CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Prof. Bernd Finkbeiner, Ph.D.
Jana Hofmann, M.Sc.
Reactive Systems Group

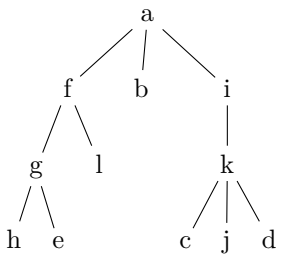
Programmierung 1 (WS 2020/21)
Aufgaben für die Übungsgruppe G (Lösungsvorschläge)

Hinweis: Diese Aufgaben wurden von den Tutoren für die Übungsgruppe erstellt. Sie sind für die Klausur weder relevant noch irrelevant. 🤔 markiert potentiell schwerere Aufgaben.

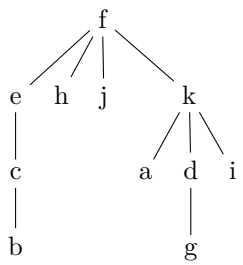
Bäume

Aufgabe TG.1 *(Projektionen)*
Geben Sie zu folgenden markierten Bäumen Prä- und Postprojektion an.

(a)



(b)



Lösungsvorschlag TG.1

(a) Präprojektion: `[a, f, g, h, e, l, b, i, k, c, j, d]`
Postprojektion: `[h, e, g, l, f, b, c, j, d, k, i, a]`

(b) Präprojektion: `[f, e, c, b, h, j, k, a, d, g, i]`
Postprojektion: `[b, c, e, h, j, a, g, d, i, k, f]`

Aufgabe TG.2 *(Markentransformation)*
Wir wollen nun, ähnlich zu Listen ein `mapltr : (`α $\rightarrow$ β`)` $\rightarrow$ α `ltr` $\rightarrow$ β `ltr` für markierte Bäume erstellen. Hierbei soll die Transformation auf alle Marken angewendet werden. Schreiben Sie `mapltr`.

Hinweis: `datatype` α `ltr = L of` α `*` α `ltr list`

Lösungsvorschlag TG.2

```
1 fun mapltr f (L(a,ls)) = L (f a, map (mapltr f) ls)
```

Aufgabe TG.3 *(Auf der Suche nach Marken)*

(a) Schreiben Sie eine Prozedur `prestl :` α `ltr` $\rightarrow$ `int` $\rightarrow$ α, die einen markierten Baum und eine Zahl n bekommt und die Marke des Knotens bestimmt, dessen Pränummer n ist. Falls es keinen solchen Knoten gibt, soll `Subscript` geworfen werden.

(b) Schreiben Sie eine Prozedur `prefind : 'α ltr → 'α → int option`, die einen markierten Baum und eine Marke x bekommt und die Pränummer des Knotens bestimmt, dessen Marke x ist. Falls es keinen solchen Knoten gibt, soll `NONE` zurückgegeben werden.

Lösungsvorschlag TG.3

(a)
```
fun prestl' (L(x, _)  :: _)  0 = x
| prestl' (L(x, lr) :: tr) n = prestl' (lr @ tr) (n - 1)
| prestl' _                _ = raise Subscript
fun prestl t = prestl' [t]
```

(b)
```
fun prefind' (L(x, lr) :: tr) a y = if x = y then SOME a else prefind' (lr @ tr) (a + 1)
  ↳ y
| prefind' _                 a y = NONE
fun prefind t = prefind' [t] 0
```

Aufgabe TG.4 *(Marken zählen)*
Schreiben Sie eine Prozedur `count : ('α * 'α → order) → 'α ltr → 'α → int`, die zählt, wie oft ein Element x als Marke in einem markierten Baum auftritt.

Lösungsvorschlag TG.4

```
fun count f (T(a, ls)) x =
    foldl (fn (v, rs) ⇒ rs + count f v x) (if f(x, a) = EQUAL then 1 else 0) ls
```

Aufgabe TG.5 *(Wahrheitsbaum)*
Schreiben Sie eine Prozedur `exists : ('α → bool) → 'α ltr → bool`, die bestimmt, ob im Baum eine Marke existiert, die eine Bedingung p erfüllt.

Lösungsvorschlag TG.5

```
fun exists p (L(x, ls)) = foldl (fn (y, a) ⇒ y orelse a) (p x) (map (exists p) ls)
```

Aufgabe TG.6 *(Markieren)*

(a) Schreiben Sie eine Prozedur `preltree : tree → int ltr`, die aus einem Baum einen markierten Baum derselben Gestalt zurückgibt, dessen Marken gerade den Pränummern der Knoten entsprechen.

Hinweis: Schreiben Sie zunächst eine Hilfsprozedur vom Typ `int → tree → int ltr`, die gegeben eine Zahl n einen derartig markierten Baum erstellt, bei dem die Pränummerierung bei n beginnen soll.

(b) Schreiben Sie eine Prozedur `postltree : tree → int ltr`, die aus einem Baum einen markierten Baum derselben Gestallt zurückgibt, dessen Marken gerade den Postnummern der Knoten entsprechen.
Hinweis: Schreiben Sie wieder zunächst eine Hilfsprozedur vom Typ `int → tree → int ltr`.

