CS 135 — L21 Randomness

Luke Lu • 2025-11-25

Randomness

Randomness does not increase computational power.

It can improve efficiency in some extends

True randomness is mostly required for security and cryptography as it is slow hard to determine as it is truely random

Pseudorandomness

Pseudorandomness is sufficient for simulations that generate random characters that appears random

```
;; prng: State -> (list State Nat)
```

Middle-square method

```
;; middle-square-4: Nat -> Nat
(define (middle-square-4 n)
  (quotient (remainder (sqr n) 1000000) 100))

;; random-list: Nat Nat -> (listof Nat)

(define (random-list n seed)
 (reverse
   (foldr (lambda (x y) (cons (middle-square-4 (first y)) y)) (list seed)
 (build-list n (lambda (n) n)))))

;;(random-list 8 519) ==> (list 519 2693 2522 3604 9888 7725 6756 6435 4092)

;; The output is always the same
```

`random`

> 🐋 **Info** — **`random`** `(random n)` generates a pseudorandom nunber from 0 to $n - 1$:

Sochastic simulations

Walking simulations

```
;; Randomly move one block from a current position
;; random-move: (list Int Int) -> (list Int Int)
(define (random-move location)
 (local
   [(define move (random 4))
   (define x (first location))
   (define y (second location))]
   (cond
    [(zero? move) (list x (add1 y))] ; North
    [(= 1 move) (list x (sub1 y))] ; South
    [(= 2 move) (list (add1 x) y)] ; East
    [else (list (sub1 x) y)])))) ; West
```

```
;; Does up to n random moves return us to the origin?
;; random-moves: Nat (list Int Int) -> Bool
(define (random-moves n current)
 (cond
   [(zero? n) false]
   [else
    (local
     [(define new (random-move current))]
     (cond
       [(and (zero? (first new)) (zero? (second new))) true]
       [else (random-moves (sub1 n) new)]))]))


;; Simulate m random walks of up to n moves,
;; producing the number of times we return to the origin
;; simulate: Nat Nat -> Nat
(define (simulate m n)
 (foldr (lambda (x y) (+ y
   (cond
     [(random-moves n (list 0 0)) 1]
     [else 0])))
    0
 (build-list m (lambda (n) n))))


(simulate 10000 20)
```