



Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

Christian Estler

Lektion 16: Die Syntax beschreiben



Sprachen, die andere Sprachen beschreiben, kennenlernen

Die Syntaxbeschreibung für Eiffel lesen und verstehen können

Einfache Syntaxbeschreibungen selbst erstellen



Eine Bedingungsanweisung besteht aus (in dieser Reihenfolge):

- Einem „If“-Teil der Form **if** *Bedingung*.
- Einem „Then“-Teil der Form **then** *Instruktion*.
- Null oder mehr „Elseif“-Teile, jeder der Form **elseif** *Bedingung* **then** *Instruktion*.
- Kein oder ein „Else-Teile“ der Form **else** *Instruktion*
- Dem Schlüsselwort **end**.

Hierbei ist jede *Bedingung* ein Boole'scher Ausdruck, und jedes *Instruktion* ist eine Verbunds-Instruktion.



Wir kennen Syntaxbeschreibungen aus natürlichen Sprachen:

- Z.B. Grammatik für Deutsch, Englisch, Französisch,...
- Gut genug für den menschlichen Gebrauch
- Uneindeutig, wie die natürliche Sprache selbst



I cdnoul blvelee taht I cluod aulacity uesdnatnrd waht I was rdgnieg. The Paomnehal Pweor of the Hmuan Mnid Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, is deosn't mtttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be in the rghit pclae. The rset can be a taotl mse and you can sitll raed it wouthit any porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Ptrety Amzanig Huh?

Compiler benutzen Algorithmen, um

- Die Gültigkeit des Programmtextes zu überprüfen
- Den Programmtext zu analysieren um Elemente für einen abstrakten Syntaxbaum zu extrahieren
- Den Programmtext in Maschineninstruktionen zu übersetzen



Compiler brauchen eine strikte formale Definition einer Programmiersprache



Benutzen Sie eine formale Sprache, um Programmiersprachen zu beschreiben.

Sprachen, die andere Sprachen beschreiben, heissen **Meta-Sprachen**

Die Meta-Sprache, die Eiffel beschreibt:

BNF-E (Variante der Backus-Naur-Form, BNF)



1954 FORTRAN: Erste weitgehend bekannte Programmiersprache (entwickelt von John Backus et Al.)

1958 ALGOL 58: Zusammenarbeit von europäischen und amerikanischen Gruppen

1960 ALGOL 60: Die Vorbereitung zeigte den Bedarf einer formalen Beschreibung auf → John Backus (ALGOL Team) schlug die Backus-Normal-Form (BNF) vor

1964: Donald Knuth schlug vor, Peter Naur für sein Mitwirken zu ehren → Backus-Naur-Form

Viele weitere Varianten seither, z.B. die graphische Variante von Niklaus Wirth



Mit BNF kann man **syntaktische** Eigenschaften einer Sprache beschreiben:

- Zulässige Struktur einer Sprache
- Ähnlich Grammatiken in normaler Sprache

Erinnerung: Die Beschreibung einer Programmiersprache beinhaltet auch **lexikalische** und **semantische** Eigenschaften
→ Andere Werkzeuge

