

Lektion 9

In dieser Lektion weitere Begriffe im Zusammenhang mit kontextfreien Grammatiken eingeführt. Es wird u.a. gezeigt, wie mit sog. Ableitungsbäumen anschaulich nachgewiesen werden kann, ob ein Wort zur Sprache einer Grammatik gehört.

Für die Beschreibung der Syntax von Programmiersprachen wird heutzutage üblicherweise nicht mehr die ursprüngliche Form der kontextfreien Grammatiken verwendet, sondern eine Weiterentwicklung, genannt EBNF (extended Backus-Naur form), die in dieser Lektion auch erläutert wird. Mit sog. *Syntaxdiagramme* lernen Sie auch eine grafische Methode zur Beschreibung der Syntax von Programmiersprachen kennen, die eng mit EBNF verwandt ist.

6.2.4 Ableitungsbäume

Das bisher verwendete Konzept für Ableitungen ist sehr einfach: pro Schritt wird eine Regel angewendet. Es gibt allerdings meist sehr viele Möglichkeiten, wo und in welcher Reihenfolge die Produktionen angewendet werden können und die Ableitungen werden schnell unübersichtlich.

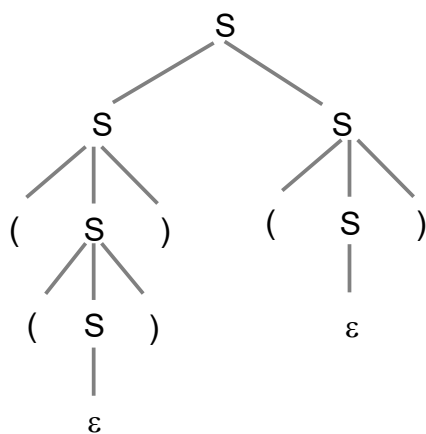
Es gibt neben den (schrittweisen) Ableitungen eine zweite Möglichkeit, wie man zeigen kann, dass ein Wort aus dem Startsymbol ableitbar ist. Der Ableitungsvorgang wird dazu in Baumform repräsentiert.

Beispiel 6.22 - Ableitungsbaum

Wir betrachten folgende Grammatik für die Dyck-Sprache (Klammerfolgen).

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

Ein Ableitungsbaum, der zeigt, dass das Wort $((()())$ ableitbar ist, sieht so aus:



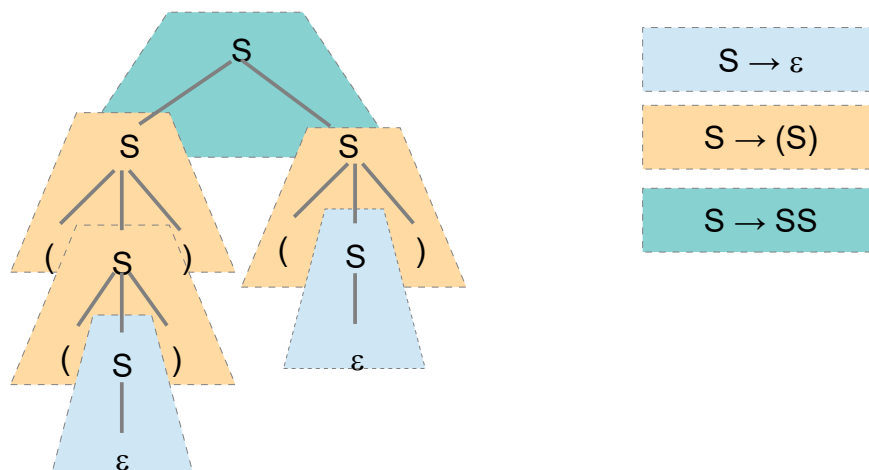
Definition 6.23 - Ableitungsbaum

Ein Baum ist ein **Ableitungsbaum** zu einer kontextfreien Grammatik $G = (N, T, P, S)$ für ein Wort w wenn gilt:

- ❑ Die Wurzel des Baums ist mit dem Startsymbol S markiert.
- ❑ Die inneren Knoten des Baums sind mit einem nichtterminalen Symbol aus N markiert.
- ❑ Die Blätter des Baums sind mit einem terminalen Symbol aus T oder mit ε markiert. Die Blätter ergeben zusammen von links nach rechts das Wort w .
- ❑ Sind für einen inneren Knoten mit Markierung A die Kinder von links nach rechts mit X_1 bis X_n markiert, dann ist $A \rightarrow X_1 X_2 \dots X_n$ eine Produktion der Grammatik.

