

Programmierung 1

Vorlesung 16

Livestream beginnt um 10:20 Uhr

Induktive Korrektheits- beweise, Teil 2

Programmierung 1

Wohlfundierte Induktion (Noethersche Induktion)

- ▶ Sei $\forall x \in X : A(x)$ eine **allquantifizierte Aussage** über eine Menge X .
- ▶ Eine **Induktionsrelation** $>$ ist eine **terminierende Relation** auf X .
Um die allquantifizierte Aussage zu beweisen, zeigen wir den **Induktionsschritt**:
für jedes Argument x folgt aus der Tatsache,
dass für jedes kleinere Argument y $A(y)$ gilt, dass $A(x)$ gilt.
- ▶ **Wohlfundierte Induktion:**

$$(\forall x \in X (\forall y \in X : x > y \Rightarrow A(y)) \Rightarrow A(x)) \Rightarrow \forall x \in X : A(x)$$

Induktionsschritt

Natürliche vs. strukturelle Induktion

- ▶ Wenn es sich bei der Induktionsrelation um Ter handelt, sprechen wir von **natürlicher Induktion**.

$$Ter := \{(x, y) \in \mathbb{N}^2 \mid x > y\}$$

- ▶ Wenn es sich bei der Induktionsrelation um eine **strukturelle Relation** handelt, sprechen wir von **struktureller Induktion**.
Dazu später mehr.

Bestimmte Iteration

$$\text{iter}: (\mathbb{N} \times X \times (X \rightarrow X)) \rightarrow X$$

$$\text{iter}(0, x, f) = x$$

$$\text{iter}(n, x, f) = \text{iter}(n-1, f x, f) \quad \text{für } n > 0$$

Behauptung: (Vertauschungseigenschaft von *iter*)

$$\forall n \in \mathbb{N} \quad \forall x \in X \quad \forall f \in X \rightarrow X: \quad \text{iter}(n+1, x, f) = f(\text{iter}(n, x, f)).$$

Beweis durch Induktion über $n \in \mathbb{N}$. Wir unterscheiden zwei Fälle.

Sei $n=0$. Sei $x \in X$ beliebig, sei $f \in X \rightarrow X$ beliebig.

$$\begin{aligned} \text{iter}(n+1, x, f) &= \text{iter}(1, x, f) = \text{iter}(0, f x, f) = f x \\ &= f(\text{iter}(0, x, f)) = f(\text{iter}(n, x, f)) \end{aligned}$$

Sei $n > 0$. Sei $x \in X$ beliebig, sei $f \in X \rightarrow X$ beliebig.

$$\begin{aligned} \text{iter}(n+1, x, f) &= \text{iter}(n, f x, f) && \text{Definition } \text{iter} \\ &= f(\text{iter}(n-1, f x, f)) && \text{Induktion für } n-1 \\ &= f(\text{iter}(n, x, f)) && \text{Definition } \text{iter} \end{aligned}$$

Iterative Berechnungen

$$\text{iter}: (\mathbb{N} \times X \times (X \rightarrow X)) \rightarrow X$$

$$\text{iter}(0, x, f) = x$$

$$\text{iter}(n, x, f) = \text{iter}(n-1, f x, f) \quad \text{für } n > 0$$

► Iterative Bestimmung von Potenzen

$$1 \rightarrow x^1 \rightarrow x^2 \rightarrow x^3 \rightarrow \dots \rightarrow x^n$$

$$f = \lambda a. a \cdot x$$

► Iterative Bestimmung der Fakultäten

$$(1, \text{fac } 0) \rightarrow (2, \text{fac } 1) \rightarrow (3, \text{fac } 2) \rightarrow \dots \rightarrow (n+1, \text{fac } n)$$

$$f = \lambda (k, x). (k+1, k \cdot x) \quad \text{fac } n = \#2(\text{iter}(n, (1, 1), f))$$

► Iterative Bestimmung der Fibonacci-Zahlen

$$(\text{fib } 0, \text{fib } 1) \rightarrow (\text{fib } 1, \text{fib } 2) \rightarrow (\text{fib } 2, \text{fib } 3) \rightarrow \dots \rightarrow (\text{fib } n, \text{fib}(n+1))$$

$$f = \lambda (x, y). (y, x + y) \quad \text{fib } n = \#2(\text{iter}(n-1, (0, 1), f))$$

