Einführung in die Informatik I

Winter 2005/2006

5 Formale Sprachen

5 Formale Sprachen ...

Motivation: Ziele dieses Abschnittes

- Verständnis grundlegender Beschreibungsmerkmale für Sprachen
  - Backus-Naur Form
  - Syntaxdiagramme, Ableitungsbäume

Herausforderungen:

- erste „richtige“ Methodik zur Abstraktion
- erstes Auftreten einer Rekursion

Literatur: Diverse Abschnitte in [Claus] und

- Kapitel 7 (insb. Abschnitt 7.2) in [Ernst]

Sprache und Kommunikation

Kommunikation: Austausch von Information zwischen zwei oder mehreren, unabhängigen Instanzen

Kommunikationsform & -kanal: Wie wird die Information übertragen?

→ Sprechen, Zeigen, Schreiben, Zeichnen, . . .

Sprache allgemein:

- Verschlüsselte und übertragbare Form der Information
- baut auf einem Alphabet (Zeichensatz) auf

Natürliche Sprache: Durch Evolution entstandene Sprache (Deutsch, Englisch, aber auch Gene, Bienensprache etc.)

Künstliche Sprache:

- Durch Menschen geschaffene Sprache mit festen Regeln
- Beispiele: Programmiersprache, Mathematik

Sprachbestandteile: Notwendig zur Verwendung von Sprachen:

1. ***Syntax***: Regeln für gültige Verwendung des Alphabets zur Sprachbildung
   Bei natürlichen Sprachen: Rechtschreibung, Zeichenregeln, Grammatik

   Beispiel: Großbuchstaben am Satzanfang, Punkt am Satzende

2. ***Semantik***: Lehre von der inhaltlichen Bedeutung einer Sprache

   „Das Haus ißt Kuchen“ syntaktisch korrekt, semantisch nicht

Formale Sprache: Abstraktion von der konkreten Sprache

Formale Sprache wird durch konkrete Syntax beschrieben bzw. definiert

Motivation: Warum formale Sprachen?

Problem: Natürliche Sprachen haben große Nachteile:

- kompliziertes Regelwerk (z.B. unregelmässige Verben)
- hohe Abhängigkeit vom Kontext (Semantik von „Bank“, „Flügel“ etc.)

Aber: „Natürliche Kommunikation“ ermöglicht „parallele“ Kommunikation über mehrere Kanäle („Zwischen den Zeilen Lesen“, „Im Gesicht lesen“)

Rechner brauchen Sprachen, die

- vollständig sind (alle Aspekte müssen definiert sein)
- unabhängig vom umgangssprachlichen oder kulturellen Kontext sind

5.1 Die Backus-Naur-Form (BNF)

Ziel: Definition einer Syntax zu einer Sprache

BNF: Konzept geht auf Noam Chomsky (1955,1957) und John Backus & Peter Naur (1959) zurück

Grundidee:
- Zerlegung komplexer Ausdrücke in einfache Teilausdrücke
- Zerlegung endet bei ***Terminalsymbolen***, die nicht weiter zerlegbar sind

Grundlage ist eine verallgemeinerte Form von
- ***Alphabet***: Menge unterschiedlicher Zeichen, z.B. $A = \{X, a, \%, \$\}$
- ***Wortmenge*** (Nachrichtenraum): Menge aller beliebiger Zeichenfolgen, z.B. $A^* = \{\varepsilon, X, XXaa, aa\%\%, \ldots\}$ (ε steht für das leere Wort)

Bestandteile einer BNF
1. Menge von ***Terminalsymbolen*** (auch ***Terminale***)
2. Menge von ***Nichtterminalsymbolen*** (auch ***Nonterminale***), die der Worterzeugung dienen
3. Menge von ***Regeln***
4. ein ***Startsymbol*** (ein Nonterminal)

Definition der Backus-Naur Form

Regeln haben folgende Form

```
<Nonterminal> ::= rechts
```

- `Nonterminal` ist ein Nonterminal
- `rechts` ist eine beliebige Folge von Terminalen und Nonterminalen
- lies „`<A> ::= B`“ als „`B` kann `A` ersetzen“

Ableitung: Konkrete Anwendung einer Regel

Regel: `<A> ::= B` Ableitung: `A` ⟶ `B` („`B` ersetzt `A`“)

Startsymbol ist linkes Nonterminal in der ersten Regel

Worte der Sprache sind alle Sequenzen von Terminalsymbolen, die sich aus den Regeln, beginnend beim Startsymbol, ableiten lassen

⇒ Nonterminale sind nur Hilfsmittel zur Worterzeugung

Definition der Backus-Naur Form (Forts.)

