Aufgabe 3 (Listen): (3 + 3 + 6 + 3 + 6 + 15 + 4 + 10 = 50 Punkte)

Verwenden Sie in dieser Aufgabe **keine** vordefinierten Prädikate. Nutzen Sie Prädikate, deren Implementierung in früheren Teilaufgaben gefordert wurde, falls dies sinnvoll ist.

a) Eine Möglichkeit, Listen in Prolog darzustellen, ist die Verwendung eines nullstelligen Funktionssymbols `nil` zur Repräsentation der leeren Liste und eines zweistelligen Funktionssymbols `cons` zur Repräsentation nicht-leerer Listen, wobei das erste Argument von `cons` der in dem aktuellen Listenelement gespeicherte Wert und das zweite Argument von `cons` die Restliste ist. Auf diese Art und Weise kann z.B. die Liste `[1,2,3]` durch den Term `cons(1, cons(2, cons(3, nil)))` dargestellt werden, vgl. Aufgabe 1a).

Implementieren Sie ein Prädikat `userDefinedList(X)`, das genau dann wahr ist, wenn `X` eine derartige mit Hilfe der Funktionssymbole `nil` und `cons` beschriebene Liste ist.

b) Implementieren Sie ein Prädikat `asPrologList(X,Y)`, das genau dann wahr ist, wenn

- `X` eine Liste ist, die die vordefinierte Prolog-Schreibweise für Listen nutzt und
- `Y` eine mit Hilfe der Funktionssymbole `nil` und `cons` (siehe vorheriger Aufgabenteil) beschriebene Liste ist und die gleiche Liste wie `X` beschreibt.

Es soll also beispielsweise `asPrologList([1,2,3], cons(1, cons(2, cons(3, nil))))` gelten.

c) Sie haben bereits die Darstellung der natürlichen Zahlen mittels der Repräsentation `0`, `s(0)`, `s(s(0))`, ... kennengelernt. Schreiben Sie ein dreistelliges Prädikat `maximum(X,Y,Z)`, das auf solchen Zahlen operiert, sodass das Prädikat genau dann erfüllt ist, wenn `Z` das Maximum von `X` und `Y` ist. Beispielsweise gilt also `maximum(s(0),0,s(0))`. Schreiben Sie ein weiteres Prädikat `maximumList(XS,X)`, welches genau dann erfüllt ist, wenn `X` das Maximum der Liste `XS` ist. Beispielsweise gilt `maximumList([s(0),s(s(0)),s(0),0,s(s(s(0)))], s(s(s(0))))`.

d) In dieser Teilaufgabe sollen Sie ein Prädikat `remove(XS, X, YS)` schreiben. Falls `X` in der Liste `XS` mindestens einmal vorkommt, soll `remove(XS, X, YS)` genau dann erfüllt sein, wenn `YS` aus der Liste `XS` entsteht, indem ein Vorkommen von `X` entfernt wird. Kommt `X` nicht in `XS` vor, dann kann sich Ihr Prädikat beliebig verhalten. Zum Beispiel gilt `remove([s(0),s(s(0)),s(0),0,s(s(s(0)))], s(0), [s(s(0)),s(0),0,s(s(s(0)))])` und `remove([s(0),s(s(0)),s(0),0,s(s(s(0)))], s(0), [s(0), s(s(0)),0,s(s(s(0)))])`.

e) Schreiben Sie nun ein zweistelliges Prädikat `sortList(XS,YS)`, welches genau dann erfüllt ist, wenn `YS` die Liste ist, welche aus den Elementen der Liste `XS` besteht und die Elemente dieser Liste absteigend nach Größe sortiert sind. Verwenden Sie hierzu das Prädikat `maximumList`, um zunächst das Maximum von `XS` zu bestimmen und `remove`, um das Maximum aus `XS` zu löschen. Sortieren Sie dann zunächst die dadurch entstehende Liste und fügen Sie schließlich das Maximum wieder vorne ein. Es würde zum Beispiel `sortList([s(0),s(s(0)),s(0),0,s(s(s(0)))], Res)` das Ergebnis `Res = [s(s(s(0))),s(s(0)),s(0),s(0),0]` liefern.

f) Implementieren Sie ein Prädikat `flattenConsecutive(X,Y)`, das genau dann wahr ist, wenn

- `X` eine Liste von Listen ist und
- `Y` eine Liste ist, die entsteht, wenn man alle Elemente von `X` konkateniert. Hierbei dürfen mehrere gleiche, aufeinander folgende Elemente entfernt werden, sodass mindestens eines dieser gleichen Elemente überbleibt. Die Reihenfolge der Elemente der Liste darf nicht weiter verändert werden.

