Informatik-Tag Berlin-Brandenburg 2010

„Formale Sprachen”

Walter Gussmann (Paul-Natorp-Oberschule)

Inhaltsverzeichnis

1 Sprachen

Jede Sprache besteht aus Wörtern, die nach gewissen Regeln zu einem Satz zusammengesetzt werden. Dabei besteht jedes Wort aus einem Zeichenvorrat der Sprache (Alphabet). Im allgemeinen Sprachgebrauch kommt man mit diesen Begriffen gut zurecht.

In der theoretischen Informatik versucht man, den Sprachbegriff durch Abstraktion zu verallgemeiner. Auch hier besteht eine Sprache aus bestimmten erlaubten Symbolen, die *Terminalsymbole* genannt werden. Alle Terminalsymbole bilden das Alphabet der Sprache.

Ein *Wort* einer Sprache stellt eine beliebige Kombinationen von Terminalsymbolen dar. In diesem Sinne stellt auch ein vollständiger Satz ein Wort dar. Nicht alle Worte ergeben dabei einen Sinn. Mit Hilfe von Regeln werden nur bestimmte Wörter als gültig eingestuft. Diese Regeln bilden die *Grammatik* einer Sprache.

Beispiel: Die himmlische Sprache „Frohlocke”

*Als der Münchener Alois Huber in den Himmel kam überreicht ihm Petrus ein Harfe und wies ihn in die himmlische Sprache **Frohlocke** ein.*

Einige Wörter dieser Sprache sind

```
halleluja
halleleluja
hahahalleluuuja
```

Formal wird diese Sprache durch ein 4-er Tupel (T,N,S,E) und die Produktionsregeln beschrieben:

```
Terminalsymbole          T = { "h", "a", "l", "e", "u", "j" }
Nichtterminale Symbole   N = { HA, LE, LU, JA, FROHLOCKE }
Startsymbol              S = Frohlocke
Produktionsregeln:
                  FROHLOCKE = HA {HA} "l" LE {LE} LU JA
                  HA        = "h" "a"
                  LE        = "l" "e"
                  LU        = "l" "u" {"u"}
                  JA        = "j" "a"
```

Die Produktionsregeln basieren häufig auf Abkürzungen (nichtterminale Symbole) für bestimmte Eigenschaften. Diese sind zwar nicht zwingend nötig, machen aber die Regeln besser lesbar.

1.1 Definition „formale Sprache”

Mit dieser Definition wird lediglich die Sprachsyntax festgelegt, die Bedeutung (Semantik) der Worte spielt hierbei keine Rolle. Die formale Sprache L besitzt eine Grammatik G mit folgenden Eigenschaften:

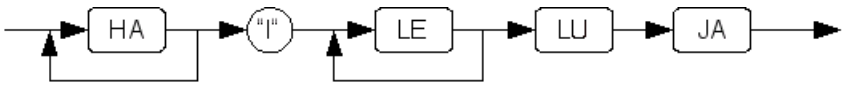
- Es gibt endlich viele Symbole, aus denen sich die Worte zusammensetzen.
  Terminalsymbole T
- Es gibt endlich viele Hilfssymbole, die die Struktur der Sprache beschreiben.
  Nichtterminalsymbole N
- Mit Hilfe von Regeln wird festgelegt, welche Worte erlaubt sind.
  Produktionen
- Jedes Wort beginnt und endet mit einem Element aus N oder T.
  Start-/Endsymbol der Sprache S,E

Kurz: **L(G) = (T,N,S,E,P)**

Die Sprachen werden entsprechend ihrer Produktionsregeln klassifiziert (Chomsky-Hierarchie). Einfache Sprachen können durch einen äquivalenten endlichen Automaten realisiert werden. Diese Sprachen heißen *regulär*.

1.2 Syntaxdiagramme

Die Regeln können anschaulich in Form von Syntaxdiagrammen angegeben werden. Dabei werden terminale Symbole in einem Kreis ausgegeben, nichtterminale Symbole in einem Rechteck. Jedes nichtterminale Symbol wird durch ein Sytaxdiagramm beschrieben.



Syntaxdiagramm für das Symbol "FROHLOCKE":

1.3 EBNF-Notation

Häufig werden die Produktionsregeln in der **Erweiterten Backhaus-Naur-Form (EBNF)** beschrieben. Das nichtterminale Symbol, das definiert wird, befindet sich auf der linken Seite. Auf der rechten Seite stehen terminale und nichtterminale Symbole mit folgender Bedeutung:

```
{ A }   Das Symbol A wird 0 bis n-mal wiederholt
[ A ]     Das Symbol A ist optional
A | B     Alternative: A oder B
(A B)     Gruppierung: A und B
Eine Regel wird durch einen Punkt beendet.
```

Beispiel:

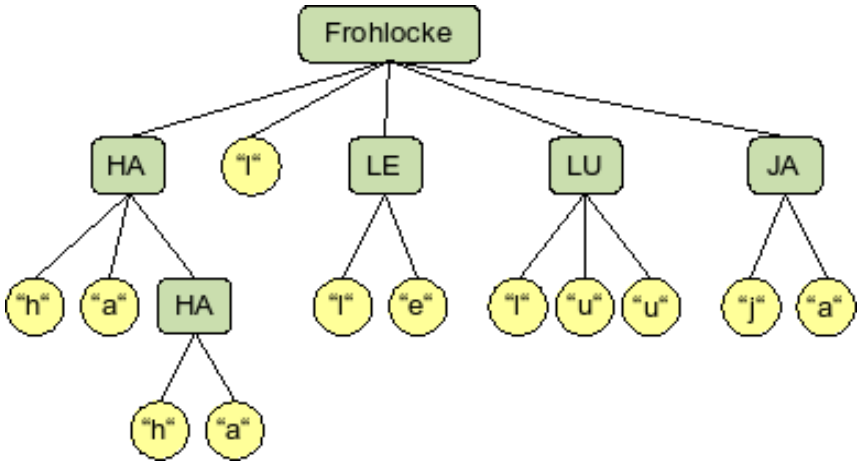
```
Preis = Betrag ['.' ('-'|Ziffer Ziffer)] ('€'|'$') .
```

1.4 Syntaxbäume

Will man überprüfen, ob eine Zeichenfolge ein gültiges Wort der Sprache „Frolocke" darstellt, so kann diese Überprüfung mit einem endlichen Automaten geschehen. Dieser Vorgang heißt „parsen" (engl.: to parse, analysieren).

