A1.1.

Sorte: SORT Kunden

Diskriminator: FUN Kunde?: Kunden → bool

Selektoren:

FUN name: Kunden → denotation

FUN alter: Kunden → nat

FUN lieblingssorte: Kunden → Eissorte

FUN umsatz: Kunden → Betrag

Konstruktor:

FUN Kunde: denotation × nat × Eissorte
× Betrag → Kunden

Produkttyp, da nur ein Konstruktor

A1.2

grosskunde(name: denotation,
firma: denotation,
lieblingssorte: Eissorte,
umsatz: Betrag)

alle ausser alter, firma, da alter, firma nur für bestimmte Kunden definiert sind

A1.3:

FUN < : Betrag × Betrag → bool

```
DEF betrag(e1,c1) < betrag(e2,c2) ==
    e1 < e2 or (e1 = e2 and c1 < c2)

-- oder

DEF a < b == IF euro(a) = euro(b)
             THEN cent(a) < cent(b)
             ELSE euro(a) < euro(b)
             FI
```

A1.4 : FUN gute Kunden : seq[Kunden] -> seq[denotation]

```
DEF gutekunden(kunde(n,_,_,u) :: R) ==
    IF betrag(24,50) < u THEN n :: gutekunden(R)
    ELSE gutekunden(R) FI

DEF gutekunden(grosskunde(n,_,_,u) :: R) ==
    IF betrag(1000,0) < u THEN n :: gutekunden(R)
    ELSE gutekunden(R) FI

DEF gute kunden(<>) == <>
```

A2.1 FUN time : seq[auftrag] -> nat

DEF time(S) == reduce(+, 0)(map(bzeit)(S))

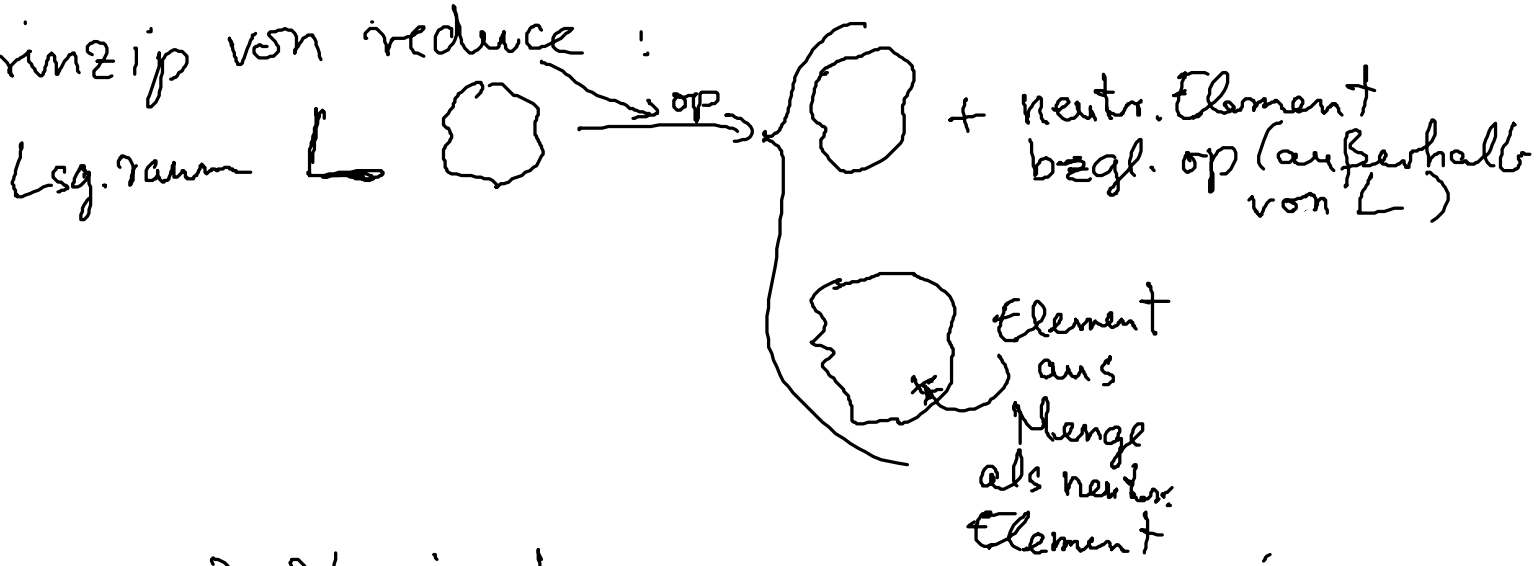
reduce(+, 0) : \\e, n . +(e, n)

