Specification and Verification with Higher-Order Logic

Vorlesung SS 2012

Prof. Dr. A. Poetzsch-Heffter

AG Softwaretechnik
TU Kaiserslautern

Chapter 0

Preliminaries

Overview of Chapter

Section 0.1

Organisation

Contact

- Arnd Poetzsch-Heffter
- Patrick Michel
- Christoph Feller
- Information about course: **http://softech.informatik.uni-kl.de/**
- Wiki for the course and Isabelle/HOL: **http://svhol.pbmichel.de/**

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

Literature

- T. Nipkow, L. C. Paulson and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic.*. Springer LNCS 2283, 2002.
- P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Acedemic Press, 1986
- L. C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 1996.
- R. Harper. *Programming in Standard ML*. Available at `http://www.cs.cmu.edu/~rwh/smlbook/book.pdf` Carnegie Mellon University, 2009.

Further reading

1. M. J. C. Gordon, T. F. Melham, *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
2. Peter Aczel. *An Introduction to Inductive Definitions.* Handbook of Mathematical Logic, pages 739-782. North-Holland, 1977.
3. Franz Baader, Tobias Nipkow. *Term Rewriting and All*. Cambridge University Press, 1998.
4. Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56-68, 1940.
5. Gerhard Gentzen. Untersuchungen ueber das logische Schliessen. *Mathematische Zeitschrift*, 39:176-210, 405-431, 1935.
6. Jean-Yves Girard, Yves Lafont, Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

Further reading (2)

7. Cordelia V. Hall, Kevin Hammond, Simon L. Peyton Jones, Philipp Wadler. Type classes in Haskell. *ACM Transactions on Programming Languages and Systems*, 18(2):109-138, 1996.
8. Steffen Hoelldobler. Conditional equational theories and complete sets of transformations. *Theoretical Computer Science*, 75(1&2):85-110, 1990.
9. Jan Willem Klop. *Term Rewriting Systems*. Handbook of Logic in Computer Science, Vol. 2, Chap. 1, pages 1-117. Oxford University Press, 1992.
10. Harry R. Lewis, Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.
11. Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348-375, 1978.

Further reading (3)

12. Tobias Nipkow. *Order-Sorted Polymorphism in Isabelle*. Logical Environments, pages 164-188. Cambridge University Press, 1993.
13. Wolfgang Naraschewski, Tobias Nipkow. Type inference verified: Algorithm $\mathcal{W}$ in isabelle/hol. *Journal of Automated Reasoning*, 23(3-4):299-318, 1999.
14. Dag Prawitz, Per-Erik Malmnas. A survey of some connections between classical, intuitionistic and minimal logic. In A. Schmidt, H. Schuette, editors, *Contributions to Mathematical Logic*, pages 215-229. North-Holland, 1968.
15. Dag Prawitz. *Natural Deduction: A proof theoretical study*. Almqvist and Wiksell, 1965.
16. M. E. Szabo. *The Collected Papers of Gerhard Gentzen*. North-Holland, 1969.

Further reading (4)

17. Simon Thompson. *Miranda: The Craft of Functional Programming*. Addison-Wesley, 1995.
18. Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, 1999. Second Edition.
19. Dirk van Dalen. *Logic and Structure*. Springer-Verlag, 1980.
20. Daniel J. Velleman. *How to Prove It*. Cambridge University Press, 1994.
21. Jean van Heijenoort, editor. *From Frege to Goedel: A Source Book in Mathematical Logic, 1879-193*. Harvard University Press, 1967. (Contains translations of original works by David Hilbert.)
22. Phillip Wadler, Stephen Blott. *How to make ad-hoc polymorphism less ad-hoc*. In Conference Record of the 16th ACM Symposium on Principles of Programming Languages, pages 60-76, 1989.
23. Alfred N. Whitehead, Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1925. 2nd edition.

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

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
- 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

Chapter 1

Introduction

Overview of Chapter

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

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 tupel 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

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 $\to$

Example (Language)

- each $p \in \mathcal{V}$ is a formula
- if ϕ is a formula, then $\neg\phi$ is a formula
- if ϕ and ψ are formulas, then $\phi \to \psi$ is a formula

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 $\phi \to \phi$
- *False* denotes $\neg$*True*
- $\phi \vee \psi$ denotes $(\neg\phi) \to \psi$
- $\phi \wedge \psi$ denotes $\neg((\neg\phi) \vee (\neg\psi))$
- $\phi \leftrightarrow \psi$ denotes $((\phi \to \psi) \wedge (\psi \to \phi))$

Subsection 1.1.2

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

Interpretation/semantics

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 ρ in propositional logic is a mapping

- $\rho : \mathcal{V} \to \mathcal{D}_{bool}$

Interpretation/semantics (2)

Example (Semantics of propositional formulas)

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

$\mathcal{J}(\neg)$	
false	true
true	false

$\mathcal{J}(\to)$	false	true
false	true	true
true	false	true

The semantics of propositional formulas is defined by the function *sem* that takes a variable and a formula:

- $sem\ \rho\ p = \rho(p)$ for $p \in \mathcal{V}$
- $sem\ \rho\ (\neg\phi) = \mathcal{J}(\neg)(sem\ \rho\ \phi)$
- $sem\ \rho\ (\phi \to \psi) = \mathcal{J}(\to)(sem\ \rho\ \phi, sem\ \rho\ \psi)$