Anwendung: Entsprechend prüft der Compiler einer Programmiersprache, ob ein eingegebenes Programm alle Regeln der Sprache einhält. Das ganze Programm wird darauf überprüft, ob es ein gültiges Wort der Sprache darstellt. Ein fehlerhaftes Wort führt zu einem Syntaxfehler. Nebenbei baut der Parser auch den zugehörenden Syntaxbaum auf.

Ein Syntaxbaum stellt eine baumförmige Ableitung eines Wortes der Sprache dar. Das Startsymbol bildet die Wurzel des Baumes. Die inneren Knoten entsprechen den nichtterminalen Symbolen, die Blätter stellen die terminalen Symbole dar. Das Wort „hahalleluuja" gehört zur Sprache *Frohlocke*, da es mit folgendem Syntaxbaum aus den Regeln ableitbar ist.



1.5 Beispiele

Emailadresse

*Gesucht ist eine Sprache, die alle Email-Adressen mit dem vereinfachten Format **xxx@yyy.zz** beschreibt. Jeder Name **xxx** besteht aus mindestens 2 Buchstaben, jeder Providername ist mindestens drei Buchstaben lang. Als Domäne kommt **de** oder **com** in Frage.*

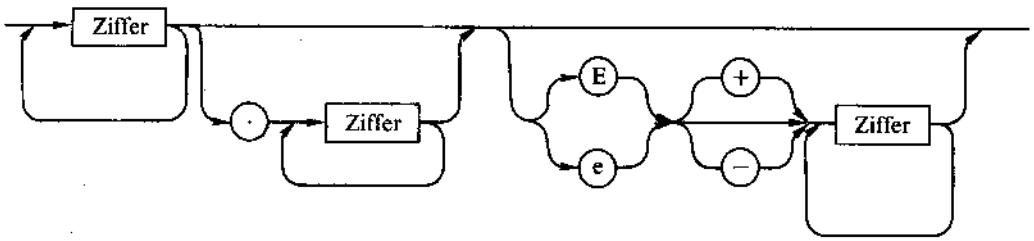
Eine formale Beschreibung der „E-Mail-Sprache" hat folgende Form:

```
T = {"a",..,"z","A",..,"Z","@","."}
N = {Email, Name, Provider, Zeichen}
S = Email

Produktionsregeln (EBNF):

  Email    = Name "@" Provider .
  Name     = Zeichen Zeichen {Zeichen} .
  Provider = Zeichen Name "." Land .
  Land     = "d" "e" | "c" "o" "m" .
  Zeichen  = "a"|"b"|..|"z"|"A"|..|"Z" .
```

Mit den Produktionsregeln können nun die Worte der Sprache auf ihre Gültigkeit überprüft werden. `hans@info.com` ist eine gültige Adresse. Dagegen ist die Adresse `emma.klug@ibm.de` fehlerhaft, da ein Name nach der 2. Regel nur aus Zeichen (d. h. Buchstaben) bestehen darf.

Darstellung von Dezimalzahlen

Gegeben ist das Syntaxdiagramm einer „Sprache" (mit der gewöhnlichen Bedeutung von Ziffer).



Die Regeln lassen sich mit EBNF sehr übersichtlich formulieren:

```
Dezimalzahl = Zahl [ "." Zahl ] [ Exponent] .
Zahl        = Ziffer { Ziffer } .
Exponent    = ("E"|"e") ["+"|"-"] Zahl .
Ziffer      = "0" | "1" | .. | "9" .
```

Diese Sprache erkennt positive Dezimalzahlen mit eventuellem Exponenten in der IEEE-Darstellung. Soll eine als String vorliegende Zahl auf ihre Gültigkeit hin überprüft werden, so kann auch hier der äquivalente endliche Automat zur Erkennung benutzt werden.

Das MU-Rätsel

D. Hofstadter beschreibt in seinem Werk „Gödel Escher Bach" eine formale Sprache, die nur drei Buchstaben kennt: **M**, **I** und **U**. Die Wörter, die die Sprache bilden, werden durch vier Regeln beschrieben. Als Startsymbol wird die Zeichenfolge „**MI**" vorgegeben. Eine Umkehrung der Regeln ist nicht erlaubt.

1. Hat ein Wort als letzten Buchstaben ein „I", dann darf ein „U" angehängt werden.
2. Ein Wort der Form „Mx" führt auf ein neues Wort „Mxx".
3. Enthält ein Wort „III", dann können diese Buchstaben auch durch ein „U" ersetzt werden.
4. Zwei aufeinanderfolgende „UU" dürfen auch gestrichen werden.

