Aufgabe 1 (Programmanalyse): (12 + 6 = 18 Punkte)

a) Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Tragen Sie hierzu jeweils die ausgegebenen Zeichen in die markierten Stellen hinter „`OUT:`“ ein.

```
public class A {
    long x;
    Integer y;

    public A() {
      x = 0;
      y = 0;
    }
    public A(int a, int b) {
      x = a;
      y = b + 1;
    }
    public A(long a, Integer b) {
      x = a;
      y = b + 2;
    }

    public void f(int a) {
      x = 1;
    }
    public void f(float a) {
      x = 2;
    }
}
```

```
public class B extends A {
    int y;

    public B() {
      this.y = super.y;
    }
    public B(int a) {
      super(a, a);
      y = 2 * a;
    }

    public void f(int a) {
      x = 3;
    }
    public void f(Integer a) {
      x = 4;
    }
    public void f(double a) {
      x = 5;
    }
}
```

```
public class M {
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x + " " + a1.y);  // OUT: [    ] [    ]

        A a2 = new A(1, 2);
        System.out.println(a2.x + " " + a2.y);  // OUT: [    ] [    ]

        A a3 = new A(3, Integer.valueOf(4));
        System.out.println(a3.x + " " + a3.y);  // OUT: [    ] [    ]

        B b1 = new B();
        System.out.println(b1.x + " " + b1.y);  // OUT: [    ] [    ]

        A ab = new B(5);
        System.out.println(ab.x + " " + ab.y);  // OUT: [    ] [    ]

        System.out.println(((B)ab).y);          // OUT: [    ]

        b1.f(2.0f);
        System.out.println(b1.x);               // OUT: [    ]

        ab.f(Integer.valueOf(3));
        System.out.println(ab.x);               // OUT: [    ]
    }
}
```

b) Wir schreiben zusätzlich zu `A` und `B` eine neue Klasse `C`. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
1  public class C extends A {
2      long z;
3
4      public C(long x) {
5        super(x, x);
6        z = x;
7      }
8
9      public void f(float z) {
10       y *= z;
11     }
12
13     public void f(int a, b) {
14       int i, j;
15       for(i = 0; i < a; ++i) {
16         for(j = 0; j < b; ++j) {
17           z += b;
18         }
19       }
20     }
21 }
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
        A a1 = new A();
        System.out.println(a1.x + " " + a1.y);      // OUT: [  0 ] [  0 ]

        A a2 = new A(1, 2);
        System.out.println(a2.x + " " + a2.y);      // OUT: [  1 ] [  3 ]

        A a3 = new A(3, Integer.valueOf(4));
        System.out.println(a3.x + " " + a3.y);      // OUT: [  3 ] [  6 ]

        B b1 = new B();
        System.out.println(b1.x + " " + b1.y);      // OUT: [  0 ] [  0 ]

        A ab = new B(5);
        System.out.println(ab.x + " " + ab.y);      // OUT: [  5 ] [  6 ]

        System.out.println(((B)ab).y);              // OUT: [ 10 ]

        b1.f(2.0f);
        System.out.println(b1.x);                   // OUT: [  2 ]

        ab.f(Integer.valueOf(3));
        System.out.println(ab.x);                   // OUT: [  3 ]
    }
}
```

b)
- Zeile 5: Es existiert kein passender Konstruktor in der Klasse `A`, der zwei Argumente vom Typ `long` akzeptiert.

- Zeile 10: Da der Operand `z` den Typ `float` hat, kann das Ergebnis der Multiplikation nicht implizit nach `int` umgewandelt und in dem Attribut `y` gespeichert werden.
- Zeile 13: In Funktionsdeklarationen muss für jeden Parameter der Typ angegeben werden. (Bei Variablendeklarationen ist es hingegen möglich, mehrere Variablen gleichen Typs durch Komma getrennt zu deklarieren, wie in Zeile 14.)

Aufgabe 2 (Hoare-Kalkül): (14 Punkte)

Gegeben sei folgendes Java-Programm P, das zu zwei ganzzahligen Eingaben `a` und `b`, wobei `b` > 0 ist, den Rest bei der Division von `a` durch `b` berechnet, d.h. es berechnet `a` mod `b`.

$\langle$ `b` > 0 $\rangle$ (Vorbedingung)