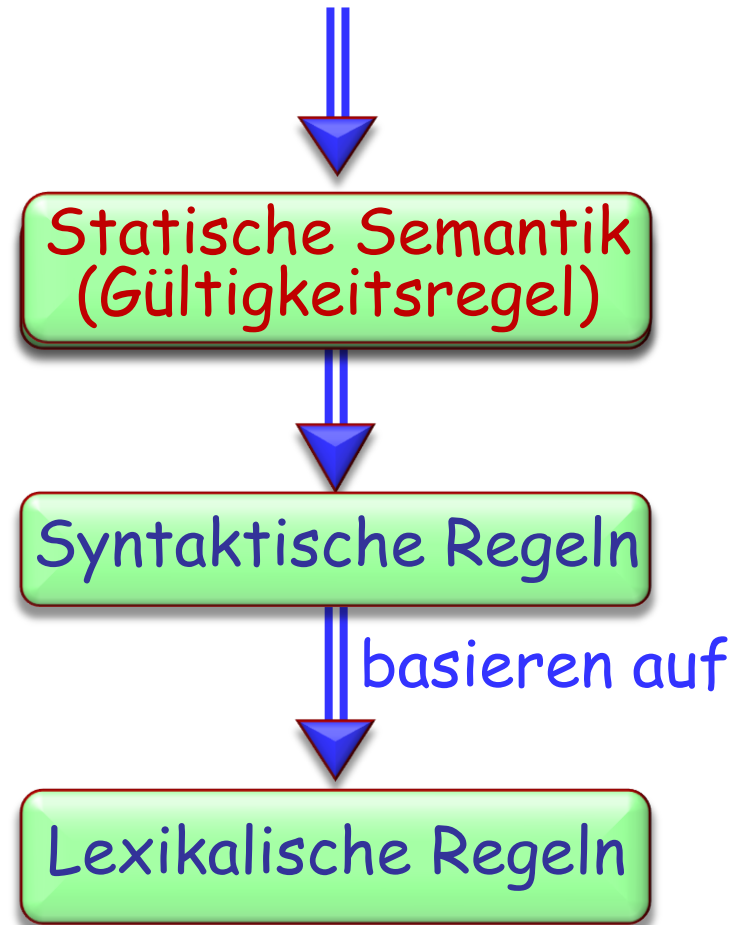
(Erinnerung: von Lektion 3)



Semantische Regeln definieren den Effekt eines Programms, das den syntaktischen Regeln genügt

Syntaktische Regeln definieren, wie man Exemplare aus Tokens, die den lexikalischen Regeln genügen, herstellt

Lexikalische Regeln definieren, wie man aus Zeichen Tokens macht





Eine Sprache ist eine Menge von Phrasen

Eine Phrase ist eine endliche Sequenz von Zeichen (Tokens) eines gewissen „Vokabulars“

Nicht jede mögliche Sequenz ist eine Phrase der Sprache

Eine Grammatik spezifiziert, welche Sequenzen Teil der Sprache sind und welche nicht.

BNF wird benutzt, um eine **Grammatik** für eine Programmiersprache zu definieren.

Beispiele von Phrasen



```
class PERSON
feature
    age: INTEGER
        -- Alter
end
```

```
class
    age: INTEGER
        -- Alter
end PERSON
feature
```



Definition

Eine **Grammatik** für eine Sprache ist eine endliche Menge von Regeln zum Erstellen von (Token) Sequenzen, so dass gilt:

1. Jede Sequenz, die man durch endlich häufiges Anwenden von Regeln der Grammatik erhält, ist eine Phrase der Sprache
2. Jede Phrase der Sprache kann durch eine endliche Anzahl von Anwendungen der Grammatik-Regeln erzeugt werden



Terminale

Zeichen der Sprache, die nicht durch eine Produktion der Grammatik definiert sind.
Z.B. Schlüsselworte von Eiffel wie **if**, **then**, **end**
oder Symbole wie das Semikolon ";" oder die
Zuweisung " := "



Nonterminale

Namen von syntaktischen Strukturen oder Unterstrukturen, die benutzt werden, um Phrasen zu erstellen



