COL100: Introduction to Computer Science

4.2: More examples of recursive algorithms

Summing a series

Suppose you want to compute, say,

$$f(a, b) = \sum_{i=a}^{b} i.$$

Pretend you don't know a closed-form formula for this.
How will you define an algorithm?

Break off a small piece of the problem.

$$\sum_{i=a}^{b} i = \left(\sum_{i=a}^{b-1} i\right) + b$$

Know when to stop.

$$\sum_{i=a}^{b} i = 0 \quad \text{if } b < a$$

So,

$$f(a, b) = \begin{cases} 0 & \text{if } b < a, \\ f(a, b-1) + b & \text{otherwise.} \end{cases}$$

What about this summation?

$$g(a, b) = \sum_{i=a}^{b} i^2$$

$$g(a, b) = \begin{cases} 0 & \text{if } b < a, \\ g(a, b-1) + b^2 & \text{otherwise.} \end{cases}$$

What about this summation?

$$h(a, b) = \sum_{i=a}^{b} \frac{1}{i}$$

$$h(a, b) = \begin{cases} 0 & \text{if } b < a, \\ h(a, b-1) + 1/b & \text{otherwise.} \end{cases}$$

Abstract over the function as well!

$$sum(f, a, b) = \begin{cases} 0 & \text{if } b < a, \\ sum(f, a, b-1) + f(b) & \text{otherwise.} \end{cases}$$

In SML,

```
fun sum(f, a, b) =
  if b < a
  then 0
  else sum(f, a, b - 1) + f(b);
```

Note the type of this *higher-order function*:

```
sum : (int -> int) * int * int -> int
```

```
fun id(n) = n;
sum(id, 1, 10);

fun square(n) = n * n;
sum(square, 1, 10);

sum(fn n => n * n, 1, 10);
```

← using an *anonymous function* for squaring

```
fun realSum(f, a, b) =
  if b < a then 0.0
  else sum(f, a, b - 1) + f(b);

fun reciprocal(n) = 1.0/real(n);
realSum(reciprocal, 1, 10);
```

Variations

Check whether all the numbers between *a* and *b* satisfy property *P*.

$$all(a, b, P) = \begin{cases} \text{true} & \text{if } b < a, \\ all(a, b-1, P) \wedge P(b) & \text{otherwise.} \end{cases}$$

Not the best algorithm! Can terminate immediately if you find a counterexample:

$$all(a, b, P) = \begin{cases} \text{true} & \text{if } b < a, \\ \text{false} & \text{if } \neg P(b), \\ all(a, b-1, P) & \text{otherwise.} \end{cases}$$

Find the *smallest* number between *a* and *b* that satisfies property *P*.

$$
\mathit{findSmallest}(a, b, P) = \begin{cases} ? & \text{if } a > b, \\ a & \text{if } P(a), \\ \mathit{findSmallest}(a+1, b, P) & \text{otherwise.} \end{cases}
$$

Specification is incomplete…

Find the smallest number between *a* and *b* that satisfies property *P*, or return *b* + 1 if no such number exists.

Integer square root

Problem: Given a natural number n, find $\lfloor \sqrt{n} \rfloor$ using only integer arithmetic.

$$\lfloor \sqrt{n} \rfloor = k \text{ if and only if } k \geq 0,\ k^2 \leq n < (k+1)^2.$$

How to compute this k?

Algorithm 1

k is the smallest number such that $(k + 1)^2 > n$.

$$intSqrt1(n) = findSmallest(P, 0, n)$$

where $P(i)$ is the property that $(i + 1)^2 > n$.

Algorithm 2

Suppose $m = \lfloor n/4 \rfloor$, $i = \lfloor \sqrt{m} \rfloor$.
What can you say about $k = \lfloor \sqrt{n} \rfloor$ using i?

$$4m \leq n < 4(m+1)$$

$$i^2 \leq m < (i+1)^2$$
$$\Rightarrow m + 1 \leq (i+1)^2$$

So

$$(2i)^2 \leq 4m \leq n,$$
$$n < 4(m+1) \leq (2i+2)^2$$

$$\Rightarrow (2i)^2 \leq n < (2i+2)^2$$

$$(2i)^2 \leq n < (2i + 2)^2$$

So $\lfloor \sqrt{n} \rfloor$ is either $2i$ or $2i + 1$.

$$intSqrt2(n) = \begin{cases} 0 & \text{if } n = 0, \\ 2i & \text{if } (2i+1)^2 > n, \\ 2i+1 & \text{otherwise,} \end{cases}$$

where $i = intSqrt2(\lfloor n/4 \rfloor)$.

Afterwards

- Prove the correctness of all the functions discussed here.
- Prove that *intSqrt1* takes about $\lfloor \sqrt{n} \rfloor$ recursive function calls to find the solution, while *intSqrt2* takes only about $\log_4 n$ recursive function calls.
- Design recursive functions *any* and *none*, analogous to the *all* function, which check if any or none (respectively) of the integers in a given range have the given property. Implement and test them in SML.
    - Can *any* and *none* be implemented in terms of *all*, without writing a new recursion?