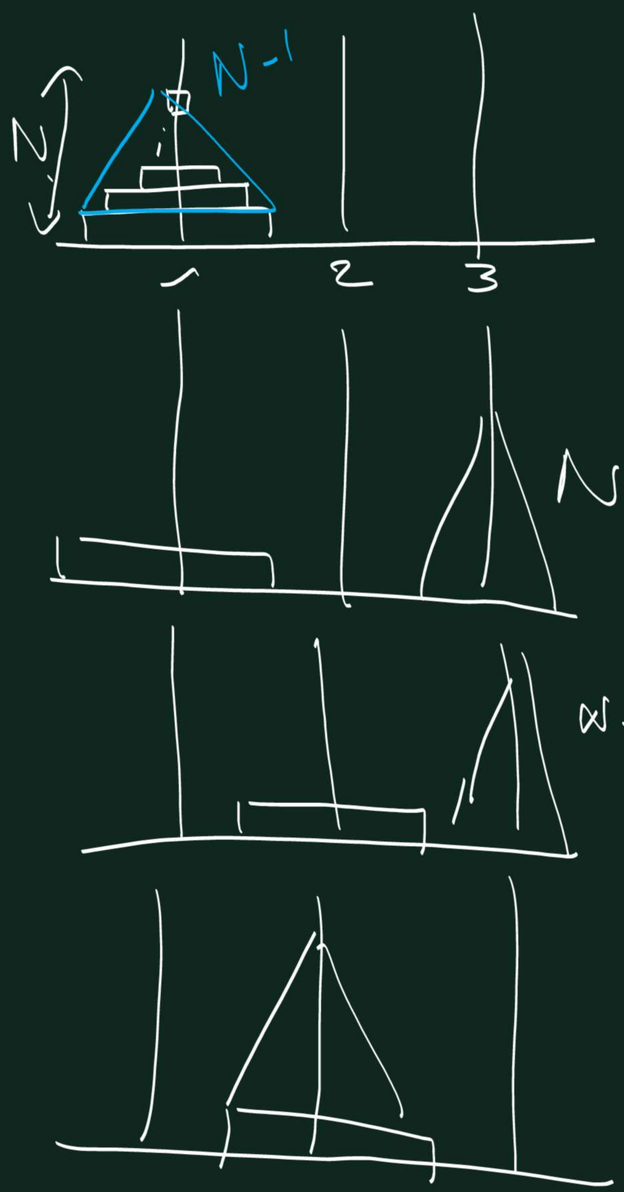


hanoi(3,1,2,3)

hanoi(n, from, to, using) {

if (n > 0) {

hanoi(n-1, from, using, to)

move(from, to)

hanoi(n-1, using, to, from)

param: n
unit op: move

$N - \begin{cases} C(0) = 0 \\ C(n) = 2(C(n-1)) + 1 \end{cases}$

$N-1$

h(3, 1, 2, 3)

h(2, 1, 3, 2)

m(1, 2)

h(2, 3, 2, 1)

h(1, 1, 2, 3)

m(1, 3)

h(1, 2, 3, 1)

h(1, 3, 1, 2)

m(3, 1)

h(1, 1, 3, 3)

$$C(0) = 0$$

$$C(n) = 2\,C(n-1) + 1$$

$$C(n) + 1 = 2\,C(n-1) + 2$$

$$= 2\left(C(n-1) + 1\right)$$

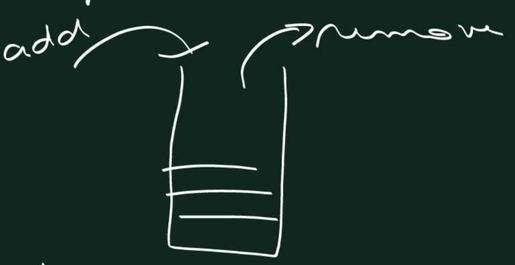
$$D(n) = C(n) + 1$$

$$D(0) = 1$$

$$D(n) = 2\,D(n-1) = 2^n$$

$$C(n) = 2^n - 1 = \Theta(2^n)$$

ADT Stack



- check whether it's empty (isEmpty)
- look for topmost elt (top)
- add an elt on top (push)
- remove top elt (pop)

