FORMALE SPRACHEN

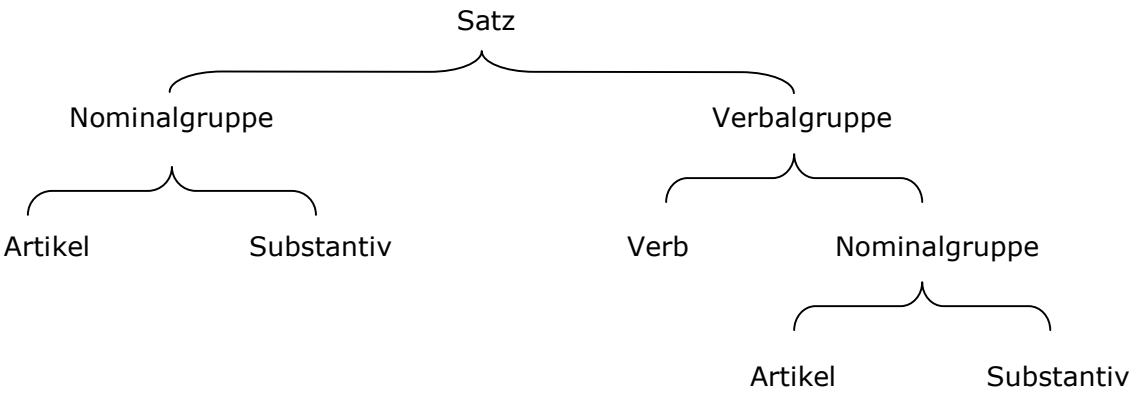
Syntax: „Lehre vom Satzbau"
Die Syntax beschreibt die Gesetzmäßigkeiten, gemäß denen Wörter zu einem Satz zusammengefügt werden.

Semantik: „Lehre von der Bedeutung (von Zeichen, Wörtern und Sätzen)"

Analyse folgenden Satzes:

„Die Katze jagt die Maus"

Syntaxbaum:



Eine Grammatik wird wesentlich beschrieben durch ein System von Regeln, z.B. in der ***Backus-Naur-Form*** (BNF-Notation):

<SATZ> ::= <NOMINALGRUPPE> <VERBALGRUPPE>

<NOMINALGRUPPE> ::= <ARTIKEL> <SUBSTANTIV>

<VERBALGRUPPE> ::= <VERB> <NOMINALGRUPPE>

<VERB> ::= jagt | sieht | beißt | frißt

<ARTIKEL> ::= der | die | das

<SUBSTANTIV> ::= Katze | Maus | Merlin | Tablet

„ ::= " ist zu lesen als: „wird ersetzt durch"
„ | " ist zu lesen als: „oder"

Bemerkung: In spitzen Klammern eingeschlossene Symbole sind nicht endgültige Zeichen, sogenannte **Non-Terminalzeichen (Nonterminals)**; die anderen Zeichen, wie z. B. "der" oder "Katze" im obigen Beispiel, werden nicht mehr durch andere Zeichen ersetzt und heißen daher endgültige Zeichen, **Terminalzeichen (Terminals)**.
Die Ersetzungsregel „ <ARTIKEL> ::= der | die | das " bedeutet, daß das Non-Terminal „<ARTIKEL>" durch die Terminals „der" oder „die" oder „das" ersetzt werden kann und auch zu ersetzen ist, denn der endgültige Satz besteht aus lauter Terminals.

*Beachte: Mit dem Begriff „Zeichen" ist hier nicht ein Buchstabe oder eine Ziffer im Sinne von „character" (char) gemeint, sondern die Terminalzeichen sind bei einer natürlichen Sprache die einzelnen Wörter der Sprache, bei einer Programmiersprache Schlüsselwörter (z. B. **while**, **print**, **if**, **else** etc. in Python) oder Bezeichner. Folglich sind die Sätze, die gemäß den Syntaxregeln einer die Programmiersprache definierenden Grammatik gebildet werden können, nichts anderes als die in dieser Sprache formulierten Programmtexte.*

Sätze, die gemäß obenstehenden Regeln aufgebaut sind:

"*der Merlin beißt das Tablet*"
"die Maus sieht die Katze"
"das Katze frißt die Maus"

Die Syntaxregeln müssen zur Bildung korrekter Sätze eingehalten werden; andererseits impliziert die Einhaltung der Regeln nicht zwingend, daß ein syntaktisch korrekter Satz auch semantisch Sinn macht.

DEFINITION:

Wenn A eine endliche Menge von Zeichen ist, erhält man durch deren Hintereinanderschreiben Zeichenketten. Die Menge A heißt auch Alphabet, die Zeichenketten heißen Wörter über dem Alphabet A; das leere Wort, das keine Zeichen enthält, heißt ε.
Jede Menge von Wörtern über A heißt eine formale Sprache; ein System von Regeln, welches entscheidet, ob ein Wort w über A zur Sprache gehört, heißt Grammatik G (oder Syntax) einer formalen Sprache.

In *Python* sind

- die Zeichen oder Symbole der *formalen Sprache*: Schlüsselwörter wie print, if, input, else, elif, return, . . . , Namen
- die Zeichenketten oder Wörter der *formalen Sprache*: Python-Programme

Eine *Grammatik* beinhaltet *Regeln*, mit Hilfe derer entschieden wird, ob ein *Wort* (also ein *Programm-Text*) z. B. ein gültiges *Python*-Programm ist. Dieser Vorgang heißt *Syntax-Analyse*; letztere erledigt ein Parser, der Bestandteil jedes Compilers und jedes Interpreters ist. Ein syntaktisch korrekter *Programm-Text* ist nicht hinreichend, daß das Programm auch etwas "Vernünftiges" leistet; die Bedeutung eines *Programm-Textes* (oder eines Textes einer *natürlichen Sprache*) wird mit dem Begriff *"Semantik"* erfaßt.

Wir unterscheiden bei einer *formalen Sprache terminale* ("endgültige") und *nicht-terminale* ("nicht endgültige") Zeichen oder Symbole.

DEFINITION:

Eine *Satzgliederungsgrammatik* **G** ist durch folgende Bestandteile gegeben:

(1) eine *endliche Menge* **T**; ihre Elemente heißen *Terminalzeichen* oder *Terminals*.
(2) eine *endliche Menge* **N**; ihre Elemente heißen *nicht-terminale Zeichen* oder *Non-Terminals*; in dieser Menge **N** ist ein *Startzeichen* **S** ausgezeichnet.
(3) endlich viele *Ersetzungsregeln*, genannt *Produktionen* **P**.

Die von der *Grammatik* **G** bestimmte *formale Sprache* **L(G)** besteht aus allen *Wörtern* (bzw. *Zeichenketten, Sätzen, Programmtexten*) über **T**, die – ausgehend vom Startzeichen **S** – durch endlich viele Anwendungen der Produktionen erzeugt werden können.

Somit ist eine Grammatik **G** durch das Quadrupel **(T, N, P, S)** bestimmt.

Beispiel einer einfachen *formalen Sprache*:

Das *Non-Terminal* **S** sei Startzeichen, **a**, **b** seien *terminale* Symbole.
Ersetzungsregeln **P** der Grammatik **G**:

(1) S ::= a
(2) S ::= a S a
(3) S ::= S b

Bemerkung:
Die in der BNF-Notation für Nonterminals vorgesehenen spitzen Klammern wurden hier weggelassen.

Zeige jeweils mittels einer Linksableitung und eines Syntaxbaums, daß die Wörter

a) aaa , b) aaba , c) abbb , d) aababb

zur Sprache L(G) gehören.

Beachte:

Linksableitung bedeutet, daß jeweils das am weitesten links stehende *Nonterminal-Zeichen* ersetzt wird.

Solange noch ein **S** vorkommt, muß **S** ersetzt werden, bis das entstandene *Wort* aus lauter *Terminal-Zeichen* besteht.

Linksableitungen (ausgehend vom Startsymbol **S**; „top-down"):

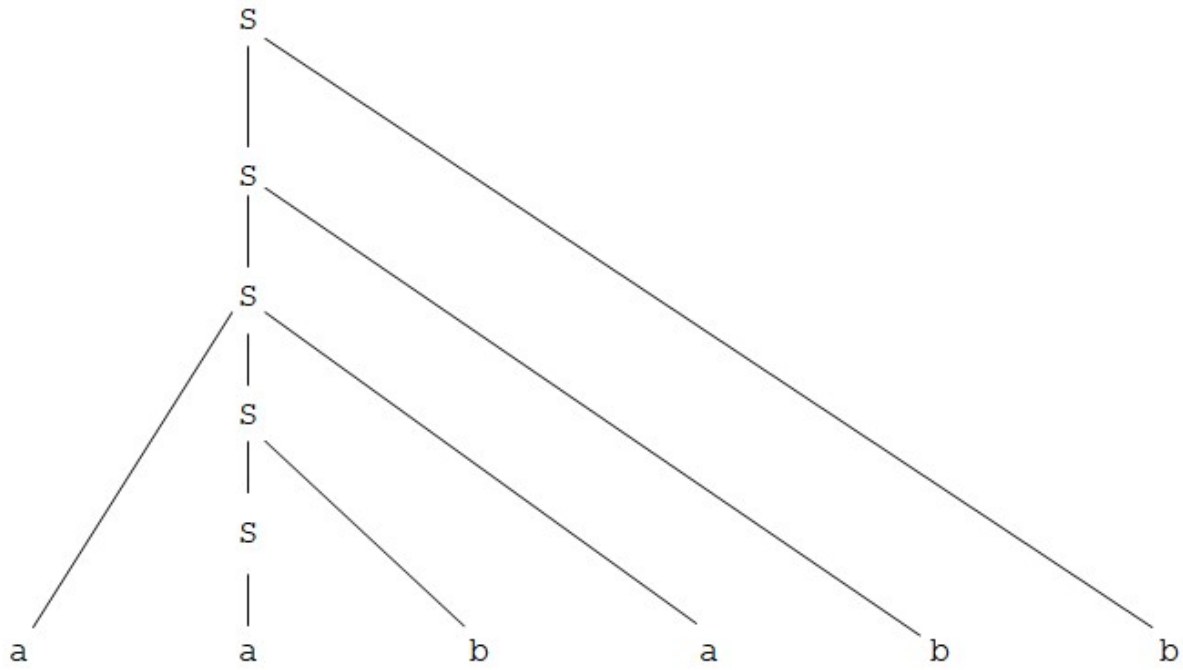
a) $S \xrightarrow{2} a\ S\ a \xrightarrow{1} a\ a\ a$

b) $S \xrightarrow{2} a\ S\ a \xrightarrow{3} a\ S\ b\ a \xrightarrow{1} a\ a\ b\ a$

c) $S \xrightarrow{3} S\ b \xrightarrow{3} S\ b\ b \xrightarrow{3} S\ b\ b\ b \xrightarrow{1} a\ b\ b\ b$

d) $S \xrightarrow{3} S\ b \xrightarrow{3} S\ b\ b \xrightarrow{2} a\ S\ a\ b\ b \xrightarrow{3} a\ S\ b\ a\ b\ b \xrightarrow{1} a\ a\ b\ a\ b\ b$

Syntaxbaum zu d):



Das Wort a a b a b b läßt sich auf das Startsymbol S zurückführen („bottom-up").

