

Kapitel 2: Methoden zur Beschreibung von Syntax

Prof. Dr. David Sabel

Lehr- und Forschungseinheit für Theoretische Informatik

Institut für Informatik, LMU München

WS 2018/19

Stand der Folien: 26. Oktober 2018



Übersicht: Inhalt und Ziele

Zwei Methoden zur Beschreibung (Definition) der
Syntax von Programmiersprachen

- Backus-Naur-Form **BNF** und deren Erweiterung **EBNF**
- Syntax-Diagramme



Peter Naur
1928-2016

Mitwirkung bei ALGOL 60
2005 ACM Turing Award



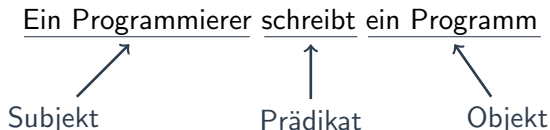
John Backus
1924-2007

Entwicklung von FORTRAN
1977 ACM Turing Award

Bildnachweise: (1) The original uploader was Erik tj at English Wikipedia. (https://commons.wikimedia.org/wiki/File:Peter_naur.JPG), Peter_naur, <https://creativecommons.org/licenses/by-sa/3.0/legalcode>
(2) Plerre.Lescanne (https://commons.wikimedia.org/wiki/File:John_Backus.jpg), <https://creativecommons.org/licenses/by-sa/4.0/legalcode>

Aufbau von Programmen

- Java-Programme lassen sich ähnlich wie natürlichsprachliche Texte in verschiedenartige Einzelteile zerlegen
- Natürliche Sprache



- Java-Programm

```
public class Test {  
    public static void main(String[] args){  
        System.out.print("Hal");  
        System.out.println("lo!");  
    }  
}
```

Methodenname Stringliteral

Anweisungen } Methoden
-definition

- Der **Aufbau von Programmen** ist durch die Syntax der Programmiersprache festgelegt.
- **Syntax einer Programmiersprache:**
System von Regeln, das festlegt,
 - aus welchen Einzelteilen Programme bestehen können,
 - wie man diese Einzelteile textuell aufschreibt und
 - wie man sie zu einem Programm kombinieren kann.
- **Unterschied zu natürlicher Sprache:**
Die Syntax von Programmiersprachen ist vollständig und präzise festgelegt.

Syntax von Programmiersprachen (2)

- Seit ALGOL 60 wird die Syntax von Programmiersprachen präzise durch **formale Grammatiken** beschrieben.

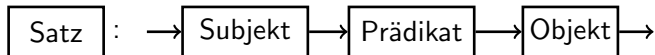
Der Compiler benutzt die Grammatik der Programmiersprache, um ein gegebenes Programm auf Richtigkeit zu überprüfen.

- Die Compiler-Phase, in der das geschieht, nennt man **Syntaxanalyse** oder Parsen.
- Im Fehlerfall tritt ein „**Syntax Error**“ (auch „Parse Error“ genannt) auf.

BNF und Syntaxdiagramme: Grundideen

Beispiel für eine Grammatikregel:

- Als Syntaxdiagramm:



- In BNF-Form:

$\langle \text{Satz} \rangle ::= \langle \text{Subjekt} \rangle \langle \text{Prädikat} \rangle \langle \text{Objekt} \rangle$

- In EBNF-Form:

$\text{Satz} = \text{Subjekt Prädikat Objekt}$

Das Syntaxdiagramm und die (E)BNF (Backus-Naur-Form) erlauben das Bilden syntaktisch korrekter Sätze:

- Ein Programmierer schreibt ein Programm

Ein Programmierer ← Subjekt
schreibt ← Prädikat
ein Programm ← Objekt

BNF und Syntaxdiagramme: Grundideen (2)

Weiterer Beispielsatz:

- Ein Programm schreibt einen Programmierer

Ein Programm	← Subjekt
schreibt	← Prädikat
einen Programmierer	← Objekt

Dieser Satz ist zwar **semantischer Unsinn**, aber **syntaktisch korrekt**.

Der Compiler benutzt die Grammatik um ein gegebenes Programm auf **syntaktische** Richtigkeit zu überprüfen

Backus-Naur-Form

- Die **Backus-Naur-Form (BNF)** wurde erstmals zur Beschreibung der Syntax von Algol 60 verwendet.
- Die BNF ist eine **Notation für Grammatiken**, die vor allem für die Beschreibung von Programmiersprachen verwendet wird.
- Heute ist die BNF (in notationellen Varianten) die **Standardbeschreibungstechnik** für Programmiersprachen und andere strukturierte Texte.
- Wir verwenden in der Vorlesung die **„Erweiterte Backus-Naur-Form“ (EBNF)** (eingeführt zur Beschreibung von PL1, 60er Jahre).
- Auch die **Syntax von Java** ist in einer Variante der Backus-Naur-Form beschrieben.
- Wichtige Begriffe: **Symbole, Regeln, Grammatik, Ableitung**

In Grammatiken (BNF / EBNF) kommen drei Arten von **Symbolen** vor:

- Nichtterminalsymbole
- Terminalsymbole
- Operatorsymbole

Nichtterminalsymbole:

Symbole, **die noch ersetzt werden**, und daher durch **Regeln** definiert werden.

- In der BNF werden Nichtterminalsymbole gekennzeichnet, indem sie durch $\langle \dots \rangle$ umschlossen werden.
- In der EBNF gibt es keine besondere Kennzeichnung der Nichtterminalsymbole.

