Aufgabe 1 Dynamische Semantik (__ /20 Punkte)

a) Geben Sie jeweils an, zu welchem Wert die folgenden Ausdrücke auswerten (ohne Rechnung, Endergebnis genügt):

__ /10

1. `let x = 10 in let y = 3 in x % y`

   1

2. `if 1 * 2 < 1 + 2 then 3 + 4 else 3 * 4`

   7

3. `(fun x -> 3 * x + 4) 5`

   19

4. `let x = 4 in (let x = 3 * x in 2 * x) + x`

   28

5. `let a = 2 in let f (x: Nat): Nat = a * x in f 3`

   6

b) Werten Sie den Ausdruck `f x` bezüglich der Umgebung

__ /10

$$\delta := \{\mathtt{x} \mapsto 2, \mathtt{a} \mapsto 3, \mathtt{f} \mapsto \langle \{\mathtt{a} \mapsto 4\}, \mathtt{x}, \mathtt{a}\ \mathtt{+}\ \mathtt{x} \rangle\}$$

aus. Geben Sie einen vollständigen Beweisbaum auf Grundlage der Auswertungsregeln aus der Vorlesung an.
Tipp: Legen Sie das Blatt quer und zeigen Sie $\delta \vdash \mathtt{f\ x} \Downarrow \ldots$

definiere aus Platzgründen $\delta_2 := \{\mathtt{a} \mapsto 4, \mathtt{x} \mapsto 2\}$

$$\dfrac{\dfrac{}{\delta \vdash \mathtt{f} \Downarrow \langle \{\mathtt{a} \mapsto 4\}, \mathtt{x}, \mathtt{a}\ \mathtt{+}\ \mathtt{x} \rangle} \quad \dfrac{}{\delta \vdash \mathtt{x} \Downarrow 2} \quad \dfrac{\dfrac{}{\delta_2 \vdash \mathtt{a} \Downarrow 4} \quad \dfrac{}{\delta_2 \vdash \mathtt{x} \Downarrow 2}}{\delta_2 \vdash \mathtt{a}\ \mathtt{+}\ \mathtt{x} \Downarrow 6}}{\delta \vdash \mathtt{f\ x} \Downarrow 6}$$

alternative Schreibweise:

$$\dfrac{\dfrac{}{\delta(\mathtt{f}) = \langle \{\mathtt{a} \mapsto 4\}, \mathtt{x}, \mathtt{a}\ \mathtt{+}\ \mathtt{x} \rangle} \quad \dfrac{}{\delta \vdash \mathtt{x} \Downarrow 2} \quad \dfrac{\dfrac{}{\delta_2 \vdash \mathtt{a} \Downarrow 4} \quad \dfrac{}{\delta_2 \vdash \mathtt{x} \Downarrow 2}}{\delta_2 \vdash \mathtt{a}\ \mathtt{+}\ \mathtt{x} \Downarrow 6}}{\delta \vdash \mathtt{f\ x} \Downarrow 6}$$

Aufgabe 2 Statische Semantik (__ /20 Punkte)

Füllen Sie die Lücken, sodass sich **gültige Aussagen der Statischen Semantik** ergeben. Alle vorkommenden Zahlen sind vom Typ `Nat`.

Hinweis: In einigen Teilaufgaben gibt es mehrere richtige Lösungen. Die angegebene Lösung ist nur ein Beispiel.

