Aufgabe 1 (Programmanalyse): (14 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 {
  public long x = 0;
  public static int y = 20;

  public A(float x) {
    this((int) x);
    A.y++;
  }

  public A(int x) {
    this.x = x;
  }

  public int f(double d) {
    return 1;
  }

  public int f(long l) {
    return 2;
  }
}
```

```
public class B extends A {
  public int x = 1;

  public B() {
    this(42);
    B.y++;
  }

  public B(int x) {
    super(x + .0f);
    this.x += this.x;
  }

  public int f(long l) {
    return 3;
  }

  public int f(float f) {
    return 4;
  }
}
```

```
public class M {
  public static void main(String[] args) {
    A a = new A(10f);
    System.out.println(a.x);              // OUT: [     ]

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

    B b = new B();
    System.out.println(((A) b).x);        // OUT: [     ]

    System.out.println(b.x);              // OUT: [     ]

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

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

    System.out.println(b.f(1.));          // OUT: [     ]

    System.out.println(b.f(1.d));         // OUT: [     ]

    A ab = b;
    System.out.println(ab.f(1));          // OUT: [     ]
  }
}
```

Lösung:

a)
```
public class M {
    public static void main(String[] args) {
      A a = new A(10f);
      System.out.println(a.x);              // OUT: [ 10  ]

      System.out.println(A.y);              // OUT: [ 21  ]

      B b = new B();
      System.out.println(((A) b).x);        // OUT: [ 42  ]

      System.out.println(b.x);              // OUT: [  2  ]

      System.out.println(A.y);              // OUT: [ 23  ]

      System.out.println(B.y);              // OUT: [ 23  ]

      System.out.println(b.f(1.));          // OUT: [  1  ]

      System.out.println(b.f(1.d));         // OUT: [  1  ]

      A ab = b;
      System.out.println(ab.f(1));          // OUT: [  3  ]
    }
  }
```

Aufgabe 2 (Hoare-Kalkül): (9 + 3 = 12 Punkte)

Gegeben sei folgendes Java-Programm P, das zu einer Eingabe $\texttt{a}, \texttt{b} \geq 0$ den Wert $(\texttt{a}+\texttt{b})^2$ berechnet.

$\langle\, \texttt{b} \geq 0 \,\rangle$ (Vorbedingung)

```
res = a * a;
n = 0;
while (n < b) {
    n = n + 1;
    res = res + 2 * a + b;
}
```

$\langle\, \texttt{res} = (\texttt{a}+\texttt{b})^2 \,\rangle$ (Nachbedingung)

a) 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.

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und unter Verwendung des Hoare-Kalküls die Terminierung unter der Voraussetzung $\texttt{b} \geq 0$ bewiesen werden. Begründen Sie, warum es sich bei der von Ihnen angegebenen Variante tatsächlich um eine gültige Variante handelt.

Lösung:

a)

$\langle \texttt{b} \geq 0 \wedge 0 = 0 \wedge 0 = 0 \rangle$

```
res = a * a;
```

$\langle \texttt{b} \geq 0 \wedge \texttt{res} = \texttt{a} \cdot \texttt{a} \wedge 0 = 0 \rangle$

```
n = 0;
```

$\langle \texttt{b} \geq 0 \wedge \texttt{res} = \texttt{a} \cdot \texttt{a} \wedge \texttt{n} = 0 \rangle$
$\langle \texttt{n} \leq \texttt{b} \wedge \texttt{res} = \texttt{a} \cdot \texttt{a} + 2 \cdot \texttt{a} \cdot \texttt{n} + \texttt{b} \cdot \texttt{n} \rangle$

```
while (n < b) {
```

$\langle \texttt{n} < \texttt{b} \wedge \texttt{n} \leq \texttt{b} \wedge \texttt{res} = \texttt{a} \cdot \texttt{a} + 2 \cdot \texttt{a} \cdot \texttt{n} + \texttt{b} \cdot \texttt{n} \rangle$
$\langle \texttt{n} + 1 \leq \texttt{b} \wedge \texttt{res} + 2 \cdot \texttt{a} + \texttt{b} = \texttt{a} \cdot \texttt{a} + 2 \cdot \texttt{a}(\texttt{n}+1) + \texttt{b}(\texttt{n}+1) \rangle$

