252-0027
Einführung in die Programmierung

1. EBNF

Manuela Fischer, Malte Schwerhoff

Departement Informatik
ETH Zürich

1. EBNF

1.1 Motivation

1.2 Definition

1.3 Kontrollformen

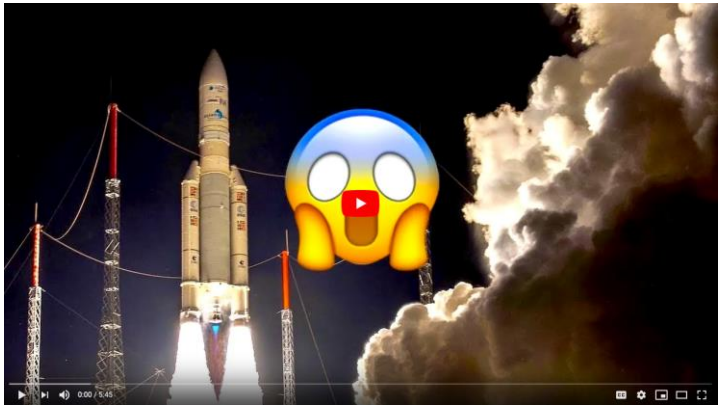
1.4 Ableitungen

1.5 Graphische Darstellung von EBNF-Beschreibungen

1.6 Kontrollformen II: Rekursion

Genaue Beschreibungen

- wichtiges Thema für Informatik
  - Programmiersprachen
  - Werte
  - Input
  - …



The code that exploded a rocket

Ariane 5 Explosion | A Very Costly Coding Error

18,406 views • Jan 2, 2019

Top Quark
894 subscribers

Beispiel

Universität Zürich[UZH]

Waist Circumference und Waist-to-Height-Ratio bei Schweizer Stellungspflichtigen 2016

Entwurf Schlussbericht zuhanden des Bundesamtes für Gesundheit (BAG-Vertragsnummer 16.008898)

- Format der eingelesenen Körpergrösse
  - Einheit
    - cm (166), m (1.66) oder mm (1660)?
  - Genauigkeit
    - 1660 mm, 1663 mm, 1700 mm?
  - Darstellung
    - 1.66E+03 mm, 1660 mm, 1’660 mm

Erlaubte Werte (in einem Format)

- Sind eine Sprache (Menge von erlaubten Zeichenfolgen)
- Brauchen präzise und verständliche Beschreibung
  - Beschreibung mit Text
    - Missverständnisse…
  - Beschreibung mit Formalismus (Menge von Regeln)
    - Präzise
    - Erlaubt automatische Prüfung

EBNF (Extended Backus Naur/Normal Form)

- Formalismus zur Beschreibung der Syntax einer Sprache
  - nur Form/Struktur, nichts über die Bedeutung
- Automatische Prüfung möglich
- Viele Anwendungsgebiete
  - Beschreibung von Programmiersprachen
  - Beschreibung von Inputs
  - Food for thought für später (3. Semester):
    Welche Sprachen sind mit EBNF beschreibbar?

Wieso EBNF in EProg?

- Wichtig für weiterführende Vorlesungen
  - Theoretische Informatik, Compiler Design, ...
- Schult formales und präzises Denken
  - Sehr wichtig fürs Programmieren und fürs Informatik-Studium!
- Programmieren im Kleinen
  - Programmieren (in Java) ↔ Erstellen einer EBNF-Beschreibung
  - ähnliche Konzepte
- Gibt Ihnen Zeit, Java zu installieren
- Neu für (fast) alle, auch für die mit Programmierkenntnissen

1. EBNF

Definition EBNF-Beschreibung

- **EBNF-Beschreibung** («**EBNF description**»)
  - Formale Beschreibung der Struktur einer Sprache
  - Besteht aus Menge von EBNF-Regeln
    - Reihenfolge unwichtig
  - gibt an, welche Wörter (Zeichenfolgen) erlaubt sind
    - Erlaubt, falls nach Regeln gebildet (wird später erklärt!

- **EBNF-Regel** («**EBNF rule**»): LHS ⟵ RHS

Definition EBNF-Regel

Form `LHS ⟵ RHS`

- linke Seite («left-hand side») LHS
  - Name der EBNF-Regel in eckigen Klammern: <name>
  - Beliebiger Name möglich; deskriptive und relevante Namen sinnvoll!
  - Jeder Name darf nur einmal vorkommen auf linker Seite!
- Pfeil ⟵ ausgesprochen als «ist definiert als»
- rechte Seite («right-hand side») RHS

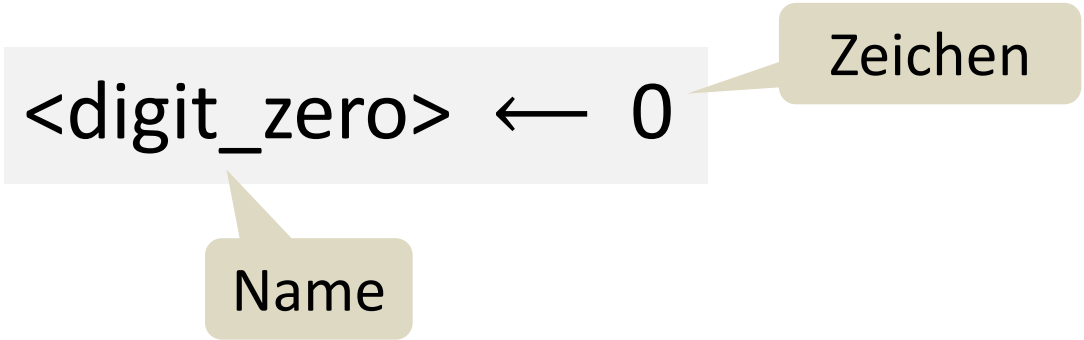
Rechte Seite

- Definition (genaue Beschreibung) für den Namen LHS
  - Beschreibt die Menge der (erlaubten) Wörter

- kann bestehen aus
  1. Zeichen (auch **Literal** oder **Terminal** genannt)
  2. Namen von EBNF-Regeln (auch **Nonterminal** genannt)
  3. Kombinationen der vier **Kontrollformen** («**control forms**»)
     - Aufreihung («sequence»)
     - Entscheidung («decision»): Auswahl und Option
     - Wiederholung («repetition»)
     - Rekursion («recursion»)