Iterative Bestimmung von Potenzen

$iter: (\mathbb{N} \times X \times (X \rightarrow X)) \rightarrow X$

$iter(0, x, f) = x$

$iter(n, x, f) = iter(n-1, f x, f) \quad \text{für } n > 0$



Bitte **immer** angeben:

- durch Induktion über ...
- Induktion für ...
- Begründungen für Umformungen

Behauptung: $\forall x \in \mathbb{Z} \quad \forall n \in \mathbb{N}: x^n = iter(n, 1, \lambda a. a \cdot x).$

Beweis: Sei $x \in \mathbb{Z}$ und $f = \lambda a. a \cdot x$.

Wir zeigen $\forall n \in \mathbb{N}: x^n = iter(n, 1, \lambda a. a \cdot x)$ durch Induktion über $n \in \mathbb{N}$.

Sei $n=0$. $iter(n, 1, f) = iter(0, 1, f) = 1 = x^0 = x^n$.

Sei $n > 0$.
 $iter(n, 1, f) = f(iter(n-1, 1, f))$
 $= f(x^{n-1})$
 $= x^{n-1} \cdot x$
 $= x^n$

Vertauschungseigenschaft
Induktion für $n-1$
Definition f

Unbestimmte Iteration

$$\textit{first}: \underbrace{(X \rightarrow \mathbb{B})}_{\text{Zielbedingung}} \times \underbrace{(X \rightarrow X)}_{\text{Schrittfunktion}} \times \underbrace{X}_{\text{Startwert}} \rightarrow X$$

$$\textit{first}(p, f, x) = \text{if } p\ x \text{ then } x \text{ else } \textit{first}(p, f, f\ x)$$

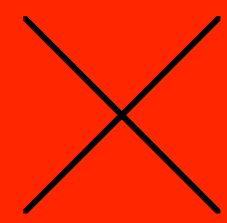


Terminiert die mathematische Prozedur first?

$first: (X \rightarrow \mathbb{B}) \times (X \rightarrow X) \times X \rightarrow X$

$first(p, f, x) = \text{if } p\ x \text{ then } x \text{ else } first(p, f, f\ x)$

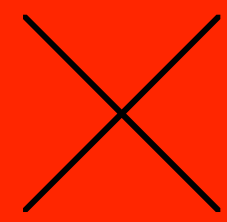
A: Ja, für alle Argumente



B: Ja, für manche (aber
nicht alle) Argumente




C: Nein



Unbestimmte Iteration

$$\text{first}: (X \rightarrow \mathbb{B}) \times (X \rightarrow X) \times X \rightarrow X$$



$$\text{first}(p, f, x) = \text{if } p \ x \text{ then } x \text{ else } \text{first}(p, f, f \ x)$$

- ▶ Im Gegensatz zu *iter* **terminiert** *first* **nicht unbedingt!**
- ▶ *first* kann die Ergebnisfunktion jeder endrekursiven Prozedur berechnen.

Beispiel:

$$\text{iter}(n, x, f) = \#2 (\text{first} (\lambda (k, a) . k=0, \lambda (k, a) . (k-1, f \ a), (n, x)))$$

Berechnung von *iter* mit *first*

$first: (X \rightarrow \mathbb{B}) \times (X \rightarrow X) \times X \rightarrow X$

$first(p, f, x) = \text{if } p\ x \text{ then } x \text{ else } first(p, f, f\ x)$

$iter: (\mathbb{N} \times X \times (X \rightarrow X)) \rightarrow X$

$iter(0, x, f) = x$

$iter(n, x, f) = iter(n-1, f\ x, f) \quad \text{für } n > 0$

Behauptung: $\forall n \in \mathbb{N}, x \in X, f: X \rightarrow X: iter(n, x, f) = \#2(first(p, g, (n, x)))$

mit $p = \lambda(k, a). k=0, \quad g = \lambda(k, a). (k-1, f\ a)$

Beweis: Durch Induktion über $n \in \mathbb{N}$. Wir unterscheiden zwei Fälle.