```
    n = n + 1;
```

$\langle \texttt{n} \leq \texttt{b} \wedge \texttt{res} + 2 \cdot \texttt{a} + \texttt{b} = \texttt{a} \cdot \texttt{a} + 2 \cdot \texttt{an} + \texttt{bn} \rangle$

```
    res = res + 2 * a + b;
```

$\langle \texttt{n} \leq \texttt{b} \wedge \texttt{res} = \texttt{a} \cdot \texttt{a} + 2 \cdot \texttt{a} \cdot \texttt{n} + \texttt{b} \cdot \texttt{n} \rangle$

```
}
```

$\langle \neg(\texttt{n} < \texttt{b}) \wedge \texttt{n} \leq \texttt{b} \wedge \texttt{res} = \texttt{a} \cdot \texttt{a} + 2 \cdot \texttt{a} \cdot \texttt{n} + \texttt{b} \cdot \texttt{n} \rangle$
$\langle \texttt{res} = (\texttt{a}+\texttt{b})^2 \rangle$

b) Eine gültige Variante für die Terminierung ist $V = \texttt{b} - \texttt{n}$, denn die Schleifenbedingung $B = \texttt{n} < \texttt{b}$ impliziert $\texttt{b} - \texttt{n} \geq 0$ und es gilt:

$\langle \texttt{b} - \texttt{n} = m \wedge \texttt{n} < \texttt{b} \rangle$
$\langle \texttt{b} - (\texttt{n}+1) < m \rangle$