REGULÄRE SPRACHEN (TYP 3)

DEFINITION:

Eine Grammatik **G** heißt *regulär*, wenn alle Produktionen von der Form

(R) **A ::= aB ,** **A ::= a** **(Rechtslinearität)**

sind,

oder wenn alle Produktionen die Form

(L) A ::= Ba , A ::= a (Linkslinearität)

haben.

Die zugehörige formale Sprache **L(G)** heißt ***regulär***.

Wir beschränken uns im folgenden auf linkslineare Grammatiken, was keine Einschränkung darstellt; ***eine linksreguläre Grammatik nennen wir auch Grammatik vom Typ 3, die zugehörige reguläre Sprache heißt vom Typ 3****.*

Vereinbarung: Im Folgenden schreiben wir statt „ ::= " auch „ → " .

DEFINITION:

Ein **endlicher Automat (Akzeptor)** ist bestimmt durch

- eine nichtleere, endliche Menge **Z** von Zuständen,
- eine nichtleere, endliche Menge **E** von Eingabesymbolen (Eingabealphabet),
- eine Überführungsfunktion f : Z x E → Z, die jedem Paar aus aktuellem Zustand und Eingabe eindeutig einen Folgezustand zuordnet,
- einen Anfangszustand $\mathbf{z_0}$ aus der Menge **Z**,
- mindestens einen Endzustand $\mathbf{z_E}$ aus der Menge **Z**.

Wir verdeutlichen einen endlichen Automaten durch einen **Graphen**: Für jedem Zustand aus **Z** zeichnen wir einen Knoten. Von den Knoten gehen gerichtete Kanten aus, wobei eine Kante mit dem jeweiligen Eingabesymbol aus der Menge **E** beschriftet wird. Eine Kante endet bei demjenigen Knoten (Zustand), in den der Automat nach Lesen des Eingabesymbols übergeht.

Es gilt folgender

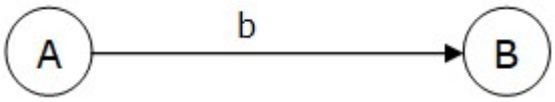
SATZ: Ein endlicher Automat erkennt eine Sprache genau dann, wenn sie regulär ist.

Der strenge Beweis dieses Satzes ist schwierig; für linkslineare Grammatiken führen wir konstruktiv eine Plausibilitätsbetrachtung durch:

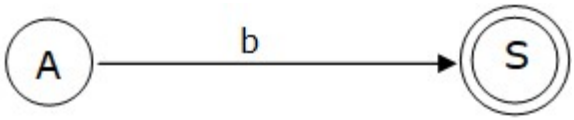
(1) Jedem Element der Menge **N** der Nonterminals einer Sprache ist ein Zustand (Knoten) des endlichen Automaten zugeordnet; Ausnahme: dem Anfangszustand entspricht kein Nonterminalzeichen. Das Nonterminal **S** (Startzeichen) entspricht dem Endzustand.

(2) Einer Produktion B → Ab entspricht eine gerichtete Kante mit der Bewertung b (b ∈ **T**) vom Knoten A (Zustand A) zum Knoten B (Zustand B).

B → Ab

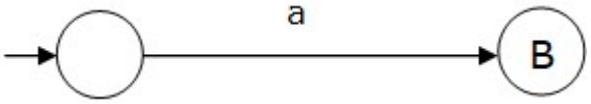


S → Ab



(3) Einer Produktion B → a entspricht eine gerichtete Kante vom Anfangszustand zum Knoten B (Zustand B) mit der Bewertung a, a ∈ **T**. Auf den Anfangszustand dürfen keine Kanten hinführen.

B → a



Ein Wort **w** einer Sprache **L(G)** wird genau dann erkannt, wenn, ausgehend vom Anfangszustand, das Wort vollständig gelesen werden kann und der Automat in einen Endzustand gerät.

Um den Zusammenhang zwischen einer regulären Sprache und einem endlichen Automat, der diese Sprache erkennt, zu verdeutlichen, betrachten wir folgende durch das Quadrupel **(N, T, P, S)** gegebene Grammatik **G** vom Typ 3:

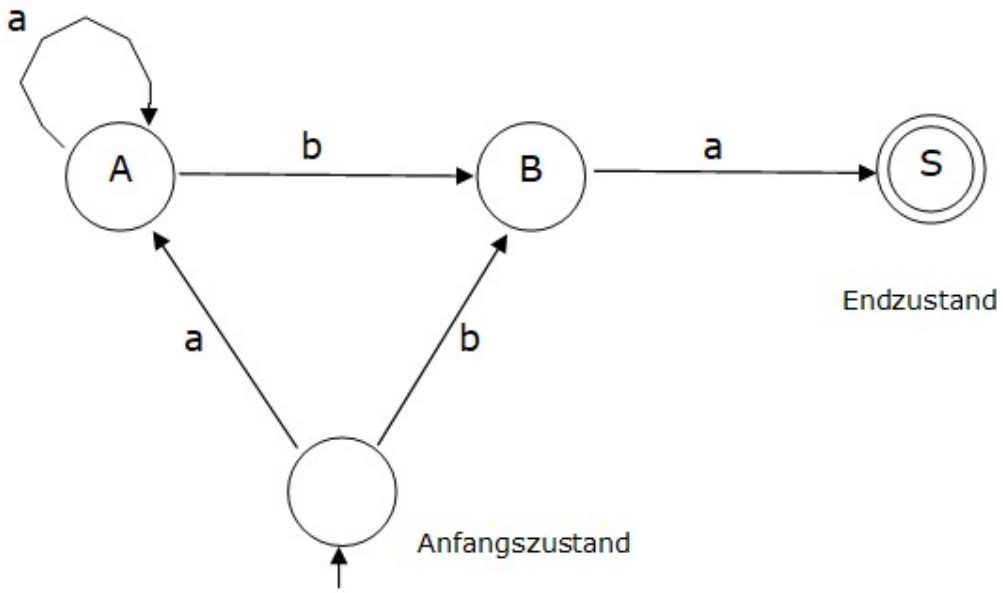
G = (N, T, P, S) mit

T := {a, b} (Eingabealphabet des endlichen Automaten)
N := {A, B, **S**} (Menge der Zustände mit **S** = Startzeichen = Endzustand)

Produktionen **P**:

(1) A → a | Aa
(2) B → b | Ab
(3) S → Ba

Der als Graph konzipierte endliche Automat **DFA** („**d**eterministic **f**inite **a**cceptor"), der zu dieser Grammatik **G** gehört:



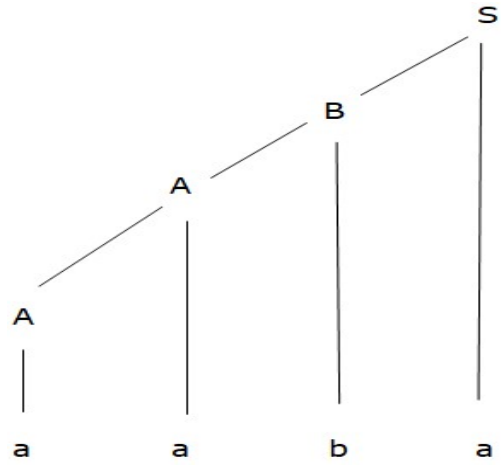
Zu dieser Grammatik **G** gehört offensichtlich die Sprache **L(G):**

L(G) = {ba, aba, aaba, aaaba, aaaaba, aaaaaba, ... }
= $\{w \mid w = a^n ba \text{ mit } n \in \mathbb{N}_0\}$ (lies: „die Menge aller Wörter w, für die gilt: $w = a^n ba$")

Linksableitung für das Wort **aaaaba** („top-down"):

S → Ba → Aba → Aaba → Aaaba → Aaaaba → aaaaaba

Syntaxbaum für das Wort **aaba** („bottom-up"):



Ein Beispiel für eine Sprache **L**, die nicht regulär ist und folglich von einem endlichen Automaten nicht erkannt wird:

Eingabe-Alphabet = **T** := {a, b}

L = {w | w = $a^n b^n$ mit n $\in$ **N** } = {ab, aabb, aaabbb, aaaabbbb, ... }

Daß wir gerade diese Sprache betrachten, hat folgenden Grund:

Interpretiert man a als öffnende, b als schließende Klammer, so stellt **L** die Menge der Klammerstrukturen beliebiger Tiefe dar. Solche Klammern treten nicht nur bei arithmetischen Ausdrücken auf, sondern auch bei allen blockorientierten Sprachen wie C++, Pascal, Java oder Python; in Pascal erfolgt die Klammerung eines Blocks mit ***begin*** und ***end***, in Java oder C++ mit geschweiften Klammern ***{*** und ***}***, in Python wird ein Anweisungsblock durch Einrücken gekennzeichnet.

Versuch, ein Regelsystem für eine die Sprache **L** beschreibende reguläre Grammatik zu finden:

N := {A, **S**}

Produktionen **P**:

(1) S $\rightarrow$ Sb | Ab
(2) A $\rightarrow$ Aa | a

Aufgabe:
a) Konstruiere den endlichen Automat Au, der zu dieser Grammatik gehört.
b) Zeige, daß auch die "falschen" Wörter $a^n b^m$ mit $n \neq m$ erkannt werden.
c) Gib ein Regelsystem (Produktionen) an, so daß die Sprache **L** erkannt wird. (Diese Sprache ist nicht *regulär*, sondern heißt *contextfrei* oder *vom Typ 2*.)