map(bzeit)(S) : seq[nat]

oder

```
DEF time(S) == reduce(\\a,n. bzeit(a)+n, 0)(S)
```

Prinzip von reduce:



Wenn 2. Variante, muss man leere Liste als Pattern berücksichtigen

DEF time(a::R) == reduce(lla,n. bZeit(a)+n, a)(R)

falsch (a::R)

DEF time(<>) == 0

A2.7: FUN slowService:

seq[auftrag] x nat -> seq[auftrag]

DEF slowService(s, z) ==

filter(lla. bZeit(a) > z)(s)

-- oder FUN slowService: nat -> seq[auftrag] -> seq[auftrag]

DEF slowService(z) == filter(lla.bZeit(a) > z)

~~(s)~~ η-Reduktion ~~(s)~~

-- oder filter(bZeit(_) > z)

A2.3 FUN slowest : seq[Auftrag] x (anliegen -> bool)
→ nat

[1. Variante: "Es gibt in der Liste mind. ein Auftrag mit diesem Anliegen"]

DEF slowest(S, p) == LET l == filter(λa. p(art(a)))(S)

IN schalter(reduce(λa,b. aZeit(a) > bZeit(b) THEN a ELSE b FI, ft(l))(rt(l))
-- auch richtig (l)
: anliegen

[2. Variante: "nicht zwingend enthalten, 1 auftrag option verwenden"]

FUN slowest : ..... → option[nat]

DEF .... == LET l == .....

IN IF <>?(l) THEN nil
ELSE avail(*) FI

falsch: l = <>

auch möglich: nur reduce und filter-Fkt in " -Fkt integrieren

---

A3.1.2 Kalkül, Applikation, d.h. f(x)

(λ.....)

in Literatur auch als f x

(f x) ← Klausur-Variante

$(f)\,x$

in Klammern weglassen

$= (\lambda y.((x\,y)(\lambda x.x)))[z/y]\,[(x\,y)/x]$

$= (\lambda z.((x\,y)(\lambda x.x)[z/y])\,[\quad u \quad]$

~~$= (\lambda z.((x\,y)[z/y](\lambda x.x)[z/y]))\,[\quad u \quad]$~~

$= (\lambda z.((x\,z)(\lambda x.x)))\,[\quad u \quad]$

$= (\lambda z.((x\,z)(\lambda x.x))\quad [\quad])$

~~$= (\lambda z((x\,z)[(x\,y)/x](\lambda x.x)[(x\,y)/x]))$~~ Keine freie Var.

$= (\lambda z.((x\,y)\,z)(\lambda x.x)))$

A 4.1 :

