Specification and Verification in Higher Order Logic

Prof. Dr. K. Madlener

13. April 2011

Chapter 0

Organisation, Overview

Organisation

Contact

- Klaus Madlener
- Patrick Michel
- Christoph Feller
- **http://www-madlener.informatik.uni-kl.de/teaching/ss2011/**

Dates, Time, and Location

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

Organisation

Course Webpage

- **http://www-madlener.informatik.uni-kl.de/teaching/ss2011/svhol/**

Literature

Organisation (cont.)

- 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/offline.pdf. Carnegie Mellon University, 2009.
- T. Nipkow, L. C. Paulson and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic.* Springer LNCS 2283, 2002
- Prof. Basin, Dr. Brucker, Dr. Smaus, Prof. Wolff *Material of course CSMR* - http://www.infsec.ethz.ch/education/permanent/csmr/slides

Organisation (cont.)

Acknowledgements

- to Dr. Jens Brandt who designed most of the slides
- Prof. Dr. Arnd Poetzsch-Heffter for providing his 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.
- to the Isabelle/HOL community

Overview

Course Outline

- Introduction
- Concepts of functional programming
- Higher-order logic
- Verification in Isabelle/HOL (and other theorem provers)
- Verification of algorithms: A case study
- Modeling and verification of finite software systems: A case study
- Specification of programming languages
- Verification of a Hoare logics
- Beyond interactive theorem proving

Overall structure

1. Introduction
2. Functional specification and programming
3. Language and semantical aspects of higher-order logic
4. Proof system for higher-order logic
5. Sets, functions, relations, and fixpoints
6. Verifying functions
7. Inductively defined sets
8. Specification of programming language semantics
9. Program verification and programming logic

Chapter 1: Introduction

1. Terminology: Specification, verification, logic
2. Language: Syntax and semantics
3. Proof systems
   3.1 Hilbert style proof systems
   3.2 Proof system for natural deduction

Chapter 2: Functional programming and specification

1. Functional programming in ML
2. A simple theorem prover: Structure and unification
3. Functional specification in isabelle/HOL

- `slides_02: 1-65`
- `slides_02: 77-101`
- `Chapter 2 and 3 of Isabelle/HOL Tutorial`

Chapter 3: Language and semantical aspects of HOL

1. Introduction to higher-order logic
2. Foundation of higher-order logic
3. Conservative extension of theories

Chapter 4: Proof system for HOL

1. Formulas, sequents, and rules revisited
2. Application of rules
3. Fundamental methods of Isabelle/HOL
4. An overview of theory Main
   - 4.1 The structure of theory Main
   - 4.2 Set construction in Isabelle/HOL
   - 4.3 Natural numbers in Isabelle/HOL

Chapter 4: Proof system for HOL (cont.)

5. Rewriting and simplification
6. Case analysis and structural induction
7. Proof automation
8. More proof methods

» slides of Sessions 2, 3.1, 3.2, and 4 & 5 by T. Nipkow
» Chapter 5 of Isabelle/HOL Tutorial til page 99

Chapter 5: Sets, functions, relations, and fixpoints

1. Sets
2. Functions
3. Relations
4. Well-founded relations
5. Fixpoints

» `Chapter 6 of Isabelle/HOL Tutorial til page 118`

Chapter 6: Verifying functions

1. Conceptual aspects
2. Case study: Gcd
3. Case study: Quicksort – Shallow embedding of algorithms

» `theories for Gcd and Quicksort`

Chapter 7: Inductively defined sets

1. Defining sets inductively
2. Specification of transitions systems
   2.1 Transition systems
   2.2 Modeling: Case study Elevator
   2.3 Reasoning about finite transition systems

» Section 7.1 of Isabelle/HOL Tutorial
» slides of Sessions 6.1 T. Nipkow
» theory for Elevator

Chapter 8: Specification of programming language semantics

1. Introduction to programming language semantics
2. Techniques to express semantics
   2.1 Natural semantics / big step semantics
   2.2 Structured operational semantics / small step semantics
   2.3 Denotational semantics
3. Formalizing semantics in HOL

» `slides about operational semantics by P. MÃ¼ller`
» `theory for while-language`

Chapter 9:
Program verification and programming logic

1. Hoare logic
2. Program verification based on language semantics
3. Program verification with Hoare logic
4. Soundness of Hoare logic

» `theory for while-language`
» `theory for Hoare logic`

Chapter 1

Introduction .

