Mathematical Induction

Victor Adamchik

Fall of 2005

Lecture 3 (out of three)

Plan

1. Recursive Definitions
2. Recursively Defined Sets
3. Program Correctness

Recursive Definitions

Sometimes it is easier to define an object using a self-reference. For example,

A linked list is either empty list or a node followed by a linked list.

A binary tree is either empty tree or a node containing left and right binary trees.

Sometimes self-referencing leads to paradoxes:

The set of all sets which aren't elements of themselves.

The process of defining an object using a self-reference is called recursion. A primitive recursion is a restricted way of defining $f(n+1)$ in terms of $f(n)$. A recursive function is a function that calls itself in order to return an answer. A recursive definition of a function is given in two parts

1. a set of base values (or initial values)
2. a rule for calculating $f(n)$ in terms of previous values

This is a typical example of recursive definition (or inductive definition)

$$f(0) = 5$$
$$f(n+1) = f(n) + 1$$

Here is another example,

$$\text{GCD}(0, b) = b$$
$$\text{GCD}(a, b) = \text{GCD}(b, a), \text{ if } a > b$$
$$\text{GCD}(a, b) = \text{GCD}(b \bmod a, \ a)$$

A palindrome is an expression that reads the same backwards and forwards. For example,

Rats live on no evil star

Madam I'm Adam

Let us define a palindrome over $\{a, \ b, \ c, \ d\}$ alphabet recursively.

Initial values:

$$P_0 = \{\}$$

$$P_1 = \{a, b, c, d\}$$

General rule:

$$P_{n+1} = \{a \lambda a, b \lambda b, c \lambda c, d \lambda d \mid \lambda \in P_{n-1}\}, n \geq 1$$

■ Recursively Defined Sets

We start with an example,

$2 \in S$

if $a \in S \land b \in S \Longrightarrow a + b \in S$

Claim. The above set is a set of positive even integers.

Proof. Let E be a set of ALL positive even integers. We have to prove

$$1.\ E \subset S$$

$$2.\ S \subset E$$

Prove 1). We need to prove that EVERY even positive integer belongs to S. The proof is by induction. Let $P(n) := 2\,n \in S$.

It's easy to see that the basis step holds: $P(1) = 2 \in S$.

Assume that $P(n)$ is true. What can we say about $P(n+1)$?

$$P(n+1) = 2\,(n+1) = 2\,n + 2 \in S$$

since $2\,n \in S$ and $2 \in S$.

Prove 2). We need to prove $S \subset E$, namely that any element in S is divisible by 2. We use the recursive definition of S:

$$\text{if } a \in S \wedge b \in S \Longrightarrow a + b \in S$$

Let us choose any element $x \in S$. By the above rule

$$x = a + b = (a_1 + b_1) + (a_2 + b_2) = \ \ldots$$

We continue splitting until we get

$$x = 2 + 2 + \ldots + 2$$

which means that x is divivible by 2. Hence, $S \subset E$.

Set of Strings

Given an alphabet Σ. We define a set Σ^* of all strings over this alphabet:

1. empty string $\in \Sigma^*$

2. $\lambda\, x \in \Sigma^*$ if $\lambda \in \Sigma^*$ and $x \in \Sigma$

The second rule says that new strings are generated by concatenation. The length of a string $L(\lambda)$ is defined by

1. $L(\text{empty}) = 0$

2. $L(\lambda\, x) = L(\lambda) + 1, \quad x \in \Sigma$

Based on the above two definition we prove

$$L(\lambda_1\, \lambda_2) = L(\lambda_1) + L(\lambda_2), \; \lambda_1 \in \Sigma^*, \lambda_2 \in \Sigma^*$$

Proof (by induction on λ_2)

Basis step: $\lambda_2 = \text{empty}$. By the definition of the length of a string,

$$L(\lambda_1\, \lambda_2) = L(\lambda_1)$$

$$L(\lambda_1) + L(\lambda_2) = L(\lambda_1) + 0$$

Inductive step: we assume that

$$L(\lambda_1\, \lambda_2) = L(\lambda_1) + L(\lambda_2), \; \lambda_1 \in \Sigma^*, \lambda_2 \in \Sigma^*$$

for all $1 \leq L(\lambda_2) \leq n$. We have to prove the above formula for $L(\lambda_2) = n + 1$. Note that by recursive definition,

$$\lambda_2 = \hat{\lambda}\, x, \; \hat{\lambda} \in \Sigma^*, x \in \Sigma$$

Therefore,

$$L(\lambda_1\, \lambda_2) = L\left(\lambda_1\, \hat{\lambda}\, x\right) = L\left(\lambda_1\, \hat{\lambda}\right) + 1 \overset{\text{by} IH}{=} L(\lambda_1) + L(\hat{\lambda}) + 1 = L(\lambda_1) + L\left(\hat{\lambda}\, x\right)$$

which concludes the proof.