Alternative Regeln für ein Nonterminal werden durch „|“ getrennt

```
<Nonterminal> ::= rechts1
                | rechts2
                | rechts3
```

Optionaler Ausdruck

```
<Nonterminal> ::= rechts1 [ rechts2 ]
```

das Nonterminal kann durch `rechts1` oder `rechts1 rechts2` ersetzt werden

Wiederholter Ausdruck

```
<Nonterminal> ::= rechts1 { rechts2 }
```

`rechts2` tritt ein- oder mehrfach auf

Beispiel: Binäre Zahlen

Aufbau binärer Zahlen: Beliebig lange Sequenzen aus `0,1`

Variante 1: BNF für beliebig lange Binärzahlen

```
<Zahl>    ::= <Ziffer> | <Ziffer><Zahl>
<Ziffer>  ::= 0 | 1
```

Rekursion: Nonterminal `Zahl` kann durch sich selbst ersetzt werden

Beispiel: Ableitung der Binärzahl `110`

```
   <Zahl>                      → <Ziffer><Zahl>
→ <Ziffer><Ziffer><Zahl>       → <Ziffer><Ziffer><Ziffer>
→ 1<Ziffer><Ziffer>            → 11<Ziffer>
→ 110
```

Beispiel: Binäre Zahlen (Forts.)

Syntax-Definition ist für Binärzahlen mehrdeutig

Variante 2: Alternative BNF mit Option

```
<Zahl>    ::= <Ziffer> [ <Zahl> ]
<Ziffer>  ::= 0 | 1
```

Variante 3: Alternative BNF mit Wiederholung

```
<Zahl>    ::= { <Ziffer> }
<Ziffer>  ::= 0 | 1
```

Beispiel: Eine sehr eingeschränkte Sprache

BNF für eine sehr einfache Sprache

```
<Sprache>   ::= { <Satz> }
<Satz>      ::= <Subjekt> <Prädikat> [ <Objekt> ]
<Subjekt>   ::= Der Mann | Das Kind | Der Hund
<Prädikat>  ::= putzt | malt | schreibt
<Objekt>    ::= ein Haus | einen Zaun | ein Buch
```

Bedeutung der Ableitungsregeln

- `Sprache` ist eine Aneinanderreihung von `Satz`-Elementen
- ein `Satz` besteht aus der Nonterminal-Sequenz `Subjekt-Prädikat` mit optionalem `Objekt`
- Für `Subjekt, Prädikat, Objekt` sind explizite Terminale angegeben
- Beispiel für gültige Sätze in dieser Sprache: „Der Mann malt ein Haus“, „Das Kind putzt ein Buch, „Der Hund schreibt“

Ableitungssequenz

Ziel: Darstellung der Ableitung ***eines Wortes der Sprache***

Beispiel-Syntax: Binärzahlen

```
<Zahl>    ::= { <Ziffer> }
<Ziffer>  ::= 0 | 1
```

Ableitung des Wortes (der Binärzahl) `101`

```
   <Zahl>                 → <Ziffer><Ziffer><Ziffer>
→  1<Ziffer><Ziffer>      → 10<Ziffer>
→  101
```

Ableitungsbaum

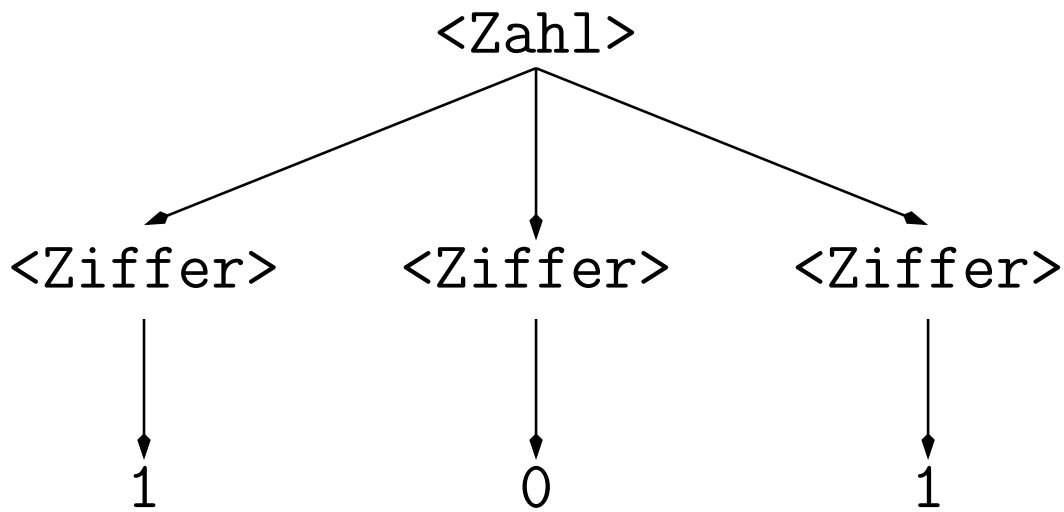
Ziel: Darstellung der Ableitung ***eines Wortes der Sprache***

Ansatz: Hierarchische Darstellung der angewandten Ableitungen

Regeln werden durch Verzweigungen bzw. Pfeile symbolisiert

Wort ergibt sich als Terminal-Folge im Baum von links nach rechts

Beispiel: Ableitung von `101` (BNF der vorherige Folie)



Eindeutigkeit von Wörtern und Syntax

Frage: Läßt sich ein Wort auf genau eine Weise ableiten?