Lexikalische Analyse von Namen (Bezeichner, identifier)

Gegeben ist folgende Grammatik **G = (N, T, P, S)** mit

T := {a, b, . . . , z, A, B, . . . , Z, _, 0, 1, ... , 9}
N := { <name>, <buchstabe>, <ziffer> } ; Startzeichen: **S** = <name>

Produktionen **P**:

(1) <name> ::= <buchstabe> | <name> <buchstabe> | <name> <ziffer>
(2) <buchstabe> ::= a | b | c | . . . . . | z | _ | A | B | C | . . . . . | Z
(3) <ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Bemerkung: In Python wie in anderen Programmiersprachen sind Schlüsselwörter oder reservierte Wörter (z. B. except, if, elif, else, ...) als Bezeichner unzulässig.

Aufgabe:

a) Zeige: Die Grammatik **G** läßt sich als linkslineare Grammatik vom Typ 3 formulieren.

b) Zeichne den Graph des zu dieser Grammatik gehörenden endlichen Automaten, der Identifier erkennt.

c) Zeige, daß die Zeichenkette **a3Xyz** ein gültiger Bezeichner, also ein Wort der von der Grammatik **G** erzeugten Sprache **L(G)** ist, indem diese Zeichenkette (Programmtext oder hier: Teil eines Programmtextes) solange reduziert wird, bis die gesamte Zeichenkette auf das Startsymbol **S** zurückgeführt wurde (Syntaxbaum!).

d) Formuliere die Linksableitung für das Wort **a3Xyz** .

Lösungen:

zu a):

Für das Nonterminal <name>, welches auch Startzeichen ist, schreiben wir **S**; dann läßt sich die Grammatik **G = (N, T, P, S)**, welche Bezeichner in der Programmiersprache Python erzeugt, als linkslineare Grammatik formulieren:

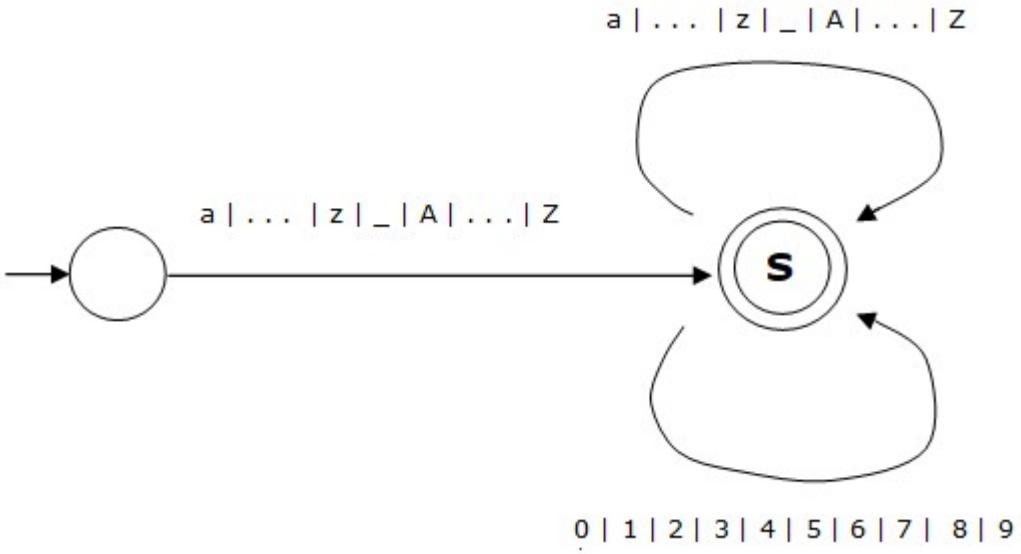
T := {a, b, c, . . . , z, _ , A, B, C, . . . , Z, 0, 1, 2, 3, ... , 9}
N := { **S** } mit **S** = Startsymbol

Produktionen **P:**

(1) **S** → a | b | . . . | z | _ | A | B | . . . | Z
(2) **S** → **S**a | **S**b | . . . . | **S**z | **S**_ | **S**A | **S**B | . . . | **S**Z
(3) **S** → **S**0 | **S**1 | **S**2 | . . . . | **S**9

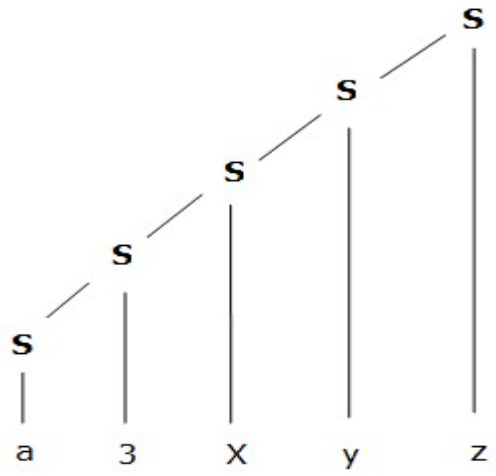
zu b):

Zu dieser Grammatik **G** gehört folgender **DFA**:



zu c):

Syntaxbaum



zu d): Linksableitung für das Wort a3Xyz („top-down"):

S $\overset{(2)}{\rightarrow}$ **S**z $\overset{(2)}{\rightarrow}$ **S**yz $\overset{(2)}{\rightarrow}$ **S**Xyz $\overset{(3)}{\rightarrow}$ **S**3Xyz $\overset{(1)}{\rightarrow}$ a3Xyz

Lokale Teilbereiche (hier: Syntax von Bezeichnern) einer Programmiersprache können durch eine reguläre Grammatik (Grammatik vom Typ 3) beschrieben werden.

Paritätsbit

Eine Folge von Bits wird um ein „Paritätsbit" ergänzt, so daß die Anzahl der mit 1 belegten Bits (einschließlich Paritätsbit) entweder gerade („gerade Parität") oder ungerade („ungerade Parität") ist.

Wir betrachten die Menge $\mathbf{T}^*$ aller Wörter, die sich über dem Alphabet $\mathbf{T} = \{0;1\}$ bilden lassen. Sämtliche Daten werden als binäre Worte über **T** auf einem Speichermedium abgelegt oder zwischen Komponenten eines Netzwerks transportiert.
Beispiel: Die 128 Zeichen der ASCII-Tabelle werden mit jeweils 7 Bit codiert, so daß das Paritätsbit als 8. Bit die Bitfolge zu einem aus 8 Bit bestehenden Byte ergänzen kann.

Um Fehler bei der Datenübertragung oder –speicherung zu erkennen, wird z. B. an jedes Wort ein „Paritätsbit" angehängt, so daß die Anzahl der Einsen im resultierenden Wort **w** gerade ist. Ein Wort **w** besteht also den Paritäts-Check, falls es zur Sprache

$$\mathbf{L(G)} = \{\mathbf{w} \in \mathbf{T}^* \mid \text{die Anzahl der Einsen in } \mathbf{w} \text{ ist gerade}\}$$

gehört.

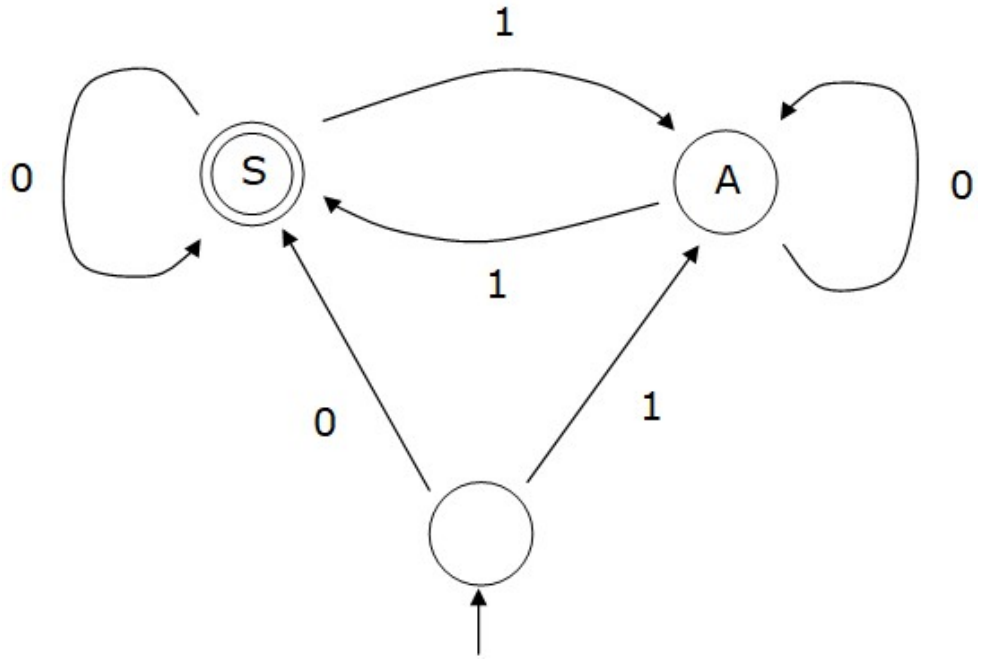
a) Konstruiere einen DFA, der **L(G)** erkennt.
b) Gib die Syntaxregeln (Produktionen **P**) an.
c) Zeige, daß es für das Wort 01101100 einen korrekten Syntaxbaum gibt.
d) Zeige: Für das Wort 01011 läßt sich weder eine Linksableitung noch ein korrekter Syntaxbaum angeben.

