

Functional Programming

Christian Sternagel Harald Zankl Evgeny Zuenko

Department of Computer Science
University of Innsbruck

WS 2017/2018

Lecture 7



Overview

- Mathematical Induction
- Induction over Lists
- Structural Induction
- Formal Verification of Functional Programs

Topics

abstract data types, algebraic data types, binary search trees, combinator parsing, efficiency, encoding data types as lambda-terms, evaluation strategies, **formal verification**, first steps, guarded recursion, Haskell introduction, higher-order functions, historical overview, implementing a type checker, **induction**, infinite data structures, input and output, lambda-calculus, lazy evaluation, list comprehensions, lists, modules, pattern matching, polymorphism, property-based testing, **reasoning about functional programs**, recursive functions, sets, strings, tail recursion, trees, tupling, type checking, type inference, types, types and type classes, unification, user-defined types

When to use Mathematical Induction?

- prove that some property P holds for all natural numbers
- more formally, prove:

$$\forall n. P(n) \quad (\text{where } n \in \mathbb{N})$$

How is it Applied?

- mathematical induction consists of two steps:
- first prove **base case** show property for 0

$$P(0)$$

- then **step case**

$$\forall k. (P(k) \longrightarrow P(k+1))$$

assume $P(k)$ (induction hypothesis), show $P(k+1)$

Why does this Work?

- we have two ingredients:
 1. P is true for 0
 2. if P is true for arbitrary k it is also true for $k + 1$
- and want to show P for every natural number ($\forall n. P(n)$)

Example – $P(3)$

- have $P(0)$
- and $P(0) \longrightarrow P(1)$
- thus $P(1)$
- with $P(1) \longrightarrow P(2)$
- have $P(2)$
- with $P(2) \longrightarrow P(3)$
- have $P(3)$

Idea

- we can reach arbitrary n
- such that $P(n)$
- hence, $\forall n. P(n)$

Remark

- base case may be changed
- e.g., if base case $P(1)$, property holds for all $n \geq 1$

General Induction Principle

$$(P(m) \wedge \forall k \geq m. (P(k) \longrightarrow P(k + 1))) \longrightarrow \forall n \geq m. P(n)$$

Domino Effect

- first domino will fall
- if a domino falls also its right neighbor falls



What is a “Property”?

- anything that depends on some input and is either true or false
- that is, some function $p :: a \rightarrow \text{Bool}$

Example – Gauß's Formula

- $P(x) = (1 + 2 + \dots + x = \frac{x(x+1)}{2})$
- base case: $P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0(0+1)}{2})$
- step case: $P(k) \rightarrow P(k + 1)$
 IH: $P(k) = (1 + 2 + \dots + k = \frac{k(k+1)}{2})$
 show: $P(k + 1)$

$$\begin{aligned} 1 + 2 + \dots + (k + 1) &= (1 + 2 + \dots + k) + (k + 1) \\ &\stackrel{\text{IH}}{=} \frac{k(k + 1)}{2} + (k + 1) \\ &= \frac{(k + 1)(k + 2)}{2} \end{aligned}$$

Algebraic Data Type of Lists

data `[a]` = `[]` | `(:)` `a` `[a]`

Notes

- lists are recursive structures
- non-recursive constructor (base case): `[]`
- recursive constructor (step case): $x : xs$

Induction Principle for Lists – Informally

- to show $P(xs)$ for all lists xs
- show base case: $P([])$
- show step case: $P(xs) \longrightarrow P(x : xs)$ for arbitrary x and xs

Induction Principle for Lists – Formally

$$(P([]) \wedge \forall x. \forall xs. (P(xs) \longrightarrow P(x : xs))) \longrightarrow \forall xs. P(xs)$$

Remark

- for lists, P can be seen as function $p :: [a] \rightarrow \text{Bool}$

Exercise – Nil is right identity of append

- definition of append

$$[] \ ++ \ ys = ys$$

$$(x:xs) \ ++ \ ys = x : (xs \ ++ \ ys)$$
- prove that $[]$ is **right identity** of $++$, that is,

$$xs \ ++ \ [] = xs$$

Exercise – Associativity of append

- recall

$$[] \ ++ \ ys = ys$$

$$(x:xs) \ ++ \ ys = x : (xs \ ++ \ ys)$$
- prove that $++$ is **associative**, that is,

$$(xs \ ++ \ ys) \ ++ \ zs = xs \ ++ \ (ys \ ++ \ zs)$$

Exercise – Length and append

- definition

$$\text{length } [] = 0$$

$$\text{length } (_ : xs) = 1 + \text{length } xs$$
- prove that length of combined list is sum of lengths, that is,

$$\text{length } (xs \ ++ \ ys) = \text{length } xs + \text{length } ys$$

Example – Terms

```
type Id = String
data Term = Var Id
          | App Term Term
          | Abs Id Term
```

General Structures – Induction Principle

- for every non-recursive constructor, show base case
 - base case: $P(\text{Var } x)$
- for every recursive constructor, show step case
 - step case 1: $(P(s) \wedge P(t)) \longrightarrow P(\text{App } s \ t)$
 - step case 2: $P(t) \longrightarrow P(\text{Abs } x \ t)$

