mathbb{N} \)
- function symbol ...
Syntax of propositional logic

Example (Symbols)

- $\mathcal{V} = \{a, b, c, \ldots\}$ is a set of propositional variables
- two function symbols: $\neg$ and $\rightarrow$

Example (Language)

- each $p \in \mathcal{V}$ is a formula
- if $\phi$ is a formula, then $\neg \phi$ is a formula
- if $\phi$ and $\psi$ are formulas, then $\phi \rightarrow \psi$ is a formula
1. Introduction 1.1 Language: Syntax and Semantics

Syntactic sugar

Purpose

• extensions to the language that do not affect its expressiveness
• simplify the description in practice

Example

Abbreviations in propositional logic

• $True$ denotes $\phi \rightarrow \phi$
• $False$ denotes $\neg True$
• $\phi \lor \psi$ denotes $(\neg \phi) \rightarrow \psi$
• $\phi \land \psi$ denotes $\neg((\neg \phi) \lor (\neg \psi))$
• $\phi \leftrightarrow \psi$ denotes $((\phi \rightarrow \psi) \land (\psi \rightarrow \phi))$
Semantics
Semantics

Purpose

- syntax only specifies the structure of terms and formulas
- semantics assigns a meaning to symbols, terms, and formulas
- semantics is often based on variable assignments, i.e., mappings that assign a value to all free variables
  - e.g., in propositional logic, variables are assigned a truth value

Bottom-up definition

- assignments give variables a value
- terms/formulas are evaluated based on the meaning of the function symbols
Notation:
$\mathcal{D}_{bool}$ denotes the domain of boolean values, $\mathcal{D}_{bool} = \{\text{true}, \text{false}\}$.

Example (Variable assignment in propositional logic)
A variable assignment $\rho$ in propositional logic is a mapping

- $\rho : \mathcal{V} \rightarrow \mathcal{D}_{bool}$
Example (Semantics of propositional formulas)

Let $\mathcal{I}$ be the standard interpretation of $\neg$ and $\rightarrow$, i.e.,

\[
\begin{array}{cc}
\mathcal{I}(\neg) & \mathcal{I}(\rightarrow) \\
\text{false} & \text{true} & \text{false} \\
\text{true} & \text{false} & \text{false} & \text{true}
\end{array}
\]

The semantics of propositional formulas is defined by the function $\text{sem}$ that takes a variable assignment $\rho$ and a formula:

- $\text{sem}\rho\ p = \rho(p)$ for $p \in V$
- $\text{sem}\rho\ (\neg\phi) = \mathcal{I}(\neg)(\text{sem}\rho\ \phi)$
- $\text{sem}\rho\ (\phi \rightarrow \psi) = \mathcal{I}(\rightarrow)(\text{sem}\rho\ \phi, \text{sem}\rho\ \psi)$
Definition (Validity of propositional formulas)

- a formula $\phi$ is valid w.r.t. an assignment $\rho$ if $\text{sem}_\rho \phi = \text{true}$
- a formula $\phi$ is a tautology if it is valid w.r.t. all assignments $\rho$
- Notations: $\rho \models \phi$ and $\models \phi$

Example (Tautology in propositional logic)

- $\phi \equiv p \lor \neg p$ is a tautology:
  - $\rho(p) = \text{false}: \text{sem}_\rho (p \lor \neg p) = \text{true}$
  - $\rho(p) = \text{true}: \text{sem}_\rho (p \lor \neg p) = \text{true}$
Section 1.2

Proof Systems/Logical Calculi
Introduction

General Concept

Fundamental principle of logic: “Establish truth by calculation”

- purely syntactical manipulations based on transformation rules
- starting point: set of formulas $Γ$, often a given set of axioms
- deriving new formulas by deduction rules from given formulas $Γ$
- $φ$ is provable from $Γ$ if $φ$ can be obtained by a finite number of derivation steps assuming the formulas in $Γ$
- notation: $Γ ⊢ φ$ means $φ$ is provable from $Γ$
- notation: $⊢ φ$ means $φ$ is provable from a given set of axioms
1. Introduction 1.2 Proof Systems/Logical Calculi

