****

Ferdinand Trendelenburg

Listen und Arithmetik

**Referat**

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

**Ferdinand Trendelenburg**

Listen und Arithmetik

Inhaltsverzeichnis

[1 Was ist eigentlich eine Liste? 4](#_Toc528770810)

[2 Implementierungsstrategien 5](#_Toc528770811)

[2.1 Is\_a\_list(liste) 5](#_Toc528770812)

[2.1.1 Allgemein: 5](#_Toc528770813)

[2.1.2 Strategie: 5](#_Toc528770814)

[2.2 diffList(liste1, liste2, ergebnissliste) 5](#_Toc528770815)

[2.2.1 Allgemein: 5](#_Toc528770816)

[2.2.2 Strategie: 5](#_Toc528770817)

[2.3 praefix(praefix-liste, liste) 6](#_Toc528770818)

[2.3.1 Allgemein: 6](#_Toc528770819)

[2.3.2 Strategie: 6](#_Toc528770820)

[2.4 suffix(suffix-liste, liste) 7](#_Toc528770821)

[2.4.1 Allgemein: 7](#_Toc528770822)

[2.4.2 Strategie: 7](#_Toc528770823)

[2.5 infix(infix-liste, liste) 8](#_Toc528770824)

[2.5.1 Allgemein: 8](#_Toc528770825)

[2.5.2 Strategie: 8](#_Toc528770826)

[2.6 eo\_count(Liste, Even, Odd) 8](#_Toc528770827)

[2.7 del\_element(position, element, liste, returnList) 8](#_Toc528770828)

[2.7.1 Allgemein: 8](#_Toc528770829)

1. Was ist eigentlich eine Liste?

Eine Liste ist eine Aneinander-kettung 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 folgenermaßen 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 Rekursion und Fallunterscheidung.

1. Strategien Listen
   1. Is\_a\_list(liste)
      1. Allgemein:

In diesem Prädikat gehen wir Rekursiv durch die „liste“, und in jedem Rekursionsschritt prüfen wir, ob es sich bei dem Aktuellen Element um ein Listen-Element handelt.

* + 1. Strategie:

1. ist das aktuelle Element ein Listen-Element?

ja: gehe zu schritt 2

nein: breche ab und gebe zurück, dass es sich um keine Liste handelt.

2. ist das aktuelle Element ein leeres Listen-Element?

ja: beende und gebe zurück, dass es sich um eine Liste handelt.

nein: gehe zu schritt 1 mit dem nächsten Listen-Element.

* 1. diffList(liste1, liste2, ergebnissliste)
     1. Allgemein:

In diesen Prädikat ermitteln wir die Listen-Elemente, die in "liste1" vorkommen aber nicht in "liste2" und geben diese Listen-Elemente in der „ergebnissliste“ zurück.

* + 1. Strategie:

1. Ist die erste Liese Leer?

Ja - so gib die leere Liste als rückgabewert zurück.

Nein – gehe zu schritt 2. Mit dem ersten Element von der „Liste1“

1. Kommt der Wert des Listen-Elements in der Liste „liste2“ vor?

Ja – gehe zu schritt 1. Mit dem nächsten Element der „liste1“

Nein – schreibe das Element in die „ergebnissliste“ und gehe mit dem nächsten Listen-Element zuschritt 1.

* 1. praefix(praefix-liste, liste)
     1. Allgemein:

In diesem Prädikat suchen wir in der Liste, ob die ersten Listen-Elemente der "liste" gleich der "praefix-Liste" ist. Die "praefix-Liste" ist hierbei selber eine Liste.

* + 1. Strategie:

1. Die "praefix-liste" darf nicht Leer sein
2. Ist der Wert des Erste Listen-Elements der "Liste" gleich dem des ersten Listen- Elements der "praefix-Liste"?

- ja: gehe zu Schritt 2 mit dem nächsten Element der "praefix-Liste" und der "Liste"

- nein: ist das Erste Listen-Element Der "Praefix-Liste" ein leeres Listen- Element und das erste Listen-Element der "Liste" kein Leeres Listen- Element

- ja: beende und gebe zurück, dass es sich bei der "praefix-liste" um das Präfix der Liste "Liste" handelt.

- nein: beende und gebe zurück, dass es sich bei der "praefix-liste" nicht um das Präfix der Liste "Liste" handelt.

* 1. suffix(suffix-liste, liste)
     1. Allgemein:

In diesem Prädikat suchen wir in der Liste, ob die letzten Elemente der Liste gleich der "Suffix-Liste" ist. Die "Suffix-Liste" ist hierbei selber eine Liste.

