Section 3.2: Recursively Defined Functions and Procedures

Function: Has **inputs** ("arguments", "operands") and **output** ("result")
No "side effects".
Procedure: May have **side effects**, e.g., "print(…)"

A recursive function (or procedure) calls itself!

A function f is **recursively defined** if at least one value of f(x) is defined in terms of another value, f(y), where $x \neq y$.

Similarly: a procedure P is **recursively defined** if the action of P(x) is defined in terms of another action, P(y), where $x \neq y$.

When an argument to a function is inductively defined, here is a technique for creating a recursive function definition:

1. Specify a value of f(x) for each basis element x in S.
2. For each inductive rule that defines an element x in S in terms of some element y already in S, specify rules in the function that compute f(x) in terms of f(y).

Example: Find a recursive definition for function $f:\mathbb{N}\rightarrow\mathbb{N}$ defined by

$f(n) = 0 + 3 + 6 + \ldots + 3n$.

e.g., $f(0) = 0$

$f(1) = 0 + 3$

$f(2) = 0 + 3 + 6$

Solution: Notice that $\mathbb{N}$ is an inductively defined set:

$0\in\mathbb{N}$; $n\in\mathbb{N}$ implies $n+1\in\mathbb{N}$

So we need to give f(0) a value and

we need to deinfe f(n+1) in terms of f(n).

The value for f(0) should be 0. What about f(n+1)?

$f(n+1) = 0 + 3 + 6 + \ldots 3n + 3(n+1)$

$= f(n) + 3(n+1)$

So here is our (recursive) definition for f:

$f(0) = 0$

$f(n+1) = f(n)+3(n+1)$

We could also write:

$f(0) = 0$

$f(n) = f(n-1)+3n$ for $n>0$

Here is a more programming-like definition:

f(n) = (**if** n=0 **then** 0 **else** f(n-1)+3n **endIf**)

Example: Find a recursive definition for

$cat: A^* \times A^* \rightarrow A^*$

defined by $cat(s,t) = st$

Solution: Notice that A^* is inductively defined.

Basis: $\Lambda \in A^*$; Induction: $a \in A$ and $x \in A^*$ imply $ax \in A^*$

We can define cat recursively using the first argument.

The definition of cat gives

$cat(\Lambda,t) = \Lambda t = t.$

For the recursive part we can write

$cat(ax,t) = axt = a(xt) = a\,cat(x,t)$

Here is a definition:

$cat(\Lambda,t) = t$

$cat(ax,t) = a\,cat(x,t)$

Here is the if-then-else form:

$cat(s,t) =$ **<u>if</u>** $s=\Lambda$ **<u>then</u>** t **<u>else</u>** $head(s)cat(tail(s),t)$

Example: Find a definition of $f: lists(\mathbb{Q}) \rightarrow \mathbb{Q}$ defined by

$$f(<x_1, \ldots, x_n>) = x_1 + \ldots + x_n$$

Solution: Notice that the set $lists(\mathbb{Q})$ is defined rescursively.

Basis: $<> \in lists(\mathbb{Q})$

Induction: $h \in \mathbb{Q}$ and $t \in lists(\mathbb{Q})$ imply $h::t \in lists(\mathbb{Q})$

To discover a recursive definition, we can use the definition of f as follows:

$$\begin{aligned} &f(<x_1, \ldots, x_n>) \\ &= x_1 + x_2 \ldots + x_n \\ &= x_1 + (x_2 + \ldots + x_n) \\ &= x_1 + f(<x_2, \ldots ,x_n>) \\ &= head(<x_1, \ldots, x_n>) + f(tail(<x_1, \ldots, x_n>)) \end{aligned}$$

So, here is our recursive definition:

$f(<>) = 0$

$f(h::t) = h + f(t)$

Expressing this in the if-then-else form:

f(L) = <u>**if**</u> L=<> <u>**then**</u> 0 <u>**else**</u> head(L)+f(tail(L))

Example: Given f:$\mathbb{N} \rightarrow \mathbb{N}$ as defined by

f(0) = 0
f(1) = 0
f(x+2) = 1+f(x)

Here is the if-then-else formulation:

f(x) = **if** (x=0 or x=1) **then** 0 **else** 1 + f(x-2)

What exactly does this function do?

