# TU Wien

**Business Informatics Group**

Institut für Softwaretechnik und Interaktive Systeme


---

## Proseminar Grundlagen wissenschaftlichen Arbeitens

## Theoretische Informatik: Aussagenlogik, Berechnungsmodelle, Grenzen der Berechenbarkeit, Formale Sprachen und Automaten

Leitung: Wolfgang Hoschek


Elisabeth Grill, 0208189

Iris Meyer, 0060519

Katrin Zöchmeister, 0206332


Wien, 14.April 2003

# Inhaltsverzeichnis

**1 Aussagenlogik** ..... **3**
**1.1 Einleitung** ..... **3**
**1.2 Grundbegriffe der Aussagenlogik** ..... **4**
1.2.1 Aussage ..... 4
1.2.2 Aussagenlogische Formel ..... 4
1.2.3 Logische Folgerungsbeziehung ..... 5
**1.3 Syntax der Aussagenlogik** ..... **6**
**1.4 Semantik der Aussagenlogik** ..... **6**
1.4.1 Das Wahrheitstafelverfahren ..... 7
1.4.2 Normalformen ..... 9
**1.5 Resolution** ..... **11**
**1.6 Zusammenfassung** ..... **13**

**2 Berechnungsmodelle** ..... **15**
**2.1 Einleitung** ..... **15**
**2.2 Speicherorientierte Modelle** ..... **15**
2.2.1 Beispiel Turingmaschine ..... 15
2.2.2 Beispiel Registermaschine (Random access machine = RAM) ..... 18
**2.3 Funktionale Modelle** ..... **19**
**2.4 Kommunikation und verteilte Systeme** ..... **19**
2.4.1 EA- Turingmaschinen ..... 20
2.4.2 Nicht abbrechende Berechnungen ..... 20
2.4.3 Verteilte Systeme ..... 21

**3 Grenzen der Berechenbarkeit** ..... **21**
**3.1 Einleitung** ..... **21**
3.1.1 Universelle Rechner ..... 22
3.1.2 Simulationen ..... 22
3.1.3 Churchsche These ..... 22
**3.2 Existenz nichtberechenbarer Funktionen** ..... **23**

**3.3 Das Halteproblem** ..... **23**
**3.4 Softwareverifikation** ..... **23**

**4 Formale Sprachen und Automaten** ..... **24**
**4.1 Einleitung** ..... **24**
**4.2 Grundbegriffe** ..... **24**
4.2.1. Alphabete und Sprachen ..... 25
4.2.2 Grammatiken, rekursive Grammatiken ..... 26
4.2.3 Kanonische Ableitungen und Mehrdeutigkeit ..... 27
**4.3 Die Chomsky-Hierarchie** ..... **27**
**4.4 Reguläre Ausdrücke und endliche Automaten** ..... **28**
4.4.1 Reguläre Mengen und reguläre Ausdrücke ..... 28
4.4.2 Endliche Automaten ..... 29
**4.5 Kontextfreie Sprachen und Kellerautomaten** ..... **33**
4.5.1 Charakterisierung ..... 33
4.5.2 Grammatiktransformationen und Normalformen ..... 34
4.5.3 Kellerautomaten ..... 35
4.5.4. Kontextfreie Grammatiken und Kellerautomaten ..... 37
**4.6 Weitere Sprachklassen** ..... **37**
**4.7 Eine Hierarchie von Sprachen** ..... **37**
**Symbolverzeichnis** ..... **38**

**Zusammenfassung** ..... **39**

**Literaturverzeichnis** ..... **40**

# 1 Aussagenlogik

Iris Meyer

## 1.1 Einleitung

Seit rund 150 Jahren hat die Logik, die als philosophische Disziplin eine rund 2500jährige Tradition besitzt, einen immensen Aufschwung in der Mathematik verzeichnet und seit wenigen Jahrzehnten erfährt sie auch im Rahmen der Informatik eine ungeheure Aufmerksamkeit. Logische Schaltungen, Boolesche Ausdrücke, automatisches Beweisen, Syntax und Semantik von Programmiersprachen, logische Anfragesprachen bei Datenbank- und Informationssystemen, ... - in fast allen Bereichen der Informatik spielt Logik eine beispielslose Rolle.

Da die Informatik ohne Logik kaum vorstellbar ist, soll es Gegenstand und Aufgabe des vorliegenden Kapitels sein, dem Leser / der Leserin einen Überblick über die logischen Grundlagen der Informatik zu vermitteln, wobei ich mich bei meinen Ausführungen nur auf ein Teilgebiet der Logik, nämlich auf das der Aussagenlogik beschränken werde.

Der Kalkül des natürlichen Schließens, der für den Beweis einer syntaktischen Folgerungsbeziehung, wie es in Kapitel 1.2.3 beschrieben wird, fundamental wichtig ist und in der ersten Fassung dieser Arbeit auch integriert war, kann aufgrund der geforderten Kürze der Arbeit leider nicht näher angeführt werden. Ich bedauere diesen Umstand sehr, beuge mich allerdings den vorgegebenen Richtlinien und kann dem interessierten Leser / der interessierten Leserin nur das Angebot machen, ihm / ihr das Kapitel über den Kalkül des natürlichen Schließens sowie dessen Anwendung zum Beweis der syntaktischen Gültigkeit eines Arguments, was meines Erachtens eines der spannendsten Gebiete der Aussagenlogik darstellt, auf Anfrage unter der E-Mail Adresse ye2028@happynet.at zuzuschicken.

## 1.2 Grundbegriffe der Aussagenlogik

### 1.2.1 Aussage

Zur Festlegung, was eine Aussage ist, bieten sich zwei Kriterien an [Ober92, ZeMa99]:

**1. Grammatikkriterium:**
*Eine Aussage ist ein sprachliches Gebilde, das aufgrund der Regeln einer Grammatik als Aussagesatz klassifiziert ist.*

**2. Wahrheitskriterium:**
*Eine Aussage ist ein sprachliches Gebilde, das nach Fixierung einer Interpretation entweder wahr oder falsch ist. Dies bedeutet nun allerdings, dass es keinen dritten Wahrheitswert gibt (Satz vom ausgeschlossenen Dritten - tertium non datur) und eine Aussage nicht gleichzeitig wahr und falsch sein kann (Satz vom ausgeschlossenen Widerspruch) [Zogl97].*

### 1.2.2 Aussagenlogische Formel

Nach [Zogl97] ist eine aussagenlogische Formel "... ein Ausdruck, der eine (oder mehrere) freie Variablen (Leerstellen) enthält und durch die Belegung aller freien Variablen in eine (wahre oder falsche) Aussage übergeht."

Eine aussagenlogische Formel, die

- ?? unter allen Interpretationen der aussagenlogischen Variablen wahr ist, wird als **Tautologie** oder als **allgemeingültig** bezeichnet.
- ?? unter keiner Interpretation der aussagenlogischen Variablen wahr ist, heißt **unerfüllbar** oder **kontradiktorisch**.

?? wenigstens unter einer Interpretation der aussagenlogischen Variablen wahr ist, heißt **erfüllbar**.

?? in wenigstens einer Interpretation der aussagenlogischen Variablen falsch ist, heißt **widerlegbar**.

?? sowohl erfüllbar als auch widerlegbar ist, heißt **kontingent**.

### 1.2.3 Logische Folgerungsbeziehung

**Argument, Schluss**

Ein Argument oder ein Schluss ist eine Menge von Aussagen, wobei für mindestens eine der Aussagen beansprucht wird, dass sie aus den anderen logisch folgt bzw. sich aus den anderen mit Notwendigkeit ergibt [Bühl92]. Die Aussage, für die gefordert wird, dass sie aus den anderen folgt, nennt man *Konklusion* (Behauptung) des Arguments bzw. des Schlusses. Die übrigen Aussagen werden als *Prämissen* (Annahmen) bezeichnet.

**Semantische Folgerungsbeziehung**

Wenn unter der Vorraussetzung, dass alle Prämissen $f_1,...,f_n$ wahr sind, auch die Konklusion f wahr ist, so schreibt man $f_1,...,f_n \models f$ und nennt die Beziehung $\models$ zwischen den Prämissen auf der linken und der Konklusion auf der rechten Seite eine *semantische Folgerungsbeziehung* [Jung99].

**Syntaktische Folgerungsbeziehung**

In der Logik versucht man die semantische Folgerungsbeziehung durch einen formalen Kalkül zu erfassen. Ein solcher Kalkül besteht aus sog. *Ableitungs*- oder *Transformationsregeln*, die es erlauben Aussagen so umzuformen, dass aus ihnen neue Aussagen entstehen, die aus den ersten Aussagen folgen. Dies führt zur *syntaktischen Folgerungsbeziehung* $\vdash$ [Kuts67, Schö95].

## 1.3 Syntax der Aussagenlogik

Eine aussagenlogische Sprache ist durch ein Alphabet und die Menge der aussagenlogischen Formeln festgelegt [Jung99, Kuts67, Mate69, Ober92, Schö95, ZeMa99]:

(a) *Das* Alphabet A(P) *einer aussagenlogischen Sprache besteht aus*

?? den aussagenlogischen Variablen $p_i$, $i$ ? N,

?? den aussagenlogischen Junktoren T, ?, ? , ? , ? , ? und ? ,

?? den Klammern ( und ).

(b) *Die Menge* Form (P) ? A (P)* *der aussagenlogischen Formeln ist wie folgt rekursiv definiert:*

?? Jede aussagenlogische Variable ist eine aussagenlogische Formel.

?? Die aussagenlogischen Junktoren T und ? sind aussagenlogische Formeln.

?? Ist f eine aussagenlogische Formel, dann auch ? f .

?? Sind f und ? aussagenlogische Formeln, dann auch (f ? ?), (f ? ?), (f ? ?) und (f ? ?) .

## 1.4 Semantik der Aussagenlogik

In der Aussagenlogik wird jeder aussagenlogischen Variablen ein Wahrheitswert (wahr oder falsch) zugeordnet. Formal entspricht diese Wahrheitswertzuordnung einer Abbildung ? von der Menge der aussagenlogischen Variablen $\{p_1,...,p_n\}$ in die zweielementige Menge {W,F}. Der Wahrheitswert ⟦ f ⟧ ? einer Formel f wird, auf ? aufbauend, induktiv bestimmt [Jung99, Ober92]:

**(a) Wahrheitswert von aussagenlogischen Variablen**

Ist f eine aussagenlogische Variable $p_i$, so ist ⟦f ⟧ ? = $_{def}$ ? ($p_i$).