Lösungsvorschlag TG.6

(a)
```
fun preltree' n (T ts) =
    foldl (fn (t, (L(x, ls), n)) ⇒
        let
            val (l, n') = preltree' n t
        in
            (L(x, ls @ [l]), n')
        end
    ) (L(n, []), n + 1) ts
val preltree = preltree' 0
```

(b)

```
fun postltree' n (T ts) =
     foldl (fn (t, L(n, ls)) ⇒
         let
             val (L(n', ls')) = postltree' n t
         in
             L(n' + 1, ls @ [L(n', ls')])
         end
       ) (L(n, [])) ts
val postltree = postltree' 0
```

Aufgabe TG.7 *(Such Forest, such!)*
Ein sogenannter *binärer Suchbaum* ist eine besondere Art von markiertem Baum, in dem Zahlen gespeichert und effizient wiedergefunden werden können. Dabei hat jeder Knoten k in einem binären Suchbaum maximal zwei Unterbäume, die man als *linkes Kind* und *rechtes Kind* bezeichnet. Diese erfüllen folgende Eigenschaften:

- Die Marke des linken Kindes ist kleiner als die Marke von k.
- Die Marke des rechten Kindes ist größer als die Marke von k.

Da man bei unseren Bäumen im Falle eines einzelnen Kindes nicht unterscheiden kann, ob es sich um das rechte oder linke handelt, fordern wir, dass jeder Knoten genau zwei Unterbäume hat und erweitern den bekannten Datentypen wie folgt:

```
datatype searchtree = L of int * searchtree list | TNULL
```

Falls ein linkes oder rechtes Kind nicht existiert, wird ein Platzhalter als `TNULL` in die Liste eingefügt. Hat ein Knoten mit Marke 5 beispielsweise den Unterbaum `t` als rechtes Kind, wird der Knoten also dargestellt als `L(5, [TNULL, t])`. Eine Wurzel mit Marke 2 ohne Kinder hätte die Darstellung `L(2, [TNULL, TNULL])`.

(a) Schreiben Sie eine Prozedur `insert : searchtree → int → searchtree`, die eine neue Zahl gemäß der obigen Vorschrift in einen binären Suchbaum einfügt. Wenn die Zahl bereits enthalten ist, soll der Baum ungeändert zurückgegeben werden.

(b) Schreiben Sie eine Prozedur `contains : searchtree → int → bool`, die testet, ob eine Zahl in einem binären Suchbaum enthalten ist. Nutzen Sie die Suchbaumeigenschaft aus.

(c) Schreiben Sie Prozeduren `min : searchtree → int` und `max : searchtree → int`, die die größte bzw. kleinste Zahl, die im Baum gespeichert ist zurückgibt.

(d) Versuchen Sie eine Prozedur `delete : searchtree → int → searchtree` zu schreiben, die eine Zahl aus einem binären Suchbaum entfernt.

Hinweis: Überlegen Sie sich, wie Ihre Prozedur vorgehen muss, wenn der zu löschende Knoten keine Kinder, ein Kind oder zwei Kinder hat.

Lösungsvorschlag TG.7

(a)

```
exception InvalidTree
fun insert TNULL x = L (x, [TNULL, TNULL])
  | insert (L(k, [t1, t2])) x =
                        if x < k then L (k, [insert t1 x, t2])
                        else if x > k then L (k, [t1, insert t2 x])
                        else L(k, [t1, t2])
  | insert _ _ = raise InvalidTree
```

(b)

```
fun contains TNULL x  = false
  | contains (L(k, [t1, t2])) x = x = k orelse
                                  if x < k then contains t1 x else contains t2 x
  | contains _ _ = raise InvalidTree
```

(c)

```
fun min TNULL = raise Empty
  | min (L(k,[t,_])) = (min t handle Empty ⇒ k)
  | min _ = raise InvalidTree
```

```
4
5  fun max TNULL = raise Empty
6    | max (L(k,[_,t])) = (max t handle Empty ⇒ k)
7    | max _ = raise InvalidTree
```

(d)

```
1  fun delete (L(k, [t1, t2])) x =
2          if x < k then L (k, [delete t1 x, t2])
3          else if x > k then L (k, [t1, delete t2 x])
4          else (case (t1, t2)
5                of (TNULL, TNULL) ⇒ TNULL
6                 | (t, TNULL) ⇒ t
7                 | (TNULL, t) ⇒ t
8                 | (t1, t2) ⇒ let val m = min t2
9                                in L (m, [t1, delete t2 m]) end)
10   | delete TNULL _ = raise Empty
11   | delete _ _ = raise InvalidTree
```

Aufgabe TG.8 *(Mengen als gerichtete Bäume)*

(a) Stellen Sie die folgenden Mengen als gerichtete Bäume dar:

(i) $\emptyset$

(ii) $\{\{\{\}\}, \{\}\}$

(iii) $\{\{\}\}$

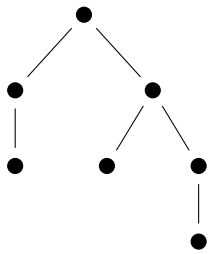
(iv) $\{\{\{\}\}, \{\{\}\}, \{\}\}$

(v) $\{\{\{\}\}, \{\{\}, \{\{\}, \{\{\}\}\}\}\}$

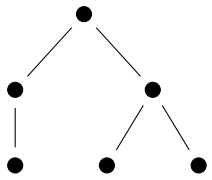
(b) Welcher dieser Bäume ist gerichtet?