Beispiele:

```
Wenn "MUI" ein Wort der Sprache ist, dann auch "MUIU"   (Regel 1)
Mit "MUM" ist auch "MUMUM" ein gültiges Wort            (Regel 2)
Aus "MUIIIU" kann das Wort "MUU" abgeleitet werden      (Regel 3)
Das Wort "MUUUI" erlaubt auch das Wort "MUI"            (Regel 4)
```

Hofstadter fragt den Leser, ob das Wort „**MU**" in seiner Sprache vorkommen kann.

Dieses Beispiel erlaubt eine mehr formale Betrachtung von Sprachen. Das Startsymbol kann als Axiom und die vier Regeln können als Gesetzmäßigkeiten aufgefasst werden. Damit kann man ähnlich einer mathematischen Beweisführung ausgehend von Axiomen und bereits bewiesenen Tatsachen neue Schlussfolgerungen ziehen.

Wenn etwa geprüft werden soll, ob das Wort „MUI" erlaubt ist, dann muss dieses aus den bisher bekannten Worten und Regeln ableitbar sein.

```
 MI ---> MII ---> MIIII ---> MUI
     R2       R2         R3
```

Aus diesem nun bekannten Wort kann das Wort „MUIIU" hergeleitet werden:

```
MUI ---> MUIU ---> MUIUUIU ---> MUIIU
     R1        R2           R4
```

Hofstadter liefert einen allgemeinen Beweis, dass das Wort „MU” nicht zur Sprache gehört. Dabei verwendet er hauptsächlich den „I”-Gehalt (Anzahl der „I” in einem Wort). Das Urwort hat einen I-Gehalt von 1. Die Regeln 1 und 4 ändern den I-Gehalt nicht. Regel 3 vermindert den I-Gehalt um 3, Regel 2 verdoppelt den I-Gehalt eines Wortes. Diese beiden Regeln erzeugen niemals ein Vielfaches von 3, es sei denn, dass ein solches von Anfang an gegeben war. Das ist aber nicht der Fall (siehe [HOF] S. 280ff).

Hierzu gibt es auch einen interessanten Aufsatz bei `http://www.educ.ethz.ch/` unter >lehrpersonen >mathematik >unterrichtsmaterialien_mat >anwendungenmathematik >spiel.

Besondere Sprachen

Sprachen, die von Kellerautomaten erkannt werden

Die in der Mathematik erlaubten Terme können ebenfalls als „Term-Sprache” aufgefasst werden. Bei der Überprüfung des Terms

```
2+3*(4-(3+1))
```

auf seine Gültigkeit erkennt man aber, dass z.B. die Anzahl der öffnenden Klammern mitgezählt werden muss. Dies geht mit gewöhnlichen endlichen Automaten nicht mehr. Es handelt sich also hierbei um keine reguläre Sprache. Zu ihrer Realisierung (und Implementierung) benötigt man z.B. einen Zähler bzw. Stack. Die entsprechenden Automaten heißen Kellerautomaten.

Kontextsensitive Sprachen

In der fraktalen Geometrie werden häufig Lindenmayer-Systeme verwendet. Dabei wird das Wachstum durch fortgesetzte Ersetzung einzelner Symbole durch andere simuliert. Einfache Systeme stellen reguläre Sprachen dar. Die Regel R1 beschreibt den Aufbau der Kochschen Kurve, wenn F für das Zeichen einer Linie, ’+’ für eine $60°$- Drehung und ’-’ für eine Drehung um $-60°$ steht.

```
(R1)    F ->  F + F - - F + F
```

Komplexere Regeln erhält man, wenn die Ersetzung vom Umfeld abhängt (kontextsensitive Sprachen). Bei der Regel R2 hängt die Ersetzung von ’F’ durch das linke Zeichen ab:

```
(R2)   XF ->  +YF-XF-YF+
       YF ->  -XF+YF+XF-
```

1.6 Sprachhierarchie

Mit Hilfe endlicher Automaten können bereits alle regulären Sprachen realisiert werden. Ergänzt man das Konzept um einen Kellerstapel, so sind fast alle gängigen Computersysteme (Programmiersprachen) modellierbar.

Alan Turing beschäftigte sich bei der Entwickelte seiner Turingmaschine mit der Berechenbarkeit von Funktionen. *Alonso Church* fasste das Ergebnis seiner Arbeiten in folgender These zusammen:

> *Die Menge der Turing-berechenbaren Funktionen ist genau die Menge der im intuitiven Sinne berechenbaren Funktionen.*

Alle gängigen Programmiersprachen sind in diesem Sinne vollständig. Diese Aussage gilt streng genommen allerdings nur, wenn unbegrenzt Speicher zur Verfügung steht (entsprechend dem unendlich langen Turingband).