Styles of proof systems

Hilbert style

- easy to understand
- hard to use

Natural deduction style

- easy to use
- harder to learn
- ...
Subsection 1.2.1

Hilbert Calculus
Hilbert-style deduction rules

Definition (Deduction rule)

- deduction rule $d$ is a $n + 1$-tuple

\[
\frac{\phi_1 \quad \cdots \quad \phi_n}{\psi}
\]

- formulas $\phi_1 \ldots \phi_n$, called premises of rule
- formula $\psi$, called conclusion of rule
Hilbert-style proofs

Definition (Proof)

- let $D$ be a set of deduction rules, including the axioms as rules without premisses
- *proofs* in $D$ are trees such that
  - axioms are proofs
  - if $P_1, \ldots, P_n$ are proofs with roots $\phi_1 \ldots \phi_n$ and
    $$
    \frac{\phi_1 \cdots \phi_n}{\psi}
    $$
    is in $D$, then
    $$
    \frac{P_1 \cdots P_n}{\psi}
    $$
    is a proof in $D$
- can also be written in a line-oriented style
Hilbert-style deduction rules

Axioms

- let $\Gamma$ be a set of axioms, $\psi \in \Gamma$, then $\psi$ is a proof
- axioms allow to construct trivial proofs

Modus Ponens

- Rule example: $\frac{\phi \rightarrow \psi \quad \phi}{\psi}$
- if $\phi \rightarrow \psi$ and $\phi$ have already been proven, $\psi$ can be deduced
Definition (Axioms of propositional logic)

All instantiations of the following schemas by arbitrary propositional formulas $\phi, \chi, \psi$ are axioms:

- $\phi \rightarrow (\chi \rightarrow \phi)$
- $(\phi \rightarrow (\chi \rightarrow \psi)) \rightarrow ((\phi \rightarrow \chi) \rightarrow (\phi \rightarrow \psi))$
- $\neg \chi \rightarrow \neg \phi \rightarrow ((\neg \chi \rightarrow \phi) \rightarrow \chi)$

Remark: Thus, there are infinitely many axioms.
Example (Hilbert proof)

- Language formed with the four propositional variables $p$, $q$, $r$, $s$
- Proof: $p \rightarrow p$

Let

$$
\begin{align*}
\psi_1 & \equiv (p \rightarrow ((p \rightarrow p) \rightarrow p)) \rightarrow ((p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p)) \\
\psi_2 & \equiv (p \rightarrow (p \rightarrow p)) \\
\psi_3 & \equiv (p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p)
\end{align*}
$$

\[
\begin{array}{c}
\psi_1 \\
\psi_2 \\
\hline
\psi_3 \\
\hline
p \rightarrow (p \rightarrow p) \\
\hline
(p \rightarrow p)
\end{array}
\]
Natural Deduction
Motivation

- introducing a hypothesis is a natural step in a proof
- Hilbert proofs do not permit this directly
  - can be only encoded by using $\rightarrow$
  - proofs are much longer and not very natural

Natural deduction

- proof style in which introduction of a hypothesis is a deduction rule
- deduction step can modify not only the proven propositions but also the assumptions $\Gamma$
Natural deduction

Definition (Natural deduction rule)

- deduction rule \( d \) is a \( n + 1 \)-tuple

\[
\begin{array}{c}
\Gamma_1 \vdash \phi_1 \quad \cdots \quad \Gamma_n \vdash \phi_n \\
\hline
\Gamma \vdash \psi
\end{array}
\]

- pairs of \( \Gamma \) (set of formulas) and \( \phi \) (formulas): sequents
- proof: tree of sequents with rule instantiations as nodes
Natural deduction

Discussion

- rich set of rules
- *elimination rules*: eliminate a logical symbol from a premise
- *introduction rules*: introduce a logical symbol into the conclusion
- reasoning from assumptions
**Definition (Natural deduction rules for propositional logic)**

**∨-introduction**

\[
\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \lor \psi}
\]

**∨-elimination**

\[
\begin{align*}
\Gamma \vdash \phi \lor \psi \\
\Gamma, \phi \vdash \xi \\
\Gamma, \psi \vdash \xi \\
\hline
\Gamma \vdash \xi
\end{align*}
\]

**→-introduction**

\[
\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi}
\]