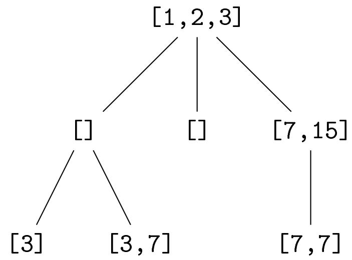
Es gilt also beispielsweise `flattenConsecutive([[1,1,3,2],[],[2,4]],[1,3,2,4])`, `flattenConsecutive([[1,1,3,2],[],[2,4]],[1,1,3,2,4])` und `flattenConsecutive([[1,1,3,2],[],[2,4]], [1,1,3,2,2,4])`.

g) Entwerfen Sie eine Datenstruktur, mit deren Hilfe sich Mehrwegbäume in Prolog darstellen lassen. Ein Mehrwegbaum ist ein Baum, dessen Knoten *beliebig viele Kindknoten* haben können. Darüber hinaus muss jeder Knoten einen beliebigen Wert speichern können. Beschreiben Sie kurz die Bedeutung der von Ihnen zu diesem Zweck verwendeten Funktionssymbole und ihrer Argumente.

h) Implementieren Sie ein Prädikat `flattenConsecutiveTree(X,Y)`, das genau dann wahr ist, wenn

- `X` ein Baum von Listen ist, wobei die von Ihnen im letzten Aufgabenteil entworfene Datenstruktur zur Repräsentation von Bäumen und die vordefinierten Prolog-Listen verwendet werden, und
- `Y` eine Liste ist, die entsteht, indem alle in `X` enthaltenen Listen konkateniert werden. Hierbei dürfen mehrere gleiche, aufeinander folgende Elemente entfernt werden, sodass mindestens eines dieser Elemente überbleibt.

Die Reihenfolge, in der die Listen aus `X` konkateniert werden sollen, ist eine Preorder-Traversierung, vgl. Aufgabe 2c): Am Anfang steht jene Liste, die in der Wurzel von `X` gespeichert ist. Es folgen alle Listen, die in dem durch den ersten Kindknoten definierten Teilbaum gespeichert sind, gefolgt von allen Listen, die in dem durch den zweiten Kindknoten definierten Teilbaum gespeichert sind, usw. Es gilt also beispielsweise `flattenConsecutiveTree(X, [1,2,3,7,15,7])`, wenn `X` der folgende Mehrwegbaum ist:



Hinweise:

- Verwenden Sie das Prädikat `append` aus der Tutoraufgabe 2b).

Aufgabe 5 (Unifikation): (5 · 4 = 20 Punkte)

In dieser Aufgabe sollen allgemeinste Unifikatoren bestimmt werden. Sie sollten diese Aufgabe ohne Hilfe eines Rechners lösen, da Sie zur Lösung von Aufgaben dieses Typs in der Klausur keinen Rechner zur Verfügung haben.
Nutzen Sie den Algorithmus zur Berechnung des allgemeinsten Unifikators (MGU) aus der Vorlesung, um die folgenden Termpaare auf Unifizierbarkeit zu testen.
Geben Sie neben dem Endergebnis σ auch die Unifikatoren $\sigma_1, \sigma_2, \dots, \sigma_n$ für die direkten Teilterme der beiden Terme an. Sollte ein σ_i nicht existieren, so begründen Sie kurz, warum die Unifikation fehlschlägt. Geben Sie in diesem Fall an, ob es sich um einen *clash failure* oder einen *occur failure* handelt.

(i) **f(Z,Y,X)** und **f(g(a,a),g(X,Z),g(a,Y))**

(ii) **f(g(X),Z,a,W)** und **f(W,h(Y,Y),Y,g(b))**

(iii) **h(h(Z,a),h(Z,f(b)))** und **h(h(Z,Z),h(Z,f(Z)))**

(iv) **f(g(X,Y),g(Y,Y))** und **f(g(Y,a), g(a,X))**

(v) **f(g(Z,h(X)),X,g(Y,h(Y)))** und **f(g(h(Y),h(a)),X,g(h(X),h(a)))**

(i)
σ(f(Z, Y, X)) = σ(f(g(a, a), g(X, Z), g(a, Y)))
σ1 = MGU(Z, g(a,a)) = {Z = g(a, a)}
σ2 = MGU(σ1(Y), σ1(g(X, Z))) = MGU(Y, g(X, g(a, a))) = {Y = g(X, g(a, a))}
σ3 = MGU(σ2(σ1(X)), σ2(σ1(g(a, Y))) = MGU(X, g(a, g(X, g(a, a)))) = {X = g(a, g(X, g(a, a)))}
=> OCCUR FAILURE