R. Baumann präzisierte den Begriff des Algorithmus mit Hilfe einer Turingmaschine[1]:

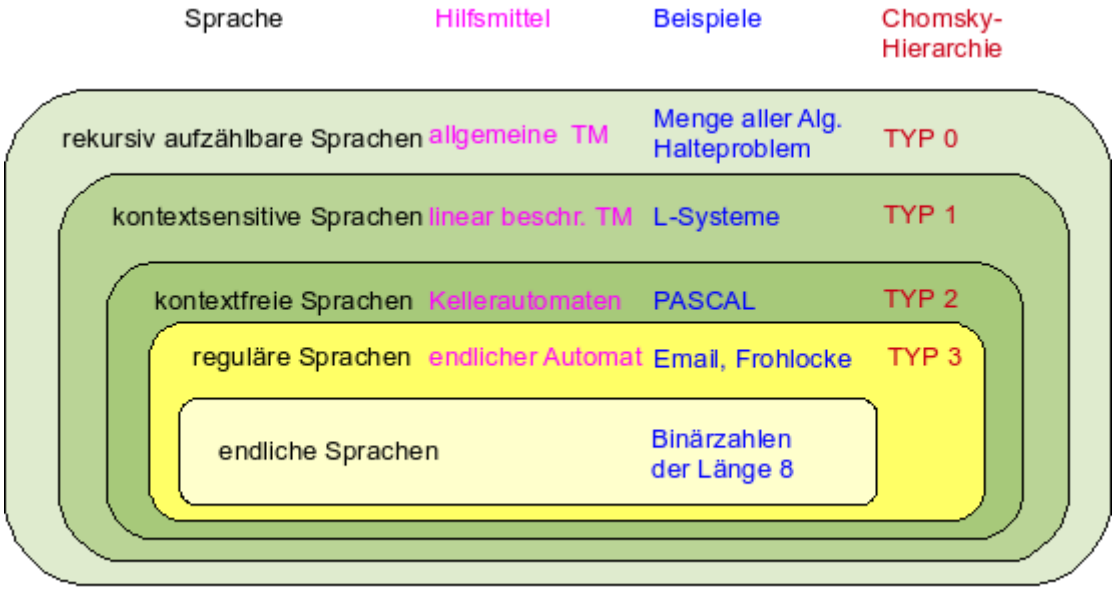
> *Ein Algorithmus ist ein Verfahren, das von einer Turingmaschine durchgeführt werden kann.*

Turingmaschinen wurden tatsächlich nie gebaut; sie stellen ein rein theoretisches Rechnermodell dar, mit dem theoretische Betrachtungen wie die Berechenbarkeit von Funktionen einfacher untersucht werden können.

Registermaschinen stellen ein Bindeglied zwischen Turingmaschinen und realem Rechner dar. Diese erlauben eine deutlich komfortablere Programmierung, da sie direkten Zugriff auf alle Register haben. Ihre Handhabung ist aber noch wesentlich mühseliger als die realer Rechner.

[1] Rüdiger Baumann: Informatik mit Pascal, Klett-Verlag, Stuttgart 1981

Der Informatiker Noam Chomsky hat formale Sprachen gemäß ihrer Komplexität klassifiziert. Bis auf Sprachen vom Typ 0 ist für alle Sprachen eindeutig entscheidbar, ob ein Wort w zu einer Sprache L gehört.



nach: L. Engelmann (Hrsg.): Informatik bis zum Abitur, Paetec-Verlag, Berlin 2002

1.7 Roboter Rob - ein Unterrichtsprojekt für den LK

Idee: Abituraufgaben für den Leistungskurs (Hessen)

Der Roboter "Rob" wird in einer Fabrik zur Bestückung von Platinen eingesetzt. Er kann von einer Home-Position $(0, 0)$ aus horizontal eine Position (x_1, y_1) anfahren, ein Bauteil aufnehmen ($Take$) und an einer anderen Position (x_2, y_2) einsetzen (Put).

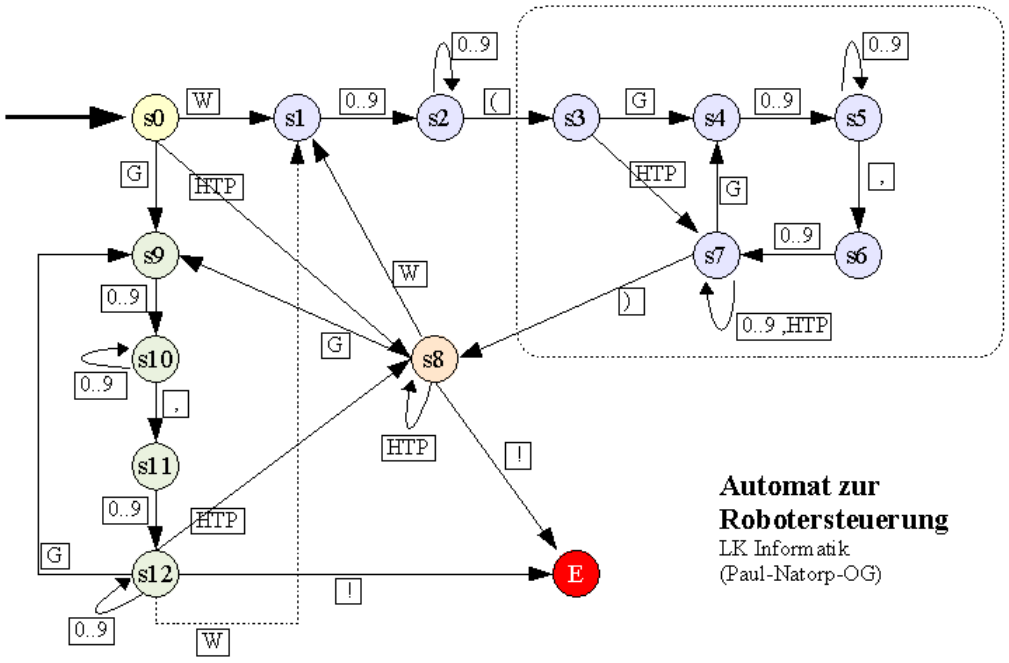
Zusätzlich kann er wiederkehrende Aufgaben durch eine Wiederholungsanweisung n-fach ausgeführen. Geschachtelte Wiederholungen sind nicht erlaubt.