**→-elimination**

\[
\frac{\Gamma \vdash \phi \rightarrow \psi \\
\Gamma \vdash \phi}{\Gamma \vdash \psi}
\]

**Assumption**

\[
\frac{}{\Gamma, \phi \vdash \phi}
\]
Example (Natural deduction proof)

- Language formed with the four proposition symbols $p$, $q$, $r$, $s$
- Proof: $p \rightarrow p$ by assumption and $\rightarrow$-introduction:

\[
\frac{p \vdash p}{\vdash p \rightarrow p}
\]
Motivation

- Specifications: Models and properties $\leadsto$ Spec-formalisms
- How do we express/specify facts? $\leadsto$ Languages
- What is a proof? What is a formal proof? $\leadsto$ Logical calculus
- How do we prove a specified fact? $\leadsto$ Proof search
- Why formal? What is the role of a theorem prover? $\leadsto$ Tools
Role of formal specifications

- Software and hardware systems must accomplish well defined tasks (requirements).
- Software engineering has as goal
  - Definition of criteria for the evaluation of SW systems
  - Methods and techniques for the development of SW systems that accomplish such criteria
  - Characterization of SW systems
  - Development processes for SW systems
  - Measures and supporting tools
- Simplified view of a SD process:
  Definition of a sequence of actions and descriptions for the SW system to be developed. Process- and product models
  
  **Goal:** A family of documents including the executable programs
Relation of specifications

Specifications

- actual needs
- informal

Validation (Test)

Specification

- formal Specification
  - Verification
  - Validation

Temporary specification

- Verification
  - Refinement

last formal Specification

- Verification of the program correctness
  - Coding Generation
  - Installation

Final System

Maintenance

Programs
Remarks

Development steps

- First specification: **Global specification**
  - Basis for the development
  - “Contract or Agreement” between developers and client

- Intermediate (partial) specifications:
  Basis of the communication between developers

- Programs: Final products

Development paradigms

- Model-driven architecture
- Object-oriented design + program
- Transformation methods
- ...
Properties of specifications

**Consistency**  
- **Validation** of the global specification regarding the requirements  
- **Verification** of intermediate specifications regarding the previous one  
- **Verification** of the programs regarding the specification  
- **Verification** of integrated final system w.r.t. to global specification  
- **Activities**: Validation, verification, testing, consistency, and completeness check  
- **Tool support** needed!

**Completeness**
Requirements

- **The global specification** describes, as exact as possible, the properties of the overall system

- **Abstraction of the how**

  **Advantages**
  - *aposteriori*: Possibility to follow and document design decisions $\leadsto$ **traceability, reusability, maintenance**

- **Problem**: Size and complexity of the systems.

**Principles to be supported**

- **Refinement principle**: Abstraction levels

- **Structuring mechanisms**: Decomposition and modularization techniques

- **Object-orientation**

- **Verification and validation concepts**
Requirements description $\leadsto$ Specification language

- Choice of the specification techniques depends on kind of system. Often more than a single specification technique is needed. *(What – How).*
- Kinds of systems:
  Pure function oriented (I/O), reactive-/embedded-/realtime systems.
- **Problem:** Universal specification technique (UST) difficult to understand, ambiguities, tools, size . . . e.g. UML
- **Desired:** Compact, legible, and exact specifications

Our focus: Specification of functional properties
Formal specifications

- A specification in a formal specification language defines
  - a model of the system and the possible behaviors
  - properties of the system

- 3 Aspects: Syntax, semantics, proof system
  - Syntax: What’s allowed to write down? 
    Specification as structured text often described by formulas from a logic
  - Semantics: What is the mathematical meaning of the specification?
    ~ Notion of models and mathematical structures
  - Proof system: Which properties of the system are true?
Formal specifications

- Two main classes:
  - **Model-oriented** (constructive)
    - Construction of a non-ambiguous model from available data structures and construction rules
    - e.g., VDM, Z, ASM
  - **Property-oriented** (declarative)
    - Signature of functions, predicates
    - Properties by formulas, axioms
    - Satisfying models
    - Algebraic specifications
    - e.g., Maude, OBJ, ASF, ...