```
r = a;
q = 0;
while (abs(r) > b/2) {
    if (r < 0) {
        q = q - 1;
        r = r + b;
    } else {
        q = q + 1;
        r = r - b;
    }
}
```

$\langle$ `r` $=$ `a` mod `b` $\rangle$ (Nachbedingung)

Hinweise:

- Es liegt folgende Definition der Division mit Rest zugrunde:
  Seien $a \in \mathbb{Z}$ und $b \in \mathbb{N}$, $b > 0$. Dann gibt es eine ganze Zahl q und eine ganze Zahl r mit $|r| \leq \lfloor \frac{b}{2} \rfloor$, sodass $a = q \cdot b + r$. Diese Zahl r heißt der Rest bei Division von a durch b, kurz $r = a \bmod b$.
- Die Ganzzahl-Division `b/2` im oben gegebenen Programm berechnet $\lfloor \frac{b}{2} \rfloor$, also z.B. $3/2 = 1 = \lfloor \frac{3}{2} \rfloor$.
- Die Funktion `abs` berechnet den Betrag, d.h. `abs(3) = 3` und `abs(-3) = 3`.
- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x+1 = y+1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen. Klammern dürfen und müssen Sie jedoch eventuell bei der Anwendung der Zuweisungsregel setzen.

Vervollständigen Sie die Verifikation des Algorithmus P auf der folgenden Seite im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Lösung:

$\langle$ `b` $> 0 \rangle$
$\langle$ `b` $> 0 \wedge$ `a` $=$ `a` $\wedge 0 = 0 \rangle$

```
r = a;
```

$\langle$ `b` $> 0 \wedge$ `r` $=$ `a` $\wedge 0 = 0 \rangle$

```
q = 0;
```

$\langle$ `b` $> 0 \wedge$ `r` $=$ `a` $\wedge$ `q` $= 0 \rangle$
$\langle$ `b` $> 0 \wedge$ `q` $\cdot$ `b` $+$ `r` $=$ `a` $\rangle$

```
while (abs(r)>b/2) {
```

$\langle$ `b` $> 0 \wedge$ `q` $\cdot$ `b` $+$ `r` $=$ `a` $\wedge$ `abs(r)` $> \lfloor \frac{\texttt{b}}{2} \rfloor \rangle$

```
    if (r < 0) {
```

$\langle$ `b` $> 0 \wedge$ `q` $\cdot$ `b` $+$ `r` $=$ `a` $\wedge$ `abs(r)` $> \lfloor \frac{\texttt{b}}{2} \rfloor \wedge$ `r` $< 0 \rangle$