(i)

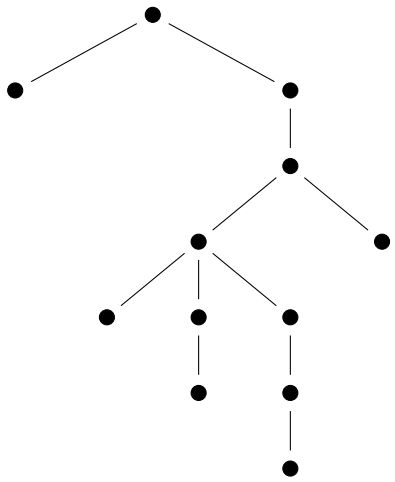
(ii)



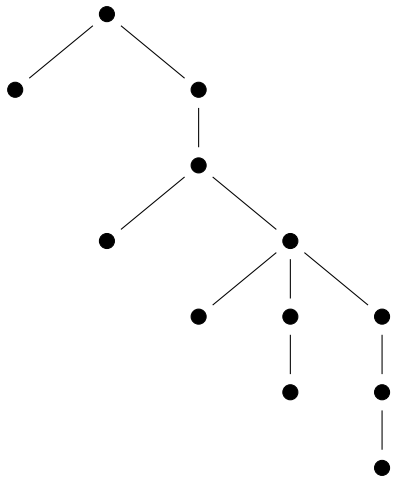
(iii)



(iv)



(v)



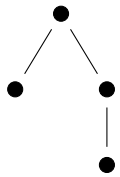
(c) Stellen Sie die gerichteten Bäume aus b) als Mengen dar.

Lösungsvorschlag TG.8

(a)

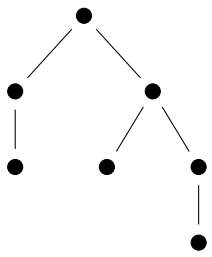
(i)

(ii)

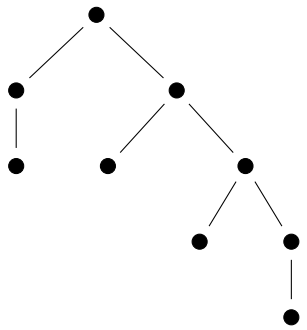


(iii)

(iv)



(v)



(b) Baum i, ii, v sind gerichtet

(c) (i) $\{\{\}\}$

(ii) $\{\{\{\}\}, \{\{\}, \{\{\}\}\}\}$

(v) $\{\{\}, \{\{\{\}, \{\{\}, \{\{\}\}, \{\{\{\}\}\}\}\}\}\}$

Aufgabe TG.9 *(Gerichtete Bäume)*
Schreiben Sie eine Prozedur `gerichtet : tree → bool`, die überprüft, ob ein Baum gerichtet ist. Verwenden Sie `strictlySorted : (α * α → order) → α list → bool`, welche überprüft, ob eine Liste gemäß einer Vergleichsprozedur *strikt* sortiert ist, `List.all : (α → bool) → α list → bool`, sowie `compareTree : tree * tree → order`, welche die lexikalische Baumordnung implementiert.

Lösungsvorschlag TG.9

```
1 fun gerichtet (T ts) = strictlySorted compareTree ts andalso List.all gerichtet ts
```

Mengenlehre

Aufgabe TG.10 *(Mengen - Warmup)*

(a) Geben Sie alle Teilmengen der folgenden Mengen an:

(i) $\{\{\{\{\}\}\},\{\{\}\},\{\}\}$
(ii) $\{\}$
(iii) $\{0,1\}$
(iv) $\{\{\}\}$

(b) Welche der folgenden Mengen sind gleich?

(i) $\emptyset$
(ii) $\{\emptyset\}$
(iii) $\{\}$
(iv) $\{\{\}\}$
(v) $\{\emptyset,\emptyset\}$
(vi) $\{\emptyset,\{\emptyset\}\}$

Lösungsvorschlag TG.10

(a)

(i) $\{\{\{\{\}\}\},\{\{\}\},\{\}\}$, $\{\{\{\}\},\{\}\}$, $\{\{\{\{\}\}\},\{\}\}$, $\{\{\{\{\}\}\},\{\{\}\}\}$, $\{\{\}\}$, $\{\{\{\}\}\}$, $\{\{\{\{\}\}\}\}$, $\{\}$
(ii) $\{\}$
(iii) $\{0,1\},\{0\},\{1\},\{\}$
(iv) $\{\{\}\},\{\}$

(b) i = iii, v = ii = iv

Aufgabe TG.11 *(Mengenoperatoren - Basics)*
Betrachten Sie folgende Mengen:

- $A = \{1,2,3\}$
- $B = \{4,5,6\}$
- $C = \{2\}$
- $D = \emptyset$

Geben Sie nun folgende Mengen an:

(a) $A \cup B$
(b) $B \cup D$
(c) $A \cup C$
(d) $A \cap C$
(e) $B \cap D$
(f) $A \cap B$
(g) $A \setminus C$
(h) $A \setminus B$
(i) $A \setminus A$
(j) $D \setminus C$
(k) $\mathcal{P}(A)$

