Lenguajes de Programación

2020 — 2º Semestre

Clase 4:

- Inductive datatypes and recursion

Federico Olmedo

Induction principle:

To prove a property P over the set of natural numbers we must:

- Prove that it holds for 0
- Prove that it holds for n+1 assuming it holds for n

$$\frac{P(0) \qquad \forall n.\ P(n) \implies P(n+1)}{\forall n.\ P(n)}$$

Induction principle:

To prove a property P over the set of natural numbers we must:

- Prove that it holds for 0
- Prove that it holds for n+1 assuming it holds for n

$$\frac{P(0) \qquad \forall n.\ P(n) \implies P(n+1)}{\forall n.\ P(n)}$$

Recursion scheme:

To define a function f over the set of natural numbers we must:

- Define it for 0
- Define it for n+1 assuming we know its value for n

$$\frac{\begin{array}{lcl} f(0) & = & \ldots\ldots\ldots \\ f(n+1) & = & \ldots f(n) \ldots \end{array}}{\forall n.\ \text{"} f(n) \text{ is univocously defined"}}$$

The natural numbers as an inductive set

$\mathbb{N}$ is the least set satisfying the following rules:

$$\frac{}{0 \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{n+1 \in \mathbb{N}}$$

Induction principle:

To prove a property P over the set of natural numbers we must:

- Prove that it holds for 0
- Prove that it holds for n+1 assuming it holds for n

$$\frac{P(0) \qquad \forall n.\ P(n) \implies P(n+1)}{\forall n.\ P(n)}$$

Recursion scheme:

To define a function f over the set of natural numbers we must:

- Define it for 0
- Define it for n+1 assuming we know its value for n

$$\frac{\begin{array}{lcl} f(0) & = & \ldots\ldots\ldots \\ f(n+1) & = & \ldots f(n) \ldots \end{array}}{\forall n.\ \text{"}f(n) \text{ is univocously defined"}}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\mathsf{empty} \in \mathsf{List}}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\mathsf{empty} \in \mathsf{List}} \qquad \frac{l \in \mathsf{List}}{\mathsf{cons}\ v\ l \ \in\ \mathsf{List}}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\mathsf{empty} \in \mathsf{List}} \qquad \frac{l \in \mathsf{List}}{\mathsf{cons}\ v\ l \ \in\ \mathsf{List}}$$

INDUCTION PRINCIPLE:

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\text{empty} \in \text{List}} \qquad \frac{l \in \text{List}}{\text{cons } v \ l \ \in \ \text{List}}$$

INDUCTION PRINCIPLE:

$$\frac{}{\forall l \in \text{List}.\ P(l)}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\text{empty} \in \text{List}} \qquad \frac{l \in \text{List}}{\text{cons } v \ l \ \in \ \text{List}}$$

INDUCTION PRINCIPLE:

$$\frac{P(\text{empty})}{\forall l \in \text{List}.\ P(l)}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\text{empty} \in \text{List}} \qquad \frac{l \in \text{List}}{\text{cons } v\ l \ \in \ \text{List}}$$

INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c} P(\text{empty}) \\ \forall v\, \forall l \in \text{List}.\ P(l) \implies P(\text{cons } v\ l) \end{array}}{\forall l \in \text{List}.\ P(l)}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\text{empty} \in \text{List}} \qquad \frac{l \in \text{List}}{\text{cons } v \; l \;\in\; \text{List}}$$

INDUCTION PRINCIPLE:

$$\frac{P(\text{empty}) \qquad \forall v \, \forall l \in \text{List}.\; P(l) \implies P(\text{cons } v \; l)}{\forall l \in \text{List}.\; P(l)}$$

RECURSION SCHEME:

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\mathsf{empty} \in \mathsf{List}} \qquad \frac{l \in \mathsf{List}}{\mathsf{cons}\ v\ l \ \in \ \mathsf{List}}$$

INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c} P(\mathsf{empty}) \\ \forall v\, \forall l \in \mathsf{List}.\ P(l) \implies P(\mathsf{cons}\ v\ l) \end{array}}{\forall l \in \mathsf{List}.\ P(l)}$$

RECURSION SCHEME:

$$\begin{array}{lcl} f(\mathsf{empty}) & = & \ldots\ldots\ldots \\ f(\mathsf{cons}\ v\ l) & = & \ldots f(l) \ldots \end{array}$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\text{empty} \in \text{List}} \qquad \frac{l \in \text{List}}{\text{cons } v \ l \ \in \ \text{List}}$$

INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c} P(\text{empty}) \\ \forall v \, \forall l \in \text{List}. \ P(l) \implies P(\text{cons } v \ l) \end{array}}{\forall l \in \text{List}. \ P(l)}$$

RECURSION SCHEME:

$$\begin{aligned} f(\text{empty}) &= \ \ldots\ldots\ldots \\ f(\text{cons } v \ l) &= \ \ldots f(l) \ldots \end{aligned}$$

$$\mathit{length}(\text{empty}) = 0$$

Inductive sets

From the definition of an inductive set one can "blindly" derive its induction principle and recursion scheme.

Lists:

INDUCTIVE DEFINITION: List is the least set satisfying the following rules:

$$\frac{}{\text{empty} \in \text{List}} \qquad \frac{l \in \text{List}}{\text{cons } v \ l \ \in \ \text{List}}$$

INDUCTION PRINCIPLE:

$$\frac{P(\text{empty}) \qquad \forall v\, \forall l \in \text{List}.\ P(l) \implies P(\text{cons } v \ l)}{\forall l \in \text{List}.\ P(l)}$$

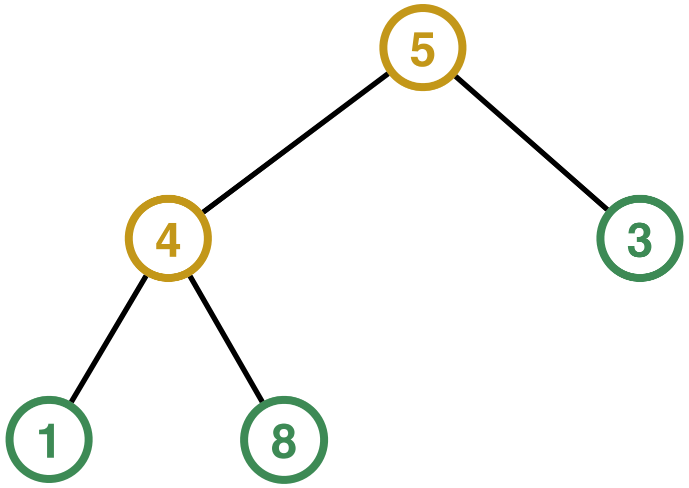
RECURSION SCHEME:

$$\begin{aligned} f(\text{empty}) &= \ .\ .\ .\ .\ .\ .\ .\ .\ . \\ f(\text{cons } v \ l) &= \ \ldots f(l) \ldots \end{aligned}$$

$$\begin{aligned} \mathit{length}(\text{empty}) &= 0 \\ \mathit{length}(\text{cons } v \ l) &= 1 + \mathit{length}(l) \end{aligned}$$

4

Inductive sets

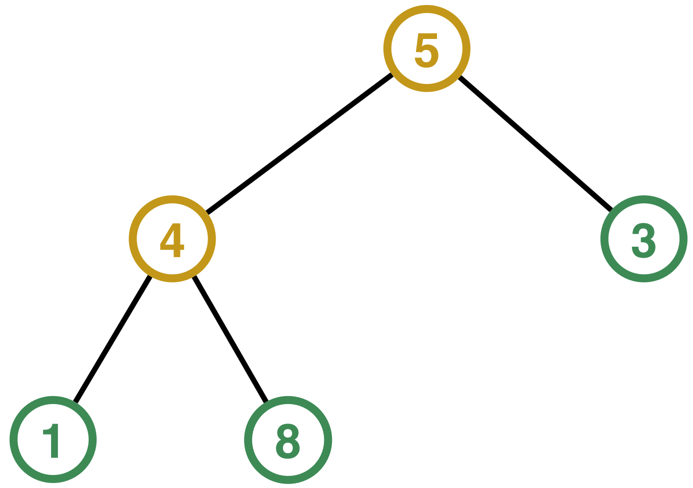


Binary trees:

INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

Inductive sets

Binary trees:

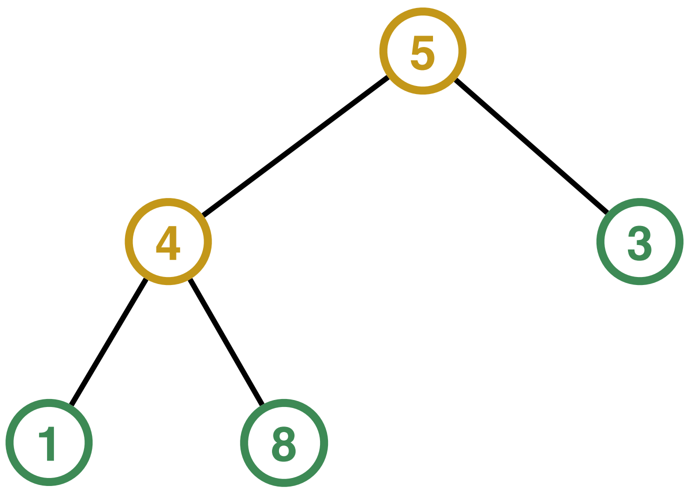


INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\mathsf{leaf}\ v) \ \in \ \mathsf{BinTree}}$$

Inductive sets

Binary trees:

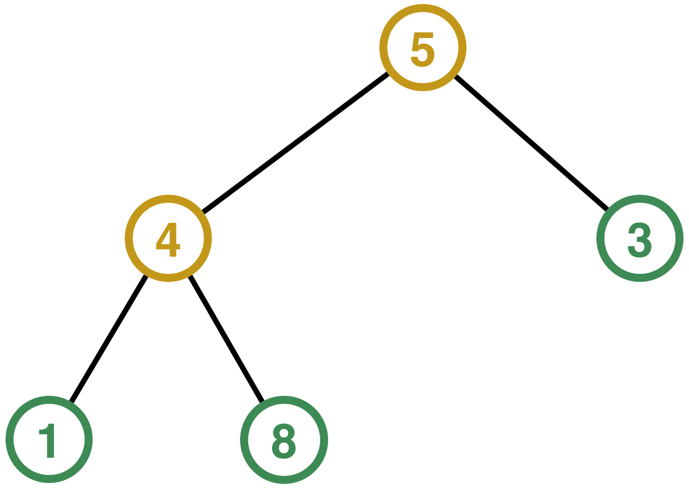


INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\mathsf{leaf}\ v) \ \in\ \mathsf{BinTree}} \qquad \frac{l, r \ \in\ \mathsf{BinTree}}{(\mathsf{in\text{-}node}\ v\ l\ r) \ \in\ \mathsf{BinTree}}$$

Inductive sets

Binary trees:



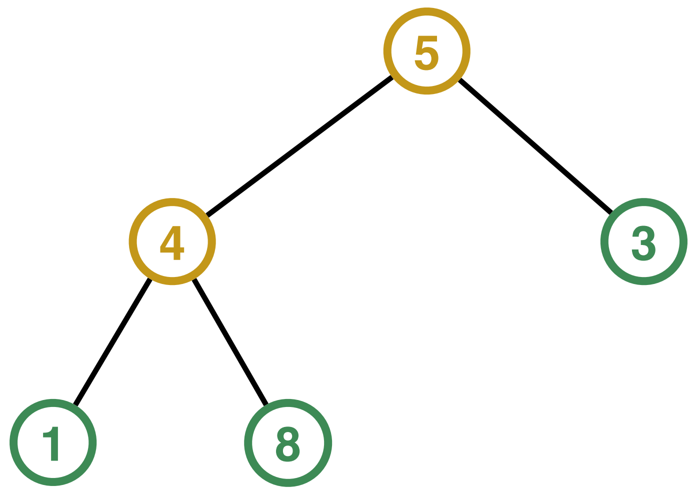
INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\textsf{leaf}\ v) \ \in\ \textsf{BinTree}} \qquad \frac{l, r \ \in\ \textsf{BinTree}}{(\textsf{in-node}\ v\ l\ r) \ \in\ \textsf{BinTree}}$$