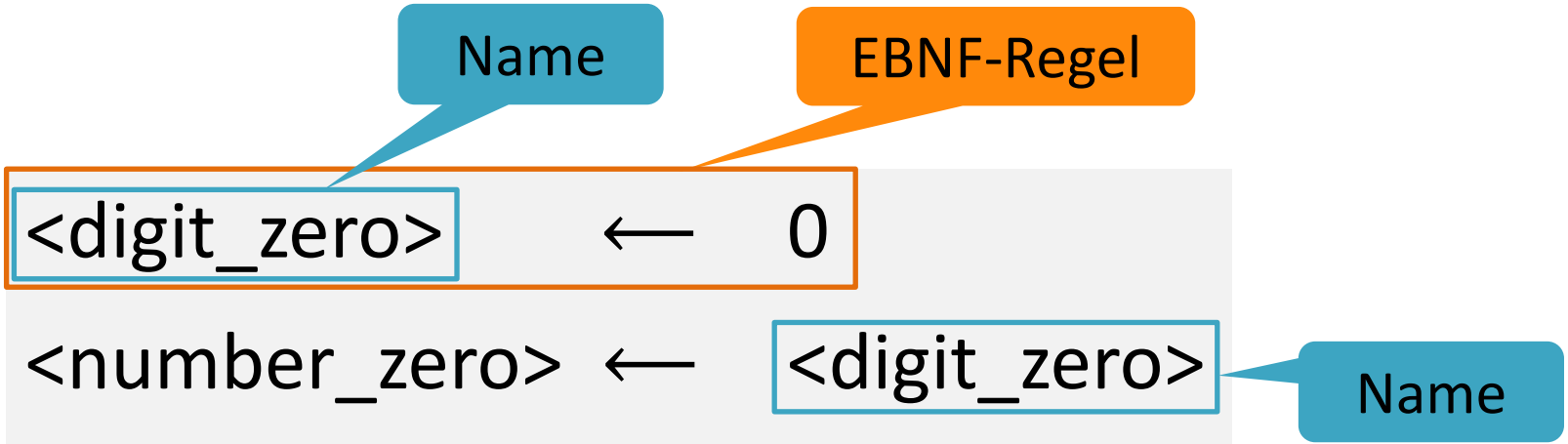
Ähnlich auch in Java

1. Zeichen als RHS



Regel <digit_zero> lässt nur genau dieses Zeichen 0 als Wort zu.

2. Namen als RHS



Regel <number_zero> lässt genau die durch <digit_zero> beschriebenen Wörter zu.

1. EBNF

1.3 Kontrollformen

1.3.1 Aufreihung

1.3.2 Entscheidung

1.3.3 Wiederholung

Aufreihung («sequence»)

E1 E2 E3

- Folge von beliebig vielen Elementen
  - von links nach rechts gelesen: Reihenfolge ist wichtig
  - Jedes Element muss gültige RHS sein
  - Abstand für Übersichtlichkeit; kein echtes Leerzeichen

Beispiel

Regeln für Beschreibung des Vorlesungssaals 1: «D28»

<letter_D> ⟵ D

<digit_2> ⟵ 2

<digit_8> ⟵ 8

<room1> ⟵ <letter_D> <digit_2> <digit_8>

<room1> ⟵ D 2 8

Aufreihung

Mehrere EBNFs, gleiche Sprache

EBNF 1

<letter_D> ⟵ D

<digit_2> ⟵ 2

<digit_8> ⟵ 8

<room1> ⟵ <letter_D> <digit_2> <digit_8>

EBNF 2

<room1> ⟵ D 2 8

EBNF 3

<room1> ⟵ <letter_D> 2 <digit_8>

<letter_D> ⟵ D

<digit_8> ⟵ 8

Durch EBNF definierte Sprache

- **Anwenden** einer Regel:
  Ersetzen des Namens durch RHS der Regel
- EBNF hat eine spezielle Regel: **Startregel** («**entry rule**»)
  - Konvention: letzte Regel ist die Startregel (falls nicht anders erwähnt)
- **Sprache** der EBNF (informell):
  Menge aller Zeichenfolgen (ohne Namen!), die ausgehend von der Startregel durch wiederholtes Anwenden der EBNF-Regeln erstellt werden können

```
<letter_D> ⟵ D
<digit_8>  ⟵ 8
<room1>    ⟵ <letter_D> 2 <digit_8>
```

Startregel

Sprache: {D28}

1. EBNF

1.3 Kontrollformen

1.3.1 Aufreihung

1.3.2 Entscheidung

1.3.3 Wiederholung

Entscheidung («decision»)

- **Auswahl** («**selection**»)

```
E1 | E2 | E3
```

- **Option** («**option**»)

```
[ E ]
```

Auswahl («selection»)

E1 | E2 | E3

- Menge beliebig vieler Alternativen
  - Durch senkrechten Strich («stroke») | getrennt
  - Alternative: gültige RHS
  - Genau eine der Alternativen wird gewählt

Beispiele

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<room1> ⟵ D 2 8

<room2> ⟵ E 1 2

<room> ⟵ <room1> | <room2>

Klammern

A | B C

- Nicht eindeutig
  - Variante 1: "A oder BC"
  - Variante 2: "AC oder BC"
- Klammern schaffen Klarheit
  - A | (B C) oder (A | B) C
  - Erscheinen nicht in resultierenden Zeichenkette (Wort)

Option («option»)

```
[ E ]
```

- Optionales Element
  - In eckigen Klammern («square brackets») [ ]
  - Element muss gültige RHS sein
  - Kann gewählt werden, muss aber nicht

Beispiele

<initials> ⟵ M [H] S

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<sign> ⟵ + | -

<number> ⟵ [<sign>] <digit>

Option und Auswahl

[E1 | E2]

- Option und Auswahl werden oft kombiniert
- Äquivalent zu **E1 | E2 | ε**
  - **ε** wird «epsilon» ausgesprochen
  - **ε** entspricht der leeren Zeichenfolge
  - **ε** erscheint nicht im Wort

1. EBNF

1.3 Kontrollformen

1.3.1 Aufreihung

1.3.2 Entscheidung

1.3.3 Wiederholung

Wiederholung («repetition»)

```
{ E }
```

- Wiederholung eines Elements
  - Element muss gültige RHS sein
  - 0, 1, 2, … Wiederholungen
  - In geschweiften Klammern («curly braces») { }

Beispiel: Ganze Zahlen

<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<sequence>	⟵	<digit> { <digit> }
<sign>	⟵	+ | -
<integer>	⟵	[<sign>] <sequence>

<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer>	⟵	[+ | -] <digit> { <digit> }

Beispiel: Ganze Zahlen

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> { <digit> }

- Umgangssprachliche Beschreibung
  - <digit> ist definiert als eines der Zeichen 0, …, 9
  - <integer> ist definiert als eine Folge von 3 Elementen
    - ein optionales Vorzeichen (eine der Alternativen + und -)
    - ein <digit>
    - Wiederholung von 0 oder mehr <digit>s

Beispiel: Kanonische Zahlen

- EBNF-Beschreibung für kanonische Zahlen,
  d.h. Zahlen, die keine führenden Nullen haben
  - 007 ist illegal, 7 ist legal

<zero>	⟵	0
<nonzero>	⟵	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit>	⟵	<zero> | <nonzero>
<canonic_int>	⟵	**(** [+ | -] <nonzero> { <digit> } **) | <zero>**

Sonst 0 nicht legal!