Eindeutigkeit:

- ein ***Wort*** ist ***eindeutig***, wenn es genau einen Ableitungsbaum dafür gibt
- eine ***Syntax*** ist ***eindeutig***, wenn jedes Wort der Syntax eindeutig ist

Beachte: Reihenfolge der Ableitungsanwendung ist unerheblich

Beispiel: Betrachte folgende BNFs

Variante 1:

```
<FormelA> ::= <Term>
            | <FormelA> + <Term>
<Term>    ::= <Faktor>
            | <Term> * <Faktor>
<Faktor>  ::= a | b | c | ...| z
```

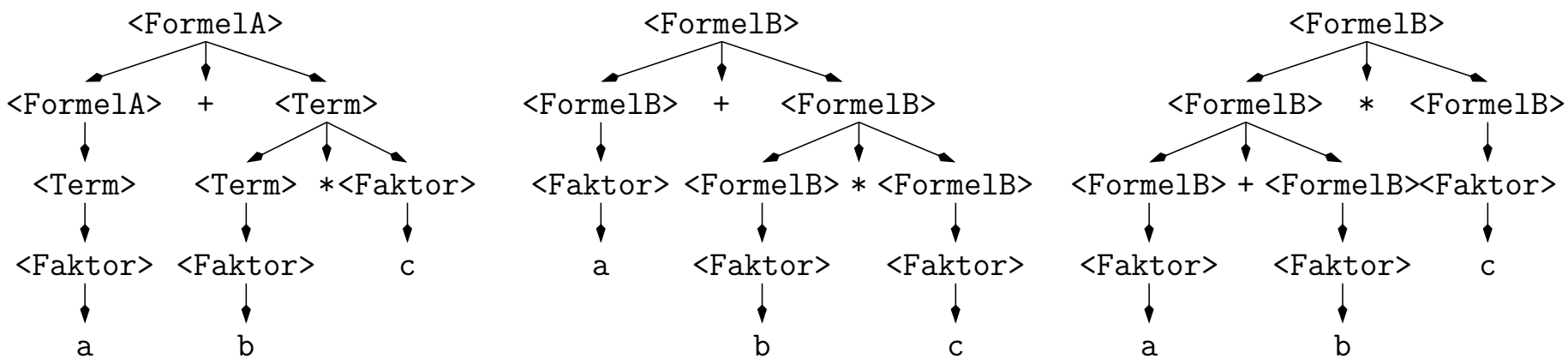
Variante 2:

```
<FormelB> ::= <Faktor>
            | <FormelB> + <FormelB>
            | <FormelB> * <FormelB>
<Faktor>  ::= a | b | c | ...| z
```

Eindeutigkeit von Wörtern und Syntax (Forts.)

Ableitungsbäume für die Formel `a + b * c`

Eindeutigkeit: Variante 1 ist eindeutig, während Variante 2 mehrdeutig ist:



Interpretation eines Ableitungsbaumes

- ❍ von Terminalen zur Wurzel (Bottom-Up)
- ❍ Nonterminal kann interpretiert werden, wenn alle Bestandteile ersetzt sind

⇒ unterschiedliche Ableitungsbäume liefern unterschiedliche Interpretationen derselben Zeichenfolge