Lösungsvorschlag TG.11

(a) $A \cup B = \{1,2,3,4,5,6\}$
(b) $B \cup D = \{4,5,6\}$
(c) $A \cup C = \{1,2,3\}$
(d) $A \cap C = \{2\}$
(e) $B \cap D = \emptyset$
(f) $A \cap B = \emptyset$
(g) $A \setminus C = \{1,3\}$
(h) $A \setminus B = \{1,2,3\}$
(i) $A \setminus A = \emptyset$
(j) $D \setminus C = \emptyset$
(k) $\mathcal{P}(A) = \{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$

Aufgabe TG.12 *(Dom, Ran und Ver)*
Betrachten Sie folgende Relation:

$$R = \{(1,4),(2,2),(2,3),(3,2),(4,2),(4,3),(5,5),(5,6)\}$$

Bestimmen Sie den Definitions- und Wertebereich, sowie die Knotenmenge von R.

Lösungsvorschlag TG.12

$$\begin{aligned} \mathit{Dom}\, R &= \{1,2,3,4,5\} \\ \mathit{Ran}\, R &= \{2,3,4,5,6\} \\ \mathit{Ver}\, R &= \{1,2,3,4,5,6\} \end{aligned}$$

Aufgabe TG.13 *(Komposition)*
Betrachten Sie folgende Relationen:

$$R = \{(1,1),(1,3),(3,2),(4,2)\} \qquad R' = \{(1,4),(2,2)\}$$

Bestimmen Sie die Komposition $R \circ R'$ der beiden Relationen.

Lösungsvorschlag TG.13
$R \circ R' = \{(1,4),(3,2),(4,2)\}$

Aufgabe TG.14 *(Total funktional)*
Betrachten Sie folgende Relation auf der Menge $\{1,2,3,4\}$:

$$R = \{(1,1),(1,3),(3,2),(4,2)\}$$

Ist sie total? Surjektiv? Funktional? Injektiv? Fügen Sie ggf. Kanten hinzu oder löschen Sie ggf. Kanten, damit die jeweilige Eigenschaft erfüllt ist. Ist dies immer eindeutig möglich oder gibt es mehrere Lösungen?

Lösungsvorschlag TG.14
Die Relation ist weder funktional, injektiv noch total noch surjektiv. Damit sie funktional ist, muss von jedem Element genau eine Kante ausgehen. Hierfür müsste eine der beiden von 1 ausgehenden Kanten, z.B. $(1,1)$, entfernt werden. Damit sie injektiv ist, müsste eine der Kanten zu 2 entfernt werden, beispielsweise $(4,2)$. Damit sie total ist, müsste eine Kante von 2 ausgehend hinzugefügt werden, beispielsweise $(2,1)$. Damit sie surjektiv ist, müsste eine Kante zu 4 hinzugefügt werden, beispielsweise $(3,4)$.

Aufgabe TG.15 *(reflexiv, transitiv, antisymmetrisch, linear)*
Betrachten Sie folgende Relation auf der Menge $\{1,2,3,4\}$:

$$R = \{(1,2),(1,4),(2,2),(2,4),(3,1),(3,3)\}$$

Ist sie reflexiv? Transitiv? Antisymmetrisch? Linear? Fügen Sie ggf. Kanten hinzu oder löschen Sie ggf. Kanten, damit die jeweilige Eigenschaft erfüllt ist. Ist dies immer eindeutig möglich oder gibt es mehrere Lösungen?

Lösungsvorschlag TG.15
Die Relation ist antisymmetrisch, aber weder reflexiv noch transitiv noch linear. Damit sie reflexiv ist, müssten die Kanten (1,1) und (4,4) hinzugefügt werden. Damit sie transitiv ist, müsste die Kante (3,2) und (3,4) hinzugefügt werden – genau diese machen die Relation auch linear.

Aufgabe TG.16 *(Im Graphen-Bootcamp)*
Wir können Graphen ganz einfach in SML darstellen:

```
type vertex = int
type edge = vertex * vertex
type graph = vertex list * edge list
```

(a) Schreiben Sie eine Prozedur `check : graph → bool`, die prüft, ob ein Graph gültig ist, also alle seine Kanten tatsächlich von Knoten ausgehen und in Knoten enden, die in der `vertex list` stehen.

Im Folgenden können Sie davon ausgehen, dass jeder Graph gültig ist.

(b) Schreiben Sie eine Prozedur `isSEdge : graph → (vertex * vertex) → bool`, die prüft, ob es zwischen zwei Knoten eine Kante in der angegebenen Richtung gibt.

(c) Schreiben Sie eine Prozedur `isWEdge : graph → (vertex * vertex) → bool`, die prüft, ob zwei Knoten benachbart sind.

(d) Schreiben Sie eine Prozedur `isPath : graph → vertex list → bool`, die prüft, ob eine Liste von Knoten einen gültigen Pfad darstellt.

(e) Schreiben Sie die Prozeduren `isTSEdge : graph → (vertex * vertex) → bool` und `isTWEdge : graph → (vertex * vertex) → bool`, die prüfen, ob es zwischen zwei gegebenen Knoten einen Pfad gibt bzw. es einen Pfad im symmetrischen Abschluss gibt. 🤔

(f) Schreiben Sie eine Prozedur `isRoot : graph → vertex → bool`, die prüft, ob ein Knoten eine Wurzel ist.

