CS 311: Computational Structures

James Hook and Tim Sheard

October 1, 2013

2 Inductively Defined Sets

Infinite sets play a large role in the study of computational systems. A natural way to define an infinite set is to use a set of inductive rules. Inductive rules have a pattern that is easy to recognize. This pattern makes it possible to easily define functions over infinite sets, and to prove many properties about these infinite sets using the principle of induction. An inductively defined set usually has several rules.

- A set of *base* rules, which enumerate primitive elements of the set.
- A set of *inductive* rules for combining smaller elements in the set to create larger elements.
- A rule which clarifies that the other rules are the only way to define elements in the set.

A natural example of this is the natural numbers.

2.1 Natural Numbers

Sipser defines the *natural numbers* as the set $\{1, 2, 3, \ldots\}$. That definition is not standard. In lecture we will generally refer to that set as the *counting numbers*.

We will include 0 in our set of *natural numbers*.

2.2 An Inductive Definition for the Natural numbers

To drive this point home, we will sometimes use the *Peano* representation of the natural numbers, building them with the constant Z and the constructor S. The *inductive definition* is as follows:

Let Z be a constant and S a function that takes one argument. The *Peano numbers* are defined inductively as follows:

1. Z is a *Peano number*. (A *base* rule.)
2. If n is a *Peano number* then $S(n)$ is a *Peano number*. (An *inductive* rule).

3. There are no other *Peano numbers.*

Applying this definition, we can see that the set of Peano numbers includes:

$$Z,\ S(Z),\ S(S(Z)),\ \ldots$$

2.2.1 Functions on inductive sets

By making the inductive structure of a definition explicit, we get two benefits. We get a mechanism for defining functions and a mechanism for inductive reasoning.

Consider defining the function *isZero* on the Peano numbers. From the inductive definition we know we only need to consider two cases:

$$\begin{aligned} \mathit{isZero}(Z) &= \mathit{True} \\ \mathit{isZero}(S(n)) &= \mathit{False} \end{aligned}$$

A more interesting example is to define addition on Peano numbers. We will do this inductively (think, using recursion) on the first argument as follows:

$$\begin{aligned} Z + y &= y \\ S(x) + y &= S(x + y) \end{aligned}$$

To see how this works, consider adding $2 + 2$.

$$\begin{aligned} S(S(Z)) + S(S(Z)) &= S(\underline{S(Z) + S(S(Z))}) \\ S(\underline{S(Z) + S(S(Z))}) &= S(S(\underline{Z + S(S(Z))})) \\ S(S(\underline{Z + S(S(Z))})) &= S(S(\underline{S(S(Z))})) \end{aligned}$$

In the first line we expand the definition of + in the S case. That gives us a simpler sub-problem (underlined on the right hand side). We write the whole right handside on the left on the next line, but focus our attention on the underlined sub-term. This subterm is also subject to the S case of +, and we rewrite it to the even simpler sum (also underlined), which is expressed on the third line. On the third line we get to apply the Z case of +, which is not recursive. This gives a direct answer, not just a simpler instance of the problem.

Why is the definition of + a good definition? In particular, why is the S case of + well defined? (Thinking of it as a program, why is this a terminating use of recursion?) Note that while it looks like we have mostly the same symbols on the right hand side of the equation, the first argument to + (in the underlined sub-terms), which is the inductive argument of the definition, is simpler. This is why in the calculation above, the first three left hand sides are simpler problems.

Also, note that because of the the seemingly silly third clause "there are no other Peano numbers," we know that by covering the Z case and the S case we have covered all cases.

2.3 Lists as inductive sets

Many other well known sets can be defined inductively. The set of lists containing natural numbers is an example. Let us define it inductively using the pattern we learned defining the Natual numbers.

> Let *Nil* be a constant and *Cons* a function that takes two arguments. The *Lists* are defined inductively as follows:
>
> 1. *Nil* is a *List*. (A *base* rule.)
> 2. If n is a *natural number* and x is a list, then $(Cons\ n\ x)$ is a *List*. (An *inductive* rule).
> 3. There are no other *Lists*.

Example lists include: Nil , $(Cons\ 3\ Nil)$, and $(Cons\ 2\ (Cons\ 1\ Nil))$.

