Formale Sprachen

by

Dr. Günter Kolousek

Sprachen

- zur Übertragung und Weitergabe von Information
- Unterteilung
  - Natürliche Sprachen
    - historisch gewachsen (langer Entwicklungsprozess!).
    - dienen der Verständigung zwischen Menschen.
    - werden gesprochen oder schriftlich verwendet.
    - meist redundant (Redundanz): Buchstaben oder ganze Worte können in einem Satz fehlen und die Bedeutung (Information!) bleibt u.U. trotzdem erhalten.
    - Deutsch, Chinesisch,…,Körpersprache
  - Künstliche Sprachen

Natürliche Sprachen

- nicht exakt
  - ‘Er ist ein großer Schwindler’ (wie groß?)
  - ‘Wir haben nur noch wenig Zeit’ (wie viel?)
- mehrdeutig
  - ‘Er sah den Mann auf dem Hügel mit dem Fernrohr’ (wer hat das Fernrohr?)
  - ‘Ich gehe zur Bank’ (um mich zu setzen oder Geld abzuheben?)
- verändern sich laufend.
  - Neue Begriffe entstehen
    - ”fingerfood”
  - Bedeutungen ändern sich
    - ”nerd”: Sonderling vs. Computerfreak
  - Begriffe fallen weg
    - ”Kanapee”

Natürliche Sprachen – 2

The programmer's wife sent him to the grocery store.
Her instructions were: "buy butter. See if they have eggs. If they do, buy 10."
He came back with 10 packs of butter. Because they had eggs.

Künstliche (formale) Sprachen

- für bestimmte Zwecke Information in kompakter Form und unmissverständlich vermitteln.
  - werden meist für ein Spezialgebiet ‘erfunden’.
  - werden meist in geschriebener Form verwendet.
  - sind exakt.
  - sind eindeutig.
  - sind unveränderlich.
- Aufbau: streng vorgegebene, tw. formalisierte Regeln
- Beispiele
  - Notenschrift
  - Mathematik
  - Chemie
  - Informatik, Programmiersprachen

Sprachen in der Informatik

- Prinzipiell 2 Arten von Daten
  - Binärformate: nur maschinenlesbar
  - Klartextformat: Inhalte werden rein textuell verfasst. $\rightarrow$ Texteditor! $\rightarrow$ Kodierung!
- Sprachen im Klartextformat
  - Programmiersprachen
  - Auszeichnungssprachen: XML, HTML, CSS, RTF, Postscript, LaTeX, Markdown, Org Mode,…
    - grundsätzlich plattformneutral
    - lesbar durch Mensch und Computer
    - veränderbar durch variable Länge
    - Speicherbedarf und Zeit zum Verarbeiten in der Regel höher als bei Binärformaten
  - Formale Sprachen

Aufbau einer Sprache

- Alphabet: Menge aller Zeichen der Worte.
- Grammatik: Regeln wie Zeichen bzw. Worte zu kombinieren sind, um ein gültiges Wort zu erhalten.
- Syntax = Grammatik und Alphabet. Legt Form (d.h. richtigen Aufbau) der Zeichenketten fest.
- Semantik: Die Bedeutung syntaktisch richtiger Zeichenketten.

```
a := b; (Pascal, Modula)
a = b; (Java, C, C++, C#, JavaScript,...)
MOVE B TO A (Cobol)
```

- Pragmatik: Teil der Bedeutung, der vom Informationsempfänger gewisse Vorkenntnisse einbezieht. Bewirkt persönliche Interpretation (Anspielungen, Wortspiele, Stimmungen,… Informatik: eleganter Algorithmus).

Alphabet – Syntax – Semantik

- Wort über dem Alphabet: Aneinanderreihung von Zeichen
- Syntax legt Worte der Sprache fest
- Semantik scheidet bedeutungslose Worte aus
- Deutsche Sprache
  - Alphabet: Groß- und Kleinbuchstaben, Ziffern,…
  - Worte über dem Alphabet: asdf*123
  - Teilworte der Sprache: sind im Duden zu finden
  - syntaktisch falsch: ”Der wenn seine morgen Auto.”
  - syntaktisch richtig, semantisch falsch: ”Der Tisch spricht gelb über Informatik.”
  - syntaktisch richtig, semantisch richtig: ”Die Ferien sind leider vorbei!”