Overview

Motivation

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

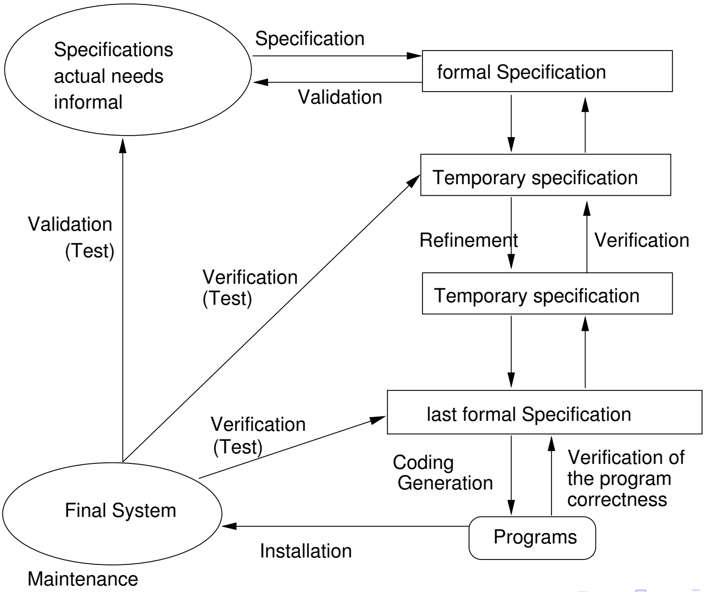
Goals

- role of formal specifications
- recapitulate logic
- introduce/review basic concepts

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: The group of documents that includes an executable program.


Specifications
actual needs
informal
Specification
Validation
formal Specification
Temporary specification
Refinement
Verification
Temporary specification
Validation
(Test)
Verification
(Test)
Verification
(Test)
last formal Specification
Coding
Generation
Verification of
the program
correctness
Final System
Installation
Programs
Maintenance

Comment

- First Specification: Global Specification
  Fundament for the Development
  “Contract or Agreement” between Developers and Client
- Intermediate (partial) specifications:
  Base of the Communication between Developers.
- Programs: Final products.

Development paradigms

- Structured Programming
- 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 the integrated final system with respect to the global specification.
- Activities: Validation, Verification, Testing, Consistency- and Completeness-Check
- Tool support needed!

Requirements

- The global specification describes, as exact as possible, what must be done.
- **Abstraction of the *how***
  Advantages
  - apriori: Reference document, compact and legible.
  - aposteriori: Possibility to follow and document design decisions $\rightsquigarrow$ **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 $\rightsquigarrow$ Specification Language

- Choice of the specification technique depends on the System. Frequently more than a single specification technique is needed. (What – How).
- Type of Systems:
  Pure function oriented (I/O), reactive- embedded- real time-systems.
- Problem : Universal Specification Technique (UST)
  difficult to understand, ambiguities, tools, size ...
  e.g. UML
- Desired: Compact, legible and exact specifications

Here: functional specification techniques

Formal Specifications

- A specification in a formal specification language defines all the possible behaviors of the specified system.
- 3 Aspects: Syntax, Semantics, Inference System
  - Syntax: What's allowed to write: Text with structure, Properties often described by formulas from a logic, e.g equational logic.
  - Semantics: Which models are associated with the specification, $\rightsquigarrow$ Notion of models.
  - Inference System: Consequences (Derivation) of properties of the system. $\rightsquigarrow$ Notion of consequence.

Formal Specifications

- Two main classes:

Model oriented	-	-	Property oriented
(constructive)			(declarative)
e.g.VDM, Z, ASM			signature (functions, predicates)
Construction of a			Properties
non-ambiguous model			(formulas, axioms)
from available			
data structures and			models
construction rules			algebraic specification
Concept of correctness			AFFIRM, OBJ, ASF, HOL,. . .

- 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 (out of the specifications generate C code)
- Testing (from the specification generate test cases for the program)

Desired:

To generate the tools out of the syntax and semantics of the specification language

Example: declarative

Example 1.1. 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.