Beispiel: Zahlen mit Hochkomma

- 1’234 234 1’123’123 12’123’123 123’123

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<prefix> ⟵ <digit> | <digit> <digit> | <digit> <digit> <digit>

<integer> ⟵ [+ | -] <prefix> { ’ <digit> <digit> <digit>}

Sonderzeichen

- Besondere Zeichen in EBNF-Beschreibungen
  - < > ⟵ | [ ] { } ( ) und Leerzeichen
  - Treten nicht in resultierenden Wörtern auf
  - Falls als Zeichen in Wort erwünscht, verwenden wir Rahmen:
    - } für Zeichen }
    - ␣ oder für Leerschlag

Beispiel: Zahlenmengen

- EBNF-Beschreibung für Zahlenmengen
  - beliebige Anzahl von Zahlen separiert durch Komma zwischen { }
    - { 1 } {3, 2} {3, 2, 3} { }
  - Mehrfachnennungen und Reihenfolge unwichtig
    - Reihenfolge der Zahlen kann nicht erzwungen werden!

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> { <digit> }

<integer_list> ⟵ <integer> { , <integer> }

<integer_set> ⟵ { [<integer_list>] }

1. EBNF

Ist ein Wort (Zeichenfolge) legal?

- Wort ist **legal** falls
  - Informell: alle Zeichen mit den Elementen der Regel übereinstimmen
  - Formal: es eine Ableitung der Zeichenfolge gibt
- **Ableitung** («**derivation**»): Sequenz von Ableitungsschritten
  - startend mit Startregel
  - endend mit Zeichenfolge (ohne Nichtterminale)
  - **Ableitungsschritt** («**derivation step**»)
    - Regel: Namen (LHS) durch Definition (RHS) ersetzen
    - Auswahl: eine Alternative wählen
    - Option: entscheiden, ob optionales Element gewählt wird oder nicht
    - Bestimmung der Anzahl Wiederholungen

Beispiel

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<number> ⟵ [+ | -] { <digit> }

- Legale Zeichenfolgen:
  4, +9, -09, -0, +, -, …
- Illegale Zeichenfolgen:
  +-6, 7-, +-,…

Darstellungsformen einer Ableitung

- Ableitungstabelle
  - Erste Zeile ist Startregel
  - Letzte Zeile ist Zeichenfolge
  - Übergang zwischen zwei Zeilen entspricht Ableitungsschritt

- Ableitungsbaum
  - Wurzel ist Namen der Startregel
  - Blätter sind Zeichen
  - Verbindungen stehen für einen Ableitungsschritt

Beispiel: Ableitung von -4 als Tabelle

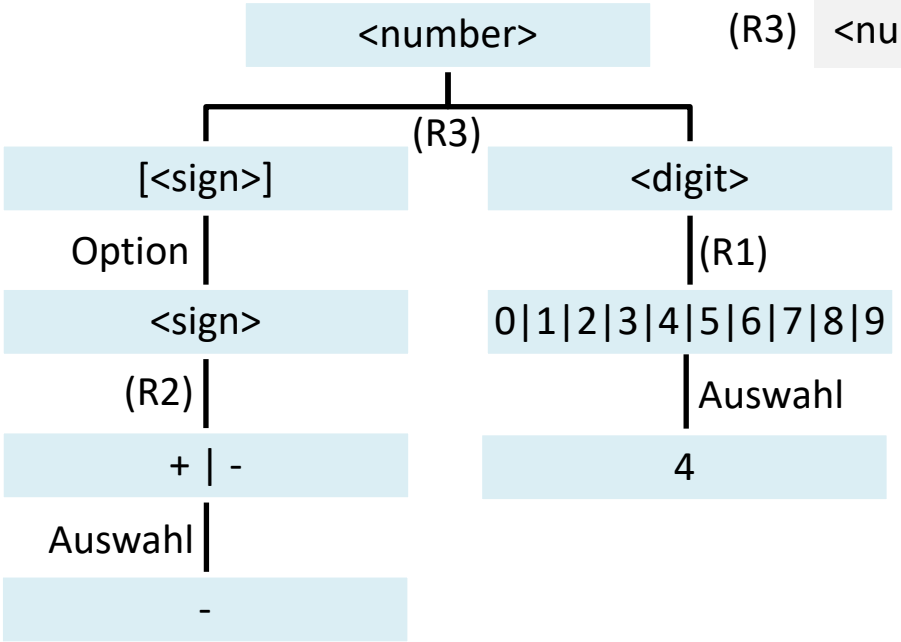
(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<sign>	⟵	+ | -
(R3)	<number>	⟵	[<sign>] <digit>

Begründungen

<number>	⟵	[<sign>] <digit>	(R3)
	⟵	<sign> <digit>	Option wählen
	⟵	(+ | -) <digit>	(R2)
	⟵	- <digit>	Auswahl - wählen
	⟵	- (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)	(R1)
	⟵	- 4	Auswahl 4 wählen

Beispiel: Ableitung von -4 als Baum

(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<sign>	⟵	+ | -
(R3)	<number>	⟵	[<sign>] <digit>