Der Roboter „4you" kennt folgende Befehle:

`Code`	Befehl	Bedeutung
`H`	Home	Bringt den Roboter in die Nullstellung
`Gx,y`	Go (x,y)	Bewegt den Roboter an die Stelle (x,y)
`T`	Take	Nimmt an der aktuellen Stelle auf
`P`	Put	Legt an der aktuellen Stelle ab
`Wn(..)`	Wiederhole n mal	Wiederholt eine Anweisungsfolge n mal
`!`	Ende	Ende einer Steuersequenz

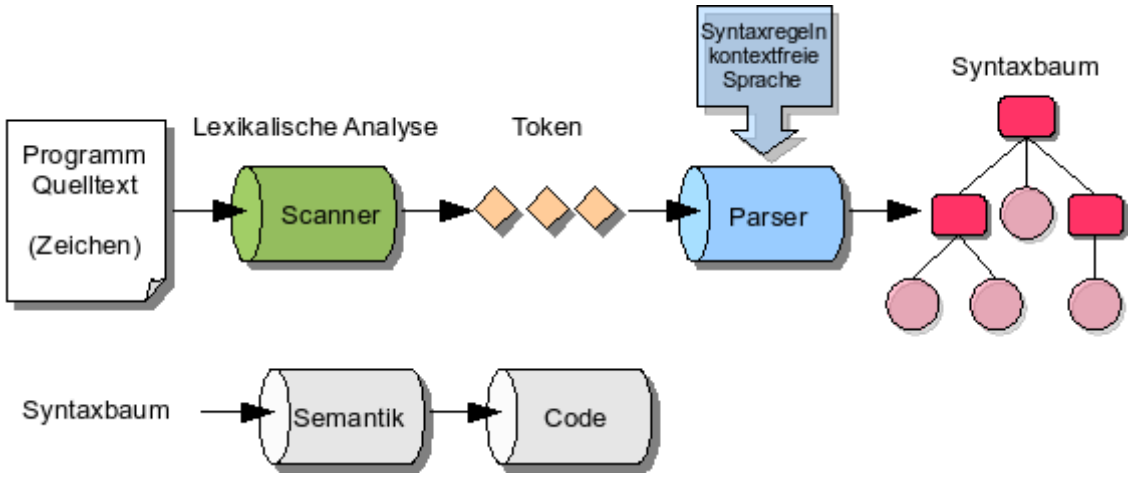
Ein mögliches „Programm" (Wort) der Sprache: "`HG100,50TG200,75P!`".

Da keine geschachtelten Wiederholungen erlaubt sind, handelt es sich um eine reguläre Sprache, die durch einen endlichen Automaten implementiert werden kann.



Dieses Projekt kann auch mit Java implementiert werden. Dabei können die unterscheidlichen Stufen eines Compilers / Interpreters schön veranschaulicht werden.

- Ein Compiler führt zunächst eine **lexikalische Analyse** durch. Dabei wird das Quellprogramm in eine Folge von Terminalsymbolen (Token) zerlegt. Für die Befehle entfällt dieser Schritt, da diese aus jeweils einem Zeichen bestehen. Auftretende Zahlen könnten aber als ein Token betrachtet werden.
- Dann wird in einer **syntaktischen Analyse** getestet,ob das Quellprogramm den Syntaxregeln der Sprache genügt. Dies kann mit Hilfe des endlichen Automaten als Akzeptor geschehen. Compiler bauen allerdings bei der Syntaxanalyse einen Syntaxbaum auf, der die weitere Bearbeitung (etwa Interpretation oder Auswertung) erheblich vereinfacht.
- Anschließend erfolgt eine **semantische Überprüfung** (etwa ob alle Namen definiert wurden). Auch dieser Schritt wird bei diesem Projekt nicht benötigt.
- Zum Schluss wird der **Code** generiert.



Anstelle der Codeerzeugung kann hier auch nach der Syntaxanalyse der Quellcode interpretiert werden. Wenn der Parser einen Syntaxbaum erzeugt, dann besteht die Interpretation in einem modifizierten Durchlaufen des Baumes in Inorder-Reihenfolge.

Ein Interpreter kann auch direkt auf den Syntaxregeln aufbauen und nach dem Prinzip der **absteigenden Rekursion** die Eingabe verarbeiten. Diese Methode soll hier kurz vorgestellt werden. Dabei wird folgende Grammatik in EBNF zugrunde gelegt:

```
Roboter      = Befehl { Befehl } '!' .
Befehl       = Befehl1 | Wiederholung .
Befehl1      = 'H' | 'T' | 'P' | Position .
Wiederholung = 'W' Zahl '(' Befehl1 { Befehl1 } ')' .
```

```
Position     = 'G' Zahl ',' Zahl .
Zahl         = Ziffer { Ziffer } .
Ziffer       = '0' | '1' | .. | '9' .
```

Die nichtterminalen Symbole sind

```
Roboter, Befehl, Befehl1, Wiederholung, Position, Zahl, Ziffer
```

Terminale Symbole sind

```
'!', 'H', 'T', 'P', 'W', 'G', '0', '1', .., '9'
```

Das Prinzip der absteigenden Rekursion beruht im Wesentlichen darauf, dass die einzelnen Regeln rekursiv implementiert werden und sich entsprechend der Syntaxregeln gegenseitig aufrufen. Dies führt in etwa zu folgender Hierarchie:

```
start
 |___ roboter
      |______ befehl
              |____ wiederholung (W) ---> befehl1
              |____ befehl1
                     |____ position (G)
                     |____ home (H)
                     |____ take (T)
                     |____ put (P)
```

Die aktuelle Leseposition sowie das nächste zu verarbeitende Zeichen (Token) im Eingabestring (Programm) werden häufig als globale Variablen gespeichert. Sinnvoll ist dazu eine Lesemethode, die zugleich überprüft, ob die Zeichen zu den erlaubten terminalen Symbolen gehören. Beim Lesen eines ungültigen Zeichens kann die Syntaxanalyse mit einer Fehlermeldung beendet werden.