Anmerkungen

- Folgende Abbildung zeigt den Zusammenhang zwischen Produktionen und den Eltern-Kind-Beziehungen im Baum:



- Andere Bezeichnungen für Ableitungsbäume sind auch:
 - (konkreter) **Syntaxbaum**
 - **Parsebaum** (von engl. *to parse* = syntaktisch analysieren)

Aufgabe 6.24 - Ableitungsbaum

Geben Sie zur Grammatik (wie in Beispiel 6.7)

$$S \rightarrow AD$$

$$A \rightarrow aAc \mid b$$

$$D \rightarrow dD \mid \varepsilon$$

einen Ableitungsbaum für das Wort $aabccdd$ an.

Aufgabe 6.25 - Grammatiken definieren

- (1) Wie kann eine Wiederholung von Teilen, so wie man es mit R^* oder R^+ bei regulären Ausdrücken angeben kann, mit kontextfreien Grammatiken definiert werden? Geben Sie dazu auch die Struktur der Ableitungsbäume an.
- (2) Geben Sie eine kontextfreie Grammatik an, die genau die Sprache erzeugt, die durch den regulären Ausdruck

$$a(b|cc)^*d$$

dargestellt wird.

- (3) Geben Sie eine kontextfreie Grammatik an, die folgende Sprache erzeugt:

$$L = \{ b^{2n} a^k c^n \mid n \geq 0, k \geq 1 \}$$

Aufgabe 6.26 - Ableitungsbäume für arithmetische Ausdrücke

Gegeben ist folgende Grammatik für arithmetische Ausdrücke. Dabei wird auch der Aufbau von Zahlen in der Grammatik definiert.

$$\begin{aligned} \text{Expr} &\rightarrow \text{Zahl} \mid \\ &\quad \text{Expr} + \text{Expr} \mid \\ &\quad \text{Expr} * \text{Expr} \mid \\ &\quad (\text{Expr}) \\ \text{Zahl} &\rightarrow \text{Ziffer} \mid \\ &\quad \text{Zahl Ziffer} \\ \text{Ziffer} &\rightarrow 0 \mid 1 \mid \dots \mid 9 \end{aligned}$$

Geben Sie einen Ableitungsbaum für $3 * 12 + 7$ an.

Anmerkung

- Für ein Wort gibt es normalerweise viele verschiedene Ableitungen, je nachdem, wo man jeweils den nächsten Ableitungsschritt ansetzt. Ein Ableitungsbaum beinhaltet eine Vielzahl dieser Möglichkeiten auf einmal und stellt den Ableitungsprozess dadurch verständlicher und mit Konzentration auf das Wesentliche dar.
- Um zu verstehen, welche Sprache durch eine Grammatik beschrieben wird, hilft es oft, sich vorzustellen, wie die Syntaxbäume aufgebaut sein können.

Der Zusammenhang zwischen sequentiellen Ableitungen und Ableitungsbäumen ist wie folgt.

Eigenschaft 6.27 - Ableitungsbäume und Ableitungen

- Ein Wort w ist genau dann aus dem Startsymbol *ableitbar*, d.h.
 $S \Rightarrow^* w$,
wenn ein *Ableitungsbaum* für das Wort w existiert.

- Zu jedem *Ableitungsbaum* gibt es genau eine *Links-* und genau eine *Rechtsableitung*.

6.2.5 Mehrdeutigkeit von Grammatiken

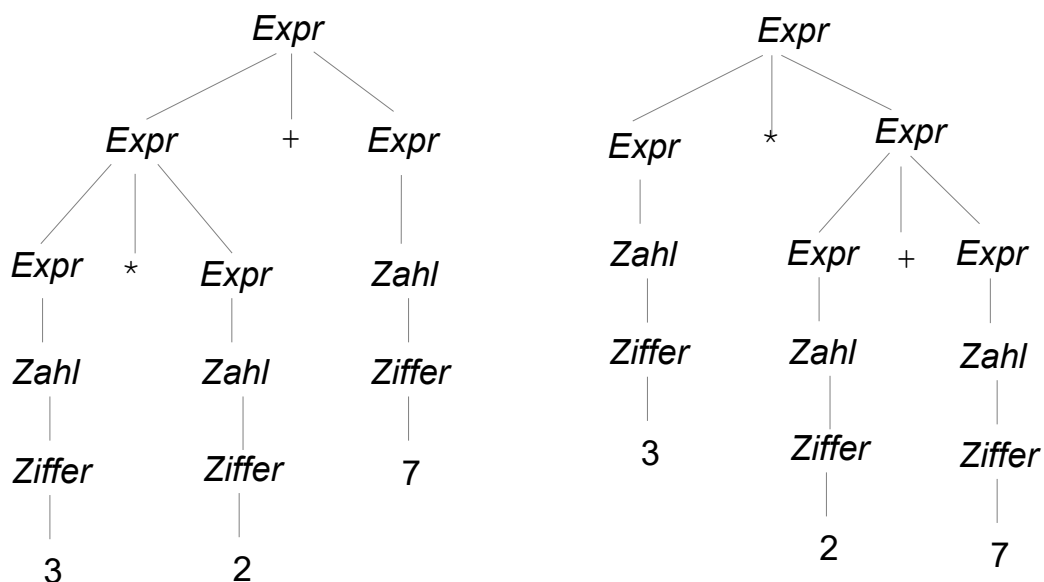
Ein Ableitungsbaum repräsentiert gleichzeitig eine Vielzahl möglicher sequentieller Ableitungen. Gibt es für jedes Wort immer nur einen Ableitungsbaum? Bei manchen Grammatiken ist das der Fall, aber nicht bei allen.

