universität innsbruck

Summer Term 2019


# Interactive Theorem Proving using Isabelle/HOL

## Session 7

Christian Sternagel

Department of Computer Science

## Topics

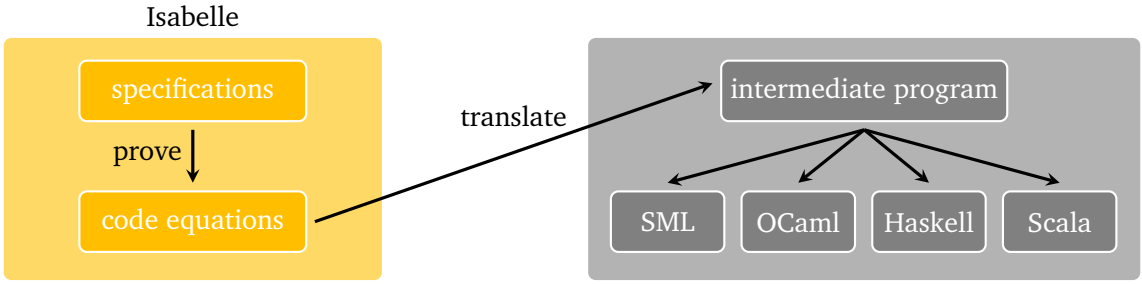
calculational reasoning, case analysis, code generation, computation induction, data type invariants, document preparation, finding theorems, first steps, functional programming in HOL, higher-order logic, history and motivation, induction, inductive definitions, Isabelle basics, Isabelle/Isar, Isabelle/ML, 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

# Overview

- Code Generation
- Computation Induction
- Multisets
- Exercises

## Code Generator Architecture

- code equations – executable subset of Isabelle/HOL specifications of shape
  `f` $t_1$ `...` $t_n$ `= ...`
- code equations translated into intermediate program with datatypes and functions
- intermediate program serialized into concrete programming language




## Note

pen-and-paper proof that translation guarantees partial correctness [1]

## Exporting Haskell Code

- `code_thms f` – print code equations for `f`
- `export_code f in Haskell` – print Haskell code for function `f`
- `export_code f in Haskell file "dir/"` – actually generate code in directory `dir`

## Demo07.thy – Reverse

```
fun rev :: "'a list ⇒ 'a list"
  where
    "rev [] = []"
  | "rev (x # xs) = rev xs @ [x]"

code_thms rev
export_code rev in Haskell
```

## Declaring Code Equations

- some commands, like `fun` and `definition`, implicitly declare code equations
- `declare` *fact* `[code del]` – remove code equation *fact*
- use attribute `[code]` to declare code equation
- use attribute `[code_unfold]` to replace subexpressions in code equations

## Demo07.thy – Efficiency

```
fun itrev :: "'a list ⇒ 'a list ⇒ 'a list"
  where
    "itrev [] acc = acc"
  | "itrev (x # xs) acc = itrev xs (x # acc)"

lemma itrev_rev [simp]: "itrev xs ys = rev xs @ ys" ⟨proof⟩
lemma rev_code [code]: "rev xs = itrev xs []" ⟨proof⟩
export_code rev in Haskell
```

## Demo07.thy – Examples of Non-Executable Specifications

- infinite sets

```
definition "evens = {n::nat. even n}" (*set of even numbers*)
definition "is_even n ⟷ n ∈ evens"
```

- inductively defined predicates

```
inductive ev :: "nat ⇒ bool"
  where
    "ev 0"
  | "ev x ⟹ ev (Suc (Suc x))"
...
```

### Note

if possible, prove executable code equations

## Computation Induction

- induction along the computation of recursive functions with well-founded call relation ("terminating function"; always the case in Isabelle/HOL)
- induction rule: one base-case for each non-recursive equation plus one step-case for each recursive call
- recursive `fun f` comes with induction rule `f.induct`
- corresponding case names are numbers

## Example – Division by Two

```
fun div2 :: "nat ⇒ nat"
  where
    "div2 0 = 0"
  | "div2 (Suc 0) = 0"
  | "div2 (Suc (Suc x)) = Suc (div2 x)"
lemma "div2 n = n div 2" sorry
```

## Demo07.thy – Quicksort

```
fun qsort :: "nat list ⇒ nat list"
  where
    "qsort [] = []"
  | "qsort (x # xs) =
      qsort (filter ((>) x) xs) @ [x] @ qsort (filter ((≤) x) xs)"
```

### Note

'(op)' with infix operator op is short for '`λx y. x op y`'

## Demo07.thy – Quicksort does not change the Multiset of Elements

```
lemma mset_qsort [simp]:
  "mset (qsort xs) = mset xs"
  sorry
```

## (Finite) Multisets (aka Bags)

- import `"HOL-Library.Multiset"`
- type `'a multiset`
- finite collection of elements (notation: {# $x_1$, ... ,$x_n$ #})
- duplicates allowed (for example, `{# 1,1,2,3 #}`)
- order irrelevant (for example, `{# 1,1,2,3 #} = {# 2,1,3,1 #}`)

## Basic Operations

- `{#}` – the empty multiset
- `add_mset x M` – add element `x` to multiset `M`
- `M + N` – multiset sum of `M` and `N`
- `mset :: 'a list ⇒ 'a multiset` – collect list elements as multiset
- `set_mset :: 'a multiset ⇒ 'a set` – turn multiset into set
- `filter_mset P M` – only keep elements from `M` that satisfy `P`
  (special syntax: `{# x ∈# M. P x #}`)

## Demo07.thy – Recombining Filtered Multisets

```
value "filter_mset even {#1,1,2,3,4,4#} :: nat multiset"

lemma filter_mset_sum_id:
  assumes "∀x. P x ⟷ ¬ Q x"
  shows "filter_mset P M + filter_mset Q M = M"
...
```

## Isabelle Notation – Subscripts and Superscripts

| symbol | internal | shortcut |
|---|---|---|
| subscript | \<^sub> | CTRL+e ⇩ |
| superscript | \<^sup> | CTRL+e ⇧ |

## Exercises (start from `Exercises07.thy`)

**URL**

http://cl-informatik.uibk.ac.at/teaching/ss19/itp/thys/Exercises07.thy

## Further Reading


Florian Haftmann and Tobias Nipkow.
Code generation via higher-order rewrite systems.
In *FLOPS*, volume 6009 of *LNCS*, pages 103–117. Springer, 2010.
doi:10.1007/978-3-642-12251-4_9.

Florian Haftmann.
Code generation from Isabelle/HOL theories.
Isabelle documentation, 2018.

## Important Concepts

- `code` (*attribute*)
- code equations
- code generation
- computation induction
- `export_code` (*HOL command*)
- `multiset` (*HOL type*)