```
                        ⟨b > 0 ∧ (q - 1) · b + (r + b) = a⟩
        q = q - 1
                        ⟨b > 0 ∧ q · b + (r + b) = a⟩
        r = r + b;
                        ⟨b > 0 ∧ q · b + r = a⟩
    } else {
                        ⟨b > 0 ∧ q · b + r = a ∧ abs(r) > ⌊b/2⌋ ∧ ¬(r < 0)⟩
                        ⟨b > 0 ∧ (q + 1) · b + (r − b) = a⟩
        q = q + 1;
                        ⟨b > 0 ∧ q · b + (r − b) = a⟩
        r = r - b;
                        ⟨b > 0 ∧ q · b + r = a⟩
    }
                        ⟨b > 0 ∧ q · b + r = a⟩
}
                        ⟨b > 0 ∧ q · b + r = a ∧ ¬(abs(r) > ⌊b/2⌋)⟩
                        ⟨r = a mod b⟩
```

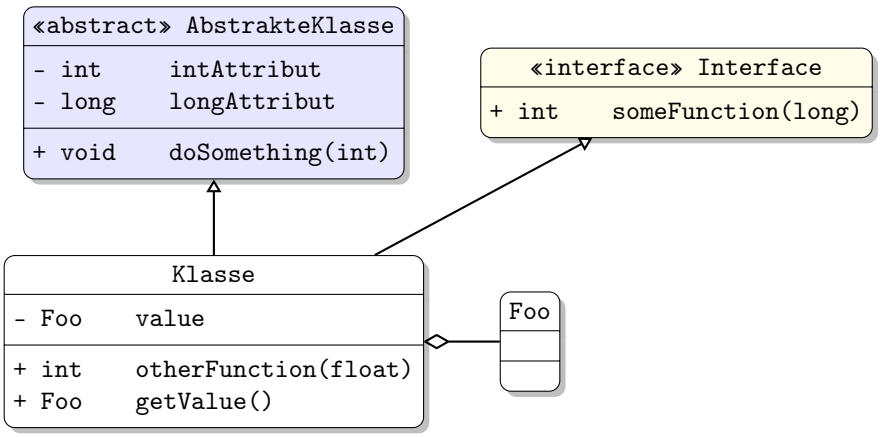
Aufgabe 3 (Klassenhierarchie): (9 + 8 = 17 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Klassenhierarchie zur Verwaltung von Ärzten, Patienten und Krankheiten.

- Ärzte und Patienten sind Personen. Jede Person hat einen Namen und ein Alter. In unserem Modell soll es keine weiteren Personen geben.
- Ein Arzt hat ein Array seiner Patienten. Er hat außerdem eine Methode, um ein Attest für eine gegebene Anzahl von Tagen auszustellen.
- Ein Patient kann genau eine Krankheit haben. (Falls ein Patient gesund ist, kann dies durch die Krankheit `null` dargestellt werden.) Manche Patienten haben einen Impfschutz gegen die saisonale Grippe. Patienten können geimpft werden und haben dazu eine Methode ohne Eingabeparameter.
- Jede Krankheit hat eine Dauer, die in Tagen gemessen wird.
- Eine Bronchitis ist eine Krankheit, die fieberhaft verlaufen kann. Wir unterscheiden zwischen einer viralen und einer bakteriellen Bronchitis. Es kann aber auch weitere Arten einer Bronchitis geben.
- Eine Bindehautentzündung ist ein weiteres Beispiel für eine Krankheit.
- Manche Krankheiten sind behandelbar. Dazu zählen die bakterielle Bronchitis und die Bindehautentzündung. Sie stellen deshalb eine Methode zur Behandlung zur Verfügung.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Sachverhalte. Notieren Sie keine Konstruktoren oder Selektoren. Sie müssen nicht markieren, ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden und markieren Sie alle Klassen als abstrakt, bei denen dies sinnvoll ist.

Verwenden Sie hierbei die folgende Notation:



Eine Klasse wird hier durch einen Kasten dargestellt, in dem der Name der Klasse sowie alle in der Klasse definierten bzw. überschriebenen Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \multimap\!\triangleright A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Der Pfeil $B \multimap\!\diamond A$ bedeutet, dass A ein Objekt vom Typ B benutzt. Benutzen Sie `-`, um `private` abzukürzen, und `+` für alle anderen Sichtbarkeiten (wie z. B. `public`). Fügen Sie Ihrem Diagramm keine Kästen für vordefinierte Klassen wie `String` hinzu.

b) Schreiben Sie eine Java-Methode `public void patientenBehandeln()` in der Klasse `Arzt`. In dieser Methode sollen alle Krankheiten derjenigen Patienten des aktuellen Patienten-Arrays behandelt werden, deren Krankheit behandelbar ist. Allen kranken Patienten soll zusätzlich ein Attest für die Dauer ihrer Krankheit ausgestellt werden und alle gesunden Patienten, die über 60 Jahre alt sind und noch keinen Impfschutz gegen die saisonale Grippe haben, sollen geimpft werden.

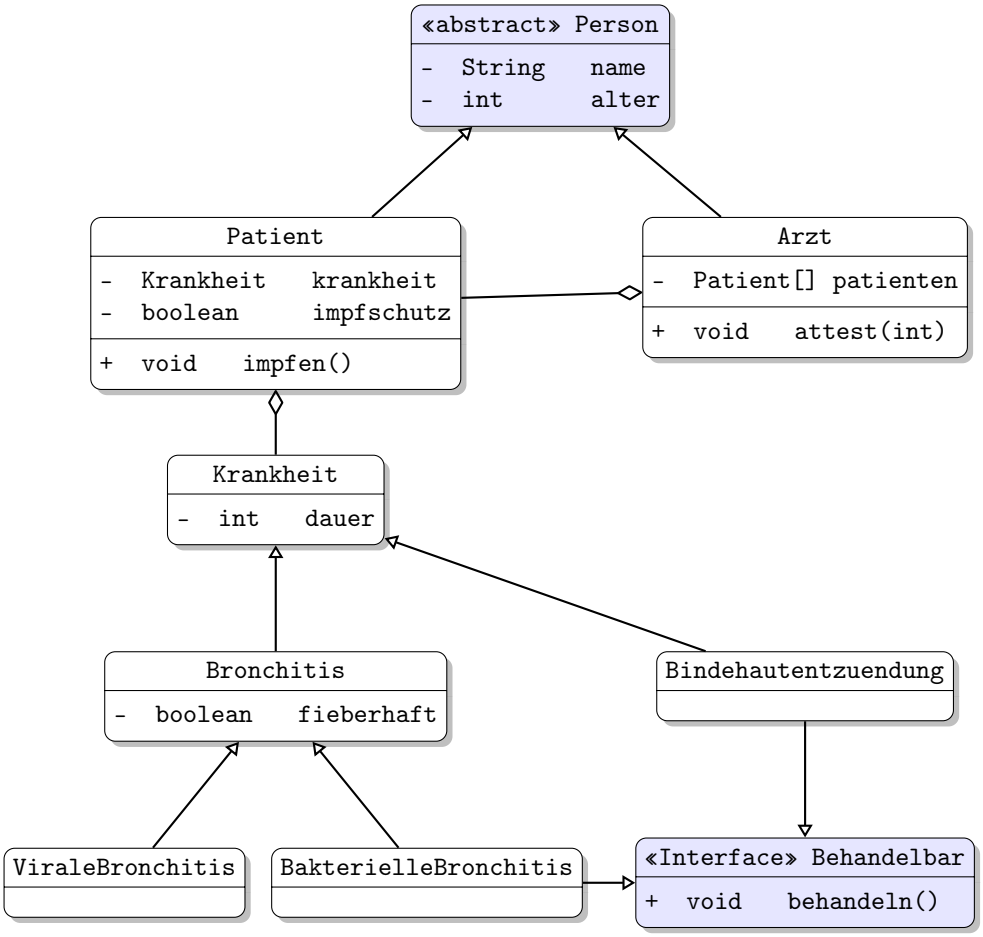
Gehen Sie davon aus, dass es zu jedem Attribut geeignete Selektoren (get- und set-Methoden) gibt.

Hinweise:

- Sie dürfen davon ausgehen, dass das Patienten-Array eines Arztes nie `null` ist und dass dieses Array auch keine `null`-Einträge enthält. Eine Krankheit eines Patienten kann hingegen `null` sein.

Lösung:

a) Die Zusammenhänge können wie folgt modelliert werden:



b)