Sei $n=0$. $\#2(first(p, g, (n, x))) = \#2(first(p, g, (0, x))) = \#2(0, x) = x$
 $= iter(0, x, f) = iter(n, x, f)$.

Sei $n > 0$. $\#2(first(p, g, (n, x))) = \#2(first(p, g, (n-1, f\ x)))$ Definition *first*, *p* und *g*
 $= iter(n-1, f\ x, f)$ Induktion für $n-1$
 $= iter(n, x, f)$ Definition *iter*

Mathematische Prozeduren auf Listen

$$\mathcal{L}(X) := \{\langle \rangle\} \cup (X \times \mathcal{L}(X))$$

Listen über X

$$|_| : \mathcal{L}(X) \rightarrow \mathbb{N}$$

Länge

$$|nil| = 0$$

$$|x::xr| = 1 + |xr|$$

$$@ : \mathcal{L}(X) \times \mathcal{L}(X) \rightarrow \mathcal{L}(X)$$

Konkatenation

$$nil@ys = ys$$

$$(x::xr)@ys = x::(xr@ys)$$

$$rev : \mathcal{L}(X) \rightarrow \mathcal{L}(X)$$

Reversion

$$rev nil = nil$$

$$rev(x::xr) = (rev xr)@[x]$$

$$foldl : (X \times Y \rightarrow Y) \times Y \times \mathcal{L}(X) \rightarrow Y$$

Faltung von links

$$foldl(f, y, nil) = y$$

$$foldl(f, y, x::xr) = foldl(f, f(x, y), xr)$$

Strukturelle Relationen

- ▶ Die **Konstituenten** einer Menge X sind rekursiv definiert:
 1. Jedes **Element** von X ist eine Konstituente von X .
 2. Jede **Konstituente eines Elements** von X ist eine Konstituente von X .
- ▶ **Beispiel:** $\{\{\{\emptyset\}\}\}$ hat die Konstituenten $\{\{\emptyset\}\}$, $\{\emptyset\}$ und \emptyset .
- ▶ Eine Menge ist genau dann **finitär** wenn sie nur **endlich viele** Konstituenten hat.
- ▶ Eine Relation heißt **strukturell**, wenn für jede Kante $(x,y) \in R$ gilt, dass y eine Konstituente von x ist.
- ▶ **Proposition:** Jede **strukturelle** Relation ist **terminierend**.
- ▶ **Beispiel:** *Ter* ist eine strukturelle Relation.

Strukturelle Terminierungsfunktionen

- ▶ Eine Funktion f heißt **strukturelle Terminierungsfunktion** für R , wenn $\text{Dom } f = \text{Ver } R$ und **für jede Kante** $(x,y) \in R$ gilt dass $f y$ eine **Konstituente** von $f x$ ist.
- ▶ **Proposition:** Jede Relation, für die es eine **strukturelle Terminierungsfunktion** gibt, ist **terminierend**.

Beweis durch Widerspruch:

- ▶ Sei R eine nicht terminierende Relation und f eine strukturelle Terminierungsfunktion für R .
- ▶ Also ist $\{(f x, f y) \mid (x,y) \in R\}$ eine strukturelle Relation die fortschreitend ist.
- ▶ Strukturelle Relationen sind aber terminierend.
- ▶ **Widerspruch.**

Strukturelle Terminierungsfunktionen für Prozeduren

- ▶ Sei $p: X \rightarrow Y$ eine Prozedur. Dann heißt eine Funktion f mit $\text{Dom } f = X$ **strukturelle Terminierungsfunktion** für p , wenn für jeden Rekursionsschritt (x, x') von p gilt, dass $f x'$ eine **Konstituente** von x ist.
- ▶ **Proposition:** Jede Prozedur, für die es eine **strukturelle Terminierungsfunktion** gibt, **terminiert für alle Argumente**.

