Preliminaries

Overview of Chapter

0. Preliminaries
0.1 Organisation
0.2 Course Overview

Organisation
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

Further reading


Contact

- Arnd Poetzsch-Heffter
- Patrick Michel
- 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/)

Literature

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
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
- Verification of Hoare logics
- 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

Introduction
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
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
- set of variables

Example (Natural Numbers)

- constant symbol: 0
- function symbol `suc` : \( \mathbb{N} \rightarrow \mathbb{N} \)
- function symbol `plus` : \( \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \)
- function symbol ...

Syntax (2)

Example (Propositional Logic)

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

Example (Symbols)

- each \( p \in V \) 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

Example (Language)

- \( True \) 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 \( ((\phi \rightarrow \psi) \land (\psi \rightarrow \phi)) \)

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 \( ((\phi \rightarrow \psi) \land (\psi \rightarrow \phi)) \)
Subsection 1.1.2

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

Interpretation/semantics

Notation:
$D_{\text{bool}}$ denotes the domain of boolean values, $D_{\text{bool}} = \{\text{true, false}\}$.

Example (Variable assignment in propositional logic)
A variable assignment $\rho$ in propositional logic is a mapping
- $\rho : V \rightarrow D_{\text{bool}}$

Example (Semantics of propositional formulas)
Let $\mathcal{J}$ be the standard interpretation of $\neg$ and $\rightarrow$, i.e.,

<table>
<thead>
<tr>
<th>$\mathcal{J}(\neg)$</th>
<th>\text{false}</th>
<th>\text{true}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{true}</td>
<td>\text{false}</td>
<td>\text{true}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{J}(\rightarrow)$</th>
<th>\text{false}</th>
<th>\text{true}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{false}</td>
<td>\text{true}</td>
<td>\text{true}</td>
</tr>
<tr>
<td>\text{true}</td>
<td>\text{true}</td>
<td>\text{true}</td>
</tr>
</tbody>
</table>

The semantics of propositional formulas is defined by the function $\text{sem}$ that takes a variable and a formula:
- $\text{sem} \rho p = \rho(p)$ for $p \in V$
- $\text{sem} \rho (\neg \phi) = \mathcal{J}(\neg)(\text{sem} \rho \phi)$
- $\text{sem} \rho (\phi \rightarrow \psi) = \mathcal{J}(\rightarrow)(\text{sem} \rho \phi, \text{sem} \rho \psi)$
Validity

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:
  - $\rho(p) = \text{false}$: $\text{sem}\, \rho(p \lor \neg p) = \text{true}$
  - $\rho(p) = \text{true}$: $\text{sem}\, \rho(p \lor \neg p) = \text{true}$

Introduction

General Concept

Fundamental principle of logic: “Establish truth by calculation”

- purely syntactical manipulations based on transformation rules
- starting point: set of formulas $\Gamma$, often a given set of axioms
- deriving new formulas by deduction rules from given formulas $\Gamma$
- $\phi$ is provable from $\Gamma$ if $\phi$ can be obtained by a finite number of derivation steps assuming the formulas in $\Gamma$
- notation: $\Gamma \vdash \phi$ means $\phi$ is provable from $\Gamma$
- notation: $\vdash \phi$ means $\phi$ is provable from a given set of axioms

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

$$
\frac{\phi_1 \ldots \phi_n}{\psi}
$$

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

Axioms

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

Modus Ponens

- Rule example: $\frac{\phi \rightarrow \psi \phi}{\psi}$
- if $\phi \rightarrow \psi$ and $\phi$ have already been proven, $\psi$ can be deduced
Hilbert calculus for propositional logic

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.

