COL100: Introduction to Computer Science

2.2: Recursion

Motivation

$$sumOfSquares(x, y) = x^2 + y^2$$

$$var(a, b, c) = ((a - m)^2 + \cdots)/3$$
$$\text{where } m = (a + b + c)/3$$

$$\max(a, b) = \begin{cases} a & \text{if } a > b, \\ b & \text{otherwise} \end{cases}$$

$$\begin{array}{ccccc} & & 3 & 8 & 4 \\ \times & & & 5 & 6 \\ \hline & 2 & 3 & 0 & 4 \\ 1 & 9 & 2 & 0 & \\ \hline 2 & 1 & 5 & 0 & 4 \end{array}$$

A toy problem

Suppose your computational model only provides +, −, × on $\mathbb{N}$.
Can you compute the remainder of two numbers?

Definition. Given two natural numbers $n \geq 0$ and $d > 0$, there is a unique natural number q such that $qd \leq n < (q + 1)d$. Then the remainder is $r = n - qd$.

$$rem(n, r) = n - qd$$
$$\text{where } q \in \mathbb{N} \text{ is such that}$$
$$qd \leq n < (q + 1)d$$

Not an algorithm!

$$rem(n, d) = \begin{cases} n & \text{if } n < d, \\ rem(n - d, d) & \text{otherwise} \end{cases}$$

For any $n \geq 0$ and $d > 0$, we know how to evaluate this, e.g.

$$\begin{aligned} & rem(10, 3) \\ = \; & rem(7, 3) \\ = \; & rem(4, 3) \\ = \; & rem(1, 3) \\ = \; & 1 \end{aligned}$$

SML implementation is straightforward:

```
fun rem(n, d) =
  if n < d then n
  else rem(n - d, d)
```

$$rem(n, d) = \begin{cases} n & \text{if } n < d, \\ rem(n - d, d) & \text{otherwise} \end{cases}$$

We have defined the function in terms of itself! This is called *recursion*.

Evaluating *rem*(*n*, *d*) results in a computational process which might take a large number of steps

1. Will it always stop, for every input?
2. When it stops, will it give the right answer?

For *rem*, not so hard: after each step, *n* decreases by a nonzero amount *d*, so eventually it will reach the first case and terminate.

The factorial function

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n$$

The factorial function

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n$$

Try to break into simpler problems… such as the same problem on smaller input

$$(n-1)! = 1 \times 2 \times \cdots \times (n-1)$$

$$n! = (n-1)! \times n$$

A factorial algorithm?

$$factorial : \mathbb{N} \to \mathbb{N}$$

$$factorial(n) = factorial(n-1) \times n$$

A factorial algorithm

$$factorial : \mathbb{N} \to \mathbb{N}$$

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

What is *factorial*(3)?

$$\begin{gathered} factorial(3) \\ = factorial(2) \times 3 \\ = (factorial(1) \times 2) \times 3 \\ = ((factorial(0) \times 1) \times 2) \times 3 \\ = ((1 \times 1) \times 2) \times 3 \\ = (1 \times 2) \times 3 \\ = 2 \times 3 \\ = 6 \end{gathered}$$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$factorial(n) \mid n = 3$

$factorial(n-1) \times n$
$= factorial(2) \times 3$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$factorial(n) \mid n = 3$ $factorial(n-1) \times n$ $= factorial(2) \times 3$	$factorial(n) \mid n = 2$ $factorial(n-1) \times n$ $= factorial(1) \times 2$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$factorial(n) \mid n = 3$ $factorial(n-1) \times n$ $= factorial(2) \times 3$	$factorial(n) \mid n = 2$ $factorial(n-1) \times n$ $= factorial(1) \times 2$	$factorial(n) \mid n = 1$ $factorial(n-1) \times n$ $= factorial(0) \times 1$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$factorial(n) \mid n = 3$	$factorial(n) \mid n = 2$	$factorial(n) \mid n = 1$	$factorial(n) \mid n = 0$
$factorial(n-1) \times n$ $= factorial(2) \times 3$	$factorial(n-1) \times n$ $= factorial(1) \times 2$	$factorial(n-1) \times n$ $= factorial(0) \times 1$	1

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$factorial(n) \mid n = 3$	$factorial(n) \mid n = 2$	$factorial(n) \mid n = 1$
$factorial(n-1) \times n$ $= factorial(2) \times 3$	$factorial(n-1) \times n$ $= factorial(1) \times 2$	$factorial(n-1) \times n$ $= factorial(0) \times 1$ $= 1 \times 1$ $= 1$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$factorial(n) \mid n = 3$

$factorial(n-1) \times n$
$= factorial(2) \times 3$

$factorial(n) \mid n = 2$

$factorial(n-1) \times n$
$= factorial(1) \times 2$
$= 1 \times 2$
$= 2$

Evaluating factorial

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$$factorial(n) \mid n = 3$$

$$\begin{aligned} & factorial(n-1) \times n \\ &= factorial(2) \times 3 \\ &= 2 \times 3 \\ &= 6 \end{aligned}$$

Summary

In mathematics, a *function* is just any association of inputs to unique outputs

To define an *algorithm*, the specification of the function must:

1. define a precise computational procedure
2. that always terminates
3. in desired value

Afterwards

- Read the notes, Sec. 3.4 and 3.5.
- Give a recursive definition of a function *power*(x, n) which, given any natural numbers x and n, computes x^n. Implement and test it in SML.
- Define an SML function `rightAlign : string * int -> string` which, given a string `s` and a natural number `n` such that `size(s) <= n`, computes a string of size exactly `n` containing some spaces followed by `s`. For example:

```
rightAlign("hi", 8)       = "      hi"
rightAlign("COL100", 8)   = "  COL100"
rightAlign("12345678", 8) = "12345678"
```