INDUCTION PRINCIPLE:

$$\frac{}{\forall bt \in \textsf{BinTree}.\quad P(bt)}$$

Inductive sets



Binary trees:

INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\text{leaf } v) \;\in\; \text{BinTree}} \qquad \frac{l, r \;\in\; \text{BinTree}}{(\text{in-node } v\ l\ r) \;\in\; \text{BinTree}}$$

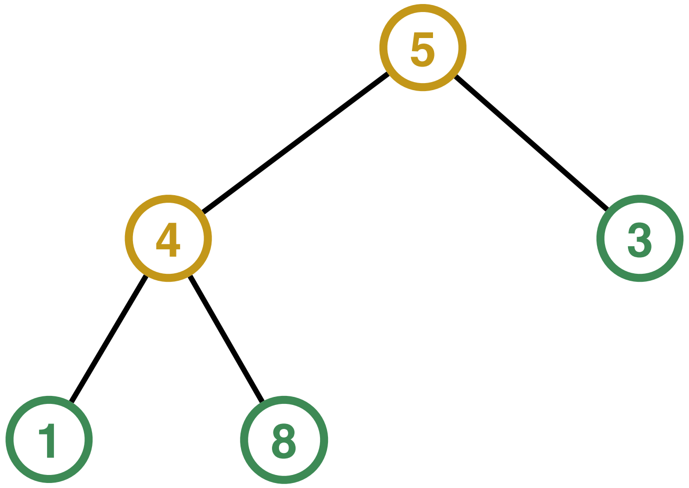
INDUCTION PRINCIPLE:

$$\frac{\forall v.\;\; P(\text{leaf } v)}{\forall bt \in \text{BinTree}.\;\; P(bt)}$$

4

Inductive sets

Binary trees:



INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

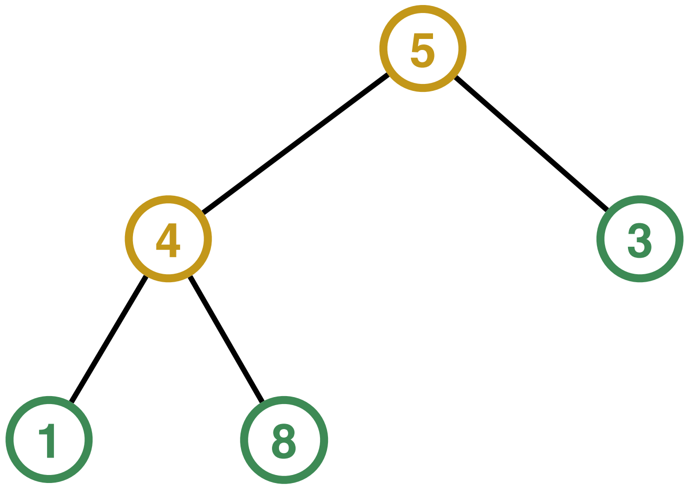
$$\frac{}{(\text{leaf } v) \;\in\; \text{BinTree}} \qquad \frac{l, r \;\in\; \text{BinTree}}{(\text{in-node } v\ l\ r) \;\in\; \text{BinTree}}$$

INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c}\forall v.\;\; P(\text{leaf } v) \\ \forall v\, \forall l, r \in \text{BinTree}.\;\; P(l) \wedge P(r) \implies P(\text{in-node } v\ l\ r)\end{array}}{\forall bt \in \text{BinTree}.\;\; P(bt)}$$