Lösungen:

zu a):

Menge der Terminalzeichen: **T** = {0;1}
Menge der Nonterminalzeichen: **N** = {A; **S**} mit **S** als Startzeichen

DFA:

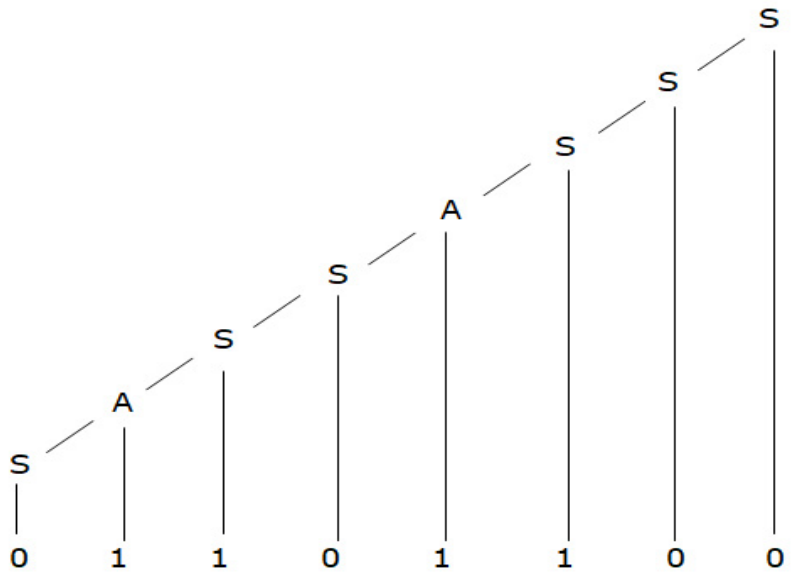


zu b):

Produktionen **P**:

(1) $A \rightarrow 1$
(2) $A \rightarrow A0$
(3) $A \rightarrow S1$
(4) $S \rightarrow 0$
(5) $S \rightarrow S0$
(6) $S \rightarrow A1$

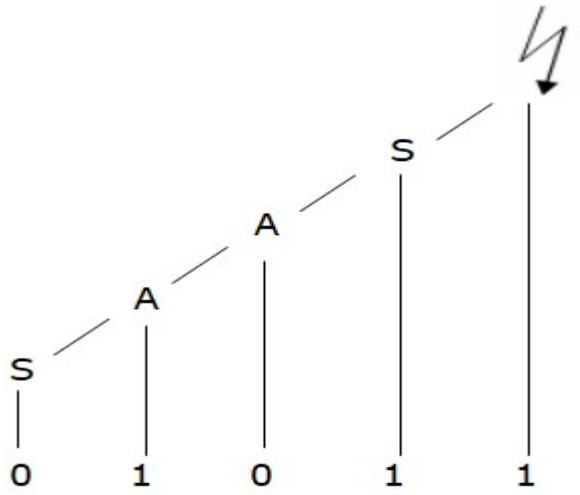
zu c):



zu d):
Linksableitung:

$$S \overset{6}{\rightarrow} A1 \overset{3}{\rightarrow} S11 \overset{5}{\rightarrow} S011 \overset{6}{\rightarrow} A1011 \overset{2}{\rightarrow} A01011 \rightarrow ?$$

Syntaxbaum:



Da das Wort 01011 sich auf das Startsymbol S nicht reduzieren läßt, gehört 01011 nicht zur Sprache **L(G)** mit **G = (N, T, P, S)**.

Wir oben (S. 6) bereits gezeigt, läßt sich die Sprache

L(G) = {ab, aabb, aaabbb, ... } = {w | w = $a^n b^n$ mit n $\in$ **N** } mit T = {a, b}

nicht durch eine reguläre Grammatik erzeugen; wenn man allerdings allgemeinere als reguläre Produktionsregeln zuläßt, gelingt die Formulierung einer Grammatik **G**, zu der **L(G)** gehört:

T = {a, b}; **N** = {**S**}; **S** = Startsymbol

P: (1) S $\rightarrow$ ab
(2) S $\rightarrow$ aSb

Übung: Gib eine Linksableitung und einen Syntaxbaum für das Wort aaabbb an!

KONTEXTFREIE GRAMMATIKEN UND KONTEXTFREIE SPRACHEN (TYP 2)

Context free grammars (CFG) and context free languages (CFL)

Zur Grammatik **G** = (**T**, **N**, **P**, **S**) mit Eingabealphabet **T** = {a; b; c} und **N** = {A; **S**} (**S**=Startsymbol) sei die Sprache

L(G) := {w | w = $a^n cb^n$, n = 0, 1, 2, . . . } = {c, acb, aacbb, aaacbbb, ... }

gegeben. Falls man versucht, zu dieser Sprache **L(G)** ein linksreguläre Grammatik mit den Produktionen **P**

(1) S → Sb | Ac | c
(2) A → Aa | a

zu formulieren (siehe auch Seite 6), erkennt man, daß der zugehörige DFA zwar die „richtigen" Wörter c, acb, aacbb, aaacbbb, ... erkennt und daß es korrekte Syntaxbäume für diese Wörter gibt, daß allerdings ebenso die „falschen" Wörter ac, acbb, aaaacbb, . . . (also $a^n cb^m$ mit $n \neq m$) erkannt werden; denn zum Abarbeiten der a bedarf es der rekursiven Regel A → Aa, zum Abarbeiten der b der rekursiven Regel S → Sb. Und der DFA ermöglicht nicht zu zählen und festzuhalten, wie oft diese rekursiven Regeln jeweils angewandt wurden!
Bemerkung: Eine Produktionsregel heißt rekursiv, wenn ein Nonterminal auf der linken Seite der Regel auch auf deren rechter Seite vorkommt.

Mehrfach geschachtelte Klammerungen, wie durch **L(G)** = {w | w = $a^n cb^n$ } beschrieben, treten nicht nur in arithmetischen Termen, sondern auch als Programmstruktur in allen blockorientierten Sprachen wie Python, Pascal, Java, C++ usw. auf. Um die oben formulierte Sprache **L(G)** zu erkennen, muß man das Regelsystem erweitern.

Vereinbarung:
Im Folgenden verstehen wir unter dem Symbol ω eine beliebige Aneinanderreihung von Terminals oder Nonterminals, z. B. ω = AbaSa.

DEFINITION:
Eine zur Grammatik G gehörende Sprache L(G) heißt kontextfrei (oder kontextunabhängig, engl.: contextfree) genau dann, wenn alle Produktionen die Form

A → ω (andere Schreibweise: **A ::= ω**)

mit A $\in$ N haben.

Bemerkungen:
Eine kontextfreie Grammatik nennen wir auch Grammatik vom Typ 2, die zugehörige kontextfreie Sprache heißt vom Typ 2.
Eine Produktion aAb ::= aBaS ist dagegen nicht kontextfrei in dem Sinne, daß man das Nonterminal A nicht einfach durch aBaS ersetzen darf, sondern nur dann, wenn es im Zusammenhang (im Kontext) mit einem voranstehenden a und einem folgenden b vorkommt; hier ist die Zeichenkette aAb durch aBaS zu ersetzen. Bei kontextfreien Sprachen steht das auf der linken Seite einer Produktion stehende Nonterminal in keinem Kontext anderer Zeichen.
Teilbereiche (z. B. Identifier, Paritäts-Check) einer Programmiersprache lassen sich durch eine reguläre Grammatik (siehe Seiten 6 - 9) beschreiben; insgesamt ist Python mindestens eine kontextfreie Programmiersprache, also vom Typ 2.
Natürliche Sprachen sind nicht kontextunabhängig (daß man den Satz "die Maus jagt die Katze" bilden konnte, liegt daran, daß die Produktionen kontextfrei definiert waren; solche semantisch unsinnigen Sätze kann man dadurch ausschließen, indem die Produktionen kontextabhängig formuliert werden.).

DEFINITION:

Zwei Grammatiken G und G' heißen äquivalent genau dann, wenn gilt:

$$\mathbf{L(G)} = \mathbf{L(G')}$$

Beispiel:

Definiere die Grammatiken **G** und **G'** wie folgt:

G = (**T**, **N**, **P**, **S**) mit Eingabealphabet **T** = {a; b; c}, **N** = {R; **S**}, **S**=Startsymbol und den Produktionen **P**

(1) S ::= c
(2) S ::= Rb
(3) R ::= aS

G' = (**T**, **N**, **P**, **S**) mit **T** = {a; b; c}, **N** = {**S**}, **S**=Startsymbol und den Produktionen **P**

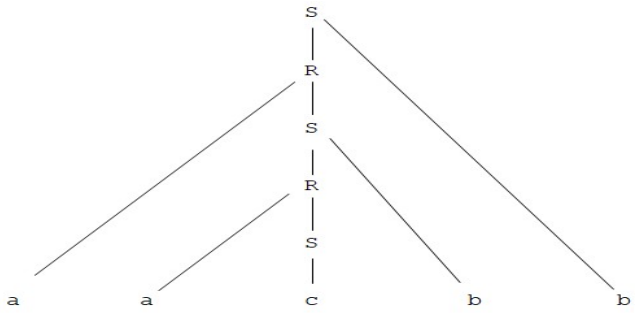
(1) S ::= c
(2) S ::= aSb (zentralrekursive Regel)

G und **G'** sind äquivalent, denn:

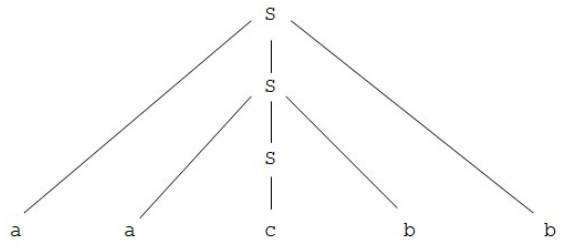
$$\mathbf{L(G)} = \mathbf{L(G')} = \{w \mid w = a^n c b^n , n = 0, 1, 2, \ldots\}$$

Beispiel: w = aacbb

Syntaxbaum gemäß Grammatik **G**:



Syntaxbaum gemäß Grammatik **G'**:



Offensichtlich lassen sich die Wörter $a^n c b^n$ sowohl in **G** als auch in **G'** auf das Startsymbol **S** reduzieren, indem man obenstehende Syntaxbäume jeweils geeignet erweitert.

DEFINITION:
Eine Grammatik G heißt strukturell mehrdeutig (structurally ambiguous) genau dann, wenn die zugehörige Sprache L(G) Wörter (die Wörter einer Programmiersprache sind die Quelltexte!) enthält, für die es strukturell unterschiedliche Syntaxbäume gibt.

Hinweis:
Von lexikalischer Mehrdeutigkeit spricht man, wenn ein Terminal (in einer natürlichen Sprache: ein Wort) mehrere Bedeutungen besitzt; Beispiel: In dem Satz „Das Schloß wurde im 16. Jahrhundert gebaut" kann mit dem Wort „Schloß" ein Gebäude oder eine Schließvorrichtung gemeint sein.

Weitere Beispiele für lexikalische Mehrdeutigkeit in natürlichen Sprachen:
"Der Gefangene floh" ↔ "Der gefangene Floh"
"time flies like an arrow" ↔ "fruit flies like a banana".

Bemerkungen:

- Es gibt kontextfreie Sprachen (CFLs; Sprachen vom Typ 2), die inhärent mehrdeutig (inherently ambiguous) sind, d. h. jede Grammatik für diese Sprache ist mehrdeutig. Eine kontextfreie Sprache heißt eindeutig, sobald sich eine eindeutige Grammatik angeben läßt, die diese Sprache erzeugt.
- Eine reguläre Sprache (Sprache vom Typ 3) kann nicht inhärent mehrdeutig sein, da sich stets eine eindeutige Grammatik angeben läßt, die diese Sprache erzeugt.
- Die Frage, ob zwei Grammatiken dieselbe Sprache erzeugen und damit äquivalent sind, ist allgemein nicht entscheidbar.
- Es ist grundsätzlich nicht möglich, für eine gegebene kontextfreie Grammatik mit einem allgemeinen Algorithmus zu entscheiden, ob sie eindeutig oder mehrdeutig ist.
- Gleichwohl gelingt es in der Praxis in aller Regel, eine eindeutige kontextfreie Grammatik zu formulieren (indem man z. B. die möglichen Fälle durchspielt).
- Syntaxbäume, bei denen das Startzeichen als Wurzel, die Nonterminals als innere Knoten und die Terminals als Endknoten (Blätter) auftreten, lassen sich nur bei Typ-3 oder Typ-2-Sprachen sinnvoll erstellen, also bei Sprachen, bei denen die „linke" Seite jeder Produktionsregel aus genau einem Nonterminal-Zeichen besteht.

