

## Structural induction

We now introduce a powerful method for proving claims about inductively-defined sets. . .

In structural induction,  
the structure of the inductive proof  
(that every element of a set  $S$  has some property)  
follows the structure of the inductive definition (of  $S$ ).

For instance, to prove that every binary tree  $T$  (over  $A$ ) has a property  $P$ , we would simply argue that

- ▶  $P(\langle \rangle)$ , and
- ▶ for all binary trees  $L$  and  $R$  (and all  $x \in A$ ),  
**if**  $P(L)$  and  $P(R)$ , **then**  $P(\langle L, x, R \rangle)$ .

(Compare the structure of this argument to the structure of the inductive definition of the set of binary trees over  $A$ .)

## Soundness of structural induction

Let  $S$  be an inductively-defined set.

Let  $P(x)$  be a property of  $x$ .

**Claim.** To show that, for all  $x \in S$ ,  $P(x)$ , it suffices to show that:

- ▶ *Basis:* For all initial elements  $x$  of  $S$ ,  $P(x)$ .
- ▶ *Induction:* For each construction rule using elements  $x_1, \dots, x_k$  of  $S$  to construct an element  $x$ , show that

$$\mathbf{if} \ P(x_1), \dots, P(x_k), \ \mathbf{then} \ P(x).$$

Let's verify the soundness of this claim. . .

So assume that we can meet the obligations of the proof by structural induction described above.

What we'll show is that the set

$$S_P = \{ x \mid x \in S, P(x) \}$$

satisfies both the basis and induction parts of the inductive definition of  $S$ . It follows that  $S_P$  is a superset of  $S$  (WHY?), and so that, for every  $x \in S$ ,  $P(x)$ .



## Structural induction example 2

Claim: For every binary tree  $T$  over  $A$ ,

$$2^{\text{depth}(T)+1} \geq \text{nodes}(T) + 1.$$

Proof by structural induction (on the inductive definition of  $BT$  over  $A$ ).

**Basis:**  $2^{\text{depth}(\langle \rangle)+1} = 1 = \text{nodes}(\langle \rangle) + 1.$

**Induction:**  $L, R \in BT$ , and  $x \in A$ .

**IH:**  $2^{\text{depth}(L)+1} \geq \text{nodes}(L) + 1$  and  $2^{\text{depth}(R)+1} \geq \text{nodes}(R) + 1.$

**NTS:**  $2^{\text{depth}(\langle L, x, R \rangle)+1} \geq \text{nodes}(\langle L, x, R \rangle) + 1.$

$$\begin{aligned}
 2^{\text{depth}(\langle L, x, R \rangle)+1} &= 2 \cdot 2^{\text{depth}(\langle L, x, R \rangle)} \\
 &= 2 \cdot 2^{\max(\text{depth}(L), \text{depth}(R))+1} && \text{(defn depth)} \\
 &= 2 \cdot 2^{\max(\text{depth}(L)+1, \text{depth}(R)+1)} \\
 &= 2 \cdot \max(2^{\text{depth}(L)+1}, 2^{\text{depth}(R)+1}) \\
 &\geq 2 \cdot \max(\text{nodes}(L) + 1, \text{nodes}(R) + 1) && \text{(IH)} \\
 &= (2 \cdot \max(\text{nodes}(L), \text{nodes}(R))) + 2 \\
 &\geq (\text{nodes}(L) + \text{nodes}(R)) + 2 \\
 &= (\text{nodes}(L) + \text{nodes}(R) + 1) + 1 \\
 &= \text{nodes}(\langle L, x, R \rangle) + 1 && \text{(defn nodes)}
 \end{aligned}$$

## Another structural induction example

Recall: For any alphabet  $A$ ,  $A^*$  can be defined inductively as follows:

- ▶ **Basis:**  $\Lambda \in A^*$ .
- ▶ **Induction:** If  $x \in A^*$  and  $a \in A$ , then  $ax \in A^*$ .

And here's a recursively defined function we've discussed before:

Take  $r : A^* \rightarrow A^*$  s.t.

$$\begin{aligned}
 r(\Lambda) &= \Lambda \\
 r(ax) &= r(x)a \quad (x \in A^*, a \in A)
 \end{aligned}$$

**Claim:** For all  $x, y \in A^*$ ,  $r(xy) = r(y)r(x)$ .

Hmmm, should we induct on  $x$  or on  $y$ ?

You can try it both ways. One way is much nicer...

For any alphabet  $A$ ,  $A^*$  is defined inductively as follows:

- ▶ *Basis*:  $\Lambda \in A^*$ .
- ▶ *Induction*: If  $x \in A^*$  and  $a \in A$ , then  $ax \in A^*$ .