```
public void patientenBehandeln() {
  for (Patient patient : this.patienten) {
    Krankheit k = patient.getKrankheit();
    if (k != null) {
      if (k instanceof Behandelbar) {
        ((Behandelbar)k).behandeln();
      }
      this.attest(k.getDauer());
    } else {
      if (patient.getAlter() > 60 && !patient.getImpfschutz()) {
        patient.impfen();
      }
    }
  }
```

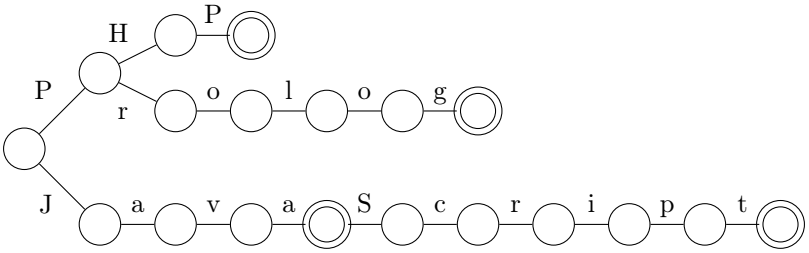
```
    }
```

Aufgabe 4 (Programmierung in Java): (7 + 2 + 12 + 3 + 7 = 31 Punkte)

In dieser Aufgabe beschäftigen wir uns mit *Präfixbäumen*. Ein Präfixbaum speichert eine Menge von Wörtern. Dabei stehen die Buchstaben an den Kanten zwischen den Knoten. Jeder Knoten hat ein Attribut `end`, das angibt, ob das Wort auf dem Pfad von der Wurzel bis zum aktuellen Knoten in der Menge enthalten sein soll. Wir nutzen die folgenden Klassen zur Repräsentation von Präfixbäumen.

```
public class Node {
  private LinkedList<Edge> edges;
  private boolean end;

  public Node(boolean end) {
    this.end = end;
    this.edges = new LinkedList<>();
  }

  public void addEdge(Edge e) {
    this.edges.add(e);
  }
}

public class Edge {
  private char value;
  private Node child;