Beispiele:

- in BNF: $\langle \text{Digit} \rangle$, $\langle \text{Sign} \rangle$
- in EBNF: Digit, Sign

Terminalsymbole:

Zeichen oder Folgen von Zeichen, die **genau so** in der zu definierenden Sprache **vorkommen**.

- In der BNF werden Terminalsymbole nicht gekennzeichnet.
- In der EBNF werden Terminalsymbole gekennzeichnet, indem sie durch doppelte Anführungszeichen umschlossen werden.

Beispiele:

- in BNF: 0, 1, class
- in EBNF: "0", "1", "class"

Operatorsymbole:

- In der **BNF** gibt es
 - das Operatorsymbol **|** für den **Auswahloperator**
- In der **EBNF** gibt es:
 - Operatorsymbol **|** für den **Auswahloperator**
 - Operatorsymbol **[]** für den **Optionsoperator**
 - Operatorsymbol **{ }** für den **Wiederholungsoperator**

Regeln in EBNF

Jede **EBNF-Regel** (oder auch **Produktion/Produktionsregel**) hat die Form

$$\text{Nichtterminalsymbol} = \text{Ausdruck}$$

Ein *Ausdruck* ist entweder

- ein Terminalsymbol oder
- ein Nichtterminalsymbol oder
- ein **zusammengesetzter Ausdruck**.

Regeln in EBNF

Jede **EBNF-Regel** (oder auch **Produktion/Produktionsregel**) hat die Form

$$\text{Nichtterminalsymbol} = \text{Ausdruck}$$

Ein *Ausdruck* ist entweder

- ein Terminalsymbol oder
- ein Nichtterminalsymbol oder
- ein **zusammengesetzter Ausdruck**.

Aus gegebenen Ausdrücken E , E_1 und E_2 können mit den Operatorsymbolen **zusammengesetzte Ausdrücke** gebildet werden:

- **Sequentielle Komposition** $E_1 E_2$ („ E_1 gefolgt von E_2 “)

Regeln in EBNF

Jede **EBNF-Regel** (oder auch **Produktion/Produktionsregel**) hat die Form

$$\text{Nichtterminalsymbol} = \text{Ausdruck}$$

Ein *Ausdruck* ist entweder

- ein Terminalsymbol oder
- ein Nichtterminalsymbol oder
- ein **zusammengesetzter Ausdruck**.

Aus gegebenen Ausdrücken E , E_1 und E_2 können mit den Operatorsymbolen **zusammengesetzte Ausdrücke** gebildet werden:

- **Sequentielle Komposition** $E_1 E_2$ („ E_1 gefolgt von E_2 “)
- **Auswahl** $E_1 \mid E_2$ („ E_1 oder E_2 “)

Regeln in EBNF

Jede **EBNF-Regel** (oder auch **Produktion/Produktionsregel**) hat die Form

$$\text{Nichtterminalsymbol} = \text{Ausdruck}$$

Ein *Ausdruck* ist entweder

- ein Terminalsymbol oder
- ein Nichtterminalsymbol oder
- ein **zusammengesetzter Ausdruck**.

Aus gegebenen Ausdrücken E , E_1 und E_2 können mit den Operatorsymbolen **zusammengesetzte Ausdrücke** gebildet werden:

- **Sequentielle Komposition** $E_1 E_2$ („ E_1 gefolgt von E_2 “)
- **Auswahl** $E_1 \mid E_2$ („ E_1 oder E_2 “)
- **Option** $[E]$ („ E darf weggelassen werden“)

Regeln in EBNF

Jede **EBNF-Regel** (oder auch **Produktion/Produktionsregel**) hat die Form

$$\text{Nichtterminalsymbol} = \text{Ausdruck}$$

Ein *Ausdruck* ist entweder

- ein Terminalsymbol oder
- ein Nichtterminalsymbol oder
- ein **zusammengesetzter Ausdruck**.

Aus gegebenen Ausdrücken E , E_1 und E_2 können mit den Operatorsymbolen **zusammengesetzte Ausdrücke** gebildet werden:

- **Sequentielle Komposition** $E_1 E_2$ („ E_1 gefolgt von E_2 “)
- **Auswahl** $E_1 \mid E_2$ („ E_1 oder E_2 “)
- **Option** $[E]$ („ E darf weggelassen werden“)
- **Wiederholung** $\{E\}$
(„ E kann 0-mal oder mehrmals hintereinander vorkommen“)

Eine **Grammatik** besteht aus

- einer Menge von Regeln für jedes Nichtterminal sowie
- einem Startsymbol (Nichtterminalsymbol)

Ein **Wort** ist eine Folge von Terminalzeichen.

Eine Grammatik G definiert eine Menge von Worten,
genannt: **Die Sprache von G** (Definition folgt noch)

Beispiele für EBNF-Grammatiken (1)

Grammatik für Ziffern (mit Startsymbol Digit)

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Digit = "0" | NonZeroDigit

Beispiele für EBNF-Grammatiken (1)

Grammatik für Ziffern (mit Startsymbol Digit)

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"



EBNF-Regel

Digit = "0" | NonZeroDigit

Beispiele für EBNF-Grammatiken (1)

Grammatik für Ziffern (mit Startsymbol Digit)

Nichtterminalsymbol

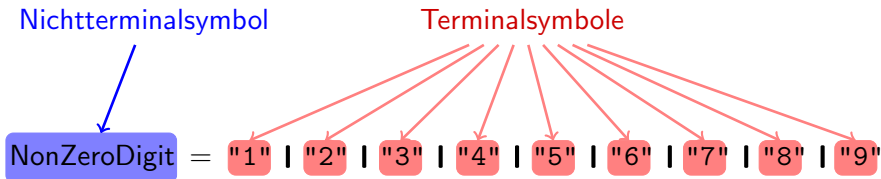
 NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

