F4

Structural induction and Lambda calculus

Lennart Andersson

Revision 2009-03-19

2009

Peano's axioms

1. $0 \in \mathbb{N}$.
2. $n \in \mathbb{N} \Rightarrow n' \in \mathbb{N}$.
3. $n \in \mathbb{N} \Rightarrow n' \neq 0$.
4. If $n, m \in \mathbb{N}$ then $n' = m' \Rightarrow n = m$.
5. If
   - $A \subseteq \mathbb{N}$
   - $0 \in A$
   - $n \in A \Rightarrow n' \in A$

   then $A = \mathbb{N}$.

Haskell representation

```
data N = Zero | Suc N

add Zero m = m
add (Suc n) m = Suc (add n m)
```


Induction over N

Theorem. *Let $p \in N \to \mathbb{B}$ be a property. If*

1. *$p(Zero)$ is true and*
2. *$p(k)$ implies $p(Suc\ k)$ for every $k \in N$*

then $p(n)$ is true for all $n \in N$. ■

Induction over Expr

```
data Expr = Num Integer | Add Expr Expr | Mul Expr Expr
```

Theorem. *Let $p \in Expr \to \mathbb{B}$ be a property. If*

1. *$p($`Num n`$)$ is true for all `n`.*
2. *$p($`e1`$)$ and $p($`e2`$)$ implies $p($`Add e1 e2`$)$ for every `e1, e2` $\in$ `Expr`*
3. *$p($`e1`$)$ and $p($`e2`$)$ implies $p($`Mul e1 e2`$)$ for every `e1, e2` $\in$ `Expr`*

then $p(e)$ is true for all $e \in$ `Expr`. ■

Mirror example

```
value :: Expr -> Integer
value (Num n) = n
value (Add expr1 expr2) = value expr1 + value expr2
value (Mul expr1 expr2) = value expr1 * value expr2

mirror :: Expr -> Expr
mirror (Num i) = Num i
mirror (Add e1 e2) = Add (mirror e2) (mirror e1)
mirror (Mul e1 e2) = Mul (mirror e2) (mirror e1)
```


Application of the principle

Theorem.

$$\texttt{value}\ e = \texttt{value}(\texttt{mirror}\ e)$$

for all `e` in `Expr`. ■

Induction principle

Theorem. *Let $p \in Expr \to \mathbb{B}$ be a property. If*

1. *$p($`Num n`$)$ is true for all `n`.*
2. *$p($`e1`$)$ and $p($`e2`$)$ implies $p($`Add e1 e2`$)$ for every `e1, e2` $\in$ `Expr`*
3. *$p($`e1`$)$ and $p($`e2`$)$ implies $p($`Mul e1 e2`$)$ for every `e1, e2` $\in$ `Expr`*

then $p(e)$ is true for all $e \in$ `Expr`. ■

Lambda calculus grammars

Concrete grammar

$$term ::= id \mid (\lambda\, id\, .\, term) \mid (term\ term)$$

Haskell representation

```
data Lambda = Id String
            | Abstr String Lambda
            | Appl Lambda Lambda
```

Abstract grammar for **Lexp** (Nielsen style)

$$M ::= x \mid \lambda x\, .\, M \mid M_1\ M_2$$

Simplifed concrete syntax

Applications associate to the left

$$M_1\ M_2\ M_3\ \ldots\ M_n \triangleq (\ldots((M_1\ M_2)\ M_3)\ \ldots\ M_n)$$

Abstractions associate to the right

$$\lambda\sigma_1\, .\, \lambda\sigma_2 \ldots . \lambda\sigma_n\, .\, M \triangleq (\lambda\sigma_1.(\lambda\sigma_2\, .\, (\ldots\ (\lambda\sigma_n\, .\, M)\ldots)))$$

Abstractions extends as far as possible

$$\lambda\sigma\, .\, M_1\ M_2\ M_3\ \ldots\ M_n \triangleq \lambda\sigma.(M_1\ M_2\ M_3\ \ldots\ M_n)$$

Free and bound identifiers

$$\begin{aligned}\mathcal{F}(\sigma) &\triangleq \{\,\sigma\,\}\\ \mathcal{F}(\lambda\sigma\, .\, M) &\triangleq \mathcal{F}(M) \setminus \{\,\sigma\,\}\\ \mathcal{F}(M_1\ M_2) &\triangleq \mathcal{F}(M_1) \cup \mathcal{F}(M_2)\end{aligned}$$

$$\begin{aligned}\mathcal{B}(\sigma) &\triangleq \emptyset\\ \mathcal{B}(\lambda\sigma\, .\, M) &\triangleq \mathcal{B}(M) \cup \{\,\sigma\,\}\\ \mathcal{B}(M_1\ M_2) &\triangleq \mathcal{B}(M_1) \cup \mathcal{B}(M_2)\end{aligned}$$

$$\mathcal{I}(M) \triangleq \mathcal{F}(M) \cup \mathcal{B}(M)$$