(g) Schreiben Sie die Prozeduren `isSConnected : graph → bool` und `isWConnected : graph → bool`, die prüfen, ob ein Graph (stark) zusammenhängend ist. 🤔

(h) Schreiben Sie eine Prozedur `isTreeLike : graph → graph`, die prüft, ob ein Graph baumartig ist.

Lösungsvorschlag TG.16

```
type vertex = int
type edge = vertex * vertex
type graph = vertex list * edge list

 (* checks if x is in xs *)
fun isElement x  nil    = false
  | isElement x (y::yr) = x = y orelse isElement x yr

 (* check : graph → bool *)
fun check (vl,el) = List.all (fn (a,b) ⇒ isElement a vl
                                         andalso isElement b vl)
                             el

 (* isSEdge : graph → (vertex * vertex) → bool *)
fun isSEdge (vl,el) e = isElement e el

 (* isWEdge : graph → (vertex * vertex) → bool *)
fun isWEdge g (v1,v2) = isSEdge g (v1,v2) orelse isSEdge g (v2,v1)

 (* isPath : graph → vertex list → bool *)
fun isPath g  nil       = false
  | isPath g  [_]       = true
  | isPath g (x::y::xr) = isSEdge g (x,y) andalso isPath g (y::xr)

 (* generates all possible combinations of vertices *)
fun genMaxEdges (vl,_) = List.concat (map (fn x ⇒ map (fn y ⇒ (x,y)) vl) vl)

 (* iterates until there is no change *)
fun fixpoint f s = let val next = f s
                   in  if s = next then s else fixpoint f next end

 (* extends a list with all elements a generator function created *)
fun extend f l = foldl (fn (x,s) ⇒ List.filter (fn y ⇒ not (isElement y l))
                                               (f x) @ s)
                       l l

 (* transitive closure *)
fun tClosure (vl,el) = let fun extender (a,b) =
                               List.map (fn (_,c) ⇒ (a,c))
                                        (List.filter (fn (x,y) ⇒ x = b) el)
                           val tel = fixpoint (extend extender) el
                       in (vl, tel) end

 (* symmetric closure *)
fun sClosure (vl,el) = (vl, extend (fn (a,b) ⇒ [(b,a)]) el)

 (* combines two closures *)
fun cc c1 c2 = fixpoint (c1 o c2)

(* transitive symmetric closure *)
fun tsClosure g = cc tClosure sClosure g

 (* isTSEdge : graph → (vertex * vertex) → bool *)
fun isTSEdge g e = case tClosure g of
```

```
                    (vl,el) ⇒ isElement e el

 (* isTWEdge : graph → (vertex * vertex) → bool *)
fun isTWEdge g e = case tsClosure g of
                    (vl,el) ⇒ isElement e el

 (* isRoot : graph → vertex → bool *)
fun isRoot (vl,el) r = List.all (fn x ⇒ isTSEdge (vl,el) (r,x)) vl

 (* isSConnected : graph → bool *)
fun isSConnected g = case tClosure g of (_,el) ⇒ el = genMaxEdges g

 (* isWConnected : graph → bool *)
fun isWConnected g = isSConnected (sClosure g)

 (* count predecessors *)
fun countPredec (vl,el) v = List.length (List.filter (fn (a,b) ⇒ b = v) el)

 (* isTreeLike : graph → graph *)
fun isTreeLike (vl,el)
   = List.exists (fn x ⇒ isRoot (vl,el) x
                  andalso countPredec (vl,el) x = 0
                  andalso List.all (fn y ⇒ x = y
                                      orelse countPredec (vl,el) y = 1)
                                   vl)
                 vl
```

Aufgabe TG.17
Sei ein Graph $G = (V, E)$ gegeben:

$$V = \{1, 2, 3, 4, 5, 6\}$$
$$E = \{(1, 5), (2, 1), (2, 3), (2, 4), (3, 5), (6, 2), (6, 3)\}$$

(a) Zeichnen Sie diesen Graphen ohne überkreuzende Kanten.

(b) Welche Tiefe hat der Graph?

(c) Welche Quellen, Senken und Wurzeln hat der Graph?

(d) Ist der Graph zyklisch? Wenn ja, geben Sie einen Zyklus an.

(e) Geben Sie einen einfachen Pfad maximaler Länge an.

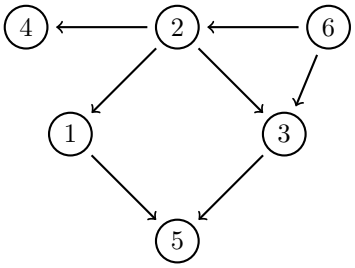
(f) Geben Sie den vom Knoten 2 aus erreichbaren Teilgraphen an.

(g) Ist der Graph zusammenhängend? Stark zusammenhängend?

(h) Ist der Graph baumartig?

Lösungsvorschlag TG.17

(a)



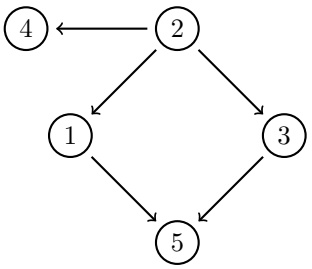
(b) Tiefe: 3

(c) Quelle : 6; Senken : 4,5; Wurzel: 6

(d) Nein

(e) $\langle 6, 2, 3, 5 \rangle$

(f)



(g) zusammenhängend; nicht stark zusammenhängend

(h) nein

Aufgabe TG.18 *(Es Russellt)*
Betrachten Sie folgende Fragen. Können Sie sie beantworten? Falls Nein, warum nicht?

