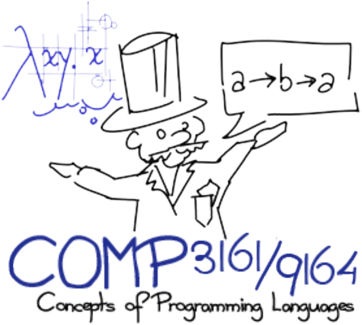

Structural Induction with Haskell

Thomas Sewell
UNSW

Term 3 2024

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

- Show $P(0)$

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

- Show $P(0)$
- Assuming $P(k)$ (the *inductive hypothesis*), show $P(k + 1)$.

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

- Show $P(0)$
- Assuming $P(k)$ (the *inductive hypothesis*), show $P(k+1)$.

Example (Sum of Integers)

Write a recursive function *sumTo* to sum up all integers from 0 to the input n.

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

- Show $P(0)$
- Assuming $P(k)$ (the *inductive hypothesis*), show $P(k+1)$.

Example (Sum of Integers)

Write a recursive function *sumTo* to sum up all integers from 0 to the input n.
Show that:

$$\forall n \in \mathbb{N}.\ \mathit{sumTo}\ n = \frac{n(n+1)}{2}$$

Haskell Data Types

We can define natural numbers as a Haskell data type, reflecting this inductive structure.

$$\textbf{data}\ \mathit{Nat} = \mathsf{Z} \mid \mathsf{S}\ \mathit{Nat}$$

Haskell Data Types

We can define natural numbers as a Haskell data type, reflecting this inductive structure.

$$\textbf{data}\ \mathit{Nat} = \mathrm{Z} \mid \mathrm{S}\ \mathit{Nat}$$

Example

Define addition, prove that $\forall n.\ n + \mathrm{Z} = n$.

Haskell Data Types

We can define natural numbers as a Haskell data type, reflecting this inductive structure.

$$\textbf{data}\ \mathit{Nat} = \mathrm{Z} \mid \mathrm{S}\ \mathit{Nat}$$

Example

Define addition, prove that $\forall n.\ n + \mathrm{Z} = n$.

Inductive Structure

Observe that the non-recursive constructors correspond to base cases and the recursive constructors correspond to inductive cases

Lists

Lists are singly-linked lists in Haskell. The empty list is written as `[]` and a list node is written as `x : xs`. The value `x` is called the head and the rest of the list `xs` is called the tail. Thus:

```
"hi!" == ['h', 'i', '!'] == 'h' : 'i' : '!' : []
                         == 'h':('i':('!':[]))
```

Lists

Lists are singly-linked lists in Haskell. The empty list is written as `[]` and a list node is written as `x : xs`. The value `x` is called the head and the rest of the list `xs` is called the tail. Thus:

```
"hi!" == ['h', 'i', '!'] == 'h' : 'i' : '!' : []
                         == 'h':('i':('!':[]))
```

When we define recursive functions on lists, think about the `x : xs`/`[]` representation to write pattern matches.

Example

(Re)-define the functions *length*, *take* and *drop*.

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\textbf{data}\ \mathit{List}\ a = \mathrm{Nil} \mid \mathrm{Cons}\ a\ (\mathit{List}\ a)$$

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\textbf{data}\ \mathit{List}\ a = \mathsf{Nil} \mid \mathsf{Cons}\ a\ (\mathit{List}\ a)$$

Induction

If we want to prove that a proposition holds for all lists:

$$\forall xs.\ P(xs)$$

It suffices to:

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\textbf{data}\ \mathit{List}\ a = \mathsf{Nil} \mid \mathsf{Cons}\ a\ (\mathit{List}\ a)$$

Induction

If we want to prove that a proposition holds for all lists:

$$\forall xs.\ P(xs)$$

It suffices to:

1. Show $P([\,])$ (the base case from nil)

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\textbf{data}\ \mathit{List}\ a = \mathsf{Nil} \mid \mathsf{Cons}\ a\ (\mathit{List}\ a)$$

Induction

If we want to prove that a proposition holds for all lists:

$$\forall xs.\ P(xs)$$

It suffices to:

1. Show $P(\texttt{[]})$ (the base case from nil)
2. Assuming the inductive hypothesis $P(\texttt{xs})$, show $P(\texttt{x:xs})$ (the inductive case from cons).

Induction on Lists

Example (Take and Drop)

- Show that *take* (*length xs*) *xs* = *xs* for all *xs*.

Induction on Lists

Example (Take and Drop)

- Show that $take\ (length\ xs)\ xs = xs$ for all xs.
- Show that $take\ 5\ xs \mathbin{++} drop\ 5\ xs = xs$ for all xs.

Induction on Lists

Example (Take and Drop)

- Show that $take\ (length\ xs)\ xs = xs$ for all xs.
- Show that $take\ 5\ xs +\!\!+\ drop\ 5\ xs = xs$ for all xs.

  $\Longrightarrow$ Sometimes we must generalise the proof goal.

Induction on Lists

Example (Take and Drop)

- Show that $take\ (length\ xs)\ xs = xs$ for all xs.
- Show that $take\ 5\ xs\ {+}{+}\ drop\ 5\ xs = xs$ for all xs.

$\Longrightarrow$ Sometimes we must generalise the proof goal.

$\Longrightarrow$ Sometimes we must prove auxiliary lemmas.