**(b) Wahrheitswert von Verum und Falsum**

⟦T ⟧ ? = $_{def}$ W und ⟦? ⟧ ? = $_{def}$ F

**(c) Wahrheitswerte von Negationen**

Ist f von der Form ? ?, so ist

$$[\![ f ]\!] ? =_{def} \begin{cases} W, \text{ wenn } [\![ ? ]\!] ? = F \\ F, \text{ wenn } [\![ ? ]\!] ? = W \end{cases}$$

**(d) Wahrheitswerte von Konjunktionen**

Ist f von der Form $? _1 ? ?_2$, so ist

$$[\![ f ]\!] ? =_{def} \begin{cases} W, \text{ wenn } [\![ ?_1 ]\!] ? = [\![ ?_2 ]\!] ? = F \\ F, \text{ sonst} \end{cases}$$

**(e) Wahrheitswerte von Disjunktionen**

Ist f von der Form $? _1 ? ?_2$, so ist

$$[\![ f ]\!] ? =_{def} \begin{cases} W, \text{ sonst} \\ F, \text{ wenn } [\![ ?_1 ]\!] ? = [\![ ?_2 ]\!] ? = F \end{cases}$$

**(f) Wahrheitswerte von Implikationen**

Ist f von der Form $? _1 ? ?_2$, so ist

$$[\![ f ]\!] ? =_{def} \begin{cases} W, \text{ sonst} \\ F, \text{ wenn } [\![ ?_1 ]\!] ? = W \text{ und } [\![ ?_2 ]\!] ? = F \end{cases}$$

**(g) Wahrheitswerte von Äquivalenzen**

Ist f von der Form $? _1 ? ?_2$, so ist

$$[\![ f ]\!] ? =_{def} \begin{cases} W, \text{ wenn } [\![ ?_1 ]\!] ? = [\![ ?_2 ]\!] ? \\ F, \text{ sonst} \end{cases}$$

## 1.4.1 Das Wahrheitstafelverfahren

Wahrheitstafeln oder Wahrheitstabellen sind eine tabellarische Darstellung aller möglichen Wahrheitswertzuordnungen für die Aussagenvariablen einer Formel. Allgemein gilt: wenn eine Formel aus n Aussagenvariablen zusammengesetzt ist, so gibt es $2^n$ verschiedene Wahrheitswertzuordnungen. Eine Wahrheitstafel erhält man, indem man eine Tabelle mit n + 1 Spalten und $2^n$ Zeilen anlegt. In die Spaltenköpfe trägt man $p_1,...,p_n$ und die Formel f ein. In den $2^n$ Zeilen der ersten n Spalten trägt man alle Möglichkeiten ein, die aussagenlogischen Variablen $p_1,...,p_n$ mit W oder F zu bewerten. Die Wahrheitstafel wird dann ausgerechnet, indem man in jeder Zeile die für $p_1,...,p_n$ gegebenen Werte in die f-Spalte überträgt und dort unter die Konstanten $p_1,...,p_n$ von f schreibt, aus denen f aussagenlogisch aufgebaut ist. Im Folgenden rechnet man sukzessive die Werte weiterer Teilformen, wobei man den Wert einer Teilformel jeweils unter den Hauptjunktor der Teilformel schreibt.

Als Beispiel betrachten wir nun die aussagenlogische Formel $(p_1 \ ? \ p_2 \ ? \ p_3) \ ? \ (p_1 \ ? \ p_3) \ ? \ (p_2 \ ? \ p_3)$

Die Wahrheitstafel schaut wie folgt aus (die Zahlen in der ersten Zeile geben dabei die Reihenfolge der Berechnungsschritte an):

| 1 | 1 | 1 | 1 | 2 | 1 | 3 | 1 | **4** | 1 | 2 | 1 | 3 | 1 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_1$ | $p_2$ | $p_3$ | $(p_1$ | ? | $p_2$ | ? | $p_3)$ | **?** | $(p_1$ | ? | $p_3)$ | ? | $(p_2$ | ? | $p_3)$ |
| F | F | F | F | F | F | W | F | **W** | F | W | F | W | F | W | F |
| F | F | W | F | F | F | W | W | **W** | F | W | W | W | F | W | W |
| F | W | F | F | F | W | W | F | **W** | F | W | F | W | W | F | F |
| F | W | W | F | F | W | W | W | **W** | F | W | W | W | W | W | W |
| W | F | F | W | F | F | W | F | **W** | W | F | F | W | F | W | F |
| W | F | W | W | F | F | W | W | **W** | W | W | W | W | F | W | W |
| W | W | F | W | W | W | F | F | **W** | W | F | F | F | W | F | F |
| W | W | W | W | W | W | W | W | **W** | W | W | W | W | W | W | W |

Tabelle 1: Wahrheitstafel für die aussagenlogisch Formel $(p_1 \ ? \ p_2 \ ? \ p_3) \ ? \ (p_1 \ ? \ p_3) \ ? \ (p_2 \ ? \ p_3)$

Mit Hilfe einer Wahrheitstabelle kann man nun auch überprüfen, ob ein Argument semantisch gültig ist oder nicht. Wie bereits erläutert wurde, ist ein Argument genau

dann semantisch gültig, wenn unter der Vorraussetzung, dass alle Prämissen $f_1,...,f_n$ wahr sind, auch die Konklusion f wahr ist. Gibt es aber nur eine einzige Zuordnung, die die Prämissen wahr, aber das Argument falsch macht, so handelt es sich um ein ungültiges Argument.

Als Beispiel wollen wir das Argument $p_1$ ? $p_2$, $p_2$ ? $p_3$, $p_1$ |- $p_3$ auf seine semantische Gültigkeit untersuchen. Wir betrachtet dazu die Wahrheitstafel:

| | | | **1. Prämisse** | | | | **2. Prämisse** | | | | **3. Prämisse** | | **Konklusion** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_1$ | $p_2$ | $p_3$ | $p_1$ | **?** | $p_2$ | , | $p_2$ | **?** | $p_3$ | , | $\mathbf{p_1}$ | \|- | $\mathbf{p_3}$ |
| F | F | F | F | **W** | F | | F | **W** | F | | **F** | | **F** |
| F | F | W | F | **W** | F | | F | **W** | W | | **F** | | **W** |
| F | W | F | F | **W** | W | | W | **F** | F | | **F** | | **F** |
| F | W | W | F | **W** | W | | W | **W** | W | | **F** | | **W** |
| W | F | F | W | **F** | F | | F | **W** | F | | **W** | | **F** |
| W | F | W | W | **F** | F | | F | **W** | W | | **W** | | **W** |
| W | W | F | W | **W** | W | | W | **F** | F | | **W** | | **F** |
| W | W | W | W | **W** | W | | W | **W** | W | | **W** | | **W** |

Tabelle 2: Wahrheitstafel für das Argument $p_1$ ? $p_2$, $p_2$ ? $p_3$, $p_1$ |- $p_3$

In der letzten Zeile der Tabelle sehen wir, dass sowohl allen Prämissen als auch der Konklusion der Wahrheitswert W zugeordnet wurde. Da es keine weitere Zuordnung gibt, die alle Prämissen wahr werden lässt, wissen wir, dass das Argument semantisch gültig ist und können daher auch schreiben: $p_1$ ? $p_2$, $p_2$ ? $p_3$, $p_1$ |= $p_3$.

## 1.4.2 Normalformen

Zwei Arten von Normalformen werden in diesem Abschnitt unterschieden:

(a) disjunktive Normalform

(b) konjunktive Normalform

Eine disjunktive Normalform ist eine Disjunktion von Konjunktionen, wobei in jedem Disjunktionsglied jede aussagenlogische Variable oder deren Negation genau einmal vorkommt. Eine aussagenlogische Formel, die nur aus einer Variable oder deren Negation besteht, nennt man auch Literal. Dementsprechend kann man nun die disjunktive Normalform auch als Disjunktion von Konjunktionen von Literalen definieren. Analog lässt sich die konjunktive Normalform als Konjunktion von Disjunktionen von Literalen erklären [Jung99, Zogl97].

Man kann nun jeden aussagenlogischen Ausdruck in eine konjunktive oder disjunktive Normalform transformieren. Um eine aussagenlogische Formel in eine disjunktive Normalform überzuführen, betrachtet man am besten die Wahrheitstafel dieser Formel: "Zu jeder Zeile der Tafel, für die f wahr wird, wählt man eine Konjunktion von Literalen, die für genau diese Zeile wahr wird. Die disjunktive Normalform ist dann die Disjunktion dieser Konjunktionen. Die konjunktive Normalform erhält man entsprechend aus denjenigen Zeilen, für die die Formel falsch ist" [Jung99].

**Beispiel für eine konjunktive Normalform:**

Die aussagenlogische Formel $((p_1 \ ? \ p_2) \ ? \ p_3)$ ist in eine konjunktive Normalform zu überführen.

| $p_1$ | $p_2$ | $p_3$ | $((p_1$ | ? | $p_2)$ | ? | $p_3)$ |
|---|---|---|---|---|---|---|---|
| **F** | **F** | **F** | F | F | F | **F** | F |
| F | F | W | F | F | F | W | W |
| **F** | **W** | **F** | F | F | W | **F** | F |
| F | W | W | F | F | W | W | W |
| **W** | **F** | **F** | W | F | F | **F** | F |
| W | F | W | W | F | F | W | W |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| W | W | F | W | W | W | W | F |
| W | W | W | W | W | W | W | W |

Tabelle 3: Wahrheitstafel für die aussagenlogische Formel ((p1 ? p2) ? p3)

Die Lösung lautet somit ($p_1$ ? $p_2$ ? $p_3$) ? ($p_1$ ? ? $p_2$ ? $p_3$) ? (? $p_1$ ? $p_2$ ? $p_3$).