Exercise – Perfect Binary Trees

- a binary tree is **perfect** if all leaf nodes have same depth

```
perfect Empty          = True
perfect (Node _ l r) =
  height l == height r && perfect l && perfect r
```

```
height Empty          = 0
height (Node _ l r) =
  max (height l) (height r) + 1
```

```
size Empty          = 0
size (Node _ l r) = size l + size r + 1
```

- lemma: a perfect binary tree t of height n has exactly $2^n - 1$ nodes, that is,

$$P(t) = (\text{perfect } t \longrightarrow \text{size } t = 2^{\text{height } t} - 1)$$

Example – Binary Trees

```
data BTree a = Empty
             | Node a (BTree a) (BTree a)
```

Induction Principle for Binary Trees

$$(P(\text{Empty}) \wedge \forall x. \forall l. \forall r. ((P(l) \wedge P(r)) \longrightarrow P(\text{Node } x \ l \ r))) \longrightarrow \forall t. P(t)$$

Isabelle/HOL in a Nutshell

Obvious question:

- What is Isabelle?

Common answer:

- An **LCF-style proof assistant**.

Typical follow-up questions:

- What is a **proof assistant**?
- What does **LCF-style** mean?
- ...

What is a Proof Assistant?

- combination of automated theorem prover and proof checker
- some subproofs are found **automatically**
- others are user-supplied but **checked** rigorously

Example

- automatic methods: classical tableaux (`blast`), equational reasoning (`simp`), combination of former (`auto`), ...
- manual steps: induction (`induct`), case analysis (`cases`), ...

Higher-Order Logic

- HOL = Functional Programming + Logic
- data types (`datatype`)
- recursive functions (`fun`)
- logical operators ($\wedge, \vee, \longrightarrow, \forall, \exists, \dots$)

What does LCF-style mean?

- theorems represented by abstract datatype (`thm`)
- set of (basic) logical inferences provided as interface (**trusted kernel**)
- strong typing guarantees that there is no other way to create theorems (values of type `thm`)

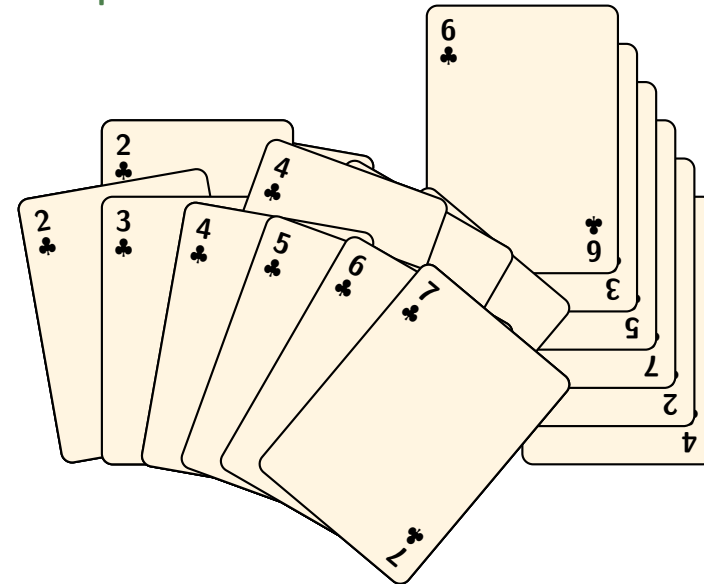
Example

certified term

- functions `assume : cterm -> thm` and `implies_elim : thm -> thm -> thm`
- implement inference rules

$$\frac{}{A \vdash A} \quad \frac{\Gamma \vdash A \implies B \quad \Delta \vdash A}{\Gamma, \Delta \vdash B}$$

Example – Insertion Sort



A Functional Implementation

- inserting an element into a sorted list

```
insert x [] = [x]
insert x (y:ys) =
  if x <= y then x : y : ys
  else y : insert x ys
```

- sorting by repeatedly inserting elements into the empty list

```
insertionSort = foldr insert []
```

Exercise – Insertion sort is a valid sorting algorithm

- prove that result after applying insertion sort is sorted
- prove that result contains exactly the same elements as input
- see `Insertion_Sort.thy`

Exercises (for December 1st)

- Read the lecture notes on reasoning about functional programs.
- Download Isabelle/HOL from <http://isabelle.in.tum.de> and start its Isabelle/jEdit interface.
- Prove the following equation by induction:

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

- Prove $\text{rev } (xs ++ ys) = \text{rev } ys ++ \text{rev } xs$ for

```
rev [] = []
```

```
rev (x:xs) = rev xs ++ [x]
```

using the equations

$$xs ++ [] = xs \quad (*)$$

$$(xs ++ ys) ++ zs = xs ++ (ys ++ zs) \quad (**)$$

- Prove Exercise 4 in Isabelle/HOL using the HOL data type `datatype 'a list = Nil | Cons 'a "'a list"` and your own definitions of reversal (`rev`) and append (`app`).
- Prove $\text{rev } (\text{rev } xs) = xs$ in Isabelle/HOL.