**Team**: 11, Mesut Koc und Anton Kirakozov

**Aufgabenaufteilung**: Mesut: 1.1) , 1.2), 1,3), 1.8) und 2) (Aufgabe\_1.pl und Dokumentation)

Anton: 1.3), 1.4), 1.5), 1.7) und 2) (Aufgabe\_1.pl und Dokumentation)

**Quellenangaben**: Prologeinführung(.pdf), <http://en.wikibooks.org/wiki/Prolog/Lists>, <http://www.fb10.uni-bremen.de/homepages/hackmack/prolog/texte/kap5.pdf>, Skriptbeispiele von Herrn Klauck.

**Bearbeitungszeitraum**: Mesut: 30.03.15 ca. 1h, Anton: 01.04.15 2h, 05.04.15 ca. 4h, 06.04.15 ca. 6h, 10.04.15 ca. 5h

**Gemeinsame Bearbeitungszeit:** ca. 18 std.

**Aktueller Stand**: Alle Funktionen sind implementiert, soweit gestet und funktionsfähig. Die Dokumentation ist vollständig und überarbeitet.

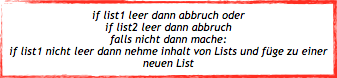
*Konzepte unserer fertigen Aufgaben:*

Aufgabe 1 Teil 1

**1.1**) Wir haben mit der ersten Regel versucht, den Basisfall aufzustellen. Im Grunde genommen versuchen wir zu unifizieren. Sollte die Unifikation bei einem Fakt oder einer Regel fehlschlagen, können wir davon ausgehen, dass die Anfrage nicht erfolgreich sein kann. Wir gehen davon aus, dass eine Liste eine leere Liste sein kann oder mindestens ein Element enthält . Sobald wir von beiden Fakten ausgehen, können wir sichergehen, dass jede Liste identifiziert wird. Darüber hinaus prüfen wir, ob der Rest der Liste eine leere Liste beinhaltet, was bei unserem ersten Fakt mit einem positiven Wahrheitswert belegt wird(true). Um das zu ermöglichen entfernen wir immer das erste Element aus der Liste, solange bis eine leere Liste erkannt wird. Wir arbeiten somit alle Fälle aus und lassen keinen Fakt außen stehen. Dazu haben wir vorerst zwei Pseudocodes aufgestellt:



**1.2**) Wir haben zwei Listen die unterschiedlich lang sein können. Falls die erste Liste leer ist kann keine dritte Liste entstehen und es bleibt einfach nur die Liste 2. Die kann allerdings auch leer sein. Wir können den Inhalt der beiden Listen, ob Integer oder Char zu einer neuen Liste ohne Probleme erzeugen. Das Suchverfahren vom Inhalt der Liste gestaltet sich Rekursiv.



**1.3**) Wir nehmen an, dass es zwei Listen gibt und die sind wie folgt definiert: Liste 1, Liste 2. Liste 1 ist also dann ein Infix von Liste 2, wenn die Elemente, die sich in Liste 1 befinden in der gleichen Reihenfolge sortiert sind wie in der zweiten Liste. Zusätzlichen dürfen sich vor und nach “L1” weitere Elemente befinden. Also bedeutet dies, dass die Liste 1 weder Präfix noch Suffix der Liste 2 sein darf.

Ein Beispiel um das zu verdeutlichen.

**Liste 2 = [1,2,3,4]**

**Liste 1 = [2,3]**

d.h also **L1** ist infix **L2** → **True**

In unserer ersten Regel gehen wir schon davon aus bzw. stellen sicher, dass das Element schon nicht an der ersten Stelle ist, und teilen dann Rekursiv unsere Listen auf, bis das angegebene Element nicht an der vordersten Stelle und hintersten Stelle nachdem Element auftaucht. Dabei geht er immer wieder rekursiv alles durch. Unser Pseudocode dazu:



**1.4**) Übernehmen wir einfach wieder die beiden Listen L1 und L2. L1 ist also dann ein Suffix von L2, wenn die Elemente der Liste L1 sortiert am ende der Liste L2 stehen. Schauen wir uns wieder ein Beispiel an.

**Liste 2 = [1,2,3,4]**

**Liste 1 = [3,4]**

also **L1** suffix **L2** → **True**

**1.5**) Wir behalten unsere beiden Listen und widmen uns nun dem Präfix. L1 ist genau dann ein Präfix von L2, wenn die Elemente aus L1 in der richtigen Reihenfolge am Anfang der Liste L2 stehen.

Beispiel:

**Liste 2 = [1,2,3,4]**

**Liste 1 = [1,2]**

also **L1** präfix **L2** → **True**

**1.6**)Wir bauen eine Liste, die entweder gerade oder ungerade ist. In dieser Liste können weitere Listen vorkommen, die genau auf die gleich Eigenschaften geprüft werden sollen. Dabei können in der Liste weitere Elemente vorkommen die nicht als Liste definiert werden.

Wir möchten es anhand eines Beispiels mehr verdeutlichen.