Vergleich Ableitungssequenz und Ableitungsbaum

Beide Formen dienen u.a. dem „Nachweis“, dass ein Wort zur Sprache gehört

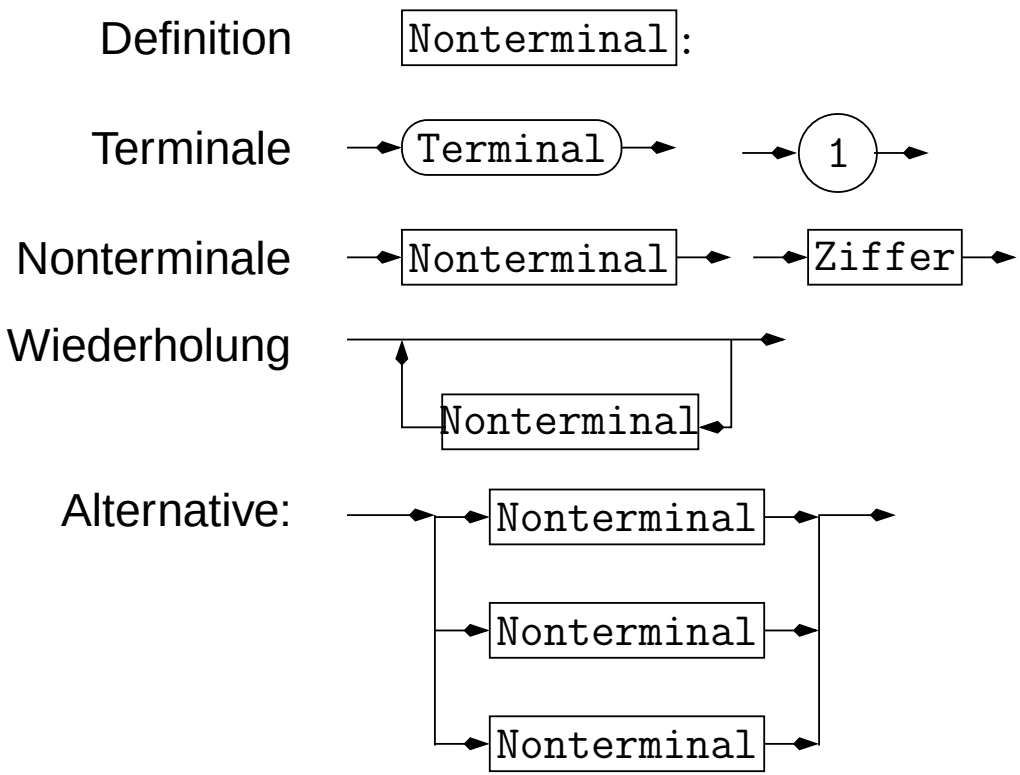
Vergleich der beiden Darstellungsformen

Eigenschaft	Ableitungssequenz	Ableitungsbaum
Reihenfolge d. Ableitungsschritte	erkennbar	nicht erkennbar
Transparenz der Ersetzung	gering	hoch
Bindungsregel erkennbar	nein	ja
Eignung für formalen Beweis	ja	eingeschränkt
Übersichtlichkeit	gering	hoch
Fehleranfälligkeit bei Ersetzung	hoch	gering
Kompaktheit der Darstellung	hoch	gering
Ablesen des Wortes	letztes Ableitungsele-ment	Terminalzeichen in „Blättern“ von links nach rechts

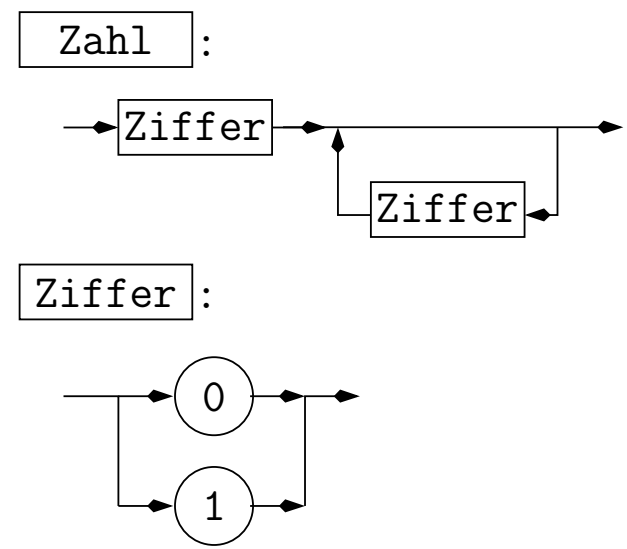
Syntaxdiagramm

Ziel: Graphische Darstellung der ***Regeln***

Verwendete Symbole:



Beispiel: Binärzahlen



Beachte: Wiederholung beinhaltet auch den optionalen Ausdruck

5.3 Beispiel: Arithmetik auf Ganzen Zahlen

Ganze Zahlen

Ziel: Syntax für die ganzen Zahlen $\ldots, -2, -1, 0, 1, 2, \ldots$

Vorüberlegung: Erstes Symbol ist $-, 0, \ldots, 9$, wobei 0 nur bei Zahl 0 auftritt