Definition 6.28 - Mehrdeutigkeit von Grammatiken

Eine kontextfreie Grammatik G heißt **mehrdeutig**, wenn es für mindestens ein Wort mindestens *zwei unterschiedliche Ableitungsbäume* gibt.

Beispiel 6.29

Die Grammatik für arithmetische Ausdrücke aus Beispiel 6.26 ist mehrdeutig. Zwei Ableitungsbäume für $3 * 2 + 7$ sind hier dargestellt:



Anmerkung

- ▶ Die Mehrdeutigkeit ist in praktischen Anwendungen (bei künstlichen Sprachen) unerwünscht, da
 - die Semantik vom strukturellen Aufbau abhängt. Mehrdeutigkeit in der Syntax bedeutet dann auch semantische Mehrdeutigkeit (z.B. wäre der rechte Baum oben die falsche Leseweise und würde ein anderes Ergebnis liefern, wenn man so auswerten würde).
 - eine effiziente Syntaxanalyse bei Mehrdeutigkeit nicht möglich ist.
- ▶ Die Grammatik natürlicher Sprachen ist mehrdeutig, wie an folgendem

Beispielsatz deutlich wird:

"Anna wurde mit dem Auto von Berta abgeholt."

Auch wenn die Deutsch-Grammatikregeln einem nicht bewusst sind, ist klar, dass zu den unterschiedlichen Bedeutungen ("Anna wurde von Berta abgeholt, und zwar mit einem Auto" bzw. "Anna wurde mit einem Auto abgeholt, das Berta gehört") auch unterschiedliche syntaktische Strukturen gehören.

Aufgabe 6.30 - Mehrdeutigkeit von Grammatiken

Folgende zwei Grammatiken erzeugen die Dyck-Sprache (siehe Aufgabe 6.18). Sind diese Grammatiken mehrdeutig?

► Grammatik G_1 :

$$\begin{aligned} S &\rightarrow \varepsilon \\ &| (S) \\ &| SS \end{aligned}$$

► Grammatik G_2 :

$$\begin{aligned} S &\rightarrow \varepsilon \\ &| (S) S \end{aligned}$$

6.2.6 Äquivalenz von Grammatiken

Eine Sprache (= Menge von Wörtern) kann durch unterschiedliche kontextfreie Grammatik beschrieben werden.

Definition 6.31 - Äquivalenz von Grammatiken

Zwei Grammatiken G_1 und G_2 heißen **äquivalent**, wenn sie dieselbe Sprache erzeugen, d.h. wenn

$$L(G_1) = L(G_2)$$

Beispiel 6.32 - äquivalente Grammatiken

Die beiden Grammatiken für die Dyck-Sprache sind äquivalent, da sich jeweils die gleiche Menge von Wörtern ableiten lässt.

Beispiel 6.33 - äquivalente Grammatiken

► Grammatik G_1 :

$$\begin{aligned} S &\rightarrow Aa \\ A &\rightarrow aA \mid \varepsilon \end{aligned}$$

► Grammatik G_2 :

$$B \rightarrow Ba \mid a$$

Beide Grammatiken erzeugen eine Folge von einem oder mehreren a, d.h.

$$L(G_1) = L(G_2) = \{ a^n \mid n > 0 \}.$$

Somit sind beide Grammatiken äquivalent.

Aufgabe 6.34 - Äquivalenz von Grammatiken

Sind folgende Grammatiken äquivalent?