Beispiel: Liste1= **[** [] , [1,2] , [a,b,c] **]**

Die leere Liste und die Liste mit den Elementen 1 und 2 gelten als gerade, wohingegen die Liste mit den Elementen a,b,c und die „äußere“ Liste1 mit den **fetten** Klammern ungerade sind.

**1.7**)Für das Löschen eines Elementes aus einer Liste , ist es erforderlich die gesamte Liste durchzulaufen und nachdem Element zu suchen , welches gelöscht werden soll. Dabei ist es wichtig , dass das zu löschende Element mit dem Element aus der Liste verglichen wird damit kein falsches Ergebnis entsteht.

Nehmen wir uns ein Beispiel:

**Liste =[1,2,2,3]** das Element was gelöscht werden soll **E=2**

[1,2,2,3] Am anfang wird überprüft ob das erste Element dem E gleicht

**1(!=E)** Der Rest der Liste wird gebildet und lautet **[2,2,3]**. Da die 1 in dem Fall

nicht gelöscht wurde, wird sie „gemerkt“. Die Liste wird solange durchlaufen, bis keine Elemente mehr zur Verfügung stehen. Das Endergebnis lautet dann **[1,3]**.

**1.8**) In dieser Aufgabe soll das Element 1 in einer Liste gegen Element 2 vertauscht werden und in einer neuen Liste ausgegeben werden. Wir durchlaufen die Liste und vergleichen die Elemente mit dem Kopf der Liste, falls die Elemente gleich sind wird nichts weiter unternommen, sobald es keine Übereinstimmung gibt, wird das Element ersetzt.

**Aufgabe 1 Teil 2**

**2.1**) Beweis der Gültigkeit der Funktion für eine beliebige natürliche Zahl mittels Induktion:

**nat(0)** Induktionsanfang.

**nat(s(X))** Induktionsbehauptung.

:- **nat(N)** Induktionsschritt.

s^n+1(0) → s(s^n(0)) = s(n) Ist unsere Induktionsbehauptung.

s(n)= 1 + n = n + 1 Induktionsschritt

**2.2.1**) In dieser Aufgabe möchten wir jede natürliche Zahl, die größer als 0 sein muss in eine S-Zahl umwandeln(S-Struktur).

**2.2.2**) In dieser Aufgabe sollen wir zwei S-Zahlen addieren. Wichtig für die Addition ist, dass die zu addierenden Zahlen, Elemente der natürlichen Zahlen sind. Im Ergebnis werden genauso S-Zahlen erwartet.

**2.2.3**) In der Aufgabe bei der die Subtraktion gefragt ist, gehen wir auch davon aus , dass die Elemente die angegeben werden auch aus dem Bereich der natürlichen Zahlen kommen. Des Weiteren darf der Subtrahend nicht größer als der Minuend sein. Diesen Fall decken wir ab, indem wir die beiden Elemente mit einander auf kleiner/größer vergleichen. Bei einer nicht erlaubten Subtraktion wird der Wert 0 geliefert.

**2.2.4**)Wir möchten nun zwei natürliche Zahlen(S-Zahlen) miteinander multiplizieren.

Beispiel : **2\*2=4**.

Praxisnah bezogen : **(s(s(0)),s(s(0)),Y). Y = s(s(s(s(0))))**.

**2.2.5**) Wir berechnen Potenzen von Zahlen. Man kann einen beliebigen Exponenten wählen und eine beliebige Basis. Betrachten wir nun folgendes Beispiel:

**2^5 = 32**. Dies bedeutet, dass die Zahl 2, 5mal mit sich selber multipliziert wird, sodass wir folgendes schreiben können: **2\*2\*2\*2\*2= 32**. Aus diesem Grund kann man bei der Funktion auch einfache Multiplikation als Hilfsfunktion anwenden.

**2.2.6**) Mit Hilfe der Fakultätsfunktion bilden wir das Produkt einer natürlichen Zahl 1 bis zu einer natürlichen Zahl **N**. N!= 1\*2\*3\*...\*N. Wir zerlegen also unsere gewünschte Zahl und führen eine Multiplikation durch.

Beispiele : **1! = 1** , **2! = 2**, **3!= 6**, **4!= 24**, **5! = 120**. usw.

**2.2.7**) Wir möchten nun in dieser Aufgabe prüfen, ob Zahl\_1 kleiner Zahl\_2 ist.

Beispiele : **1 < 2** → **True**, **2 < 1** → **False**.

**2.2.8**)Mit unserer Funktion berechnen wir den Modulo aus zwei Zahlen, dabei „dividieren“ wir diese und speichern den erhaltenen Rest ab.

Beispiel: **7 : 3 = 2 Rest 1** , **2 : 3 = 0 Rest 2** , **3 : 3 = 1 Rest 0**

**2.2.9)** In dieser Aufgabe soll die S-Struktur in eine natürliche Zahl umgewandelt werden. Mit s2nat(..) können wir dann S-Zahlen wieder in natürliche Zahlen umwandeln.