- Operational specifications:
  - Petri nets, process algebras, automata based (SDL)
1. Introduction 1.3 Specification and Verification in SE

Tool support

- Syntactic support (grammars, parser,...)
- Verification: theorem proving (proof obligations)
- Prototyping (executable specifications)
- Code generation (generate programs from specifications)
- Testing (generate test cases from the specification)

Prerequisite for automation:
Formal syntax and semantics of the specification language
Declarative specification

Example

Restricted logic: e.g. equational logic

- **Axioms**: $\forall X \ t_1 = t_2$ $t_1, t_2$ terms.
- **Rules**: Equals are replaced with equals (directed).
- **Terms** $\approx$ names for objects (identifier), structuring, construction of the object.
- **Abstraction**: Terms as elements of an algebra, term algebra.
Algebraic specification: Example STACK

Example

Elements of an algebraic specification: **Signature** (sorts (types), operation names with arities), **Axioms** (often only equations)

```plaintext
spec STACK using NATURAL, BOOL  
sorts stack  
ops init : → stack  
push : stack nat → stack  
pop : stack → stack  
top : stack → nat  
is_empty : stack → bool  
stack_error : → stack  
nat_error : → nat
```

“names of known specs”

“principal type”

“empty stack”

(Signature fixed)
Axioms for Stack

\[ \text{FORALL } s : \text{stack} \quad n : \text{nat} \]

\[ \text{eqns} \]

\[
\begin{align*}
\text{is\_empty} (\text{init}) &= \text{true} \\
\text{is\_empty} (\text{push} (s, n)) &= \text{false} \\
\text{pop} (\text{init}) &= \text{stack\_error} \\
\text{pop} (\text{push} (s, n)) &= s \\
\text{top} (\text{init}) &= \text{nat\_error} \\
\text{top} (\text{push} (s,n)) &= n
\end{align*}
\]

Terms or expressions: \( \text{top} (\text{push} (\text{push} (\text{init}, 2), 3)) \) “means” 3

Semantics? Operationalization?

Apply equations as rules from left to right \( \rightarrow \)

Notion of rules and rewriting
Example: Sorting of lists over arbitrary types

Example

Formal ::

\[
\begin{align*}
\text{spec} & \quad \text{ELEMENT} \\
\text{using} & \quad \text{BOOL} \\
\text{sorts} & \quad \text{elem} \\
\text{ops} & \quad \mathcal{L} : \text{elem}, \text{elem} \rightarrow \text{bool} \\
\text{eqns} & \quad (x \leq x) = \text{true} \\
& \quad \text{imp}(x \leq y \text{ and } y \leq z, x \leq z) = \text{true} \\
& \quad x \leq y \text{ or } y \leq x = \text{true}
\end{align*}
\]
Example (Cont.)

```plaintext
spec LIST[ELEMENT]
using ELEMENT
sorts list
ops nil :→ list
   . : elem, list → list  ("infix")
insert : elem, list → list
insertsort : list → list
case : bool, list, list → list
sorted : list → bool
```
Example (Cont.)

eqns

\[
\begin{align*}
\text{case}(\text{true}, l_1, l_2) &= l_1 \\
\text{case}(\text{false}, l_1, l_2) &= l_2
\end{align*}
\]

insert(x, nil) = x.nil
insert(x, y.l) = case(x \leq y, x.y.l, y.insert(x, l))

insertsort(nil) = nil
insertsort(x.l) = insert(x, insertsort(l))

sorted(nil) = true
sorted(x.nil) = true
sorted(x.y.l) = \text{if } x \leq y \text{then} \text{sorted}(y.l) \text{else} false

Property: \text{sorted}(\text{insertsort}(l)) = true
Section 1.4

Summary
Summary

Foundations of theorem proving

- Syntax: symbols, terms, formulas
- Semantics: (mathematical) structures, variable assigments, denotation/meaning of terms and formulas
- Proof systems/logical calculi: axioms, deduction rules, proofs, theories

Fundamental principle of logic: “Establish truth by calculation”
Questions

1. Give an overview of the course
2. Motivate specification and verification
3. Explain language and semantics of propositional logic
4. Give and explain a logical rule. How is this rule applied?
5. What is a Hilbert style, what a natural deduction style proof system?
6. What is the advantage of a Hilbert style proof system?
7. Why is a natural deduction style proof system chosen for interactive proof assistants?