EBNF-Regel

Digit = "0" | NonZeroDigit

Beispiele für EBNF-Grammatiken (1)

Grammatik für Ziffern (mit Startsymbol Digit)

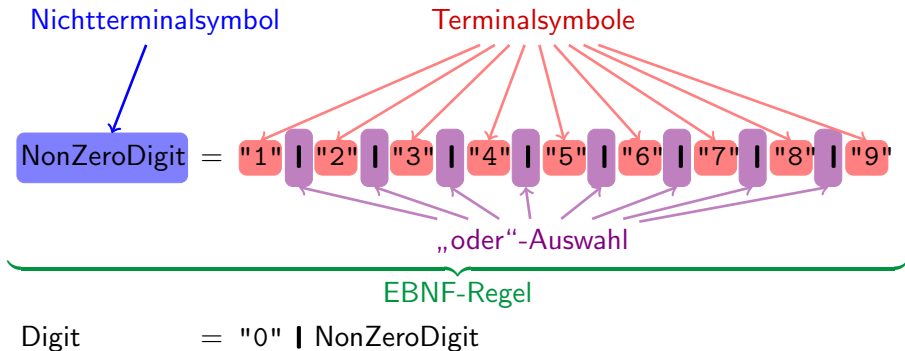


EBNF-Regel

Digit = "0" | NonZeroDigit

Beispiele für EBNF-Grammatiken (1)

Grammatik für Ziffern (mit Startsymbol Digit)



Beispiele für EBNF-Grammatiken (2)

Grammatik für **ganze Zahlen**, wobei **IntegerValue** das Startsymbol ist:

IntegerValue = [Sign] DecimalNumeral

Sign = "+" | "-"

DecimalNumeral = "0" | NonZeroDigit [Digits]

Digits = Digit { Digit }

Informell: Eine ganze Zahl besteht aus einer nichtleeren Folge von Ziffern ohne führende Null, eventuell mit vorangestelltem Vorzeichen.

Beispiele für EBNF-Grammatiken (2)

Grammatik für **ganze Zahlen**, wobei `IntegerValue` das Startsymbol ist:

`IntegerValue` = `[Sign] DecimalNumeral`

`Sign` = `"+" | "-"`

`DecimalNumeral` = `"0" | NonZeroDigit [Digits]`

`Digits` = `Digit { Digit }`

Informell: Eine ganze Zahl besteht aus einer nichtleeren Folge von Ziffern ohne führende Null, eventuell mit vorangestelltem Vorzeichen.

Andere mögliche Varianten für die Regel für `Digits` ohne den Wiederholungsoperator zu verwenden: (alle verwenden **Rekursion**):

- `Digits` = `Digit [Digits]` oder `Digits` = `[Digits] Digit`

- Ohne Optionsoperator:

`Digits` = `Digit | Digit Digits` oder `Digits` = `Digit | Digits Digit`

Wie wendet man Regeln an?

Ist $+42$ eine ganze Zahl?

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue →

[Sign] DecimalNumeral

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumeral

Ausführen des Operators [] → Sign DecimalNumeral

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumeral

Ausführen des Operators [] → Sign DecimalNumeral

Regel für Sign → ("+" | "-") DecimalNumeral

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumerical

Ausführen des Operators [] → Sign DecimalNumerical

Regel für Sign → ("+" | "-") DecimalNumerical

Ausführen des Operators | → "+" DecimalNumerical

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumerical

Ausführen des Operators [] → Sign DecimalNumerical

Regel für Sign → ("+" | "-") DecimalNumerical

Ausführen des Operators | → "+" DecimalNumerical

Regel für DecimalNumerical → "+" ("0" | NonZeroDigit [Digits])

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumeral

Ausführen des Operators [] → Sign DecimalNumeral

Regel für Sign → ("+" | "-") DecimalNumeral

Ausführen des Operators | → "+" DecimalNumeral

Regel für DecimalNumeral → "+" ("0" | NonZeroDigit [Digits])

Ausführen des Operators | → "+" NonZeroDigit [Digits]

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumeral

Ausführen des Operators [] → Sign DecimalNumeral

Regel für Sign → ("+" | "-") DecimalNumeral

Ausführen des Operators | → "+" DecimalNumeral

Regel für DecimalNumeral → "+" ("0" | NonZeroDigit [Digits])

Ausführen des Operators | → "+" NonZeroDigit [Digits]

Regel für NonZeroDigit → "+" ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") [Digits]

Wie wendet man Regeln an?

Ist +42 eine ganze Zahl?

Wir bilden folgende Ableitung:

IntegerValue

Regel für IntegerValue → [Sign] DecimalNumeral

Ausführen des Operators [] → Sign DecimalNumeral

Regel für Sign → ("+" | "-") DecimalNumeral

Ausführen des Operators | → "+" DecimalNumeral

Regel für DecimalNumeral → "+" ("0" | NonZeroDigit [Digits])

Ausführen des Operators | → "+" NonZeroDigit [Digits]

Regel für NonZeroDigit → "+" ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") [Digits]

Ausführen des Operators | → "+" "4" [Digits]