**Beispiel für eine disjunktive Normalform:**

Die aussagenlogische Formel ($p_1$ ? ? $p_3$) ? ($p_1$ ? ($p_2$ ? $p_3$)) ist in eine disjunktive Normalform zu überführen.

| $p_1$ | $p_2$ | $p_3$ | ($p_1$ | ? | ? $p_3$) | ? | ($p_1$ | ? | ($p_2$ | ? | $p_3$)) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | F | F | F | F | W | F | F | F | F | F | F |
| F | F | W | F | F | F | F | F | F | F | W | W |
| F | W | F | F | F | W | F | F | F | W | W | F |
| F | W | W | F | F | F | F | F | F | W | W | W |
| **W** | **F** | **F** | W | W | W | **W** | W | F | F | F | F |
| **W** | **F** | **W** | W | F | F | **W** | W | W | F | W | W |
| **W** | **W** | **F** | W | W | W | **W** | W | W | W | W | F |
| **W** | **W** | **W** | W | F | F | **W** | W | W | W | W | W |

Tabelle 4: Wahrheitstafel für die aussagenlogische Formel ($p_1$ ? ? $p_3$) ? ($p_1$ ? ($p_2$ ? $p_3$))

Die Lösung lautet somit ($p_1$ ? ? $p_2$ ? ? $p_3$) ? ($p_1$ ? ? $p_2$ ? $p_3$) ? ($p_1$ ? $p_2$ ? ? $p_3$) ? ($p_1$ ? $p_2$ ? $p_3$).

## 1.5 Resolution

Die Resolution ist eine einfache syntaktische Umformungsregel und dient als Methode zum automatischen Beweisen, d.h. zum Beweisen durch ein Computerprogramm. Mit dem Resolutionsverfahren versucht man die Unerfüllbarkeit einer gegebenen Formelmenge nachgewiesen. D.h. will man überprüfen, ob f aus einer gegebenen Formelmenge $\{f_1, \ldots, f_n\}$ folgt, dann reicht es

die Unerfüllbarkeit von $f_1 ? f_2 ? \ldots ? f_n ? ? f$ nachzuweisen. Das Beweisverfahren, das zu diesem Zweck angewendet wird, ist dabei auf einer einzigen Regel, der sog. *Resolventenregel* aufgebaut:

$$\frac{p ? ? r, q ? r}{p ? q}$$

Vorraussetzung für die Anwendung der Resolution ist allerdings, dass die Formeln in *Klausel-Repräsentationen* gegeben sind. Klausel-Repräsentationen sind konjunktive Normalformen in Mengenschreibweise [Jung99, Schö95, ZeMa99]. Auf die Klauselmenge wird so dann die bereits angeführte Resolventenregel angewendet, indem zwei Klauseln zu einer neuen Klausel zusammengefasst werden, wobei inverse Elemente eliminiert werden. Nach endlich vielen Schritten lassen sich dann entweder keine neuen Klauseln mehr erzeugen oder die leere Klausel wurde gefunden. Der erste Fall bedeutet, dass die Formelmenge $\{f_1, \ldots, f_n, ? f\}$ erfüllbar und damit die Folgerungsbeziehung $f_1, \ldots, f_n \models f$ falsch ist. Der zweite Fall bedeutet Unerfüllbarkeit der Formelmenge $\{f_1, \ldots, f_n, ? f\}$, womit die Korrektheit der Folgerungsbeziehung $f_1, \ldots, f_n \models f$ bewiesen ist [Jung99, Schö95, ZeMa99].

Als Beispiel wollen wir die Gültigkeit des Arguments $p ? ? q ? r, (p ? q ? r) ? (? p ? q ? r), (? p ? q ? r) ? (? p ? ? q ? r) \models r$ nachweisen. Um die Richtigkeit des Arguments nachzuweisen, gilt die Unerfüllbarkeit von $(p ? ? q ? r) ? (p ? q ? r) ? (? p ? q ? r) ? (? p ? q ? r) ? (? p ? ? q ? r) ? ? r$ zu beweisen. Wir schreiben in Klauselrepräsentation $\{\{p,? q, r\}, \{p, q, r\}, \{? p, q, r\}, \{? p, q, r\}, \{? p, ? q, r\}, \{? r\}\}$ und wollen nun die einzelnen Resolutionsschritte in Tabellenform notieren:

| | | |
|---|---|---|
| 1 | {p,? **q**, r} | Ausgangsklausel |
| 2 | {**p**, q, r} | Ausgangsklausel |
| 3 | {? **p**, q, r} | Ausgangsklausel |
| 4 | {? p, **q**, r} | Ausgangsklausel |
| 5 | {? p, ? **q**, r} | Ausgangsklausel |
| 6 | {? **r**} | Ausgangsklausel |
| 7 | {**q**, r} | Resolvente von 2 und 3 |

| | | |
|---|---|---|
| 8 | {? **p**, r} | Resolvente von 4 und 5 |
| 9 | {**p**, r} | Resolvente von 1 und 7 |
| 10 | {**r**} | Resolvente von 8 und 9 |
| 11 | {} | Resolvente von 6 und 10 |

Tabelle 5: Darstellung der einzelnen Resolutionsschritte für die Klauselmenge {{p,? q, r}, {p, q, r}, {? p, q, r}, {? p, q, r}, {? p, ? q, r}, {? r}}

Die leere Klausel ist gefunden worden, womit die Gültigkeit des Arguments p ? ? q ? r, (p ? q ? r) ? (? p ? q ? r), (? p ? q ? r) ? (? p ? ? q ? r) |= r bewiesen wäre.

## 1.6 Zusammenfassung

Am Schluss dieses Kapitels hoffe, dass das Ziel dieser Arbeit, nämlich dem Leser / der Leserin einen kurzen Überblick über die Aussagenlogik zu verschaffen, erreicht worden ist. Mir war es in dieser Arbeit ein wichtiges Anliegen, zunächst einige Grundbegriffe der Aussagenlogik zu erklären um mich sodann mit der Syntax und Semantik der Aussagenlogik beschäftigen zu können und schließlich den Bogen zu Kapitel 1.5 zu spannen, das sich mit dem Resolutionsverfahren auseinandersetzt und meines Erachtens gerade für Studierende der Informatik von besonderem Interesse sein sollte, da der Resolutionskalkül ja besonders in der Informatik als automatisches Beweisverfahren eine bedeutende Rolle spielt.

## Literaturverzeichnis

[Bühl92] A. Bühler, Einführung in die Logik, Argumentation und Folgerung, Karl Alber Verlag, München, 1992

[Copi98] I. Copi, Einführung in die Logik, Wilhelm Fink Verlag, München,

1998

[Erns00] H. Ernst, Grundlagen und Konzepte der Informatik, Eine Einführung in die Informatik ausgehend von den fundamentalen Grundlagen, 2. Auflage, Vieweg & Sohn Verlagsgesellschaft, Braunschweig, Wiesbaden, 2000

[Jung99] A. Jung, Logik, in: P. Rechenberg, G. Pomberger (Hrsg.), Informatik-Handbuch, 2.Auflage, Carl Hanser Verlag, München, Wien, 1999, S. 33 - 72

[KlWü76] R. Kleinknecht, E. Wüst, Lehrbuch der elementaren Logik, Band 1: Aussagenlogik, Deutscher Taschenbuch Verlag, 1976

[Kreo91] H.-J. Kreowski, Logische Grundlagen der Informatik, (Handbuch der Informatik, Bd. 1.1), Oldenbourg Verlag, München, Wien, 1991

[Kuts67] F. v. Kutschera, Elementare Logik, Springer-Verlag, Wien, New York, 1967

[Mate69] B. Mates, Elementare Logik, Prädikatenlogik der ersten Stufe, Vandenhoeck und Ruprecht, Göttingen, 1969

[Ober92] A. Oberschelp, Logik für Philosophen, BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1992

[Schö95] U. Schöning, Logik für Informatiker, 4. Auflage, Spektrum Akademischer Verlag, Berlin, Heidelberg, Oxford, 1995

[ZeMa99] P. Zeitz, B. Mahr, Aussagenlogik, in: F. Cornelius, H. Ehring, M. Große-Rhode, B. Mahr, P. Zeitz, Mathematisch-strukturelle Grundlagen der Informatik, Springer-Verlag, Barcelona, Berlin,

Heidelberg, Hongkong, London, Mailand, New York, Paris, Singapur, Tokio, 1999, S.199 - 304

[Zogl97] T. Zoglauer, Einführung in die formale Logik für Philosophen, Vandenhoeck und Ruprecht, Göttingen, 1997

# 2 Berechnungsmodelle

Katrin Zöchmeister

## 2.1 Einleitung

Im Sinne der Informatik ist eine Berechnung die Ausführung eines Programms durch eine Maschine. Um diese Berechnungen so abstrakt und allgemein wie möglich zu halten, spricht man von Berechnungsmodellen. In diesem Kapitel werden 3 verschiedene Modelle dargestellt.

## 2.2 Speicherorientierte Modelle

Bei den Speicherorientierten Modellen bedeutet eine Berechnung eine schrittweise Veränderung des Speichers.

### 2.2.1 Beispiel Turingmaschine

Turingmaschinen haben einen Speicher, der ähnlich einer Videokassette aufgebaut ist. Es gibt kleine Speichereinheiten, die man von einer Zelle zur nächsten

durchlaufen muss. Direkte Adressierung gibt es nicht. In jeder dieser kleinen Speichereinheiten steht jeweils ein Zeichen eines (endlichen) Alphabets.

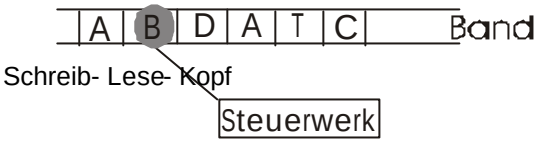


Abbildung 1: Schema einer Turingmaschine

Die Turingmaschine besteht aus:

- Einem endlichen Alphabet, das auf dem Band aufgezeichnet ist und aus mindestens einem Zeichen besteht. Zusätzlich gibt es, welches im Alphabet nicht enthalten ist.

- Einem unendlichen Band (nach beiden Seiten), welches in Zellen unterteilt ist, in denen wiederum je ein Zeichen des Alphabets oder ein Leerzeichen steht.