Beispiele strukturell mehrdeutiger Grammatiken

Beispiel 1

Gegeben ist die Sprache **L(G)** zur Grammatik **G** = (**T**, **N**, **S**, **P**) mit
T = { +, *, (,), a, b, c, . . . , z }
N = { **S**, V }, **S** = Startzeichen

Produktionen **P**:

(1) S → V | (S) | S + S | S * S
(2) V → a | b | c | . . . | z

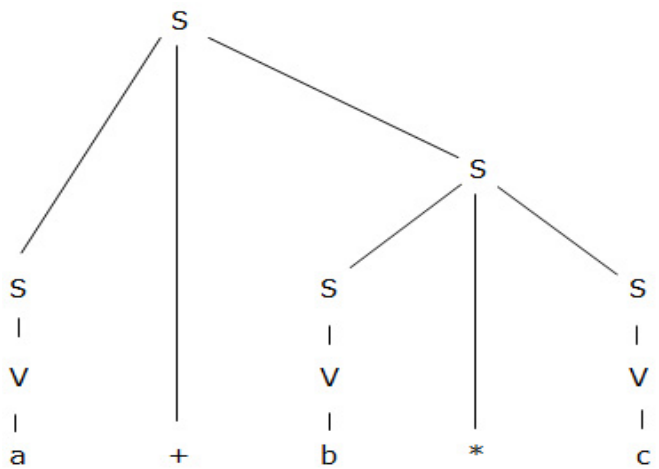
Zeige:

a) **(a + b) * c** ∈ **L(G)** (Linksableitung, Syntaxbaum)
b) Für das Wort **a + b * c** lassen sich Syntaxbäume auf zwei strukturell verschiedene Arten angeben! Erläutere die Konsequenzen für die Abfolge der Rechenschritte.

Lösung zu b):

1. Lösung:

Syntaxbaum:

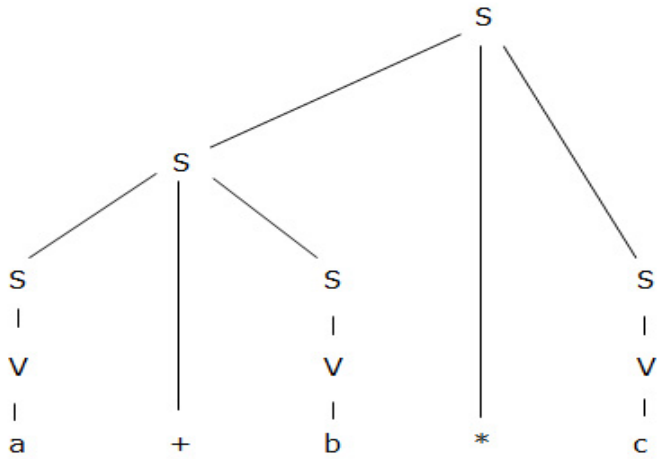


Linksableitung:

S → S + S → V + S → a + S → a + S * S → a + V * S → a + b * S
→ a + b * V → a + b * c

2. Lösung:

Syntaxbaum:



Linksableitung:

S → S * S → S + S * S → V + S * S → a + S * S → a + V * S → a + b * S
→ a + b * V → a + b * c

Der Term **a + b * c** wird gemäß dem ersten Syntaxbaum als Summe mit den Summanden **a** und **b * c**, gemäß dem zweiten Syntaxbaum als Produkt mit der Summe **a + b** als ersten Faktor und **c** als zweiten Faktor aufgefaßt.

Da sich zu dem Wort **a + b * c** zwei strukturell verschiedene Syntaxbäume in der Grammatik **G** angeben lassen, ist die Grammatik **G** strukturell mehrdeutig.

Beispiel 2 „dangling else" - ambiguity

Gegeben: Grammatik **G** = (**T**, **N**, **P**, **S**) mit

- **T** := { if, else, s1, s2, c1, c2 }
- **N** := { E, **S** } mit **S** = Startsymbol
- Produktionsregeln **P**:

  (1) S → if E S
  (2) S → if E S else S
  (3) S → s1 | s2
  (4) E → c1 | c2

Bedeutung der Terminals:
s1, s2 (statement1, statement2) stehen jeweils für Anweisungen oder Anweisungsblöcke
c1, c2 (condition1, condition2) stehen jeweils für Boolesche Terme

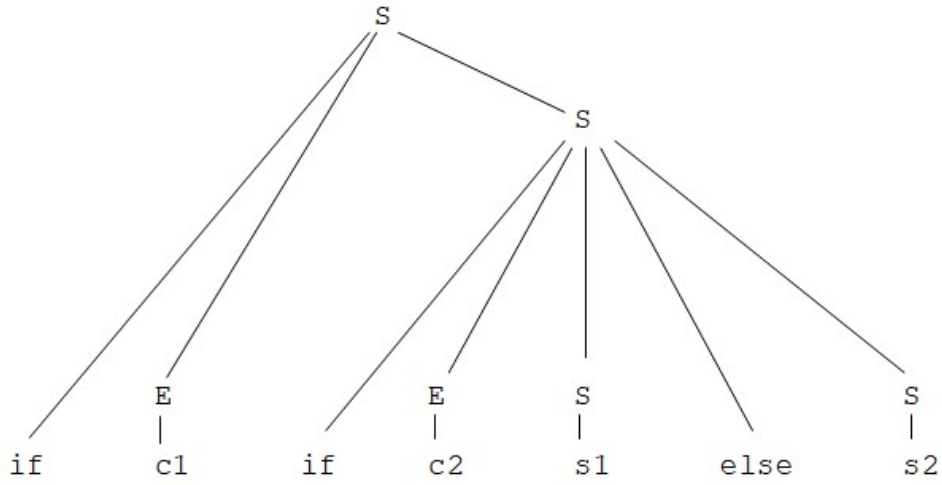
Zeige:

Das Wort

`if c1 if c2 s1 else s2`

läßt sich auf zwei strukturell verschiedene Weisen auf das Startsymbol S reduzieren.

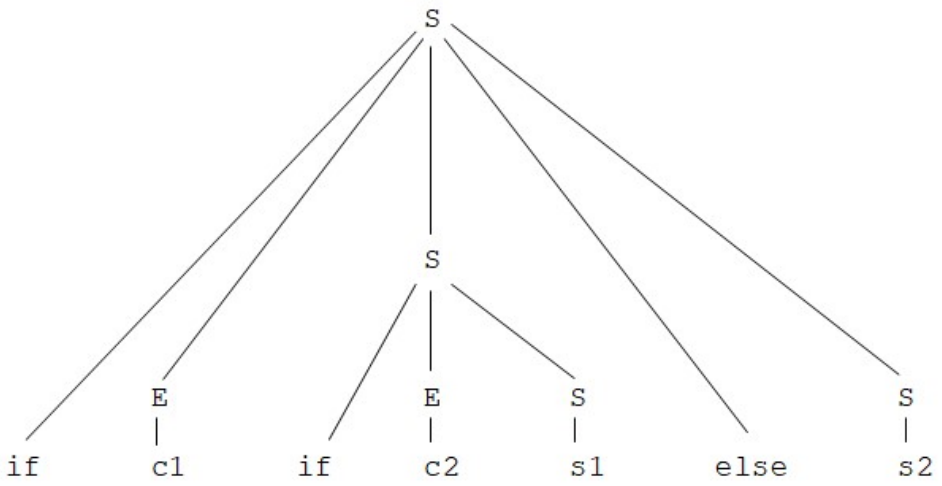
Syntaxbaum 1:



Formulierung in Python:

```
if c1:
    if c2:
        s1
    else:
        s2
```

Syntaxbaum 2:



Formulierung in Python:

```
if c1:
    if c2:
        s1
else:
    s2
```

Möglichkeiten, um der Mehrdeutigkeit zu begegnen:

- *Der else-Zweig bezieht sich immer auf das nächst voranstehende if.*
- *Kennzeichnung von Anweisungsblöcken durch entsprechende Strukturierung des Quelltextes;*
  in Python: durch Einrücken;
  in Pascal: mit den Schlüsselwörtern **begin** *und* **end***;*
  in C++, Java: mit **{** *und* **}** *.*

Beispiel 3 Mehrdeutigkeit bei einer Grammatik für eine natürliche Sprache

Gegeben ist die Sprache **L(G)** zur Grammatik **G** = (**T**, **N**, **Satz**, **P**) mit

T = {mit, in, auf, Hans, Frau, Fernglas, Park, sieht, geht, der, die, das, einem}
N = {**Satz**, NP, VP, PP, N, A, V, P} mit **Satz**=Startsymbol; NP ↔ Nominalphrase usw.