Let's try to get an idea by enumerating a few values.

map(f,<0,1,2,3,4,5,6,7,8,9>) = <0,0,1,1,2,2,3,3,4,4>

So f(x) returns the floor of x/2. That is, f(x) = $\lfloor x/2 \rfloor$.

Example: Find a recursive definition for the function $f: \text{lists}(\mathbb{Q}) \to \mathbb{Q}$ as defined by:

$$f(<x_1, \ldots, x_n>) = x_1x_2 + x_2x_3 + \ldots + x_{n-1}x_n$$

Approach:

Let $f(<>) = 0$ and $f(<x>) = 0$. Then for $n \geq 2$ we can write:

$$f(<x_1, \ldots, x_n>)$$
$$= x_1x_2 + x_2x_3 + \ldots + x_{n-1}x_n$$
$$= x_1x_2 + (x_2x_3 + \ldots + x_{n-1}x_n)$$
$$= x_1x_2 + f(<x_2, \ldots, x_n>)$$

So here is our recursive definition:

$$f(<>) = 0$$
$$f(<x>) = 0$$
$$f(h::t) = h \cdot \text{head}(t) + f(t).$$

We can express this in if-then-else form as:

```
f(L) =  if  (L=<> or tail(L)=<>)
        then
            0
        else
            head(L)·head(tail(L)) + f(tail(L))
        endIf
```

Example: Find a recursive definition for the function

isin : $A \times lists(A) \rightarrow \{true, false\}$

where isin(x,L) means that x occurs in the list L.

Solution:

isin(x,<>) = false

isin(x,x::t) = true

isin(x,y::t) = isin(x,t), where $x \neq y$

Here's the if-then-else form:

```
isin(x,L) =  if L=<>
             then
                  false
             else
                  if x=head(L)
                  then
                       true
                  else
                       isin(x,tail(L))
                  endIf
             endIf
```

Example: Find a recursive definition for the function

isin : A × lists(A) → {true,false}

where isin(x,L) means that x occurs in the list L.

Solution:

isin(x,<>) = false
isin(x,x::t) = true
isin(x,y::t) = isin(x,t), where $x \neq y$

Here's the if-then-else form:

```
isin(x,L) =  if L=<>
             then
                 false
             else
                 x=head(L) or isin(x,tail(L))
             endIf
```

Example: Find a recursive definition for

sub: lists(A) × lists(A) → {true,false}

where sub(L,M) means the elements of L are elements of M.

Solution:

From Previous Slide

Here is a pattern-matching solution:

```
sub(<>,M) = true
sub(h::t,M) = if isin(h,M) then sub(t,M) else false
```

Here is a programmatic (executable) version:

```
sub(L,M) = if L=<>
           then
               true
           else
               if isin(head(L),M)
               then
                   sub(tail(L),M)
               else
                   false
               endIf
           endIf
```

Example: Find a recursive definition for

intree: $\mathbb{Q} \times \text{binSearchTrees}(\mathbb{Q}) \rightarrow \{\text{true},\text{false}\}$

where intree(x,T) means x is in the binary search tree T.

Solution:

intree(x,<>) = false
intree(x,<L,x,R>) = true
intree(x,<L,y,R>) = **if** x<y **then** intree(x,L) **else** intree(x,R)

Why is this a better definition?

intree(x,<>) = false

$$\text{intree}(x,\langle L,y,R\rangle) = \begin{cases} \text{true, if } x=y \\ \text{intree}(x,L)\text{, if } x<y \\ \text{intree}(x,R)\text{, if } x>y \end{cases}$$

Here is the if-then-else form:

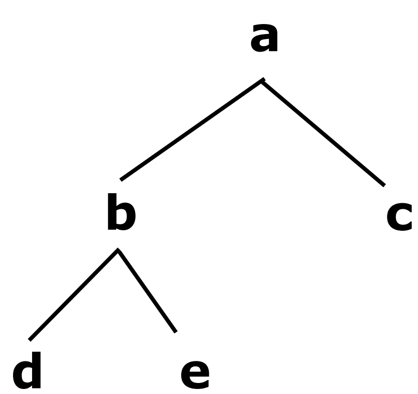
intree(x,T) = **if** T=<> **then** false
elseIf x=root(T) **then** true
elseIf x<root(T) **then** intree(x,left(T))
else intree(x,right(T))
endIf