► Grammatik G_1 :

$$S \rightarrow AB$$

$$A \rightarrow Aa \mid \varepsilon$$

$$B \rightarrow bB \mid b$$

► Grammatik G_2 :

$$S \rightarrow ASB \mid b$$

$$A \rightarrow aA \mid \varepsilon$$

$$B \rightarrow Bb \mid \varepsilon$$

6.2.7 Kontextfreie Sprachen

Erinnern Sie sich: eine Sprache ist eine Menge von Wörtern. Es gibt unterschiedliche Methoden, wie man Sprachen definieren kann. Sprachen, die durch *kontextfreie Grammatiken* beschrieben werden können, werden *kontextfreie Sprachen* genannt.

Definition 6.35 - kontextfreie Sprachen

Eine **Sprache** L heißt **kontextfrei**, wenn es eine kontextfreie Grammatik G gibt, die die Sprache darstellt, d.h. mit $L(G) = L$.

Aufgabe 6.36 - kontextfreie Sprachen

Welche der folgenden Sprachen über dem Alphabet $\Sigma = \{a, b, c\}$ sind kontextfrei?

$$L_0 = \{ a^n b^k \mid n \geq 0, k \geq 0 \}$$

$$L_1 = \{ a^n b^n \mid n \geq 0 \}$$

$$L_2 = \{ a^n b^n c^n \mid n \geq 0 \}$$

$$L_3 = \{ w \in \Sigma^* \mid w \text{ ist ein Palindrom} \}$$

6.3 Weitere Formen der Syntaxbeschreibung

Ausgehend vom Konzept der kontextfreien Grammatiken wurden im Laufe der Zeit verschiedene Beschreibungsmethoden für die Syntax von Programmiersprachen (und auch anderen Sprachen) entwickelt. Die gebräuchlichsten werden in den folgenden Abschnitten vorgestellt:

- ▶ Backus-Naur-Form (BNF)
- ▶ Erweiterte Backus-Naur-Form (EBNF)
- ▶ Syntaxdiagramme

Diese Beschreibungsformen sind gleich mächtig wie kontextfreie Grammatiken!

6.3.1 Backus-Naur-Form (BNF)

Der 1960 veröffentlichte Entwurf für die Programmiersprache Algol 60 war ein Meilenstein in der Geschichte der Programmiersprachen, denn es wurde erstmals exakt die Syntax einer Programmiersprache unabhängig von einer Implementierung definiert. Letztendlich ist dadurch erst der Begriff der Programmiersprache entstanden. Bis dahin gab es immer nur rechnerabhängige Programmiersysteme, so dass Programme auch nicht zwischen Rechnern unterschiedlicher Hersteller ausgetauscht werden konnten.

Für Algol 60 wurde die kurz zuvor in der Linguistik entwickelte Methode der kontextfreien Grammatiken verwendet. Die Produktionen wurden in der sog. *Backus-Naur-Form (BNF)* geschrieben, benannt nach John Backus und Peter Naur, die maßgeblich am Entwurf der Sprache beteiligt waren. Es wurde folgende Notation verwendet:

- ▶ In Produktionen wird ::= statt → verwendet
- ▶ Nichtterminale Symbole werden mit $\langle \dots \rangle$ geklammert
- ▶ Terminale Symbole werden mit Apostroph "..." eingeschlossen (oder ggf. durch Fettdruck gekennzeichnet).

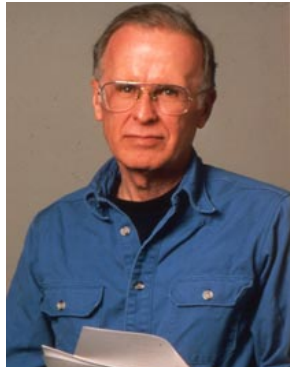
Beispiel 6.37 - Algol 60-Syntax in BNF

Folgendes ist ein Ausschnitt aus dem Algol 60-Report:

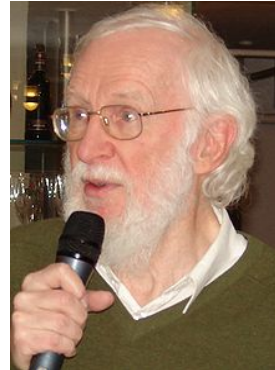
```
 $\langle \text{unlabelled basic statement} \rangle ::= \langle \text{assignment statement} \rangle \mid \langle \text{go to statement} \rangle \mid$   
 $\langle \text{dummy statement} \rangle \mid \langle \text{procedure statement} \rangle$   
 $\langle \text{basic statement} \rangle ::= \langle \text{unlabelled basic statement} \rangle \mid \langle \text{label} \rangle " : " \langle \text{basic statement} \rangle$   
 $\langle \text{unconditional statement} \rangle ::= \langle \text{basic statement} \rangle \mid \langle \text{for statement} \rangle \mid \langle \text{compound statement} \rangle \mid$   
 $\langle \text{block} \rangle$ 
```