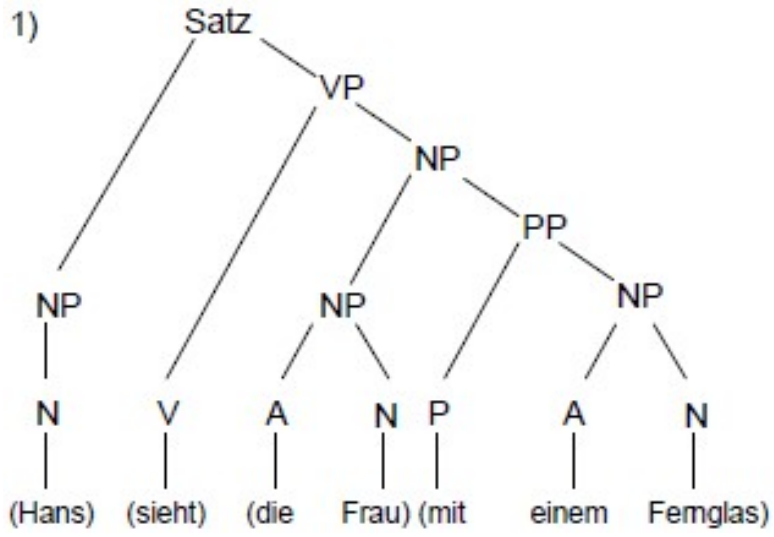
Produktionen **P**:

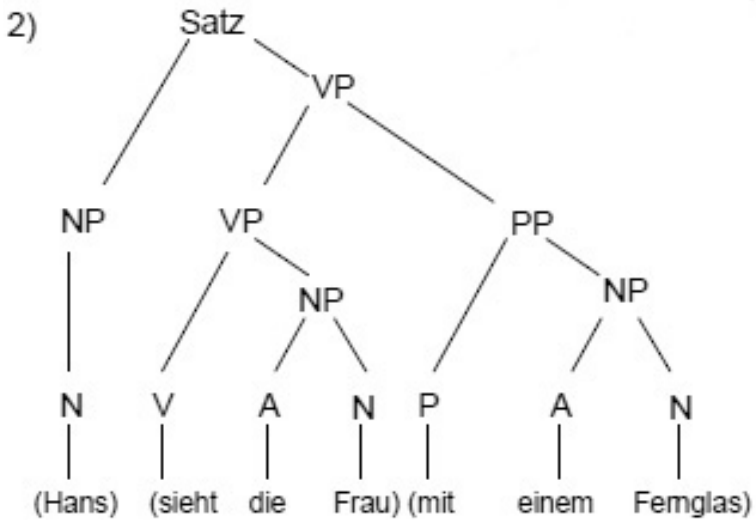
(1) Satz → NP VP
(2) NP → NP PP | A N | N
(3) VP → VP PP | V NP | V
(4) PP → P NP
(5) P → mit | in | auf
(6) N → Hans | Frau | Fernglas | Park
(7) V → sieht | geht
(8) A → der | die | das | einem

Zu dem Satz

„**Hans sieht die Frau mit einem Fernglas**"

lassen sich zwei strukturell verschiedene Syntaxbäume angeben:





Beachte: Die Klammern sind nicht Bestandteil des zu analysierenden Satzes, sondern dienen dazu, die unterschiedliche Semantik zu verdeutlichen.

Dagegen erweist die folgende Grammatik sich als eindeutig:

Gegeben ist die Grammatik **G** = (**T**, **N**, **P**, **S**), bestehend aus der Menge **T** der Terminalzeichen, der Menge **N** der Nonterminalzeichen, dem Element **S** $\in$ **N** als Startzeichen und der Menge **P** der Produktionen:

T := { a, b, p, q, if, then, else}

N := {**S**, S_1, S_2, B, T}

Produktionen **P**:

(1) S $\rightarrow$ S_1 | S_2

(2) S_1 $\rightarrow$ T | if B then S_1 else S_2

(3) S_2 $\rightarrow$ T | if B then S | if B then S_1 else S_2

(4) B $\rightarrow$ p | q

(5) T $\rightarrow$ a | b

Man überzeuge sich: Das Wort **`if p then if q then a else b`**
besitzt in dieser Grammatik nur einen einzigen Syntaxbaum!

Bedeutung der Terminals:
p, q bezeichnen Boolesche Terme;
a, b stehen für Anweisungen oder Anweisungsblöcke.

Wir betrachten im Folgenden Grammatiken $\mathbf{G_1}$ und $\mathbf{G_2}$, deren Sprachen $\mathbf{L(G_1)}$ und $\mathbf{L(G_2)}$ aus arithmetischen Termen bestehen und die wegen $\mathbf{L(G_1) = L(G_2)}$ äquivalent sind:

Gegeben ist die Menge der Terminalzeichen **T** = {a, b, c, d, (,), +, *}.
Wir definieren die folgenden Grammatiken $\mathbf{G_1}$ und $\mathbf{G_2}$:

$\mathbf{G_1}$ = **(T, N, P, S)** mit **N** = {V, R, Q, **S**}, **S**=Startsymbol
Produktionen **P**:

(1) V $\rightarrow$ a | b | c | d
(2) Q $\rightarrow$ R
(3) Q $\rightarrow$ Q * R
(4) R $\rightarrow$ V
(5) R $\rightarrow$ (S)
(6) S $\rightarrow$ Q
(7) S $\rightarrow$ S + Q

$\mathbf{G_2}$ = **(T, N, P, S)** mit **N** = {V, **S**} , **S**=Startsymbol
Produktionen **P**:

(1) V $\rightarrow$ a | b | c | d
(2) S $\rightarrow$ V
(3) S $\rightarrow$ V * S | S * S
(4) S $\rightarrow$ V + S | S + S
(5) S $\rightarrow$ (S)

a) Zeige: Das Wort **a + b * (c + d)** gehört sowohl zur Sprache $\mathbf{L(G_1)}$ als auch zur Sprache $\mathbf{L(G_2)}$, indem man bei $\mathbf{G_1}$ und $\mathbf{G_2}$ jeweils einen Syntaxbaum und eine Linksableitung angibt.

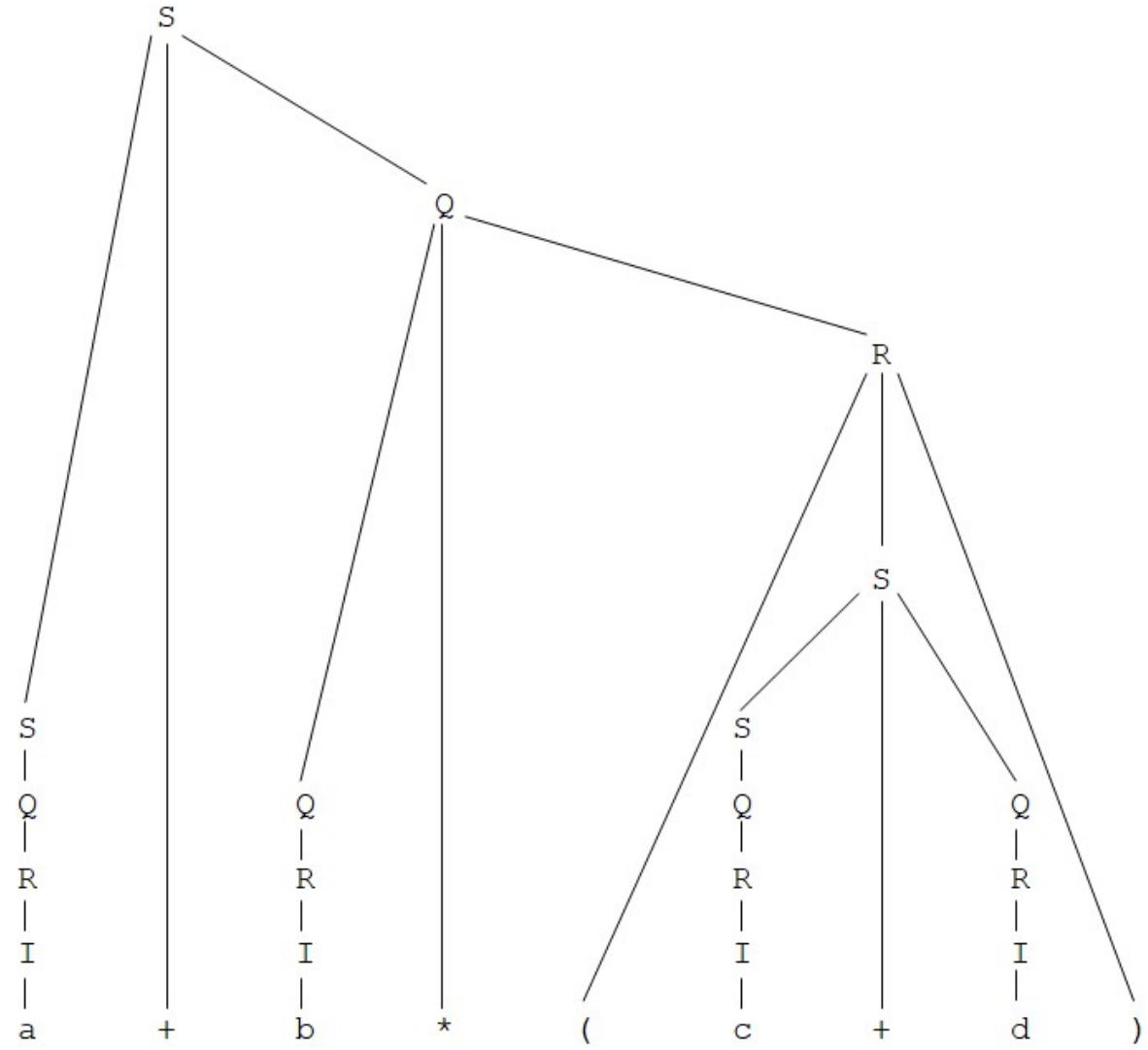
b) Analysiere das Wort **a * b + a * c** sowohl nach $\mathbf{G_1}$ als auch nach $\mathbf{G_2}$ (Linksableitung, Syntaxbaum).
Welche der Grammatiken $\mathbf{G_1}$ und $\mathbf{G_2}$ verdient den Vorzug, obwohl sie äquivalent sind (Begründung!)?

c) Analysiere das Wort **a * (b + c)** sowohl nach $\mathbf{G_1}$ als auch nach $\mathbf{G_2}$.

Lösungen:

a) α) **a + b * (c + d) $\in$ L($\mathbf{G_1}$)**

Syntaxbaum:



Linksableitung:
Beachte: das am „weitesten links" stehende Nonterminal wird jeweils ersetzt.

