



Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

# **Referat**

Ferdinand Trendelenburg

Listen und Arithmetik

# **Ferdinand Trendelenburg**

Listen und Arithmetik

Referat eingereicht im Rahmen der Vorlesung Logik und Berechenbarkeit

im Studiengang Angewandte Informatik (AI)  
am Department Informatik  
der Fakultät Technik und Informatik  
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. C. Klauck

Abgegeben am 5.11.2018

# Inhaltsverzeichnis

<b>1</b>	<b>Was ist eigentlich eine Liste? .....</b>	<b>5</b>
<b>2</b>	<b>Strategien Listen.....</b>	<b>6</b>
2.1	Is_a_list(liste) .....	6
2.1.1	Allgemein: .....	6
2.1.2	Strategie: .....	6
2.2	diffList(liste1, liste2, ergebnissliste).....	7
2.2.1	Allgemein: .....	7
2.2.2	Strategie: .....	7
2.3	praefix(praefix-liste, liste) .....	7
2.3.1	Allgemein: .....	7
2.3.2	Strategie: .....	7
2.4	suffix(suffix-liste, liste) .....	8
2.4.1	Allgemein: .....	8
2.4.2	Strategie: .....	8
2.5	infix(infix-liste, liste) .....	8
2.5.1	Allgemein: .....	8
2.5.2	Strategie: .....	8
2.6	eo_count(liste, even, odd) .....	9
2.6.1	Allgemein: .....	9
2.6.2	Strategie: .....	9
2.7	del_element(position, element, liste, returnList).....	9
2.7.1	Allgemein: .....	9

2.7.2	Strategie: .....	9
2.8	substitute (position, element, neuelement, liste, returnList).....	10
2.8.1	Allgemein: .....	10
2.8.2	Strategie: .....	10
<b>3</b>	<b>Strategie: Arithmetik .....</b>	<b>11</b>
3.1	nat2s(nat, s).....	11
3.1.1	Allgemein: .....	11
3.1.2	Strategie: .....	11
3.2	s2nat(s, nat).....	12
3.2.1	Allgemein: .....	12
3.2.2	Strategie: .....	12
3.3	add(s1, s2, ergebnis) .....	12
3.3.1	Allgemein: .....	12
3.3.2	Strategie: .....	12
3.4	sub(s1, s2, ergebnis).....	13
3.4.1	Allgemein: .....	13
3.4.2	Strategie: .....	13
3.5	mul(s1, s2, ergebnis) .....	13
3.5.1	Allgemein: .....	13
3.5.2	Strategie: .....	13
3.6	power(s1, s2, ergebnis).....	14
3.6.1	Allgemein: .....	14
3.6.2	Strategie: .....	14
3.7	fac(s, ergebnis) .....	14
3.7.1	Allgemein: .....	14
3.7.2	Strategie: .....	14
3.8	lt(s1, s2) .....	15
3.8.1	Allgemein: .....	15
3.8.2	Strategie: .....	15
3.9	mods(s1, s2, ergebnis) .....	15
3.9.1	Allgemein: .....	15
3.9.2	Strategie: .....	15

# 1 Was ist eigentlich eine Liste?

Eine Liste ist eine Aneinanderkettung von Daten, die in Prolog eine feste Struktur haben. Sie ist die Referenz auf das erste Listenelement der Liste. Bei einer Liste:

[1,2,3,4,5,6] wäre die "1" das erste Listenelement und würde so aussehen: "[Wert| [Schwanz der Liste]]".

Bei diesem Beispiel hält der "Wert" den Wert des aktuellen Listenelements also die "1" und "Schwanz der Liste" das nächste Listenelement, welches die gleiche Struktur besitzt, wie das aktuelle Listenelement, welches wir betrachten.

Das Ende der Liste wird durch ein leeres Listenelement realisiert, das folgendermaßen aussieht: "[]"

Das heißt, wenn ich die Liste, die wir betrachten, in der originalen Schreibweise, wie sie Prolog intern verarbeitet, aufschreibe, wird aus der Liste: "[1,2,3,4]" die Liste: "[1, [2, [3, [4, []]]]]"

Beide Schreibweisen sind in Prolog implementiert und können beide genutzt werden, nur eignet sich die Schreibweise: "[1, []]" deutlich besser zur Verarbeitung durch eine Rekursion oder Fallunterscheidungen.