LIFO

ADT Stack

Use Element, Boolean

Operation

* new	:	⟶ Stack
isEmpty	:	Stack ⟶ Boolean
top	:	Stack ⇀ Element
* push	:	Stack × Element ⟶ Stack
pop	:	Stack ⇀ Stack

precondition

top(s) iff !isEmpty(s)

s != new

pop(s) ——————

Axioms

A_1 : isEmpty(new) ≡ T

A_2 : isEmpty(push(s,e)) ≡ F

A_3 top(push(s,e)) ≡ e

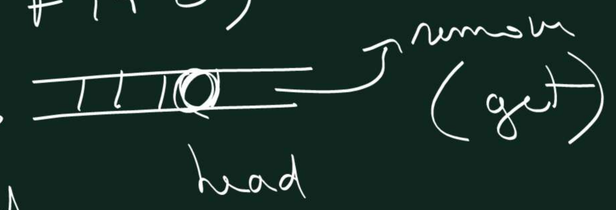
A_4 pop(push(s,e)) ≡ s

top(pop(push(push(new, e_1), e_2)))

A_4 top(push(new, e_1))

A_3 e_1

Queue (FIFO)



ADT Queue

Use Element, Boolean

Operations

* new : ⟶ Queue

isEmpty : Queue ⟶ Boolean

head : Queue ⟶ Element

* put : Queue × Element ⟶ Queue

get : Queue ⟶ Queue

Preconditions

head(q) if q ≠ new

get(q) ——— isEmpty(q) ≡ F

Axioms

A_1 isEmpty(new) ≡ T

A_2 isEmpty(put(q,e)) ≡ F

A_3 head(put(new,e)) ≡ e

A_4 head(put(q,e)) ≡ head(q)

A_5 get(put(new,e)) ≡ new

A_6 get(put*(q,e)) ≡ put*(get(q),e)

ADT List

Use Element

Operations

* newr: → List

first: list → Element

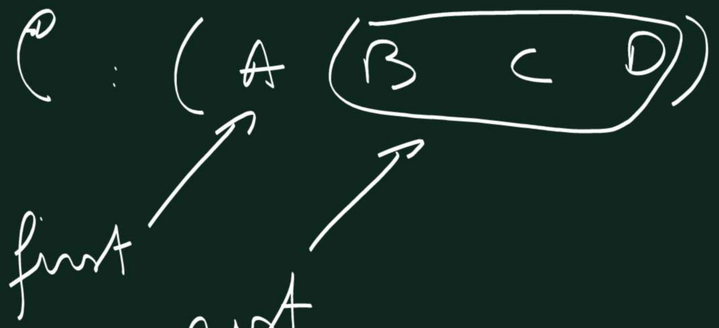
rest: List → List

* cons: Element × List → List

Preconditions

first (l) ## l != newr

rest (l)



cons (A, (B C D))

↳ (A B C D)

Axioms

A1 first (cons (e, l)) ≡ e

A2 rest (cons (e, l)) ≡ l

ADT extension List
Use: Boolean, Natural

Operations

isEmpty : List → Boolean
length : List → Natural
contains : List × Element → Boolean