4

Inductive sets



Binary trees:

INDUCTIVE DEFINITION:

BinTree is the least set satisfying the following rules:

$$\frac{}{(\text{leaf } v) \in \text{BinTree}} \qquad \frac{l, r \in \text{BinTree}}{(\text{in-node } v\ l\ r) \in \text{BinTree}}$$

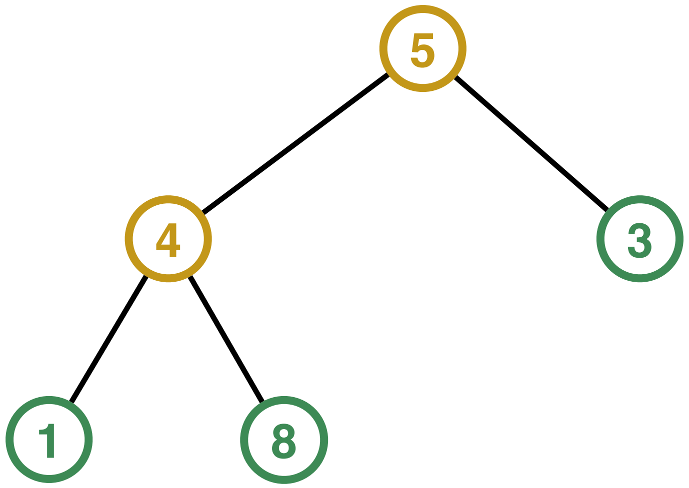
INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c}\forall v.\ \ P(\text{leaf } v) \\ \forall v\, \forall l, r \in \text{BinTree}.\ \ P(l) \wedge P(r) \implies P(\text{in-node } v\ l\ r)\end{array}}{\forall bt \in \text{BinTree}.\ \ P(bt)}$$

RECURSION SCHEME:

Inductive sets

Binary trees:



INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\text{leaf } v) \ \in \ \text{BinTree}} \qquad \frac{l, r \ \in \ \text{BinTree}}{(\text{in-node } v\ l\ r) \ \in \ \text{BinTree}}$$

INDUCTION PRINCIPLE:

$$\frac{\forall v. \ \ P(\text{leaf } v) \qquad \forall v\, \forall l, r \in \text{BinTree}. \ \ P(l) \wedge P(r) \implies P(\text{in-node } v\ l\ r)}{\forall bt \in \text{BinTree}. \ \ P(bt)}$$

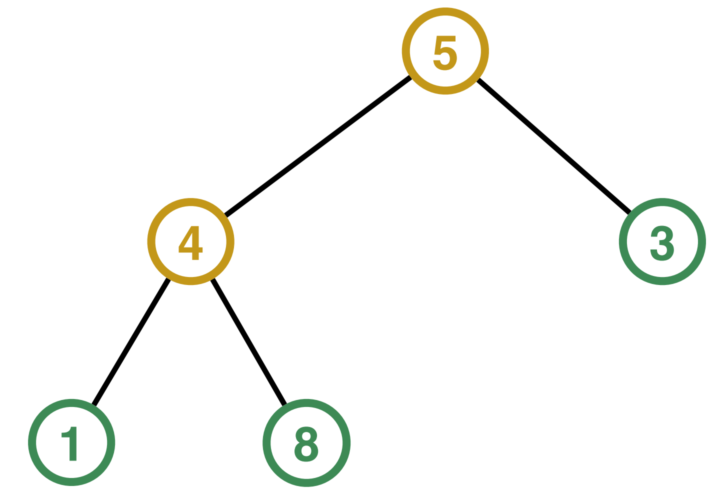
RECURSION SCHEME:

$$f(\text{leaf } v) \quad = \quad \ldots\ldots\ldots$$

4

Inductive sets

Binary trees:



INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\text{leaf } v) \in \text{BinTree}} \qquad \frac{l, r \in \text{BinTree}}{(\text{in-node } v\ l\ r) \in \text{BinTree}}$$