$\langle \text{statement} \rangle ::= \langle \text{unconditional statement} \rangle \mid \langle \text{conditional statement} \rangle$
 $\langle \text{compound tail} \rangle ::= \langle \text{statement} \rangle \text{"end"} \mid \langle \text{statement} \rangle \langle \text{compound tail} \rangle$
 $\langle \text{block head} \rangle ::= \text{"begin"} \langle \text{declaration} \rangle \mid \langle \text{block head} \rangle \langle \text{declaration} \rangle$
 $\langle \text{unlabelled compound} \rangle ::= \text{"begin"} \langle \text{compound tail} \rangle$
 $\langle \text{unlabelled block} \rangle ::= \langle \text{block head} \rangle \text{";" } \langle \text{compound tail} \rangle$
 $\langle \text{compound statement} \rangle ::= \langle \text{unlabelled compound} \rangle \mid \langle \text{label} \rangle \text{":" } \langle \text{compound statement} \rangle$
 $\langle \text{block} \rangle ::= \langle \text{unlabelled block} \rangle \mid \langle \text{label} \rangle \text{":" } \langle \text{block} \rangle$

[Peter Naur, J.W. Backus et. al.: Report on the Algorithmic Language ALGOL 60, COMMUNICATIONS OF THE ASSOCIATION FOR COMPUTING MACHINERY, Vol. 3, No.5, May 1960]



John Backus (1928 - 2007)
1977 Turing award



Peter Naur (*1928 - 2016)
2005 Turing award

BNF ist also nichts anderes als kontextfreie Grammatiken mit einer speziellen Schreibweise für die Produktionen.

6.3.2 Erweiterte Backus-Naur-Form (EBNF)

Wie Sie in vorangehenden Beispiele schon gesehen haben, kann man mit kontextfreien Grammatiken auch Wiederholungen, alternative und optionale Teile beschreiben. Allerdings ist dies, im Vergleich zu regulären Ausdrücken, relativ umständlich.

Von Nikolaus Wirth wurde deshalb Anfang der 1970er Jahre zur Beschreibung der Syntax der Programmiersprache Pascal die sog. *Erweiterte Backus-Naur-Form* (EBNF) eingeführt, die die Vorteile kontextfreier Grammatiken und regulärer Ausdrücke kombiniert.

EBNF-Produktionen

Auf der rechten Seite von Produktionen sind bei EBNF folgende Konstrukte erlaubt, analog wie bei regulären Ausdrücken.

EBNF-Notation	Erläuterung
M^*	Wiederholung: M beliebig oft (auch 0 mal)
M^+	Wiederholung: M ein- oder mehrfach
$M_1 \mid M_2$	Alternative: M_1 oder M_2
$[M]$	Optionalen Teil: M kann entfallen
(M)	Klammern zur Strukturierung

Beispiel 6.38 - EBNF-Grammatik

$S \rightarrow (bA \mid aa)^*$
 $A \rightarrow a [Ab] a$

Anmerkung

- ▶ In der Praxis wird EBNF häufig für Sprachdefinitionen verwendet, z.B. auch für die Definition der Syntax von Java, siehe <https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html>.
- ▶ Es gibt dabei oft leicht unterschiedliche Notationsvarianten. Gängig sind beispielsweise auch folgende Notationsformen:

$\{M\}$ statt M^* beliebige Wiederholung
 $(M)?$ statt $[M]$ optionaler Teil

Beispiel 6.39 - EBNF-Syntax von HSULogo

Die Syntax der in Kapitel 4 schon vorgestellten Programmiersprache HSULogo ist folgendermaßen mit EBNF definiert. Terminale Symbole sind hier in einfache Anführungszeichen '...' eingeschlossen.

$\text{program} \rightarrow \text{'program' STRING (definition \mid statement)^* EOF}$
 $\text{definition} \rightarrow \text{'to' ID '(' (VAR)^* ') 'statement'* 'end'}$
 $\text{statement} \rightarrow$
 $\quad (\text{'fd' \mid 'forward'}) \text{expression}$
 $\quad \mid (\text{'tr' \mid 'turnright'}) \text{expression}$
 $\quad \mid (\text{'tl' \mid 'turnleft'}) \text{expression}$
 $\quad \mid \text{'home'}$
 $\quad \mid \text{'clear'}$
 $\quad \mid (\text{'pu' \mid 'penup'})$
 $\quad \mid (\text{'pd' \mid 'pendown'})$
 $\quad \mid \text{'setpencolor' expression expression expression}$
 $\quad \mid \text{'moveto' expression expression}$
 $\quad \mid \text{'make' VAR expression}$
 $\quad \mid \text{'repeat' expression block}$
 $\quad \mid \text{'if' condition block}$

```

| 'read' STRING VAR
| ID '(' expression* ')'

block      →   '[' statement* ']'

expression →   ..