Günstigerweise liefern alle Methoden einen Wahrheitswert zurück. Damit lassen sich die einzelnen Regeln sehr übersichtlich implementieren.

```
// Befehl ::= 'H' | 'T' | 'P' | Wiederholung | Position
private boolean befehl()   {
     switch (c)       {
          case 'T' : nextChar(); return true;
          case 'P' : nextChar(); return true;
          case 'H' : nextChar(); return true;
          case 'W' : nextChar(); return wiederholung();
          case 'G' : nextChar(); return position();
          default  : return false;
     }
 }
```

Mögliche Fragestellungen

1. Gesucht ist eine Grammatik für die Robotersprache L. Diese soll in der EBNF-Notation sowie als Syntaxdiagramm angegeben werden.

2. Was bewirkt die folgende Befehlsfolge? $G30, 10W2(TG60, 20PG50, 80)H$!
3. Mit welchen Steuerbefehlen kann der Roboter einen Gegenstand von einem Band (Position (20,40)) aufnehmen, an einem Arbeitsplatz (Position (200,60)) ablegen und wieder die Home-Position einnehmen?
4. Der Roboter soll 10 mal ein Quadrat der Seitenlänge 50 entlangfahren und in jedem Eckpunkt etwas ablegen. Die linke, obere Ecke des Quadrats hat die Koordinaten (30,70). Am Ende soll der Roboter in die Home-Position gefahren werden.
5. Die Sprache L, die der Roboter akzeptiert, kann durch einen endlichen Automaten simuliert werden. Entwerfe den Zustandsgraphen des Automaten.
6. Die Sprache „Rob" soll durch relative Bewegungen erweitert werden. Gib eine mögliche Syntaxerweiterung an. Dabei müssen auch negative Zahlenangaben berücksichtigt werden.

1.8 Übungsaufgaben

Übung 1 Eine Sprache enthält folgende Regeln:

```
Definition = Var ':' '=' Ausdruck .
Ausdruck   = Var | Var '+' Ausdruck .
Var        = 'a' | 'b' | 'c' .
```

- Konstruieren Sie ein Wort, das mindestens 6 Zeichen lang ist.
- Zeichnen Sie den zugehörigen Syntaxbaum.
- Entwerfen Sie mit JFLAP einen passenden Akzeptor.

Übung 2 Für die Sprache der römischen Ziffern soll ein Akzeptor entwickelt werden, der Zahlen zwischen 1 und 28 erkennt. Beschreiben Sie die Regeln für diese Zahlen in der BNF-Notation.

Weitere Aufgaben

Aufgabe 1 Gesucht sind alle erlaubten Wörter (Sätze) zu folgender Syntaxregeln (Startsymbol ist „*Teddybär*"):

```
Teddybär     = "Fell" Körperteile [Kleidung] .
Körperteile  = Kopf "Rumpf" Gliedmaße .
Kopf         = "Augen" ["Nase" ["Barthaare"]] "Ohren" .
Gliedmaße    = "Arme" "Beine" .
Kleidung     = "Hut" | "Hose" | "Jacke" .
```

Aufgabe 2 Einfache deutsche Sätze können mit der folgenden Grammatik erzeugt werden:

```
Satz          = Nominalgruppe Verbalgruppe .
Nominalgruppe = Artikel Substantiv .
Verbalgruppe  = Verb | Verb Nominalgruppe .
Verb          = "jagt" | "beißt" | "sieht" .
Substantiv    = "Hund" | "Katze" | "Frau" .
Artikel       = "der" | "die" | "das" .
```

- Bilden Sie zwei mögliche Sätze.
- Leiten Sie den folgenden Satz nach den Regeln ab und skizzieren Sie den Syntaxbaum:
  `Der Hund jagt die Katze.`

- Erweitern Sie die Grammatik so, dass Substantive mit einem (optionalen) Attribut versehen werden können.
  (Attribute: `schwarze`, `junge`, `gefährliche`)
- Beschreiben Sie formal die Sprache.

Aufgabe 3 Gesucht ist eine formale Sprache mit Grammatik zur Erzeugung von Zahlen, die durch 5 oder 2 teilbar sind. Geben Sie auch das zugehörige Syntaxdiagramm sowie einen äquivalenten Automaten an.

Aufgabe 4 Die folgende EBNF erlaubt die (freie) Zusammenstellung der kommenden Informatiklausur. Stellen Sie sich drei Aufgaben zusammen. (STRING ist vordefiniert)

```
Klausur  = Aufgabe [Aufgabe] .
Aufgabe  = Nummer Punkte Thema Inhalt .
Thema    = "Formale Sprachen" | "Automaten" | "Grammatiken".
Inhalt   = STRING .
Nummer   = Ziffer .
Punkte   = Zahl .
Zahl     = Ziffer {Ziffer} .
Ziffer   = "0" | "1" | .. | "9" .
```

Aufgabe 5 Gesucht ist die EBNF-Syntaxdefinition für

1. für eine ungerade ganze Zahl.
2. für einen Währungsbetrag im Format `*.xx Euro`.
3. für eine Zeitausgabe im Format `std:min:sek` ohne führende Nullen
   Beispiel: `"12:8:9"`

2 Literatur und Links

Literatur

[MOD] Modrow, E.: Theoretische Informatik mit Delphi. emu-online Scheden 2005.(`www.emu-online.de`)

Ein neues Schulbuch mit einer vollständigen Darstellung schulrelevanter Themen (Automaten, Turingmaschinen, Sprachen). Als Programmiersprache wird Delphi verwendet.

