# Lecture 7: Trees and Structural Induction

Natural numbers, positive numbers, rational numbers, pairs, tuples etc. are the objects that we usually study in discrete math. But when we consider discrete math in terms of information technology, we find many more structure in information then just numbers. For example, we can find information that is captured in lists. Or trees, for this matter. In an object oriented language, classes are organized in form of inheritance trees, there are decisions trees, or game trees. HTML, XML, and the like are trees that describe documents. Trees are therefore interesting and important. We will need to learn more about them.

Let's spend some time understanding lists. Lists are different form sets. They are ordered. Elements can appear several times in a list, but they only appear one time in a list. We use the :: notation for lists. For example 2::3::5::7::11::nil is a list. We can think of a list as being constructed from a head 2, and the rest of the list 5::7::11::nil. Sometimes, I will just write $[2, 3, 5, 7, 11]$ as an abbreviation. Recall that we introduced the principle of induction for natural number: During a previous lecture, we convinced ourselves that if

$$P(0)$$

and

$$\forall n \in \mathbb{N}.P(n) \rightarrow P(n+1)$$

then we may conclude that

$$\forall n \in N.P(n)$$

Lists are constructed just like natural numbers. Natural numbers are constructed by 0 and the successor operation, lists are constructed by nil and the :: (over elements from some set $S$). By analogy we can come up with an induction principle for lists.

If

$$P(nil)$$

and

$$\forall x \in S.\forall l \in \mathbb{L}.P(l) \rightarrow P(x :: l)$$

then we may conclude that

$$\forall l \in \mathbb{L}.P(l)$$

Let's demonstrate how to use this by the means of an example. From last lecture, we had some examples already, lists of coins, like nickels, dimes, and quarters. Or bits, such as 0s and 1s.

**Definition 34** *Let $\mathbb{B}$ be the set of all finite binary strings, which are lists of the set $0, 1$.*

For example $0 :: 0 :: 1 :: 0 :: 1 :: 0 :: nil \in \mathbb{B}$, $1 :: 1 :: 0 :: 1 :: 0 :: nil \in \mathbb{B}$, and $1 :: 0 :: 0 :: 1 :: nil \in \mathbb{B}$. Of course, we prefer the less verbose form $001010 \in \mathbb{B}$, $11010 \in \mathbb{B}$, and $1001 \in \mathbb{B}$, which we will use hence force.

We can define a simple functions on lists, for example:

$$l_1 @ l_2 = \begin{cases} l_2 & \text{if } l_1 = \text{nil} \\ x::(l_1' @ l_2) & \text{if } l_1 = x::l_1' \end{cases}$$

And we may now apply the induction principle to show properties about lists.

**Lemma 35 (Appending an empty list from the right)** *For all list $l \in \mathbb{L}$, the following holds:* $l@\text{nil} = l$

**Proof:** by induction on the structure of $l$.

**Case:** $l = nil$. The claim follows directly from the definition of @.

**Case:** $l = x::l'$.

| | |
|---|---|
| Let $l'$ be arbitrary and fixed. | |
| Assume $l'@\text{nil} = l'$ | the induction hypothesis |
| $(x::l')@\text{nil}$ | |
| $= x::(l'@\text{nil})$ | by definition of @ |
| $= x::l'$ | by induction hypothesis |

□

**Theorem 36 (Append is associative)** $l_1@(l_2@l_3) = (l_1@l_2)@l_3$

**Proof:** $l_1 = nil$.

| | |
|---|---|
| $\text{nil}@l_2 = l_2$ | by definition of append |
| $\text{nil}@(l_2@l_3) = l_2@l_3$ | by definition of append |
| $\text{nil}@(l_2@l_3) = l_2@l_3 = (\text{nil}@l_2)@l_3$ | by equality reasoning |