  public Edge(char value, Node child) {
    this.value = value;
    this.child = child;
  }
}
```

Die folgende Grafik repräsentiert einen Präfixbaum, der die Werte
`'J','a','v','a'`,
`'J','a','v','a','S','c','r','i','p','t'`,
`'P','H','P'` und
`'P','r','o','l','o','g'` speichert.
Ein normaler Knoten wird dabei durch einen Kreis repräsentiert. Ein Endknoten, also ein Objekt der Klasse `Node`, bei dem `end` auf `true` gesetzt ist, wird durch einen doppelten Kreis repräsentiert.



Sie dürfen in allen Teilaufgaben beliebige Hilfsmethoden zu den Klassen `Node` und `Edge` hinzufügen. Wenn Sie von dieser Möglichkeit Gebrauch machen, müssen Sie eindeutig kennzeichnen, welche Hilfsmethoden zu welchen Klassen gehören. Außerdem können Sie davon ausgehen, dass für alle Attribute jeder Klasse geeignete Selektoren (get– und set–Methoden) bereitstehen.

Sie dürfen außerdem in allen Teilaufgaben davon ausgehen, dass Methoden nie `null` als Parameter übergeben bekommen. Gehen Sie weiterhin davon aus, dass die Liste `edges` nie `null` enthält. In anderen Arrays, Listen und Ähnlichem kann `null` aber als Wert vorkommen.

a) Schreiben Sie in der Klasse `Node` eine Methode `public int size()`. Der Rückgabewert ist die Anzahl der Endknoten in dem Baum. Ein Aufruf der Methode auf der Wurzel des Beispielbaums soll `4` zurück geben.

Hinweise:

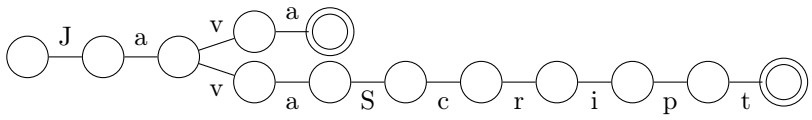
- Sie können die Methode `int size()` und die Methode `Edge get(int index)` aus der Klasse `LinkedList<Edge>` verwenden, um die Länge einer Linked List bzw. das Element an der Stelle `index` in einer Linked List zu erhalten.
- Sie können auch for-each Schleifen verwenden, um die Elemente einer `LinkedList<Edge>` zu durchlaufen.

b) Schreiben Sie eine Exception-Klasse `AmbiguousPathException`. Die einzige Methode dieser Klasse ist eine `toString`-Methode, die erklärt, dass der Pfad nicht eindeutig ist.

c) Implementieren Sie die Methode `boolean contains(char[] sequence)` in der Klasse `Node`. Diese soll `true` zurück geben, falls das Wort in der Zeichenreihe `sequence` im Baum gespeichert ist. Andernfalls soll sie `false` zurück geben.

Es soll eine `AmbigiousPathException` geworfen werden, falls bei der Suche nach der gegebenen Sequenz ein nicht eindeutiger Pfad im Baum gefunden wird (d.h. falls ein Knoten mit zwei Kanten mit gleicher Beschriftung gefunden wird).

Ein Aufruf der Methode auf dem Beispielbaum mit `contains(new char[]{'J','a','v','a'})` würde `true` zurück geben, ein Aufruf von `contains(new char[]{'H','a','s','k','e','l','l'})` hingegen `false`. Ein Aufruf mit `contains(new char[]{'J','a','v','a'})` auf dem folgenden Beispiel würde jedoch zu einer `AmbiguousPathException` führen.



Hinweise:

- Sie können davon ausgehen, dass die Methode nie `null` übergeben bekommt. Der Parameter `sequence` kann aber ein leeres Array sein.

d) Passen Sie die Klassen `Node` und `Edge` sowie ihre Attribute so an, dass Sequenzen beliebiger Objekte eines Typs als Werte gespeichert werden können. Erweitern Sie die Klassen zu diesem Zweck um einen generischen Typparameter `T`, der den Typ der Werte angibt, die an die Kanten geschrieben werden. **Es ist nicht notwendig, die Konstruktoren und sonstige Methoden in den Klassen `Node` und `Edge` anzupassen!**

e) Implementieren Sie eine statische Methode
`public static <T> LinkedList<T[]> intersection(Node<T> tree, LinkedList<T[]> ll)`, die eine neue Liste aller in `ll` gespeicherten Sequenzen zurück gibt, die auch in `tree` enthalten sind. Duplikate in `ll` sollen also auch mehrfach im Ergebnis enthalten sein. Verwenden Sie hierzu die Methode `contains` aus Aufgabenteil **c)**, auch wenn Sie sie nicht implementiert haben. Gehen Sie davon aus, dass die Signatur dieser Methode zu `contains(T[] sequence)` angepasst wurde.

Fangen Sie die unter Umständen auftretende `AmbiguousPathException` und behandeln Sie diesen Fall so, als wäre die entsprechende Sequenz nicht in `tree` enthalten.

Hinweise:

- Der Typ `LinkedList<T[]>` bezeichnet eine Liste von Arrays, wobei die Arrays Elemente vom Typ `T` enthalten. Wenn Sie also `get(i)` auf `ll` aufrufen, erhalten Sie das Array vom Typ `T[]` an der Position `i` in der Liste `ll`. Entsprechend ist das aktuelle Element einer for-each Schleife immer vom Typ `T[]`, wenn die for-each Schleife die Liste `ll` durchläuft.
- Sie können wieder davon ausgehen, dass die Methode nie `null` übergeben bekommt und `ll` keine Elemente enthält, die `null` sind. Eine leere Liste als zweiter Parameter `ll` ist aber möglich.

Lösung:

a)
```
public int size() {
  int res = 0;
  if(this.end) {
    res += 1;
  }

  for(Edge e : this.edges) {
    res += e.getChild().size();
  }
  return res;
}
```

b)
```
public class AmbiguousPathException extends Exception {

  public String toString() {
    return "A path is ambiguous!";
  }
}
```

c)
```
public boolean contains(char[] sequence) throws AmbiguousPathException {
  Node current = this;

