CSCE 222

Discrete Structures for Computing

Inductive Sets and Recursive Functions

Dr. Hyunyoung Lee

Based on slides by Andreas Klappenecker

Inductively Defined Sets

Motivating Example

Consider the set

A = {3,5,7,...}

There is a certain ambiguity about this “definition” of the set A.

Likely, A is the set of odd integers >= 3.

[However, A could be the set of odd primes...]

Motivating Example

In Computer Science, we prefer to avoid such ambiguities. You will often encounter sets that are inductively defined.

We can specify the set as follows:

$3 \in A$ and if n is in A, then n+2 is in A.

In this definition, there is

(a) an initial element in A, namely 3.

(b) you construct additional elements by adding 2 to an element in A,

(c) nothing else belongs to A.

We will call this an inductive definition of A.

Inductively Defined Sets

An inductive definition of a set S has the following form:

(a) Basis: Specify one or more "initial" elements of S.

(b) Induction: Give one or more rules for constructing "new" elements of S from "old" elements of S.

(c) Closure: The set S consists of exactly the elements that can be obtained by starting with the initial elements of S and applying the rules for constructing new elements of S.

The closure condition is usually omitted, since it is always assumed in inductive definitions.

Example 1: Natural Numbers

Let S be the set defined as follows:

Basis: $0 \in S$

Induction: If $n \in S$, then $n+1 \in S$

Then S is the set of natural numbers (with 0).

Closure? Implied!

Example 2

Let S be the set defined as follows:

Basis: $0 \in S$

Induction: If $n \in S$, then $2n+1 \in S$.

Can you describe the set S?

$S = \{0,1,3,7,15,31,\ldots\} = \{ 2^n-1 \mid n \text{ is a nonnegative integer} \}$

since $2^0-1=0$ and $2^{n+1}-1 = 2(2^n-1)+1$

Example 3: Well-Formed Formulas

We can define the set of well-formed formulas consisting of variables, numerals, and operators from the set {+,-,*,/} as follows:

Basis: x is a well-formed formula if x is a numeral or a variable.

Induction: If F and G are well-formed formulas, then

(F+G), (F-G), (F*G), and (F/G)

are well-formed formulas.

Examples: 42, x, (x+42), (x-y), (3/0), (x*(y+z))

Example 4: Lists

We can define the set L of finite lists of integers as follows.

Basis: The empty list () is contained in L

Induction: If i is an integer, and l is a list in L, then (cons i l) is in L.

[Note: This is the Lisp style of lists, where (cons i l) prepends the data item i at the front of the list l]

Example: (cons 1 (cons 2 (cons 3 ()))) is the list (1 2 3) in Lisp.

Example 5: Binary Trees

We can define the set B of binary trees over an alphabet A as follows:

Basis: $\langle\rangle \in B$.

Induction: If $L, R \in B$ and $x \in A$, then $\langle L,x,R\rangle \in B$.

Example: $\langle\langle\rangle,1,\langle\rangle\rangle$ // tree with one node (1)

Example: $\langle\ \langle\langle\rangle,1,\langle\rangle\rangle,\ r,\ \langle\langle\rangle,2,\langle\rangle\rangle\ \rangle$

// tree with root r and two children (1 and 2).

Applications of Inductively Defined Sets

In Computer Science, we typically use inductively defined sets (a.k.a. recursively defined sets) when defining:

- programming languages (via grammars)

- logic (via well-formed logical formulas)

- data structures (binary trees, rooted trees, lists).

- fractals

We also use them in connection with functional programming languages.

Extremely popular in Computer Science!

Recursively Defined Functions

Recursively Defined Functions

Suppose we have a function with the set of nonnegative integers as its domain.

We can specify the function as follows:

Basis step: Specify the value of the function at 0

Inductive step: Give a rule for finding its value at an integer from its values at smaller integers.

This is called a recursive or inductive definition.

Example 1: Factorial Function

We can define the factorial function n! as follows:

Base step: 0! = 1

Inductive step: n! = n (n-1)!

Example 2: Fibonacci Numbers

The Fibonacci numbers f_n are defined as follows:

Base step: $f_0=0$ and $f_1=1$

Inductive step: $f_n = f_{n-1} + f_{n-2}$ for $n >= 2$

We can use the recursive definition of the Fibonacci numbers to prove many properties of these numbers. The recursive structure actually helps to formulate the proofs.

Recursively Defined Functions

One can define recursively defined functions for domains other than the nonnegative integers.

In general, a function f is called recursively defined if and only if at least one value f(x) is defined in terms of another value f(y), where x and y are distinct elements.

[However, we will typically consider recursively defined functions that have more structure than this definition suggests.]

Example 3: Ackermann Function

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0, \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0, \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

The Ackermann function has particular significance in computability theory. The values of A(m,n) grow very, very quickly.

Ackermann Function (Cont.)

```
# In Ruby, the Ackermann function can be defined as follows:
def A(m,n)
  return n+1 if m==0
  return A(m-1,1) if n==0
  return A(m-1,A(m,n-1))
end
# Now try calculating A(0,0), A(1,1), A(2,2), A(3,3), A(4,4)
```

Ackermann Function Example

A(0,0) => 1

A(1,1) => A(0,A(1,0)) => A(0,A(0,1)) => A(0, 2) => 3

A(2,2) => A(1,A(2,1)) => A(1,A(1,A(2,0))) => A(1,A(1,A(1,1))) =>
A(1,A(1,A(0,A(1,0)))) => A(1,A(1,A(0,A(0,1)))) => A(1,A(1,A(0,2)))
=> A(1,A(1,3)) => A(1,A(0,A(1,2))) => A(1,A(0,A(0,A(1,1)))) => ... =>
A(1,A(0,A(0,3))) => A(1,A(0,4)) => A(1,5) => A(0,A(1,4)) =>
A(0,A(0,A(1,3))) => A(0,A(0,A(0,A(1,2)))) => A(0,A(0,A(0,4))) =>
A(0,A(0,5)) => A(0,6) => 7

Ackermann Function (Cont.)

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0, \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0, \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

Why does the recursion terminate?

Lexicographically order the pairs (m,n).

So (m,n) < (m',n') iff m<m' or (m=m' and n<n').

Note that arguments used in the RHS are lexicographically smaller than the arguments in the LHS. Thus, eventually we need to end up in case m=0, though it might take an extremely long time.