- Einem Schreib- und Lesekopf, der sich ständig über einer Zelle des Bandes befindet. Er kann Zeichen lesen, löschen oder überschreiben. In jedem Takt wird er um eine Stelle nach rechts oder links verschoben.

- Einem Steuerwerk, dass sich zu jedem Zeitpunkt in einem bestimmten Zustand befindet.

Die Turingmaschine erhält Anweisungen, die aus einer Vorbedingung und einer Aktion bestehen. Die Vorbedingung gibt an, welches Zeichen in der aktuellen Zelle steht, und in welchem Zustand sich das Steuerwerk befinden muss, damit die Anweisung gültig ist. Die Aktion schreibt vor, welches Zeichen der Schreib- und Lesekopf in die Zelle schreiben soll. Außerdem legt die Aktion fest, ob nach links oder rechts weitergegangen wird und in welchen Zustand das Steuerwerk wechseln soll.

Nach diesen einfachen Regeln kann man auch ein Programm definieren: Es ist eine endliche Menge von Anweisungen für die Turingmaschine. Dabei darf jede Anweisung nur einmal auftreten. Die Reihenfolge dieser Anweisungen ist unwichtig. Die Ausführung eines Programms ist die Suche nach einer zum aktuellen Zustand passenden Anweisung und deren Aktion durchzuführen.

Eine Berechnung wird wie folgt durchgeführt:

- Der Schreib- Lesekopf befindet sich über dem ersten Zeichen.
- Das Steuerwerk ist im Anfangszustand.
- Es wird im Programm sequentiell nach aufeinander passende Anweisungen gesucht.
- Wenn keine Anweisungen mehr aufeinander passen, ist das Programm zu Ende.

So eine Maschine nennt man auch terministisch, man sagt aber auch, dass die Turingmaschine hält. Wenn es in jeder Situation höchstens eine Anweisung gibt, die auf die Vorhergehende folgt, so heißt die Maschine auch deterministisch.

#### 2.2.1.1 Variationen der Turingmaschinen

Das hier beschriebene Modell kann sehr stark variiert werden. So ist es zum Beispiel möglich, auch mehrere Bänder zu verknüpfen oder ein Band in eine Richtung unendlich beschränken. Auch kann es auf einem Band mehrere Schreib-Leseköpfe geben. Statt einer eindimensionalen Anordnung der Zeichen in den Zellen ist es auch möglich, 2 oder mehr Zeichen hineinzuschreiben.
Jedoch kann man alle Variationen dieses Modells durch eine Einbandmaschine simulieren.

#### 2.2.1.2 Vorteile der Turingmaschine

Jeden bis heute untersuchten Formalismus kann man auf einer Turingmaschine simulieren. Die Berechnungsschritte der Turingmaschine sind sehr einfach und man

kann sie daher ohne Zweifel auch direkt in die Praxis übersetzen. Weiters ist die Beschreibung einer solchen Maschine sehr einfach, anschaulich und daher leicht verständlich.

#### 2.2.1.3 Nachteile der Turingmaschine

Teile der Variationen sind nicht realisierbar, wie zum Beispiel das unendlich lange Band, denn ein solches würde einen unendlichen Bedarf an Speicher verlangen. Auch die einfache „Programmiersprache“ der Turingmaschine macht das Programmieren selbst sehr kompliziert.
Die heutigen Rechner nach dem von Neumann- Prinzip haben fast keine Ähnlichkeit mit den Programmen der Turingmaschine. In der v. Neumannarchitektur gebaute Computer verlangen ein vorgegebenes, strukturiertes Programm, über welches die Turingmaschine nicht verfügt. Aus der Struktur des Programms ist der Ablauf der Berechnungen nicht ersichtlich, da der im Programm festgelegte Ablauf unwichtig ist. Die letzte Aktion bestimmt die nächste Anweisung. Auch die Laufzeit einer Turingmaschine ist extrem lange. Da selbst einfache Aufgabe wie zum Beispiel das Kopieren eines Bandes auf ein anderes mehrere Abläufe verlangen, steigt der Zeitbedarf mit jedem Schritt enorm an.

### 2.2.2 Beispiel Registermaschine (Random access machine = RAM)

Die Registermaschinen haben wie die vorhin besprochenen Turingmaschine einen Speicher der aus einzelnen Zellen besteht. Dieser kann aber unendlich viel Information enthalten. Weiters ist eine direkte Adressierung möglich. Programm für Registermaschinen bestehen aus einer endlichen Anzahl an nummerierten Anweisungen. Da die Ausführung eines Programms auf einer Registermaschine deterministisch ist, realisiert ein Programm stets eine Funktion von den Eingabewerten zu den Ausgabewerten.

#### 2.2.2.1 Vor- und Nachteile einer Registermaschine

Getreues Modell der Zugriffsweise im Arbeitsspeicher einer v. Neumann-Architektur, sowie kurze und leicht verständliche Programme mit einfachen Rekursionen. Konkrete Rechner können gut nachgebildet werden für kleine Eingaben.
Als einziger Nachteil ist hier zu erwähnen, dass sowohl für die Anzahl der Register, als auch für deren Fassungsvermögen eine unbegrenzte Zahl angenommen wird. Dies ist aber nur eine Idealisierung, die in der Realität nicht durchführbar ist.

## 2.3 Funktionale Modelle

Bei funktionalen Modellen geht es weniger darum, wie etwas berechnet werden soll, als eher darum, was berechnet wird.
Wenn man von dem Bereich der natürlichen Zahlen ausgeht und nur die 0 und die Nachfolgefunktion bekannt ist, so lautet die Grundfrage funktionaler Berechnungsmodelle: Gibt es weitere Funktionen, und wenn ja, welche, die sich aus 0 und der Nachfolgefunktion berechnen lassen und sich für gegebene Parameter effektiv auswerten lassen.
Das Hauptaugenmerk liegt hier auf den Regeln, die es erlauben, aus bekannten, berechenbaren Funktionen neue berechenbare Funktionen zu kreieren. Dabei wird die genaue Arbeitsweise einer Funktion nicht betrachtet, sondern nur ihr Ergebnis. Die funktionalen Modelle sind daher für Abstraktion von Beispielen sehr geeignet.

## 2.4 Kommunikation und verteilte Systeme

In den vorherigen Kapiteln ging es ausschließlich um die numerische Berechnung von Funktionen. Durch die Betrachtung der Umgebung des Prozesses und die Kommunikation mit dieser, erhält man neue Sichtweisen.

## 2.4.1 EA- Turingmaschinen

Sie sind eine Abwandlung der bereits vorgestellten Turingmaschine. Denn hier stehen mehrere Bänder zur Verfügung. Ein Eingabeband, von dem nur gelesen wird, und ein Ausgabeband, auf welchem nur geschrieben wird, sind mit einem Arbeitsband im Steuerwerk verbunden. Auf den Ein- und Ausgabebändern darf nur nach rechts gegangen werden.

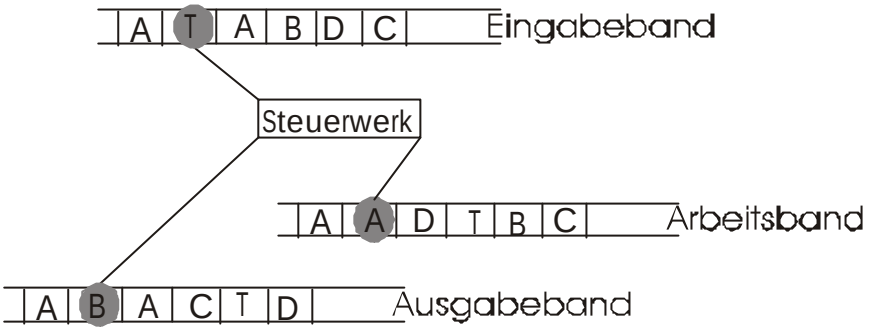


Abbildung 2: EA - Turingmaschine

## 2.4.2 Nicht abbrechende Berechnungen

Anders als die Turingmaschine aus Kapitel 2.2.1 kann die EA- Turingmaschine in eine nicht abbrechende Berechnung kommen. Dann werden beispielsweise keine Befehle vom Eingabeband mehr gelesen und ein unendlicher Strom an Ausgabe produziert. Um dies zu verhindern, werden sogenannte Grundzustände eingeführt. Diese markierten Zustände geben an, ob sich die Maschine regulär verhält, also immer wieder einen Grundzustand erreicht.

In einem Betriebssystem wäre solch ein Grundzustand etwa ein Zustand, in dem ein Befehl des Users akzeptiert wird. Das System verhält sich dann also regulär.

### 2.4.3 Verteilte Systeme

Zwei Turingmaschinen können miteinander kommunizieren, indem das Ausgabeband der einen, gleichzeitig das Eingabeband der anderen Maschine ist. Es ist hier nicht notwendig, die beiden Systeme zu synchronisieren. Wenn man schließlich endlich viele Turingmaschinen auf diese Weise verknüpft, spricht man von einem Datenflussnetz.