  for(int i = 0; i < sequence.length; ++i) {
    boolean found = false;
    for(Edge e: current.getEdges()) {
      if(e.getValue() == sequence[i]) {
        if(!found) {
          current = e.getChild();
          found = true;
        } else {
          throw new AmbiguousPathException();
        }
      }
    }
    if(!found) {
      return false;
    }
  }

  return current.isEnd();
}
```

d)
```
public class Node<T> {
  private LinkedList<Edge<T>> edges;
  private boolean end;
}

public class Edge<T> {
  private T value;
  private Node<T> child;
}
```

e)
```
public static <T> LinkedList<T[]>intersection(Node<T> tree,
                                              LinkedList<T[]> ll) {
  LinkedList<T[]> res = new LinkedList<>();
  for(T[] sequence : ll) {
    try {
```

```
        if(tree.contains(sequence)) {
          res.add(sequence);
        }
      } catch (AmbiguousPathException e) {
        //ignore and just don't add the sequence to the result
      }
    }
    return res;
  }
```

Aufgabe 5 (Haskell): (4 + 4 + 2 + 4 + 6 = 20 Punkte)

a) Geben Sie zu den folgenden Haskell-Funktionen `f` und `g` jeweils den allgemeinsten Typ an. Gehen Sie hierbei davon aus, dass alle Zahlen den Typ `Int` haben.

```
f [] [] z = if z then 1 else 0
f (x:xs)((y:ys):yss) _ = x+y

g z (x:xs) y = if y then x else (g x xs y)
```

b) Bestimmen Sie, zu welchem Ergebnis die Ausdrücke `i` und `j` jeweils auswerten.

```
i::Int
i = length ((\f x y -> f x y) (++) (filter (> 0) [-1, 4, 0]) (map (+ 1) [0,3,8]))
```

Hinweise:

- Die vordefinierte Funktion `length::[a] -> Int` gibt die Anzahl der Elemente einer Liste zurück. So ist `length [] = 0` und `length [7,11,14,9] = 4`. Außerdem ist `(++)` die Funktion zur Listenverkettung in Präfix-Schreibweise, d.h. `(++) [1,2] [3] = [1,2,3]`.

```
j::[[Int]]
j = map (\x->[x]) (filter even (map length [[],[1],[1,1],[1,1,1],[1,1,1,1]]))
```

Hinweise:

- Für die vordefinierte Funktion `even::Int -> Bool` ist `even x = True` genau dann, wenn `x` eine gerade ganze Zahl ist.

c) Wir verwenden die folgende Datenstruktur zur Darstellung von Polynomen in einer Variablen in Haskell. Wir nennen diese Variable x. Der Typ `Word` repräsentiert die natürlichen Zahlen $\mathbb{N}$ in Haskell. Er hat analoge Funktionalität wie `Int`, d.h. Sie können Ausdrücke vom Typ `Word` genauso behandeln wie Ausdrücke vom Typ `Int`.