## 2 Strategien Listen

### 2.1 Is\_a\_list(liste)

#### 2.1.1 Allgemein:

In diesem Prädikat wird rekursiv durch die „liste“ gegangen, und in jedem Rekursionsschritt wird geprüft, ob es sich bei dem aktuellen Element um ein Listen-Element handelt.

In Java sieht die Implementierung ein wenig anders aus, da Java eine stark typisierte Programmiersprache ist. Hier ist geprüft, ob es sich um eine Liste handelt, indem wir das erste Listen-Element prüfen. Wenn es ein Listen-Element ist, sind alle nächsten Elemente auch Listen-Elemente, da im Konstruktor nur ein Listen-Element als „next“ parametrisiert werden kann.

#### 2.1.2 Strategie:

1. Ist das aktuelle Element ein Listen-Element?
  - a. Ja: gehe zu Schritt 2
  - b. Nein: brich ab und gib zurück, dass es sich um keine Liste handelt.
2. Ist das aktuelle Element ein leeres Listen-Element?
  - a. Ja: beende und gib zurück, dass es sich um eine Liste handelt.
  - b. Nein: geh zu schritt 1 mit dem nächsten Listen-Element.

## 2.2 diffList(liste1, liste2, ergebnisliste)

### 2.2.1 Allgemein:

In diesem Prädikat werden die Listen-Elemente, die in "liste1" vorkommen aber nicht in "liste2", ermittelt und in der Liste: „ergebnisliste“ zurückgegeben.

### 2.2.2 Strategie:

1. Ist die erste Liste („liste1“) eine leere Liste?
  - a. Ja: so gib die "ergebnisliste" zurück.
  - b. Nein: geh zu Schritt 2 mit dem ersten Element von der „liste1“
2. Kommt der Wert des Listen-Elements in der Liste „liste2“ vor?
  - a. Ja: geh zu Schritt 1 mit dem nächsten Element der „liste1“
  - b. Nein: schreib das Element in die „ergebnisliste“ und geh mit dem nächsten Listen-Element zu Schritt 1.

## 2.3 praefix(praefix-liste, liste)

### 2.3.1 Allgemein:

In diesem Prädikat wird ermittelt, ob die ersten Listen-Elemente der "liste" gleich der "praefix-liste" sind. Die "praefix-liste" ist hierbei selber eine Liste.

### 2.3.2 Strategie:

1. Die "praefix-liste" darf nicht Leer sein
2. Ist der Wert des ersten Listen-Elements der "liste" gleich dem des ersten Listen-Elements der "praefix-liste"?
  - a. ja: Geh zu Schritt 2 mit dem nächsten Element der "praefix-liste" und dem nächsten Listen-Element der der Liste: "liste"
  - b. Nein: Ist das erste Listen-Element der "praefix-liste" ein leeres Listen-Element und das erste Listen-Element der "liste" kein leeres Listen-Element?
    - i. Ja: beende und gib zurück, dass es sich bei der "praefix-liste" um das Präfix der Liste "liste" handelt.
    - ii. Nein: beende und gib zurück, dass es sich bei der "praefix-liste" nicht um das Präfix der Liste "liste" handelt.

## 2.4 suffix(suffix-liste, liste)

### 2.4.1 Allgemein:

In diesem Prädikat wird in der Liste: „liste“ gesucht, ob die letzten Listen-Elemente der Liste: „liste“ gleich der "suffix-liste" ist. Die "suffix-liste" ist hierbei selber eine Liste.

### 2.4.2 Strategie:

1. dreh die „suffix-liste“ um
2. dreh die „liste“ um
3. geh zu praefix() mir den umgedrehten Listen

## 2.5 infix(infix-liste, liste)

### 2.5.1 Allgemein:

In diesem Prädikat wird in der Liste: „liste“ gesucht, ob in der Liste: „liste“ die Reihenfolge der Listen-Elemente der Liste: „infix-liste“ vorkommt. Diese darf weder ganz am Anfang, noch ganz am Ende der Liste: „liste“ stehen.