a) $\emptyset \vdash$ `4711 :` _____

$\emptyset \vdash$ `4711 : Nat`

b) $\emptyset \vdash$ _____ `: Nat`

$\emptyset \vdash$ `4711 : Nat`

c) $\emptyset \vdash$ **`if true then`** _____ **`else`** _____ : Nat

$\emptyset \vdash$ **`if true then`** `1` **`else`** `2 : Nat`

d) $\{$ `f` $\mapsto$ `Bool` $\rightarrow$ `Nat` $\}$ $\vdash$ `f` _____ : _____

$\{$ `f` $\mapsto$ `Bool` $\rightarrow$ `Nat` $\}$ $\vdash$ `f` **`true`** `: Nat`

e) $\{$ `x` $\mapsto$ _____ $\}$ $\vdash$ `x % 5 :` _____

$\{$ `x` $\mapsto$ `Nat` $\}$ $\vdash$ `x % 5 : Nat`

f) $\{$ `x` $\mapsto$ `Bool` $\}$ $\vdash$ **`fun`** `(x: Nat) -> x :` _____

$\{$ `x` $\mapsto$ `Bool` $\}$ $\vdash$ **`fun`** `(x: Nat) -> x : Nat` $\rightarrow$ `Nat`

g) $\emptyset \vdash$ **`let`** `f (x: Bool): Bool =` **`if`** `x` **`then false else`** `x :` _____

$\emptyset \vdash$ **`let`** `f (x: Bool): Bool =` **`if`** `x` **`then false else`** `x :` $\{$ `f` $\mapsto$ `Bool` $\rightarrow$ `Bool` $\}$

Aufgabe 3 Entwurfsmuster (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

a) Schreiben Sie die Funktion `interval: Nat -> List<Nat>`, welche für eine gegebene natürliche Zahl `n` die Liste `[n; ...; 1]` berechnet. Gehen Sie **strikt nach Peano Entwurfsmuster** vor.

Beispiele:

```
interval 0 = []          interval 1 = [1]          interval 2 = [2; 1]
```

__ /5

```
let rec interval (n: Nat): List<Nat> =
    if n = 0N then []
    else n :: interval (n - 1N)
```

b) Schreiben Sie die Funktion `ntimes<'a>: ('a -> 'a) -> 'a -> Nat -> 'a`, welche eine Funktion `f` vom Typ `'a -> 'a`, einen Startwert `x` vom Typ `'a` sowie eine natürliche Zahl `n` nimmt und die Funktion `f` auf `x` `n`-mal anwendet. Für `n = 0` soll `x` unverändert zurückgegeben werden. Gehen Sie **strikt nach Peano Entwurfsmuster** vor.

Beispiele:

```
ntimes (fun x -> x * x) 3 0 = 3          ntimes (fun x -> x * 2) 1 5 = 32
```

__ /5

```
let rec ntimes<'a> (f: 'a -> 'a) (x: 'a) (n: Nat): 'a =
    if n = 0N then x
    else f (ntimes f x (n - 1N))
```

Für die nächsten beiden Teilaufgaben verwenden wir folgenden Typen für Binärbäume:

```
type Tree = | Leaf | Node of Tree * Tree
```

c) Schreiben Sie die Funktion `complete: Nat -> Tree`, die für eine gegebene natürliche Zahl `n` den Baum der Höhe `n` berechnet, der die maximal mögliche Anzahl an Blättern besitzt (einen sogenannten *vollständigen Binärbaum* der Höhe `n`). Gehen Sie **strikt nach Peano Entwurfsmuster** vor. Sie dürfen **nur einen rekursiven Aufruf** verwenden.

Beispiele:

```
complete 0 = Leaf
complete 1 = Node (Leaf, Leaf)
complete 2 = Node (Node (Leaf, Leaf), Node (Leaf, Leaf))
```

```
let rec complete (n: Nat): Tree =
    if n = 0N then Leaf
    else
        let t = complete (n - 1N)
        Node (t, t)
```

__/5

d) Schreiben Sie die Funktion `nodes: Tree -> Nat`, welche die Anzahl der inneren Knoten im gegebenen Baum bestimmt. Gehen Sie **nach Struktur Entwurfsmuster des Datentyps** vor.