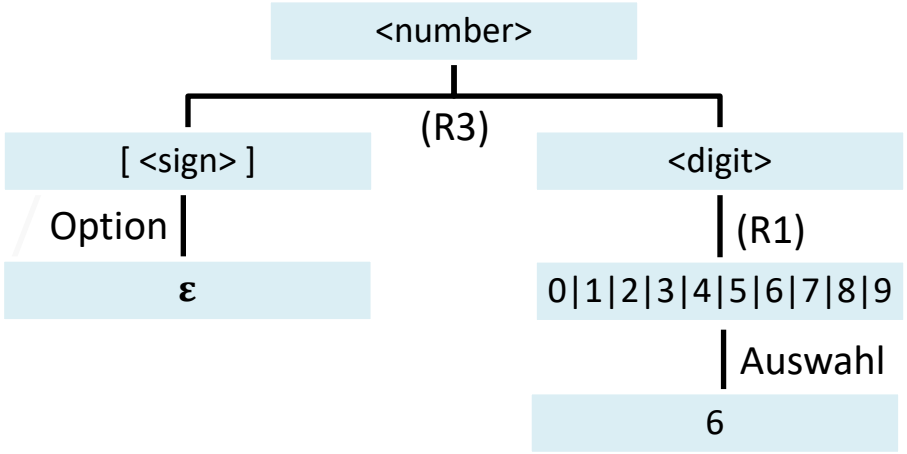
Beispiel: Ableitung von 6 als Tabelle

(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<sign>	⟵	+ | -
(R3)	<number>	⟵	[<sign>] <digit>

<number>	⟵	[<sign>] <digit>	(R3)
	⟵	<digit>	Option nicht wählen
	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	(R1)
	⟵	6	Auswahl 6 wählen

Beispiel: Ableitung von 6 als Baum

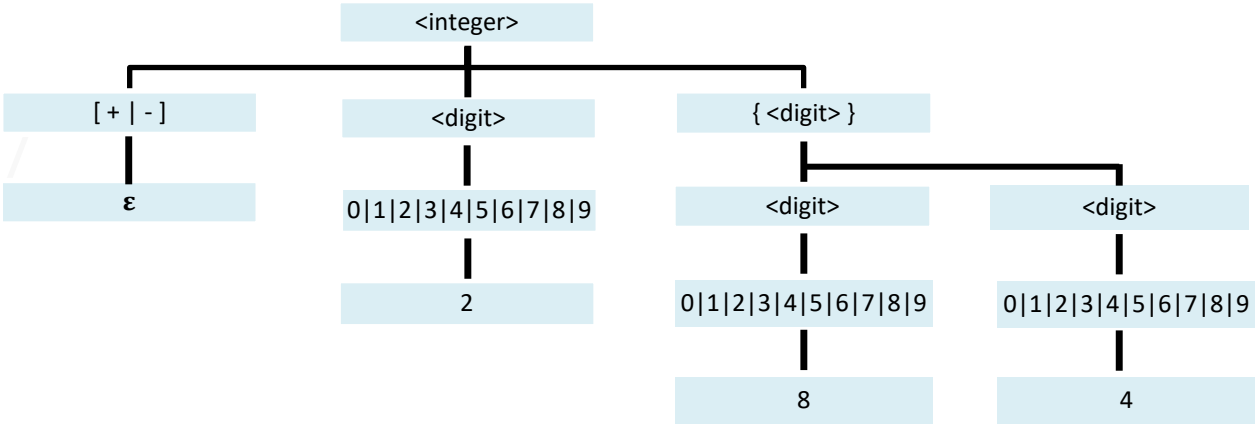
(R1)	<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(R2)	<sign>	⟵	+ | -
(R3)	<number>	⟵	[<sign>] <digit>



Beispiel: Ableitung von 284 als Baum

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

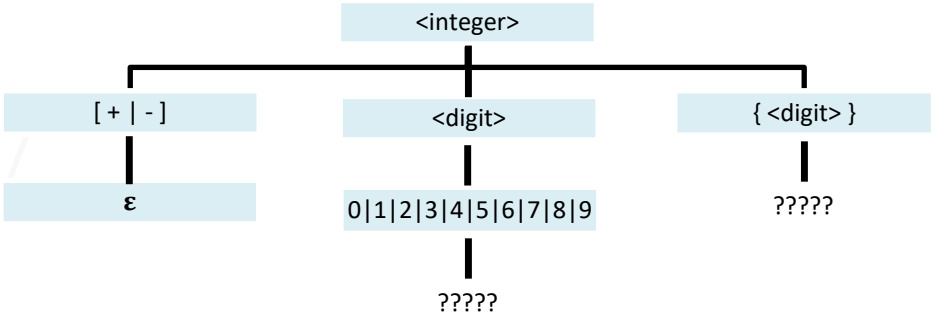
<integer> ⟵ [+ | -] <digit> { <digit> }



Beispiel: Ableitungsversuch für A15

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> { <digit> }



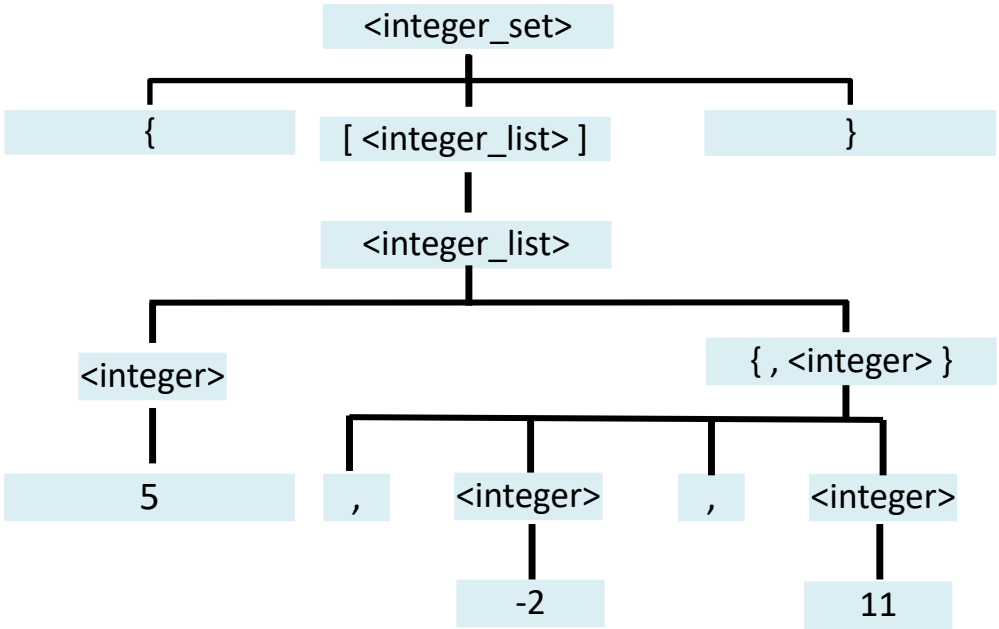
Ableitung von {5, -2, 11}

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> { <digit> }

<integer_list> ⟵ <integer> { , <integer> }

<integer_set> ⟵ { [<integer_list>] }



Äquivalenz von EBNF-Beschreibungen

- **Sprache** der EBNF: Menge ihrer legalen Zeichenfolgen
- Zwei EBNF-Beschreibungen B_1 und B_2 sind **äquivalent** falls ihre Sprachen gleich sind:
  - Zeichenfolge legal für $B_1 \Leftrightarrow$ legal für B_2
  - Zeichenfolge illegal für $B_1 \Leftrightarrow$ illegal für B_2
- Anzahl und Namen der Regeln irrelevant für Äquivalenz