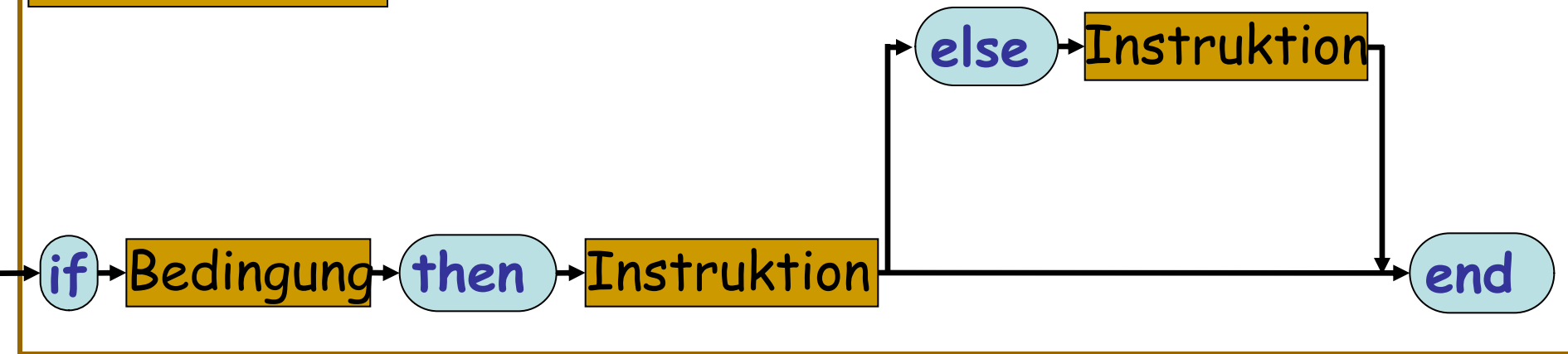
Produktionen

Regeln, die durch eine Kombination von Terminalen und (anderen) Nonterminalen die Nonterminale einer Grammatik definieren

Eine Beispielsproduktion



Konditional:



Terminal

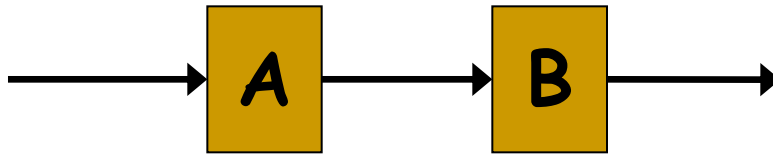


Nonterminal



Produktion

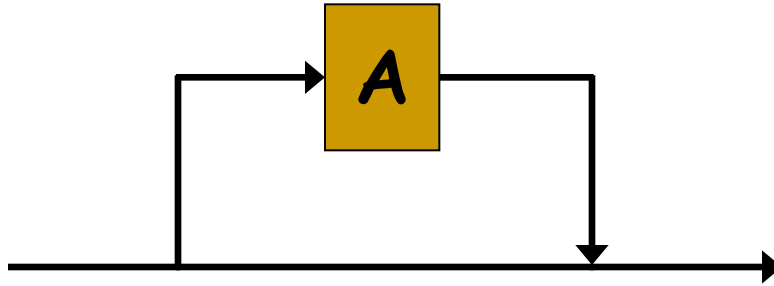
Graphische Repräsentation:



BNF: $A B$

Bedeutung: A gefolgt von B

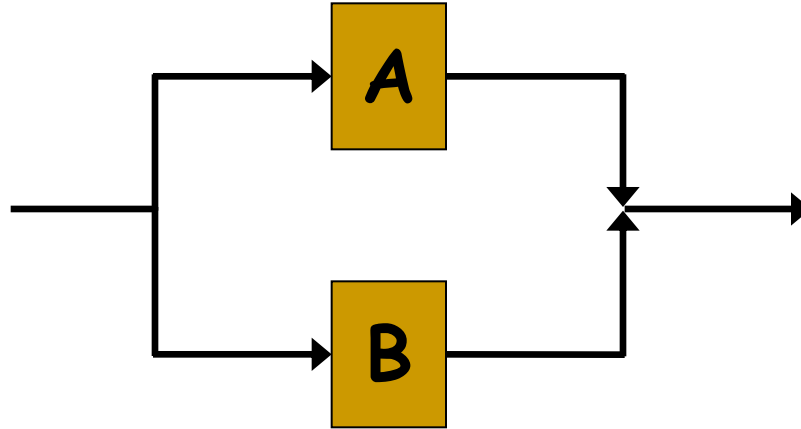
Graphische Repräsentation:



BNF: [A]

Bedeutung: A oder nichts

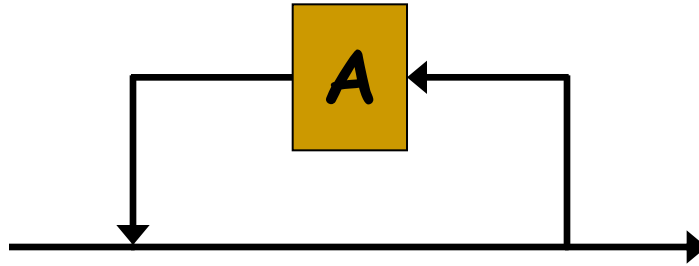
Graphische Repräsentation:



BNF: $A \mid B$

Bedeutung: entweder A oder B

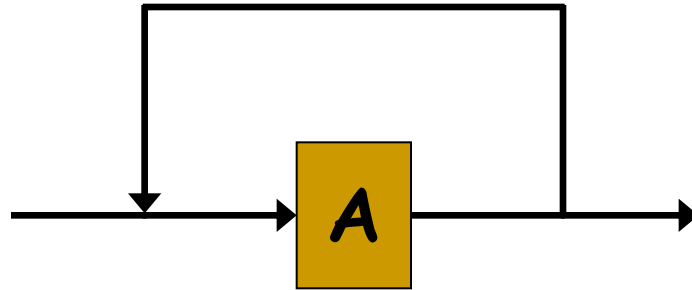
Graphische Repräsentation:



BNF: $\{ A \}^*$

Bedeutung: Sequenz von null oder mehreren A

Graphische Repräsentation:



BNF: $\{ A \}^+$

Bedeutung: Sequenz von einem oder mehreren A

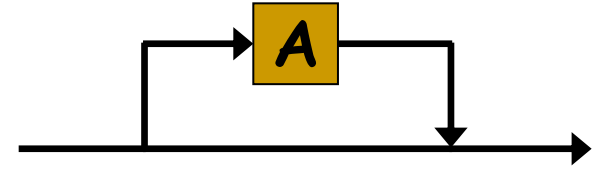
BNF Elemente: Übersicht



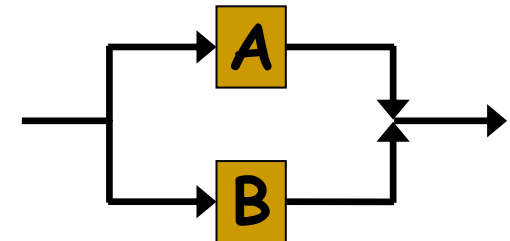
Verkettung: $A B$



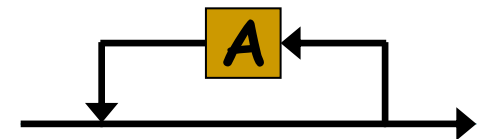