INDUCTION PRINCIPLE:

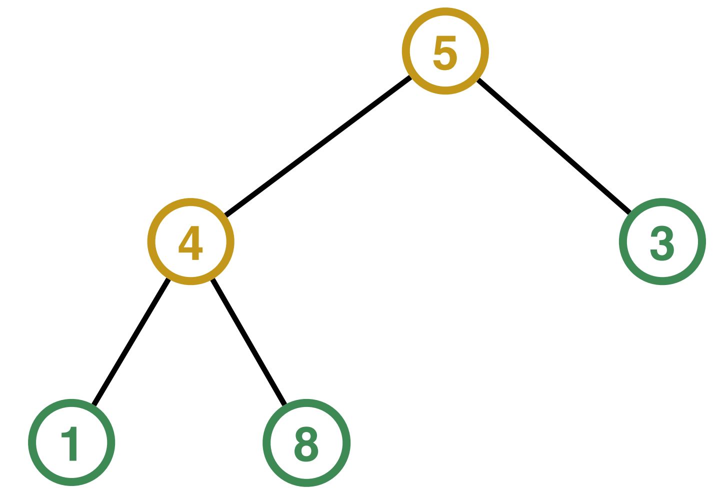
$$\frac{\begin{array}{c}\forall v.\ \ P(\text{leaf } v) \\ \forall v\, \forall l, r \in \text{BinTree}.\ \ P(l) \wedge P(r) \implies P(\text{in-node } v\ l\ r)\end{array}}{\forall bt \in \text{BinTree}.\ \ P(bt)}$$

RECURSION SCHEME:

$$\begin{array}{lcl} f(\text{leaf } v) & = & \ldots\ldots\ldots \\ f(\text{in-node } v\ l\ r) & = & \ldots f(l) \ldots f(r) \ldots \end{array}$$

4

Inductive sets



Binary trees:

INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\mathsf{leaf}\ v) \ \in \ \mathsf{BinTree}} \qquad \frac{l, r \ \in \ \mathsf{BinTree}}{(\mathsf{in\text{-}node}\ v\ l\ r) \ \in \ \mathsf{BinTree}}$$

INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c}\forall v.\ \ P(\mathsf{leaf}\ v) \\ \forall v\, \forall l, r \in \mathsf{BinTree}.\ \ P(l) \wedge P(r) \implies P(\mathsf{in\text{-}node}\ v\ l\ r)\end{array}}{\forall bt \in \mathsf{BinTree}.\ \ P(bt)}$$

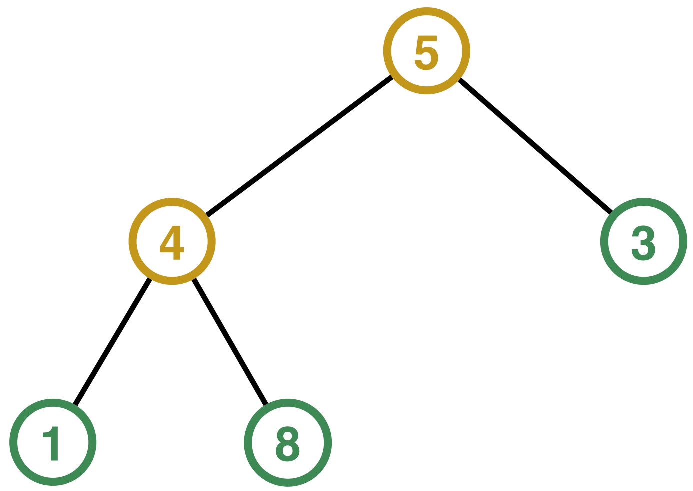
RECURSION SCHEME:

$$\begin{aligned} f(\mathsf{leaf}\ v) &= \ \ \ldots\ldots\ldots \\ f(\mathsf{in\text{-}node}\ v\ l\ r) &= \ \ \ldots f(l) \ldots f(r) \ldots \end{aligned}$$

$$\begin{aligned} height(\mathsf{leaf}\ v) &= \\ height(\mathsf{in\text{-}node}\ v\ l\ r) &= \end{aligned}$$