Axioms

A_1 isEmpty(new) ≡ T
A_2 isEmpty(cons(e, l)) ≡ F
A_3 length(new) ≡ 0
A_4 length(cons(e, l)) ≡ length(l) + 1
A_5 contains(new, e) ≡ F
A_6 contains(cons(e, l), e) ≡ T
A_6^* contains(l, first(l)) ≡ T

length((A B C))
A_4^* length((B C)) + 1
A_4^* length((C)) + 2
A_4^* length(new) + 3
A_3 0 + 3 ≡ 3

contains((A B), B)
A_7^* contains((B), B)
A_6^* T

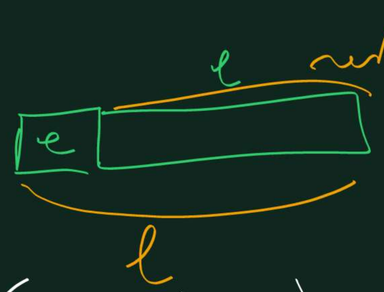
contains((A B), C)
A_7^* contains((B), C)
A_7^* contains(new, C)
A_5 F

isEmpty((A B C)) → F
length((A B C)) → 3
contains((A B C), D) → F

↓

contains(cons(A, cons(B, cons(C, new))), D)

A_4^* length(l) ≡ length(rest(l)) + 1



A_7 contains(cons(e1, l), e2) ≡ contains(l, e2)
A_7^* contains(l, l) ≡ contains(rest(l), l)

new extension with nth, insert, removeElt and remove

- nth((A B C D), 3) → C
- insert((A B C D), E, 3) → (A B E C D)
- removeElt((A B C D), C) → (A B D)
- remove((A B C D), 2) → (A C D)

ADT extension List

Operations

nth: List × Natural → Element

insert: List × Element × Natural → List

removeElt: List × Element → List

remove: List × Natural → List

Preconditions:

nth(l, n) if $1 \leq n \leq \text{length}(l)$

insert(l, e, n) if ——————

remove(l, n) if ——————

Axioms

A_1 nth(l, 1) ≡ first(l)

A_2 nth(l, n) ≡ nth(rest(l), n-1)

A_3 insert(l, e, 1) ≡ Cons(e, l)

A_4 insert(l, e, n) ≡ insert(rest(l), e, n-1)

insert((A B), C, 2)

A_4 insert((B), C, 1)

A_3 Cons(C, (B)) ≡ (C B)

A_4 insert(l, e, n) ≡

Cons(first(l), insert(rest(l), e, n-1))

A_5 removeElt(nil, e) ≡ nil

A_6 removeElt(cons(e, l), e) ≡ removeElt(l, e)

A_7 removeElt(cons(e_1, l), e_2) ≡ cons(e_1, removeElt(l, e_2))

removeElt((A B C), B)

A_7 cons(A, removeElt((B C), B))

A_6 cons(A, removeElt((C), B))

A_7 cons(A, cons(C, removeElt(nil, B)))

cons(A, cons(C, nil)) ≡ (A C)

$\forall l, n \quad remove(l, 1) \equiv rest(l)$

$\forall l, n \quad remove(l, n) \equiv cons(first(l), remove(rest(l), n-1))$

$remove((A\ B\ C), 2)$

$\forall x, l \quad cons(x, nil) \equiv (x)$

$\forall x, l \quad cons(x, (y\ z)) \equiv (x\ y\ z)$

$\equiv cons(A, remove((B\ C), 1))$

$\equiv cons(A, rest((B\ C)))$

$\equiv cons(A, (C))$

$\equiv (A\ C)$

Complexity of remove!

param: n (position of removal)

unit op: cons

$C(1) = 0$

$C(n) = C(n-1) + 1$

$= (C(n-2) + 1) + 1$

$= C(n-2) + 2$

$= C(n-i) + i$

$i = n-1$

$= C(1) + n - 1 = \Theta(n)$

param : n (length of list)

unit op : cons

hyp : all positions are equally likely

$$D(n) = \sum_{i=1}^{n} \underbrace{p(i)}_{\frac{1}{n}} C(i) = \frac{1}{n} \sum_{i=1}^{n} i - 1 = \frac{1}{n} \frac{n(n-1)}{2}$$

$$= \frac{n-1}{2}$$

exercise: define an extension to ADT List with the following operations:

- isSorted(l): tests if list l is sorted
- merge(l_1, l_2): merge 2 sorted lists

extension — Use Element($\leq$)