universität innsbruck

Summer Term 2019


# Interactive Theorem Proving using Isabelle/HOL

## Session 11

Christian Sternagel

Department of Computer Science

## Topics

calculational reasoning, case analysis, code generation, computation induction, data type invariants, document preparation, finding theorems, first steps, functional programming in HOL, higher-order logic, history and motivation, induction, inductive definitions, Isabelle basics, Isabelle/Isar, Isabelle/ML, IsaFoR/CeTA, locales, manual termination proofs, multisets, natural deduction, notation, proof methods, PSL: a high-level proof strategy language, rule induction, rule inversion, session management, sets, simplification, sledgehammer, structural induction, structured proof, The Archive of Formal Proofs, the certification approach, total recursive functions, type classes, type definitions, well-foundedness

## Topics

calculational reasoning, case analysis, code generation, computation induction, data type invariants, document preparation, finding theorems, first steps, functional programming in HOL, higher-order logic, history and motivation, induction, inductive definitions, Isabelle basics, Isabelle/Isar, Isabelle/ML, IsaFoR/CeTA, locales, manual termination proofs, multisets, natural deduction, notation, proof methods, PSL: a high-level proof strategy language, rule induction, rule inversion, session management, sets, simplification, sledgehammer, structural induction, structured proof, The Archive of Formal Proofs, the certification approach, total recursive functions, type classes, type definitions, well-foundedness

# Overview

- Semester Projects
- Locales
- Data Type Invariants
- Exercises

# Semester Projects

## Project Submission

- submit project through Project Folder of course page in OLAT

## Project Submission

- submit project through Project Folder of course page in OLAT
- before August 1, 2019

## Project Submission

- submit project through Project Folder of course page in OLAT
- before August 1, 2019
- if you want grade before August 1, notify me of your submission via email

## Project Submission

- submit project through Project Folder of course page in OLAT
- before August 1, 2019
- if you want grade before August 1, notify me of your submission via email

## Submission Format

single ZIP archive containing files

- `ROOT` – defines single session for whole project (do not use "`quick_and_dirty`") that generates PDF in directory `output/`
- `Project_*.thy` – documented (using antiquotations) formalization (without `sorry`s)
- `document/root.tex` – main LaTeX file with appropriate title, authors, and date

## Project Submission

- submit project through Project Folder of course page in OLAT
- before August 1, 2019
- if you want grade before August 1, notify me of your submission via email

## Submission Format

single ZIP archive containing files

- `ROOT` – defines single session for whole project (do not use "`quick_and_dirty`") that generates PDF in directory `output/`
- `Project_*.thy` – documented (using antiquotations) formalization (without `sorry`s)
- `document/root.tex` – main LaTeX file with appropriate title, authors, and date

## Before Submission

make sure that ZIP archive `name.zip` passes

```
$ unzip name.zip && isabelle build -D .
```

without errors

# Locales

## Limitations of Type Classes

- “one shot” instances (if you chose instance once, it is fixed)

## Limitations of Type Classes

- “one shot” instances (if you chose instance once, it is fixed)
- only single type variable as parameter

## Limitations of Type Classes

- “one shot” instances (if you chose instance once, it is fixed)
- only single type variable as parameter, hence
- no constructor classes

## Limitations of Type Classes

- “one shot” instances (if you chose instance once, it is fixed)
- only single type variable as parameter, hence
- no constructor classes and
- no multi-parameter type classes

## Limitations of Type Classes

- “one shot” instances (if you chose instance once, it is fixed)
- only single type variable as parameter, hence
- no constructor classes and
- no multi-parameter type classes
- (in particular: no functors, no monads, . . . )

## Locales – The Module System of Isabelle

- locale provides abstract “interface” (of constants) together with (their) properties

## Locales – The Module System of Isabelle

- locale provides abstract “interface” (of constants) together with (their) properties
- interpretation yields concrete “implementation” and involves proving required properties

## Locales – The Module System of Isabelle

- locale provides abstract “interface” (of constants) together with (their) properties
- interpretation yields concrete “implementation” and involves proving required properties

## `Demo11.thy` – Containers

an interface for container types providing

- empty container
- insertion of elements
- check whether element is present

# Data Type Invariants

## Invariants

- employ `typedef` to obtain all elements of type `T` that satisfy invariant `P :: T ⇒ bool`

## Invariants

- employ `typedef` to obtain all elements of type `T` that satisfy invariant `P :: T ⇒ bool`
- operations on new type have to be defined (in terms of operations on underlying type)

## Invariants

- employ `typedef` to obtain all elements of type `T` that satisfy invariant `P :: T ⇒ bool`
- operations on new type have to be defined (in terms of operations on underlying type)
- moreover, we need to prove that operations preserve invariant

## Invariants

- employ `typedef` to obtain all elements of type `T` that satisfy invariant `P :: T ⇒ bool`
- operations on new type have to be defined (in terms of operations on underlying type)
- moreover, we need to prove that operations preserve invariant

## `Demo11.thy` – Binary Search Trees

- start from binary trees

## Invariants

- employ `typedef` to obtain all elements of type `T` that satisfy invariant `P :: T ⇒ bool`
- operations on new type have to be defined (in terms of operations on underlying type)
- moreover, we need to prove that operations preserve invariant

## `Demo11.thy` – Binary Search Trees

- start from binary trees
- maintain invariant that for every node $n$ in tree all nodes in left subtree are strictly smaller than $n$ and all nodes in right subtree are strictly greater than $n$

# Exercises

## Exercises (start from `Exercises11.thy`)

**URL**

`http://cl-informatik.uibk.ac.at/teaching/ss19/itp/thys/Exercises11.thy`

## Further Reading

Clemens Ballarin.
Tutorial to Locales and Locale Interpretation.
Isabelle documentation, 2018.