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators []
→

"+" "4" Digits

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators [] →

"+" "4" **Digits**

Regel für Digits →

"+" "4" Digit { Digit }

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators [] →

"+" "4" Digits

Regel für Digits →

"+" "4" **Digit** { Digit }

Regel für Digit →

"+" "4" ("0" | NonZeroDigit) { Digit }

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators [] →

"+" "4" Digits

Regel für Digits →

"+" "4" Digit { Digit }

Regel für Digit →

"+" "4" ("0" | NonZeroDigit) { Digit }

Ausführen des Operators | →

"+" "4" NonZeroDigit { Digit }

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators [] →

"+" "4" Digits

Regel für Digits →

"+" "4" Digit { Digit }

Regel für Digit →

"+" "4" ("0" | NonZeroDigit) { Digit }

Ausführen des Operators | →

"+" "4" NonZeroDigit { Digit }

Regel für NonZeroDigit →

"+" "4" ("1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9") { Digit }

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators [] →

"+" "4" Digits

Regel für Digits →

"+" "4" Digit { Digit }

Regel für Digit →

"+" "4" ("0" | NonZeroDigit) { Digit }

Ausführen des Operators |

"+" "4" NonZeroDigit { Digit }

Regel für NonZeroDigit →

"+" "4" ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") { Digit }

Ausführen des Operators |

"+" "4" "2" { Digit }

Wie wendet man Regeln an? (2)

"+" "4" [Digits]

Ausführen des Operators [] →

"+" "4" Digits

Regel für Digits →

"+" "4" Digit { Digit }

Regel für Digit →

"+" "4" ("0" | NonZeroDigit) { Digit }

Ausführen des Operators | →

"+" "4" NonZeroDigit { Digit }

Regel für NonZeroDigit →

"+" "4" ("1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9") { Digit }

Ausführen des Operators | →

"+" "4" "2" { Digit }

Ausführen des Operators { } →

"+" "4" "2"

Ableitung von Worten

Ein Wort w kann vom Startsymbol der Grammatik G **abgeleitet** werden, falls es eine **Ableitung** der Form $E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_k$ gibt, wobei

- E_0 ist das **Startsymbol** der Grammatik G
- E_k ist identisch zum **Wort** w
- $E_i \rightarrow E_{i+1}$ entsteht **entweder** durch
 1. Ersetzung eines oder mehrerer Nichtterminale in E_i durch die rechte Seite ihrer Regeln, **oder**
 2. durch Ausführung von (einem oder mehreren) Operator(en), d.h.
 - $[E]$ darf gelöscht werden oder durch E ersetzt werden
 - $E \mid F$ darf durch E oder durch F ersetzt werden
 - $\{E\}$ darf gelöscht werden oder durch $E\{E\}$ ersetzt werden.

Kurze Ableitung

Häufig wird Schritt 1) und Schritt 2) in einem Schritt zusammengefasst. Man spricht dann von einer **kurzen Ableitung**.

- $E_i \rightarrow E_{i+1}$ entsteht **entweder** durch
 1. Ersetzung eines oder mehrerer Nichtterminale in E_i durch die rechte Seite ihrer Regeln, **oder**
 2. durch Ausführung von (einem oder mehreren) Operator(en)

Kurze Ableitung

Häufig wird Schritt 1) und Schritt 2) in einem Schritt zusammengefasst. Man spricht dann von einer **kurzen Ableitung**.

- $E_i \rightarrow E_{i+1}$ entsteht **entweder** durch
 1. Ersetzung eines oder mehrerer Nichtterminale in E_i durch die rechte Seite ihrer Regeln, **oder** **und** / **oder**
 2. durch Ausführung von (einem oder mehreren) Operator(en)

Kurze Ableitung

Häufig wird Schritt 1) und Schritt 2) in einem Schritt zusammengefasst. Man spricht dann von einer **kurzen Ableitung**.

- $E_i \rightarrow E_{i+1}$ entsteht durch
 1. Ersetzung eines oder mehrerer Nichtterminale in E_i durch die rechte Seite ihrer Regeln, **und / oder**
 2. durch Ausführung von (einem oder mehreren) Operator(en)sodass mindestens ein Nichtterminal ersetzt oder ein Operator ausgeführt wurde.

Kurze Ableitung

Häufig wird Schritt 1) und Schritt 2) in einem Schritt zusammengefasst. Man spricht dann von einer **kurzen Ableitung**.

- $E_i \rightarrow E_{i+1}$ entsteht durch
 1. Ersetzung eines oder mehrerer Nichtterminale in E_i durch die rechte Seite ihrer Regeln, **und / oder**
 2. durch Ausführung von (einem oder mehreren) Operator(en) **sodass mindestens ein Nichtterminal ersetzt oder ein Operator ausgeführt wurde.**

„Darf“-Konvention: Bei einer kurzen Ableitung dürfen Schritt 1) und 2) zusammengefasst werden (müssen aber nicht).

Folgerung: Jede lange Ableitung ist auch eine kurze
(Die Umkehrung gilt nicht!)

Beispiele (1)

Betrachte die folgende Grammatik mit Startsymbol PEx:

PEx = NonZeroDigit | PEx "+" PEx

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Eine (lange) Ableitung von 1+2+3:

PEx

Beispiele (1)

Betrachte die folgende Grammatik mit Startsymbol PEx:

PEx = NonZeroDigit | PEx "+" PEx

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Eine (lange) Ableitung von 1+2+3:

PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" PEx

Beispiele (1)

Betrachte die folgende Grammatik mit Startsymbol PEx:

PEx = NonZeroDigit | PEx "+" PEx

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Eine (lange) Ableitung von 1+2+3:

PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" (NonZeroDigit | PEx "+" PEx)

Beispiele (1)

Betrachte die folgende Grammatik mit Startsymbol PEx:

PEx = NonZeroDigit | PEx "+" PEx

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Eine (lange) Ableitung von 1+2+3:

PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" (NonZeroDigit | PEx "+" PEx)

Ausführung von | (an 2 Stellen) →

PEx "+" (PEx "+" PEx)

Beispiele (1)