Using the principle (pattern matching over the different ways to define something) we introduced for defining addition over the natural numbers we can define the length function over lists as follows:

$$\begin{aligned}\text{length}(Nil) &= 0\\ \text{length}(Cons\ n\ xs) &= 1 + \text{length}(xs)\end{aligned}$$

2.3.1 Reasoning about inductive sets

The inductive definition of the Peano numbers also gives us the familiar principle of mathematical induction. Induction is not a just a property of the natural numbers, it is a property of any set we can define inductively.

An *induction princple* is a logical pattern that lets us prove a property of all elements of an inductively defined set. Whenever we have a base case of the inductive definition, like the Z case of the Peano numbers, we get a base case of induction princple. In those cases we show the property directly. When the inductive definition is recursive, like the S case of the Peano numbers, we must show that the property is preserved by applying the constructor.

Consider the associativity property of the + function:

$$a + (b + c) = (a + b) + c$$

We can prove this by induction over the variable a in two steps.

1. We must show it is true when $a = Z$

$$Z + (b + c) = (Z + b) + c$$

2. We must show that the property is preserved when applying the constructor S.

$$a + (b + c) = (a + b) + c \Rightarrow (Sa) + (b + c) = (S(a) + b) + c$$

A detailed proof carries out each step, justifying the step by one of the rules that define +. The scheme for proof we use in this class has several parts.

1. **The facts.**

$$\begin{aligned} \mathit{isZero}(Z) &= \mathit{True} \\ \mathit{isZero}(S(n)) &= \mathit{False} \end{aligned}$$

   This often includes function defintitions over the inductively defiuned sets, but may also include lemmas, axioms, or assumptions. For example we may assume facts like $x + 0 = x$. We are careful to be precise about what we are assuming.

2. **Precise statement of what is being proven.**

   Addition (+) is associative.

$$a + (b + c) = (a + b) + c$$

   We often state the proposistion we are trying to prove in English, and also a precise mathematical description.

3. **The detailed steps, each justified by the facts, and a strategy or *proof idea* we will use**.

   Proof by induction in two parts. One for the base case Z, and one for the induction case $S(a)$. For each part we will transform the lefthand side into the right hand side, justfying each step by one of the facts.

   Part 1. Base case where $a = Z$

$$Z + (b + c) = (Z + b) + c$$
$$Z + (b + c) = b + c \quad \text{(by the Z equation of +)}$$
$$b + c = (Z + b) + c \quad \text{(by the Z equation of +, backwards)}$$

   Part 2. Induction case where $a = S(a)$

   Assume:

$$a + (b + c) = (a + b) + c$$

   Prove

$$(Sa) + (b + c) = (S(a) + b) + c$$

$$(Sa) + (b + c) = S(a + (b + c)) \quad \text{(by the S equation of +)}$$
$$S(a + (b + c)) = S((a + b) + c) \quad \text{(by the assumption)}$$
$$S((a + b) + c) = (S(a + b) + c) \quad \text{(by the S equation of + backwards)}$$
$$(S(a + b) + c) = (S(a) + b) + c \quad \text{(by the Z equation of + backwards)}$$

We can write down the general form of the induction rule for Peano numbers. Let $\mathcal{P}(x)$ be some property of Peano numbers (x ranges over Peano numbers). To show $\mathcal{P}(x)$ for all Peano numbers, show $\mathcal{P}(Z)$ and $\mathcal{P}(x) \to \mathcal{P}(S(x))$.

In symbols, this induction rule is written:

$$\frac{\mathcal{P}(Z) \quad \mathcal{P}(x) \to \mathcal{P}(S(x))}{\forall x.\mathcal{P}(x)}$$

Exercise 2.1 *Write down the defintion of $\mathcal{P}$ for the associativity of $+$ problem.*

2.4 Trees as inductive sets

To see how other inductive structures besides Peano numbers give rise to principles of inductive definition and induction, consider a structure for a binary tree with integer labels on the nodes. For simplicity we will call this a *tree*.

Let E be a constant and T a three-place constructor. A *tree* is defined inductively as follows:

1. The empty tree, E, is a tree.
2. If l and r are trees and i is an Integer, then $T\ l\ i\ r$ is a tree. l is called the left subtree, r the right subtree, and i the label.
3. There are no other trees.