S → S + Q → Q + Q → R + Q → I + Q → a + Q → a + Q * R

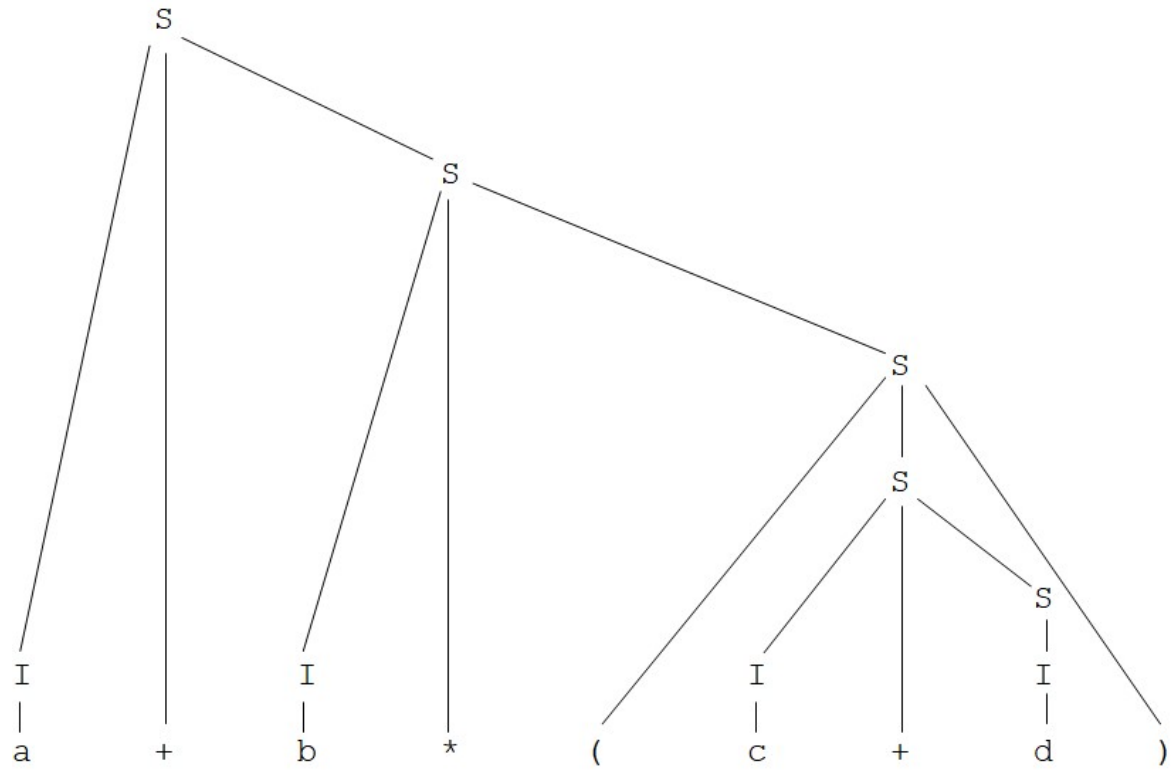
→ a + R * R → a + I * R → a + b * R → a + b * (S) → a + b * (S + Q)

→ a + b * (Q + Q) → a + b * (R + Q) → a + b * (I + Q) → a + b * (c + Q)

→ a + b * (c + R) → a + b * (c + I) → a + b * (c + d)

a) β) **a + b * (c + d) $\in$ L($\mathbf{G_2}$)**

Syntaxbaum:



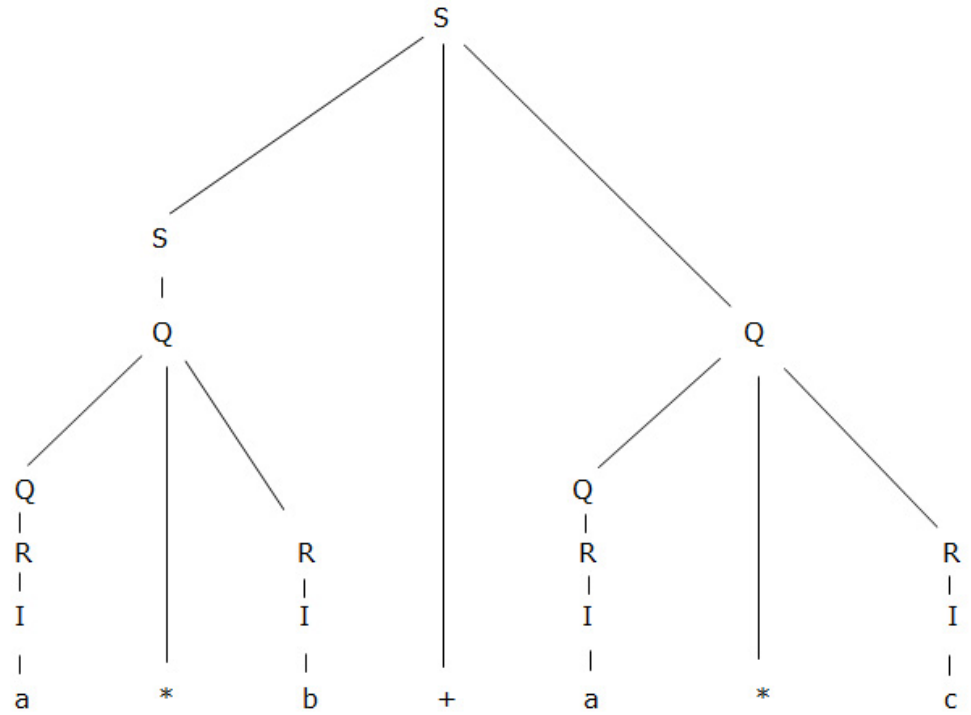
Linksableitung:

$S \rightarrow I + S \rightarrow a + S \rightarrow a + I * S \rightarrow a + b * S \rightarrow a + b * (S) \rightarrow a + b * (I + S)$

$\rightarrow a + b * (c + S) \rightarrow a + b * (c + I) \rightarrow a + b * (c + d)$

b) α) **a * b + a * c $\in$ $L(G_1)$**

Syntaxbaum:



Linksableitung:

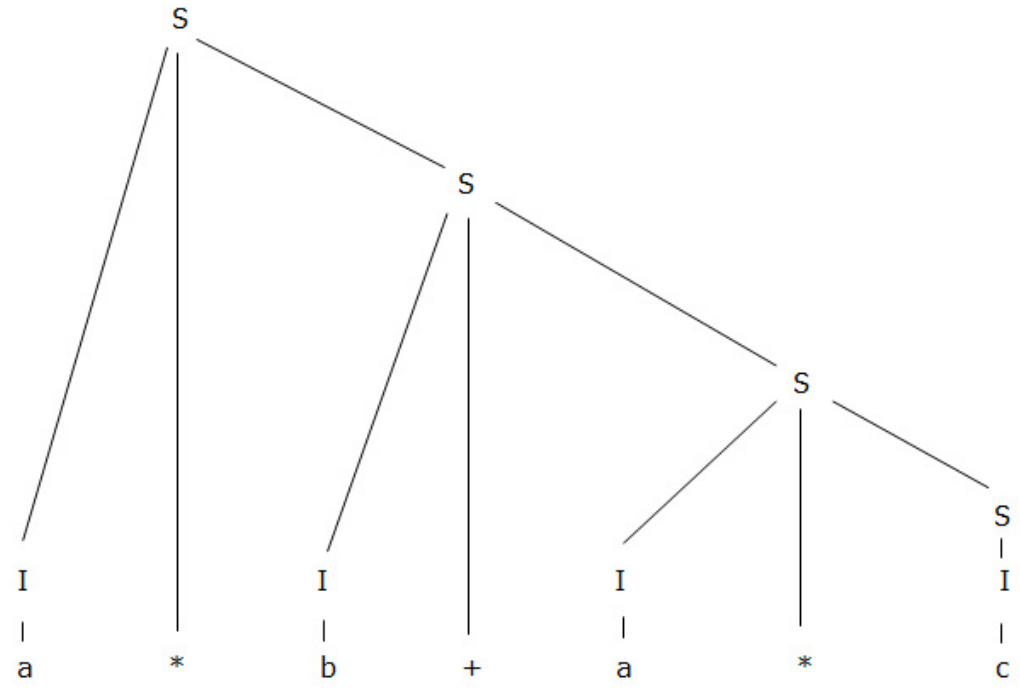
$S \rightarrow S + Q \rightarrow Q + Q \rightarrow Q * R + Q \rightarrow R * R + Q \rightarrow I * R + Q \rightarrow a * R + Q$

$\rightarrow a * I + Q \rightarrow a * b + Q \rightarrow a * b + Q * R \rightarrow a * b + R * R \rightarrow a * b + I * R$

$\rightarrow a * b + a * R \rightarrow a * b + a * I \rightarrow a * b + a * c$

b) β) **a * b + a * c** $\in$ **$L(G_2)$**

Syntaxbaum:



Linksableitung:

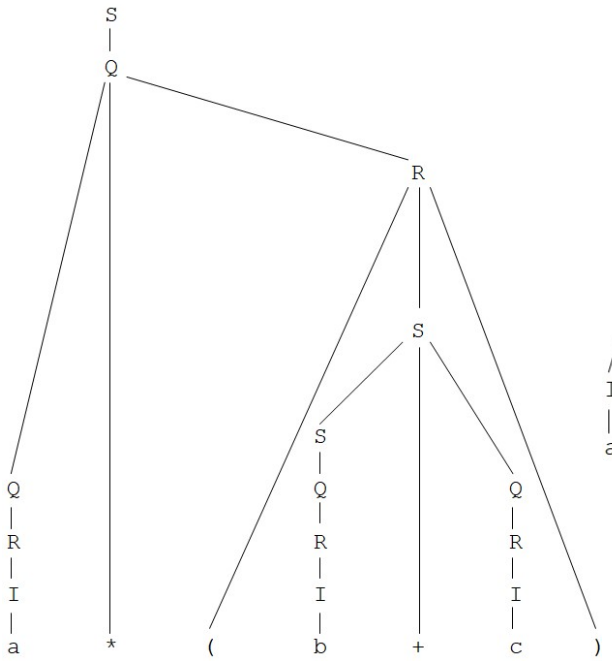
$S \rightarrow I * S \rightarrow a * S \rightarrow a * I + S \rightarrow a * b + S \rightarrow a * b + I * S$

$\rightarrow a * b + a * S \rightarrow a * b + a * I \rightarrow a * b + a * c$

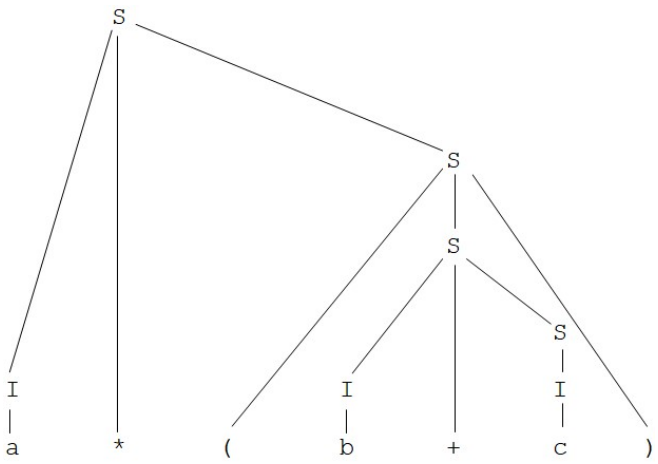
Der Term **a * b + a * c** wird in der Grammatik **G_1** als Summe, deren Summanden jeweils die Produkte **a * b** und **a * c** sind, verstanden; dagegen faßt die Grammatik **G_2** den Term **a * b + a * c** als Produkt mit den Faktoren **a** und **b + a * c** auf und beachtet folglich nicht die allgemeingültige Vereinbarung „Punkt vor Strich". Daher ist die „kompliziertere" Grammatik **G_1** der „einfacheren" Grammatik **G_2** vorzuziehen, obwohl beide Grammatiken **G_1** und **G_2** äquivalent sind, denn **$L(G_1) = L(G_2)$**.