Betrachte die folgende Grammatik mit Startsymbol PEx:

PEx = NonZeroDigit | PEx "+" PEx

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Eine (lange) Ableitung von 1+2+3:

PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" (NonZeroDigit | PEx "+" PEx)

Ausführung von | (an 2 Stellen) →

PEx "+" (PEx "+" PEx)

Ersetzung von PEx (an 2 Vorkommen) →

(NonZeroDigit | PEx "+" PEx) "+" (PEx "+" (NonZeroDigit | PEx "+" PEx))

Beispiele (1)

Betrachte die folgende Grammatik mit Startsymbol PEx:

PEx = NonZeroDigit | PEx "+" PEx

NonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Eine (lange) Ableitung von 1+2+3:

PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" PEx

Ersetzung von PEx →

NonZeroDigit | PEx "+" (NonZeroDigit | PEx "+" PEx)

Ausführung von | (an 2 Stellen) →

PEx "+" (PEx "+" PEx)

Ersetzung von PEx (an 2 Vorkommen) →

(NonZeroDigit | PEx "+" PEx) "+" (PEx "+" (NonZeroDigit | PEx "+" PEx))

Ausführung von | (an 2 Stellen) →

NonZeroDigit "+" (PEx "+" NonZeroDigit)

Beispiele (2)

$PEx = \text{NonZeroDigit} \mid PEx \text{ "+" } PEx$

$\text{NonZeroDigit} = \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \text{"4"} \mid \text{"5"} \mid \text{"6"} \mid \text{"7"} \mid \text{"8"} \mid \text{"9"}$

Fortsetzung der (langen) Ableitung von $1+2+3$:

$\text{NonZeroDigit "+" (PEx "+" NonZeroDigit)}$

Ersetzung von NonZeroDigit ($2\times$) und PEx



$(\text{"1"} \mid \dots \mid \text{"9"}) \text{ "+" } ((\text{NonZeroDigit} \mid PEx \text{ "+" } PEx) \text{ "+" } (\text{"1"} \mid \dots \mid \text{"9"}))$

Ausführung von \mid (mehrere Stellen)

→ $\text{"1"} \text{ "+" } \text{NonZeroDigit} \text{ "+" } \text{"3"}$

Ersetzung von NonZeroDigit

→ $\text{"1"} \text{ "+" } (\text{"1"} \mid \dots \mid \text{"9"}) \text{ "+" } \text{"3"}$

Ausführung von \mid

→ $\text{"1"} \text{ "+" } \text{"2"} \text{ "+" } \text{"3"}$

Beispiele (3)

$PE_x = \text{NonZeroDigit} \mid PE_x "+" PE_x$

$\text{NonZeroDigit} = "1" \mid "2" \mid "3" \mid "4" \mid "5" \mid "6" \mid "7" \mid "8" \mid "9"$

Eine kurze Ableitung von $1+2+3$:

Ersetzung von PE_x und
Ausführung von \mid

→

PE_x

Ersetzung von PE_x ($2\times$)
und Ausführung von \mid ($2\times$)

→

$PE_x "+" PE_x$

Ersetzung von NonZeroDigit und
 PE_x ($2\times$) Ausführung von \mid (mehrfach)

→

$\text{NonZeroDigit} "+" PE_x "+" PE_x$

Ersetzung von NonZeroDigit ($2\times$)
und Ausführung von \mid (mehrfach)

→

$"1" "+" \text{NonZeroDigit} "+" \text{NonZeroDigit}$

$"1" "+" "2" "+" "3"$

Die Sprache einer Grammatik

Jede Grammatik G definiert eine Menge von Wörtern, die als **Sprache von G** bezeichnet wird.

- Wir schreiben $L(G)$ für die Sprache von G . (L steht für Language.)
- Die Sprache $L(G)$ besteht genau aus den Wörtern, die vom Startsymbol der Grammatik **abgeleitet** werden können:

$$L(G) = \{w \mid S \rightarrow \dots \rightarrow w, S \text{ Startsymbol von } G, w \text{ ein Wort}\}$$

Beispiele

Sei G die folgende Grammatik mit Startsymbol Start :

$\text{Start} = \text{"a" Mitte "c"}$

$\text{Mitte} = \{ \text{"b"} \}$

$L(G) =$

Sei G die folgende Grammatik mit Startsymbol Start :

$$\text{Start} = \text{"a" Mitte "c"}$$
$$\text{Mitte} = \{ \text{"b"} \}$$

$L(G)$ = Menge aller Worte, die mit a anfangen,
dann beliebig viele b's folgen und schließlich mit c enden

Sei G die folgende Grammatik mit Startsymbol Brackets:

Brackets = Brackets Brackets
| "(" ")"
| "[" "]"
| "(" Brackets ")"
| "[" Brackets "]"

$L(G) =$

Sei G die folgende Grammatik mit Startsymbol Brackets:

Brackets = Brackets Brackets
| "(" ")"
| "[" "]"
| "(" Brackets ")"
| "[" Brackets "]"

$L(G)$ = Menge aller korrekt geklammerten Klammersausdrücke
aus runden und eckigen Klammern

EBNF-Grammatik für Bezeichner

- Ein **Bezeichner** (Identifier) ist eine nichtleere Folge von Buchstaben oder Ziffern, beginnend mit einem Buchstaben.
- Bezeichner sind z.B. A, A200D3, Muenchen
- Keine Bezeichner sind 007, *O.K.*, 1A
- EBNF-Grammatik (mit Startsymbol Identifier)

Letter = "A" | ... | "Z" | "a" | ... | "z"
Identifier = Letter { Letter | Digit }
Digit = ... (wie vorher)