Strukturelle Terminierungsfunktionen

$$|_| : \mathcal{L}(X) \rightarrow \mathbb{N}$$

$$|nil| = 0$$

$$|x::xr| = 1 + |xr|$$

$$\lambda xs \in \mathcal{L}(X). xs$$

$$rev : \mathcal{L}(X) \rightarrow \mathcal{L}(X)$$

$$rev nil = nil$$

$$rev(x::xr) = (rev xr) @ [x]$$

$$\lambda xs \in \mathcal{L}(X). xs$$

$$foldl : (X \times Y \rightarrow Y) \times Y \times \mathcal{L}(X) \rightarrow Y$$

$$foldl(f, y, nil) = y$$

$$foldl(f, y, x::xr) = foldl(f, f(x, y), xr)$$

$$\lambda (f, y, xs) \in (X \times Y \rightarrow Y) \times Y \times \mathcal{L}(X). xs$$



Welche der folgenden Funktionen sind strukturelle Terminierungsfunktionen für @?

$$@ : \mathcal{L}(X) \times \mathcal{L}(X) \rightarrow \mathcal{L}(X)$$

$$\text{nil} @ ys = ys$$

$$(x :: xr) @ ys = x :: (xr @ ys)$$

A: $\lambda (xs, ys) \in \mathcal{L}(X)^2. xs$ ✓

B: $\lambda (xs, ys) \in \mathcal{L}(X)^2. ys$ ✗

C: $\lambda (xs, ys) \in \mathcal{L}(X)^2. (xs, ys)$ ✗

$$@ : \mathcal{L}(X) \times \mathcal{L}(X) \rightarrow \mathcal{L}(X)$$

$$nil @ ys = ys$$

$$(x :: xr) @ ys = x :: (xr @ ys)$$

Beispiel für strukturelle Induktion

► Behauptung: Assoziativität der Konkatination

Sei X eine Menge und seien $xs, ys, zs \in \mathcal{L}(X)$.

Dann gilt: $(xs @ ys) @ zs = xs @ (ys @ zs)$.

► Beweis: Seien $ys, zs \in \mathcal{L}(X)$. Wir beweisen

$$\forall xs \in \mathcal{L}(X) : (xs @ ys) @ zs = xs @ (ys @ zs)$$

durch strukturelle Induktion über $xs \in \mathcal{L}(X)$. Wir unterscheiden zwei Fälle.

Sei $xs = nil$. Dann

$$(xs @ ys) @ zs = ys @ zs$$

$$= xs @ (ys @ zs)$$

Definition @

Definition @

$$@ : \mathcal{L}(X) \times \mathcal{L}(X) \rightarrow \mathcal{L}(X)$$

$$nil @ ys = ys$$

$$(x :: xr) @ ys = x :: (xr @ ys)$$

Beispiel für strukturelle Induktion

► Behauptung: Assoziativität der Konkatination

Sei X eine Menge und seien $xs, ys, zs \in \mathcal{L}(X)$.

Dann gilt: $(xs @ ys) @ zs = xs @ (ys @ zs)$.

► Beweis: Seien $ys, zs \in \mathcal{L}(X)$. Wir beweisen

$$\forall xs \in \mathcal{L}(X) : (xs @ ys) @ zs = xs @ (ys @ zs)$$

durch strukturelle Induktion über $xs \in \mathcal{L}(X)$. Wir unterscheiden zwei Fälle.

Sei $xs = x :: xr$. Dann

$$(xs @ ys) @ zs = (x :: (xr @ ys)) @ zs$$

$$= x :: ((xr @ ys) @ zs)$$

$$= x :: (xr @ (ys @ zs))$$

$$= xs @ (ys @ zs)$$

Definition @

Definition @

Induktion für xr

Definition @

Länge von Listen mit *foldl* berechnen

$$\text{foldl} : (X \times Y \rightarrow Y) \times Y \times \mathcal{L}(X) \rightarrow Y$$

$$\text{foldl}(f, y, \text{nil}) = y$$

$$\text{foldl}(f, y, x::xr) = \text{foldl}(f, f(x, y), xr)$$

$$|_| : \mathcal{L}(X) \rightarrow \mathbb{N}$$

$$|\text{nil}| = 0$$

$$|x::xr| = 1 + |xr|$$

► **Behauptung:** $\forall xs \in \mathcal{L}(X): |xs| = \text{foldl}(f, 0, xs)$

für $f = \lambda (x, a) \in X \times \mathbb{N}. a + 1$.