Stack: algebraic specification

Example 1.2. Elements of an algebraic specification: Signature (sorts (types), operation names with arities), Axioms (often only equations)

SPEC STACK
USING NATURAL, BOOLEAN “Names of known SPECs”
SORT stack “Principal type”
OPS init : $\rightarrow$ stack “Constant of the type *stack*, empty stack”
push : stack nat $\rightarrow$ stack
pop : stack $\rightarrow$ stack
top : stack $\rightarrow$ nat
is_empty? : stack $\rightarrow$ bool
stack_error : $\rightarrow$ stack
nat_error : $\rightarrow$ nat

(Signature fixed)

Axioms for Stack

FORALL s : stack n : nat
AXIOMS
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 $\rightsquigarrow$

Notion of rules and rewriting

Example: Sorting of lists over arbitrary types

Example 1.3.

$$
\text{Formal} :: \left\{ \begin{array}{ll}
\text{spec} & \text{ELEMENT} \\
\text{use} & \text{BOOL} \\
\text{sorts} & \text{elem} \\
\text{ops} & . \leq . : \text{elem}, \text{elem} \to \text{bool} \\
\text{eqns} & x \leq x = \text{true} \\
 & \text{imp}(x \leq y \text{ and } y \leq z, x \leq z) = \text{true} \\
 & x \leq y \text{ or } y \leq x = \text{true}
\end{array} \right.
$$

Example (Cont.)

spec	LIST[ELEMENT]
use	ELEMENT
sorts	list
ops	nil :$\rightarrow$ list
	. : elem, list $\rightarrow$ list (“infix”)
	insert : elem, list $\rightarrow$ list
	insertsort : list $\rightarrow$ list
	case : bool, list, list $\rightarrow$ list
	sorted : list $\rightarrow$ bool

Example (Cont.)

$$
\begin{array}{ll}
\text{eqns} & \text{case}(\text{true}, l_1, l_2) = l_1 \\
& \text{case}(\text{false}, l_1, l_2) = l_2 \\
\\
& \text{insert}(x, \text{nil}) = x.\text{nil} \\
& \text{insert}(x, y.l) = \text{case}(x \leq y, x.y.l, y.\text{insert}(x, l)) \\
\\
& \text{insertsort}(\text{nil}) = \text{nil} \\
& \text{insertsort}(x.l) = \text{insert}(x, \text{insertsort}(l)) \\
\\
& \text{sorted}(\text{nil}) = \text{true} \\
& \text{sorted}(x.\text{nil}) = \text{true} \\
& \text{sorted}(x.y.l) = \text{if } x \leq y \text{ then sorted}(y.l) \text{ else false}
\end{array}
$$

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

Syntax

Aspects of syntax

- used to designate things and express facts
- terms and formulas are formed 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 (cont.)

Example 1.4. 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

Definition 1.5. *Symbols*

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

Definition 1.6. *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* $\phi \rightarrow \psi$ *is a formula*

Semantics

Purpose

- syntax only specifies the structure of terms and formulas
- symbols and terms are assigned a meaning
- variables are assigned a value
- in particular, propositional variables are assigned a truth value

Bottom-Up Approach

- assignments give variables a value
- terms/formulas are evaluated based on the meaning of the function symbols

Interpretations/Structures

Definition 1.7. *Assignment in Propositional Logic*
A variable assignment in propositionan logic is a mapping

- $I : \mathcal{V} \to \{\text{true}, \text{false}\}$

Definition 1.8. *Valuation of Propositional Logic*
The valuation V takes an assignment I and a formula and yiels a true or false:

- *if* $\phi \in \mathcal{V}$: $V(\phi) = I(\phi)$
- $V(\neg\phi) = f_\neg(V(\phi))$
- $V(\phi \to \psi) = f_\to(V(\phi), V(\psi))$

where

$f_\neg$	
false	true
true	false

$f_\to$	false	true
false	true	true
true	false	true

Problem 1.9. Is V a well defined function?

