

# 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

### Overview

### • Week 7 - Induction

- Summary of Week 6
- Mathematical Induction
- Induction Over Lists
- Structural Induction

34511

### Overview

### Week 7 - Induction

### • Summary of Week 6

Mathematical Induction

Induction Over Lists

Structural Induction

### Outermost

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

### Outermost

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

### 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

### 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

# **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

# **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

## **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, ...

### Overview

#### Week 7 - Induction

• Summary of Week 6

### Mathematical Induction

- Induction Over Lists
- Structural Induction

### Goal

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

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

### 2 goals to show

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

### 2 goals to show

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

### Gives

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

# We have

• P(0)

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

### We have

- P(0) "property P holds for 0"
- $\forall k.(P(k) \rightarrow P(k+1))$

### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → P(k + 1)) "if property P holds for arbitrary k then it also holds for k + 1"

### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → P(k + 1)) "if property P holds for arbitrary k then it also holds for k + 1"

#### We want

 $\forall n.P(n)$ 

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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 have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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 have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

- for the moment fix n
- have *P*(0)

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

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

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

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

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

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

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

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

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

- 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)

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

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

- . . .
- have P(n − 1)
- have  $P(n-1) \rightarrow P(n)$

#### We have

- P(0) "property P holds for 0"
- ∀k.(P(k) → 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"

- for the moment fix n
- have *P*(0)
- have P(0) 
  ightarrow 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)

anything that depends on some variable and is either true or false can be seen as function p : 'a -> bool

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

### Example

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

### Example

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

### Example

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: P(0)

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0)$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$
  
IH:  $P(k) = (1+2+\cdots+k = \frac{k \cdot (k+1)}{2})$ 

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$
  
IH:  $P(k) = (1+2+\cdots+k = \frac{k \cdot (k+1)}{2})$   
show:  $P(k+1)$ 

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$
  
IH:  $P(k) = (1+2+\cdots+k = \frac{k \cdot (k+1)}{2})$   
show:  $P(k+1)$ 

$$1+2+\cdots+(k+1)$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$
  
IH:  $P(k) = (1+2+\cdots+k = \frac{k \cdot (k+1)}{2})$   
show:  $P(k+1)$ 

$$1 + 2 + \dots + (k + 1) = (1 + 2 + \dots + k) + (k + 1)$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$
  
IH:  $P(k) = (1+2+\cdots+k = \frac{k \cdot (k+1)}{2})$   
show:  $P(k+1)$ 

$$1 + 2 + \dots + (k + 1) = (1 + 2 + \dots + k) + (k + 1)$$
  
$$\stackrel{\text{\tiny IH}}{=} \frac{k \cdot (k + 1)}{2} + (k + 1)$$

anything that depends on some variable and is either true or false can be seen as function p : 'a -> <code>bool</code>

• 
$$P(x) = (1 + 2 + \dots + x = \frac{x \cdot (x+1)}{2})$$

• base case: 
$$P(0) = (1 + 2 + \dots + 0 = 0 = \frac{0 \cdot (0+1)}{2})$$

• step case: 
$$P(k) \rightarrow P(k+1)$$
  
IH:  $P(k) = (1+2+\cdots+k = \frac{k \cdot (k+1)}{2})$   
show:  $P(k+1)$ 

$$1 + 2 + \dots + (k + 1) = (1 + 2 + \dots + k) + (k + 1)$$
$$\stackrel{\text{\tiny IH}}{=} \frac{k \cdot (k + 1)}{2} + (k + 1)$$
$$= \frac{(k + 1) \cdot (k + 2)}{2}$$



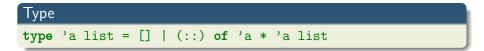
- of course the base case can be changed
- e.g., if base case P(1), property holds for all  $n \ge 1$