- Ein Barbier rasiert alle Männer des Ortes, die sich nicht selbst rasieren, und nur diese. Rasiert er sich selbst?
- Alle Bürgermeister dürfen nicht in ihrer eigenen Stadt leben, sondern müssen in die eigens dafür eingerichtete Bürgermeister-Stadt Bümstädt ziehen. Wo lebt der Bürgermeister von Bümstädt?

Lösungsvorschlag TG.18
Nein, diese Fragen sind nicht sinnvoll zu beantworten. Es handelt sich dabei nur um anschauliche Beispiele der Russellschen Antinomie bzw. derer Varianten.

- Angenommen der Barbier B rasiert sich selbst, dann darf er sich per Definition nicht selbst rasieren.
  Angenommen, er rasiert sich nicht selbst, dann muss er sich per Definition selbst rasieren.
  Daraus folgt: B rasiert B $\Leftrightarrow$ B rasiert B nicht. Ein Widerspruch.
- Ein paradoxer Befehl:
  Angenommen der Bürgermeister von Bümstädt B lebt in Bümstädt, dann muss er dort wegziehen, da er Bürgermeister dieser Stadt ist.
  Angenommen B lebt nicht in Bümstädt, dann muss er dort hinziehen, da er Bürgermeister einer Stadt ist.
  Daraus folgt (im weitesten Sinne): B lebt in Bümstädt $\Leftrightarrow$ B lebt nicht in Bümstädt. Ein Widerspruch.

Mathematische Prozeduren

Aufgabe TG.19 *(Aufwärmen...)*
Sei die folgende mathematische Prozedur gegeben:

$$
\begin{aligned}
&ggt : \mathbb{N}_+^2 \to \mathbb{N}_+ \\
&ggt(x, x) = x \\
&ggt(x, y) = ggt(x - y, y) && x > y \\
&ggt(x, y) = ggt(x, y - x) && x < y
\end{aligned}
$$

(a) Bleiben die Wohlgeformtheitsbedingungen für die definierenden Gleichungen gültig, wenn man

- den Ergebnisbereich zu $\mathbb{Z}$ verändert?
- den Argumentbereich zu $\mathbb{N} \times \mathbb{N}$ verändert?
- den Argumentbereich zu $\mathbb{N} \times \mathbb{N}$ und den Ergebnisbereich zu $\mathbb{Z}$ verändert?

(b) Geben Sie die Anwendungsgleichung für $ggt(24, 16)$ an.

(c) Geben Sie die Rekursionsfolge für $ggt(30, 24)$ an.

Lösungsvorschlag TG.19

(a)
- Ja
- Nein, da für $ggt(0, 0)$ 0 ausgegeben werden müsste, der Ergebnisbereich $\mathbb{N}_+$ allerdings die 0 nicht enthält.
- Ja

(b) $ggt(24, 16) = ggt(8, 16)$

(c) $(30, 24) \to (6, 24) \to (6, 18) \to (6, 12) \to (6, 6)$

Aufgabe TG.20 *(Wohlgeformtheit I)*
Notieren Sie zunächst die vier Wohlgeformtheitsbedingungen. Finden Sie dann für jede der Bedingungen eine Prozedur, die nur gegen diese verstößt. Ist dies möglich?

Lösungsvorschlag TG.20
Im Folgenden ist zu jeder Wohlgeformtheitsbedingung eine Prozedur angegeben, die nur gegen diese verstößt.

(a) *Rekursive Anwendungen der Prozedur erfolgen nur auf Elemente des Argumentbereichs der Prozedur.*

$$
\begin{aligned}
&p : \mathbb{N} \to \mathbb{N} \\
&p\ n = \text{if } true \text{ then } 42 \text{ else } p(n - 1)
\end{aligned}
$$

(b) *Funktionen werden nur auf Elemente ihres Definitionsbereichs angewendet.*

$$
\begin{aligned}
&f \in \mathbb{N}_+ \to \mathbb{N} \\
&f\ n = 42
\end{aligned}
$$

$$
\begin{aligned}
&p : \mathbb{N} \to \mathbb{N} \\
&p\ n = \text{if } n = 0 \text{ then } f(n) \text{ else } p(n - 1)
\end{aligned}
$$

(c) *Es werden nur Ergebnisse im Ergebnisbereich der Prozedur geliefert.*

$$
\begin{aligned}
&p : \mathbb{N} \to \mathbb{N} \\
&p\ n = -1
\end{aligned}
$$

(d) *Die definierenden Gleichungen sind disjunkt und erschöpfend.*

$$p : \mathbb{N} \to \mathbb{N}$$
$$p\ 0 = 5$$

Aufgabe TG.21 *(SML $\mapsto$ λ)*
Geben Sie die von den folgenden Prozeduren berechneten Funktionen in λ-Notation an.