► **Beweis**(versuch): Durch strukturelle Induktion über $xs \in \mathcal{L}(X)$.

Sei $xs = \text{nil}$. Dann $|xs| = 0 = \text{foldl}(f, 0, \text{nil}) = \text{foldl}(f, 0, xs)$.

Definition $|_|$ und *foldl*

Sei $xs = x::xr$. Dann $\text{foldl}(f, 0, xs) = \text{foldl}(f, 0, x::xr) = \text{foldl}(f, 1, xr) = ???$

Induktion scheitert!

Verstärkung der Korrektheitsaussage

$$\text{foldl} : (X \times Y \rightarrow Y) \times Y \times \mathcal{L}(X) \rightarrow Y$$

$$\text{foldl}(f, y, \text{nil}) = y$$

$$\text{foldl}(f, y, x::xr) = \text{foldl}(f, f(x, y), xr)$$

$$|_| : \mathcal{L}(X) \rightarrow \mathbb{N}$$

$$|\text{nil}| = 0$$

$$|x::xr| = 1 + |xr|$$

► **Statt:** $\forall xs \in \mathcal{L}(X): |xs| = \text{foldl}(f, 0, xs)$

beweisen wir die **Verstärkung:**

$$\forall xs \in \mathcal{L}(X) \quad \forall n \in \mathbb{N}: |xs| + n = \text{foldl}(f, n, xs).$$



► **Beweis:** Durch strukturelle Induktion über $xs \in \mathcal{L}(X)$.

Sei $xs = \text{nil}$. Sei $n \in \mathbb{N}$. Dann $|xs| + n = n = \text{foldl}(f, n, \text{nil}) = \text{foldl}(f, n, xs)$.

Definition $|_|$ und foldl

Sei $xs = x::xr$. Sei $n \in \mathbb{N}$. Dann $\text{foldl}(f, n, xs) = \text{foldl}(f, n, x::xr) = \text{foldl}(f, n+1, xr)$

Definition foldl und f

$$= |xr| + n + 1$$

Induktion für xr

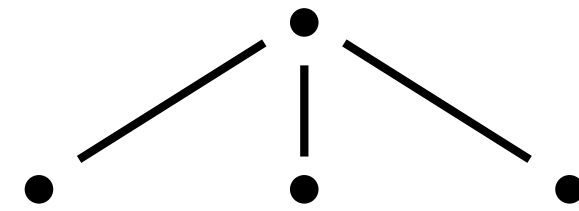
$$= |xs| + n$$

Definition $|_|$

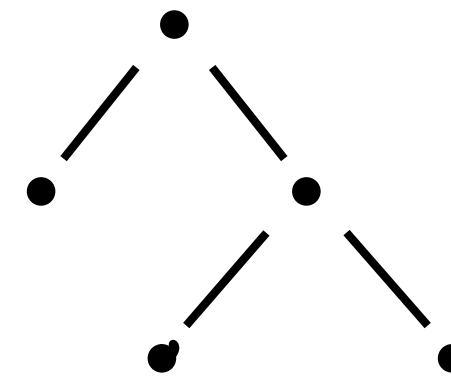
Reine Bäume



$[]$



$[[], [], []]$



$[[], [[], []], []]$

$$\mathcal{T} := \mathcal{L}(\mathcal{T})$$

Bäume

$$s : \mathcal{T} \rightarrow \mathbb{N}_+$$

Größe

$$s[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 1 \text{ else } 1 + s t_1 + \dots + s t_n$$

$$b : \mathcal{T} \rightarrow \mathbb{N}_+$$

Breite

$$b[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 1 \text{ else } b t_1 + \dots + b t_n$$

$$d : \mathcal{T} \rightarrow \mathbb{N}$$

Tiefe

$$d[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 0 \text{ else } 1 + \max\{d t_1, \dots, d t_n\}$$

Sekundäre Listenrekursion

$$s : \mathcal{T} \rightarrow \mathbb{N}_+$$

$$s[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 1 \text{ else } 1 + s t_1 + \dots + s t_n$$