Einteilung der Sprachen

- auf Grund ihrer Mächtigkeit in verschiedene Sprachklassen
- Chomsky-Hierarchie:
  - Typ-0 Sprachen: unbeschränkte Sprachen, d.h. alle Sprachen, die durch eine beliebige Grammatik erzeugt werden können.
  - Typ-1 Sprachen: kontextsensitive Sprachen
  - Typ-2 Sprachen: kontextfreie Sprachen
  - Typ-3 Sprachen: reguläre Sprachen
- Typ-3 $\subseteq$ Typ-2 $\subseteq$ Typ-1 $\subseteq$ Typ-0

Grammatik – Definition

- Objektsprache: Sprache, deren Syntax beschrieben werden soll
- Metasprache: Sprache zur Darstellung einer Grammatik
  - mit Metasprache wird Objektsprache beschrieben
- Grammatik G = (Φ, Σ, P, S)
  - Φ … Menge der Hilfssymbole (Non-Terminalsymbole)
  - Σ … Menge der Terminalsymbole $\rightarrow$ Alphabet
  - P … Menge der Produktions- oder Ersetzungsregeln
  - S … Startsymbol (aus der Menge der Hilfssymbole)
- Es muss gelten: $\Phi \cap \Sigma = \{\}$
- Σ^+: Menge aller Worte über Σ
- $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

Grammatik – Beispiele

- Grammatik

$$
\begin{aligned}
G &= (\Phi, \Sigma, P, S) \\
\Phi &= \{S, L, E\} \\
\Sigma &= \{a, b, c, ;, (,)\} \\
P &= \{S \to L, E \to a, E \to b, L \to L, L \to (L; E), L \to cc\} \\
S &= S
\end{aligned}
$$

- Menge konstruierbarer Worte

$$
\begin{aligned}
\Sigma &= \{a, b\} \\
\Sigma^+ &= \{a, b, aa, ab, ba, bb, aaa, aab, \ldots\} \\
\Sigma^* &= \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}
\end{aligned}
$$

Wortlänge und Verkettung

- Wort der Länge n über Σ ist eine Folge von n Terminalsymbole
  - $x = x_1x_2x_3\ldots x_n$ mit $x_i \in \Sigma$ und $1 \leq i \leq n$: $|x| = n$.
  - $|\epsilon| = 0$.
  - Beispiel: $\Sigma = \{a, b, c\}$

$$x = aabcab \qquad y = ccc \qquad xy = aabcabccc$$
$$|x| = 6 \qquad |y| = 3 \qquad |xy| = 9$$

- Verkettung: $x, y \in \Sigma^+, x = x_1x_2x_3\ldots x_m, y = y_1y_2y_3\ldots y_n$ dann ist die **Verkettung** $xy = x_1x_2x_3\ldots x_my_1y_2y_3\ldots y_n$
  - Kurzschreibweise:

$$aa \quad \ldots \quad a^2$$
$$abbbaab \quad \ldots \quad ab^3a^2b$$

Ersetzungen und Ableitungen

- Zeichen $\rightarrow$: mögliche Ersetzung
  - Abkürzung für $E \rightarrow a, E \rightarrow b$: $E \rightarrow a|b$
- Zeichen $\Rightarrow$: tatsächliche Ersetzung oder Ableitung
- Es gilt: $uxw \Rightarrow uyw$ genau dann, wenn $x \rightarrow y \in P$
- $\Rightarrow^*$: Ableitung in beliebig vielen Schritten
- Beispiel: Grammatik G wie vorher!

$$P = \{S \rightarrow L, E \rightarrow a, E \rightarrow b, L \rightarrow L, L \rightarrow (L;E), L \rightarrow cc\}$$
$$S \Rightarrow L \Rightarrow (L;E) \Rightarrow (cc;E) \Rightarrow (cc;a)$$
$$S \Rightarrow^* (cc;a)$$

- $(cc;a)$ ist ein Terminalwort

Sprache

- L(G) … die durch die Grammatik G erzeugte Sprache.
- L(G) besteht aus genau allen Terminalworten, die sich aus der Startvariable ableiten lassen:

$$L(G) = \{w | w \in \Sigma^*, S \Rightarrow^* w\}$$

  w … Terminalworte: bestehen nur aus Terminalsymbolen
  S … Startsymbol
- L(G) $\subseteq \Sigma^*$

Sprachen und Grammatiken

Typ	Sprache	erzeugt durch
Typ-0	unbeschränkte Sprachen	beliebige Grammatik
Typ-1	kontextsensitive Sprachen	kontextsensitive Grammatik
Typ-2	kontextfreie Sprachen	kontextfreie Grammatik
Typ-3	reguläre Sprachen	reguläre Grammatik

