CMPSCI 250: Introduction to Computation

Lecture #16: Recursive Definition
David Mix Barrington
9 October 2013

Recursive Definition

- The Peano Axioms for the Naturals
- Pseudo-Java for the Naturals
- Forms of the Fifth Peano Axiom
- Recursion and the Fifth Axiom
- Defining Addition and Multiplication
- Other Recursive Systems

Axioms for the Naturals

- Our mathematical arguments should always be subject to questioning. For any step of reasoning we can ask "Why is that true?".
- The ultimate answers are always definitions because there is no questioning them -- if you and I disagree about how the natural numbers are defined, then we are dealing with two different number systems rather than the same one.

Axioms for the Naturals

- About 100 years ago logicians sought a definition of the natural numbers that was as simple as possible, while still allowing all the familiar properties to be proved. Giuseppe Peano's axioms define the naturals using three undefined terms: "natural", "zero", and "successor".
- The process of axiomatization is similar to the definition of a class in Java, where need to say what the objects in the class are (their data fields) and what can be done with them (the methods they support).

The Five Peano Axioms

- **Zero** is a **natural**.
- Every natural has exactly one **successor**, which is a natural.
- Zero is not the successor of any natural.
- No two naturals have the same successor.
- If you start with zero, and keep taking successors, you eventually reach all of the naturals.

Clicker Question #1

- We've been working with number systems like the "integers mod 7", where the numbers are the congruence classes 0, 1, 2, 3, 4, 5, and 6. "Successor" is easy to define for these numbers. Which of these Peano axioms is *false* for the integers mod 7?
- (a) Zero is a number.
- (b) Every number has exactly one successor.
- (c) No two numbers share a successor.
- (d) Zero is not the successor of any number.

Answer #1

- We've been working with number systems like the "integers mod 7", where the numbers are the congruence classes 0, 1, 2, 3, 4, 5, and 6. "Successor" is easy to define for these numbers. Which of these Peano axioms is *false* for the integers mod 7?
- (a) Zero is a number.
- (b) Every number has exactly one successor.
- (c) No two numbers share a successor.
- (d) *Zero is not the successor of any number.*

Pseudo-Java for the Naturals

- We can imagine pseudo-Java methods to test whether a natural is zero and to return the successor of a natural.
- The fourth and fifth axioms imply that every nonzero natural is the successor of another natural, which we will call its **predecessor**.
- We'll assume that these methods are primitives of our language.

Pseudo-Java for the Naturals

```
boolean isZero (natural x)
// Returns true if and only if x is
// zero

natural successor (natural x)
// Returns the successor of x

natural pred (natural x)
// Returns the predecessor of x, if x
// is not zero
// Throws an exception if x is zero

pred(successor(x)) == x
if !isZero(x), successor(pred(x)) == x
```

Forms of the Fifth Peano Axiom

- There are many equivalent ways to express the fifth axiom:
- Version 1: There aren't any naturals other than those forced to exist by the first four axioms.
- Version 2: If you keep taking predecessors of a natural, you will eventually reach zero.

More Forms of the Fifth Axiom

- Version 3: If S is a set of naturals, 0 is in S, and successor(x) is in S whenever x is in S, then S is the set of all naturals.
- Version 4: If P is a unary predicate on naturals, P(0) is true, and $\forall x$: P(x) → P(successor(x)) is true, then $\forall x$: P(x) is true.
- Version 5: Any non-empty set of naturals contains a least element.

About The Forms of the Axiom

- Version 4 is the **Law of Mathematical Induction**, which will become our primary tool for proving things about naturals.
- Version 4 is pretty clearly equivalent to Version 3, because you can replace the set S in Version 3 with the set {n: P(n)} in Version 4, and replace P(n) in version for with the predicate "n $\in$ S".

The Least Number Principle

- Version 5 is the **Least Number Principle** that we used in Discussion #1.
- Here's a proof of Version 4 using Version 5. Given a predicate P satisfying P(0) and $\forall x: P(x) \rightarrow P(x+1)$, let Z be the set $\{n: \neg P(n)\}$. If $Z = \emptyset$, then $\forall x: P(x)$ is true.
- If $Z \neq \emptyset$, by Version 5 it has a least element x. This element can't be 0 because P(0) is true. But if x has a predecessor y, y must also be in Z because if P(y) were true, P(x) would be as well.

