COSC 3015: Lecture 7

Lecture given by Prof. Caldwell and scribed by Sunil Kothari

September 16, 2008

1 Inductive Datatype

```
data Nat = Zero | Succ Nat
```

There is recursion in the datatype equation. We are defining the datatype *Nat* - and in doing so, use the data type *Nat*.

Zero and *Succ* are constructors for the data type.
Zero::Nat - is a constant of the type nat.
Succ:: Nat $\rightarrow$ *Nat* - is a constant that maps Nats to Nats

```
Main> :t Zero
Zero :: Nat
Main> :t Succ Zero
Succ Zero :: Nat
Main>
```

To apply the constructor Succ - we must have a previously constructed Nat to apply to it. What objects have type Nat ?? $Nat = \{Zero, SuccZero, Succ(SuccZero), \cdots\}$

This representation goes back to Peano, an Italian logician and mathematician. In Haskell, we have the most refined form of equality.

```
instance Eq Nat where
  Zero == Zero = True
  Zero == Succ n = False
  Succ n == Zero = False
  Succ n == Succ m = m == n
```

```
Main> :t (==)
(==) :: Eq a => a -> a -> Bool
```

By default, we also get not equal to

```
:t (/=)
(/=) :: Eq a => a -> a -> Bool
Main>
```

If we had said data Nat = Zero |Succ Nat deriving Eq

Haskell would have derived an equivalent == function to one we have written.

Succ Zero == Succ (Succ Zero)
$\leadsto$ Zero == Succ Zero
$\leadsto$ False

Nat terms have a tree structure.

We can define our own $+\!+\!+$ as :

$(+\!+\!+) :: Nat \rightarrow Nat \rightarrow Nat$
$m +\!+\!+ \; zero = m$
$m +\!+\!+ \; (Succ\, n) = Succ\ (n +\!+\!+ \; m)$

Theorem 1. $\forall m : Nat.m +\!+\!+ \; (succ\ 0) = succ\ m$

Proof. Choose an arbitrary m and show $m +\!+\!+ \; (succ\ 0) = succ\ m$. Starting with LHS

$$\begin{aligned} m +\!+\!+ \; (succ\ 0) &\overset{\text{<<defn of +++>>}}{=} succ\ (m +\!+\!+ \; zero) \\ &= succ\ m \end{aligned}$$

□

Similarly, we can check if zero is same as our notion of 0 in mathematics

Theorem 2. $\forall m : Nat.zero +\!+\!+ \; m = m$

Proof by induction on m. . □

1.1 Induction Principle

Consider again inductively defined data type.
data Nat = Zero | Succ Nat deriving Eq
The induction principle for Nat. (as is in the book)

Case (Zero): $P(Zero)$ holds
Case (succ n): Assuming $P(n)$ holds, show $P(succ\, n)$ holds.

What is P?? $P :: Nat \rightarrow Bool$ - P is a property of natural numbers.

Another way of thinking this is :

$$(P(0) \wedge \forall m : Nat.P(m) \Rightarrow P(m+1)) \Rightarrow \forall m : Nat.P(m)$$

Theorem 3. $\forall m, n : Nat.n +\!+\!+ \; m = m +\!+\!+ \; n$

By induction on m. What is P(m) ? we can look at the statement as $\forall m, \forall n : Nat.n +++ m = m +++ n$ So the property is $P(m) \stackrel{\text{def}}{=} \forall n : Nat.n{+}m = m{+}n$. So we have to show:

case (P(zero) : $\forall n : Nat.n +++ zero = zero{+}{+}{+}n$. Choose an arbitrary n.

L.HS. $n +++ \quad zero = n$ (by definition of +)

$zero +++ n = n$ (by Lemma 1)

case P(Succ n) : Assume P(k), *i.e.* $\forall n : Nat.n + k = k + n$

Show P(Succ k) - $\forall n : Nat.n{+}{+}{+}\,(Succ\ k) = (Succ\ k) +++ n$

Choose arb. n and show that

$n +++ (succ\ k) = (succ\ k) +++ n$

On the left,

$$
\begin{aligned}
n{+}{+}{+}(Succ\ k) &= succ\ (n +++ k) \text{(by defnition of plus)} \\
&= succ\,(k{+}{+}{+}n) \text{ (by P(k))} \\
&= k + (succ\, n)
\end{aligned}
$$

On the right, since $\forall n : Nat, n + k = k + n$

we know (succ n) +++ k = k +++ (succ n)

□

Lemma 1. $\forall n : Nat.zero +++ n = n$

Proof. By induction on n. What is P(n)? $P(n) \stackrel{\text{def}}{=} zero +++ n = n$

Base case : show $zero +++ zero = zero$. But $zero +++ zero = zero$ so the base case holds.

Induction case show P(succ k): assume $P(k)$ and show $P(succ\ k)$.

$P(k) \stackrel{\text{def}}{=} zero +++ k = k$

$P(succ\ k) \stackrel{\text{def}}{=} zero +++ (succ\ k) = succ\ k$

$$
\begin{aligned}
\text{Start on the left } P(succ\ k) \stackrel{\text{def}}{=} \quad & zero +++ (succ\ k) \\
& = succ\ (zero + k) \\
& = succ\ k
\end{aligned}
$$

So the induction principle holds.

□

We are trying to prove properties of our recursive program. We are proving properties of our haskell function