Aufgabe 2 (Prolog mit Listen und eigenen Datenstrukturen): (1 + 1 + 2.5 + 1 + 3.5 = 9 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.

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

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

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

- `X` eine Liste von Listen ist und
- `Y` jene Liste ist, die entsteht, wenn man alle Element von `X` konkateniert.

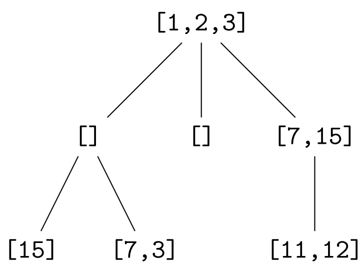
Es soll also beispielsweise `flatten([[1,2,3],[],[3,4]], [1,2,3,3,4])` gelten.

d) 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.

e) Implementieren Sie ein Prädikat `flattenTree(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.

Die Reihenfolge, in der die Listen aus `X` konkateniert werden sollen, ist eine Preorder-Traversierung wie folgt: 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 `flattenTree(X, [1,2,3,15,7,3,7,15,11,12])`, wenn `X` der folgende Mehrwegbaum ist:



Hinweise:

- Verwenden Sie das Prädikat `append` aus der Tutoraufgabe.

Aufgabe 4 (Unifikation): (6 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 auch 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 $\sigma_1, \sigma_2, \ldots, \sigma_n$ 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(X, a, g(X))** und **f(h(Y), Y, g(h(Y)))**

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

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

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

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

(vi) **f(g(X), h(m(Z), Z))** und **f(g(m(Y)), h(X, g(Z))**

Aufgabe 6 (Beweisbäume): (4 + 1 = 5 Punkte)

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

```
q(X, Y) :- q(s(X), Y).
q(0, Y) :- h(s(Y), Y).
q(s(X), X).
h(X, X) :- q(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 Lösungen (Antwortsubstitutionen) zur obigen Anfrage an.

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.

Aufgabe 8 (Arithmetik in Prolog): (4 + 1 = 5 Punkte)

a) Formulieren Sie ein Prolog-Programm mit einem Prädikat `prime(N)`, das wahr ist, wenn `N > 1` und `N` eine Primzahl ist. Benutzen Sie nur die vordefinierten Prädikate `is/2`, `</2`, `>/2` und die Funktion `mod`.

b) Erweitern Sie das Programm um ein Prädikat `only_primes(XS)`, das wahr ist, wenn `XS` eine Liste ist, welche nur Primzahlen enthält. Beispielsweise soll `only_primes([5, 7, 5, 3, 2])` wahr sein.