Kontextsensitive Grammatik

- Kontext (Umgebung): Zeichen li und re eines NT-Symboles
- Definition:
  - li und re Seite einer Regel: Terminal- wie NT Symbole
  - Regeln
    - Die li Regelseite darf nicht länger als die re Regelseite sein.
    - Die Regel $S \rightarrow \epsilon$ ist zugelassen, aber wenn sie vorkommt, darf S auf keiner rechten Seite stehen.
    - Links muss mind. ein Non-Terminalsymbol stehen.
- Andere Definition:
  - Die Regeln haben folgende Gestalt: $\alpha N \beta \rightarrow \alpha \gamma \beta$, wobei $\alpha, \beta \in (\Phi \cup \Sigma)^*$ und $\gamma \in (\Phi \cup \Sigma)^+$ sein muss.
  - Die Regel $S \rightarrow \epsilon$ ist zugelassen, aber wenn sie vorkommt, darf S auf keiner rechten Seite stehen.
- Beide Definitionen führen zur selben Sprachklasse!

KS Grammatik – Beispiel

- Ges.: Ableitung für $a^4bd^2c^4$

$$G = (\Phi, \Sigma, P, S)$$
$$\Phi = \{S, B, X\}$$
$$\Sigma = \{a, b, c, d\}$$
$$P = \{S \to aBc, aB \to aaBc, Bc \to dXdc, dX \to Xd, aX \to ab\}$$

- Ges.: Ableitung für $a^3b^3c^3$

$$L(G) = \{a^n b^n c^n | n \geq 1\}$$
$$G = (\Phi, \Sigma, P, S)$$
$$\Phi = \{S, B, X\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to aBc, aB \to aaXBB, XB \to BX, Xc \to cc, B \to b\}$$

Kontextfreie Grammatik

- Definition
  - Li Seite: genau ein NT-Symbol
  - Re Seite: beliebige Symbolfolge.
  - D.h. Hilfssymbol wird unabhängig vom Kontext ersetzt
- Beispiele für KF Produktionsregeln

$$A \rightarrow aAb$$

$$S \rightarrow XYZ$$

$$B \rightarrow abcd$$

$$A \rightarrow \epsilon$$

- KF … wichtige Klasse $\rightarrow$ Syntax von Programmiersprachen!
  - Backus-Naur Form
  - Syntaxdiagramme

KF Grammatik – Beispiel

Die Menge aller arithmetischen Ausdrücke über den Variablen $x_1, x_2, \ldots x_n$ und den Operationssymbolen $+, *, -, /$ mit korrekter Klammerung ist eine kontextfreie Sprache. Die dazugehörige Grammatik sieht folgendermaßen aus:

$$G = (\Phi, \Sigma, P, S)$$
$$\Phi = \{S, E\}$$
$$\Sigma = \{+, *, -, /, (,), x_1, x_2, \ldots, x_n\}$$
$$P = \{$$

Backus-Naur Form

- 1959: John Backus & Peter Naur $\rightarrow$ Algol 60
- Metazeichen

`<>`	NT-Symbole, z.B. `<letter>`
`::=`	Definitionszeichen; trennt li von re Regelteil
`|`	trennt verschiedene Regelalternativen
Leerzeichen	Trennzeichen bei Sequenz

- Vorrangregel: Sequenz vor Alternative
- Vorteile
  - übersichtliche Darstellung
  - bequem zu schreiben
  - große Freiheit in der Bezeichnung der Objekte

Backus-Naur Form – 2

- Beispiel: Identifier

$$\Sigma = \{0, \ldots, 9, a, \ldots, z, A, \ldots, Z\} \quad (1)$$

$$\Phi = \{\langle \mathit{letter} \rangle, \langle \mathit{identifier} \rangle, \langle \mathit{digit} \rangle\} \quad (2)$$

$$P = \{ \quad (3)$$

$$S = \quad (4)$$

- Erweiterungen: EBNF, ABNF
  - jeweils *unterschiedliche* Definitionen!

Erweiterte BNF (EBNF)

- $\rightarrow$ Pascal: Metazeichen { und }
- Metazeichen

  { X } X kann beliebig oft, d.h. 0,1, oder n-Mal.
- Beispiel: Identifier

$$\Sigma = \{0, \ldots, 9, a, \ldots, z, A, \ldots, Z\}$$
$$\Phi = \{\langle letter \rangle, \langle identifier \rangle, \langle digit \rangle, \langle letterordigit \rangle\}$$
$$P = \{$$

$$S =$$

Ges.: außerdem Ableitung für `Ab3c`.