Proof example

Example (Hilbert proof)

- Language formed with the four propositional variables $p, q, r, s$
- Proof: $p \rightarrow p$
  
  Let
  
  $\psi_1 \equiv (p \rightarrow ((p \rightarrow (p \rightarrow p))) \rightarrow ((p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p))$
  
  $\psi_2 \equiv (p \rightarrow (p \rightarrow p))$
  
  $\psi_3 \equiv (p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p)$

  $\frac{\psi_1 \quad \psi_2}{\psi_3} \quad p \rightarrow (p \rightarrow p)
  
  \hline
  \hline
  (p \rightarrow p)$

Subsection 1.2.2

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

$$
\frac{\Gamma_1 \vdash \phi_1 \quad \cdots \quad \Gamma_n \vdash \phi_n}{\Gamma \vdash \psi}
$$

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

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

Natural deduction

Definition (Natural deduction rules for propositional logic)

$\lor$-introduction

$$
\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \lor \psi} \quad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \lor \psi}
$$

$\lor$-elimination

$$
\frac{\Gamma \vdash \phi \lor \psi}{\Gamma, \phi \vdash \xi} \quad \frac{\Gamma, \psi \vdash \xi}{\Gamma \vdash \xi}
$$

$\rightarrow$-introduction

$$
\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi}
$$

$\rightarrow$-elimination

$$
\frac{\Gamma \vdash \phi \rightarrow \psi}{\Gamma \vdash \phi} \quad \frac{\Gamma \vdash \phi}{\Gamma \vdash \psi}
$$

assumption

$$
\frac{\Gamma, \phi \vdash \phi}{\Gamma \vdash \phi}
$$

Proof example

Example (Natural deduction proof)
- Language formed with the four proposition symbols $p, q, r, s$
- Proof: $p \rightarrow p$ by assumption and $\rightarrow$-introduction:

$$
\frac{p \vdash p}{p \rightarrow p \vdash p}
$$
1. Introduction
1.3 Specification and Verification in SE

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

Specification and Verification in SE

Relation of specifications

Installation
Verification
Generation
1. Introduction 1.3 Specification and Verification in SE

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 Completeness
• 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!

Requirements
• The global specification describes, as exact as possible, the properties of the overall system
• Abstraction of the how
  Advantages
  ▶ apriori: Reference document, compact and legible.
  ▶ 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?

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)

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)

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

(Signature fixed)

Axioms for Stack

FORALL s : stack n : nat
eqns
is_empty (init) = true
is_empty (push (s, n)) = false
pop (init) = stack_error
pop (push (s, n)) = s
top (init) = nat_error
top (push (s,n)) = n

Terms or expressions: top (push (push (init, 2), 3)) “means” 3
Semantics? Operationalization?
Apply equations as rules from left to right
Notion of rules and rewriting

Example: Sorting of lists over arbitrary types

Example (Cont.)

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

Formal ::

spec ELEMENT
using BOOL
sorts elem
ops . ≤ . : elem, elem → bool
eqns (x ≤ x) = true
imp(x ≤ y and y ≤ z, x ≤ z) = true
x ≤ y or y ≤ x = true
Example (Cont.)

eqns case (true, \text{\texttt{l}}_1, \text{\texttt{l}}_2) = \text{\texttt{l}}_1 \\
case (false, \text{\texttt{l}}_1, \text{\texttt{l}}_2) = \text{\texttt{l}}_2

\text{insert}(x, \text{\texttt{nil}}) = x.\text{\texttt{nil}} \\
\text{insert}(x, y.\text{\texttt{l}}) = \text{\texttt{case}}(x \leq y, x.y.l, y.\text{\texttt{insert}}(x, \text{\texttt{l}}))

\text{insertsort}(\text{\texttt{nil}}) = \text{\texttt{nil}} \\
\text{insertsort}(x.\text{\texttt{l}}) = \text{\texttt{insert}}(x, \text{\texttt{insertsort}}(\text{\texttt{l}}))

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

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

---

Summary

Foundations of theorem proving

- Syntax: symbols, terms, formulas
- Semantics: (mathematical) structures, variable assignments, 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?