Inductive sets

Binary trees:



INDUCTIVE DEFINITION:

BinTree is the least set satisfying the following rules:

$$\frac{}{(\text{leaf } v) \ \in \ \text{BinTree}} \qquad \frac{l, r \ \in \ \text{BinTree}}{(\text{in-node } v\ l\ r) \ \in \ \text{BinTree}}$$

INDUCTION PRINCIPLE:

$$\frac{\forall v.\ \ P(\text{leaf } v) \qquad \forall v\, \forall l, r \in \text{BinTree}.\ \ P(l) \wedge P(r) \implies P(\text{in-node } v\ l\ r)}{\forall bt \in \text{BinTree}.\ \ P(bt)}$$

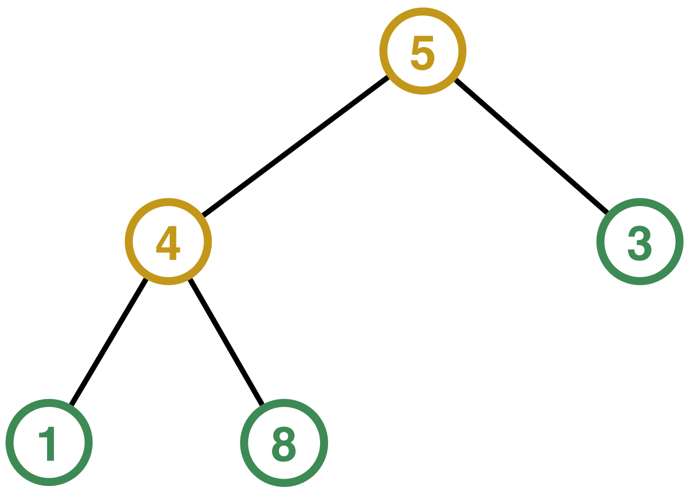
RECURSION SCHEME:

$$\begin{aligned} f(\text{leaf } v) &= \ \ldots\ldots\ldots \\ f(\text{in-node } v\ l\ r) &= \ \ldots f(l) \ldots f(r) \ldots \end{aligned}$$

$$\begin{aligned} height(\text{leaf } v) &= \ 0 \\ height(\text{in-node } v\ l\ r) &= \end{aligned}$$

Inductive sets

Binary trees:



INDUCTIVE DEFINITION: BinTree is the least set satisfying the following rules:

$$\frac{}{(\text{leaf } v) \;\in\; \text{BinTree}} \qquad \frac{l, r \;\in\; \text{BinTree}}{(\text{in-node } v\; l\; r) \;\in\; \text{BinTree}}$$

INDUCTION PRINCIPLE:

$$\frac{\begin{array}{c}\forall v.\;\; P(\text{leaf } v) \\ \forall v\, \forall l, r \in \text{BinTree}.\;\; P(l) \wedge P(r) \implies P(\text{in-node } v\; l\; r)\end{array}}{\forall bt \in \text{BinTree}.\;\; P(bt)}$$

RECURSION SCHEME:

$$\begin{aligned} f(\text{leaf } v) &= \;\ldots\ldots\ldots \\ f(\text{in-node } v\; l\; r) &= \;\ldots f(l) \ldots f(r) \ldots \end{aligned}$$

$$\begin{aligned} height(\text{leaf } v) &= 0 \\ height(\text{in-node } v\; l\; r) &= 1 + \max\{height(l), height(r)\} \end{aligned}$$

5

Inductive types and recursion in Racket

Inductive types and recursion in Racket

```
#|
<BinTree> ::=  (leaf <value>)
            |  (in-node <value> <BinTree> <BinTree>)
|#
;; Inductive type for representing binary trees
(deftype BinTree
  (leaf value)
  (in-node value left right))


;; height :: BinTree -> Integer
;; Devuelve la altura del arbol binario
(define (height bt)
  (match bt
    [(leaf _) 0]
    [(in-node _ l r) (+ 1 (max (height l) (height r)))]))
```