(ii)
σ(f(f(X), Z, a, W)) = σ(f(W, h(Y, Y), Y, g(b)))
σ1 = MGU(f(X), W) = {W = f(X)}
σ2 = MGU(σ1(Z), σ1(h(Y, Y))) = MGU(Z, h(Y, Y)) = {Z = h(Y, Y)}
σ3 = MGU(σ2(σ1(a)), σ2(σ1(Y))) = MGU(a, Y) = {Y = a}
σ4 = MGU(σ3(σ2(σ1(W))), σ3(σ2(σ1(g(b))))), MGU(f(X), g(b)) = {f(X) = g(b)}
=> σ = {f(x) = g(b), Y = a, Z = h(a, a), W = g(b)}

(iii)
σ(h(h(Z, a), h(Z, f(b)))) = σ(h(h(Z, Z), h(Z, f(Z))))
σ1 = MGU(h(Z, a), h(Z, Z)) = {Z = a}
σ2 = MGU(σ1(h(Z, f(b))), σ1(h(Z, f(Z)))) = MGU(f(b), f(a)) = {a = b}
=> σ = {a = b, Z = b}

(iv)
σ(f(g(X, Y), g(Y, Y))) = σ(f(g(Y, a), g(a, X)))
σ1 = MGU(g(X, Y), g(Y, a)) = {X = Y, Y = a}
σ2 = MGU(σ1(g(Y, Y)), σ1(g(a, X))) = MGU(g(a, a), g(a, a)) = {}
=> σ = {X = Y, Y = a}

(v)
σ(f(g(Z, h(X)), X, g(Y, h(Y)))) = σ(f(g(h(Y), h(a)), X, g(h(X), h(a))))
σ1 = MGU(g(Z, h(X)), g(h(Y), h(a))) = {Z = h(Y), h(X) = h(a)} = {Z = h(Y), X = a}
σ2 = MGU(σ1(X), σ1(X)) = {}
σ3 = MGU(σ1(g(Y, h(Y))), σ1(g(h(X), h(a)))) = MGU(g(Y, h(Y)), g(h(a), h(a)) = {Y = a, Y = h(a)}
=> OCCUR FAILURE

Aufgabe 7 (Beweisbäume): (18 + 2 = 20 Punkte)

Betrachten Sie die Anfrage `?- a(Z,s(0)).` zu folgendem Prolog-Programm:

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

a) Geben Sie den zugehörigen Beweisbaum (SLD-Baum) bis einschließlich Höhe 3 an. Die Höhe eines Baums ist der längste Pfad von der Wurzel bis zu einem Blatt. Ein Baum, welcher nur aus einem Blatt besteht, hat also die Höhe 0. Markieren Sie unendliche Pfade mit ∞ und Fehlschläge mit $(fail)$. Geben Sie alle Antwortsubstitutionen zur obigen Anfrage an und geben Sie an, welche dieser Antwortsubstitutionen von Prolog gefunden werden.

b) Strukturieren Sie das gegebene Programm so in ein logisch äquivalentes Programm um, dass Prolog mit seiner Auswertungsstrategie **mindestens eine** Lösung zur gegebenen Anfrage findet. Der Beweisbaum (SLD-Baum) muss nicht endlich sein! Sie brauchen den SLD Baum nicht angeben.

Hinweis: Bei dieser Umstrukturierung dürfen Sie nur die Reihenfolge der Prolog-Klauseln verändern.

a)

?- a(Z, s(0)).

Z' = s(Z); Y' = Y

(fail)

s(X) = Z; X = s(0)

?- a(s(Z), s(0)).

?- a(s(X), s(0)).
Z = s(s(0))

Z'' = s(Z'); Y'' = Y'

Z = X; X = s(0)

?- a(s(s(Z)), s(0)).

?- a(s(X), s(0)).
Z = s(0)

Z''' = s(Z''); Y''' = Y''

s(Z) = X; X = s(0)

?- a(s(s(s((Z))), s(0)).

?- a(s(X), s(0)).
Z = 0

s(s(Z)) = X; X = s(0)

∞

?- a(s(X), s(0)).
s(Z) = 0
=> bei natürlichen Zahlen nicht möglich
(fail)

Keine dieser Antwortsubstitutionen wird gefunden, da Prolog von links nach rechts mit depth-first search auswertet, also den unendlichen Pfad nimmt.

b)

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

Aufgabe 9 (Arithmetik mit Prolog): (10 Punkte)

Formulieren Sie ein Prolog-Programm mit einem Prädikat `factors(X,Y)`, das wahr ist, wenn `X` ≥ 1 und `Y` die aufsteigende Liste der positiven Teiler von `X` ist. Beispielsweise soll `factors(6,[1,2,3,6])` wahr sein. Benutzen Sie nur die vordefinierten Prädikate `is/2`, `>/2` und `</2` sowie die Funktion `mod` und die Addition.