- In Java müssen alle Variablennamen, Klassennamen usw. Bezeichner sein. Die Grammatik in Java ist jedoch etwas allgemeiner als oben

EBNF-Variante für Java

- Die Java-Sprachdefinition verwendet eine andere Variante der EBNF.
<https://docs.oracle.com/javase/specs/jls/se8/html/jls-2.html#jls-2.4>

Notation dort:

- Nichtterminalsymbole sind *kursiv*
- Terminalsymbole sind in **Schreibmaschinenschrift**
(z.B. `if` statt `"if"`)
- Regeln werden als $N : E$ (anstelle von $N = E$) geschrieben
- Das Auswahlsymbol, d.h. `|` wird weggelassen und durch einen Zeilenumbruch ersetzt, oder man verwendet **(one of)**
- In **älteren** Spezifikationen: Die Option wird durch tiefgestelltes **opt** gekennzeichnet: E_{opt} statt $[E]$.

Beispiele aus der Java-Spezifikation

DecimalNumeral :

0

NonZeroDigit [*Digits*]

Digits :

Digit

Digits Digit

Digit :

0

NonZeroDigit

NonZeroDigit :

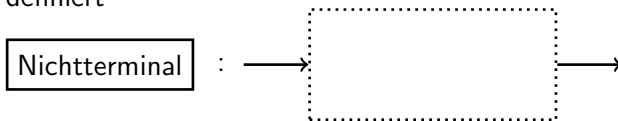
(one of)

1 2 3 4 5 6 7 8 9

Syntaxdiagramme

Ein Syntaxdiagramm ist ein einfacher grafischer Formalismus zur Definition von Sprachen. Er besteht aus

- Rechtecken, in denen Nichtterminale stehen
- Ovalen (oder Rechtecke mit abgerundeten Ecken) in den Terminale stehen
- Pfeilen, die die Elemente verbinden,
- Genau einem Eingangs- und einem Ausgangspfeil
- Je ein Nichtterminalsymbol wird durch je ein Syntaxdiagramm definiert

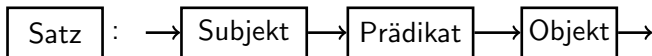


Beispiele für Syntaxdiagramme

EBNF:

Satz = Subjekt Prädikat Objekt

Syntax-Diagramm:



(Syntaxdiagramme für Subjekt, Prädikat, Objekt müssen noch hinzugefügt werden)

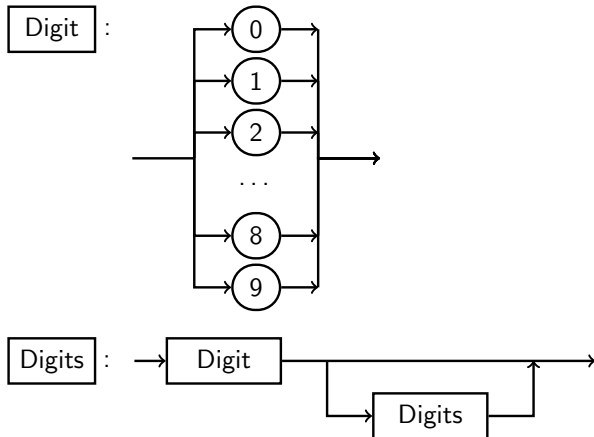
Beispiele für Syntaxdiagramme (2)

EBNF:

Digit = "0" | "1" | "2" | ... | "9"

Digits = Digit [Digits]

Syntaxdiagramm:



Vom Syntaxdiagramm definierte Sprache

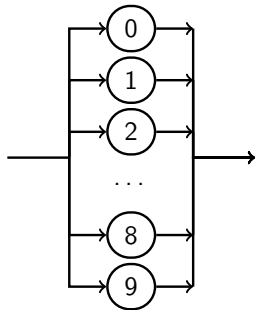
Ein Wort w liegt genau dann, in der durch ein Syntaxdiagramm definierten Sprache, wenn man das Syntaxdiagramm vom Eingangs- zum Ausgangspfeil „entlang“ w durchlaufen kann:

- Alle durchlaufenen Knoten \textcircled{T} für Terminalsymbole geben aneinander gereiht (in der Durchlaufreihenfolge) das Wort w .
- Durchlauf durch einen Knoten \boxed{N} für Nichtterminalsymbol N bedeutet Durchlauf des Syntaxdiagramms für N .
- Der durchlaufene Weg hat endliche Länge.

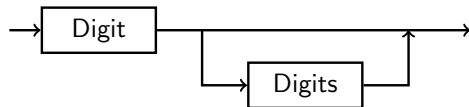
Beispiel

Wird 1019 durch das Syntaxdiagramm für Digits erkannt?

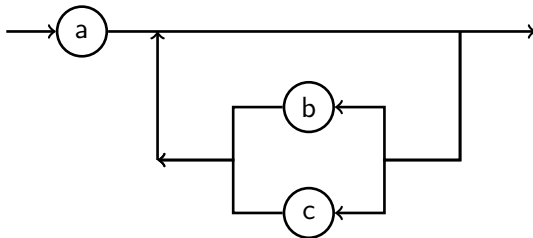
Digit :



Digits :



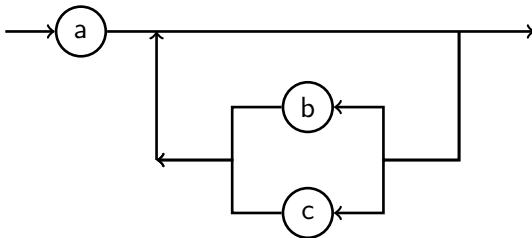
Noch ein Beispiel



Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb
- a
- aabbcc
- abccb
- bcbc
- bcabc