**Case**

| | |
|---|---|
| Let $l_1'$ be arbitrary but fixed | |
| Assume $l_1 = X :: l_1'$ | |
| Assume $l_1'@(l_2@l_3) = (l_1'@l_2)@l_3$ | |
| $(X :: l_1')@(l_2@l_3)$ | |
| $= X :: (l_1'@(l_2@l_3))$ | |
| $= X :: ((l_1'@l_2)@l_3)$ | |
| $= (X :: (l_1'@l_2))@l_3$ | |
| $= ((X :: l_1')@l_2)@l_3$ | |

□

Here is another common operation on lists: list reversal.

$$\begin{array}{lcl} \mathrm{rev}(l) & = & \mathrm{rev'}(l, \mathrm{nil}) \\ \text{where} & & \\ \mathrm{rev'}(l_1, l_2) & = & \begin{cases} l_2 & \text{if } l_1 = \mathrm{nil} \\ \mathrm{rev'}(l_1', x::l_2))) & \text{if } l_1 = x::l_1' \end{cases} \end{array}$$

Can we prove mathematically, that the reverse of the reverse of a list is the list itself?

**Conjecture 37** *For all $l \in \mathbb{L}$, it holds that* $\mathrm{rev}(\mathrm{rev}(l)) = l$.

This however, we cannot prove directly. We need to first develop a lemma, stating a useful property about rev', which is on the one hand true, and on the other necessary to prove the conjecture above.

**Lemma 38** *For all $l_1, l_2, l_3 \in \mathbb{L}$, it holds that* $\mathrm{rev'}(\mathrm{rev'}(l_1, l_2), l_3) = \mathrm{rev'}(l_2, l_1 @ l_3)$.

**Proof:** by structure induction on $l_1$

**Case:** $l_1 = \mathrm{nil}$.

$\mathrm{rev'}(\mathrm{rev'}(\mathrm{nil}, l_2), l_3)$
$= \mathrm{rev'}(l_2, l_3)$ by definition of rev'
$= \mathrm{rev'}(l_2, \mathrm{nil} @ l_3)$ by definition of @

**Case:** $l_1 = x::l_1'$.

Assume forall $l_2 \in \mathbb{L}$ it holds that $\mathrm{rev'}(\mathrm{rev'}(l_1', l_2), l_3) = \mathrm{rev'}(l_2, l_1' @ l_3)$ by induction hypothesis
$\mathrm{rev'}(\mathrm{rev'}(x::l_1', l_2), l_3)$
$\mathrm{rev'}(\mathrm{rev'}(l_1', x::l_2), l_3)$ by definition of rev'
$\mathrm{rev'}(x::l_2, l_1' @ l_3)$ by induction hypothesis
$\mathrm{rev'}(l_2, x::(l_1' @ l_3))$ by definition of rev'
$\mathrm{rev'}(l_2, (x::l_1') @ l_3))$ by definition of @

□

Now we can finish the proof the conjecture:

**Proof:**

$\mathrm{rev}(\mathrm{rev}(l))$
$= \mathrm{rev}(\mathrm{rev'}(l, \mathrm{nil}))$ by definition of rev
$= \mathrm{rev'}(\mathrm{rev'}(l, \mathrm{nil}), \mathrm{nil})$ by definition of rev
$= \mathrm{rev'}(\mathrm{nil}, l @ \mathrm{nil})$ by the above lemma
$= l @ \mathrm{nil}$ by definition of rev'
$= l$ by Lemma 35

| | |
|---|---|
| No information | $\mathcal{T}_0$ |
| Information on in the leaves | $\mathcal{T}_1$ |
| Information on in the nodes | $\mathcal{T}_2$ |
| Information on in the leaves and nodes | $\mathcal{T}_3$ |

Figure 2: Sets of trees

□

Let's look at another example of structures, which are a little bit more complicated than lists: Trees. Trees are constructed very similar to lists, except that the :: symbol plays now the role of an internal node that can have one or more children. If all nodes in a tree have the same number of children, let's say $m$, then we call the tree an $m$-ary tree.

In this sense, a list is simply a *unary* tree, where each node has only one child, namely the tail of the list. The leaf of a tree corresponds to the nil. A tree is a *binary* tree, if every node has exactly two children. For the purpose of this lecture, we consider only binary trees, but all results generalize directly to more general trees.