Beispiele:

```
nodes Leaf = 0
nodes Node (Leaf, Leaf) = 1
nodes Node (Leaf, Node (Leaf, Leaf)) = 2
```

```
let rec nodes (t: Tree): Nat =
    match t with
    | Leaf -> 0N
    | Node (l, r) -> 1N + (nodes l) + (nodes r)
```

__/5

Aufgabe 4 Mengen (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

Wir betrachten folgenden Typen, um Mengen zu modellieren:

```
type Set<'a> = 'a -> Bool
```

Ist `m` vom Typ `Set<'a>` gegeben und `x` ein Element vom Typ `'a`, so ist `m x = true` genau dann, wenn $x \in m$ gilt.

Hinweis: Sie dürfen davon ausgehen, dass Werte vom Typ `'a` mit dem Gleichheitsoperator '=' vergleichbar sind.

a) Geben Sie die leere Menge $\emptyset$ an:

__ /5

```
let empty<'a> : Set<'a> =
    fun (_: 'a) -> false
```

b) Schreiben Sie eine Funktion `add<'a>: 'a -> Set<'a> -> Set<'a>`, die das gegebene Element in die gegebene Menge einfügt.

__ /5

```
let add<'a when 'a: equality> (elem: 'a) (m: Set<'a>): Set<'a> =
    fun (e: 'a) -> e = elem || m e
```

c) Schreiben Sie eine Funktion `contains<'a>: 'a -> Set<'a> -> Bool`, die überprüft, ob das gegebene Element in der gegebenen Menge enthalten ist.

__ /5

```
let contains<'a> (elem: 'a) (m: Set<'a>): Bool =
    m elem
```

d) Schreiben Sie eine Funktion `difference<'a>: Set<'a> -> Set<'a> -> Set<'a>`, die die Differenzmenge zweier Mengen berechnet. Die Differenzmenge ist die Menge aller Elemente, die in der ersten Menge enthalten sind, aber nicht in der zweiten. Formal ausgedrückt bedeutet dies: $x \in \texttt{difference m1 m2} \Leftrightarrow x \in m1 \wedge x \notin m2$.

__ /5

```
let difference<'a> (m1: Set<'a>) (m2: Set<'a>): Set<'a> =
    fun (e: 'a) -> m1 e && not (m2 e)
```

Aufgabe 5 Bäume (__ /20 Punkte)

Lösen Sie diese Aufgabe **funktional**, d. h. `mutable` und `ref` dürfen in Ihrer Lösung nicht vorkommen. Verwenden Sie **keine Bibliotheksfunktionen!**

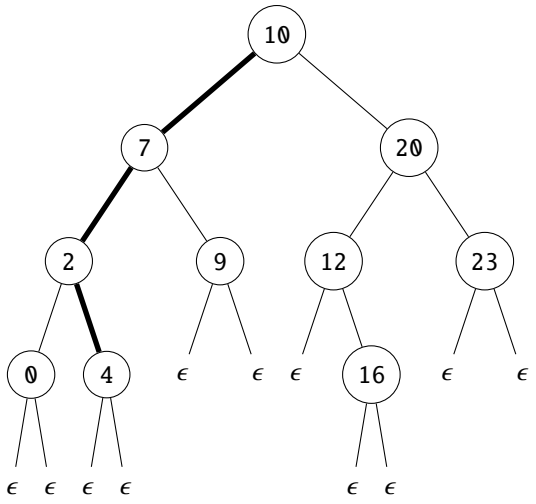
Wir betrachten folgende Typen:

```
type Tree<'a> = | Leaf | Node of Tree<'a> * 'a * Tree<'a>

type Path = | End | Left of Path | Right of Path
```

Bäume des Typs `Tree<'a>` speichern ihre Daten in den Knoten. Der Typ `Path` repräsentiert Pfade im Baum. Pfade beginnen immer in der Wurzel des Baumes.