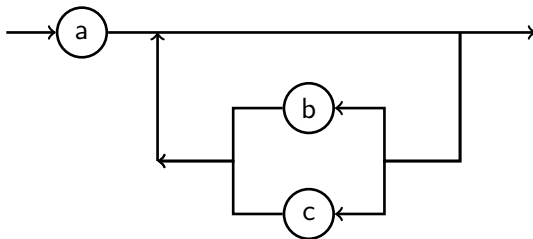
Noch ein Beispiel



Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a
- aabbcc
- abccb
- bcbc
- bcabc

Noch ein Beispiel

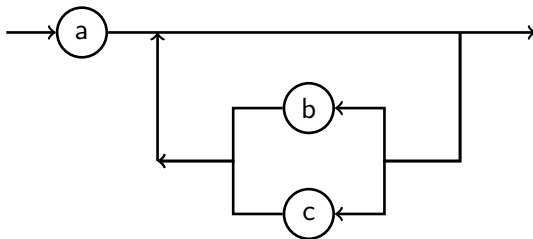


Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a ✓
- aabbcc

- abccb
- bcbc
- bcabc

Noch ein Beispiel

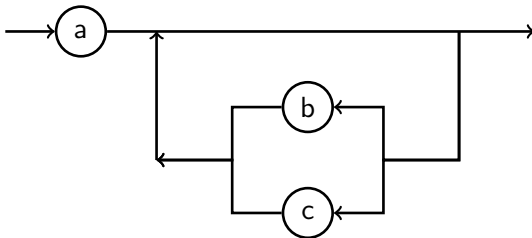


Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a ✓
- aabbcc ✗

- abccb
- bcbc
- bcabc

Noch ein Beispiel

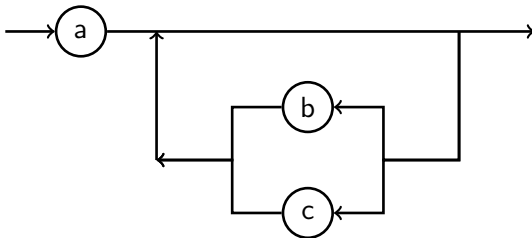


Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a ✓
- aabbcc ✗

- abccb ✓
- bcbc
- bcabc

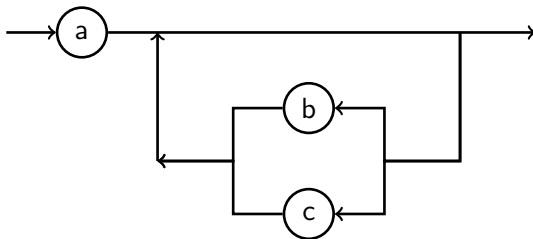
Noch ein Beispiel



Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a ✓
- aabbcc ✗
- abccb ✓
- bc bc ✗
- bcabc

Noch ein Beispiel

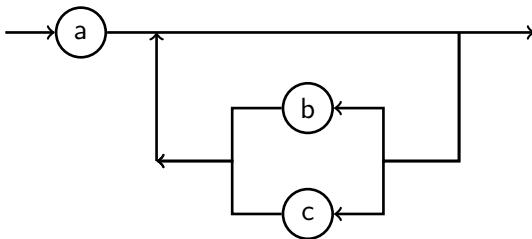


Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a ✓
- aabbcc ✗

- abc cb ✓
- bc bc ✗
- bcabc ✗

Noch ein Beispiel

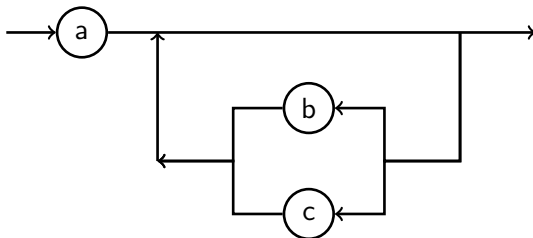


Welche Worte werden durch das Syntaxdiagramm erkannt?

- abbbbb ✓
- a ✓
- aabbcc ✗
- abc cb ✓
- bc bc ✗
- bcabc ✗

Allgemein: Alle Worte, die mit genauem einem a anfangen und anschließend beliebige b und c folgen dürfen.

Noch ein Beispiel



Welche Worte werden durch das Syntaxdiagramm erkannt?

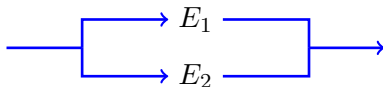
- abbbbb ✓
- a ✓
- aabbcc ✗
- abccb ✓
- bc bc ✗
- bcabc ✗

Allgemein: Alle Worte, die mit genauem einem a anfangen und anschließend beliebige b und c folgen dürfen.

EBNF dazu: $N = "a" \{ "b" \mid "c" \}$

Korrespondenz von EBNF und Syntaxdiagrammen (1)

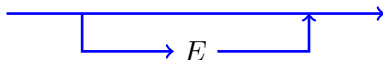
- **Auswahl** $E_1 \mid E_2$ wird durch eine Verzweigung repräsentiert:



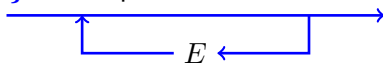
- **Sequentielle Komposition** $E_1 E_2$ wird durch Aneinanderhängen repräsentiert:



- **Option** $[E]$ wird repräsentiert durch:



- **Wiederholung** $\{E\}$ wird repräsentiert durch:



Korrespondenz von EBNF und Syntaxdiagrammen (2)

Umgekehrt geht es auch:

Jedes Syntaxdiagramm lässt sich als EBNF darstellen.