### 2.5.2 Strategie:

1. die "infix-liste" darf nicht leer sein
2. überspringe den ersten Wert der „liste“
3. ist „liste“ leer, so ist die „infix-liste“ kein Infix der „liste“.
4. Ist der Wert des ersten Listen-Elements der "liste" gleich dem des ersten Listen-Elements der "infix-Liste"?
  - a. Ja: geh zu schritt 3 mit dem nächsten Element der „infix-Liste" und der "liste"
  - b. nein: ist das erste Listen-Element der "infix-Liste" ein leeres Listen-Element und das erste Listen-Element der "liste" kein leeres Listen-Element
    - i. ja: beende und gib zurück, dass es sich bei der "infix-liste" um das Suffix der Liste: "liste" handelt.
    - ii. nein: Geh zu Schritt 3 mit dem nächsten Element von „liste“ und dem ersten Element der Liste "infix-liste".



## 2.6 eo\_count(liste, even, odd)

### 2.6.1 Allgemein:

Das Prädikat eo\_count zählt, wie viele Listen in der "liste" gerade Länge haben und wie viele ungerade Länge besitzen. Die Länge der "liste" wird dabei auch berücksichtigt.

### 2.6.2 Strategie:

1. ist „liste“ eine leere Liste?
  - a. Ja: ist das Ende der Liste erreicht: gib „even“ und „odd“ zurück
  - b. Nein: Ist das erste Element von „liste“ eine Liste(Subliste)?
    - i. Ja: Ist „liste“ eine gerade Liste?
      1. Ja: inkrementiere „even“
      2. Nein: inkrementiere „odd“
      3. geh zu Schritt 1 mit dem nächsten Element von „liste“
      4. geh zu Schritt 1 mit der Subliste von „liste“
    - ii. Nein: geh zu schritt 1 mit dem nächsten Element von „liste“

## 2.7 del\_element(position, element, liste, returnList)

### 2.7.1 Allgemein:

In diesem Prädikat wird das Element mit dem Wert: „element“ gelöscht. Das passiert an der Stelle, die der Wert: „position“ vorgibt. Wenn in „position“ „a“ steht, so werden alle Listen-Elemente mit dem Wert: „element“ gelöscht, bei „e“ nur das erste und bei „l“ nur das letzte Element.

### 2.7.2 Strategie:

1. Wenn die Position = „l“ ist
  - a. dreh die Liste um
  - b. schreib in Position: „e“
  - c. geh mit der umgedrehten Liste zu Schritt 2.
2. Wenn die Position = „e“ ist und der Wert des ersten Elementes der Liste „liste“ gleich dem „element“
  - a. Füge der „returnList“ den Rest der Liste „liste“ hinzu. (ohne das aktuelle Element)

3. Wenn die Position = „a“ ist und der Wert des ersten Elements der Liste „liste“ gleich dem „element“ ist:
  - a. Geh zu Schritt 3 mit dem nächsten Element der Liste
4. SONST: füg der Liste: „returnList“ das erste Element der „liste“ hinzu und gehe zu Schritt 1 mit dem Rest der „liste“.
5. Brich ab, wenn die „liste“ leer ist

## 2.8 substitute(position, element, newelement, liste, returnList)

### 2.8.1 Allgemein:

Im Prädikat wird das Element mit dem Wert: „element“ durch das Element „newelement“ ersetzt. Das passiert an der Stelle, die der Wert: „position“ vorschreibt. Wenn in „position“ „a“ steht, so werden alle Listen-Elemente mit dem Wert: „element“ ersetzt, bei „e“ nur das Erste und bei „l“ nur das letzte Element.