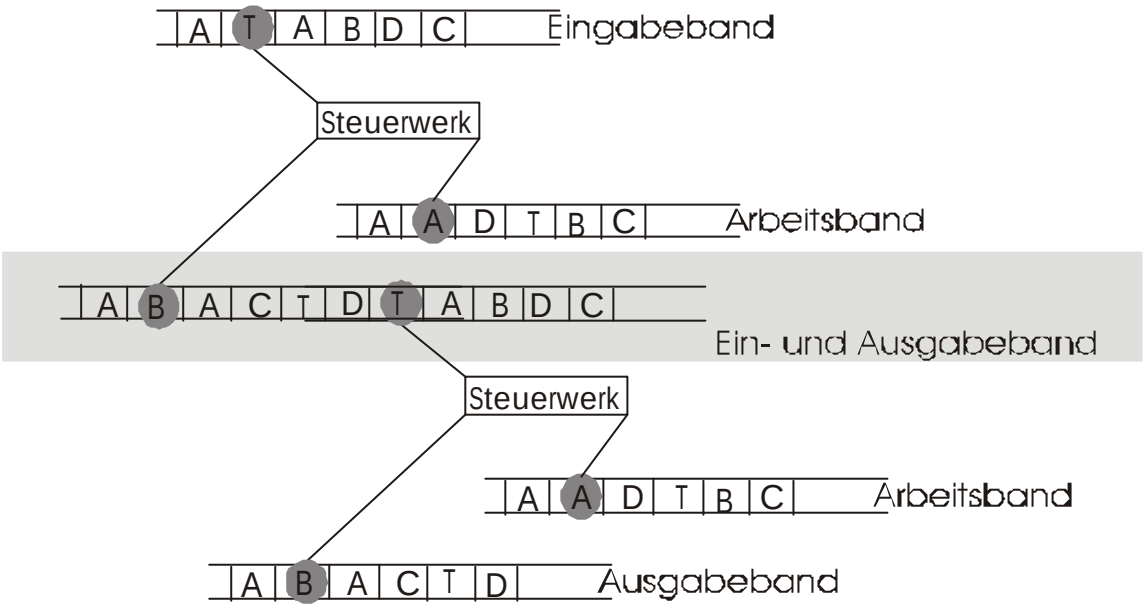

Abbildung 3: Verteilte Systeme

# 3 Grenzen der Berechenbarkeit

Katrin Zöchmeister

## 3.1 Einleitung

Moderne Rechner haben die Welt verändert. Es scheint so, als ob sie alles können und keine Grenzen haben. Aber entspricht das der Realität? Und wenn nicht, wo liegen die Grenzen moderner Computer?

### 3.1.1 Universelle Rechner

Universelle Rechner sind frei programmierbar. Im Gegensatz dazu verfügen Turing- und Registermaschinen über festgelegte Programme und können dadurch nur ein Problem lösen. Moderne Rechner sind mit universellen Programmen ausgestattet. Dies ermöglicht ihnen spezielle Programme auf spezielle Daten anzuwenden.
Doch dies ist keine Erfindung von modernen Rechnern, denn schon früher wurden universelle Turing- und Registermaschinen entwickelt.

### 3.1.2 Simulationen

Das ist ein Verfahren, mit dem Programme von einem Rechnertyp für einen anderen Rechnertyp so nachgeahmt werden, dass sie die selben Ein- und Ausgabewerte haben. Das heißt, wenn sich Rechner gegenseitig simulieren können, können sie auch die gleichen Programme berechnen und daher die selben Probleme lösen. Es ist nur sinnvoll, in einem solchen Fall eine Theorie der Berechenbarkeit zu entwickeln. Sonst würde man für jeden Rechner eine neue Theorie aufstellen müssen.
Ein Beispiel für eine solche Simulation ist das Ersetzen einer mehrdimensionalen Turingmaschine (also mit mehreren Bändern) durch einfache Turingmaschinen. Dabei wächst die Rechenzeit um einen konstanten Faktor (t), welcher auf die aufwendigere Verwaltung des Speichers zurückzuführen ist.

### 3.1.3 Churchsche These

Funktionen, die (Turing-) berechenbar sind, können durch eine Turingmaschine berechnet werden. Bis heute hat man keinen Rechner gefunden, der Probleme lösen und Programme berechnen kann, welche auf den Turing- oder Registermaschinen nicht zu berechnen sind.

Die Churchsche These besagt, dass die Klasse der wirklich berechenbaren Funktionen gleich groß wie die Klasse der Turing- berechenbaren Funktionen ist. Diese These ist jedoch nicht beweisbar.

## 3.2 Existenz nichtberechenbarer Funktionen

Wenn eine Funktion als Ergebnis entweder 0 oder 1 hat, dies aber nicht beweisbar ist (z.b. Axiome in der Mathematik), so entscheidet ein Programm, ob 0 oder 1 als Ergebnis geliefert wird. Für die Berechenbarkeit ist also die Existenz eines Turingmaschinenprogramms wichtig, nicht aber, ob jemals ein Programm gefunden werden kann, welches das Problem löst. Denn das Turingmaschinenprogramm entscheidet nach Eingabe UND Aktion, welches Ergebnis geliefert wird.
Die Existenz nichtberechenbarer Probleme ist leicht nachzuweisen. Die Menge aller korrekten Programme für eine Turingmaschine ist abzählbar unendlich, da jedes Programm eine endliche Zeichenkette über einem endlichen Alphabet ist. Auf der anderen Seite gibt es unendliche Funktionen für eine bestimmte Abbildung. Da jedes Programm schließlich nur eine Funktion berechnen kann, müssen nichtberechenbare Funktionen existieren.

## 3.3 Das Halteproblem

Dabei soll für eine Turingmaschine entschieden (berechnet) werden, ob es in endlicher Zeit möglich ist, vom Anfangszustand in den Endzustand zu gelangen. Endlosschleifen zum Beispiel gehören zu den am meisten gemachten Fehlern in der Programmierung, doch auch hier wird nie ein Endzustand erreicht. Wenn das Halteproblem lösbar wäre, könnte man einem Programm einfach ein anderes vorschalten, welches die Endlosschleifen vermeidet.

## 3.4 Softwareverifikation

Wie uns allen bekannt, machen Rechner nicht immer das, was sie machen sollten. Dies liegt an dem Problem, dass weder Software noch Hardware verifiziert sind. Die

Softwareverifikation besteht nun darin, bei einer angegebenen Eingabe, die in diesem Fall aus einem Programm besteht, zu überprüfen, ob Verifikation und ursprüngliches Programm übereinstimmen.
Da es aber kein Programm gibt, dass bei beliebiger Eingabe entscheidet, ob das Programm hält oder nicht (siehe Halteproblem), ist es genauso unmöglich, zu sagen, ob 2 Programme bei gleicher Eingabe gleiche Ausgaben liefern.

# 4 Formale Sprachen und Automaten

Elisabeth Grill

## 4.1 Einleitung

Diese Themen sind deshalb interessant, da sie nicht unbedingt theoretisch sind. (wird benötigt für: Compilerbau, Sprachentwurf,...)

Informatik beschäftigt sich mit der mechanischen Verarbeitung, Übermittlung, Speicherung und Wiedergewinnungen von Informationen und mit der Konstruktion von Systemen, die diesem Zweck dienen.
Es werden nur die Repräsentationen der Informationen verarbeitet und nicht die Information selbst. Diese Informationen werden durch Zeichenketten über einem Alphabet repräsentiert.
Eine (Formale) Sprache ist aber nichts anderes als eine Menge von Zeichenketten (Wörtern) über einem Alphabet.
- Beschreibung von Klassen von Informationen
- Beschreibung von Struktur von Informationen

Ein Automat ist ein abstraktes Modell eines mechanischen Systems zur Verarbeitung von Informationen (Zeichenreihen)
- Abstraktion von technischen Details
- Konzentration auf das Wesentliche
- Klassifikation von Automatentypen und Untersuchung von Beziehungen

Mehr dazu im folgenden Kapitel.

## 4.2 Grundbegriffe

### 4.2.1. Alphabete und Sprachen

Ein Alphabet ist eine endliche Menge von Zeichen (Symbolen). Ein Beispiel ist das Alphabet ? ={a,b,c}. Eine endliche Sequenz $a_1, a_2, \ldots a_n$ ? i

Ein Alphabet ? ist eine endliche Menge von Zeichen (Symbolen). Durch das Hintereinanderschreiben von Zeichen entstehen Zeichenketten wie aa, ba, ca.
Ein Wort w über ? ist eine endliche Sequenz von Symbolen aus ?, wobei |w| die Wortlänge, ? das Leerwort, und ?* die Menge aller Worte, (einschließend ?) über ? bezeichnen.

**Definition (formale Sprache).**
*Eine formale Sprache (oder kurz Sprache) L über dem Alphabet ? ist eine Teilmenge von ?*: L ? ?*. Ihre Elemente heißen Sätze (in der mathematisch orientierten Literatur meist Wörter (Zeichenreihen, Zeichenketten)).*

Auch L=? (die leere Sprache) und L={?} (die Sprache, die aus dem leeren Satz besteht) und L=?* (die Allsprache) sind formale Sprachen über ?.

**Definition (Wort).**
*Ein Wort w? ?* ist eine Funktion w: {1,..., |w|}? ?, wobei w(i), i=1,...,|w| das i-te Symbol in Wort w ist.*
**Definition (Konkatenation).**

w=x?y

1. |w|=|x|+|y|,
2. w(i)=x(i), i=1,...|x|,
3. w(|x|)+i)=y(i), i=1,...|y|.

Notation: $w^0 = ?$, $w^{i+1} = w^i ? w$

Beispiel: $(bla)^2 = blabla$

### 4.2.2 Grammatiken, rekursive Grammatiken

Grammatiken bestehen aus Regeln (Ersetzungsregeln, Produktionsregeln), die ihrerseits zwei Arten von Symbolen enthalten: Terminalsymbole, die in der Zeichenkette selbst vorkommen und Nonterminalsymbole, die nur zur Strukturbeschreibung genutzt werden.

**Definition (Grammatik).**

*Eine Grammatik G ist ein Quadrupel G=(?$_N$, ?$_T$, P,S)*

*?$_N$ = Alphabet der Nonterminalsymbole,*

*?$_T$ = Alphabet der Terminalsymbole,*

*P = Menge von Ersetzungsregeln der Form ? ? ?, wobei ? ? ? *?$_N$ ? * und ? ? ? *,*

*S = Satzsymbol (Startsymbol) aus ?$_N$*

*Es ist ?$_N$ ? ?$_T$ = ? , ?$_N$ ? ?$_T$ = ?*

Praktisch anwendbar sind besonders Grammatiken, bei denen ? ? ?$_N$ ist (kontextfreie Grammatiken)

**Definition (Satzform, Satz, Sprache).**

*G=(?$_N$, ?$_T$, P,S) sei eine Grammatik. Die Kette ? wird Satzform genannt, wenn S? *?. Ein Satz ist eine Satzform, die nur aus Terminalsymbolen besteht.*

*Die Sprache L(G) ist die Menge aller Sätze: L(G)={? : S ? *? und ? ? ?$_T$*}*

**Rekursive Grammatiken.**

Zur Beschreibung von Sprachen mit unendlich vielen Sätzen bedient man sich rekursiver Grammatiken, das sind solche, die Ableitungen der Form A? $w_1Aw_2$ mit $w_1, w_2$? ? * gestatten. Man unterscheidet linksrekursive Regeln (A? ? |A? ), rechtsrekursive Regeln (A? ? |? A) und zentralrekursiven Regeln (A? ? |? A?).