- ▶ Neben der **primären Baumrekursion** verwenden die Prozeduren eine **sekundäre Listenrekursion** die durch "... " formuliert ist.
- ▶ In den **Anwendungsgleichungen** ist die sekundäre Listenrekursion nicht mehr sichtbar:

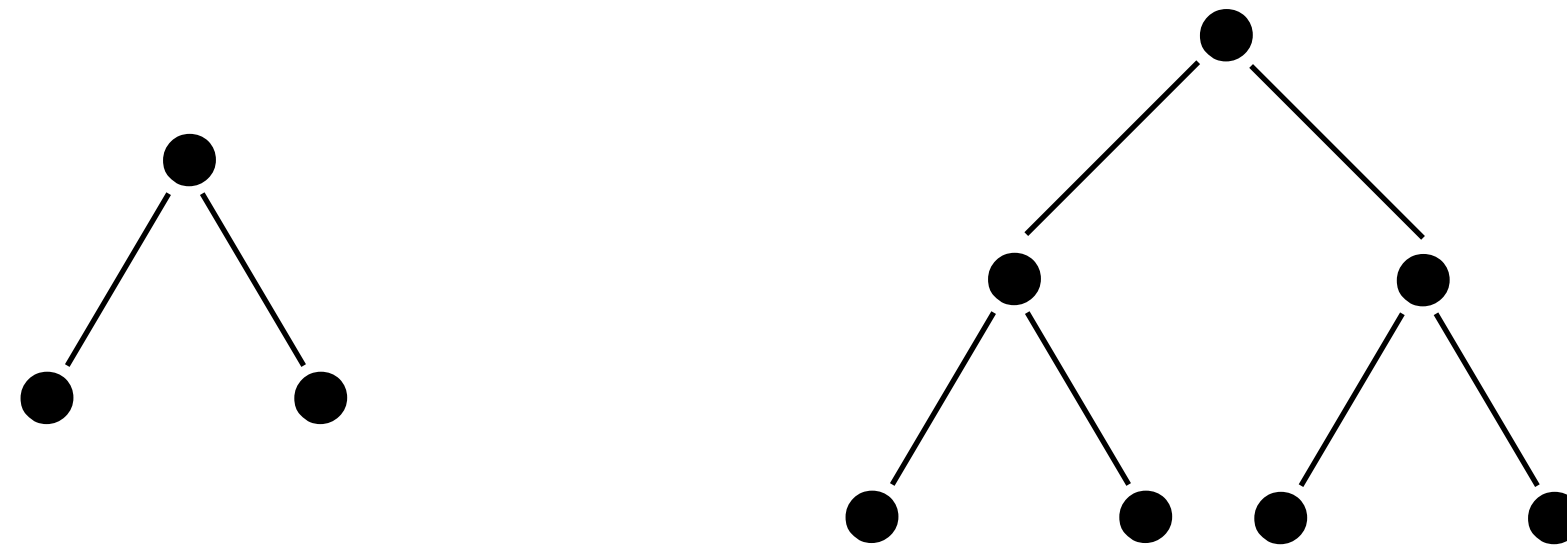
$$s[t_1, t_2] = 1 + s t_1 + s t_2$$

$$b[t_1, t_2, t_3] = b t_1 + b t_2 + b t_3$$

$$d[t_1, t_2] = 1 + \max\{d t_1, d t_2\}$$

- ▶ **Rekursionsfunktion:** $\lambda [t_1, \dots, t_n] \in \mathcal{T}. \langle t_1, \dots, t_n \rangle$
- ▶ **Terminierungsfunktion:** $\lambda t \in \mathcal{T}. t.$

Balancierte Binärbäume

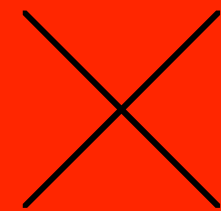


- ▶ Ein Baum ist balanciert, wenn die Adressen seiner Blätter alle die gleiche Länge haben.
- ▶ Dies ist genau dann der Fall, wenn alle Unterbäume balanciert sind, und alle die gleiche Tiefe haben (Vorlesung 12).
- ▶ Die **Menge** $\mathcal{B} \subseteq \mathcal{T}$ **der balancierten Binärbäume** definieren wir durch Rekursion:
 1. $[] \in \mathcal{B}$.
 2. Wenn $t_1 \in \mathcal{B}$, $t_2 \in \mathcal{B}$ und $d t_1 = d t_2$, dann $[t_1, t_2] \in \mathcal{B}$.