Syntax versus Semantik

- **Syntax**: Form/Struktur
  - Wird von EBNF-Formulierung beschrieben
- **Semantik** («**semantics**»): Bedeutung/Interpretation
  - Falls nicht legal (falsche Syntax), dann keine Bedeutung (undefiniert)

- Zwei wichtige Fragen zur Semantik:
  - Können unterschiedliche Zeichenfolgen die gleiche Bedeutung haben?
    - «Herr Wirth», «Professor Wirth» und «Niklaus Wirth» meinen die gleiche Person
  - Kann eine Zeichenfolge mehrere (verschiedene) Bedeutungen haben?
    - «nächste Vorlesung» ist abhängig vom aktuellen Datum

Beispiele Semantik

- **<integer>-EBNF**
  - Bedeutung ist der Wert der Zahl
  - Unterschiedliche Zeichenfolgen können gleiche Bedeutung haben
    - 1 und +1
    - 0, +0 und -0
    - 0012 und 12? Je nachdem… (z.B. Mathematik vs. PIN-Code)
- **<integer_set>-EBNF**
  - Bedeutung ist mathematische Menge
  - Unterschiedliche Darstellungen mit gleicher Bedeutung
    - {1, 2, 3}, {3, 2, 1} und {1, 1, 1, 2, 2, 2, 3, 3, 3}

1. EBNF

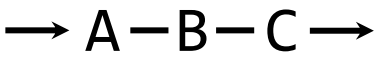
Graphische Darstellung von EBNF-Regeln

- EBNF-Beschreibung: Menge von Syntax-Graphen
  - Ein Syntax-Graph für jede Regel
- gültige Zeichenfolge: Pfad durch Graph von links nach rechts
- Substitution: Syntax-Graphen in andere einsetzen
  - Interne Namen verschwinden
  - Entspricht einem Anwenden der Regel