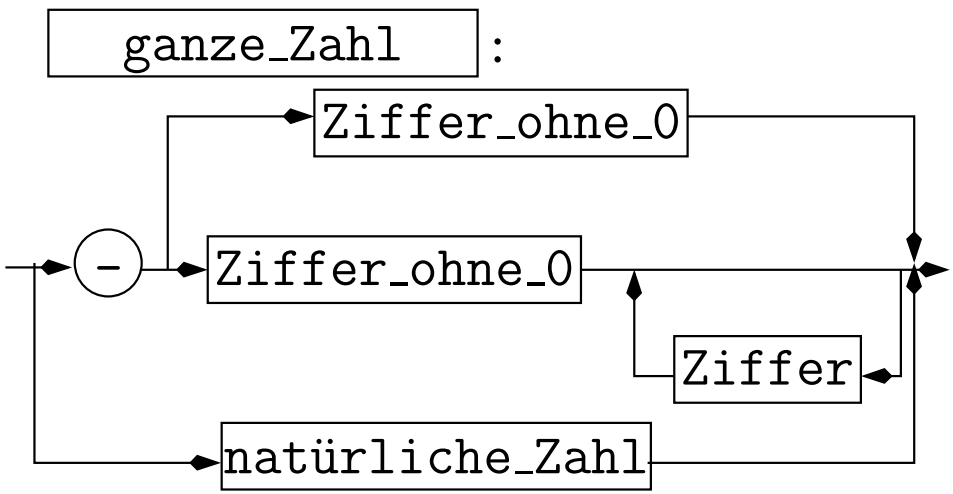
BNF: Startsymbol <ganze_Zahl>

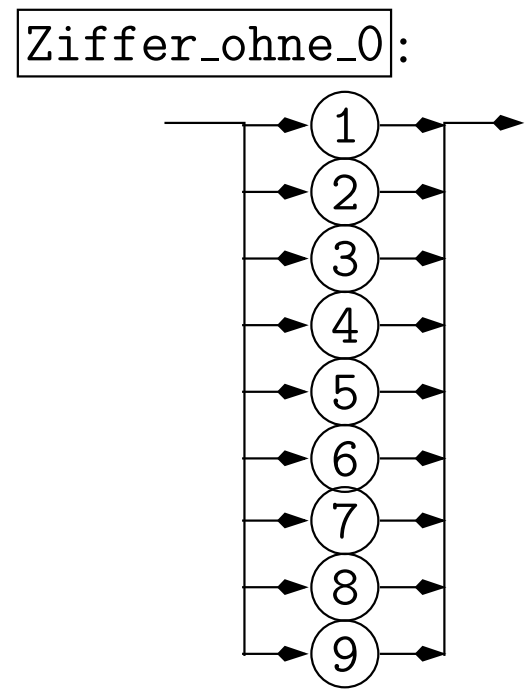
```
<ganze_Zahl>        ::= <natürliche_Zahl>
                      | - <Ziffer_ohne_0> { <Ziffer> }
                      | - <Ziffer_ohne_0>
<natürliche_Zahl>   ::= <Ziffer>
                      | <Ziffer_ohne_0> { <Ziffer> }
<Ziffer>            ::= 0 | <Ziffer_ohne_0>
<Ziffer_ohne_0>     ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

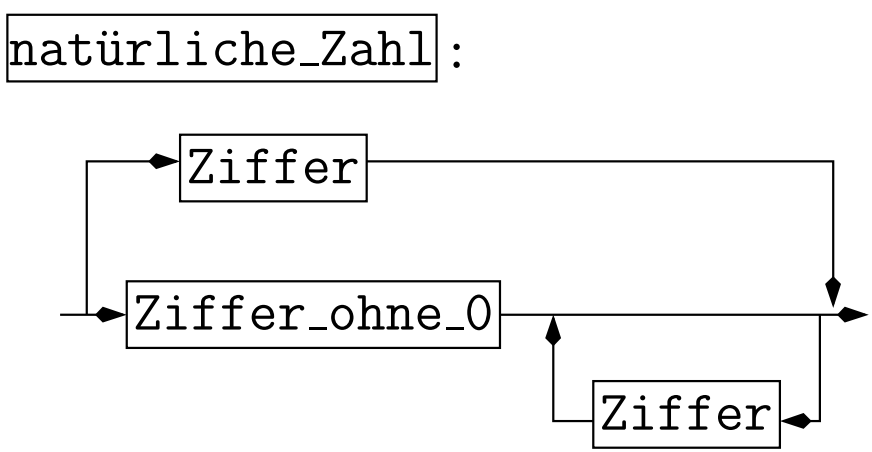
Beachte:
- ❍ <natürliche_Zahl> umfaßt die natürlichen Zahl mit der 0
- ❍ <ganze_Zahl> ::= - <natürliche_Zahl> | <natürliche_Zahl> wäre falsch!

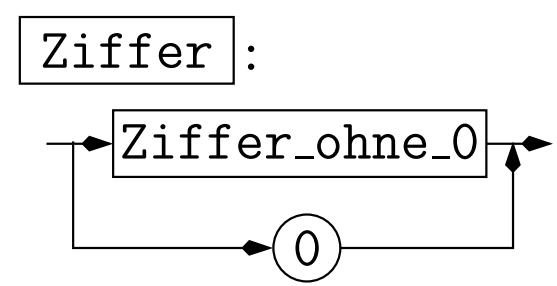
Ganze Zahlen (Forts.)

Syntaxdiagramm für Ganze Zahlen









Arithmetische Operatoren

Ziel: Operationen mit ganzen Zahlen: $*, /, +, -$ (Terminale)

BNF: Startsymbol `<Ausdruck>`