```
    n = n + 1;
```

$\langle \texttt{b} - \texttt{n} < m \rangle$

```
    res = res + 2 * a + b;
```

$$\langle \mathtt{b} - \mathtt{n} < m \rangle$$

Damit ist die Terminierung der einzigen Schleife in P gezeigt.

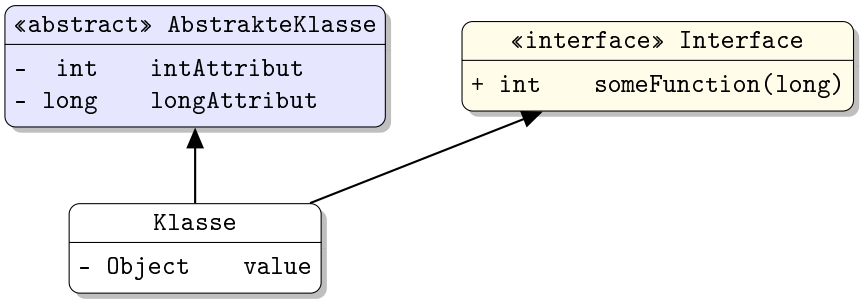
Aufgabe 3 (Klassen-Hierarchie): (3 + 6 = 9 Punkte)

In dieser Aufgabe betrachten wir verschiedene Arten von Telefonen und organisieren diese in einer Hierarchie.

- Ein Telefon hat eine Telefonnummer (bedenken Sie, dass diese eine führende 0 enthalten kann).
- Ein Mobiltelefon ist ein spezielles Telefon, das sich durch seine Displaygröße auszeichnet.
- Ein Festnetztelefon ist ein spezielles Telefon, dessen Anschlussort (als `String`) gespeichert werden soll.
- Ein Schnurlostelefon ist ein spezielles Festnetztelefon, bei dem die maximale Entfernung von der Basisstation bei der Nutzung von Interesse ist.
- Mobil- und Schnurlostelefone müssen regelmäßig aufgeladen werden und sollen eine Methode `aufladen()` zur Vergügung stellen, die den dafür benötigten Strom in Wattstunden als Gleitkommazahl zurückgibt.
- Alle Telefone sind entweder Festnetz- oder Mobiltelefone.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Telefonen. 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:

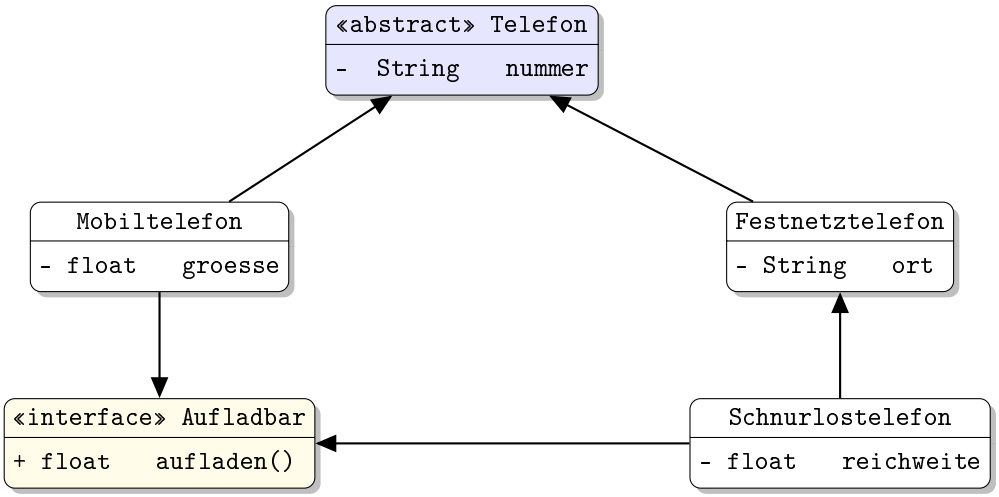


Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \blacktriangleright A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Benutzen Sie `-`, um `private` abzukürzen, und `+` für alle anderen Sichtbarkeiten (wie z. B. `public`).

b) Schreiben Sie eine Java-Methode mit der folgenden Signatur:
`public static float aufladen(Telefon[] telefone)`.
Diese Methode soll alle aufladbaren Telefone im Eingabearray aufladen und zurückgeben, wieviel Strom dafür insgesamt benötigt wurde. Nehmen Sie dazu an, dass das übergebene Array `telefone` nicht `null` ist.

Lösung:

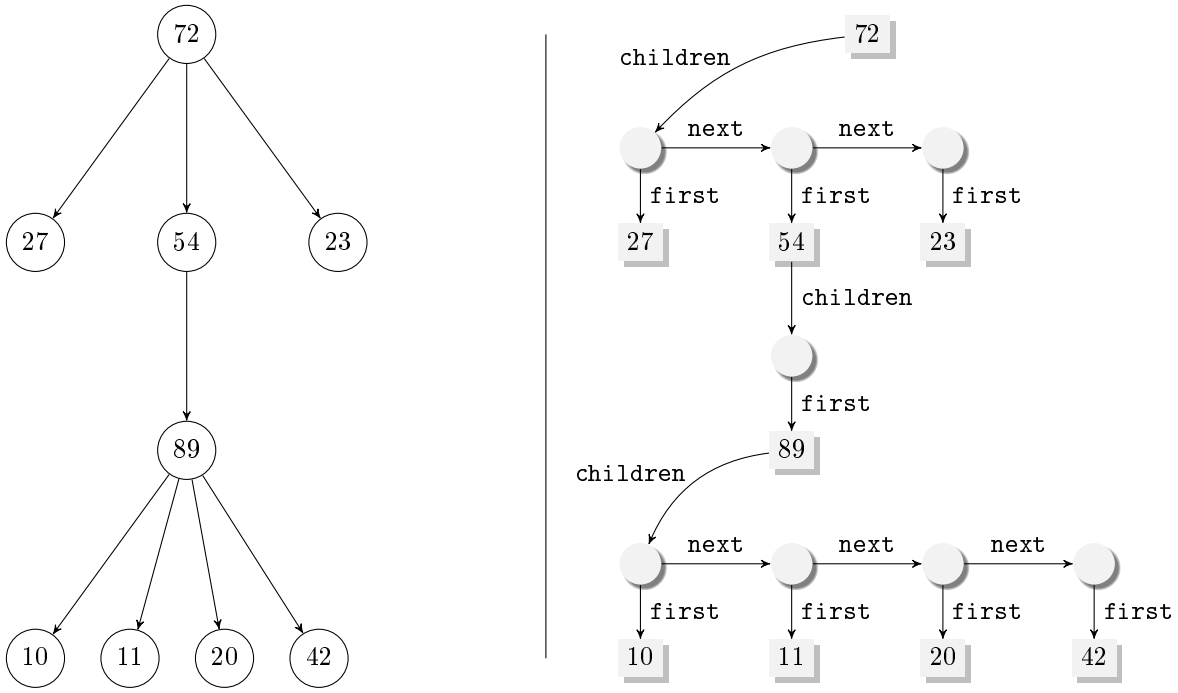
a) Die Zusammenhänge können wie folgt modelliert werden:



b)
```
public static float aufladen(Telefon[] telefone) {
    float sum = 0;
    for (Telefon t : telefone) {
        if (t instanceof Aufladbar) {
            sum += ((Aufladbar) t).aufladen();
        }
    }
    return sum;
}
```

Aufgabe 4 (Programmieren in Java): (9 + 13 + 13 + 3 + 13 = 51 Punkte)

Bäume, in denen jeder Knoten beliebig viele Nachfolger haben kann, sind eine Verallgemeinerung von Binärbäumen. Diese Bäume werden Vielwegebäume genannt. Ein solcher Baum ist in der folgenden Grafik illustriert:



Wir stellen solche Bäume mit der folgenden Datenstruktur dar:

```
public class Tree {
  public int value;
  public List children;

  public Tree(int v) {
    this.value = v;
    this.children = null;
  }
}
```

```
public class List {
  public Tree first;
  public List next;

  public List(Tree t) {
    this.first = t;
    this.next = null;
  }
}
```

Wir speichern also pro Baumelement einen Verweis auf eine Liste von Kindern (in `children`). Die Liste von Kindern ist mit der Klasse `List` realisiert, wobei jedes Listenelement einen Baum (in `first`) und die restliche Liste (in `next`) referenziert. Die leere Liste stellen wir hier mit `null` dar. Ein Baum ohne Kinder (also ein Blatt) verweist also im Attribut `children` auf `null`.
Gehen Sie in allen Teilaufgaben davon aus, dass das Attribut `first` der Klasse `List` niemals `null` ist.

a) Implementieren Sie eine Methode `length` in der Klasse `List`, die die Länge der aktuellen Liste zurückgibt. Verwenden Sie **nur Schleifen** und **keine Rekursion**.

b) Implementieren Sie eine Methode `getMaxSuccessors` in der Klasse `Tree`. Diese Methode soll die höchste Anzahl von Kindern zurückgeben, die ein Knoten im aktuellen Baum hat. Auf ein Blatt angewendet ist das Ergebnis also immer `0`, angewendet auf den in der Grafik dargestellten Baum ist das Ergebnis `4` (da der Knoten mit Wert `89` vier Kinder hat). Verwenden Sie **nur Rekursion** und **keine Schleifen**.

Hinweise:

- Implementieren und verwenden Sie auch eine Hilfsmethode in der Klasse `List`.
- Verwenden Sie auch die Methode `length` der Klasse `List` aus dem ersten Aufgabenteil.
- Sie dürfen davon ausgehen, dass `max(int a, int b)` und `max(int a, int b, int c)` existieren, die den größten der zwei bzw. drei Argument-Werte zurückgibt.

c) Implementieren Sie die Methode `insert` in der Klasse `Tree`. Diese bekommt einen `int`-Wert übergeben und fügt einen neuen Knoten mit diesem Wert in den aktuellen Baum ein, wobei der neue Knoten ein Blatt ist. Der neue Knoten soll als Kind des rechtesten, untersten Blatts des Baums eingefügt werden. Um das rechteste, unterste Blatt eines Baums zu finden, gehen Sie jeweils zum rechtesten Kind und dann „nach unten" zu den Nachfolgern dieses Kinds. Für den Beispielbaum aus der Grafik soll also ein neuer Knoten als Kind des Knotens mit Wert `23` eingefügt werden. Verwenden Sie **nur Rekursion** und **keine Schleifen**.

Hinweise:

- Verwenden Sie die beiden Konstruktoren der Klassen `Tree` und `List`.
- Implementieren und verwenden Sie auch eine Hilfsmethode in der Klasse `List`.

d) Wir betrachten das folgende Interface:

```
public interface Updater {
  public int update(int v);
}
```

Schreiben Sie eine Klasse `Verdoppler`, die das Interface `Updater` implementiert. Die Implementierung der `update`-Methode soll das Doppelte des übergebenen `int`-Werts zurückgeben. Für den Aufruf `x.update(5)` bei einem Objekt `x` der Klasse `Verdoppler` soll die Rückgabe also `10` sein.

e) Implementieren Sie die Methode `apply` in der Klasse `Tree`. Als Argument bekommt diese Methode ein Objekt `x` einer Klasse übergeben, die das Interface `Updater` implementiert. Ihre Implementierung soll die `update`-Methode des Objekts `x` für die `int`-Werte **aller** Knoten des aktuellen Baums aufrufen und die zurückgegebenen Werte in den jeweiligen Knoten speichern.

Wird die Methode `apply` auf dem in der Grafik dargestellten Baum angewendet und ein Objekt der Klasse `Verdoppler` übergeben, enthält der Baum also anschließend die gleichen Knoten, wobei aber sämtliche Werte verdoppelt wurden.

Verwenden Sie **nur Rekursion** und **keine Schleifen**.

Hinweise:

- Implementieren und verwenden Sie auch eine Hilfsmethode in der Klasse `List`.

Lösung:

a)
```
public int length() {
  List cur = this;
  int res = 0;
  while (cur != null) {
    res++;
    cur = cur.next;
  }
  return res;
}
```

Alternativ kann die Methode auch `static` mit explizitem Parameter für die Liste implementiert werden.

b) In der Klasse Tree:

```
public int getMaxSuccessors() {
  if (this.children == null) {
    return 0;
  } else {
    return this.children.getMaxSuccessors();
  }
}
```

In der Klasse List:

```
public int getMaxSuccessors() {
  int nextMaxSucc = 0;
  if (this.next != null) {
    nextMaxSucc = this.next.getMaxSuccessors();
  }
  return max(this.length(), this.first.getMaxSuccessors(), nextMaxSucc);
}
```

Alternative:

In der Klasse Tree:

```
public int getMaxSuccessors() {
  if (this.children == null) {
    return 0;
  } else {
    int childrenLength = this.children.length();
    int childrenMaxSucc = this.children.getMaxSuccessors();
    return max(childrenLength, childrenMaxSucc);
  }
}
```

In der Klasse List:

```
public int getMaxSuccessors() {
  int nextMaxSucc = 0;
  if (this.next != null) {
    nextMaxSucc = this.next.getMaxSuccessors();
  }
  return max(this.first.getMaxSuccessors(), nextMaxSucc);
}
```

c) In der Klasse Tree:

```
public void insert(int v) {
  if (this.children == null) {
    Tree t = new Tree(v);
    this.children = new List(t);
  } else {
    this.children.insert(v);
  }
}
```

In der Klasse List:

```
public void insert(v) {
  if (this.next == null) {
    this.first.insert(v);
  } else {
    this.next.insert(v);
  }
}
```

d)

```
class Verdoppler implements Updater {
  public int update(int v) {
    return v*2;
  }
}
```

e) In der Klasse In der Klasse Tree:

```
public void apply(Updater updater) {
  this.value = updater.update(this.value);
  if (this.children != null) {
    this.children.apply(updater);
  }
}
```

In der Klasse List:

```
public void apply(Updater updater) {
  this.first.apply(updater);
  if (this.next != null) {
    this.next.apply(updater);
  }
}
```

Aufgabe 5 (Haskell): (3 + 3 + (4 + 7) = 17 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 und die Funktion `+` den Typ `Int -> Int -> Int` hat.

i)
```
f x y z = f x (x y) (y + z)
```

ii)
```
g x [] = (x, x)
g x y  = (\z -> g x (x:z)) y
```

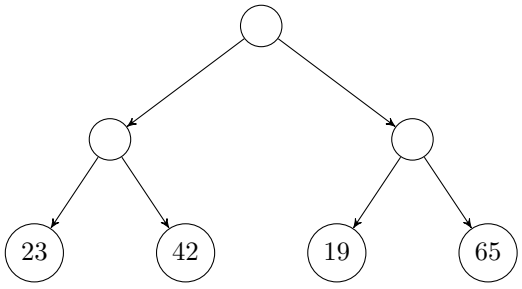
b) Bestimmen Sie, zu welchem Ergebnis die Ausdrücke `i` und `j` jeweils auswerten.