EBNF – ISO Variante

- Terminalsymbolen in ” ” oder in ’ ’
  - z.B.: ”1” oder ’1’
- Non-Terminalsymbole ohne Maskierung
  - z.B.: `digit`
- = anstatt `::=`
  - z.B.: `digit = ”1” | ”2”...`
- Sequenz von Symbolen durch `,` getrennt
  - z.B.: `digit, digit, digit, digit`
- Bestimmte Anzahl der Wiederholung durch `*`
  - z.B.: `4 * digit`
- Endezeichen einer Produktionsregel `;`
  - z.B.: `year = 4 * digit;`

EBNF – ISO Variante – 2

- Beliebige Wiederholung durch { und }
  - z.B.: `{ digit }`
- Mindestens einmalige Wiederholung mittels { }-
  - z.B.: `{ digit }-`
- Optional durch [und]
  - z.B.: `[ sign ] { digit }-`
- Gruppierung mittels ()
  - z.B.: `char (digit | char)`
- Ausnahme mittels - (infix)
  - z.B.: `comment_character = character - ";";`
- Kommentar mittels (* *)
  - z.B.: `(* Kommentar *)`

EBNF – ISO Variante – 3

- Vorrangregeln
  1. Wiederholung, d.h. *
  2. Ausnahme, d.h. -
  3. Sequenz, d.h. ,
  4. Alternative, d.h. |
- Vorrangregeln bzgl. Klammern
  1. ' $\rightarrow$ Terminalzeichen
  2. " $\rightarrow$ Terminalzeichen
  3. Kommentare, d.h. (* und *)
  4. Gruppierung, d.h. (und)
  5. optionaler Term, d.h. [und]
  6. Wiederholung, d.h. { und }

ABNF

- Angereicherte BNF (engl. augmented backus-naur form)
- Verwendung: Spezifikation in RFCs der IETF
- Ähnlich EBNF
- Terminalsymbole (nur) in ”
  - aber case insensitive, außer, wenn `%s”pRoGramm”`
- Alternative: `/`, z.B.: `bit = ”0” / ”1”`
  - inkrementell: `/=`, z.B.: `fruit /= apple`
- Zeichencodes
  - wie z.B. CR: `%d13` oder `%x0d` oder `%b00001101`
  - Bereiche: `%x30-39` ≡ `”0” / ”1” /...`
- Non-Terminalsymbole, nur A-Z,a-z,0-9 sowie `-`, aber *muss* mit mit Buchstaben beginnen; case-insensitive!
- Sequenz: durch Leerzeichen getrennt
- Produktionsregel: li durch re Seite getrennt mittels =

ABNF – 2

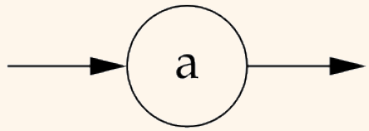
- Gruppierung mittels `()`, z.B.: `char (digit | char)`
- Wiederholung mittels `*`, z.B.: `*digit` (beliebig), `4digit` (genau), `2*digit` (min), `*8digit` (max), `2*8digit` (Bereich)
- Zeilenkommentar mittels `;` (wie in Python/Shell/PHP #)
- Vorrangregeln
  1. Kommentare
  2. Zeichenketten ($\rightarrow$ Terminalsymbole) und Non-Terminalsymbole
  3. Bereiche
  4. Wiederholung
  5. Gruppierung
  6. Sequenz
  7. Alternative
- Beispiel:
```
number = *1'-' digit-without-zero *digit / "0" ;
```

Syntaxdiagramm

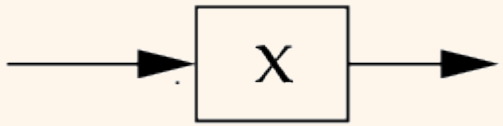
- graphische Beschreibungsmethode für KF Grammatiken
- durch Pfeile verbundene Menge abgerundeter und rechteckiger Felder
- genau ein Eingang und genau ein Ausgang
- Diagramm *muss* einen Namen haben
- Beispiel: Menge von Wörtern, die mit einer Ziffer beginnen und enden.