### 2.8.2 Strategie:

1. Wenn die Position = „l“ ist
  - a. dreh die Liste um
  - b. schreib in Position: „e“
  - c. geh mit der umgedrehten Liste zu schritt 2.
2. Wenn die Position = „e“ ist und der Wert des ersten Elementes der Liste „liste“ gleich dem „element“
  - a. Füge der „returnList“ erst das Element „newelement“ und dann Rest der Liste „liste“ hinzu (ohne das aktuelle Element)
3. Wenn die Position = „a“ ist und der Wert des ersten Elementes der Liste „liste“ gleich dem „element“ ist:
  - a. Füg der Liste „returnList“ das Element „newelement“ hinzu.
  - b. Geh zu schritt 3. mit dem nächsten Element der Liste
4. SONST: füg der Liste: „returnList“ das erste Element der Liste „liste“ hinzu und geh zu schritt 1. mit dem Rest der Liste „liste“.
5. brich ab, wenn Die Liste „liste“ Leer ist

## 3 Strategie: Arithmetik

Die S-Zahlen sind so aufgebaut: eine „0“ wird als 0 dargestellt. Die Zahl „1“ hingegen sieht ein wenig anders aus als die numerische Schreibweise: Die numerische Ziffer „1“ wird in meiner Realisierung so dargestellt: „S(0)“. Für jede höhere Zahl kommt ein „S“ dazu. Das bedeutet eine „2“ ist in der S-Zahlen Schreibweise: „S(S(0))“. Das lässt sich beliebig erweitern.

### 3.1 nat2s(nat, s)

#### 3.1.1 Allgemein:

Das Prädikat: nat2s ermittelt aus einer natürlichen Zahl eine S-Zahl, indem sie rekursiv arbeitet. In jedem Rekursionsschritt wird die natürliche Zahl um eins dekrementiert und die S-Zahl um ein S() erweitert.

#### 3.1.2 Strategie:

1. Ist „nat“ = 0?
  - a. Ja – brich ab und gib „s“ zurück.
  - b. Nein – geh zu Schritt 1 und dekrementiere „nat“ um eins und erweitere die „s“ um ein „S()“

## 3.2 s2nat(s, nat)

### 3.2.1 Allgemein:

Das Prädikat: s2nat ermittelt aus einer S-Zahl eine natürliche Zahl, indem sie rekursiv arbeitet. In jedem Rekursions-Schritt wird die natürliche Zahl um eins Inkrementiert und die S-Zahl um ein S() verkleinert.

### 3.2.2 Strategie:

1. Ist „s“ = 0?
  - a. Ja – brich ab und gib „nat“ zurück.
  - b. Nein – gehe zu Schritt 1 und inkrementiere „nat“ um eins und verringere die „s“ um ein „S()“

## 3.3 add(s1, s2, ergebnis)

### 3.3.1 Allgemein:

Dieses Prädikat fügt der S-Zahl „s1“, die S-Zahl „s2“ hinzu. Das wird über eine Rekursion realisiert.

### 3.3.2 Strategie:

1. Ist „s1“ != „0“
  - a. Ja – geh zu Schritt 1 mit „ergebnis“ um ein „S()“ inkrementiert und „s1“ um ein „S()“ dekrementiert
  - b. Nein – geh zu Schritt 2.
2. Ist „s2“ != „0“
  - a. Ja – geh zu Schritt 2 mit „ergebnis“ um ein „S()“ inkrementiert und „s2“ um ein „S()“ dekrementiert
  - b. Nein – geh zu Schritt 3.
3. Sonst: sind „s1 und “ „s2“ == „0“  
gib „ergebnis“ zurück.

### 3.4 sub(s1, s2, ergebnis)

#### 3.4.1 Allgemein:

Dieses Prädikat zieht von der S-Zahl „s1“, die S-Zahl „s2“ ab. Das wird über eine Rekursion realisiert.

#### 3.4.2 Strategie:

1. Ist „s1“ != „0“
  - a. Ja – geh zu Schritt 1 mit „ergebnis“ um ein „S()“ inkrementiert und „s1“ um ein „S()“ dekrementiert
  - b. Nein – geh zu Schritt 2.
2. Ist „s2“ != „0“
  - a. Ja – Ist „s1“ GLEICH „0“
    - i. Ja – abbrechen und „ergebnis“ zurückgeben. („ergebnis“ kann nicht kleiner werden als „0“)
    - ii. Nein - geh zu Schritt 2 mit „ergebnis“ um ein „S()“ dekrementiert und mit „s2“ um ein „S()“ dekrementiert
  - b. Nein – geh zu Schritt 3.
3. Sonst: (sind „s1 und “ „s2“ == „0“)  
Gib „ergebnis“ zurück.

