

Functional Programming

WS 2012/13

Harald Zankl (VO)

Cezary Kaliszyk (PS) Thomas Sternagel (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

week 7



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 laziness)

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

Gives

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

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 $P(n - 1)$ |
| ▶ have $P(0) \rightarrow P(1)$ | ▶ have $P(n - 1) \rightarrow P(n)$ |
| ▶ have $P(1)$ | ▶ hence $P(n)$ |
| ▶ have $P(1) \rightarrow P(2)$ | |

What is Meant by 'Property'?

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

Example

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

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

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

```
type 'a list = [] | (:::) of 'a * 'a 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

$$(P([]) \wedge \forall x : \alpha. \forall xs : \alpha \text{ list}. \overbrace{(P(xs) \rightarrow P(x :: xs))}^{\text{IH}})) \rightarrow \forall ls : \alpha \text{ list}. P(ls)$$

Remarks

- ▶ $y : \beta$ reads '*y is of type β* '
- ▶ for lists, P can be seen as function $p : 'a \text{ list} \rightarrow \text{bool}$

Example - Lst.append

Recall

```
let rec (@) xs ys = match xs with
| []      -> ys
| x::xs   -> x :: (xs @ ys)
```

Lemma

`[]` is *right identity* of `@`, i.e.,

$$xs @ [] = xs$$

Proof.

Blackboard



Example - Lst.length

Recall

```
let rec length = function []      -> 0
                       | _::xs -> 1 + length xs
```

Lemma

sum of lengths equals length of combined list, i.e.,

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

Proof.

Blackboard



General Structures

Type

```
type term = Var of var
          | Abs of (var * term)
          | App of (term * term)
```

Induction Principle

- ▶ for every non-recursive constructor there is a base case
 - ▶ base case: `Var x`
- ▶ for every recursive constructor there is a step case
 - ▶ step case: `Abs(x, t)`
 - ▶ step case: `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

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

Induction Principle

$$\begin{aligned} & (P(\text{Empty}) \wedge \\ & \forall v : \alpha. \forall l : \alpha \text{ btree}. \forall r : \alpha \text{ btree}. \\ & ((P(l) \wedge P(r)) \rightarrow P(\text{Node}(l, v, r)))) \\ & \rightarrow \\ & \forall t : \alpha \text{ btree}. P(t) \end{aligned}$$

Example - Trees

Definition (Perfect Binary Trees)

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

```
let rec perfect = function
  | Empty          -> true
  | Node(l,_,r) -> height l = height r && perfect l
                  && perfect r
```

Example - Trees (cont'd)

Recall

```

let rec height = function
  | Empty          -> 0
  | Node(l,_,r)   -> max (height l) (height r) + 1

let rec size = function Empty          -> 0
                    | Node(l,_,r)   -> size l + 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

