7.1.2003

Grammatiken, formale Sprachen und Automaten

Rekapitulation: Termersetzungssysteme (z.B. Logik Kalküle, Texters.-systeme, etc) sind unterschiedliche Formalisierungsmittel zur Abbildung von Sachverhalten der Realität.

Formale Sprachen: Gegeben sei ein Alphabet A und die Menge A^*. Dann heißt jede Teilmenge $L \subseteq A^*$ eine formale Sprache.

Beispiele:

a.) Formalsprache der Mathematik

$A = \{$ Sonderzeichen, Zeichensätze der jeweiligen Sprache $\}$

b.) Menge aller (sinnvollen) deutschen Sätze

$A = \{$ ASCII-Code, Sonderzeichen $\}$

c.) Menge aller syntaktisch korrekten (OCaml-) - Programme

$A = \{$ ASCII-Code, if, let, function, ... $\}$

Beachte: Bei c.) bestehen die Einzelsymbole aus Zeichenfolgen (Tokens). Solche Einzelsymbole müssen vom Compiler als solche erkannt werden.

Frage: Wie beschreibe ich die ∞-große Menge einer Sprache durch endliche Objekte

⇒ Lösung: Verwendung von Grammatiken oder Automaten.

Beispiel: Teilmenge der Regeln der deutschen Sprache zur Erzeugung von Sätzen.

S → NP VP : Ein Satz setzt sich aus einer Nominalphrase (NP) und einer Verbalphrase (VP) zusammen.

NP → N : Nominalphrase ist entw. Nomen

NP → A N : oder Artikel + Nomen

NP → NP PP : oder NP + Präpositionalphrase

VP → V : VP ist entweder V

VP → V NP : oder V gefolgt von NP

VP → V PP : oder V gefolgt von PP

Die Symbole S, NP, N, A, ... sind Nonterminale; diese Nonterminale setzen sich aus Terminalsymbolen zusammen

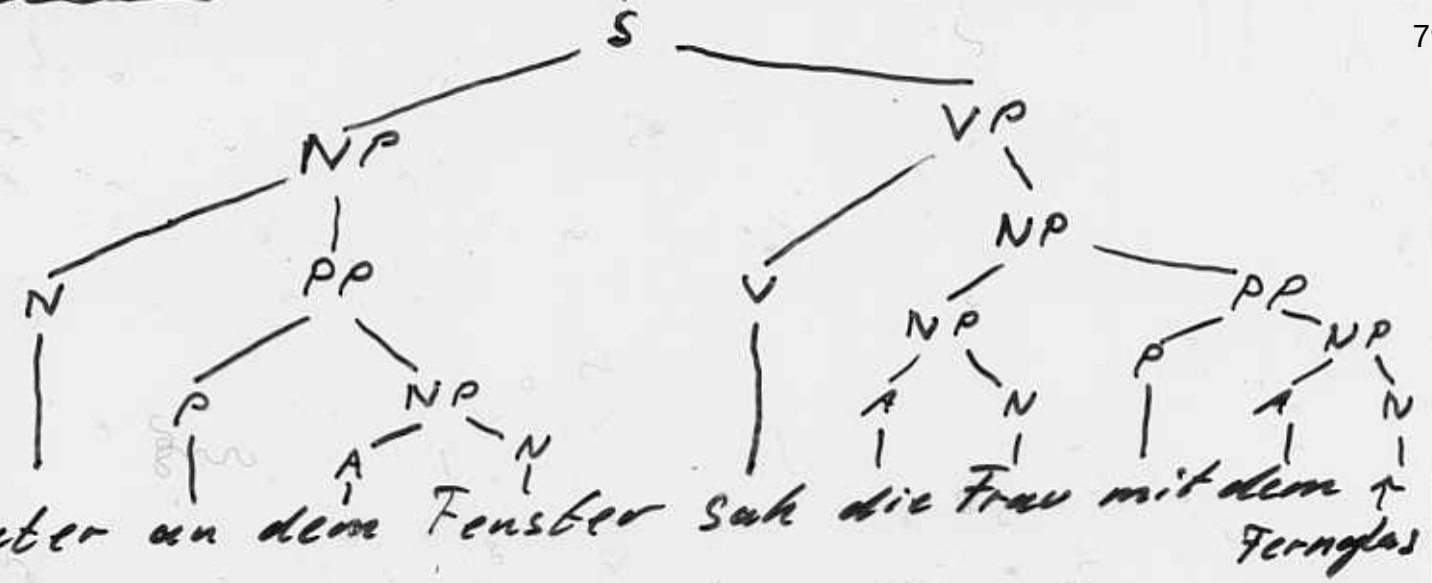
A → der | die | das | ein | ...

P → mit | in | ...

V ...