Clicker Question #2

- Suppose I want to use Version 5, the Least Number Principle, to prove Version 2, "If you keep taking predecessors of a natural, you will eventually reach zero." To what set should I apply the Least Number Principle?
- (a) The set of all naturals
- (b) The set of naturals that cannot be brought to zero by repeatedly taking predecessors
- (c) The empty set
- (d) The set of positive integers

Answer #2

- Suppose I want to use Version 5, the Least Number Principle, to prove Version 2, "If you keep taking predecessors of a natural, you will eventually reach zero." To what set should I apply the Least Number Principle?
- (a) The set of all naturals
- (b) *The set of naturals that cannot be brought to zero by repeatedly taking predecessors*
- (c) The empty set
- (d) The set of positive integers

Recursion and the Fifth Axiom

- Version 2 says that repeatedly taking predecessors always gets you to 0.
- Here's another form: Suppose that a method takes one argument of type `natural`, that it terminates when called with argument 0, and that when called with any nonzero argument x it terminates, except possibly for a call to itself with argument `pred(x)`. Then the method terminates with any argument.

Recursion and the Fifth Axiom

- This is a common-sense fact about the naturals -- our point is that it is the *same* common-sense fact as the Law of Induction or the Least Number Principle. This form is most useful for proving correctness of a method, and induction is most useful for lots of other purposes.
- Note that the `factor` method from last lecture does not meet the conditions of this statement, since the recursive call does not always have argument pred(x).

Defining Addition

- If we want to define a function that takes a natural as an argument, we can often define it **recursively**.
- For example, we can define x + 0 to be x, and define x + (successor(y)) to be successor(x + y). This definition suggests the recursive method below that adds two naturals, making calls on the pred and successor methods.

```
public natural plus (natural x, natural y) {
   if (isZero(y)) return x;
   return successor (plus (x, pred(y));}
```

Defining Multiplication

- Similarly we can define multiplication by the rules x × 0 = 0 and x × successor(y) = (x × y) + x, which also turns into recursive code.
- We'll be able to prove properties of these operations from these definitions.

```
public natural times (natural x, natural y) {
   if (isZero(y)) return 0;
   return plus (times(x, pred(y)), x);}
```

Other Recursive Systems

- Lots of other data types from computer science can be defined recursively.
- A **stack** is either an empty stack or a stack with an element pushed onto it, and from this we can recursively define the pop and peek operations.
- A "fifth axiom" for stacks might say that repeatedly popping will eventually lead to an empty stack.

Peano Axioms for Strings

- Similarly we have "Peano" axioms for strings:

1. λ is a string.
2. If w is a string and a is a letter, then there is a unique string wa.
3. If va = wb, for strings v and w and letters a and b, then v = w and a = b.
4. Any string w ≠ λ can be written as va, for some string v and letter a.
5. Every string is derived from λ by adding letters according to rule 2.

Peano Axioms for Strings

- We can use this definition to recursively define operations on strings like concatenation and reversal.
- We can then define recursive methods on strings that perform these operations.
- In Discussion #3 we took two properties of the string operations on faith: $(uv)^R = v^R u^R$ and $(w^R)^R = w$. With the string axioms and our definitions, we will be able to *prove* these.

Clicker Question #3

- To define the reversal operation recursively, I need to define λ^R and I need to define $(wa)^R$, where w is a string and a is a letter. What should these two strings be?
- (a) $\lambda^R = \lambda$ and $(wa)^R = aw^R$
- (b) $\lambda^R = \lambda$ and $(wa)^R = w^R a^R$
- (c) $\lambda^R = a$ and $(wa)^R = aw^R$
- (d) $\lambda^R = \lambda$ and $(wa)^R = (w^R)^R$

Answer #3

- To define the reversal operation recursively, I need to define λ^R and I need to define $(wa)^R$, where w is a string and a is a letter. What should these two strings be?
- (a) *$\lambda^R = \lambda$ and $(wa)^R = aw^R$*
- (b) $\lambda^R = \lambda$ and $(wa)^R = w^R a^R$
- (c) $\lambda^R = a$ and $(wa)^R = aw^R$
- (d) $\lambda^R = \lambda$ and $(wa)^R = (w^R)^R$