```
data Polynom a = Koeff a Word (Polynom a) | Null
```

Hierbei bezeichnet `Null` das Nullpolynom und `Koeff c n p` stellt das Polynom $c \cdot x^n + p$ dar.

Beispielsweise wird das Polynom $4 \cdot x^3 + 2 \cdot x + 5$ mit ganzzahligen Koeffizienten dargestellt als `q1 = Koeff 4 3 (Koeff 2 1 (Koeff 5 0 Null))` mit dem Typ `q1::Polynom Int`.

Der Grad eines Polynoms ist wie folgt definiert: Das Nullpolynom hat den Grad 0. Das Polynom $c \cdot x^n + p$ hat den Grad $\max(n, \text{grad}(p))$.

Schreiben Sie eine Haskell-Funktion `grad::Polynom a -> Word`, die den Grad eines Polynoms berechnet. Der Aufruf `grad q1` soll beispielsweise den Rückgabewert `3` liefern.

Hinweise:

- Sie können die vordefinierte Methode `max::Word->Word->Word` benutzen, die das Maximum zweier natürlicher Zahlen berechnet.

d) Wie Sie vielleicht gemerkt haben, ist die hier gewählte Darstellung eines Polynoms nicht eindeutig. Das mathematische Polynom $4 \cdot x^3 + 2 \cdot x + 5$ kann sowohl durch `q1`, als auch durch `q2::Polynom Int` dargestellt werden mit `q2 = Koeff 4 3 (Koeff 1 1 (Koeff 5 0 (Koeff 1 1 Null)))`, was $4 \cdot x^3 + x + 5 + x$ entspricht. Schreiben Sie daher eine Funktion `gesKoeff:: Polynom Int -> Word -> Int`, die zu einem Polynom `p` und einer Potenz `n` den gesamten Koeffizienten von x^n in `p` zurückgibt.

Rufen wir also `gesKoeff q1 1` auf, so soll sich `2` ergeben. Rufen wir hingegen `gesKoeff q2 1` auf, so ergibt sich ebenfalls `2`.

e) Schreiben Sie eine Haskell-Funktion `vereinfache::Polynom Int -> Polynom Int`, die ein Polynom vom Typ `Polynom Int` in eine Standarddarstellung umformt. Ein Polynom `p` mit Typ `Polynom Int` ist eine Standarddarstellung von `t` vom Typ `Polynom Int`, falls gilt

- `grad t = grad p`.
- `gesKoeff p n = gesKoeff t n` für jedes `n` zwischen `0` und `grad t`.
- Kommt im Polynom `p` der Teilausdruck `Koeff c m r` vor, dann ist `c` der gesamte Koeffizient der Potenz x^m in `t` und `c` $\neq 0$.

Dies entspricht der Vereinfachung eines mathematischen Polynoms durch Zusammenfassen von Koeffizienten und Weglassen von Summanden der Form $0 \cdot x^m$.

Der Aufruf `vereinfache q2` liefert z.B. den Rückgabewert `Koeff 4 3 (Koeff 2 1 (Koeff 5 0 Null))`. Dies entspricht der Vereinfachung des mathematischen Polynoms $4 \cdot x^3 + x + 5 + x$ zu $4 \cdot x^3 + 2 \cdot x + 5$. Der Aufruf `vereinfache (Koeff 0 3 (Koeff 0 1 Null))` liefert `Null`.

Hinweise:

- Benutzen Sie die Funktionen `grad` und `gesKoeff` aus den vorherigen Teilaufgaben. Hierfür ist es unerheblich, ob Sie diese implementiert haben.
- Sie dürfen die vordefinierten Operationen `+`, `-`, `*` sowie `>`, `<`, `==` und `/=` (für ungleich) auf Zahlen benutzen.
- Die Reihenfolge der Teilausdrücke `Koeff c m r` in einer Standarddarstellung ist unerheblich. So ist z.B. auch `Koeff 5 0 (Koeff 2 1 (Koeff 4 3 Null))` eine Standarddarstellung von `q2`.

Lösung:

a) i) `f :: [Int] -> [[Int]] -> Bool -> Int`

ii) `g :: a -> [a] -> Bool -> a`

b)
```
i = 4
j = [[0],[2],[4]]
```

c)
```
grad :: Polynom a -> Word
grad Null = 0
--Die Abfrage c == 0 wurde in der Aufgabenstellung vernachlässigt
--und daher auch nicht bewertet.
grad (Koeff c n p) =  if (c == 0) then (grad p) else max n (grad p)
```

d)
```
gesKoeff :: Polynom Int -> Word -> a
gesKoeff Null _ = 0
gesKoeff (Koeff c n p) m = if (m == n) then c + (gesKoeff p m) else (gesKoeff p m)
```

e)
```
vereinfache :: Polynom Int -> Polynom Int
vereinfache p = vereinfache' p (grad p)
  where vereinfache' p n =
    if ((gesKoeff p n)/= 0) then
      if n > 0 then (Koeff (gesKoeff p n) n (vereinfache' p (n-1)))
      else (Koeff (gesKoeff p n) n Null)
    else if n > 0 then (vereinfache' p (n-1)) else Null
```

Aufgabe 6 (Prolog): (2 + 7 + 4 + 1 + 6 = 20 Punkte)

a) Geben Sie zu den folgenden Termpaaren jeweils einen allgemeinsten Unifikator an oder begründen Sie, warum sie nicht unifizierbar sind. Hierbei werden Variablen durch Großbuchstaben dargestellt und Funktionssymbole durch Kleinbuchstaben.

i) $f(a, s(X), s(s(Y))), f(Z, s(Z), s(Z))$

ii) $g(s(Y), X, s(s(a))), g(Z, s(Z), s(Y))$

b) Gegeben sei folgendes Prolog-Programm P.