Validity

Definition 1.10. *Validity of formulas in propositional logic*

- *a formula ϕ is valid if $VI\phi$ evaluates to* true *for all assigments I*
- *notation:* $\models \phi$

Example 1.11. Tautology in Propositional Logic

- $\phi = a \vee \neg a$ (where $a \in \mathcal{V}$) is valid
  - $I(a) =$ false: $V(a \vee \neg a) =$ true
  - $I(a) =$ true: $V(a \vee \neg a) =$ true

Syntactic Sugar

Purpose

- additions to the language that do not affect its expressiveness
- more practical way of description

Example 1.12. Abbreviations in Propositional Logic

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

Proof Systems/Logical Calculi: Introduction

General Concept

- purely syntactical manipulations based on designated transformation rules
- starting point: set of formulas, 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

Proof System Styles

Hilbert Style

- easy to understand
- hard to use

Natural Deduction

- easy to use
- hard to understand

- …

Hilbert-Style Deduction Rules

Definition 1.13. *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 1.14. *Proof*

- *let D be a set of deduction rules, including the axioms as rules without premisses*
- *proofs in D are (natural) trees such that*
  - *axioms are proofs*
  - *if $P_1, \ldots, P_n$ are proofs with roots $\phi_1 \ldots \phi_n$ and*

    $\dfrac{\phi_1 \cdots \phi_n}{\psi}$ *is in D, then*

    $\dfrac{P_1 \cdots P_n}{\psi}$ *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 $\dfrac{}{\psi}$ is a proof
- axioms allow to construct trivial proofs

Rule example

- Modus Ponens: $\dfrac{\phi \to \psi, \quad \phi}{\psi}$
- if $\phi \to \psi$ and ϕ have already been proven, ψ can be deduced

Proof Example

Example 1.15. Hilbert Proof

- language formed with the four proposition symbols P, Q, R, S
- axioms: P, Q, $Q \to R$, $P \to (R \to S)$

$$\dfrac{\dfrac{\dfrac{}{P \to (R \to S)} \quad \dfrac{}{P}}{R \to S} \qquad \dfrac{\dfrac{}{Q \to R} \quad \dfrac{}{Q}}{R}}{S}$$

Hilbert Calculus for Propositional Logic

Definition 1.16. *Axioms of Propositional Logic*
All instantiations of the following schemas:

- $A \to (B \to A)$
- $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
- $(\neg B \to \neg A) \to ((\neg B \to A) \to B)$
- *where A, B, C are arbitrary propositions*

Rules: All instantiations of modus ponens.

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

- alternative definition where introduction of a hypothesis is a deduction rule
- deduction step can modify not only the proven propositions but also the assumptions Γ

Natural Deduction Rules

Definition 1.17. *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 Rules

Natural Deduction Rules

- 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
- Assumption Introduction, Assumption weakening:

$$\frac{}{\Gamma \vdash \phi}\ \phi \in \Gamma \qquad\qquad \frac{\Gamma \vdash \phi}{\Gamma, \psi \vdash \phi}$$

Natural Deduction Rules

Definition 1.18. *Natural Deduction Rules for Propositional Logic*

- $\vee$-*introduction*

$$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$$

- $\vee$-*elimination*

$$\frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \xi \qquad \Gamma, \psi \vdash \xi}{\Gamma \vdash \xi}$$

- $\rightarrow$-*introduction*

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

- $\rightarrow$-*elimination*

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

Natural Deduction Example

Example 1.19. $\{A \to C, B \to C\} \vdash (A \vee B) \to C$

$$
\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \vee B} \qquad \dfrac{\dfrac{}{\Gamma, A \vdash A \to C} \qquad \dfrac{}{\Gamma, A \vdash A}}{\Gamma, A \vdash C} \qquad \dfrac{\dots}{\Gamma, B \vdash C}}{\Gamma := \{A \to C, B \to C, A \vee B\} \vdash C}}{\{A \to C, B \to C\} \vdash (A \vee B) \to C}
$$

Summary

Specification and verification

- Algebraic specification - Functional specification

Theorem-Proving Fundamentals

- syntax: symbols, terms, formulas
- semantics: (mathematical structures,) variable assigments, denotations for terms and formulas
- proof system/(logical) calculus: axioms, deduction rules, proofs, theories

Fundamental Principle of Logic: “Establish truth by calculation” (APH, 2010)