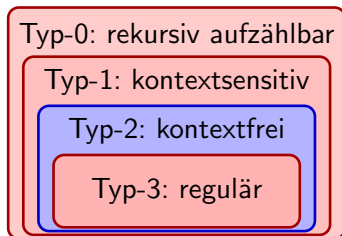
Folgerung

Syntaxdiagramme und EBNF sind äquivalent in dem Sinne, dass sie die gleiche Klasse von formalen Sprachen beschreiben.

Man nennt diese Klasse: Die **Kontextfreien Sprachen**

Der Begriff „kontextfrei“ rührt daher, dass Nichtterminale ersetzt werden, ohne benachbarte Symbole zu betrachten (den Kontext)

- Die Informatik verwendet zur Klassifizierung von formalen Grammatiken häufig die sogenannte **Chomsky-Hierarchie** (benannt nach Noam Chomsky)



- Beispiel für eine formale Sprache die **nicht kontextfrei** ist:

Die Sprache bestehend aus allen Wörtern der Form $a^n b^n c^n$ mit $n \geq 1$, d.h. die Wörter $abc, aabbcc, aaabbbccc, aaaabbbbcccc, \dots$

Reguläre Grammatiken haben nur Regeln der Form

$$E = F_1 \mid \dots \mid F_n$$

wobei jedes der F_i von einer der folgenden Formen ist:

- w mit w ist ein Wort (d.h. Folge von Terminalsymbolen),
- wE' mit w ein Wort und E' ein Nichtterminalsymbol, oder
- ε (das leere Wort)

Die durch reguläre Grammatiken erzeugten Sprachen heißen
reguläre Sprachen

Reguläre Grammatiken: Beispiele

Grammatik G_1 mit Startsymbol A:

$$A = "a" "a" \mid "a" "a" A$$

Die Grammatik G_1 ist regulär und $L(G_1)$ ist die Menge aller Worte mit gerader Anzahl an a's (mindestens 2).

Grammatik mit G_2 mit Startsymbol B:

$$B = "a" "a" \mid "a" B "a"$$

Die Grammatik ist **nicht regulär**, aber $L(G_2) = L(G_1)$.

Beachte: Die Sprache bestehend aus allen Worten mit einer geraden Anzahl an a's ist eine **reguläre Sprache**, da sie durch **eine** reguläre Grammatik erzeugt wird.

Reguläre Sprachen (3)

Reguläre Sprachen sind eine **echte Teilmenge** der kontextfreien Sprachen (d.h. der durch EBNFs erzeugten Sprachen).

Beispiel: Die Sprache aller Worte der Form $a^n b^n$ mit $n \geq 1$ ist nicht regulär, aber die EBNF

$$A_n B_n = "a" "b" \mid "a" A_n B_n "b"$$

erzeugt diese (d.h. die Sprache ist kontextfrei).

Beispiel: Palindrome

Ein **Palindrom** ist ein Wort das von links oder von rechts gelesen dasselbe Wort ergibt (griechisch palíndromos: „rückwärts laufend“)

Palindrome sind z.B.

”legeaneinebrandnarbenienaegel”

ANNA

ANANA

A

37873

Keine Palindrome:

37687

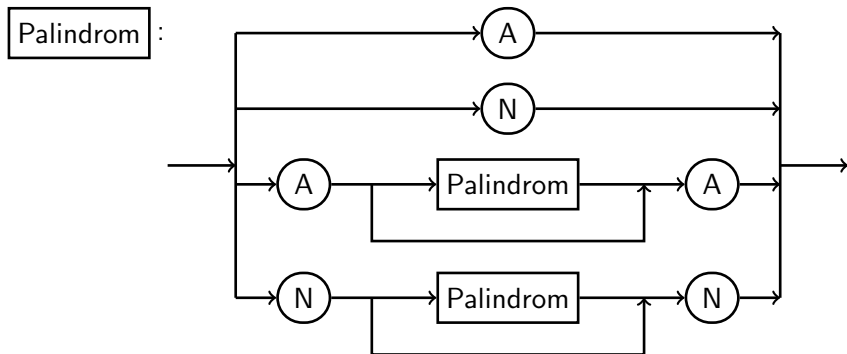
ANANAS

ANNAN

ANAA

Syntaxdiagramm für Palindrome

Syntaxdiagramm für Palindrome, die mit den Buchstaben A und N gebildet werden können:



EBNF für Palindrome

EBNF-Grammatik für Palindrome, die mit den Buchstaben A und N gebildet werden können:

```
Palindrom = "A"  
           | "N"  
           | "A" [ Palindrom ] "A"  
           | "N" [ Palindrom ] "N"
```

- Formalismen zur Beschreibung der **Syntax von Programmiersprachen**
- Backus-Naur-Form (BNF)
- Erweiterte Backus-Naur-Form (EBNF)
Jede (E)BNF-Grammatik besteht aus
 - einem Startsymbol und
 - einer Menge von Regeln (eine Regel pro Nichtterminalsymbol)
- Jede (E)BNF-Grammatik G erzeugt eine Sprache $L(G)$ (Menge von Wörtern)

Zusammenfassung (2)

- **Syntaxdiagramme** sind ein zu (E)BNF-äquivalenter grafischer Formalismus, wobei
 - Nichtterminalsymbole in Rechtecken
 - Terminalsymbole in Ovalen
 - Pfeile führen von einem Sprachelement zum anderen
 - jedes Syntaxdiagramm besitzt genau einen Eingangs- und Ausgangspfeil.
- All diese Formalismen repräsentieren die sogenannten kontextfreien Sprachen (vgl. Chomsky-Hierarchie)