* + 1. Strategie:

1. Die "Suffix-Liste" liste darf nicht Leer sein

2. überspringe den ersten Wert der Liste „liste“

3. ist die Liste „liste“ leer, so ist die Liste „suffix-liste“ kein Suffix der Liste „liste“.

4. Ist der Wert des ersten Listen-Elements der "Liste" gleich dem des ersten Listen- Elements der "Suffix-Liste"?

- ja: gehe zu schritt 3 mit dem nächsten Element der "Suffix-Liste" und der "Liste"

- nein: ist das Erste Listen-Element Der "Suffix-Liste" ein leeres Listen-Element und das erste Listen-Element der "Liste" auch ein leeres Listen-Element

- ja: beende und gebe zurück, dass es sich bei der "Suffix-Liste" um das Suffix der Liste "Liste" handelt.

- nein: Gehe zu Schritt 3 mit dem nächsten Element von der Liste „liste“ und dem ersten Element Der Liste “praefix-liste“.

* 1. infix(infix-liste, liste)
     1. Allgemein:

In diesem Prädikat suchen wir in der Liste, ob in der Liest „liste“ die Reinfolge der Listen-Elemente der Liste „infix-liste“ vorkommt. Diese darf weder ganz am Anfang als auch ganz am Ende der Liste „liste“ stehen.

* + 1. Strategie:

1. Die "infix-Liste" liste darf nicht Leer sein

2. überspringe den ersten Wert der Liste „liste“

3. ist die Liste „liste“ leer, so ist die Liste „infix-liste“ kein Infix der Liste „liste“.

4. Ist der Wert des ersten Listen-Elements der "Liste" gleich dem des ersten Listen- Elements der "infix-Liste"?

- ja: gehe zu schritt 3 mit dem nächsten Element der „infix-Liste" und der "Liste"

- nein: ist das Erste Listen-Element Der "infix-Liste" ein leeres Listen-Element und das erste Listen-Element der "Liste" kein leeres Listen-Element

- ja: beende und gebe zurück, dass es sich bei der "infix-Liste" um das Suffix der Liste "Liste" handelt.

- nein: Gehe zu Schritt 3 mit dem nächsten Element von der Liste „liste“ und dem ersten Element Der Liste “infix-liste“.

* 1. eo\_count(liste, even, odd)
     1. Allgemein:

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

* + 1. Strategie:
  1. del\_element(position, element, liste, returnList)
     1. Allgemein:

In diesem Prädikat wird das Element mit dem Wert: “element” gelöscht. Das passiert an der Stelle, die Der wert: “position” vorschreibt. 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.

* + 1. Strategie:

1. Wenn die position = “l” ist
   1. drehe die Liste um
   2. gehe 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”
   1. Füge der “Ergebnissliste” den 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:
   1. Gehe zu schritt 3. mit dem nächsten Element der Liste
4. SONST: füge der Liste: “ergebnissliste” das erste Element der Liste “liste” hinzu und gehe zu schritt 1. mit dem Rest der Liste “liste”.
   1. substitute (position, element, newelement, liste, returnList)
      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.

* + 1. Strategie:

1. Wenn die position = “l” ist
   1. drehe die Liste um
   2. gehe 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”
   1. Füge der “Ergebnissliste” erst das element “newelement” und dan 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:
   1. Füge der Liste “Ergebnissliste” das element “newelement” hinzu.
   2. Gehe zu schritt 3. mit dem nächsten Element der Liste
4. SONST: füge der Liste: “ergebnissliste” das erste Element der Liste “liste” hinzu und gehe zu schritt 1. mit dem Rest der Liste “liste”.
5. Strategie: Arithmetik

Die S-Zahlen sind so aufgebaut: eine „0“ wird als 0 dargestellt. Die Zahl „1“ hingegen ist ein wenig anders aus als die Normale Schreibweise: S(0) das ist die Schreibweise von der „1“. Und für jede höhere Zahl kommt ein „S“ dazu. Das bedeutet eine 2 ist in den S-Zahlen: S(S(0)). Das lässt sich natürlich beliebig erweitern.

* 1. nat2s(nat, s)
     1. Allgemein:

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

* + 1. Strategie:

1. Ist “nat” = 0?

Ja – brich ab und gib “s” zurück.

Nein – gehe zu schritt 1 und dekrementiere nat um eins und erweitere die S- Zahl um ein “S()”

* 1. s2nat(s, nat)
     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.

* + 1. Strategie:

1. Ist “s” = 0?

Ja – brich ab und gib “nat” zurück.