```
<Ausdruck> ::= <Term>
             | <Ausdruck> + <Term>
             | <Ausdruck> - <Term>
<Term>     ::= <Faktor>
             | <Term> * <Faktor>
             | <Term> / <Faktor>
<Faktor>   ::= <ganze_Zahl> | ( <Ausdruck> )
```

Beachte:

- Zweistufigkeit `<Ausdruck>`, `<Term>` realisiert Operatorpriorität:

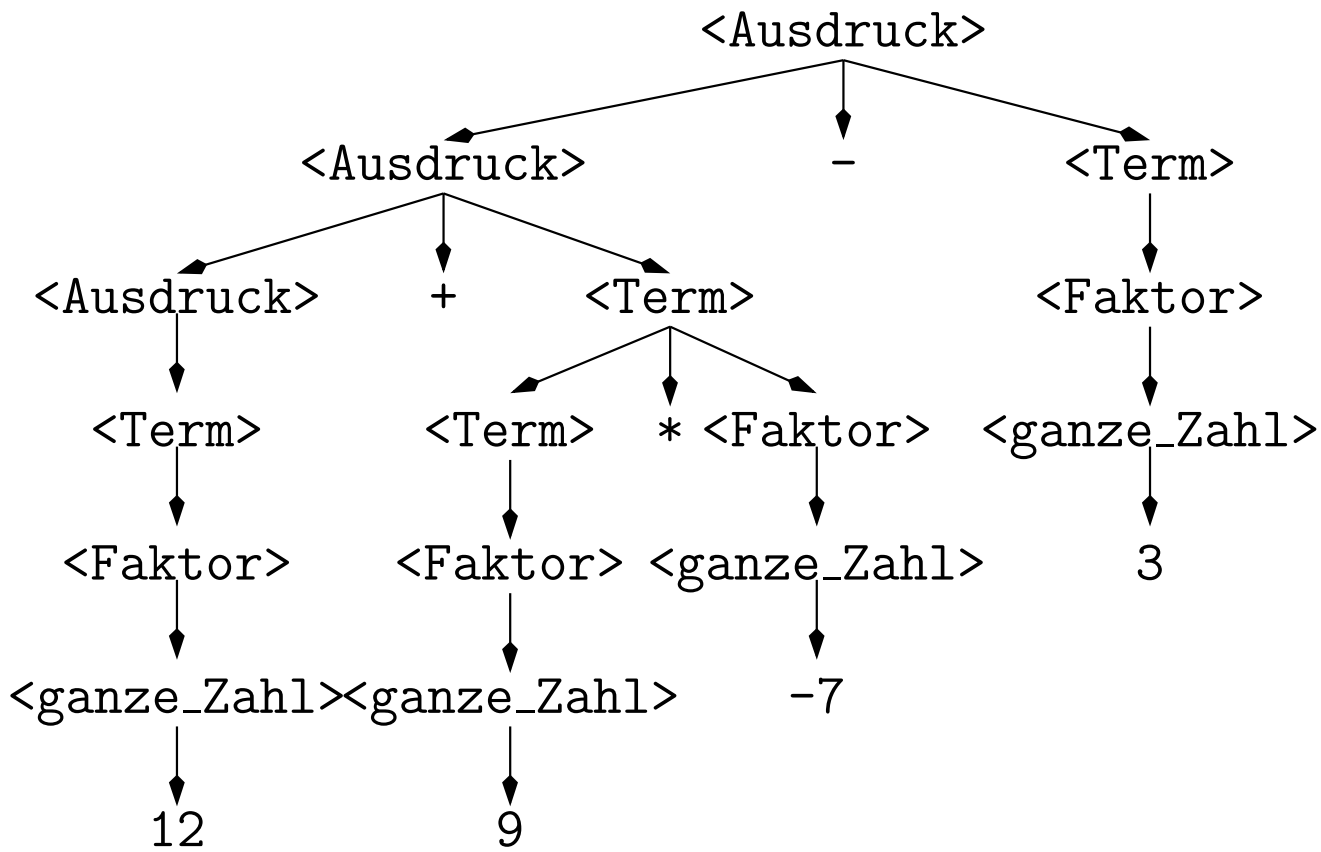
$$2 + 3 * 7 \quad \rightarrow \quad 2 + (3 * 7)$$

- `<Ausdruck>`-`<Term>`-Reihenfolge realisiert korrekte Linksbindung

$$24 - 3 - 9 \quad \rightarrow \quad ((24 - 3) - 9)$$ (analog für Division)

Arithmetische Operationen (Forts.)

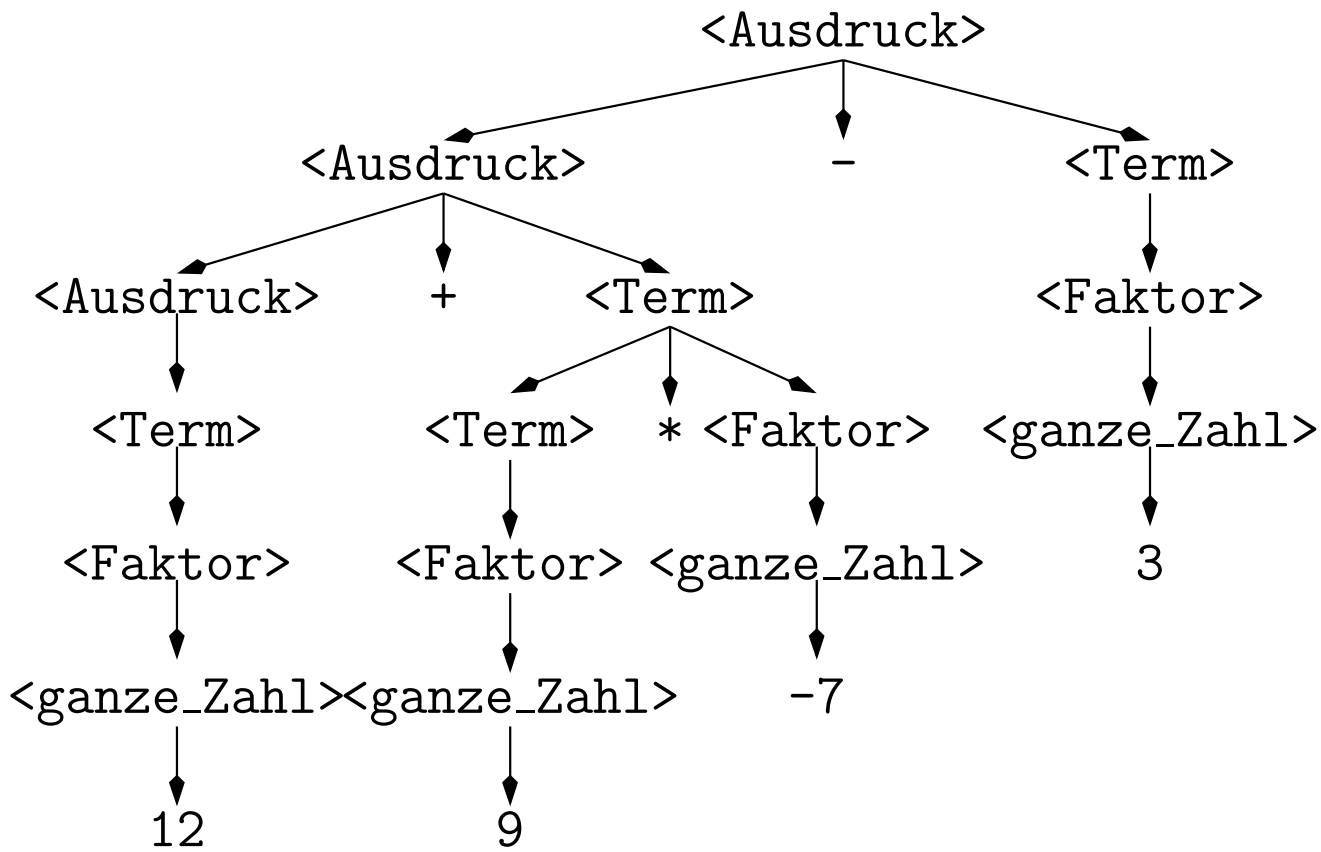