Graphische Darstellung von EBNF-Regeln

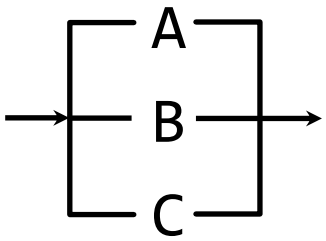
- Aufreihung

```
A B C
```



- Auswahl

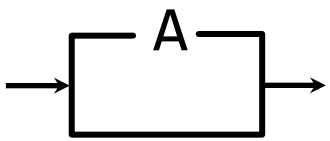
```
A | B | C
```



Graphische Darstellung von EBNF-Regeln

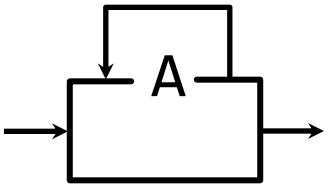
- Option

```
[ A ]
```



- Wiederholung

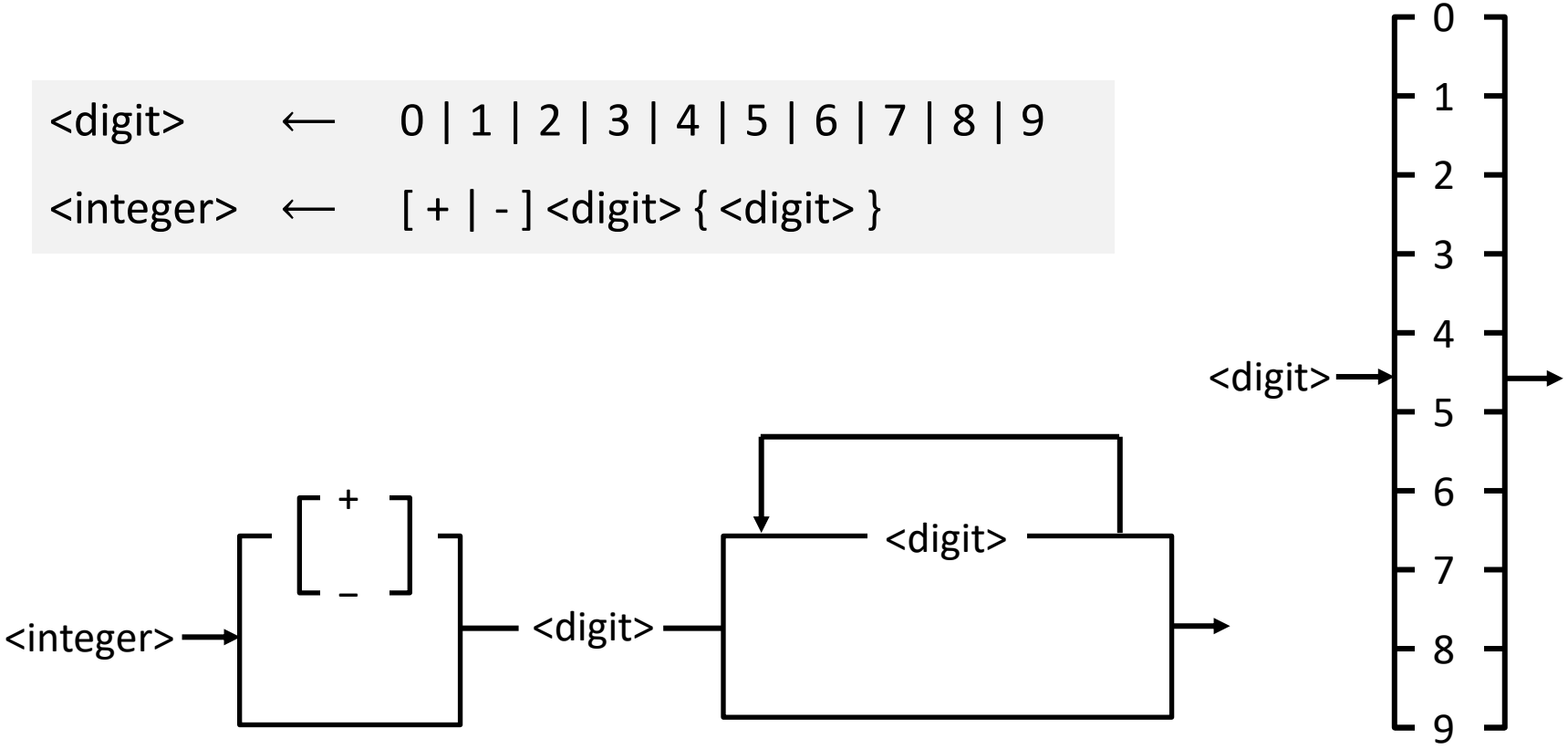
```
{ A }
```



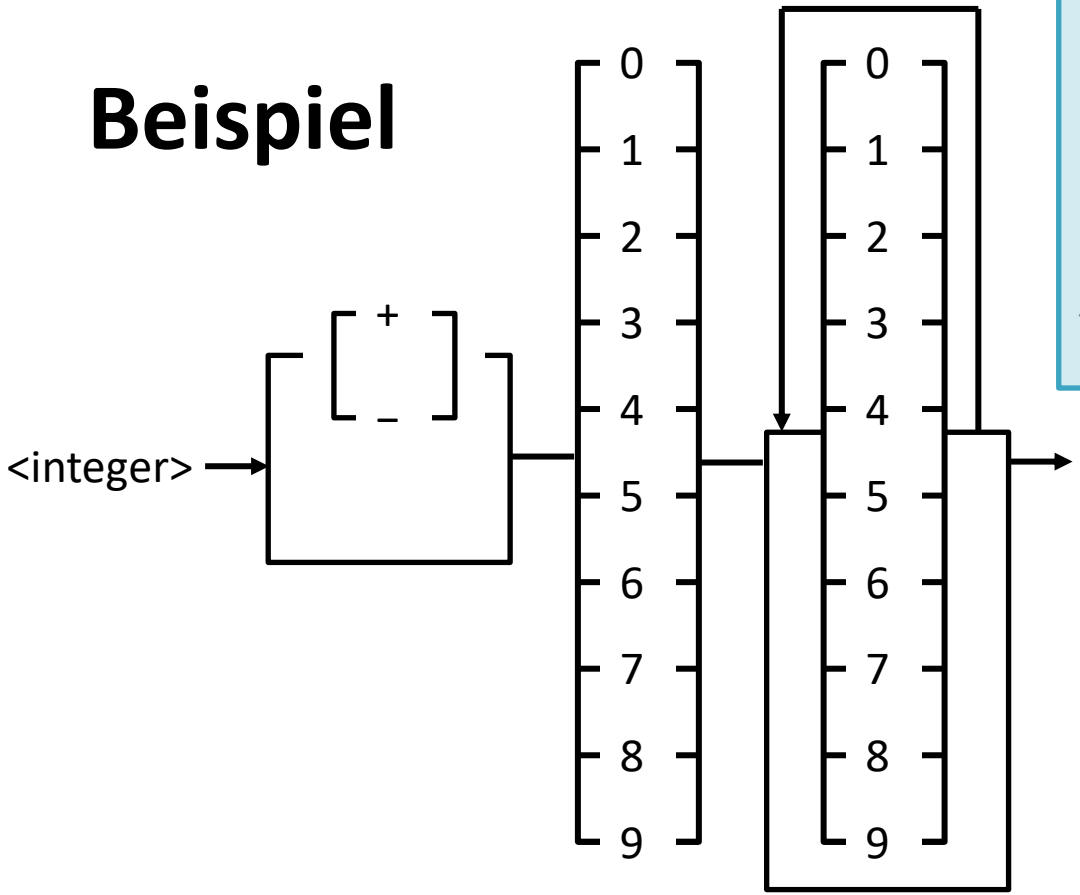
Beispiel

<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> { <digit> }

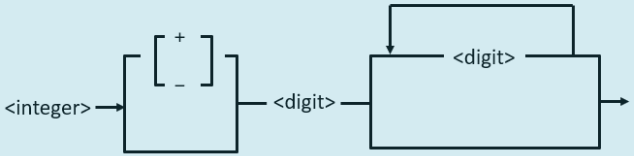


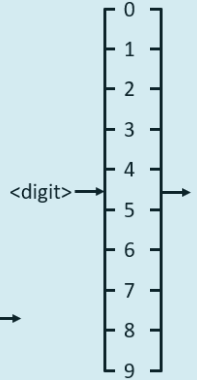
Beispiel



<digit> ⟵ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ⟵ [+ | -] <digit> { <digit> }

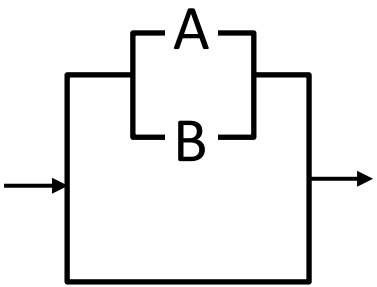




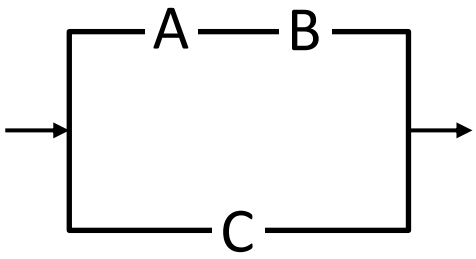
<integer> ⟵ [+ | -] (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

Welche Wörter sind legal für Graph?

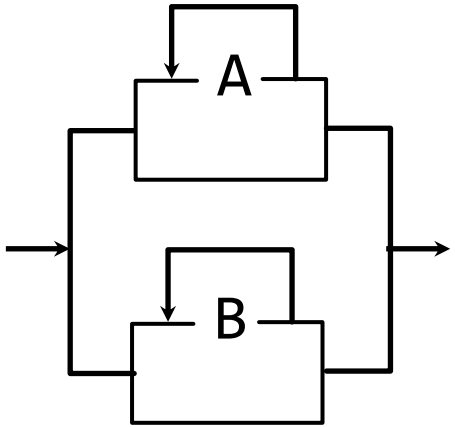
a)



b)



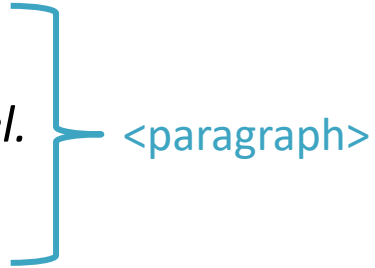
c)



1. EBNF

Eine unendliche Geschichte mit Rekursion

Es war einmal ein Mann.
Der hatte einen hohlen Zahn.
In diesem hohlen Zahn befand sich eine Schachtel.
In der Schachtel war ein Brief.
In diesem Brief stand:



Es war einmal ein Mann.
Der hatte einen hohlen Zahn.
In diesem hohlen Zahn befand sich eine Schachtel.
In der Schachtel war ein Brief.
In diesem Brief stand:

...