```

Beispielsweise sieht man in der Produktion für *program*, dass ein Programm mit dem Schlüsselwort `program` und einem String als Programmnamen beginnt und danach aus einer beliebig langen Folge von Definitionen und Anweisungen besteht.

Aufgabe 6.40 - Ausdrücke mit Funktionsaufrufen in EBNF

In Aufgabe 6.26 - Ableitungsbäume für arithmetische Ausdrücke wurde eine kontextfreie Grammatik für arithmetische Ausdrücke mit Zahlen, Variablen und binären Operatoren vorgestellt. Verwenden Sie nun EBNF, um auf möglichst kompakte Weise den Aufbau von Ausdrücken zu beschreiben, die auch Funktionsaufrufe enthalten können.

Ausdrücke sollen also aus folgenden Teilen aufgebaut werden können:

- ▶ Zahlen,
- ▶ Variablen,
- ▶ binären Operatoren (+, -, *, /),
- ▶ Klammern,
- ▶ Funktionsaufrufen mit beliebig vielen Argumenten, die ggf. durch Kommas getrennt sind.

Hier sind einige Beispiele für Ausdrücke:

```

7
(13 + var) * 7
f1(x, 3 * (y + 1), f2())

```

Zahlen (z.B. 13) und Variablen (z.B. `var`) sollen dabei als ein terminales Symbol betrachtet werden. Funktionsnamen und Variablennamen seien beide als Bezeichner (engl. *Identifier*) definiert.

6.3.3 Umformung von EBNF in kontextfreie Grammatik

Jede EBNF-Grammatik lässt sich systematisch in eine äquivalente kontextfreie Grammatik umformen. Wir betrachten dazu, wie die EBNF-Operatoren $*$, $+$, $|$ und $[]$ innerhalb der rechten Seite einer EBNF-Produktion eliminiert werden können:

(1) $A \rightarrow u v^* w$: wobei $u, v, w \in (N \cup T)^*$

Wir führen ein neues nichtterminales Symbol X für den v^* -Teil ein. Durch die Produktionen für X wird eine beliebige Wiederholung von v beschrieben.

$$\begin{aligned} A &\rightarrow u X w \\ X &\rightarrow v X \\ &\quad | \varepsilon \end{aligned}$$

(2) $A \rightarrow u v^+ w$:

Möglichkeit 1: Dies kann einfach umgewandelt werden zu

$$A \rightarrow u v v^* w$$

und dann kann v^* wie oben behandelt werden.

Möglichkeit 2: Alternativ könnte auch gleich umgewandelt werden mit neuem Nichtterminal X , dass X eine Folge von mindestens einmal v erzeugen kann.

$$\begin{aligned} A &\rightarrow u X w \\ X &\rightarrow v X \\ &\quad | v \end{aligned}$$

(3) $A \rightarrow u (v_1 | v_2) w$:

Auch hier führen wir eine neues Nichtterminal X für den Teil $(v_1 | v_2)$ ein.

$$\begin{aligned} A &\rightarrow u X w \\ X &\rightarrow v_1 \\ &\quad | v_2 \end{aligned}$$

(4) $A \rightarrow u [v] w$:

Möglichkeit 1: Der optionale Teil $[v]$ wird durch ein Nichtterminal X ersetzt

$$\begin{aligned} A &\rightarrow u X w \\ X &\rightarrow v \\ &\quad | \varepsilon \end{aligned}$$

Möglichkeit 2: Wir könnten $[v]$ als Abkürzung für $(v | \varepsilon)$ betrachten und dann wie oben umwandeln.

Durch wiederholte Anwendung dieser Umformungen kann jede rechte Seite von EBNF-Produktionen nach und nach so weit vereinfacht werden, bis alle Produktionen einer kontextfreien Grammatik entsprechen.