Beispiel: Ableitungsbaum von $12 + 9 * -7 - 3$



Arithmetische Operationen (Forts.)

Beispiel: Ableitungsbaum von $12 + 9 * -7 - 3$

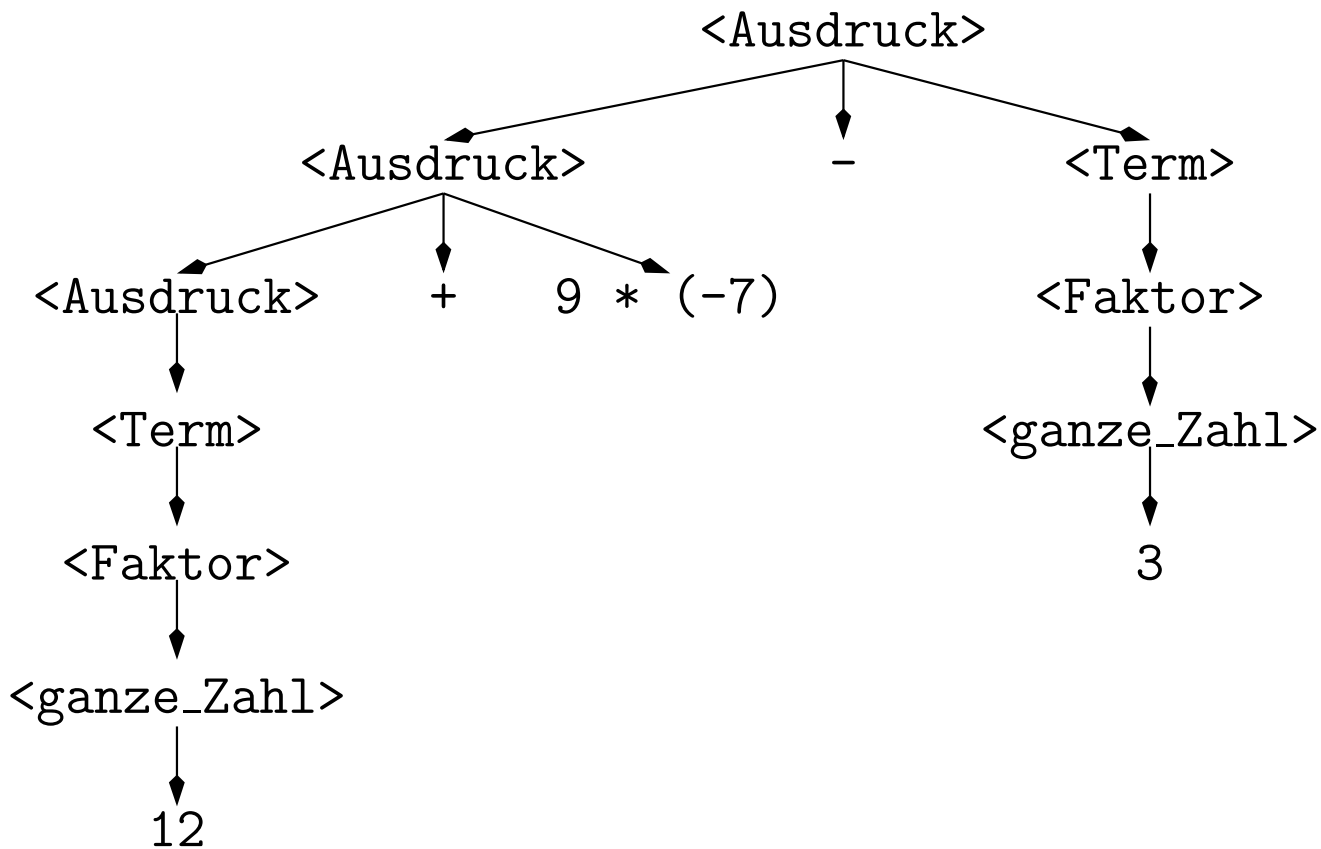
Interpretation des Ausdrucks $12 + 9 * -7 - 3$



Arithmetische Operationen (Forts.)

Beispiel: Ableitungsbaum von $12 + 9 * -7 - 3$

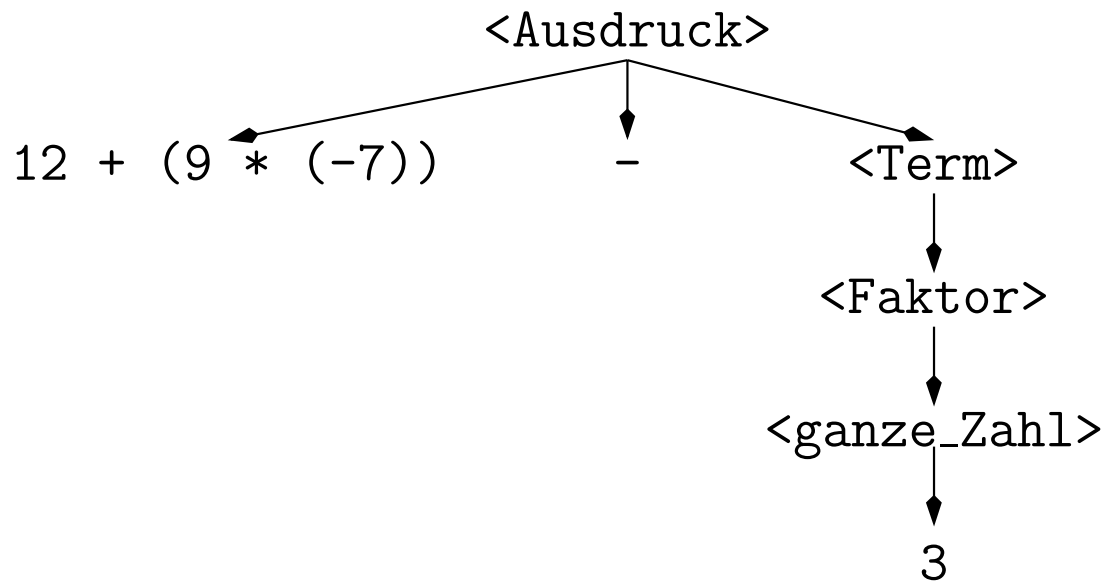
Interpretation des Ausdrucks $12 + 9 * -7 - 3$



Arithmetische Operationen (Forts.)

Beispiel: Ableitungsbaum von $12 + 9 * -7 - 3$

Interpretation des Ausdrucks $12 + 9 * -7 - 3$



Arithmetische Operationen (Forts.)

Beispiel: Ableitungsbaum von $12 + 9 * -7 - 3$

Interpretation des Ausdrucks $12 + 9 * -7 - 3$