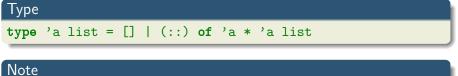
#### Overview

73+SIGI

#### Week 7 - Induction

- Summary of Week 6
- Mathematical Induction
- Induction Over Lists





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

#### 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

#### 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

# $(P([]) \land \forall x \quad . \forall xs \quad .(P(xs) \rightarrow P(x :: xs))) \ \rightarrow \forall ls \quad .P(ls)$

#### 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

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

#### 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

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

#### 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

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

#### Remarks

- $y : \beta$  reads 'y is of type  $\beta$ '
- for lists, P can be seen as function p : 'a list -> bool

### Example - Lst.append

#### Recall

```
let rec (0) xs ys = match xs with
    [] -> ys
    | x::xs -> x :: (xs 0 ys)
```

### Example - Lst.append

#### Recall

```
let rec (0) xs ys = match xs with
    [] -> ys
    | x::xs -> x :: (xs 0 ys)
```

#### Lemma

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

*xs* @ [] = *xs* 

### Example - Lst.append

#### Recall

#### Lemma

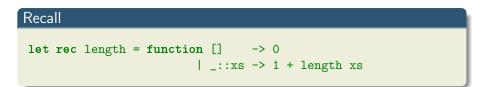
```
[] is right identity of @, i.e.,
```

$$xs @ [] = xs$$

#### Proof.

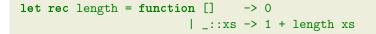
Blackboard

### Example - Lst.length



### Example - Lst.length

#### Recall



#### Lemma

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

length xs + length ys = length(xs @ ys)

### Example - Lst.length

#### Recall

#### Lemma

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

length xs + length ys = length(xs @ ys)

#### Proof.

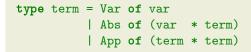
#### Blackboard

#### Overview

#### Week 7 - Induction

- Summary of Week 6
- Mathematical Induction
- Induction Over Lists
- Structural Induction

#### Туре



# Type type term = Var of var | Abs of (var \* term) | App of (term \* term)

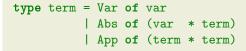
#### Induction Principle

• for every non-recursive constructor there is a base case

# Type type term = Var of var | Abs of (var \* term) | App of (term \* term)

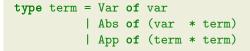
- for every non-recursive constructor there is a base case
  - base case: Var x

# Туре



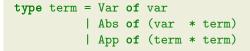
- for every non-recursive constructor there is a base case
  - base case: Var x
- for every recursive constructor there is a step case

#### Туре



- 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)

#### Туре



- 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)

#### Туре

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

$$(P(\texttt{Empty}) \land \forall v . \forall l . \forall r . ((P(l) \land P(r)) \rightarrow P(\texttt{Node}(l, v, r)))) \rightarrow \forall t . P(t)$$

#### Recall

### Туре

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

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

#### Example - Trees

#### Definition (Perfect Binary Trees)

binary tree is perfect if all leaf nodes have same depth

#### Example - Trees

#### Definition (Perfect Binary Trees)

binary tree is perfect if all leaf nodes have same depth

### Example - Trees (cont'd)

## Recall let rec height = function | Empty -> 0 | Node(1,\_,r) -> max (height 1) (height r) + 1 let rec size = function Empty -> 0 | Node(1,\_,r) -> size 1 + size r + 1

### Example - Trees (cont'd)

### Recall let rec height = function | Empty -> 0 | Node(1,\_,r) -> max (height 1) (height r) + 1 let rec size = function Empty -> 0 | Node(1,\_,r) -> size 1 + size r + 1

#### Lemma

perfect binary tree t of height n has exactly  $2^n - 1$  nodes

### Example - Trees (cont'd)

#### Recall

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

#### Lemma

perfect binary tree t of height n has exactly  $2^n - 1$  nodes

#### Proof.

To show: 
$$P(t) = ig( ext{perfect}\ t o ( ext{size}\ t = 2^{( ext{height}\ t)} - 1)ig)$$
Blackboard