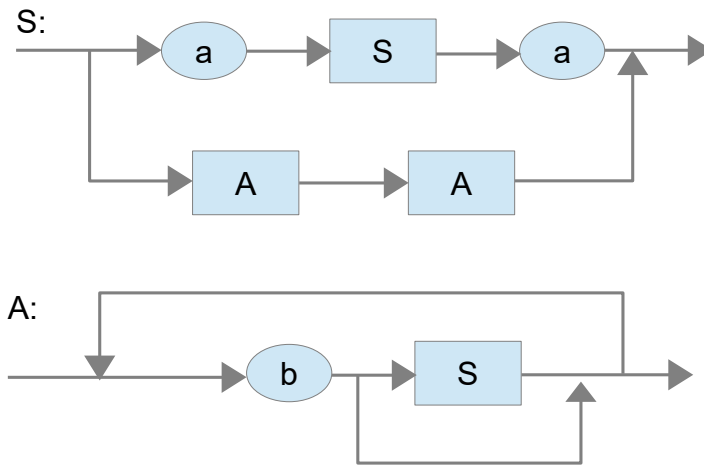
Aufgabe 6.41 Umwandlung EBNF/kontextfreie Grammatik

Wandeln Sie folgende EBNF-Grammatik in eine kontextfreie Grammatik um:

$$\begin{aligned} S &\rightarrow (bA | aa)^* \\ A &\rightarrow a [Ab] a \end{aligned}$$

6.3.4 Syntaxdiagramme

Syntaxdiagramme (engl. auch *railroad diagrams*) sind eine anschauliche grafische Darstellung von EBNF-Regeln, die von **Nikolaus Wirth** populär gemacht wurde, um die Syntax der Programmiersprache Pascal benutzerfreundlich darstellen zu können.



Nikolaus Wirth (* 1934)
1984 Turing award

Ein Syntaxdiagramm ist ein gerichteter Graph mit einem Eingang und einem Ausgang, in dem die Knoten terminale oder nichtterminale Symbole sind. Für jedes nichtterminale Symbol gibt es einen eigenen Graphen. Ein Pfad vom Eingang zum Ausgang entspricht einer syntaktisch korrekten Symbolfolge.

Beispiel 6.42 - Syntaxdiagramm für arithmetische Ausdrücke

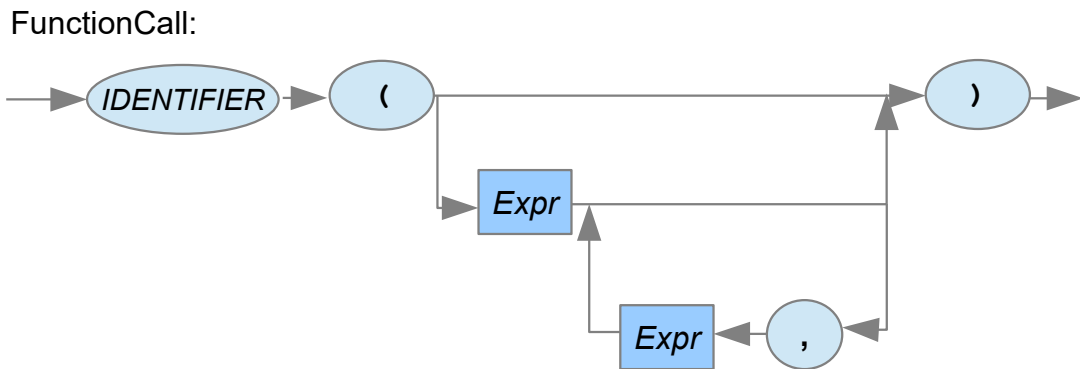
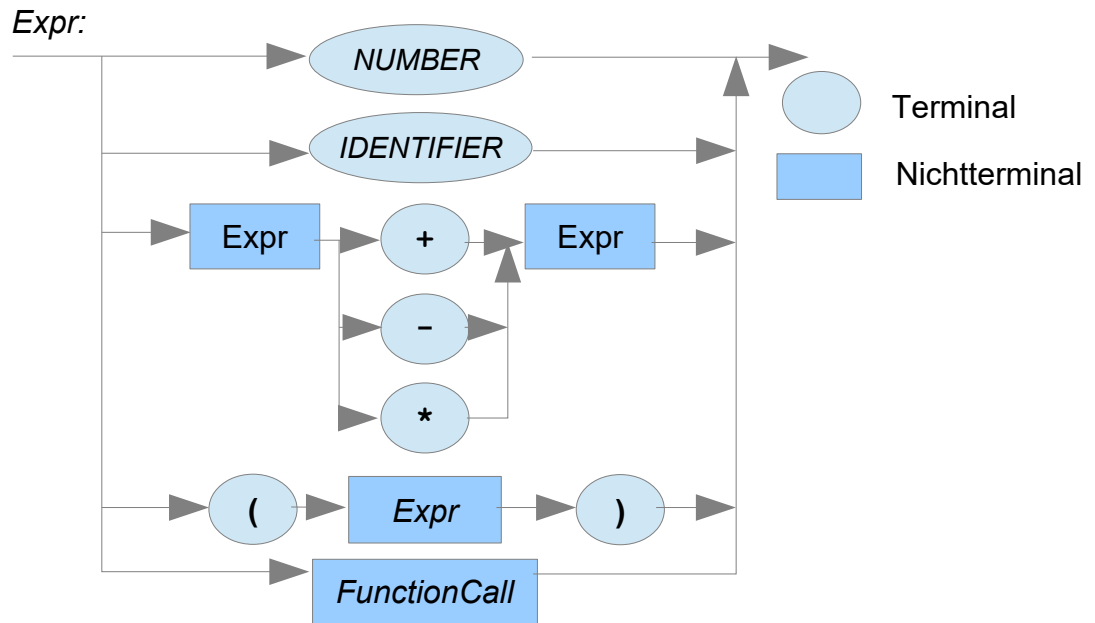
Die EBNF-Grammatik für Ausdrücke mit Funktionsaufrufen aus Aufgabe 6.40