c) Syntaxbäume zum Wort **a * (b + c)**

gemäß Grammatik $\mathbf{G_1}$:



gemäß Grammatik $\mathbf{G_2}$:



KONTEXTSENSITIVE GRAMMATIKEN UND KONTEXTSENSITIVE SPRACHEN (TYP 1)

Rückblick:

*Eine Grammatik **G** und die zugehörige Sprache **L(G)** heißen **kontextfrei** oder **vom Typ 2** genau dann, wenn die linke Seite jeder Produktionsregel aus genau einem Nonterminal-Symbol und die rechte Seite aus einer beliebigen Aneinanderreihung von Terminal- und Nonterminal-Symbolen besteht.*

Beachte:
*Die Syntaxanalyse („bottom-up") eines zur **kontextfreien Sprache L(G)** gehörenden Wortes **w** (z. B. arithmetischer Term, Quelltext einer Programmiersprache) läßt sich mittels eines Syntaxbaums realisieren, dessen Wurzel das Startsymbol **S** ist, dessen innere Knoten aus Nonterminals und dessen Endknoten („Blätter") aus Terminals bestehen.*
*Somit gehört ein Wort **w** zur Sprache **L(G)**, wenn es sich unter Anwendung der Produktionsregeln auf das Startsymbol **S** reduzieren läßt.*

Folgende Grammatik **G** sei geben durch das Quadrupel **(T; N; S; P)**:

Menge der Terminalsymbole: **T** := {a, b, c}

Menge der Nonterminalsymbole: **N** := {B, C, **S**} mit **S** = Startsymbol

Produktionen **P**:

(1) S $\rightarrow$ aSBC | aBC

(2) CB $\rightarrow$ BC

(3) aB $\rightarrow$ ab

(4) bB $\rightarrow$ bb

(5) bC $\rightarrow$ bc

(6) cC $\rightarrow$ cc

Die zu dieser Grammatik **G** gehörende Sprache ist

L(G) = { w | w = $a^n b^n c^n$, n $\in \mathbb{N}$ } = { abc, aabbcc, aaabbbccc, aaaabbbbcccc, ... } .

Die Grammatik **G** ist nicht kontextfrei im Sinne der Definition auf Seite 10; vielmehr verlangen die Regeln (2) bis (6), daß die Nonterminals auf der linken Seite nur dann ersetzt werden können und auch zu ersetzen sind, wenn sie in einem bestimmten Kontext mit anderen Zeichen (Terminals oder Nonterminals) stehen. Die Ersetzungsregeln (2) bis (6) sind folglich kontextsensitiv.

Hinweis:
*Zu dieser Sprache **L(G)** läßt sich keine kontextfreie Grammatik (Grammatik vom Typ 2) angeben. Die oben definierte Grammatik **G** ist **kontextsensitiv** (Grammatik vom Typ 1).*

Auf die exakte Definition einer Typ-1- und einer Typ-0-Grammatik verzichten wir an dieser Stelle.

Aufgabe: Verifiziere jeweils durch eine Linksableitung, daß die Worte

a) **abc**
b) **aabbcc**
c) **aaabbbccc**

zu **L(G)** gehören.

Lösung zu c):

$$S \overset{1}{\rightarrow} aSBC \overset{1}{\rightarrow} aaSBCBC \overset{1}{\rightarrow} aaaBCBCBC \overset{2}{\rightarrow} aaaBBCCBC \overset{2}{\rightarrow} aaaBBCBCC$$

$$\overset{2}{\rightarrow} aaaBBBCCC \overset{3}{\rightarrow} aaabBBCCC \overset{4}{\rightarrow} aaabbBCCC \overset{4}{\rightarrow} aaabbbCCC \overset{5}{\rightarrow} aaabbbcCC$$

$$\overset{6}{\rightarrow} aaabbbccC \overset{6}{\rightarrow} aaabbbccc$$

Es erhebt sich die Frage, von welchem Typ natürliche Sprachen sind. Folgende Beispiele erhellen, daß neben höheren Programmiersprachen (Pascal, C++, Python, Java) auch natürliche Sprachen mindestens kontextfrei, also mindestens vom Typ 2 sind:

Beispiel 1:

Ein Schüler, der die Qualifikation Block I, für die 35 Kurse, von denen höchstens sieben mit weniger als 5 Punkten bewertet wurden, gemäß §10 (1)-(8) einzubringen sind, erreicht hat, wird zur mündlichen Prüfung zugelassen.

Die Struktur dieses Satzes wird durch eine geeignete Formatierung des Textes deutlich:

```
Ein Schüler,                                                    wird zur mdl. Prüfung zugelassen.
   der die Qualifikation Block I,                                                  erreicht hat,
       für die 35 Kurse,                                gemäß §10(1)-(8) einzubringen sind,
          von denen höchstens sieben mit weniger als 5 Punkten     bewertet wurden,
```

Damit hat dieser Satz eine Syntax, die dem Regelsystem der Grammatik auf Seite 9 unten bzw. Seite 10 entspricht (hier: 4 mal „Klammer auf", gefolgt von genau 4 mal „Klammer zu") und der folglich eine kontextfreie Grammatik (Typ 2) zugrunde liegt.

Bemerkung:
Daß ein mit 0 Punkten bewerteter Kurs nicht eingebracht werden kann, wird in obenstehendem Beispielsatz nicht erwähnt, ergibt sich aber aus § 10 (8) AbiPrO, auf den der Satz Bezug nimmt.

Beispiel 2:

Das Mädchen, das den Hund, der die Katze, die schnurrte, biß, sah, weinte.

Die Sätze aus diesen Beispielen sind syntaktisch korrekt gebildet; dennoch werden in der Praxis solche vierfachen Verschachtelungen gemieden, dreifache kommen kaum vor, zweifache dagegen sind durchaus üblich:

Dreifache Verschachtelung:

Der Schüler, der die Qualifikation Block I, für die er mindestens 200 Punkte benötigt, erreicht hat, wird zur mündlichen Prüfung zugelassen.

Zweifache Verschachtelung:

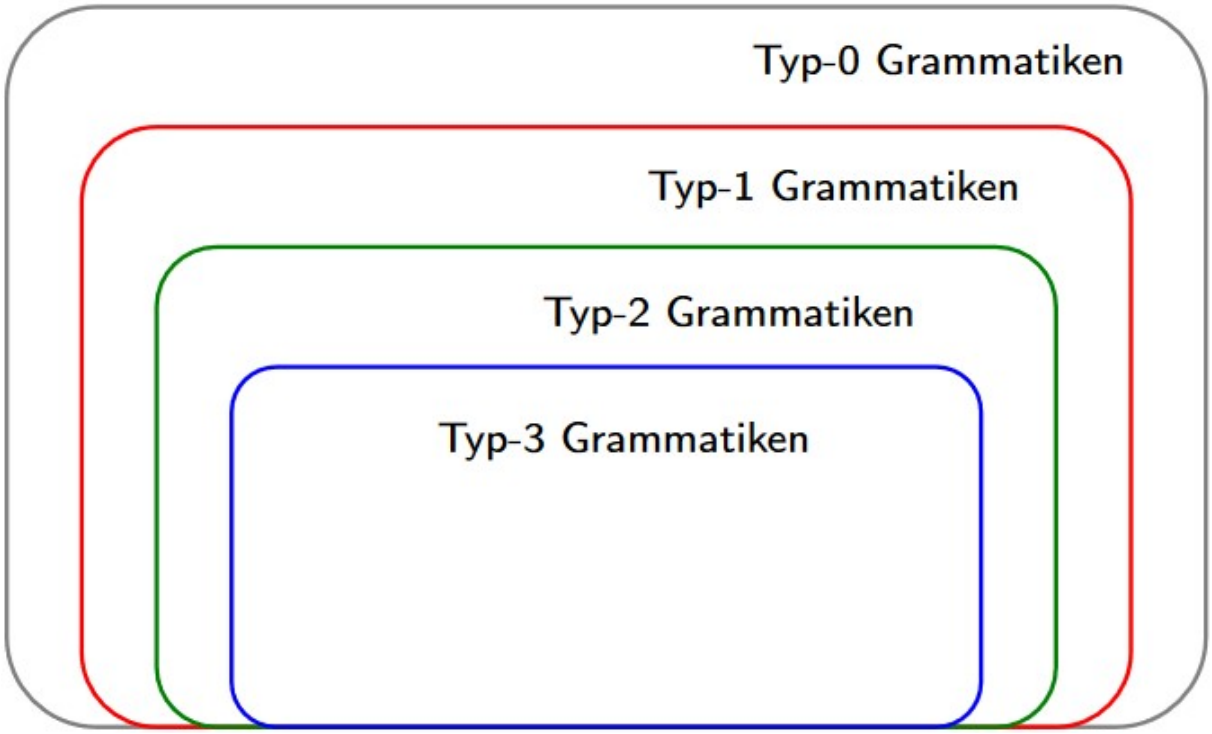
Der Schüler, der die Qualifikation Block I erreicht hat, wird zur mündlichen Prüfung zugelassen.

Wenn man solcher eher komplexen grammatikalischen Strukturen vom Typ 2 nicht mächtig ist, wird man den Inhalt des Beispiels 2 auch folgendermaßen formulieren können:

Das Mädchen weinte, das den Hund sah, der die schnurrende Katze biß.

Seit NOAM CHOMSKY (* 07.12.1928; o. Professor am MIT (Massachusetts Institute of Technology)) grundlegende Arbeiten zur Klassifizierung formaler Sprachen (Typ 3 ↔ regulär, Typ 2 ↔ kontextfrei, Typ 1 ↔ kontextsensitiv, Typ 0 ↔ rekursiv-aufzählbar) verfaßt hat, ist man der Auffassung, daß natürliche Sprachen mindestens die Komplexität einer kontextsensitiven Sprache aufweisen. Allerdings ist zu vermuten, daß kontextsensitive grammatikalische Konstruktionen in der Praxis eher gemieden werden, was sogar für kontextfreie Konstruktionen gilt (siehe obige Beispiele).

Hierarchie der Grammatiken nach Noam Chomsky:



Typ-3 Grammatiken bilden eine echte Teilmenge der Typ-2 Grammatiken usw.