Take  $r : A^* \rightarrow A^*$  s.t.

$$\begin{aligned} r(\Lambda) &= \Lambda \\ r(ax) &= r(x)a \quad (x \in A^*, a \in A) \end{aligned}$$

---

**Claim:** For all  $x, y \in A^*$ ,  $r(xy) = r(y)r(x)$ .

Take an arbitrary  $y \in A^*$ . We'll prove, by structural induction on the inductive defn of  $A^*$ , that for all  $x \in A^*$ ,

$$r(xy) = r(y)r(x).$$

*Basis:*

$$\begin{aligned} r(\Lambda y) &= r(y) \\ &= r(y)\Lambda \\ &= r(y)r(\Lambda) \quad (\text{defn } r) \end{aligned}$$

*Induction:*  $x \in A^*$ ,  $a \in A$

IH:  $r(xy) = r(y)r(x)$

NTS:  $r(axy) = r(y)r(ax)$

$$\begin{aligned} r(axy) &= r(xy)a \quad (\text{defn } r) \\ &= r(y)r(x)a \quad (\text{IH}) \\ &= r(y)r(ax) \quad (\text{defn } r) \end{aligned}$$

## Format for structural induction proof

**In this class, please use the format of the previous example for proofs by structural induction.**

Essential elements:

- ▶ Identify the claim you are proving by structural induction. It should be a claim about all elements of some inductively-defined set.
- ▶ Say it is proof by structural induction, and clearly indicate the inductive definition on which the proof is based.
- ▶ Clearly label *basis* and *induction* cases as such.
- ▶ In the *induction* case, state IH and NTS.
- ▶ Clearly indicate where and how the IH is used.

## Proving correctness of the insert function

Recall the inductive definition of  $\text{lists}(A)$ :

- ▶ *Basis*:  $\langle \rangle \in \text{lists}(A)$ .
- ▶ *Induction*: If  $x \in A$  and  $L \in \text{lists}(A)$ , then  $x :: L \in \text{lists}(A)$ .

Recall also the following recursive definition of a function from  $\mathcal{N} \times \text{lists}(\mathcal{N})$  to  $\text{lists}(\mathcal{N})$ :

$$\begin{aligned}\text{insert}(x, \langle \rangle) &= \langle x \rangle \\ \text{insert}(x, y :: L) &= \begin{cases} x :: y :: L & , \text{ if } x \leq y \\ y :: \text{insert}(x, L) & , \text{ otherwise} \end{cases}\end{aligned}$$

**Claim:** For all  $L \in \text{lists}(\mathcal{N})$ , for all  $x \in \mathcal{N}$ , if  $L$  is sorted, then  $\text{insert}(x, L)$  is a sorted permutation of  $x :: L$ .

We previously discussed the idea for an argument proving this claim. The idea of that argument can be made precise using structural induction on  $L$ .

Consider any  $x \in \mathcal{N}$ .

We'll show that, for all  $L \in \text{lists}(\mathcal{N})$ , if  $L$  is sorted, then  $\text{insert}(x, L)$  is a sorted permutation of  $x :: L$ .

$$\begin{aligned}\text{insert}(x, \langle \rangle) &= \langle x \rangle \\ \text{insert}(x, y :: L) &= \begin{cases} x :: y :: L & , \text{ if } x \leq y \\ y :: \text{insert}(x, L) & , \text{ otherwise} \end{cases}\end{aligned}$$

---

We'll show that, for all  $L \in \text{lists}(\mathcal{N})$ , if  $L$  is sorted, then  $\text{insert}(x, L)$  is a sorted permutation of  $x :: L$ .

Proof by structural induction on  $L$ .

*Basis:*  $\text{insert}(x, \langle \rangle) = \langle x \rangle$ , which is a sorted permutation of  $x :: \langle \rangle = \langle x \rangle$ .

*Induction:*  $L \in \text{lists}(\mathcal{N})$ ,  $y \in \mathcal{N}$ .

IH: If  $L$  is sorted, then  $\text{insert}(x, L)$  is a sorted permutation of  $x :: L$ .

NTS: If  $y :: L$  is sorted, then  $\text{insert}(x, y :: L)$  is a sorted permutation of  $x :: y :: L$ .

Assume  $y :: L$  is sorted. Consider two cases:

*Case 1:*  $x \leq y$ .

Then  $\text{insert}(x, y :: L) = x :: y :: L$ . Since  $y :: L$  is sorted and  $x \leq y$ ,  $x :: y :: L$  is also sorted. And of course  $x :: y :: L$  is a permutation of  $x :: y :: L$ .

$$\begin{aligned} \text{insert}(x, \langle \rangle) &= \langle x \rangle \\ \text{insert}(x, y :: L) &= \begin{cases} x :: y :: L & , \text{ if } x \leq y \\ y :: \text{insert}(x, L) & , \text{ otherwise} \end{cases} \end{aligned}$$


---

Case 2:  $x > y$ .

So  $\text{insert}(x, y :: L) = y :: \text{insert}(x, L)$ .

Since  $y :: L$  is sorted (by assumption), so is  $L$ .

So by IH we can conclude that  $\text{insert}(x, L)$  is a sorted permutation of  $x :: L$ .

It follows easily that  $y :: \text{insert}(x, L)$  is a permutation of  $x :: y :: L$ .