$Expr \rightarrow$

- $NUMBER$
- $IDENTIFIER$
- $Expr \, ("+" \, | \, "-" \, | \, "*") \, Expr$
- $" (" \, Expr \, ") "$
- $FunctionCall$

$FunctionCall \rightarrow IDENTIFIER \, " (" \, [\, Expr \, ("," \, Expr \,)^* \,] \, ") "$



würde folgendermaßen dargestellt:



6.3.5 Zusammenhang Syntaxdiagramme und EBNF

Jede EBNF-Grammatik lässt sich ganz systematisch nach folgenden Regeln in ein Syntaxdiagramm umwandeln.

EBNF-Konstrukt	Bedeutung	Darstellung im Syntaxdiagramm
$M \mid N$	Alternative	
M^*	Wiederholung ≥ 0 mal	

EBNF-Konstrukt	Bedeutung	Darstellung im Syntaxdiagramm
M^+	Wiederholung ≥ 1 mal	
$[M]$	optionaler Teil	

Aufgabe 6.43 - JSON-Syntax in EBNF und als Syntaxdiagramm

Neben XML ist *JSON* (*JavaScript Object Notation*) ein weiteres populäres textbasiertes Austauschformat für strukturierte Daten, das in zahlreichen Anwendungen verwendet wird. Ein Beispiel für ein JSON-Dokument, das zwei JSON-Objekte beschreibt, ist hier zu sehen:

```
{
  "Name" : "Mustermann",
  "Vornamen" : [ "Xaver", "Yannik", "Zorro" ],
  "Adresse" : { "Strasse" : "Paradiesgasse",
                "Hausnummern" : [ 5, 7, 9 ],
                "PLZ" : 93462,
                "Ort" : "Himmelreich"
              }
}

{
  "Name" : "Musterfrau",
  "Vornamen" : ["Anny", "Betty"]
}
```

Ein **JSON-Dokument** besteht entweder aus einem einzelnen Wert oder einer Folge von mehreren Werten. Ein **Wert** kann ein String `"..."` sein, eine Zahl, ein Wahrheitswert `true` oder `false`, der Wert `null`, ein Array oder ein Objekt.

- ▶ Ein **String** wird in Anführungszeichen `"..."` eingeschlossen.
- ▶ Ein **JSON-Objekt** wird in geschweifte Klammern `{ ... }` eingeschlossen und besteht aus einer möglicherweise leeren Folge von Paaren der Form *Name : Wert*, die durch Kommas getrennt werden. Dabei ist der Name ein String.
- ▶ Ein **Array** ist eine Folge von beliebig vielen Werten, ggf. durch Kommas getrennt und mit eckigen Klammern `[...]` eingeschlossen.
- ▶ Objekte und Arrays können beliebig ineinander verschachtelt sein.

Ihre Aufgabe ist nun folgende:

- (1) Definieren Sie eine EBNF-Grammatik für JSON-Dokumente. *String* und *Number* können Sie dabei als bereits definierte Symbole voraussetzen. Als Startsymbol
- (2) Stellen Sie die Grammatik als Syntaxdiagramm dar.

Fazit für Lektion 9

Diese Fragen sollten Sie nun beantworten können

- ▶ Was stellt ein Ableitungsbaum dar?
- ▶ Welcher Zusammenhang besteht zwischen Ableitungen und Ableitungsbäumen?
- ▶ Wann ist eine kontextfreie Grammatik mehrdeutig?
- ▶ Wann sind zwei kontextfreie Grammatiken äquivalent?
- ▶ Was ist BNF und EBNF?
- ▶ Wie können EBNF-Grammatiken in kontextfreie Grammatiken umgewandelt werden?
- ▶ Was Syntaxdiagramme?
- ▶ Welche Zusammenhang besteht zwischen EBNF und Syntaxdiagrammen?