(a) `fun f x = x * x`

(b) `fun g (x:int) (y:int) = y`

(c) `fun h (x, y) = 2 * x + 1 - y`

(d) `fun i (x, y) = 2.0 * x + 1.0 - y`

(e) `val j = fn x => if x then 3 else 4`

(f) `val k =fn (a, b)=>if a >b then a else b`

Lösungsvorschlag TG.21

(a) $\lambda x \in \mathbb{Z}.\ x^2$

(b) $\lambda x \in \mathbb{Z}.\ \lambda y \in \mathbb{Z}.\ y$

(c) $\lambda(x, y) \in \mathbb{Z}^2.\ 2x + 1 - y$

(d) $\lambda(x, y) \in \mathbb{R}^2.\ 2x + 1 - y$

(e) $\lambda x \in \mathbb{B}.$ if x then 3 else 4

(f) $\lambda(x, y) \in \mathbb{N}^2.$ if $x > y$ then x else y

Aufgabe TG.22 *($\lambda \mapsto$ SML)*
Geben Sie geschlossene Abstraktionen an, die die folgenden Funktionen berechnen.

(a) $\lambda x \in \mathbb{N}.\ \lambda y \in \mathbb{N}.\ x$

(b) $\lambda x \in \mathbb{N}.\ x$

(c) $\lambda x \in \mathbb{B}.\ \neg x$

(d) $\lambda(x, y) \in \mathbb{B}^2.\ x \wedge y$

(e) $\lambda(x, y) \in \mathbb{N}^2.\ x + y$

(f) $\lambda(x, y) \in \mathbb{R}^2.\ x * y$

Lösungsvorschlag TG.22

(a) `fn x => (fn y => x)`

(b) `fn x => x`

(c) `fn x => not x`

(d) `fn (x, y) => x andalso y`

(e) `fn (x, y) => x + y`

(f) `fn (x, y) => x * y`

Aufgabe TG.23 *(Lambdas)*
Welche der folgenden Funktionen sind gleich?

(a) $\lambda x \in \mathbb{N}.\ x$

(b) $\lambda x \in \mathbb{N}.\ x + x$

(c) $\lambda x \in \mathbb{N}.\ (2 \cdot x)$ div 2

(d) $\lambda x \in \mathbb{N}.\ (x$ div $2) \cdot 2$

(e) $\lambda f \in \mathbb{N} \to \mathbb{N}.\ f\ 42$

(f) $\lambda(x, y) \in \mathbb{N}^2.$ if $y \leq x$ then $x - y$ else 0

(g) $\lambda x \in \mathbb{N}.\ 2 \cdot x$

(h) $\{(x, x) \in \mathbb{N}^2\}$

(i) $f(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 1 + f(n-1), & \text{falls } n > 0 \end{cases}$

(j) $f(n) = \begin{cases} 0, & \text{falls } n = 0 \\ 2 + f(n-1), & \text{falls } n > 0 \end{cases}$

(k) $\{(x, y) \in \mathbb{N}^2 \mid y = x + x\}$

(l) $\lambda(x, y) \in \mathbb{N}^2.\ \max\{x - y, 0\}$

(m) $\{(f, f\ 42) \mid f \in \mathbb{N} \to \mathbb{N}\}$

Lösungsvorschlag TG.23

- $(a) = (c) = (h) = (i)$
- $(b) = (g) = (j) = (k)$
- $(f) = (l)$
- $(e) = (m)$

Rekursion

Aufgabe TG.24 *(relativ rekursive Funktion)*
Geben Sie die Rekursionsrelation und die Rekursionsfunktion folgender Prozeduren an:

(a) $a : \mathbb{Z} \to \mathbb{Z}$
$a(n) = a(n - 2)$

(b) $b : \mathbb{N} \to \mathbb{N}$
$b(n) = b(n) + b(n)$

(c) $c : \mathbb{Z} \to \mathbb{N}$
$c(n) = c(n\ mod\ 2) \qquad n < 0$
$c(n) = n \qquad n \geq 0$

(d) $d : \mathbb{Z} \to \mathbb{N}$
$d(n) = d(n + 10) + d(n \operatorname{div} 10) \qquad n \geq 5$
$d(n) = n \qquad n < 5$

(e) $e : \mathbb{N} \to \mathbb{Z}$
$e(n) = e(n^2) - e(n^2) + e(n^2) \qquad n\ mod\ 2 = 0$
$e(n) = 3 \cdot n - 2 \cdot e(n) \qquad n\ mod\ 2 \neq 0$

(f) $f : \mathbb{Z} \times \mathbb{N} \to \mathbb{Z}$
$f(z, n) = f(z - 1, n) + f(z, n^2) \qquad z < 0$
$f(z, n) = f(n, z) \qquad z \geq 0$