```
i :: [Int]
i = map (\(x, y) -> x + y) (map (\x -> (x, x)) [1,2,3])

j :: Int
j = (\(x:xs) -> x) (filter (\x -> x > 0) [0,-1,1,-2,2])
```

c) Wir betrachten nun die Datenstruktur `Tree a`, die Daten in den Blättern eines Baums speichert:

```
data Tree a = Node (Tree a) (Tree a) | Leaf a
```

Ein Beispiel ist der Baum `Node (Node (Leaf 23) (Leaf 42)) (Node (Leaf 19) (Leaf 65))`:



i) Implementieren Sie die Funktion `toList :: Tree a -> [a]`, die alle Werte im Baum in einer Liste zurückgibt. Für den Baum von oben soll sich also `[23, 42, 19, 65]` ergeben.

Hinweise:

- Sie dürfen beliebige vordefinierte Funktionen verwenden.

ii) Wir betrachten nun einen weiteren Datentyp, um Richtungen zu beschreiben:

```
data Richtung = Links | Rechts
```

Implementieren Sie die Funktion `woMax :: Tree Int -> (Int, [Richtung])`, die das Maximum in einem Baum mit `Int`-Werten bestimmt und dazu den Pfad dorthin als eine Folge von Richtungen ermittelt. So soll das Ergebnis `(i, [Links, Rechts])` bedeuten, dass das Maximum `i` von der Wurzel aus erreicht wird, indem man zuerst den linken und dann den rechten Nachfolger wählt. Das Ergebnis `(i, [])` soll bedeuten, dass das Maximum `i` an der Wurzel zu finden ist. Für unseren Beispielbaum von oben soll `woMax` das Ergebnis `(65, [Rechts, Rechts])` ergeben.

Hinweise:

- Sie dürfen beliebige vordefinierte Funktionen verwenden.

Lösung:

a) i)
```
f :: (Int -> Int) -> Int -> Int -> a
```

ii)
```
g :: a -> [a] -> (a, a)
```

b)
```
i = [2,4,6]
j = 1
```

c) i)
```
toList :: Tree a -> [a]
toList (Leaf x) = [x]
toList (Node l r) = (toList l) ++ (toList r)
```

ii)
```
woMax :: Tree Int -> (Int, [Richtung])
woMax (Leaf x) = (x, [])
woMax (Node l r) = if lmax > rmax then (lmax, Links:lpath)
                                  else (rmax, Rechts:rpath)
  where (lmax, lpath) = woMax l
        (rmax, rpath) = woMax r
```

Aufgabe 6 (Prolog): (2 + 8 + 7 = 17 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) $f(s(X), Y, X)), f(Y, Z, s(Z))$

ii) $g(X, s(Y), c), g(X, X, Y)$

b) Gegeben sei folgendes Prolog-Programm P.

```
p(0, b).
p(X, s(Y)) :- q(X), p(X, b).
q(s(X)) :- q(X).
q(0).
```

Erstellen Sie für das Programm P den Beweisbaum zur Anfrage `?- p(W,Z).` bis zur Höhe 4 (die Wurzel hat dabei die Höhe 1). Markieren Sie Pfade, die zu einer unendlichen Auswertung führen, mit ∞ und alle Knoten, die keine Kinder haben, mit $\lightning$. Geben Sie außerdem alle Antwortsubstitutionen zur Anfrage `?- p(W,Z).` im Graphen an.

c) Fahrpläne von öffentlichen Verkehrsmitteln lassen sich leicht mit Prolog modellieren. Wir definieren hierfür ein fünfstelliges Prädikat `busFahrt`, welches die Fahrpläne der Busgesellschaft beschreibt. Beispielsweise gibt das erste der beiden folgenden Fakten an, dass ein Bus der Linie 4 um 14:20 Uhr vom Driescher Gässchen zum Karlsgraben fährt und dort um 14:24 Uhr ankommt.