[RNH] Reichert, R. Nievergelt, J. Hartmann, W.: Programmieren mit Java. Springer Berlin, Heidelberg, New York 2005

Das Buch beschreibt die unterschiedlichen Einsatzmöglichkten des Kara-Modells im Schulunterricht. Dabei werden die unterschiedlichen Kara-Modelle exemplarisch vorgestellt.

[ENG] Engelmann, L.: Informatik bis zum Abitur. Paetec, Berlin, 2002 (ISBN 3-89818-600-8)

Das Unterrichtsbuch enthält eine ausführliche Darstellung der theoretischen Informatik mit einigen Beispielaufgaben. Neben formalen Sprachen und Automaten werden die Berechenbarkeitstheorie behandelt und effiziente Algorithmen mit Komplexitätsbetrachtungen vorgestellt.

[HOF] Douglas R. Hofstadter: Gödel, Escher, Bach ein endloses geflochtenes Band. Ernst Klett Verlag, Stuttgart 1985 (ISBN 3423300175)

Ein absolutes Kultbuch, das in zentrale Aspekte der theoretischen Informatik einführt. Zu jedem Kapitel gibt es eine einführende Kurzgeschichte mit den Hauptfiguren Achilles und Theo Schildkröte.

[Leh] Lehmann, E.: Die Turing-Maschine im Anfangsunterricht. LOGIN 1999, Heft 6, S.44-52

Ein Bericht von den ersten Stunden eines Informatikkurses in Klasse 11

Eine umfassendere Literaturliste finden Sie auf den Seiten des Berliner Bildungsservers.

Internet

`http://www.jflap.org/`

Homepage des vorgestellten Javaprogramms zur Modellierung endlicher Automaten. Mit diesem Programm können auch Kellerautomaten, Sprachen, Turing- und Registermaschinen veranschaulicht werden.

`http://www.swisseduc.ch/informatik/exorciser`

Dieses Javaprogramm beeinhaltet eine Aufgabensammlung zu den einzelnen Themen. Auf Wunsch werden auch die Lösungen angezeigt.

`http://www.swisseduc.ch/informatik/karatojava/`

Mit dem Kara-Modell können Automaten direkt simuliert werden. Auch Turingmaschinen können damit veranschaulicht werden. Zusätzlich enthalten diese Seiten viele Unterrichtsmaterialien zum Thema.

`http://oszhdl.be.schule.de/gymnasium/faecher/informatik/automaten`

Es werden endliche Automaten an Beispielen vorgestellt und die Begriffe erläutert. Das Automatenmodell der ETH-Zürich (KaraToJava) wird u.a. an Beispiel vorgestellt.

http://www.tinohempel.de/info/info/ti/informatik.htm

Tino Hempel beschreibt seine Unterrichtserfahrungen im Grund- und Leistungskurs zum Thema. Die Seiten enthalten gut dokumentierte Beispiele, weitere Anregungen und Hinweise. Beispiele sind teilweise in Prolog implementiert.

http://www.oberstufeninformatik.de/theorie

Startseite von H. Gierhardt zur theoretischen Informatik. Zahlreiche Verweise zu weiterführenden Seiten und Software.

http://www.pns-berlin.de

Erkennende Automaten und formale Sprachen werden hier unter dem Aspekt einer Implementierung in Haskell und/oder Java dargestellt. Die Informationen beruhen auf Unterrichtserfahrungen aus unterschiedlichen Informatikkursen (Basiskurs, Profilkurs, GK, LK) an der Paul-Natorp-Oberschule.

Hier sind auch die Materialien zu diesem Workshop erhältlich.

http://de.wikipedia.org/wiki/Turingmaschine

Allgemeine Informationen zu Turingmaschinen.

http://de.wikipedia.org/wiki/Fleißiger_Biber

Das Busy-Beaver-Problem beschreibt eine Turingmaschinen, die zu einer vorgegebenen Zahl von Zuständen möglichst viele Einsen auf ein leeres Band schreibt und dabei zum Stehen kommt. (siehe auch `http://www.dbg.rt.bw.schule.de/lehrer/ritters/info/turing/biber.htm` und `http://www.fmi.uni-stuttgart.de/ti/projects/beaver/bbb.html`)

Software

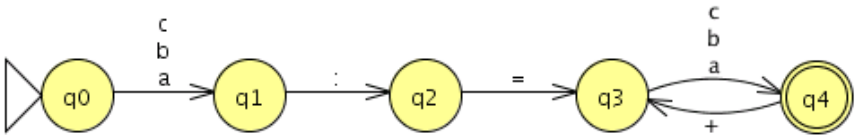
- `www.swisseduc.ch/informatik/karatojava/`
- `www.jflap.org`
- `http://math.hws.edu/TMCM/java/labs/xTuringMachineLab.html`
- `www.matheprisma.uni-wuppertal.de/Module/Turing/`
- `www.dbg.rt.bw.schule.de/lehrer/ritters/info/turing/vtur.htm`
- `wwwsys.informatik.fh-wiesbaden.de/weber1/turing/tm.html`