Optional: $[A]$



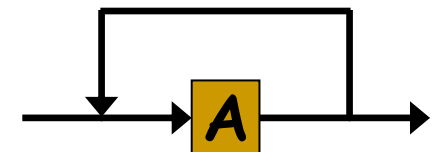
Wahl: $A | B$



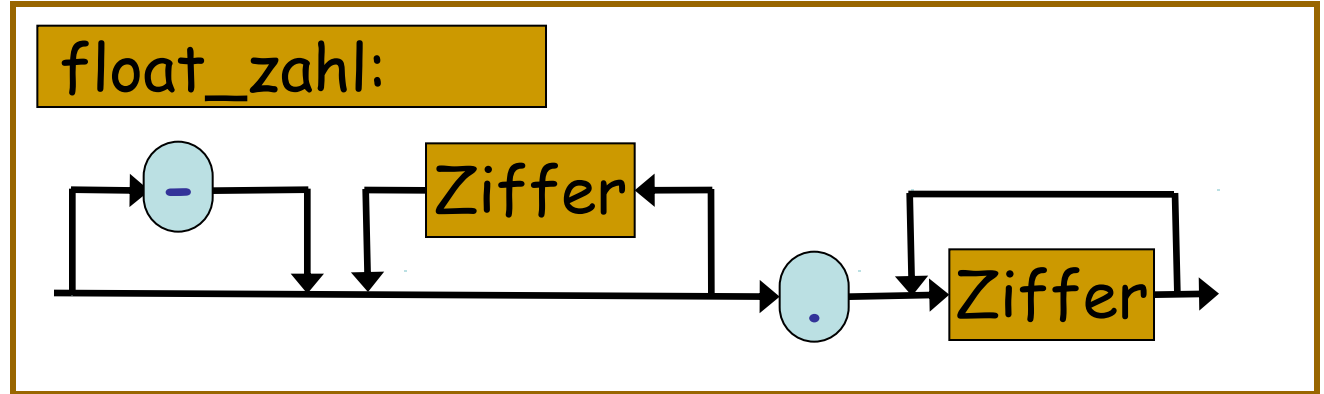
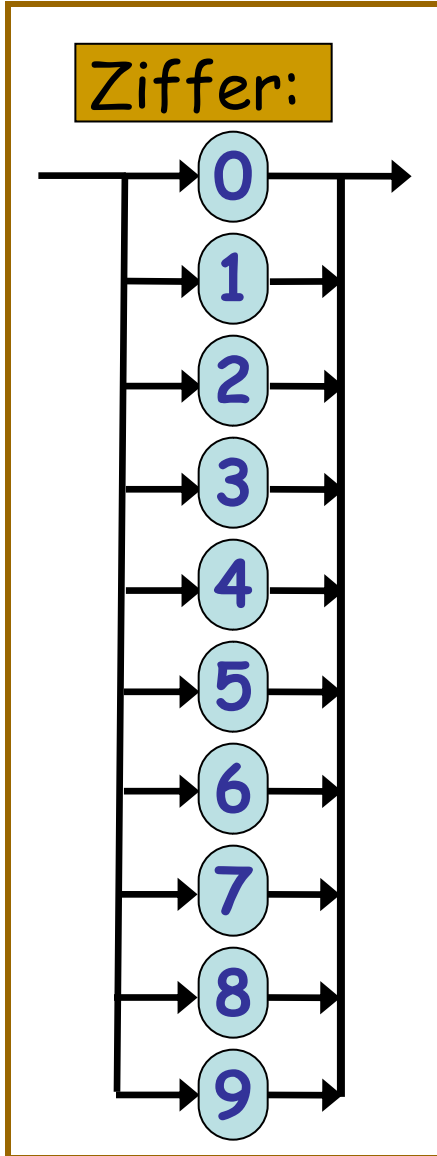
Repetition (0 oder mehr): $\{ A \}^*$



Repetition (mind. einmal): $\{ A \}^+$



Ein einfaches Beispiel



Beispielphrasen:

.76

-.76

1.56

12.845

-1.34

13.0

Übersetzen Sie es in die schriftliche Form!

Ein einfaches Beispiel

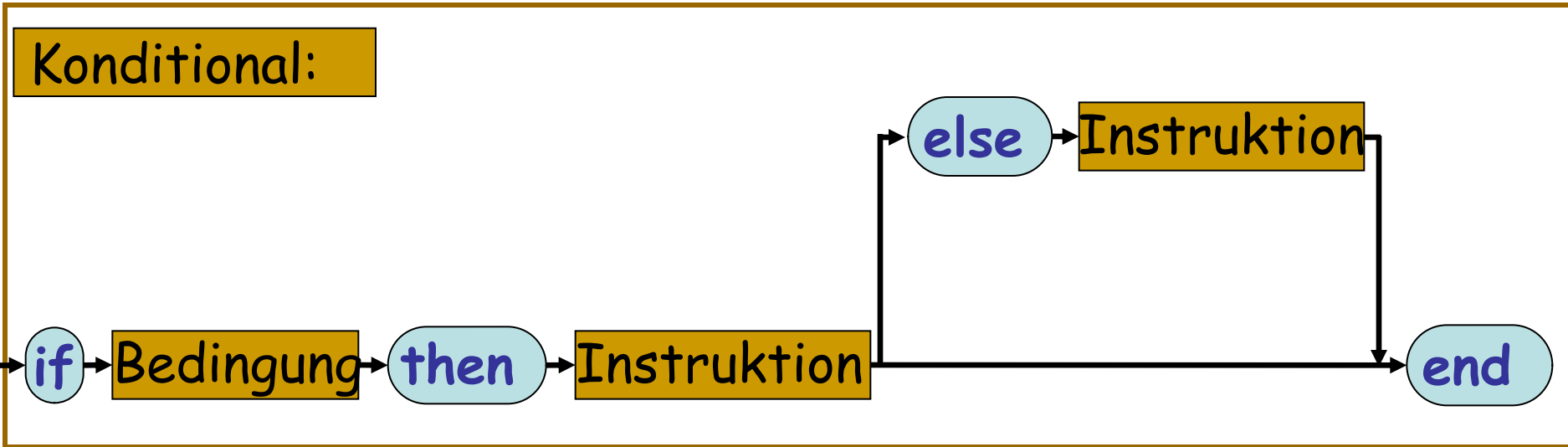


In BNF:

Ziffer $\hat{=}$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

float_zahl $\hat{=}$ [-] { **Ziffer** }* . { **Ziffer** }+

BNF Elemente kombiniert



In BNF geschrieben:

Konditional $\hat{=}$

if **Bedingung** **then** **Instruktion** [**else** **Instruktion**] **end**

Konditional \triangleq if Then_teil_liste [Else_teil] end

Then_teil_liste \triangleq Then_teil { elseif Then_teil }*

Then_teil \triangleq Boolescher_ausdruck then Verbund

Else_teil \triangleq else Verbund

Andere Grammatik für Konditional



Konditional $\hat{=}$ If_teil Then_teil Else_liste end

If_teil $\hat{=}$ if Boolescher_ausdruck

Then_teil $\hat{=}$ then Verbund

Else_liste $\hat{=}$ { Elseif_teil }* [else Verbund]

Elseif_teil $\hat{=}$ elseif Boolescher_ausdruck Then_teil

Satz	\triangleq	I [don't] Verb Namen Quant
Namen	\triangleq	Name {and Name}*
Name	\triangleq	tomatoes shoes books football
Verb	\triangleq	like hate
Quant	\triangleq	a lot a little

Welche der folgenden Phrasen sind korrekte Sätze?

I like tomatoes and football

I don't like tomatoes a little

I hate football a lot

I like shoes and tomatoes a little

I don't hate tomatoes, football and books a lot

Schreiben Sie die BNF um, damit sie auch die inkorrekten Phrasen beinhaltet



Welche der folgenden Sätze sind korrekt?

- I like tomatoes and football
- ✓ I don't like tomatoes a little
- ✓ I hate football a lot
- ✓ I like shoes and tomatoes a little
- I don't hate tomatoes, football and books a lot

Schreiben Sie die BNF um, damit sie auch die inkorrekten Phrasen beinhaltet

Satz	$\hat{=}$	I [don't] Verb Namen [Quant]
Namen	$\hat{=}$	Name [{ , Name } * and Name]
Name	$\hat{=}$	tomatoes shoes books football
Verb	$\hat{=}$	like hate
Quant	$\hat{=}$	a lot a little

Wird in der offiziellen Beschreibung von Eiffel benutzt.
Jede Produktion ist eine der folgenden

Verkettung

$$A \triangleq BC[D]$$

Wahl

$$A \triangleq B | C | D$$

Repetition

$$A \triangleq \{ B \text{ delimiter } \dots \}^*$$

$$A \triangleq \{ B \text{ delimiter } \dots \}^+$$

Interpretiert als

$$A \triangleq [B \{ \text{delimiter } B \}^*]$$

Interpretiert als

$$A \triangleq B \{ \text{delimiter } B \}^*$$



- Jedes Nonterminal muss auf der linken Seite von genau **einer** Produktion auftreten. Diese Produktion ist seine **definierende Produktion**.
- Jede Produktion ist von **einer** Art: Verkettung, Wahl oder Repetition



Konditional \triangleq if Then_teil_liste [Else_teil] end

Then_teil_liste \triangleq Then_teil { elseif Then_teil }*

Then_teil \triangleq Boolescher_ausdruck then Verbund

Else_teil \triangleq else Verbund

Konditional \triangleq **if** Then_teil_liste [Else_teil] **end**

Then_teil_liste \triangleq { Then_teil **elseif** ... }⁺

Then_teil \triangleq Boolescher_ausdruck **then** Verbund

Else_teil \triangleq **else** Verbund



Konstrukte können verschachtelt sein

In BNF wird dies mit **rekursiven Grammatiken** ausgedrückt.

Rekursion: zirkuläre Abhängigkeiten von Produktionen

Konditionale können in anderen Konditionalen verschachtelt sein:

Else_teil $\hat{=}$ else Verbund

Verbund $\hat{=}$ { Instruktion ; ... }^{*}

Instruktion $\hat{=}$ Konditional | Schleife | Aufruf | ...