Binary Trees

$$\begin{array}{lcl}\textbf{data}\ \mathit{Tree}\ a & = & \mathsf{Leaf} \\ & \mid & \mathsf{Branch}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)\end{array}$$

Binary Trees

$$\textbf{data}\ \mathit{Tree}\ a \;=\; \mathrm{Leaf} \;\mid\; \mathrm{Branch}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

Induction Principle

To prove a property $P(t)$ for all trees t:

- Prove the base case $P(\mathit{Leaf})$.

Binary Trees

$$\textbf{data}\ \mathit{Tree}\ a \ = \ \text{Leaf} \ \mid \ \text{Branch}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

Induction Principle

To prove a property $P(t)$ for all trees t:

- Prove the base case $P(\mathit{Leaf})$.
- Assuming the two *inductive hypotheses*:

Binary Trees

$$\textbf{data}\ \mathit{Tree}\ a \;=\; \text{Leaf} \;\mid\; \text{Branch}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

Induction Principle

To prove a property $P(t)$ for all trees t:

- Prove the base case $P(\mathit{Leaf})$.
- Assuming the two *inductive hypotheses*:
  - $P(l)$ and
  - $P(r)$

Binary Trees

$$\textbf{data}\ \mathit{Tree}\ a = \mathrm{Leaf} \mid \mathrm{Branch}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

Induction Principle

To prove a property $P(t)$ for all trees t:

- Prove the base case $P(\mathit{Leaf})$.
- Assuming the two *inductive hypotheses*:
  - $P(l)$ and
  - $P(r)$

  We must show $P(\mathit{Branch}\ x\ l\ r)$.

Binary Trees

$$\begin{array}{lcl}\textbf{data}\ \mathit{Tree}\ a & = & \text{Leaf} \\ & \mid & \text{Branch}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)\end{array}$$

Induction Principle

To prove a property $P(t)$ for all trees t:

- Prove the base case $P(\mathit{Leaf})$.
- Assuming the two *inductive hypotheses*:
  - $P(l)$ and
  - $P(r)$

  We must show $P(\mathit{Branch}\ x\ l\ r)$.

Example (Tree functions)

Define *leaves* and *height*, and show $\forall t.\ \mathit{height}\ t < \mathit{leaves}\ t$

Rose Trees

$$\textbf{data}\ \mathit{Forest}\ a = \mathrm{Empty} \mid \mathrm{Cons}\ (\mathit{Rose}\ a)\ (\mathit{Forest}\ a)$$

$$\textbf{data}\ \mathit{Rose}\ a = \mathrm{Node}\ a\ (\mathit{Forest}\ a)$$

Note that *Forest* and *Rose* are defined *mutually*.

Rose Trees

$$\textbf{data}\ \mathit{Forest}\ a = \mathsf{Empty} \mid \mathsf{Cons}\ (\mathit{Rose}\ a)\ (\mathit{Forest}\ a)$$

$$\textbf{data}\ \mathit{Rose}\ a = \mathsf{Node}\ a\ (\mathit{Forest}\ a)$$

Note that *Forest* and *Rose* are defined *mutually*.

Example (Rose tree functions)

Define *size* and *height*, and try to show

$$\forall t.\ \mathit{height}\ t \leq \mathit{size}\ t$$

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

$$\textbf{data}\ \mathit{Forest}\ a = \text{Empty} \mid \text{Cons}\ (\mathit{Rose}\ a)\ (\mathit{Forest}\ a)$$

$$\textbf{data}\ \mathit{Rose}\ a = \text{Node}\ a\ (\mathit{Forest}\ a)$$

Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and a property $Q(ts)$ about all *Forest*s ts simultaneously:

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

$$\textbf{data}\ \mathit{Forest}\ a = \mathrm{Empty} \mid \mathrm{Cons}\ (\mathit{Rose}\ a)\ (\mathit{Forest}\ a)$$

$$\textbf{data}\ \mathit{Rose}\ a = \mathrm{Node}\ a\ (\mathit{Forest}\ a)$$

Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and a property $Q(ts)$ about all *Forests* ts simultaneously:

- Prove $Q(\mathit{Empty})$

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

$$\textbf{data}\ \mathit{Forest}\ a = \text{Empty} \mid \text{Cons}\ (\mathit{Rose}\ a)\ (\mathit{Forest}\ a)$$

$$\textbf{data}\ \mathit{Rose}\ a = \text{Node}\ a\ (\mathit{Forest}\ a)$$

Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and a property $Q(ts)$ about all *Forests* ts simultaneously:

- Prove $Q(\mathit{Empty})$
- Assuming $P(t)$ and $Q(ts)$ (inductive hypotheses), show $Q(\mathit{Cons}\ t\ ts)$.

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

$$\textbf{data}\ \mathit{Forest}\ a = \text{Empty} \mid \text{Cons}\ (\mathit{Rose}\ a)\ (\mathit{Forest}\ a)$$

$$\textbf{data}\ \mathit{Rose}\ a = \text{Node}\ a\ (\mathit{Forest}\ a)$$

Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and a property $Q(ts)$ about all *Forests* ts simultaneously:

- Prove $Q(\mathit{Empty})$
- Assuming $P(t)$ and $Q(ts)$ (inductive hypotheses), show $Q(\mathit{Cons}\ t\ ts)$.
- Assuming $Q(ts)$ (inductive hypothesis), show $P(\mathit{Node}\ x\ ts)$.