### 4.2.3 Kanonische Ableitungen und Mehrdeutigkeit

Eine Satzform mit mehreren Nonterminalsymbolen kann auf mehrere Weisen abgeleitet werden.

**Definition (Links- und rechtskanonische Ableitung, Mehrdeutigkeit).**
*Eine direkte Ableitung heißt linkskanonisch oder Linksableitung, wenn A das linkeste Nonterminalsymbol ist. In entsprechender Weise werden rechtskanonische Ableitungen oder Rechtsableitungen (? $^{R}$) und die Hüllenoperationen von beiden (? $^{L+}$? $^{L*}$ ? $^{R+}$ ? $^{R*}$) definiert.*

Eine Grammatik G heißt mehrdeutig, wenn die von ihr erzeugte Sprache Sätze enthält, die mehr als eine links- oder rechtskanonische Ableitung und damit mehr als einen Syntaxbaum haben. Grammatiken, die nicht mehrdeutig sind, heißen eindeutig.

## 4.3 Die Chomsky-Hierarchie

N. Chomsky hat 1959 die Grammatiken in eine vierstufige Hierarchie eingeteilt, die seitdem als Gliederung genutzt wird.

**Definition (Chomsky´s Grammatik-Typen).**
Eine Grammatik *G=(?* $_{N,}$ *?* $_{T,}$ *P,S) heißt*

- *vom Typ 0 oder unbeschränkt, wenn ihre Regel gemäß den oben genannten Nennungen aufgebaut sind;*
- *vom Typ 1 oder kontextsensitiv, wenn in jeder Regel ? ? ? die linke Seite nicht länger als die rechte ist: |? |?|? |; die Regel S? ? ist zugelassen, aber wenn sie vorkommt, darf S auf keiner rechten Regelseite stehen;*
- *vom Typ 2 oder kontextfrei, wenn jede Regel die Form A? ? mit A? ?* $_{N}$ *hat;*
- *vom Typ 3 oder regulär, wenn jede Regel eine der Formen A? ? oder A? ? B hat, mit A,B? ?* $_{N}$ *und ? ? ?* $_{T}$**. (Diese Grammatiken heißen auch rechtslinear, weil*

*ihre Regeln auf der rechten Seite höchstens ein Nonterminalsymbol und das am Ende enthalten.)*

Die von den Grammatiktypen erzeugten Sprachen haben die gleichen Namen: Unbeschränkte, kontextsensitive, kontextfreie und reguläre Sprachen. Die Hierarchie-Eigenschaft spricht folgender Satz aus:

**Satz (Chomsky-Hierarchie).**
Wenn $L_i$ die Menge aller Sprachen bedeutet, die von einer Grammatik vom Typ i erzeugt werden, dann ist $L_0$ ? $L_1$ ? $L_2$ ? $L_3$.

## 4.4 Reguläre Ausdrücke und endliche Automaten

### 4.4.1 Reguläre Mengen und reguläre Ausdrücke

Mit regulären Ausdrücken lassen sich einige unendliche Sprachen mit endlichen Mitteln generieren.

**Definition (Reguläre Ausdrücke).**

1. *? und jedes Element des Alphabets ? ist ein regulärer Ausdruck*
2. *Wenn ? und ? reguläre Ausdrücke sind, dann sind auch (? ? ), (? ? ? ), und ?* $^*$ *reguläre Ausdrücke.*

**Definition (Reguläre Sprachen/Mengen).**
*Die Menge der regulären Ausdrücke über einem Alphabet ? ist induktiv definiert durch:*

1. *? ist ein regulärer Ausdruck*
2. *? ist ein regulärer Ausdruck*
   *? können wir auch sparen, da (semantisch) gilt {?}=? *.*
3. *Für a ? ? ist a ein regulärer Ausdruck*
4. *Für zwei reguläre Ausdrücke r und s sind*

*Syntax!*

*(r*? s) = *(r+s)*

*(rs) und*

*(r*)*

*reguläre Ausdrücke.*

Zur Vermeidung von Mengenklammern beschreibt man reguläre Mengen durch reguläre Ausdrücke, indem man „? “ durch „+“ ersetzt.

Beispiel: Regulärer Ausdruck ? : (((a? b)*(aa))(a? b)*),
vereinfacht: (a? b)*aa(a? b)*, beschreibt Menge L(? )={w? {a,b}*| in w kommen zwei as nacheinander vor}.

*Die Semantik (Bedeutung) eines regulären Ausdrucks ist eine Sprache über ? . Die durch einen regulären Ausdruck bezeichnete Sprache ist induktiv definiert durch:*

1. *? bezeichnet die leere Sprache ?*
2. *? bezeichnet die Sprache {?}*
3. *a bezeichnet die Sprache {a}*
4. *Bezeichnen r die Sprache R und s die Sprache S, so bezeichnen*

   *(r+s) die Sprache R ? S*

   *(rs) die Sprache R ? S*

   *(r*) die Sprache R**

*Semantik!*

*Syntax* *Semantik*

Eine Sprache heißt regulär, wenn sie durch einen regulären Ausdruck bezeichnet wird.

### 4.4.2 Endliche Automaten

Endliche Automaten sind mathematische Modelle zur Beschreibung des Verhaltens digitaler Systeme. Man unterscheidet *endliche Automaten* mit Ausgabe (für die

Modellierung von Schaltwerken) und *erkennende endliche Automaten* (für die Erkennung regulärer Sprachen). Erkennende endliche Automaten weisen gegenüber endlichen Automaten mit Ausgabe folgende charakteristische Unterschiede auf:

1. Sie liefern während des Lesens und Verarbeitens einer Eingabe-Zeichenkette keine Ausgabe.

2. Sie haben ausgezeichnete Zustände: Den *Anfangszustand* in dem sie starten, und *Endzustände*, in denen sie anhalten können, wodurch sie die eingegebene Zeichenkette als erkannt signalisieren.

3. Sie existieren in zwei Varianten: als *deterministischer* und als *nichtdeterministischer* endlicher Automat.

Ein erkennender endlicher Automat (Abbildung 11) besteht aus einer endlichen Anzahl von Zuständen $z_1$ bis $z_n$, einem Eingabeband endlicher Länge und einem Lesekopf. Einer der Zustände ist als Startzustand, ein oder mehrere Zustände sind als Endzustände ausgezeichnet. Das Band enthält eine Zeichenkette $s_1$ bis $s_m$. Der Automat hat die Aufgabe, durch schrittweises Lesen der Zeichenkette festzustellen, ob sie eine bestimmte, durch die Übergangsfunktion des Automaten festgelegte Struktur hat.

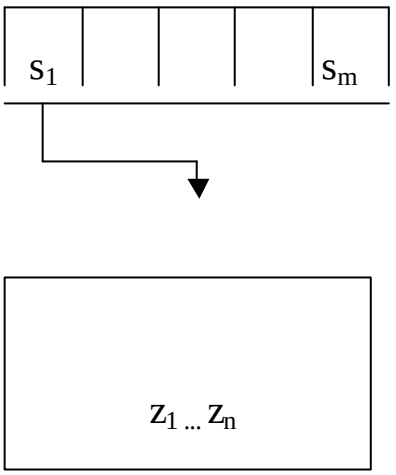


Abbildung 4: Endlicher Automat mit n Zuständen zur Erkennung der Kette $s_{,...,}s_m$

Mit endlichen Automaten läßt sich die Frage w? L? für reguläre Sprachen algorithmisch entscheiden.

**Definition (Deterministisch endlicher Automat (DEA)).**

*Ein DEA ist ein Quintupel M=(K, ? , ? , S, F) sodass*

1. *K eine endliche Menge von Zuständen,*
2. *? ein Alphabet,*
3. *S ? K ein Anfangszustand,*
4. *F ? K eine Menge von Endzuständen*
5. *? : K? ? ? K eine Übergangsfunktion ist*

*Die von einem DEA akzeptierte Sprache, ist die Menge aller des DEA akzeptierten Wörter.*

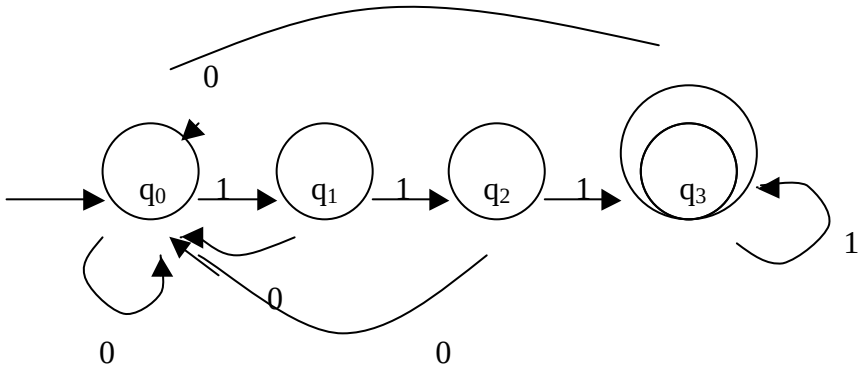


Abbildung 5: Ein deterministischer Automat für Wörter, die mit mindestens drei Einsern enden

**Definition (Nicht Deterministischer endlicher Automat (NEA)).**

*Ein NEA ist ein Quintupel M=(K, ? , ? , S, F) und ist definiert wie ein DEA außer der Übergangsfunktion ? auf K? ? *? K.*

*Ein nichtdeterministischer endlicher Automat kann von einem Zustand aus bei einem bestimmten Eingabezeichen in mehrere Folgezustände übergehen.*

*Die von einem NEA akzeptierte Sprache, ist wieder die Menge aller des NEA akzeptierten Wörter.*

Für manche Sprachen ist der entsprechende deterministische Automat sehr groß. Oft sind nichtdeterministische Automaten kleiner, übersichtlicher und einfacher zu konstruieren.

