Class: CSc 335

Date: Mar 9, 2023

Pascal Triangle

The $ (r, c) $ entry should be $ \binom{r}{c} $, which would require $ c \leq r $ and both $ r, c$ are non-negative integers.

From the table

$ \binom{5}{3} = \binom{4}{2} + \binom{4}{3} $

and similarly across the table,

$ \binom{r}{c} = \binom{r - 1}{c - 1} + \binom{r - 1}{c} $

Suggesting for the recursion.

```
(pas r c) = (+ (pas (- r 1)(- c 1))
               (pas (- r 1) c))
```

where the induction is on $ r $.

Does Zero-based indexing deliver this?

- Yes, by checking a few entries.

Divide & Conquer alone of course is not enough - we need the stopping condition(s) as well.

Observe:

- $ c = 0 => \binom{r}{c} = 1 $
- $ c = r => \binom{r}{c} = 1 $

Pre-conditions:

- $ c \leq r $ & both are non-negative integers

```
(define (pas r c)
  (cond ((zero? c) 1)
        ((= r c) 1)
        (else (+ (pas (- r 1) (- c 1))
                 (pas (- r 1) c)))
        ))
```

- still need to check the termination, and that the pre-condition holds ahead of each recursive call
- A termination condition is an argument where the stopping condition has been reached.

Higher-Order Procedures

1. Passing functions as a parameters
2. Return functions as values from procedure calls

```
(define (sigma-v0 a b)
  (cond ((> a b) 0)
        (else (+ a (sigma-v0 (+ a 1) b)))
        ))
```

Supposed instead of $ \sum_{i=a}^{b} i $, we want to compute $ \sum_{i=a}^{b} i^2 or \sum_{i=a}^{b} i^3 $, or generally $ \sum_{i=a}^{b} (term \ i) $, where $ term = N^{\geq 0} => R $

- $ N^{\geq 0} => R $ needs to be added in the pre-condition.

To avoid rewriting what is essentially the same code, we **ABSTRACT** the sigma function by introducing a new parameter - call it term.

```
(define (sigma-v1 a term b)
  (cond ((> a b) 0)
        (else (+ (term a) (sigma-v1 (+ a 1) term b)))
        ))
```

- `term` is a function - as it is a parameter, we need to mention it in the pre-condition.
- Termination argument
  - and also term(a) returns a value for the N → R

We can abstract again

- Everytime we abstract the code, we have to worry about the termination argument.

```
(define (sigma-v2 a term next b)
  (cond ((> a b) 0)
        (else (+ (term a) (sigma-v2 (next a) term next b)))
        ))
```

- And again - perphaps we're not always interested in the plus. So we can introduce the parameter `combiner`

```
(define (sigma-v3 a term next combiner init b)
  (cond ((> a b) init)
        (else (combiner (term a) (sigma-v3 (next a) term next combiner
init b)))
        ))
```