```
busFahrt(linie4, 1420, drieschergaesschen, 1424, karlsgraben).
busFahrt(linie33, 1411, ponttor, 1413, drieschergaesschen).
```

Programmieren Sie ein vierstelliges Prädikat `busVerbindung`, welches die möglichen Busverbindungen zwischen zwei Orten beschreibt. Hierbei soll eine Anfrage `busVerbindung(Start,Ziel,Zeit,Verbindung)` genau dann erfolgreich sein, falls `Verbindung` eine mögliche Busverbindung von `Start` nach `Ziel` ab der Uhrzeit `Zeit` darstellt. Beachten Sie hierbei, dass keine Busfahrt angetreten werden kann, deren Abfahrtszeit in der Vergangenheit liegt. Eine Busverbindung wird durch eine Liste von Termen der Form `mit(Linie,Abfahrtszeit,Ziel)` angegeben. Solch ein Term besagt, dass man um `Abfahrtszeit` in `Linie` einsteigen soll und an Haltestelle `Ziel` wieder aussteigen soll. Bei den oben angegebenen Fakten sollten sich dann z. B. die folgenden Antworten auf die jeweiligen Anfragen ergeben.

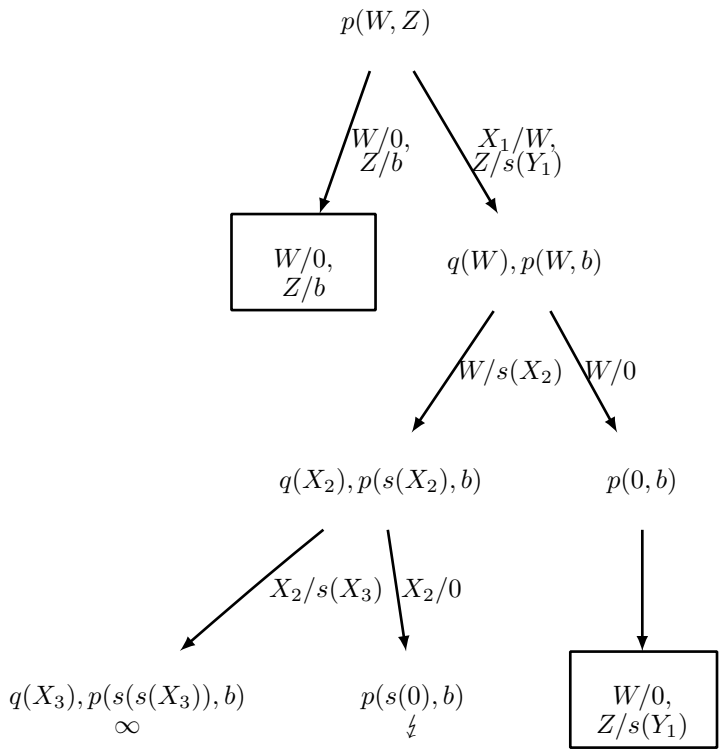
```
?- busVerbindung(ponttor, karlsgraben, 1400, P).
P = [mit(linie33, 1411, drieschergaesschen), mit(linie4, 1420, karlsgraben)] .

?- busVerbindung(ponttor, karlsgraben, 1415, P).
false.
```

Lösung:

a) i) $f(s(X), Y, X)), f(Y, Z, s(Z))$: occur failure

ii) $g(X, s(Y), c), g(X, X, Y)$: $X/s(c), Y/c$

b)



Die Antwortsubstitutionen sind $\{\texttt{W/0}, \texttt{Z/b}\}$ und $\{\texttt{W/0}, \texttt{Z/s}(\texttt{Y}_1)\}$.

c)
```
busVerbindung(0, 0, _, []).
busVerbindung(V, N, S, [mit(L, A, ZZ)|R])
    :-  busFahrt(L, A, V, AZ, ZZ),
        A >= S,
        busVerbindung(ZZ, N, AZ, R).
```