Syntaxdiagramm – 2

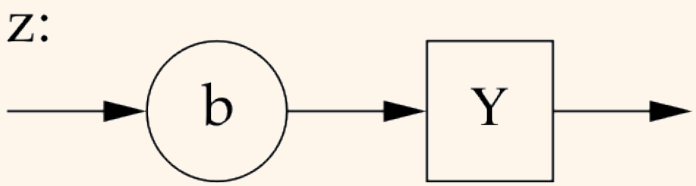
- Terminalsymbol: a



- NT-Symbol: <x>

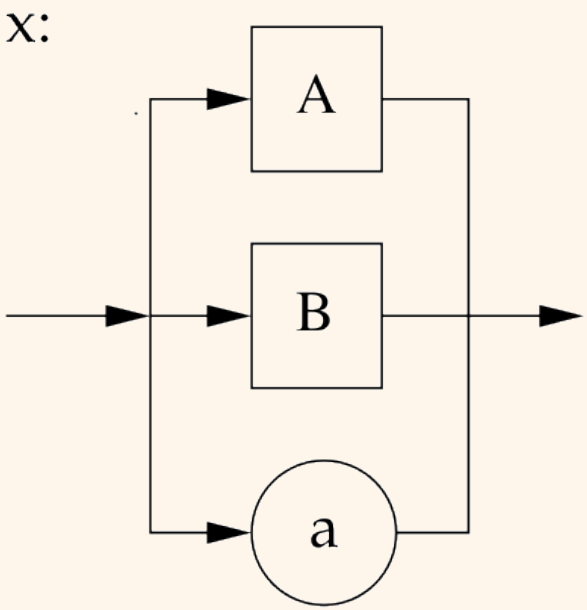


- Sequenz: <z> ::= b <Y>



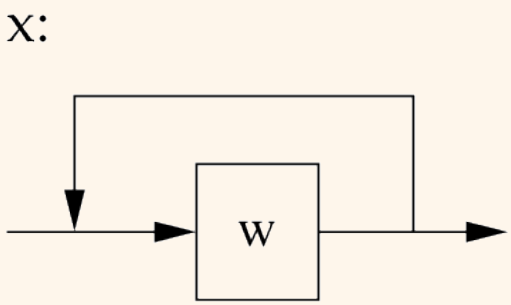
Syntaxdiagramm – 3

- Alternative: <x> ::= <A> | <B> | a

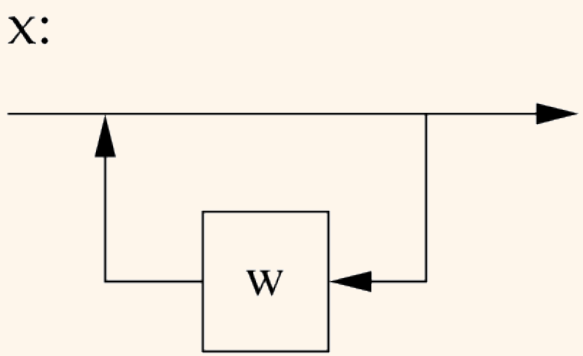


Syntaxdiagramm – 4

- Wiederholung (mind. 1 Mal): <x> ::= <w><x> | <w>



- Wiederholung (auch kein Mal): <x> ::= [<w><x>]



Reguläre Grammatik

- Definition
  - Li Seite: genau ein Non-Terminalsymbol
  - Re Seite
    - ein oder mehrere Terminalsymbole
    - ein Terminalsymbol gefolgt von genau einem NT-Symbol (rechtslinear)
    - ein NT-Symbol gefolgt von genau einem Terminalsymbol (linkslinear).

rechtslinear: $X \to a$ $X \to aY$ linkslinear: $X \to a$ $X \to Ya$

  - **entweder** rechtslinear **oder** linkslinear!
  - von S darf ϵ abgeleitet werden, wenn S nicht auf der re Seite einer Regel vorkommt.

Reguläre Grammatik – Beispiele

- Ges.: vollständige Grammatik und eine Ableitung für $(abc)^2$

$$G = (\Phi, \Sigma, P, S)$$
$$L(G) = \{(abc)^n | n \geq 0\}$$
$$\Phi = \{$$
$$\Sigma = \{$$
$$P = \{$$

Reguläre Grammatik – Beispiele 2

- $L(G) = \{a^n b^n | n \geq 1\}$

Reguläre Grammatik – Beispiele 2

- $L(G) = \{a^n b^n | n \geq 1\}$
  - nicht mittels regulärer Grammatik!
  - nur möglich Worte von li nach re (rechtlinear) oder von re nach li (linkslinear) zeichenweise aufzubauen.
  - Beim Wechsel von der a-Gruppe auf die b-Gruppe besitzt man keine Information mehr über die Länge der bisher erzeugten Zeichen.