<endless_story> ⟵ **<paragraph> <endless_story>**

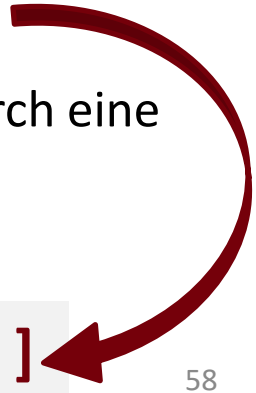
Rekursion mit EBNF

- Name der EBNF-Regel auch auf rechten Seite

  <endless_story> ⟵ <paragraph> <endless_story>

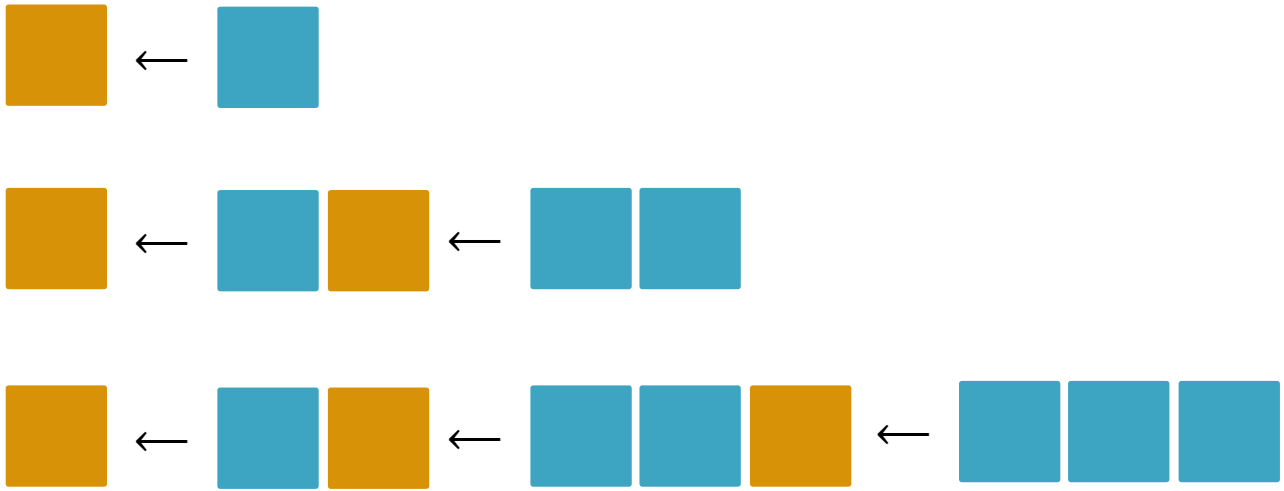
- Welche Zeichenfolgen sind gültig?
  - Keine! Immer ein Nichtterminal auf rechten Seite!
- Sinnvolle/endliche Rekursion braucht Abbruchsoption
  - Es muss (mindestens) einen Weg geben, <endless_story> durch eine RHS ohne diesen Namen zu ersetzen
  - Optionaler Name auf rechter Seite

  <endless_story> ⟵ <paragraph> [<endless_story>]



Beliebig lange Geschichte mit Rekursion

<endless_story> ⟵ <paragraph> [<endless_story>]



...

Rekursion

- **Direkte Rekursion** («**direct recursion**»): Name der Regel auf RHS

```
<A>   ⟵   a | [ <A> ]
```

- **Indirekte Rekursion** («**indirect recursion**»): Folge von Regeln $R_1, \ldots, R_k$:
  - R_{i+1} erscheint auf der rechten Seite von R_i für $1 \leq i \leq k-1$ und
  - R_1 erscheint auf der rechten Seite von R_k

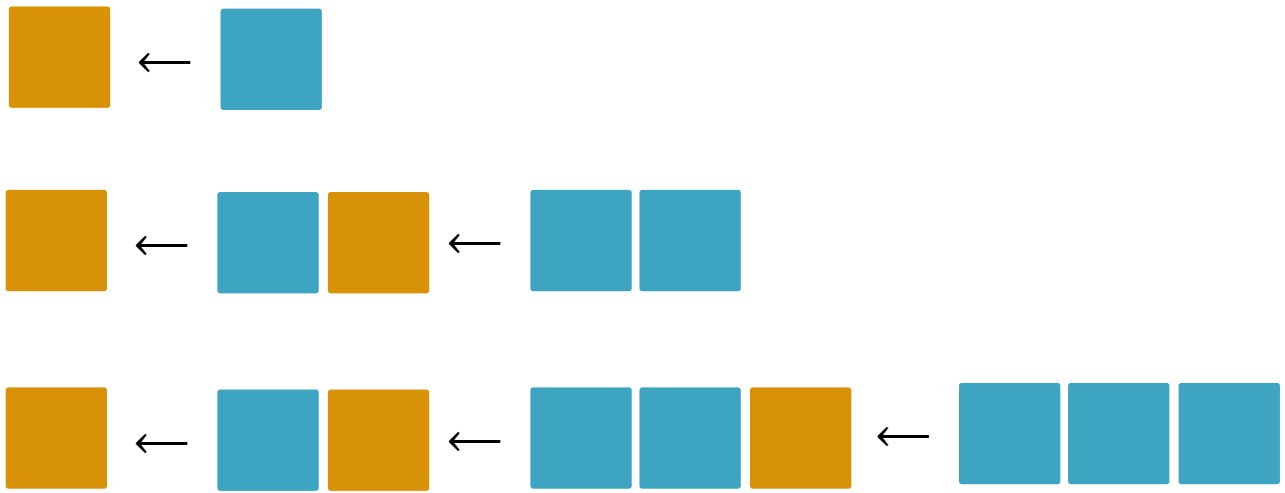
```
<A>   ⟵   a | <B>

<B>   ⟵   [ <A> ]
```

- EBNF-Beschreibung ist rekursiv, falls sie rekursive Regel hat

Beispiel: natürliche Zahlen

<positive_integer> ⟵ <digit> [<positive_integer>]



...