$$A_{toSet}(n) = A_{toSet}(n-1) + A_{in?}(n-1) + const$$

$$A_{in?}(n) = A_{in?}(n-1) + c_2 \cdot n^0$$

$\rightarrow$ z. 1, $k=0$, $b=c_2$ $\Rightarrow A_{in?} \in O(n)$

$$A_{toSet}(n) = A_{toSet}(n-1) + b\,n^k - 1 + const$$

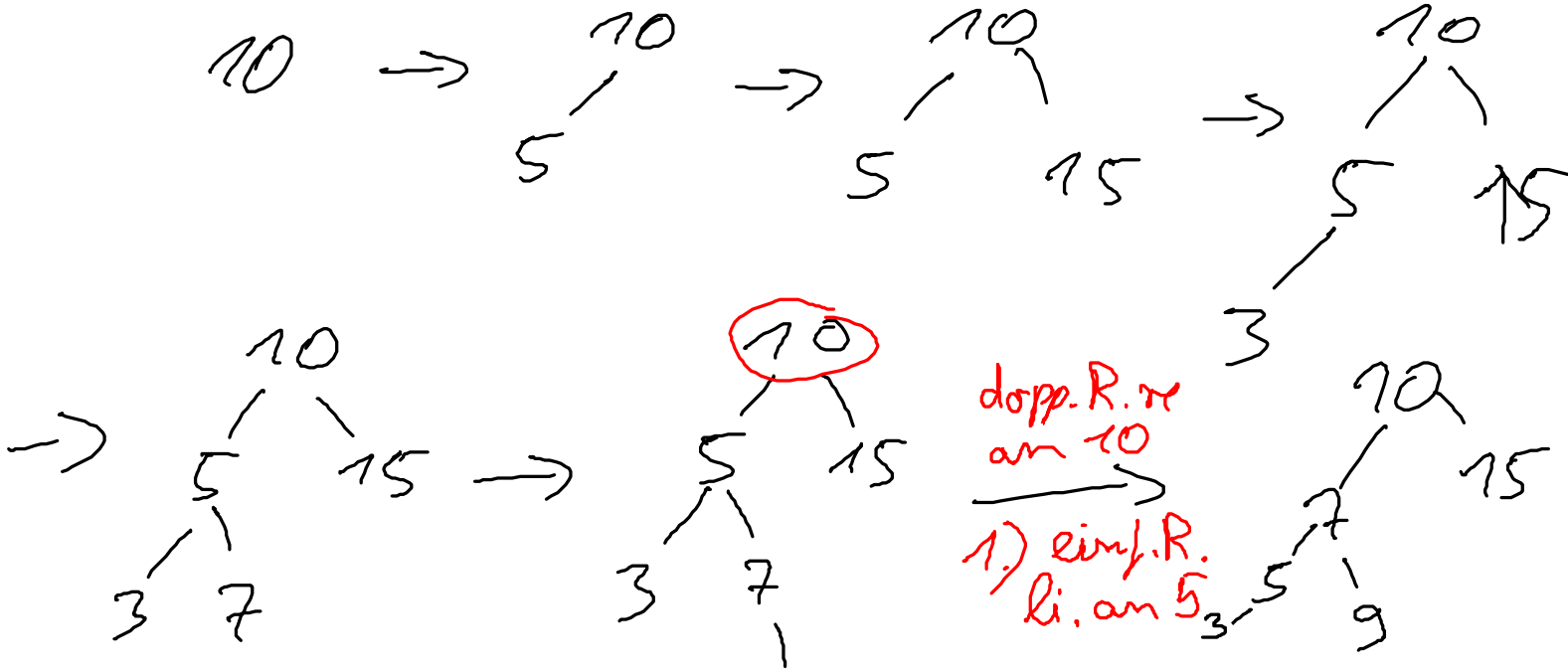
$k \geq 1$

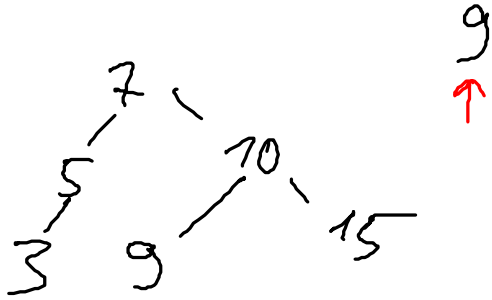
$\rightarrow$ 1. z. $b=1$, $k=1$ $\Rightarrow A_{toSet} \in O(n^{\geq 1})$

5.1 :- Ja, ist Wald
- Ja, ist linksvoll. Bin. Baum
↑ Heap aus VL, Skeew-Heap aus VL muss nicht linksvoll sein
- heapify passiert kein Einfügen, Löschen in einem Unterbaum
=> Nein (log. Aufwand)
- Nein

5.2 : a) ADOPIWMXY
b) POIDWAXMY
c) G.9, H.4
d) ADMOWXYPI

5.3 :





7.) ei. R. re. an 10

5.4) a) AVL, Bin Suchbaum und Höhen unterschied sich max um 1

b) Heap, linksvoller Bin. Baum und alle Knoten in Unterbäumen sin kleiner als akt. Knoten

c) Weder noch

6.1. Insertionsort

6.2. Quicksort