Beispiel: Rechts sehen Sie den Code zu dem hier dargestellten Baum und dem hervorgehobenen Pfad mit den Knoten 10, 7, 2, 4.




```
let exTree: Tree<Nat> =
  Node (
    Node (
      Node (
        Node (Leaf, 0, Leaf),
        2,
        Node (Leaf, 4, Leaf)
      ),
      7,
      Node (Leaf, 9, Leaf)
    ),
    10,
    Node (
      Node (
        Leaf,
        12,
        Node (Leaf, 16, Leaf)
      ),
      20,
      Node (Leaf, 23, Leaf)
    )
  )

let exPath: Path = Left (Left (Right End))
```

a) Schreiben Sie eine Funktion `lookup: Tree<'a> -> Path -> Option<'a>`, die den Wert des Knotens bestimmt, zu dem der gegebenen Pfad führt. Wenn der Pfad zu keinem Knoten führt, soll `None` zurückgegeben werden.

Beispiele:

```
lookup exTree End = Some 10
lookup exTree exPath = Some 4
lookup exTree (Left (Right (Left End))) = None
lookup exTree (Left (Right (Left (Left End)))) = None
```

__ /6

```
let rec lookup (t: Tree<'a>) (p: Path): Option<'a> =
    match (t, p) with
    | (Leaf, _)                  -> None
    | (Node (_, x, _), End)      -> Some x
    | (Node (l, _, _), Left pp)  -> lookup l pp
    | (Node (_, _, r), Right pp) -> lookup r pp
```

b) Schreiben Sie eine Funktion `update: Tree<'a> -> Path -> 'a -> Tree<'a>`, die den Baum berechnet, der entsteht, wenn man im gegebenen Baum den Wert des Knotens, zu dem der Pfad führt, zum gegebenen Wert `x` ändert. Führt der Pfad zu keinem Knoten, soll der Baum unverändert zurückgegeben werden.

Beispiele:

```
let t = Node (Node (Leaf, 2, Leaf), 3, Leaf)
update t End 4 = Node (Node (Leaf, 2, Leaf), 4, Leaf)
update t (Left End) 4 = Node (Node (Leaf, 4, Leaf), 3, Leaf)
update t (Right End) 4 = Node (Node (Leaf, 2, Leaf), 3, Leaf)
```

__/6

```
let rec update (t: Tree<'a>) (p: Path) (x: 'a): Tree<'a> =
    match (t, p) with
    | (Leaf, _)                  -> t
    | (Node (l, _, r), End)      -> Node (l, x, r)
    | (Node (l, y, r), Left pp)  -> Node (update l pp x, y, r)
    | (Node (l, y, r), Right pp) -> Node (l, y, update r pp x)
```

c) Schreiben Sie eine Funktion `search: Tree<Nat> -> Nat -> Option<Path>`, die im gegebenen *Suchbaum* einen Pfad zum gegebenen Wert `x` bestimmt. Befindet sich `x` nicht im Baum, soll `None` zurückgegeben werden.

Zur Erinnerung: Ein Suchbaum ist ein Baum, bei dem für jeden Knoten die Elemente im linken Teilbaum kleiner und im rechten Teilbaum größer sind als das Element im Knoten selbst. Der Baum `exTree` oben ist ein Suchbaum.

Nutzen Sie aus, dass es sich um einen Suchbaum handelt.

Beispiel:

```
search exTree 10 = Some End
search exTree 4 = Some (Left (Left (Right (End))))
search exTree 3 = None
```

__/8

```
let rec search (t: Tree<Nat>) (x: Nat): Option<Path> =
    match t with
    | Leaf -> None
    | Node (l, y, r) ->
        if x = y then Some End
        elif x < y then
            match search l x with
            | None -> None
            | Some p -> Some (Left p)
        else
            match search r x with
            | None -> None
            | Some p -> Some (Right p)
```

Aufgabe 6 Reguläre Ausdrücke (__ /20 Punkte)

a) Wir betrachten den regulären Ausdruck

$$b \cdot a \cdot \Big((a \cdot b \cdot a) \mid (b \cdot a \cdot b) \mid (b \cdot b)\Big)^* \cdot a$$

über dem Alphabet $\{a, b\}$.