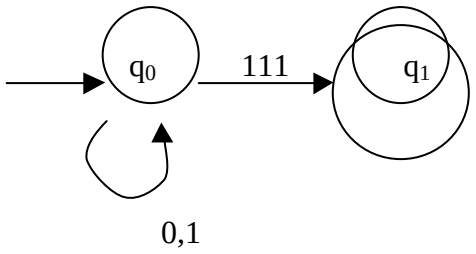


Abbildung 6: Ein nicht-deterministischer Automat für Wörter, die mit mindestens drei Einsern enden

Bemerkungen:

- Ein nicht-deterministischer Automat ist oft einfacher und übersichtlicher als der entsprechende deterministische Automat.

- Ein nicht-deterministischer Automat ist oft einfacher zu konstruieren (aus anderen Repräsentationen bzw. aus einer informellen Beschreibung der Sprache) als ein entsprechender Automat.

- Zum erkennen einer Sprache ist ein deterministischer Automat besser geeignet.

- Manche Operationen auf Sprachen lassen sich einfacher mit deterministischen Automaten durchführen (beispielsweise die Konstruktion eines Automaten für das Komplement einer Sprache); andere Operationen auf Sprachen lassen sich einfacher mit nicht-deterministischen Automaten durchführen (beispielsweise die Konstruktion eines Automaten, der die Vereinigung zweier Sprachen akzeptiert).

Deshalb sind beide Varianten sinnvoll. Besonders schön ist es, wenn man für jeden nicht-deterministischen Automaten *effektiv*[1] einen entsprechenden deterministischen Automaten konstruieren kann. Diese Überführung für endliche Automaten ist immer

[1] Effektiv bedeutet dabei, dass es ein automatisches Verfahren zur Überführung eines nicht-deterministischen Automaten in einen deterministischen gibt. Noch schöner ist es, wenn das Verfahren effizient durchgeführt werden kann (Achtung: effektiv ? effizient).

effektiv möglich. Für andere Automatentypen ist dies nicht immer möglich; beispielsweise kann nicht jeder nicht-deterministische Kellerautomat in einen entsprechenden deterministischen Kellerautomaten überführt werden.

Zwei endliche Automaten $FA_1$ und $FA_2$ heißen äquivalent, wenn sie dieselbe Sprache erkennen: $L(FA_1)=L(FA_2)$. Es gibt Algorithmen, die die Äquivalenz von endlichen Automaten feststellen und die zugleich einen äquivalenten endlichen Automaten mit minimaler Zustandszahl liefern.

Satz: Äquivalenz nichtdeterministischer und deterministischer endlicher Automaten

Jeder nichtdeterministische endliche Automat läßt sich in einen äquivalenten deterministischen endlichen Automaten transformieren.

## 4.5 Kontextfreie Sprachen und Kellerautomaten

### 4.5.1 Charakterisierung

Reguläre Sprachen stellen eine einfache Möglichkeit da, Sprachen zu definieren und auch zu analysieren. Außerdem haben sie sehr schöne Eigenschaften: Sie sind unter vielen Operationen abgeschlossen, und die meisten Probleme sind für reguläre Sprachen entscheidbar.

Allerdings sind viele praktisch relevanten Sprachen nicht regulär. Beispielsweise ist die Sprache der korrekten Klammerausdrücke nicht regulär. Generell lassen sich verschachtelte Strukturen (beliebiger Tiefe) nicht als reguläre Sprache formulieren, da man mit regulären Sprachen nicht (beliebig weit) zählen kann. Verschachtelte Strukturen kommen aber in vielen Bereichen der Informatik vor (z.B. in Ausdrücken oder bei ineinander verschachtelten Schleifen oder Alternativen).

Eine Sprachklasse, die auf die Definition verschachtelter Strukturen zugeschnitten ist, ist: die *kontextfreie Sprachen*.

Die Fähigkeit kontextfreier Grammatiken, Sprachen zu definieren, die nicht regulär sind, beruht auf den zentralrekursiven Regeln (direkt A? ? A?, indirekt A? ? B?, B? ?A?). Sie erlauben die Beschreibung von Symmetrie (Klammerstrukturen). Die Zentralrekursivität führt im Verein mit der Endlichkeit des Nonterminalalphabets zu der folgenden Charakterisierung der kontextfreien Sprachen:

**Definition (uvwxy-Satz (Schleifensatz, pumping lemma)).**

Alle Sätze ? einer kontextfreien Sprache L, die mindestens eine (von der Sprache abhängige) Länge n haben, lassen sich in fünf Abschnitte *u, v, w, x, y* zerlegen, so dass gilt

1. ? = u v w x y mit |v w x| ? n, |v x| ? 1.
2. Alle Ketten der Form $u\ v^i\ w\ x^i\ y$ für i ? 0 sind ebenfalls Sätze von L.

Der uvwxy-Satz gestattet es zu zeigen, dass gewissen Sprachen nicht kontextfrei sind. So ist z.B. die Sprache $\{a^n\ b^n\ c^n : n\ ?\ 0\}$ nicht kontextfrei, da es unmöglich ist, den Satz $a^n\ b^n\ c^n$ für festes n so in fünf Bestandteile u, v, w, x, y zu zerlegen, dass dann auch die Ketten $uv^2wx^2y, uv^3wx^3y, \ldots$ zur Sprache $\{a^n\ b^n\ c^n\}$gehören.

## 4.5.2 Grammatiktransformationen und Normalformen

Für manche Zwecke ist es wünschenswert, eine gegebene Grammatik $G_1$ mit ungünstigen Eigenschaften in eine äquivalente Grammatik $G_2$ zu transformieren, die die ungünstigen Eigenschaften nicht aufweist. Die wichtigsten solcher Transformationen sind

1. die Beseitigung leerer Alternativen (A? ?);
2. die Beseitigung linksrekursiver Regeln (A? A? |...);
3. die Beseitigung nonterminaler Anfänge (A? B? , B ? ? $_N$);
4. die Beseitigung einelementiger Regeln (A? B, B ? ? $_N$);
5. die Linksfaktorisierung (aus A? ? ? | ? ? mache A? ? B, B? ? |?).

Beim Entwurf einer Grammatik kann es vorkommen, dass sich Symbole oder Regeln einschleichen, die überflüssig sind oder zu *zirkulären Ableitungen* wie A ? $^+$ A führen. Ein Nonterminalsymbol heißt *überflüssig*, wenn es in keiner Satzform vorkommt (d.h. nicht aus S ableitbar ist) oder wenn es nicht in eine terminale Zeichenkette abgeleitet werden kann (wie A? ? A ohne nichtrekursive Alternative).

Es gibt Algorithmen, die überflüssige Symbole und Zirkularitäten feststellen [Aho72].

Es gibt *zwei Normalformen*, in die sich alle ?-freien Grammatiken transformieren lassen:

?? *Chomsky-Normalform (CNF)*: Alle Regeln haben eine der Formen A? BC oder A? b mit A, B, C ? $?_N$ und b ? $?_T$.

?? *Greibach-Normalform*: Alle Regeln haben die Form A? a? mit A ? $?_N$, a ? $?_T$, ? ? $?_N$*.

Sie sind für manche Beweisführungen nützlich, haben aber keine praktische Bedeutung.

### 4.5.3 Kellerautomaten

Zur Erkennung kontextfreier Sprachen reichen endliche Automaten nicht aus, sondern man braucht zusätzlich einen Speicher zur Zwischenspeicherung von Zeichen. Da die Kapazität des Zwischenspeichers unbeschränkt sein muss, aber das zuletzt gespeicherte Zeichen als erstes wieder entfernt wird, benutzt man einen Kellerspeicher (Abbildung 4) und nennt den so erweiterten endlichen Automaten *Kellerautomaten*.

**Definition (Kellerautomat (PDA)).**

*Ein Kellerautomat ist ein Sextupel M=(K, ? , ? , ? , S, F), wobei*

1. *K eine endliche Menge von Zuständen,*
2. *? ein Alphabet von Eingabesymbolen,*
3. *? ein Alphabet von Kellersymbolen,*
4. *S ? K ein Startzustand,*
5. *F ? K eine Menge von Endzuständen,*
6. *? eine Übergangsrelation auf (K ? ? * ? ? *) ? (K ? ? *).*

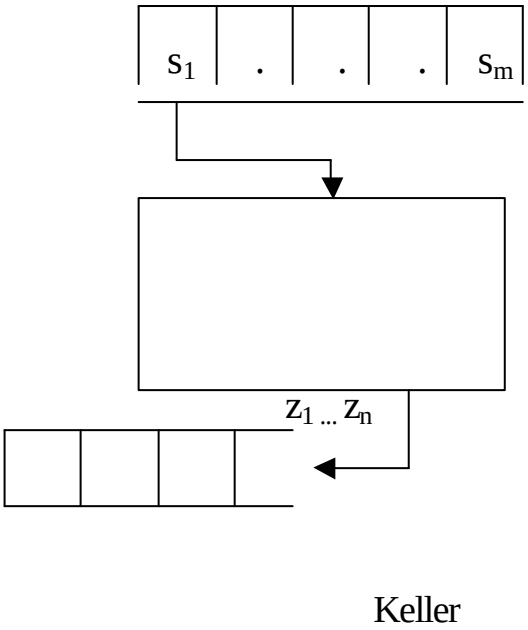


Abbildung 7: Kellerautomat mit n Zuständen zur Erkennung der Kette $s_{1,...,}s_m$

Ähnlich wie bei endlichen Automaten für reguläre Sprachen, gibt es verschiedene Varianten von Kellerautomaten; im Gegensatz zu den endlichen Automaten sind diese Automaten nicht alle äquivalent zueinander. Insbesondere sind die deterministischen Kellerautomaten nicht so ausdrucksmächtig wie die nicht-deterministischen Kellerautomaten, dafür jedoch praxisrelevanter.