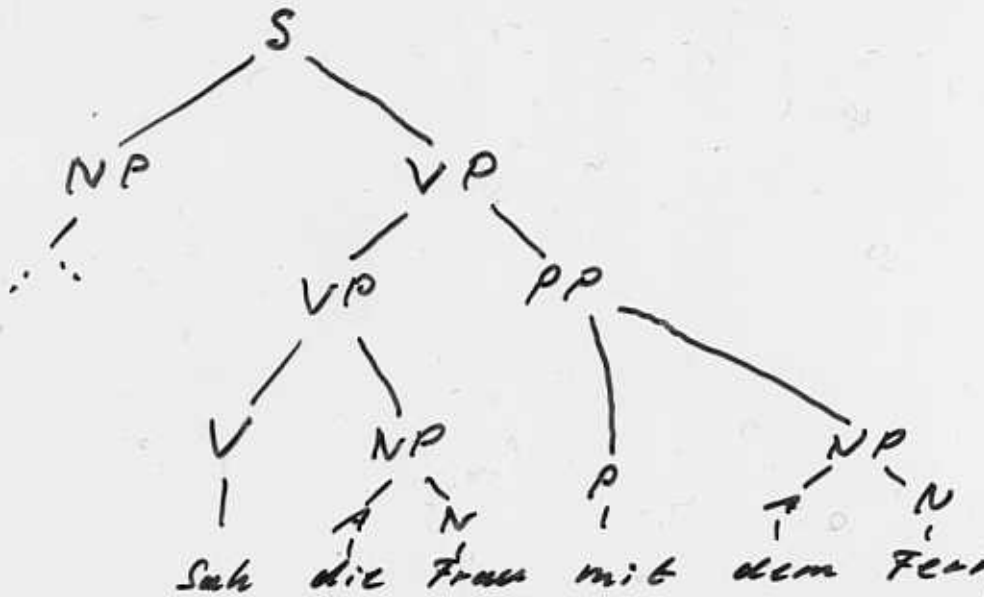
:

Beispiel: Ableitung eines Satzes



Semantik: die Frau hat ein Fernglas

Alternative Ableitung: (rechter Teilbaum)



⇒ Natürliche weisen Konstruktionsmöglichkeiten auf, die zu Mehrdeutigkeiten führen. Solche Mechanismen sollten bei maschineller Interpretation nicht verfügbar sein.

⇒ Grammatiken sollten keine Mehrdeutigkeit enthalten.

Definition von Grammatiken

1. Non-Terminale (Variablen
2. Terminale
3. Startsymbol
4. Regeln: (etwa
   $LS \to RS$, oder $LS = RS$)

Formal: Eine Grammatik ist ein Vier-tupel $G = (V, A, P, S)$

V: endliche Menge von Non-terminalen
A: (endl.) Terminal alphabet
P: Menge der Regeln
S: Startvariable

Eine Folge von Wörtern $(w_0, w_1, \dots, w_n)$ mit $w_0 = S$, $w_n \in A^*$ und
$w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$ heißt Ableitung von w_n

Die von einer Grammatik G erzeugte Sprache L ist gegeben durch

$$L(G) = \{ w \in A^* \mid S \Rightarrow^* w \}$$

"$\Rightarrow^*$" ist die reflexive, transitive Hülle von "$\Rightarrow$"

Notation von Grammatik-Regeln:

a) Backus-Naur Form (BNF)

1. Non-Terminale erhalten „sinnvolle" Namen

und werden in spitzen Klammern geschrieben.

2. Produktionen mit identischer linker Seite werden zu Regeln mit Alternative zusammengefaßt.

Beispiel: $NP \to A\ N$, $NP \to NP\ PP$ $\Big\}$ $NP \to (A\ N) \mid (NP\ PP)$

Beispiel: Natürliche Zahlen

$\langle pziffer \rangle \to 1 \mid 2 \mid \dots \mid 9$

$\langle ziffer \rangle \to 0 \mid \langle pziffer \rangle$

$\langle nat \rangle \to \langle ziffer \rangle \mid \langle pziffer \rangle \langle seq\text{-}of\text{-}ziffer \rangle$

$\langle seq\text{-}of\text{-}ziffer \rangle \to \langle ziffer \rangle \mid \langle ziffer \rangle \langle seq\text{-}of\text{-}zi \rangle$

b.) EBNF (Extended BNF) Verwendung von Gruppierungen:

- $\{\langle Non\text{-}Terminal \rangle\}^*$: Das Non-Terminal kann beliebig oft (auch 0-mal) auftreten.
- $\{\langle Non\text{-}Terminal \rangle\}_{UG}^{OG}$: Das Non-Terminal muß mindestens UG-mal und höchstens OG-mal auftreten.
- $(\langle Non\text{-}Terminal \rangle)$: Gruppierungssymbol.

Beispiel:

$\langle Integer \rangle \to 0 \mid \big((+ \mid -) \langle pziffer \rangle \{\langle ziffer \rangle\}^*\big)$.

c.) EBNF nach Wirth (Entwickler von Pascal, 1977)

1. Non-Terminale werden als Einzelwörter ohne spitze Klammern geschrieben
2. Terminale werden in Anführungszeichen geschrieben (z.B. "BEGIN", "Let", ...)
3. | bezeichnet die Alternative
4. (Non-Terminal) : Gruppierung
5. [Non-Terminal] : optionales Auftreten des Non-terminals
6. { Non-Terminal } : bezeichnet beliebige Wiederholung des Non-Term.
7. = : ersetzt →
8. . : Ende einer Regel

Beispiel:

```
Integer        = (Sign UnsignedInteger) | "0".
UnsignedInteger = Ziffer {"0" | Ziffer}.
Sign           = ["+" | "-"].
Ziffer         = "1" | "2" | ... | "9"
```

Vorteil: Platzsparende Schreibweise zur Spezifikation von Progr.-Sprachen.

Beispiel 2: Beschreibung der EBNF durch EBNF

```
EBNF       = { Produktion }
Produktion = Nonterminal "=" Expression "."
Expression = Term { "|" Term }.
Term       = Factor { Factor }.
Factor     = Non-terminal | Terminal.
           | "[" Expression "]"
           | "(" Expression ")"
           | "{" Expression "}".
Nonterminal = (* Implementierungsabh. *)
Terminale   = (* Implementierungsabh. *)
```