Traversing Binary Trees

There are 3 ways to traverse a binary tree.
Each is defined recursively.

preorder(T): if T≠<> then
visit root; preorder(left(T)); preorder(right(T))
inorder(T): if T≠<> then
inorder(left(T)); visit root; inorder(right(T))
postorder(T): if T≠<> then
postorder(left(T)); postorder(right(T); visit root

Example: Traverse this tree in each of the orders:



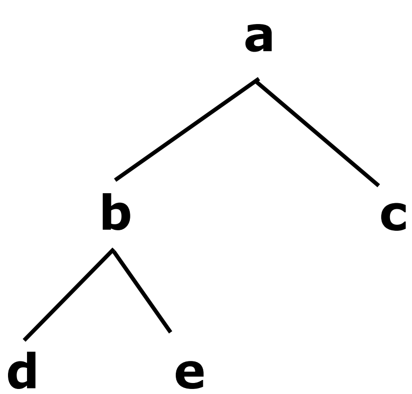
Solution:
pre-order:
in-order:
post-order:

Traversing Binary Trees

There are 3 ways to traverse a binary tree.
Each is defined recursively.

preorder(T): if T≠<> then
visit root; preorder(left(T)); preorder(right(T))
inorder(T): if T≠<> then
inorder(left(T)); visit root; inorder(right(T))
postorder(T): if T≠<> then
postorder(left(T)); postorder(right(T); visit root

Example: Traverse this tree in each of the orders:



The parentheses are not part of the answer, but adding them makes things clearer.

Solution:

pre-order: **a (b d e) (c)**
in-order: **(d b e) a (c)**
post-order: **(d e b) (c) a**

Example: Find a recursive definition for

post: binaryTrees(A) → lists(A)

where post(T) is the list of nodes from a pot-order traversal of T.

Solution:

post(<>) = <>
post(<L,x,R>) = cat(post(L),cat(post(R),<x>)

The function cat will concatenate two lists, and can be defined as:

cat(<>,L) = L
cat(h::t,L) = h::cat(t,L)

Example: Find a recursive definition for

sumnodes: binaryTrees($\mathbb{Q}$) → $\mathbb{Q}$

where sumnodes(T) returns the sum of the nodes in T.

Solution:

sumnodes(<>) = 0
sumnodes(<L,x,R>) = x + sumnodes(L) + sumnodes(R)

Infinite Sequences

We can construct recursive definitions for infinite sequences by defining a value f(x) in terms of x and f(y) for some value y in the sequence.

Example: Suppose we want to define a function f that returns an infinite sequence. The function f should return this sequence:

$f(x) = <x^1, x^2, x^4, x^8, x^{16}, \ldots >$

Approach:

Look at the definition and try to find a solution:

$f(x)$

$= <x^1, x^2, x^4, x^8, x^{16}, \ldots >$

$= x :: <x^2, x^4, x^8, x^{16}, \ldots >$

$= x :: f(x^2)$

So we can define:

$f(x) = x :: f(x^2)$

This function returns an infinite sequence.

Q: Of what use is such a function in computing???

A: We can use *"lazy evaluation"*: When we need an element from f(x), we'll need to evaluate f. Yes, this is an infinite computation, but we'll do only as much work as necessary to get the element we need.

Example: What sequence is defined by $g(x,k)= x^k :: g(x,k+1)$?

Solution: $g(x,k) = x^k :: g(x,k+1)$
$= x^k :: x^{k+1} :: g(x,k+2)$
$= <x^k, x^{k+1}, x^{k+2}, \ldots>$

Example: How do we obtain the sequence $<x, x^3, x^5, x^7, \ldots>$?

Solution: Define $f(x) = h(x,1)$
where $h(x,k) = x^k :: h(x,k+2)$

Example: How do we obtain the sequence $<1, x^2, x^4, x^6, x^8, \ldots>$?

Solution: Define $f(x) = h(x,0)$, where h is from the previous example.