Validity

Definition (Validity of propositional formulas)

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

Example (Tautology in propositional logic)

- $\phi \equiv p \vee \neg p$ is a tautology:
  - $\rho(p) =$ false: $sem\ \rho\ (p \vee \neg p) =$ true
  - $\rho(p) =$ true: $sem\ \rho\ (p \vee \neg p) =$ 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: $\Gamma \vdash \phi$ means ϕ is *provable* from Γ
- notation: $\vdash \phi$ means ϕ is *provable* from a given set of axioms

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

Modus Ponens

- Rule example: $\frac{\phi \to \psi \quad \phi}{\psi}$
- if $\phi \to \psi$ and ϕ have already been proven, ψ can be deduced

Hilbert calculus for propositional logic

Definition (Axioms of propositional logic)

All instantiations of the following schemas by arbitrary propositional formulas ϕ, χ, ψ are axioms:

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

Remark: Thus, there are infinitely many axioms.

Proof example

Example (Hilbert proof)

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

$$\begin{aligned} \psi_1 &\equiv (p \to ((p \to p) \to p)) \to ((p \to (p \to p)) \to (p \to p)) \\ \psi_2 &\equiv (p \to (p \to p)) \\ \psi_3 &\equiv (p \to (p \to p)) \to (p \to p) \end{aligned}$$

$$\dfrac{\dfrac{\overline{\psi_1} \quad \overline{\psi_2}}{\psi_3} \quad \overline{p \to (p \to p)}}{(p \to p)}$$

Subsection 1.2.2

Natural Deduction

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 $\to$
  - 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 Γ

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 Γ (set of formulas) and ϕ (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

Natural deduction

Definition (Natural deduction rules for propositional logic)

$\vee$-introduction $\quad \dfrac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \dfrac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$

$\vee$-elimination $\quad \dfrac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \xi \qquad \Gamma, \psi \vdash \xi}{\Gamma \vdash \xi}$

$\rightarrow$ -introduction $\quad \dfrac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi}$

$\rightarrow$ -elimination $\quad \dfrac{\Gamma \vdash \phi \rightarrow \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$

assumption $\quad \dfrac{}{\Gamma, \phi \vdash \phi}$

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:

$$\frac{\dfrac{}{p \vdash p}}{\vdash p \rightarrow p}$$

Section 1.3

Specification and Verification in SE

Motivation

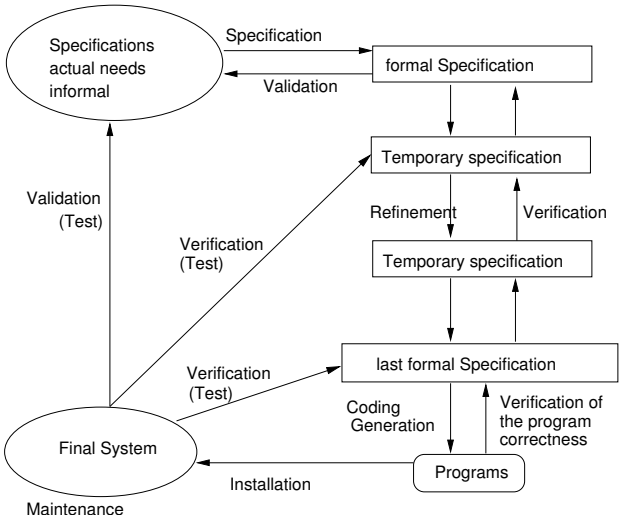
- Specifications: Models and properties ⇝ Spec-formalisms
- How do we express/specify facts? ⇝ Languages
- What is a proof? What is a formal proof? ⇝ Logical calculus
- How do we prove a specified fact? ⇝ Proof search
- Why formal? What is the role of a theorem prover? ⇝ Tools

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



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?
    $\rightsquigarrow$ Notion of models and mathematical structures
  - Proof system: Which properties of the system are true?

Formal specifications

- Two main classes:

Model oriented	**Property oriented**
(constructive)	(declarative)
Construction of a non-ambiguous model from available data structures and construction rules	Signature of unctions, predicates Properties by formulas, axioms Satisfying models
e.g., VDM, Z, ASM	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: $\forall X\ t_1 = t_2 \qquad t_1, t_2$ 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	"names of known spcecs"
sorts	stack	"principal type"
ops	init : → stack	"empty 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 $\leadsto$

Notion of rules and rewriting

Example: Sorting of lists over arbitrary types

Example

Formal ::

spec	ELEMENT
using	BOOL
sorts	elem
ops	$. \leq . : \text{elem}, \text{elem} \to \text{bool}$
eqns	$(x \leq x) = \text{true}$
	$\text{imp}(x \leq y \text{ and } y \leq z, x \leq z) = \text{true}$
	$x \leq y \text{ or } y \leq x = \text{true}$

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

Example (Cont.)

eqns $\text{case}(\text{true}, l_1, l_2) = l_1$
$\text{case}(\text{false}, l_1, l_2) = l_2$

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

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

$\text{sorted}(\text{nil}) = \text{true}$
$\text{sorted}(x.\text{nil}) = \text{true}$
$\text{sorted}(x.y.l) = \text{if } x \leq y \text{thensorted}(y.l)\text{elsefalse}$

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

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?