Functional Programming
WS 2009/10

Christian Sternagel (VO)
Friedrich Neurauter (PS)  Sarah Winkler (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

20 November 2009

Week 7 - Induction

Summary of Week 6

Rewrite Strategies

Outermost

▸ choose the (leftmost) outermost redex
▸ redex is outermost if not subterm of different redex

Innermost

▸ choose the (leftmost) innermost redex
▸ redex is innermost if no proper subterm is redex
**Reduction Strategies**

**Call-by-name**
- use outermost strategy
- stop as soon as WHNF is reached

**Call-by-value**
- use innermost strategy
- stop as soon as WHNF is reached

**WHNF (Intuition)**

*Thou shalt not reduce below lambda.*

---

**Evaluation Strategies**

**Lazy**
- call-by-name + sharing
- only evaluate if necessary
- e.g. Haskell

**Strict/Eager**
- call-by-value
- evaluate arguments before calling a function
- e.g. OCaml (also support for lazyness)
This Week

Practice I
OCaml introduction, lists, strings, trees

Theory I
lambda-calculus, evaluation strategies, induction, reasoning about functional programs

Practice II
efficiency, tail-recursion, combinator-parsing

Theory II
type checking, type inference

Advanced Topics
lazy evaluation, infinite data structures, monads, . . .

When?

Goal
"prove that some property P holds for all natural numbers"

Formally

\[ \forall n. P(n) \quad (\text{where } n \in \mathbb{N}) \]
How?

2 goals to show

1. $P(0)$
2. $\forall k. (P(k) \rightarrow P(k + 1))$

Why Does This Work?

We have

- $P(0)$ “property $P$ holds for 0”
- $\forall k. (P(k) \rightarrow P(k + 1))$ “if property $P$ holds for arbitrary $k$ then it also holds for $k + 1$”

We want $\forall n. P(n)$ “$P$ holds for every $n$”

We get

- for the moment fix $n$
- have $P(0)$
- have $P(0) \rightarrow P(1)$
- have $P(1)$
- have $P(1) \rightarrow P(2)$

- . . .
- have $P(n - 1)$
- have $P(n - 1) \rightarrow P(n)$
- hence $P(n)$
**What is Meant by ‘Property’?**

anything that depends on some variable and is either true or false can be seen as function \( p : 'a \to \text{bool} \)

**Example**

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

\[
1 + 2 + \cdots + (k + 1) = (1 + 2 + \cdots + k) + (k + 1) \\
\quad \quad \quad \quad \quad \quad \equiv \frac{k \cdot (k + 1)}{2} + (k + 1) \\
\quad \quad \quad \quad \quad \quad = \frac{(k + 1) \cdot (k + 2)}{2}
\]

**Remark**

- of course the base case can be changed
- e.g., if base case \( P(1) \), property holds for all \( n \geq 1 \)
Recall

Type

\[
\text{type } 'a\text{ list } = [] \mid (::) \text{ of } 'a \times 'a\text{ list}
\]

Note

- lists are recursive structures
- base case: \([\]\)
- step case: \(x :: xs\)

Induction Principle on Lists

Intuition

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

Formally

\[
(P([]) \land \forall x : \alpha. \forall xs : \alpha\text{ list.} (P(xs) \rightarrow P(x :: xs)))
\]

\[
\rightarrow \forall ls : \alpha\text{ list.} P(ls)
\]

Remarks

- \(y : \beta\) reads ‘\(y\) is of type \(\beta\)’
- for lists, \(P\) can be seen as function \(p : 'a\text{ list} \rightarrow \text{ bool}\)
Example - Lst.append

Recall

\[
\text{let rec } (\@) \text{ xs ys = match xs with} \\
\quad \mid [] \to ys \\
\quad \mid \text{cons} : xs \to \text{cons} : (xs \@ ys)
\]

Lemma

\[
\text{[] is right identity of } \@, \text{ i.e.,}
\]

\[
xs @ [] = xs
\]

Proof.
Blackboard

Example - Lst.length

Recall

\[
\text{let rec } \text{length} = \text{function } [] \to 0 \\
\quad \mid \text{cons} : xs \to 1 + \text{length} xs
\]

Lemma

\[
\text{sum of lengths equals length of combined list, i.e.,}
\]

\[
\text{length xs} + \text{length ys} = \text{length}(xs \@ ys)
\]

Proof.
Blackboard
Week 7 - Induction

Structural Induction

General Structures

Type

\[
\text{type term} = \text{Var of} \ \text{var} \\
| \text{Abs of} \ (\text{var} \ * \ \text{term}) \\
| \text{App of} \ (\text{term} \ * \ \text{term})
\]

Induction Principle

▶ for every non-recursive constructor there is a base case
  ▶ base case: \text{Var x}
▶ for every recursive constructor there is a step case
  ▶ step case: \text{Abs(x, t)}
  ▶ step case: \text{App(s, t)}

Induction Principle on General Structures

Intuition

▶ to show \(P(s)\) for all structures \(s\)
▶ show base cases
▶ show step cases
Recall

Type

```plaintext
type 'a btree = Empty | Node of ('a btree * 'a * 'a btree)
```

Induction Principle

\[
(P(\text{Empty}) \land \\
\forall v : \alpha. \forall l : \alpha \ btree. \forall r : \alpha \ btree. \\
((P(l) \land P(r)) \rightarrow P(\text{Node}(l, v, r)))) \rightarrow \\
\forall t : \alpha \ btree. P(t)
\]

Example - Trees

Definition (Perfect Binary Trees)

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

```plaintext
let rec perfect = function
| Empty        -> true
| Node(l,_,r)  -> height l = height r && perfect l 
               && perfect r
```
Example - Trees (cont’d)

Recall

\[
\text{let rec height} = \text{function} \\
\quad | \text{Empty} \rightarrow 0 \\
\quad | \text{Node}(l,_,r) \rightarrow \max(\text{height } l) (\text{height } r) + 1
\]

\[
\text{let rec size} = \text{function} \\
\quad | \text{Empty} \rightarrow 0 \\
\quad | \text{Node}(l,_,r) \rightarrow \text{size } l + \text{size } r + 1
\]

Lemma

*perfect binary tree* \( t \) of height \( n \) has exactly \( 2^n - 1 \) nodes

Proof.
To show: \( P(t) = (\text{perfect } t \rightarrow (\text{size } t = 2^{(\text{height } t)} - 1)) \)

Blackboard