Kreuzen Sie an, ob die folgenden Wörter in der von dem Ausdruck beschriebenen Sprache enthalten sind oder nicht. Für richtige Antworten erhalten Sie einen Punkt, für falsche Antworten wird ein Punkt abgezogen. Nicht markierte Zeilen wirken sich nicht auf die Punktzahl aus. Diese Teilaufgabe wird mit mindestens 0 Punkten bewertet.

__ /6

Wort	enthalten	nicht enthalten
bababbbababba	X	
ababaaababaab		X
baababababbbba	X	
babbbbbbbbbba	X	
bababaababbba		X
bababababababa		X

b) Geben Sie für die folgenden Beschreibungen in natürlicher Sprache einen regulären Ausdruck über dem Alphabet $\{a, b, c\}$ an:

__ /8

1. Die Sprache, deren Wörter genau ein b enthalten.

   $(a \mid c)^* \cdot b \cdot (a \mid c)^*$

2. Die Sprache, deren Wörter kein a enthalten.

   $(b \mid c)^*$

3. Die Sprache, deren Wörter aus genau drei Buchstaben bestehen.

   $(a \mid b \mid c) \cdot (a \mid b \mid c) \cdot (a \mid b \mid c)$

4. Die leere Sprache.

   $\emptyset$

c) Bestimmen Sie die folgenden Rechtsfaktoren. Geben Sie in der Rechnung **jeweils den ersten Schritt explizit** an, nachfolgende Zwischenschritte dürfen Sie zusammenfassen.

Alphabet: {a, b, c}

__ /6

$$\begin{aligned} a \setminus \Big((a \cdot (b \mid c))^* \cdot (b \cdot (c \mid a))\Big) &= \Big(\big(a \setminus (a \cdot (b \mid c))^*\big) \cdot (b \cdot (c \mid a))\Big) \mid \big(a \setminus (b \cdot (c \mid a))\big) \\ &= \Big(\big(a \setminus (a \cdot (b \mid c))\big) \cdot (a \cdot (b \mid c))^* \cdot b \cdot (c \mid a)\Big) \mid \emptyset \\ &= (b \mid c) \cdot (a \cdot (b \mid c))^* \cdot b \cdot (c \mid a) \end{aligned}$$

$$\begin{aligned} b \setminus (a \mid (b \cdot b))^* &= \big(b \setminus (a \mid (b \cdot b))\big) \cdot (a \mid (b \cdot b))^* \\ &= b \cdot (a \mid (b \cdot b))^* \end{aligned}$$

$$\begin{aligned} c \setminus \Big((a \mid \epsilon) \cdot ((c \mid b)^* \cdot (a \cdot c)^*)\Big) &= \Big(\big(c \setminus (a \mid \epsilon)\big) \cdot ((c \mid b)^* \cdot (a \cdot c)^*)\Big) \mid \big(c \setminus ((c \mid b)^* \cdot (a \cdot c)^*)\big) \\ &= \Big(\emptyset \cdot ((c \mid b)^* \cdot (a \cdot c)^*)\Big) \mid \Big(\big(c \setminus (c \mid b)^*\big) \cdot (a \cdot c)^*\Big) \mid \big(c \setminus (a \cdot c)^*\big) \\ &= \emptyset \mid \Big(\big(c \setminus (c \mid b)\big) \cdot (c \mid b)^* \cdot (a \cdot c)^*\Big) \mid \Big(\big(c \setminus (a \cdot c)\big) \cdot (a \cdot c)^*\Big) \\ &= \Big(\epsilon \cdot (c \mid b)^* \cdot (a \cdot c)^*\Big) \mid \big(\emptyset \cdot (a \cdot c)^*\big) \\ &= (c \mid b)^* \cdot (a \cdot c)^* \end{aligned}$$