Inductive types and recursion in Racket

All recursive function over binary trees will have the following template:

(which is syntactically derived from the grammar of binary trees)

```
(define (func-to-define bt)
  (match bt
    [(leaf v) ....]
    [(in-node v l r) (...(func-to-define l)...(func-to-define r)...)]))
```

Inductive types and recursion in Racket

All recursive function over binary trees will have the following template:

(which is syntactically derived from the grammar of binary trees)

```
(define (func-to-define bt)
  (match bt
    [(leaf v) ....]
    [(in-node v l r) (...(func-to-define l)...(func-to-define r)...)]))
```

1. Define function `(sum-bintree bt)` that returns the sum of the elements in the nodes of (binary tree) `bt`.
2. Define function `(max-bintree bt)` that returns the maximum of the elements in the nodes of (binary tree) `bt`.

Capturing the recursive scheme

Capturing the recursive scheme

```
;; fold-bintree :: (Number -> A) (Number A A -> A) -> (Bintree -> A)
;; fold over numeric binary trees
(define (fold-bintree f g)
  (λ (bt)
    (match bt
    [(leaf v) (f v)]
    [(in-node v l r) (g v
                        ((fold-bintree f g) l)
                        ((fold-bintree f g) r))])))

;; max-bintree :: BinTree -> Number
;; Returns the maximum element of a (numeric) binary tree
(define max-bintree
  (fold-bintree identity max))

;; sum-bintree :: Bintree -> Number
;; Returns the sum of the elements of a numeric binary tree
(define sum-bintree
  (fold-bintree identity +))
```

Capturing the recursive scheme

```
;; fold-bintree :: (Number -> A) (Number A A -> A) -> (Bintree -> A)
;; fold over numeric binary trees
(define (fold-bintree f g)
  (λ (bt)
    (match bt
    [(leaf v) (f v)]
    [(in-node v l r) (g v
                        ((fold-bintree f g) l)
                        ((fold-bintree f g) r))])))

;; max-bintree :: BinTree -> Number
;; Returns the maximum element of a (numeric) binary tree
(define max-bintree
  (fold-bintree identity max))

;; sum-bintree :: Bintree -> Number
;; Returns the sum of the elements of a numeric binary tree
(define sum-bintree
  (fold-bintree identity +))
```

Define function `(contains-bintree? bt v)` using fold-bintree.

Properties of inductive datatypes

In any data structure inductively defined using `deftype` it holds that:

Properties of inductive datatypes

In any data structure inductively defined using `deftype` it holds that:

- All constructors are injective functions. For instance `(equal? (leaf a) (leaf b))` reduces to `#t` only when `(equal? a b)` is `#t`.

Properties of inductive datatypes

In any data structure inductively defined using `deftype` it holds that:

- All constructors are injective functions. For instance `(equal? (leaf a) (leaf b))` reduces to `#t` only when `(equal? a b)` is `#t`.

- Values built from different constructors are always different. For instance `(equal? (leaf a) (in-node b (…) (…)))` reduces to `#f` (for all `a` and `b`)

Properties of inductive datatypes

In any data structure inductively defined using `deftype` it holds that:

- All constructors are injective functions. For instance `(equal? (leaf a) (leaf b))` reduces to `#t` only when `(equal? a b)` is `#t`.

- Values built from different constructors are always different. For instance `(equal? (leaf a) (in-node b (…) (…)))` reduces to `#f` (for all a and b)

- The only way of building values of the data structure is via the provided constructors.

9 Takeaways

Follow the Grammar!

When defining a function that operates on an inductive datatype, the definition should be patterned after the grammar of the datatype.

Lecture material

Bibliography

- PrePLAI: Introduction to functional programming in Racket [Sections 4-5]

- Essentials of Programming Languages (3rd Edition)
  Daniel P. Friedman
  [Chapter 1]

- Source code from the lecture [Download]