And to see that  $y :: \text{insert}(x, L)$  is sorted, it is enough to check that every element of  $\text{insert}(x, L)$  is at least  $y$ , since we already know that  $\text{insert}(x, L)$  is sorted. This is easy to check. . .

Notice first that since  $y :: L$  is sorted, every element of  $L$  is at least  $y$ . We also know that  $x$  is at least  $y$ .

So every element of  $x :: L$  is at least  $y$ , and since  $\text{insert}(x, L)$  is a permutation of  $x :: L$ , we can conclude that every element of  $\text{insert}(x, L)$  is at least  $y$ .

## Proving correctness of the sort function

Recall the following recursively-defined function, from  $\text{lists}(\mathcal{N})$  to  $\text{lists}(\mathcal{N})$ :

$$\begin{aligned} \text{sort}(\langle \rangle) &= \langle \rangle \\ \text{sort}(x :: L) &= \text{insert}(x, \text{sort}(L)) \end{aligned}$$

Given the previous result for  $\text{insert}$ , we can prove that for all  $L \in \text{lists}(\mathcal{N})$ ,  $\text{sort}(L)$  is a sorted permutation of  $L$ .

By structural induction on (the inductive definition of)  $L$ .

*Basis:*  $\text{sort}(\langle \rangle) = \langle \rangle$ , which is a sorted permutation of itself.

*Induction:*  $L \in \text{lists}(\mathcal{N})$ ,  $x \in \mathcal{N}$ .

IH:  $\text{sort}(L)$  is a sorted permutation of  $L$ .

NTS:  $\text{sort}(x :: L)$  is a sorted permutation of  $x :: L$ .

Notice that  $\text{sort}(x :: L) = \text{insert}(x, \text{sort}(L))$ , and it remains only to show that  $\text{insert}(x, \text{sort}(L))$  is a sorted permutation of  $x :: L$ . By IH,  $\text{sort}(L)$  is a sorted permutation of  $L$ . Since  $\text{sort}(L)$  is sorted, we know by the result from the previous problem that  $\text{insert}(x, \text{sort}(L))$  is a sorted permutation of  $x :: \text{sort}(L)$ , and it remains only to observe that  $x :: \text{sort}(L)$  is in turn a permutation of  $x :: L$ , which is true since  $\text{sort}(L)$  is a permutation of  $L$ .

## Remarks on these last two correctness proofs

The previous two problems are similar to Problem 12, Section 4.4 in the textbook. But the textbook's version of these problems does not give an adequate account of "correctness" of the sort and insert functions. Let me elaborate. . .

About insert, the textbook asks you to prove that  $\text{insert}(x, L)$  is sorted if  $L$  is. If you try to do that in a straightforward fashion by structural induction on  $L$ , you will likely fail. (Or at least you should!)

Why? Because at a key point in the proof, you will need to conclude from the assumption (via IH) that  $\text{insert}(x, L)$  is sorted that  $y :: \text{insert}(x, L)$  is also sorted. But it doesn't follow that easily. As it turns out, you need to know more about  $\text{insert}(x, L)$  — in particular, you need to know that every element of  $\text{insert}(x, L)$  is at least  $y$ , and although this is indeed the case, it does not follow from the IH.

This is one reason why we needed to talk about permutations. . .

Interestingly, the insert function does have the correctness property that the textbook mentions:

if  $L$  is sorted, then  $\text{insert}(x, L)$  is sorted.

Indeed, this is an immediate consequence of the result we proved above.

There we considered the stronger claim that

if  $L$  is sorted,  
then  $\text{insert}(x, L)$  is a sorted *permutation* of  $x :: L$ .

This stronger property is better for proof by induction because we get to make use of the correspondingly stronger IH.

But the textbook's version of correctness for the insert function is inappropriate for another reason.

It is *intuitively* wrong, because it provides an inadequate characterization of correctness for an insert function.

Can you see why?

To see the difficulty more starkly, suppose that we define a much simpler version of the function:

for all  $x \in \mathcal{N}$  and  $L \in \text{lists}(\mathcal{N})$ ,

$$\text{insert}(x, L) = \langle \rangle.$$

This constant function is intuitively wrong; nonetheless, it has the property that the textbook uses to characterize correctness.

That is, it satisfies the claim of problem 12b, that if  $L$  is sorted, then so is  $\text{insert}(x, L)$ .

By the way, you can see that similarly it would not be sufficient to say that what we require of  $\text{sort}(L)$  is that it be sorted, for every  $L \in \text{lists}(\mathcal{N})$ .

And yet that is all that is proved in the textbook's solution to 12a!

A general conclusion to draw from this discussion is that it is difficult to give adequate specifications concerning correctness of functions (or, say, software). It is generally hard to make our requirements sufficiently explicit. (That is not to say that it isn't worth doing!)

Another, more technical conclusion is that **in proofs by induction, it is often the case that stronger claims are easier to prove (because you get a stronger induction hypothesis).**