Exercise 2.2 *Draw the following trees as pictures:*

1. E
2. $T\ E\ 17\ E$
3. $T(T\ E\ 3(T\ E\ 4\ E))7(T\ E\ 9\ (T\ E\ 17\ E))$

Exercise 2.3 *Write functions that compute:*

1. *The* depth *of the tree.*
2. *The number of T nodes in the tree (*count*).*
3. *The sum of the labels.*
4. *The in-order traversal of the tree (called* flat*) that returns a list constructed with Nil and $Cons$.*
5. *A predicate isOrdered that returns True exactly when the in-order traversal is in asscending order.*

Symbolically the induction principle for Trees is given

$$\frac{\mathcal{P}(E) \quad \forall i.(\mathcal{P}(l) \land \mathcal{P}(r)) \to \mathcal{P}(T\ l\ i\ r)}{\forall t.\mathcal{P}(t)}$$

Exercise 2.4 *Prove that $length(flat(t)) = count(t)$.*

Exercise 2.5 *Write down the induction principle for Lists defined in section 2.3.*

2.5 Languages as inductive sets

Let the alphabet of our language be the set containing the left and right parentheses: $\{(,)\}$

1. The empty string "" is in the language of balanced parentheses.
2. If A and B are in the language of balanced parentheses, then so is the concatentaion $A\ B$.
3. If A is in the language of balanced parentheses, then so is (A).
4. No other strings are in the language of balanced parentheses.

Sample strings include "", "()", "()()". etc.

Exercise 2.6 *Define a function that counts the number of left parentheses in a string in the language of balanced parentheses.*

Exercise 2.7 *Define the inductive set of strings in the language consisting of all sequences of the digits from the alphabet* $\{1, 2, 3\}$.

2.6 Recursive definitions

Sometimes functions over infinite sets are mutually recursive. This means you will have 2 or more functions that call each other. Each function is written by cases, where the cases enumerate each of the rules in the inductive definition.

Exercise 2.8 *1. Give an inductive definition of the function isEven on the Peano numbers. It should return True if its argument is even and False if its argument is false.*

2. Give a definition of isOdd.

ONe can prove things about mutually recursive functions using conjunctions.

$$(\mathit{isEven}(x) \to \mathit{isOdd}(S(x))) \wedge (\mathit{isOdd}(x) \to \mathit{isEven}(S(x))$$

We want to prove this by induction on x.

The base case we get is

$$(\mathit{isEven}(Z) \to \mathit{isOdd}(S(Z))) \wedge (\mathit{isOdd}(Z) \to \mathit{isEven}(S(Z))$$

Assuming you developed reasonable definitions of *isEven* and *isOdd*, this becomes:

$$(\mathit{True} \to \mathit{True}) \wedge (\mathit{False} \to \mathit{False})$$

Which is true.

The step case we get is:

$$((\mathit{isEven}(x) \to \mathit{isOdd}(S(x))) \wedge (\mathit{isOdd}(x) \to \mathit{isEven}(S(x))) \\ \to ((\mathit{isEven}(S(x)) \to \mathit{isOdd}(S(S(x)))) \wedge (\mathit{isOdd}(S(x)) \to \mathit{isEven}(S(S(x))))$$

In English, we might write this as:

Given inductive hypotheses

1. that $isEven(x)$ implies $isOdd(Sx)$
2. that $isOdd(x)$ implies $isEven(Sx)$

show that

3. $isEven(Sx)$ implies $isOdd(S\ (S\ x))$
4. $isOdd(Sx)$ implies $isEven(S\ (S\ x))$

Consider an arbitrary x, if x is even, then by induction hypothesis 1, Sx is odd. In this case we use the definition of *isEven* to show conclue the $S(Sx)$ is even, establishing (4). If x is odd, then by induction hypothesis 2, Sx is even. By definition of *isOdd* we conclude $S(Sx)$ is odd, establishing (3) as required. This completes the case analysis on x, and the simultaneous proofs of (3) and (4), as required.

3 Formatting Proofs

In this class we will write many proofs. In an effort to make everyones lives easier we will use a standard format for proofs. This way everyone will have a common inderstanding of how to read and write a proof.

There are three parts to a proof

1. The facts or assumptions
2. The assertion we are trying to prove
3. The steps that comprise the proof