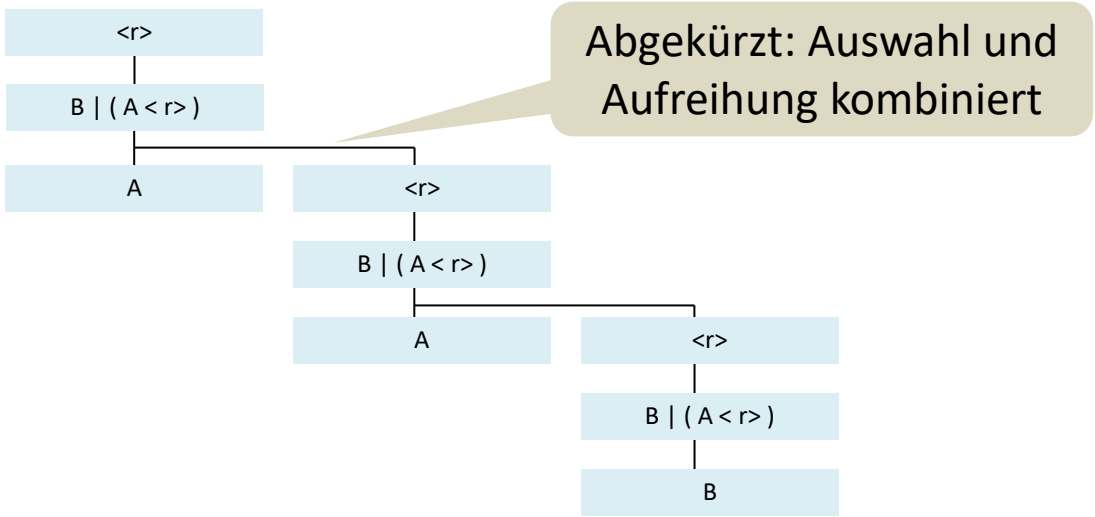
Beispiel: Ableitungstabelle von “AAB”

<r> ⟵ B | (A <r>)

<r>	⟵	B | (A <r>)	Regel
	⟵	A <r>	Auswahl (A <r>)
	⟵	A (B | (A <r>))	Regel
	⟵	A A <r>	Auswahl (A <r>)
	⟵	A A (B | (A <r>))	Regel
	⟵	A A B	Auswahl (B)

Beispiel: Ableitungbaum von ”AAB”

<r> ⟵ B | (A <r>)



Rekursion versus Wiederholung

- Viele Probleme haben kurze Lösung mit und ohne Rekursion
  - Natürliche Zahlen

<positive_int>	⟵	<digit> { <digit> }
<positive_int>	⟵	<digit> [<positive_int>]

  - Unendliche Geschichte

<endless_story>	⟵	<paragraph> { <paragraph> }
<endless_story>	⟵	<paragraph> [<endless_story>]

Rekursion versus Wiederholung

- EBNF-Beschreibung für die Sprache $\{ A^n B^n \mid n \in \mathbb{N} \}$
  - Mit Rekursion

<equalAB>	⟵	[A <equalAB> B]	

  - Ohne Rekursion

<equalAB>	⟵	{ A } { B }	**ABBB**
<equalAB>	⟵	{ A B }	**ABAB**

Rekursion versus Wiederholung

- Bei EBNF: Rekursion mächtiger
  - Es gibt Sprachen, die nur mit Rekursion beschrieben werden können!
  - Alle Sprachen, die mit Wiederholung beschrieben werden können, können auch mit Rekursion beschrieben werden.

- Nebenbemerkung: Bei Programmiersprachen gleich mächtig!
  - Jede Rekursion kann durch Wiederholung(en) ausgedrückt werden und umgekehrt!
  - Manchmal eine Formulierung einfacher/eleganter

Geschichte von EBNF

- **Zuerst gabs nur BNF…**
  - Aufreihung, Auswahl, Rekursion
- **…dann wurde sie zur EBNF**
  - Niklaus Wirth hat Option und Wiederholung hinzugefügt
  - Nicht essentiell
  - Einfachere Lesbarkeit

Short Communications
Programming Languages

What Can We Do about the Unnecessary Diversity of Notation for Syntactic Definitions?

Niklaus Wirth
Federal Institute of Technology (ETH), Zürich, and
Xerox Palo Alto Research Center

Key Words and Phrases: syntactic description language, extended BNF
CR Categories: 4.20

The population of programming languages is steadily growing, and there is no end of this growth in sight. Many language definitions appear in journals, many are found in technical reports, and perhaps an even greater number remains confined to proprietory circles. After frequent exposure to these definitions, one cannot fail to notice the lack of "common denominators." The only widely accepted fact is that the language structure is defined by a syntax. But even notation for syntactic description eludes any commonly agreed standard form, although the underlying ancestor is invariably the Backus-Naur Form of the Algol 60 report. As variations are often only slight, they become annoying for their very lack of an apparent motivation.

Out of sympathy with the troubled reader who is weary of adapting to a new variant of BNF each time another language definition appears, and without any claim for originality, I venture to submit a simple notation that has proven valuable and satisfactory in use. It has the following properties to recommend it:

1. The notation distinguishes clearly between meta-, terminal, and nonterminal symbols.
2. It does not exclude characters used as metasymbols from use as symbols of the language (as e.g. "|" in BNF).
3. It contains an explicit iteration construct, and thereby avoids the heavy use of recursion for expressing simple repetition.
4. It avoids the use of an explicit symbol for the empty string (such as ⟨empty⟩ or ϵ).
5. It is based on the ASCII character set.

This meta language can therefore conveniently be used to define its own syntax, which may serve here as an example of its use. The word *identifier* is used to denote *nonterminal symbol*, and *literal* stands for *terminal symbol*. For brevity, *identifier* and *character* are not defined in further detail.

```
syntax     = {production}.
production = identifier "=" expression ".".
expression = term {"|" term}.
term       = factor {factor}.
factor     = identifier | literal | "(" expression ")" |
             "[" expression "]" | "{" expression "}".
literal    = """" character {character} """".
```

Repetition is denoted by curly brackets, i.e. {a} stands for ϵ | a | aa | aaa | . . . . Optionality is expressed by square brackets, i.e. [a] stands for a | ϵ. Parentheses merely serve for grouping, e.g. (a|b)c stands for ac | bc. Terminal symbols, i.e. literals, are enclosed in quote marks (and, if a quote mark appears as a literal itself, it is written twice), which is consistent with common practice in programming languages.

Received January 1977; revised February 1977

Author's present address: Xerox Corporation, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.

Communications of the ACM
November 1977
Volume 20
Number 11

Beispiel: <integer> mit BNF statt EBNF

<sign>	⟵	+ | -
<digit>	⟵	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digits>	⟵	<digit> | <digit> <digits>
<integer>	⟵	(ε | <sign>) <digits>

Food for thought:
Zeichen | für Auswahl nicht nötig. Was stattdessen?