```
p(X,s(Y)) :- p(Y,s(X)).
p(b,s(Y)) :- p(b,Y).
p(X,a).
```

Erstellen Sie für das Programm P den Beweisbaum zur Anfrage "`?- p(a,R).`" bis zur Höhe 4 (die Wurzel hat dabei die Höhe 1). Markieren Sie Pfade, die zu einer unendlichen Auswertung führen, mit ∞ und geben Sie alle Antwortsubstitutionen zur Anfrage "`?- p(a,R).`" an, die im Beweisbaum bis zur Höhe 4 enthalten sind. Geben Sie außerdem zu jeder dieser Antwortsubstitutionen an, ob sie von Prolog gefunden wird. Wie muss man die Reihenfolge der Programmklauseln ändern, damit Prolog jede Antwortsubstitution zur obigen Anfrage findet?

c) Implementieren Sie ein Prädikat `sumList` mit Stelligkeit 2 in Prolog, sodass `sumList(`t_1`,`t_2`)` genau dann für eine (möglicherweise leere) Liste $t_1 = [x_0, \ldots, x_n]$ vordefinierter ganzer Prolog-Zahlen x_i, $0 \leq i \leq n$, gilt, falls das i-te Element aus t_2 der Summe $x_i + \ldots + x_n$ entspricht. Beispielsweise soll `sumList([-1,2,1,1], Res)` die einzige Antwort `Res = [3,4,2,1]` liefern.

d) Binäre Bäume lassen sich in Prolog mit Hilfe der Funktionssymbole `leaf` und `node` als Terme darstellen. Dabei repräsentiert `leaf(`t`)` ein Blatt mit dem Wert t und `node(`l`,`t`,`r`)` repräsentiert einen Baum mit dem Wert t im Wurzelknoten, der den Teilbaum l als linkes Kind und den Teilbaum r als rechtes Kind hat. Implementieren Sie ein Prädikat `value` mit Stelligkeit 2 in Prolog, sodass `value(`t`,`x`)` für einen binären Baum t genau dann gilt, wenn x dem Wert der Wurzel von t entspricht. Beachten Sie, dass die Wurzel selbst ein Blatt sein kann.

e) Implementieren Sie ein Prädikat `sumTree` mit Stelligkeit 2 in Prolog, wobei `sumTree(`t_1`,`t_2`)` für einen binären Baum t_1 genau dann gilt, wenn t_2 der Baum ist, der sich aus t_1 ergibt, indem die Werte der Blätter erhalten bleiben und die Werte der inneren Knoten jeweils der Summe der Werte ihrer Kinder entsprechen. Beispielsweise soll `sumTree(node(leaf(2),0,node(leaf(-1),3,leaf(2))), Res)` die einzige Antwort `Res = node(leaf(2),3,node(leaf(-1),1,leaf(2)))` liefern.

Hinweise:

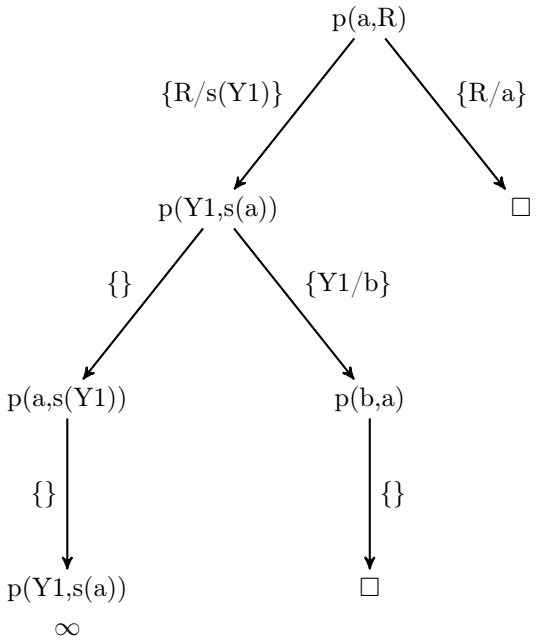
- Sie dürfen das Prädikat `value` aus Teilaufgabe **d)** verwenden, auch wenn Sie dieses Prädikat nicht implementiert haben.

Lösung:

a) i) $f(X, s(Y), Z), f(s(Z), Z, Z)$: clash failure $a \neq s$

ii) $g(s(Y), X, s(s(a))), g(Z, s(Z), s(Y))$: $X/s(s(s(a))), Y/s(a), Z/s(s(a))$

b)



Die Antwortsubstitutionen innerhalb des Beweisbaums sind {R/s(b)} und {R/a}. Diese werden von Prolog nicht gefunden.

Mit der folgenden Reihenfolge der Klauseln findet Prolog alle Antwortsubstitutionen:

```
p(X,a).
p(b,s(Y)) :- p(b,Y).
p(X,s(Y)) :- p(Y,s(X)).
```

c)
```
sumList([],[]).
sumList([X],[X]).
sumList([X|XS],[Y,Z|ZS]) :- sumList(XS,[Z|ZS]), Y is X + Z.
```

d)
```
value(leaf(X),X).
value(node(_,X,_),X).
```

e)
```
sumTree(leaf(X),leaf(X)).
sumTree(node(L,_,R),node(LS,X,RS)) :- sumTree(L,LS), sumTree(R,RS),
                                      value(LS,XL), value(RS,XR),
                                      X is XL + XR.
```