3 Lösungen zu den Aufgaben

Sprachen – Übungsaufgaben

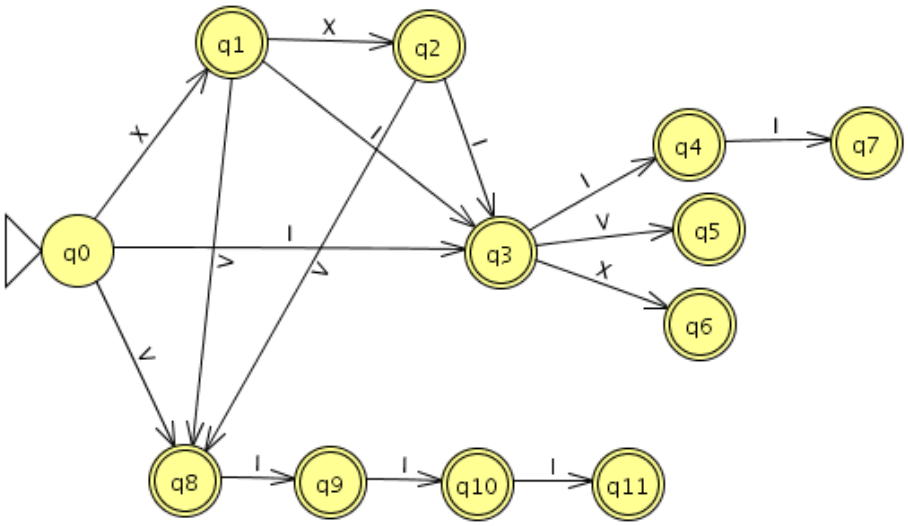
Ü1: Wort: ”a : = b + c”

```
        D e f
      / |   | \
   Var ':' '=' Ausdruck
    |             /  |  \
  'a'          'b' '+' 'c'
```



Ü2: BNF:

```
Zahl   = ['X' ['X']] Z1bis8 | ['I'] 'X' ['X'] .
Z1BIS8 = ['V'] 'I' ['I'['I']] | ['I'] 'V' .
```



Weitere Aufgaben:

Aufgabe 1

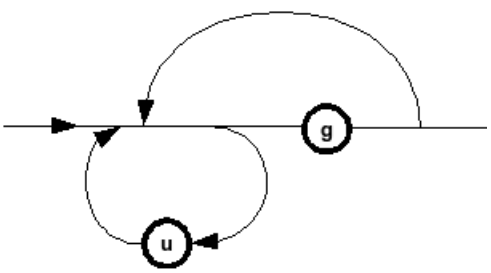
```
Fell Augen Ohren Rumpf Arme Beine Hut
Fell Augen Nase Ohren Rumpf Arme Beine Hut
Fell Augen Nase Barthaare Ohren Rumpf Arme Beine Hut

Fell Augen Ohren Rumpf Arme Beine Hose
Fell Augen Nase Ohren Rumpf Arme Beine Hose
Fell Augen Nase Barthaare Ohren Rumpf Arme Beine Hose
```

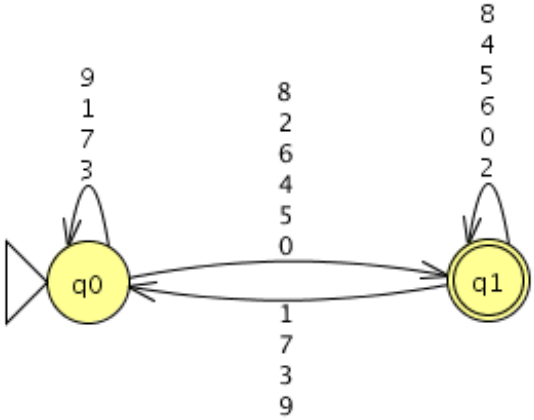
```
Fell Augen Ohren Rumpf Arme Beine Jacke
Fell Augen Nase Ohren Rumpf Arme Beine Jacke
Fell Augen Nase Barthaare Ohren Rumpf Arme Beine Jacke

Fell Augen Ohren Rumpf Arme Beine
Fell Augen Nase Ohren Rumpf Arme Beine
Fell Augen Nase Barthaare Ohren Rumpf Arme Beine
```

Aufgabe 2

```
Die Katze sieht die Katze
Die Frau beißt das Hund

Der Hund jagt die Katze:
              S a t z
            /         \
          NG           VG
        /    \        /   \
      Art    Sub    Verb   NG
       |      |      |    /   \
    "der"  "Hund" "jagt" Art  Sub
                          |    |
                       "die" "Katze"

Nominalgruppe = Artikel [Adjektiv] Substantiv .

T = { "jagt", "beißt", "sieht", "Hund", "Katze","Frau",
      "der", "die", "das" }
N = { Satz, Nominalgruppe, Verbalgruppe, Verb,
      Substantiv, Artikel }
Startsymbol = Satz
```

Aufgabe 3 Die Zahl wird Ziffernweise von links nach rechts gelesen.





Aufgabe 4

```
1 25 "Automaten" "Der Getränkeautomat ..."
2 50 "Grammatiken" "Die Sprache der Terme"
3 25 "Automaten" "Eine unglaubliche Robotersteuerung"
```

Aufgabe 5

```
(1) Z1       = [Zahl] Ungerade .
    Zahl     =  Ziffer {Ziffer} .
    Ungerade = '1' | '3' | '5' | '7' | '9' .
    Ziffer   = Ungerade | '0' | '2' | '4' | '6' | '8' .

(2) Geld     = Zahl '.' Cent "Euro" .
    Zahl     = Ziffer {Ziffer} .
    Cent     = Ziffer Ziffer .

(3) Zeit     = Std ':' Min ':' Sek .
    Std      = ('0'|'1') Ziffer | '2' ('0'|'1'|'2'|'3') .
    Min      = ('0'|'1'|'2'|'3'|'4'|'5') Ziffer .
    Sek      = Min .
    Ziffer   = '0'|'1'|'2'| .. | '9' .
```

Für Rückfragen, Anregungen und Kritik können Sie sich gerne an mich wenden. Die Materialien dieses Workshops und früherer Veranstaltungen finden Sie unter der angegebenen Internetadresse.

Walter Gussmann Paul-Natorp-Oberschule,
Internet: http://www.pns-berlin.de
Email: wagu1@web.de