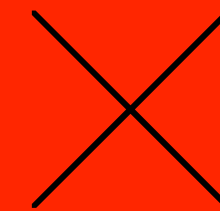


Wie breit ist ein balancierter
Binärbaum der Tiefe 3?

A: 4



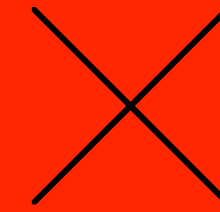
B: 6



C: 8



D: 16



Breite vs. Tiefe

$$b : \mathcal{T} \rightarrow \mathbb{N}_+$$

$$b[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 1 \text{ else } b\ t_1 + \dots + b\ t_n$$

$$d : \mathcal{T} \rightarrow \mathbb{N}$$

$$d[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 0 \text{ else } 1 + \max\{d\ t_1, \dots, d\ t_n\}$$

Proposition 10.7 (Breite versus Tiefe) $\forall t \in \mathcal{B}: \quad b\ t = 2^{d\ t}.$

Beweis Durch strukturelle Induktion über $t \in \mathcal{B}$. Wir unterscheiden zwei Fälle.

Sei $t = []$. Dann $b\ t = 1 = 2^{d\ t}$ gemäß der Definition von b und d .

Sei $t = [t_1, t_2]$. Dann gilt:

$$b\ t = b\ t_1 + b\ t_2$$

Definition b

$$= 2^{d\ t_1} + 2^{d\ t_2}$$

Induktion für t_1 und t_2

$$= 2 \cdot 2^{d\ t_1}$$

t balanciert, also $d\ t_1 = d\ t_2$

$$= 2^{1+d\ t_1}$$

$$= 2^{1+\max\{d\ t_1, d\ t_2\}}$$

t balanciert, also $d\ t_1 = d\ t_2$

$$= 2^{d\ t}$$

Definition d

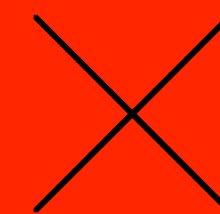


**Hat ein balancierter Binärbaum
mehr Blätter oder mehr innere Knoten?**

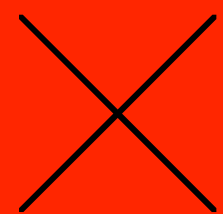
A: mehr Blätter



B: mehr innere Knoten



C: gleich viele



Größe vs. Tiefe

$$s : \mathcal{T} \rightarrow \mathbb{N}_+$$

$$s[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 1 \text{ else } 1 + s t_1 + \dots + s t_n$$

$$d : \mathcal{T} \rightarrow \mathbb{N}$$

$$d[t_1, \dots, t_n] = \text{if } n = 0 \text{ then } 0 \text{ else } 1 + \max\{d t_1, \dots, d t_n\}$$

Proposition 10.8 (Größe versus Tiefe) $\forall t \in \mathcal{B}: st = 2^{d t + 1} - 1$.

Beweis Durch strukturelle Induktion über $t \in \mathcal{B}$. Wir unterscheiden zwei Fälle.

Sei $t = []$. Dann $st = 1 = 2^{d t + 1} - 1$ gemäß der Definition von b und d .

Sei $t = [t_1, t_2]$. Dann gilt:

$$st = 1 + s t_1 + s t_2$$

Definition s

$$= 1 + 2^{d t_1 + 1} - 1 + 2^{d t_2 + 1} - 1$$

Induktion für t_1 und t_2

$$= 2 \cdot 2^{d t_1 + 1} - 1$$

t balanciert, also $d t_1 = d t_2$

$$= 2^{1 + d t_1 + 1} - 1$$

$$= 2^{1 + \max\{d t_1, d t_2\} + 1} - 1$$

t balanciert, also $d t_1 = d t_2$

$$= 2^{d t + 1} - 1$$

Definition d

www.prog1.saarland