```
(12 + (9 * (-7))) - 3
```

Frage: Wie werden Eingaben formaler Sprachen vom Rechner verarbeitet?

Syntax: Analyse des Zeichenstroms in zwei Schritten

1. ***Lexikalische Analyse:*** Analyse des Zeichenstroms
   - ❍ Ermittlung von Terminalsymbolen der Sprache
   - ❍ Zuordnung zu ***Tokens*** ($\approx$ elementare Nonterminale)
2. ***Syntaktische Analyse:*** Analyse der Tokensequenz entsprechend der Sprachsyntax

Semantik: Zuordnung syntaktisch korrekter Eingaben zu anwendungsspezifischen „Inhalten“ oder „Aktionen“; Beispiele:

- ❍ Festlegen von Attributen, z.B. in HTML

```
<table width="100%" height="100%">
```

  $\Rightarrow$ Tabelle, die Darstellungsbereich voll ausfüllt

- ❍ Zeichenoperationen, z.B. in PostScript

```
50 50 moveto 60 80 lineto
```

  $\Rightarrow$ Linie zwischen $(50, 50)$ und $(60, 80)$

Hinweis: Nutzung formaler Sprachen

Generell: Formale Sprachen dienen u.a.

- der strukturierten Speicherung von Eingabe- oder Ergebnisdaten
- der Beschreibung von Algorithmen z.B. in Programmiersprachen
- der Steuerung von Abläufen im Rechner und zwischen Rechnern

und finden sich in **allen Bereichen** der Informatik wieder

Speziell: Einige konkrete Nutzungsbeispiele sind

- Programmiersprachen, z.B. C, C++, Java
- Dokumentbeschreibungsformate, z.B. HTML, XML

Hilfsmittel zur Umsetzung einer Syntax-Analyse sind z.B.

- `lex, yacc` für allgemeine BNF-Darstellungen
- `expat, xerces` zur Verarbeitung von Daten im XML-Format