Der Produktionsname kann in der eigenen Definition vorkommen

Definition von `Then_teil_liste` mit Repetition:

`Then_teil_liste` $\hat{=}$ { `Then_teil` `elseif` ... }*

Rekursive Definition von `Then_teil_liste`:

`Then_teil_liste` $\hat{=}$ `Then_teil` [`elseif` `Then_teil_liste`]

Konditional



```
if a = b then
    a := a - 1
    b := b + 1
elseif a > b then
    a := a + 1
else
    b := b + 1
end
```

Konditional $\hat{=}$ (if) Then_teil_liste [Else_teil] (end)

Then_teil_liste $\hat{=}$ { Then_teil (elseif) ... }⁺

Then_teil $\hat{=}$ Boolescher_ausdruck (then) Verbund

Else_teil $\hat{=}$ (else) Verbund

BNF für einfache arithmetische Ausdrücke



Nehmen Sie an, Number ist als positiver Integer definiert, und Variable aus einem Buchstaben besteht

Keine eingeklammerte Ausdrücke

Beispiele von Ausdrücken:

a a + b a - b a * 7 + b

Brauchen wir eine rekursive Grammatik?

Nein:

Expr	$\hat{=}$	Factor {Operator Factor}*
Factor	$\hat{=}$	Number Variable
Operator	$\hat{=}$	+ - * /

BNF für einfache arithmetische Ausdrücke



Jetzt erlauben wir Klammern; Beispiele von Ausdrücken:

a $a + b$ $a - b$ $a * 7 + b$

$7 / ((a * (b + 12)) - c)$

$7 / ((a * b + 12) - c)$

Brauchen wir eine rekursive Grammatik?

Ja, z.B.:

Expr	\triangleq	Term {Operator Term}*
Term	\triangleq	Number Variable Nested
Nested	\triangleq	(Expr)
Operator	\triangleq	+ - * /

Ist das eine "gute" Grammatik?

Nein: sie entspricht nicht der Semantik; siehe z.B. den Unterschied zwischen

$a * b + 12$

$a + b * 12$

BNF für einfache arithmetische Ausdrücke



Eine bessere rekursive Grammatik:

Expr	$\hat{=}$	Term { Add_op Term }*
Term	$\hat{=}$	Factor { Mult_op Factor }*
Factor	$\hat{=}$	Number Variable Nested
Nested	$\hat{=}$	(Expr)
Add_op	$\hat{=}$	+ -
Mult_op	$\hat{=}$	* /

Welche der folgenden Phrasen sind korrekt?

a ✓

a + b ✓

-a + b -

a * 7 + b ✓

7 / (3 * 12) - 7 ✓

(3 * 7) ✓

(5 + a (7 * b)) -



Welche der folgenden Phrasen sind korrekt?

a ✓

a + b ✓

-a + b -

a * 7 + b ✓

7 / (3 * 12) - 7 ✓

(3 * 7) ✓

(5 + a (7 * b)) -

Expr	\triangleq	{Term Add_op ...} ⁺
Term	\triangleq	{Factor Mult_op ...} ⁺
Factor	\triangleq	Number Variable Nested
Nested	\triangleq	(Expr)
Add_op	\triangleq	+ -
Mult_op	\triangleq	* /



Halten Sie Produktionen kurz.

Einfacher zu lesen

Bessere Bewertung der Sprachengrösse

~~Konditional $\hat{=}$~~

~~if Boolescher_ausdruck then Verbund
{ elseif Boolescher_ausdruck then Verbund }*
[else Verbund] end~~

Behandeln Sie **lexikale Konstrukte** wie Terminale

Bezeichner

Konstante Werte

Identifizier $\hat{=} \text{Letter} \{ \text{Letter} \mid \text{Digit} \mid \text{"_"} \}^*$

Integer_constant $\hat{=} [-] \{ \text{Digit} \}^+$

Floating_point $\hat{=} [-] \{ \text{Digit} \}^* \text{"."} \{ \text{Digit} \}^+$

Letter $\hat{=} \text{"A"} \mid \text{"B"} \mid \dots \mid \text{"Z"} \mid \text{"a"} \mid \dots \mid \text{"z"}$