■ Program Correctness

How can we be sure that a particular algorithm implementation is correct?

```
int prod = 1;
for(int k=1; k<=n; k++)
    prod *= k;
return prod;
```

The idea is to use a loop invariant - an assertion that is true before and after each execution of the body of the loop.

```
[precondition]
while (guard) { loop-body }
[postcondition]
```

In the above example, a loop invariant is the following proposition

$$P := \text{ prod} = k! \wedge 1 \leq k \leq n$$

A loop invariant should serve two purposes: to state what the loop is supposed to accomplish and to help in proving the algorithm correctness.

To prove that P is a loop invariant we use a mathematical induction. First we note that P is true before the loop is entered, since prod $= 1!$. Next we assume that P is true for $1 \leq k < n$, namely after $n-1$ loop executions. In the next execution, k is incremented by 1 (thus, it becomes n) and prod $*=$ k. Since by inductive hypothesis the previous value of prod is $(k-1)!$, we conclude that prod $= n!$. Therefore, P remains true. Finally, we need to show that the program terminates, which is trivial in our case.

Fast Exponentiation

The following program computes a^n, where n is nonnegative integer, $n \in \mathbb{N}$.

```
int x = a, y = n, z = 1;
while (y > 0)
   if (y%2 == 0) {
      x *= x; y = y/2;}
   else {
      y -= 1; z *= x; }
return z;
```

Note, often the hardest part of a loop invariant proof is identifying the invariant. We introduce the following proposition

$$P := \ z * x^y = a^n \ \wedge \ y \in \mathbb{N}$$

and prove (by induction) that it is a loop invariant.

Basis step. P is true before the loop starts, because $x = a$, $y = n$ and $z = 1$:

$$z * x^y = \ 1 \ * \ a^n$$

Inductive step. We assume that P is true after some iterations. We must show that P remains true after the next pass. Let variables with hats $\hat{x}$, $\hat{y}$, $\hat{z}$ be the values after the loop body was computed. Therefore, we have to prove

$$\hat{z} * \hat{x}^{\hat{y}} = a^n \ \wedge \ y \in \mathbb{N}$$

is true.We consider two cases:

1) y is even. After the execution of the loop body, we have

$$\hat{x} = x^2,\ \hat{y} = y/2,\ \hat{z} = z$$

$$\hat{z} * \hat{x}^{\hat{y}} = \ z \ * \ (x^2)^{y/2} = z * x^y \overset{\text{by } IH}{=} \ a^n$$

1) y is odd. After the execution of the loop body, we have

$$\hat{x} = x,\ \hat{y} = y - 1,\ \hat{z} = z * x$$

$$\hat{z} * \hat{x}^{\hat{y}} = z * x * x^{y-1} = z * x^{y} \overset{\text{by } IH}{=} a^{n}$$

Note, in this case we decrement y by one. Is it true that $\hat{y} \geq 0$? Yes, it is, because the loop condition is $y > 0$.

Finally, we must prove that the above program terminates. It follows from the fact that the loop invariant is true when the loop terminates and the loop condition is false

$$z * x^{y} = a^{n} \wedge y \in \mathbb{N} \wedge y \leq 0$$

This means that $y = 0$ and $z * x^{0} = z = a^{n}$. So, we prove that the algorithm terminates and returns $z = a^{n}$.

Fibonacci Numbers

The following program computes n-th Fibonacci number, $n \in \mathbb{N}$.

```
int prev = 1, cur = 1;
if (n==0 || n ==1) return n;
if (n ==2) return 1;
for(int k=3; k<= n; k++) {
   int tmp = cur;
   cur += prev;
   prev = tmp; }
return cur;
```

We introduce the following proposition

$$P := \text{cur} = F_{k} \wedge \text{prev} = F_{k-1} \wedge k \geq 2$$

Exercise. Prove (by induction) that it is a loop invariant.