Trees exhibit more structure than lists which allows us to represent and store information in various places in a tree. For example, considering a tree of natural numbers, we can choose to store numbers as part of the nodes, within the leaves, or both, or none. We introduce the abbreviations for the sets of trees depicted in Figure 2.

Trees also give raise to induction principles. Here is the induction principle for trees in $\mathcal{T}_3$: If

$$\forall x \in S.P(\text{leaf}(x))$$

and

$$\forall x \in S.\forall t_1 \in \mathcal{T}_3.\forall t_2 \in \mathcal{T}_3.P(t_1) \wedge P(t_2) \to P(\text{node}(x, t_1, t_2))$$

then we may conclude that

$$\forall t \in \mathcal{T}_3.P(t)$$

If we consider only trees with information at the leaves, we obtain the following principles for $\mathcal{T}_1$. If

$$\forall x \in S.P(\text{leaf}(x))$$

and

$$\forall t_1 \in \mathcal{T}_1.\forall t_2 \in \mathcal{T}_1.P(t_1) \wedge P(t_2) \to P(\text{node}(t_1, t_2))$$

then we may conclude that

$$\forall t \in \mathcal{T}_1.P(t)$$

**Exercise 3** Write out the induction principles for $\mathcal{T}_0$ and $\mathcal{T}_2$.

With these induction principles, we can prove many interesting things. Actually, the induction principles on trees will For example, consider the mirror operation:

$$\text{mirror}(t) = \begin{cases} \text{leaf}(x) & \text{if } t = \text{leaf}(x) \\ \text{node}(\text{mirror}(t_1), \text{mirror}(t_2)) & \text{if } t = \text{node}(t_1, t_2) \end{cases}$$

Now we can prove

**Theorem 39** *For all trees $t \in \mathcal{IT}_1$:* $\text{mirror}(\text{mirror}(t)) = t$.

**Proof:** by induction on $t$, which means it is a $\mathcal{IT}_1$ induction.

**Case:** $t = \text{leaf}(x)$

$\text{mirror}(\text{mirror}(\text{leaf}(x))) = \text{leaf}(x)$ by definition of mirror, apply twice.

**Case:** $t = \text{node}(t_1, t_2)$

Assume $t_1, t_2$ arbitrary but fixed trees
Assume $\text{mirror}(\text{mirror}(t_1)) = t_1$ as induction hypothesis
Assume $\text{mirror}(\text{mirror}(t_2)) = t_2$ as induction hypothesis
$\text{mirror}(\text{mirror}(\text{node}(t_1, t_2)))$
$= \text{mirror}(\text{node}(\text{mirror}(t_2), \text{mirror}(t_1)))$
$= \text{node}(\text{mirror}(\text{mirror}(t_1)), \text{mirror}(\text{mirror}(t_2)))$
$= \text{node}(t_1, t_2)$ by equality reasoning

□

Besides mirroring, there are some other very useful operations on trees for example, converting a tree into a list by traversing the tree in preorder, i.e. depth first left to right manner. Let's consider $\mathcal{IT}_3$ trees for this:

$$\text{preorder}(t) = \begin{cases} x{::}\text{nil} & \text{if } t = \text{leaf}(x) \\ x{::}(\text{preorder}(t_1)@\text{preorder}(t_2)) & \text{if } t = \text{node}(t_1, t_2) \end{cases}$$

$$\text{inorder}(t) = \begin{cases} x{::}\text{nil} & \text{if } t = \text{leaf}(x) \\ \text{inorder}(t_1)@(x{::}\text{inorder}(t_2)) & \text{if } t = \text{node}(t_1, t_2) \end{cases}$$

$$\text{postorder}(t) = \begin{cases} x{::}\text{nil} & \text{if } t = \text{leaf}(x) \\ \text{postorder}(t_1)@(\text{postorder}(t_2)) & \text{if } t = \text{node}(t_1, t_2)@(x{::}\text{nil}) \end{cases}$$

**Exercise 4** Show, using the definition or list reversal that $\text{preorder}(t) = \text{rev}(\text{postorder}(t))$.