Digit $\hat{=} \text{"0"} \mid \text{"1"} \mid \dots \mid \text{"9"}$



Benutzen Sie eindeutige Produktionen

Eine anwendbare Produktion kann so durch Anschauen von einem lexikalen Element pro Mal gefunden werden

Konditional \triangleq `if Then_teil_liste [Else_teil] end`

Verbund \triangleq `{ Instruktion }*`

Instruktion \triangleq `Konditional | Schleife | Aufruf | ...`



Die Syntax von Eiffel ist in BNF-E geschrieben

- Eine Produktion pro Nonterminal
- Jede Produktion ist entweder eine Verkettung, eine Wahl oder eine Repetition
- Spezielle Semantik der Repetition
- Rekursion ist erlaubt
- **Terminale (lexikale) Konstrukte** benutzen nicht BNF-E für ihre Beschreibung
 - Reservierte Wörter (z.B. **if**, **end**, **class**)
 - manifeste Konstanten (**237**, **-12.93**)
 - Symbole (**+**, **:**)
 - Bezeichner (**LINKED_LIST**, **put**)



Lexikale Konstrukte werden mit einer BNF-ähnlichen *regulären Grammatik* beschrieben

- Mischen von Produktionstypen erlaubt
- Benutzen Sie Klammern für Eindeutigkeit, z.B. **Letter** (**Letter** | **Digit** | **Underscore**)*
- Keine Rekursion
- Benutzt Symbole und Zeichenintervalle, z.B. 'a'..'z'
- Einfache Repetitionsregeln (siehe BNF)
- Bei der Verkettungen müssen die Elemente nicht (lexikalisch) getrennt sein

Definieren Sie eine rekursive Grammatik in BNF-E für Boole'sche Ausdrücke mit der folgenden Beschreibung:
Einfache Ausdrücke, beschränkt auf die Variablenbezeichner **x**, **y**, oder **z**, die, neben Klammerung, als Operationen das unäre **not** und die binären **and**, **or**, und **implies** beinhalten können

Gültige Phrasen wären z.B.

not x and not y

(x or y implies z)

y or (z)

(x)



B_expr \triangleq With_par | Expr
With_par \triangleq "(" Expr ")"
Expr \triangleq Not_term | Bin_term | Variable
Bin_term \triangleq B_expr Bin_op B_expr
Bin_op \triangleq "implies" | "or" | "and"
Not_term \triangleq "not" B_expr
Variable \triangleq "x" | "y" | "z"

Bemerkung: hier brauchen wir "x" um Terminale zu bezeichnen



Ein Feature pro Produktion

Verkettung:

Sequenz von Feature-Aufrufen für Nonterminale,
überprüft auf Terminale

Wahl:

Konditional mit Verbund pro Alternative

Repetition:

Schleife



Automatische Generierung von abstrakten Syntaxbäumen für Phrasen

Basiert auf **BNF-E**

Eine Klasse pro Produktion

Die Klassen erben von vordefinierten Klassen:
AGGREGATE, CHOICE, REPETITION, TERMINAL

Das Feature **production** definiert eine Produktion



Yoc:

Übersetzt BNF-E zu EiffelParse-Klassen

Yacc / Bison:

Übersetzt BNF zu C-Parser



DTD: Beschreibung von XML-Dokumenten

```
<!ELEMENT collection (recipe*)>
<!ELEMENT recipe (title, ingredient*, preparation)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT ingredient (ingredient*, preparation)?>
<!ATTLIST ingredient name CDATA #REQUIRED
                    amount CDATA #IMPLIED
                    unit CDATA #IMPLIED>
<!ELEMENT preparation (step*)>
<!ELEMENT step (#PCDATA)>
```



Unix/Linux: Übersicht der Kommandos

SYNOPSIS

```
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C  
config_file] [-M pathlist] [-P pager] [-S section_list]  
[section] name ...
```



<http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-367.pdf>

<http://www.gobosoft.com/eiffel/syntax/>



Eine Art Syntax zu beschreiben: BNF

3 Varianten: BNF, BNF-E, graphisch

Einen weiteres Beispiel für die Rekursion