Functional Programming

WS 2007/08

Christian Sternagel[1] (VO + PS)
Friedrich Neurauter[2] (PS)
Harald Zankl[3] (PS)

Computational Logic
Institute of Computer Science
University of Innsbruck

30 November 2007

[1] christian.sternagel@uibk.ac.at
[2] friedrich.neurauter@uibk.ac.at
[3] harald.zankl@uibk.ac.at

Overview

Overview

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

Intuitively

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)

Overview

Week 7 - Induction

When?

Goal

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

Formally

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

How?

To show

- $P(0)$
- $\forall k.(P(k) \to P(k+1))$

Why Does This Work?

We have

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

We want

$\forall n.P(n)$ "P holds for arbitrary n"

We get

- for the moment fix n
- have $P(0)$
- have $P(0) \to P(1)$
- have $P(1)$
- have $P(1) \to P(2)$
- ...
- have $P(n-1)$
- have $P(n-1) \to P(n)$
- hence $P(n)$

What is Ment by ‘Property’?

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

Example

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

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

Week 7 - Induction Mathematical Induction

Remark

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

CS (ICS@UIBK) FP OCaml Bash

Overview

Recall

Type

$$\textbf{type}\ \text{'a list} = \underbrace{\text{Nil}}_{[]}\ |\ \underbrace{\text{Cons}\ \textbf{of}\ \text{'a} * \text{'a list}}_{\_ :: \_}$$

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

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

Remarks

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

Example - Lst.length

Recall

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

Lemma

adding element to list increases length by one, i.e.,

$$\text{length } (x :: xs) = \text{length } xs + 1$$

for arbitrary x

Proof.

Blackboard □

Example - Lst.append

Recall

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

Lemma

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

$$xs \mathbin{@} [] = xs$$

Proof.

Blackboard □

Overview

General Structures

Type

type arith = Var **of** char | Const **of** int | Add **of** arith ∗ arith

Induction Principle

- for every non-recursive constructor there is a base case
  - base case: Var x
  - base case: Const i
- for every recursive constructor there is a step case
  - step case: Add (s, t)

Week 7 - Induction Structural Induction

Induction Principle on General Structures

Intuition

- to show $P(s)$ for all structures s
- show base cases
- show step cases

CS (ICS@UIBK) FP OCaml Bash

Recall

Type

type 'a btree = Empty | Node **of** 'a btree * 'a * 'a btree

Induction Principle

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

Example - Trees

Definition (Perfect Binary Trees)

binary tree is perfect if all leaf nodes have same depth

Lemma

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

Proof.

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

Blackboard □