Nein – gehe zu schritt 1 und inkrementiere nat um eins und verringere die S- Zahl um ein “S()”

* 1. add(s1, s2, ergebniss)
     1. Allgemein:

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

* + 1. Strategie:

1. Ist „s1“ UNGLEICH „0“

Ja – gehe zu schritt 1 mit „ergebniss“ um ein „S()“ inkrementiert und „s1“ um ein „S()“ dekrementiert

Nein – gehe zu schritt 2.

1. Ist „s2“ UNGLEICH „0“

Ja – gehe zu schritt 2 mit „ergebniss“ um ein „S()“ inkrementiert und „s2“ um ein „S()“ dekrementiert

Nein – gehe zu schritt 3.

1. Sonst: (sind „s1 und “ „s2“ GLEICH „0“)

Gebe „ergebniss“ zurück.

* 1. sub(s1, s2, ergebniss)
     1. Allgemein:

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

* + 1. Strategie:

1. Ist „s1“ UNGLEICH „0“

Ja – gehe zu schritt 1 mit „ergebniss“ um ein „S()“ inkrementiert und „s1“ um ein „S()“ dekrementiert

Nein – gehe zu schritt 2.

1. Ist „s2“ UNGLEICH „0“

Ja – Ist „s1“ GLEICH „0“

Ja – abbrechen und ergebniss zurückgeben.(ergebniss kann nicht kleiner werden als „0“)

Nein - gehe zu schritt 2 mit „ergebniss“ um ein „S()“ dekrementiert um ein „S()“ und „s2“ um ein „S()“ inkrementiert

Nein – gehe zu schritt 3.

1. Sonst: (sind „s1 und “ „s2“ GLEICH „0“)

Gebe „ergebniss“ zurück.

* 1. mul(s1, s2, ergebniss)
     1. Allgemein:

Dieses Prädikat multipliziert die S-Zahl “s1”, mit der S-Zahl “s2”. Das wird wieder über Rekursion realisiert unter gebrauch von add(s1, s2, ergebniss).

* + 1. Strategie:

1. Ist “s1” oder “s2” GLEICH 0 dann ist das ergebniss = 0
2. Ist „s2“ UNGLEICH „0“

Ja – gehe zu Schritt 2 mit dem ergebniss = add(ergebniss, s1, ergebniss) und s2 um ein „S()“ dekrementiert

Nein – brich ab und gebe „ergebniss“ zurück.

* 1. power(s1, s2, ergebniss)
     1. Allgemein:

Dieses Prädikat rechnet die S-Zahl “s1”, hoch die S-Zahl “s2”. Das wird wieder über Rekursion realisiert unter gebrauch von mul(s1, s2, ergebniss).

* + 1. Strategie:

1. Ist “s1” oder “s2” GLEICH 0 dann ist das ergebniss = 0
2. Ist „s2“ UNGLEICH „0“

Ja – gehe zu Schritt 2 mit dem ergebniss = mul(ergebniss, s1, ergebniss) und s2 um ein „S()“ dekrementiert

Nein – brich ab und gebe „ergebniss“ zurück.

* 1. fac(s, ergebniss)
     1. Allgemein:

Dieses Prädikat rechnet das der S-Zahl “s” aus. Das wird wieder über Rekursion realisiert. In jedem rekursionsschritt wird der aktelle Wert von “s” mit “ergebniss” multibpliziert

* + 1. Strategie:

1. Ist “s” GLEICH 0 dann ist das ergebniss = 0
2. Ist „s“ UNGLEICH „0“
   1. Ja – gehe zu Schritt 2 mit dem ergebniss = mul(ergebniss, s, ergebniss) und dekrementiere s um ein „S()“.
   2. Nein – brich ab und gebe „ergebniss“ zurück.
   3. lt(s1, s2)
      1. Allgemein:

Dieses Prädikat ermittelt mittles Rekursion, ob die S-Zahl “s1” kleiner als “s2” ist.

* + 1. Strategie:

1. Ist „s1“ UNGLEICH „0“
   1. Ja – gehe zu schritt 1. Mit „s1“ und „s2“ um ein „S()“ dekrementiert
   2. Nein – Ist “s2” UNGLEICH “0”
      1. Ja - dann gib wahr zurück
      2. Nein - dann gib falsch zurück
   3. Mods(s1, s2, ergebniss)
      1. Allgemein:

Diese Prädikat ermittelt mittels Rekursion, den Modulo wert von “s1” MODULO “s2”

* + 1. Strategie:

1. Ist “s2” kleiner als “s1”?
   1. Ja – dann gib das “s2” zurück
   2. Nein – dann gehe zu schritt 2. Mit “s1” = “ergebniss” von = sub(s1, s2, ergebniss)

**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 selbständig 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*