Zunächst überlegen wir uns, wie ein Automat aussehen könnte, der eine Sprache $a^n b^n$ akzeptiert. Offentsichtlich muss ein Automat für diese Sprache „zählen“ können, da er überprüfen muß, ob die Zahl der *a´s* und *b´s* übereinstimmt. Der Automat muss also einen beliebig großen Speicher besitzen. Der Automat könnte das Wort einlesen, und dabei jedes gelesene Zeichen a auf einen *Keller* legen. Wenn der Automat auf das erste Zeichen *b* stößt, entfernt er für jedes gelesene b ein a vom Keller. Wenn am Ende alle b´s der Eingabe gelesen wurden, und kein a mehr im Keller steht, dann akzeptiert der Automat das Wort. Mit Hilfe des Kellers können wir also zählen; tatsächlich können wir den einen Keller als Verallgemeinerung eines Zählers auffassen.

Natürlich muss der Automat auch sicherstellen, dass kein a mehr im Eingabewert vorkommt, wenn bereits ein b gelesen wurde. Diese können wir wie bei einem endlichen Automaten sicherstellen. Ein Kellerautomat ist also ein um einen Keller erweiterter endlicher Automat.

### 4.5.4. Kontextfreie Grammatiken und Kellerautomaten

Zu jeder kontextfreien Grammatik G lässt sich ein Kellerautomat konstruieren, der genau die Sprache von G erkennt, und zu jedem Kellerautomaten PDA lässt sich eine kontextfreie Grammatik konstruieren, die genau die Sprache von PDA erzeugt. Daraus folgt der wichtige Satz:

*Äquivalenz der Sprachen von kontextfreien Grammatiken und Kellerautomaten*

Die Menge der von kontextfreien Grammatiken erzeugten Sprachen und die Menge der von Kellerautomaten erkannten Sprache ist gleich.

## 4.6 Weitere Sprachklassen

Neben der Chomsky-Hierarchie wurden auch andere Regelsysteme zur Definition formaler Sprachen untersucht, wie Matrixgrammatiken, programmierte Grammatiken, geordnete Grammatiken und Lindenmayer-Systeme. Eine Einführung in diese Themen mit vielen bibliographischen Angaben bietet [Salo78]. Neben Zeichenketten lassen sich auch Graphen durch Regeln zu ihrer Erzeugung beschreiben, was auf Graph-Grammatiken führt [Gött88].

## 4.7 Eine Hierarchie von Sprachen

Abbildung 16 fasst den Inhalt der Kapitel zusammen und zeigt die Hierarchie der zum Teil behandelten Sprachen. Das Beispiel zu Sprachklasse n ist so gewählt, dass es nicht auch zu Sprachklasse n-1 gehört.

| Nr | Name | Definition durch | Beispiel |
|---|---|---|---|
| 1 | Endliche Mengen | Aufzählung | {a, bbc, bac} |
| 2 | Reguläre Mengen | Reguläre Ausdrücke | zz*+“+“zz*+“-?zz* |
| | | Rechtslineare Grammatiken | $\{a^n : n ? 0\}$ |
| | | Endliche Automaten | |

| | | | |
|---|---|---|---|
| 3 | Kontextreie Sprachen | Kontextfreie Grammatiken<br>Kellerautomaten | $\{a^n b^n : n ? 0\}$ |
| 4 | Kontextsensitive Sprachen | Kontextsensitive Grammatiken<br>Linear beschränkte Automaten | $\{a^n b^n c^n : n ? 0\}$ |
| 5 | Turing-entscheidbare Sprachen | Stets haltende Turingmaschinen | $L_1$ (siehe unten) |
| 6 | Turing-erkennbare Sprachen | Unbeschränkte Grammatiken<br>Turingmaschinen | $L_2$ (siehe unten)<br>Menge aller Turing maschinen<br>Menge aller Algorithmen |
| 7 | Abzählbare Mengen | | $L_3$ (siehe unten) |
| 7 | Überabzählbare Mengen | | Menge aller reellen Zahlen<br>Menge aller Formalen Sprachen |

$L_1 = \{? _i : ? _i ? ? _T^* \text{ und } ? _i \text{ ist nicht Satz der kontextsensitiven Grammatik i}\}$

$L_2 = \{? _i : ? _i ? ? _T^* \text{ und } ? _i \text{ wird von Turingmaschine i erkannt}\}$

$L_3 = \{? _i : ? _i ? ? _T^* \text{ und } ? _i \text{ wird nicht von Turingmaschine i erkannt}\}$

Abbildung 8: Eine Hierarchie von Sprachen

## Symbolverzeichnis

| | |
|---|---|
| ? ? | trennt linke und rechte Seite von Grammatikregeln |
| ? , ? $^+$, ? * | direkte und indirekte Ableitungen |
| ? $^L$, ? $^{L+}$, ? $^{L*}$ | linkskanonische Ableitungen |
| ? $^R$, ? $^{R+}$, ? $^{R*}$ | rechtskanonische Ableitungen |
| \| | Alternativen-Trennsymbol |
| [...] | Optionssymbol |
| {...} | Mengenklammern, Wiederholungssymbol |
| ? | leere Menge |

| | |
|---|---|
| ? | leere Zeichenkette |
| $a^n$ | Kette aus n a-Symbolen |
| $a^+$ | {a, aa, aaa, ...} = { $a^n$: n ? 1} |
| a* | {?, a, aa, aaa, ...} = { $a^n$: n ? 1} |
| ? , ? , ...,? | Zeichenketten |
| $?^R$ | Spiegelung von ? , d.h. $(abc)^R$ = cba |
| \| ? \| | Länge von ? |
| ?? | Übergangsfunktion eines Automaten |
| DFA | Deterministischer endlicher Automat |
| F | Menge der Endzustände eines Automaten |
| FA | endlicher Automat |
| G | Grammatik |
| L | Sprache |
| ?L | Komplement einer Sprache = ? * - L |
| NFA | Nicht-deterministischer endlicher Automat |
| P | Regelmenge einer Grammatik |
| PDA | Kellerautomat |
| S | Satzsymbol, Kelleranfangssymbol eines Automaten |
| TM | Turingmaschine |
| ? , $?_T$, $?_N$ | Alphabet, Terminal-, Nonterminalalphabet |
| z, Z | Zustandsmenge |

# Zusammenfassung

Die behandelten Themen Aussagenlogik, Berechnungsmodelle, Grenzen der Berechenbarkeit und Formale Sprachen und Automaten sind Bestandteile der Theoretischen Informatik, wenngleich Theorie nicht gleichzeitig trockener Lernstoff bedeutet.

Es wurde in dieser Arbeit versucht diese Themen ausreichend zu behandeln, wobei teilweise gerade Details notwendig für Erklärungen sind und deshalb genauer auf einzelne grundlegende Elemente eingegangen wurde anstatt oberflächlich dem Thema entgegenzutreten.

Die Arbeit erhebt keinen Anspruch auf Vollständigkeit, soll aber für Studierende einen einfachen und verständlichen Zugang zum Gebiet der theoretischen Informatik darstellen.

# Literaturverzeichnis

## Kapitel 1

[Bühl92] A. Bühler, Einführung in die Logik, Argumentation und Folgerung, Karl Alber Verlag, München, 1992

[Copi98] I. Copi, Einführung in die Logik, Wilhelm Fink Verlag, München, 1998

[Erns00] H. Ernst, Grundlagen und Konzepte der Informatik, Eine Einführung in die Informatik ausgehend von den fundamentalen Grundlagen, 2. Auflage, Vieweg & Sohn Verlagsgesellschaft, Braunschweig, Wiesbaden, 2000

[Jung99] A. Jung, Logik, in: P. Rechenberg, G. Pomberger (Hrsg.), Informatik-Handbuch, 2.Auflage, Carl Hanser Verlag, München, Wien, 1999, S. 33 - 72

[KlWü76] R. Kleinknecht, E. Wüst, Lehrbuch der elementaren Logik, Band 1: Aussagenlogik, Deutscher Taschenbuch Verlag, 1976

[Kreo91] H.-J. Kreowski, Logische Grundlagen der Informatik, (Handbuch der Informatik, Bd. 1.1), Oldenbourg Verlag, München, Wien, 1991

[Kuts67] F. v. Kutschera, Elementare Logik, Springer-Verlag, Wien, New York, 1967

[Mate69] B. Mates, Elementare Logik, Prädikatenlogik der ersten Stufe, Vandenhoeck und Ruprecht, Göttingen, 1969

[Ober92] A. Oberschelp, Logik für Philosophen, BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1992

[Schö95] U. Schöning, Logik für Informatiker, 4. Auflage, Spektrum Akademischer Verlag, Berlin, Heidelberg, Oxford, 1995

[ZeMa99] P. Zeitz, B. Mahr, Aussagenlogik, in: F. Cornelius, H. Ehring, M. Große-Rhode, B. Mahr, P. Zeitz, Mathematisch-strukturelle Grundlagen der Informatik, Springer-Verlag, Barcelona, Berlin, Heidelberg, Hongkong, London, Mailand, New York, Paris, Singapur, Tokio, 1999, S.199 - 304

[Zogl97] T. Zoglauer, Einführung in die formale Logik für Philosophen, Vandenhoeck und Ruprecht, Göttingen, 1997

## Kapitel 2

[Fran00] Christian Franz: Vorlesungsmitschrift: Theoretische Informatik II, 2000, Potsdam (http://ddi.cs.uni-potsdam.de/Lehre/Theorie2/Mitschriften/20001201b.pdf)

[RePo99] Peter Rechenberg, Gustav Pomberger: Informatik Handbuch. Carl Hanser Verlag 1999 München Wien

## Kapitel 3

[Gärt99] Marco Gärtler, Dagmar Handke: Theoretische Grundlagen der Informatik, 1999, Konstanz (http://www.inf.uni-konstanz.de/algo/lehre/ss03/theo/skript.pdf)

[RePo99] Peter Rechenberg, Gustav Pomberger: Informatik Handbuch. Carl Hanser Verlag 1999 München Wien

## Kapitel 4

[KiMa01] Ekkart Kindler, Steffen Manthey: Automaten Formale Sprachen Berechenbarkeit I. Skript zur Vorlesung im WS 2001/02 der TU München

[RePo99] Peter Rechenberg, Gustav Pomberger: Informatik Handbuch. Carl Hanser Verlag 1999 München Wien

[Riez99] Stefan Riezler: Formale Sprachen und Automaten. Universität Stuttgart, WS 99/00
Institut für maschinelle Sprachverarbeitung