Lösungsvorschlag TG.24

	Rekursionsrelation	Rekursionsfunktion
(a)	$\{(x, x-2) \mid x \in \mathbb{Z}\}$	$\lambda n \in \mathbb{N}.\langle n - 2\rangle$
(b)	$\{(x, x) \mid x \in \mathbb{N}\}$	$\lambda n \in \mathbb{N}.\langle n, n\rangle$
(c)	$\{(x, x\ mod\ 2) \mid x \in \mathbb{Z} \wedge x < 0\}$	$\lambda n \in \mathbb{Z}$.if $n < 0$ then $\langle n\ mod\ 2\rangle$ else $\langle\rangle$
(d)	$\{(x, x+10) \mid x \in \mathbb{Z} \wedge x \geq 5\}$ $\cup \{(x, x \operatorname{div} 10) \mid x \in \mathbb{Z}\ \wedge\ x \geq 5\}$	$\lambda n \in \mathbb{Z}$.if $n \geq 5$ then $\langle n + 10, n \operatorname{div} 10\rangle$ else $\langle\rangle$
(e)	$\{(x, x^2) \mid x \in \mathbb{N} \wedge x\ mod\ 2 = 0\}$ $\cup \{(x, x) \mid x \in \mathbb{N} \wedge x\ mod\ 2 \neq 0\}$	$\lambda n \in \mathbb{N}$.if $n\ mod\ 2 = 0$ then $\langle n^2, n^2, n^2\rangle$ else $\langle n\rangle$
(f)	$\{((x, y), (x - 1, y)) \mid x \in \mathbb{Z} \wedge y \in \mathbb{N} \wedge x < 0\}$ $\cup \{((x, y), (x, y^2)) \mid x \in \mathbb{Z} \wedge y \in \mathbb{N} \wedge x < 0\}$ $\cup \{((x, y), (y, x)) \mid x \in \mathbb{Z} \wedge y \in \mathbb{N} \wedge x \geq 0\}$	$\lambda(z, n) \in \mathbb{Z} \times \mathbb{N}$.if $z < 0$ then $\langle (z - 1, n), (z, n^2)\rangle$ else $\langle (n, z)\rangle$

Aufgabe TG.25 *(Rekursionsbäume à la Fibonacci)*
Sei die folgende mathematische Prozedur zur Berechnung der Fibonacci-Zahlen gegeben:

$$\begin{aligned} &\mathit{fib} : \mathbb{N} \to \mathbb{N} \\ &\mathit{fib}\ n = \text{if } n < 2 \text{ then } n \text{ else } \mathit{fib}(n - 1) + \mathit{fib}(n - 2) \end{aligned}$$

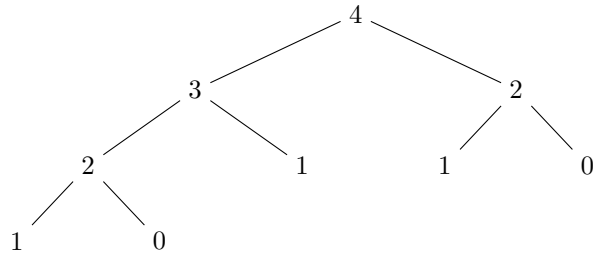
(a) Zeichnen Sie den Rekursionsbaum für das Argument 4.

(b) Gibt es eine Prozedur (ggf. mit auf Tupel abgeändertem Argument- und Ergebnisbereich), die ebenfalls die Fibonacci-Zahlen berechnet und einen linearen Rekursionsbaum hat? Geben Sie auch den Rekursionsbaum an.

(c) Welche der beiden Varianten hat insbesondere bei großen Argumenten die kleinere Laufzeit?

Lösungsvorschlag TG.25

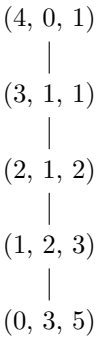
(a)



(b) Wir implementieren den iterativen Berechnungsansatz und ändern dafür Argument- und Wertebereich auf Tripel ab. Die folgende mathematische Prozedur gibt für $(n, 0, 1)$ in der zweiten Komponente die n-te Fibonacci-Zahl zurück:

$$\begin{aligned} \mathit{fib}' &: \mathbb{N}^3 \to \mathbb{N}^3 \\ \mathit{fib}'\ (0, a, b) &= (0, a, b) \\ \mathit{fib}'\ (n, a, b) &= \mathit{fib}'(n-1, b, a+b) \quad n > 0 \end{aligned}$$

Der zugehörige Rekursionsbaum sieht folgendermaßen aus:



Unsere Definition von mathematischen Prozeduren unterstützt es leider nicht direkt, Hilfsprozeduren zu verwenden.

(c) Wenn die Nebenkosten (also die Laufzeit, die für einen Rekursionsschritt benötigt werden) konstant sind, dann ist die Laufzeit insgesamt proportional zur Größe des Rekursionsbaums. Damit ist intuitiv klar, dass ein binärer Rekursionsbaum bei hinreichend großen Argumenten zu größerer Laufzeit als ein linearer Rekursionsbaum führt.

Wir können diese Überlegung noch etwas formalisieren: Der Rekursionsbaum von fib hat mindestens eine Tiefe von $\left\lfloor \frac{n}{2} \right\rfloor$ und auf jeder der ersten $k \leq \left\lfloor \frac{n+1}{2} \right\rfloor$ Ebenen 2^{k-1} Knoten. Damit hat der Rekursionsbaum mindestens $2^{\lfloor n+1/2 \rfloor} - 1$ Knoten. Der Rekursionsbaum von fib' hat offensichtlich $n+1$ Knoten. Mit dem Satz von l'Hospital (der in MfI 1 noch behandelt wird), lässt sich relativ einfach zeigen, dass $2^{\lfloor n+1/2 \rfloor} - 1$ echt stärker wächst als $n+1$ und damit ab einem gewissen n die Laufzeit von fib' echt kleiner als die von fib ist.