### 3.5 mul(s1, s2, ergebnis)

#### 3.5.1 Allgemein:

Dieses Prädikat multipliziert die S-Zahl „s1“, mit der S-Zahl „s2“. Das wird über eine Rekursion realisiert unter Gebrauch von add(s1, s2, ergebnis).

#### 3.5.2 Strategie:

1. Ist „s1“ oder „s2“ == 0? dann ist das „ergebnis“ = 0
2. Ist „s2“ != „0“?
  - a. Ja – gehe zu Schritt 2 mit dem „ergebnis“ = add(ergebnis, s1, ergebnis) und s2 um ein „S()“ dekrementiert
  - b. Nein – brich ab und gib „ergebnis“ zurück.

### 3.6 power(s1, s2, ergebnis)

#### 3.6.1 Allgemein:

Dieses Prädikat rechnet die S-Zahl "s1", hoch die S-Zahl "s2". Das wird über eine Rekursion realisiert unter Gebrauch von mul(s1, s2, ergebnis).

#### 3.6.2 Strategie:

1. Ist "s1" oder "s2" == 0 dann ist das „ergebnis“ = 1
2. Ist „s2“ != „0“
  - c. Ja – geh zu Schritt 2 mit dem „ergebnis“ = mul(ergebnis, s1, ergebnis) und s2 um ein „S()“ dekrementiert.
  - d. Nein – brich ab und gib „ergebnis“ zurück.

### 3.7 fac(s, ergebnis)

#### 3.7.1 Allgemein:

Dieses Prädikat rechnet das der S-Zahl "s" aus. Das wird über eine Rekursion realisiert. In jedem Rekursionsschritt wird der aktuelle Wert von "s" mit "ergebnis" multipliziert(mul())

#### 3.7.2 Strategie:

1. Ist "s" == 0 dann ist das „ergebnis“ = 0
2. Ist „s“ != „0“
  - a. Ja – geh zu Schritt 2 mit dem „ergebnis“ = mul(ergebnis, s, ergebnis) und dekrementiere „S()“ um ein „S()“.
  - b. Nein – brich ab und gib „ergebnis“ zurück.

### 3.8 $lt(s1, s2)$

#### 3.8.1 Allgemein:

Dieses Prädikat ermittelt mittels Rekursion, ob die S-Zahl "s1" kleiner als "s2" ist.

#### 3.8.2 Strategie:

1. Ist „s1“ != „0“
  - a. Ja – geh zu Schritt 1. Mit „s1“ und „s2“ um ein „S()“ dekrementiert
  - b. Nein – Ist „s2“ != „0“
    - i. Ja - dann gib wahr zurück
    - ii. Nein - dann gib falsch zurück

### 3.9 $mods(s1, s2, ergebnis)$

#### 3.9.1 Allgemein:

Dieses Prädikat ermittelt mittels Rekursion, den Modulowert von "s1" % "s2"

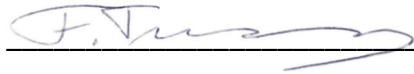
#### 3.9.2 Strategie:

1. Ist "s2" kleiner als "s1"?
  - a. Ja – dann gib das "s2" zurück
  - b. Nein – dann geh zu Schritt 2 Mit "s1" = "ergebnis" von = sub(s1, s2, ergebnis)

## Erklärung zur schriftlichen Ausarbeitung des Referates

*Hiermit erkläre ich, dass ich diese schriftliche Ausarbeitung meines Referates selbstständig und ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe sowie die aus fremden Quellen (dazu zählen auch Internetquellen) direkt oder indirekt übernommenen Gedanken oder Wortlaute als solche kenntlich gemacht habe. Zudem erkläre ich, dass der zugehörige Programmcode von mir selbstständig implementiert wurde ohne diesen oder Teile davon von Dritten im Wortlaut oder dem Sinn nach übernommen zu haben. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.*

Hamburg, den 05.11.2018

A handwritten signature in blue ink, appearing to be 'F. Thun', is written over a horizontal line.