ETH Zürich
Chair of Software Engineering

Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

Christian Estler

Lektion 16: Die Syntax beschreiben

Ziele der heutigen Vorlesung

Sprachen, die andere Sprachen beschreiben, kennenlernen

Die Syntaxbeschreibung für Eiffel lesen und verstehen können

Einfache Syntaxbeschreibungen selbst erstellen

Syntax: Konditional

Eine Bedingungsinstruktion besteht aus (in dieser Reihenfolge):

Einem „If-Teil" der Form **if** *condition*.

Einem „Then-Teil" der Form **then** *compound*.

Null oder mehr „Elseif-Teile", jeder der Form **elseif** *condition* **then** *compound*.

Null oder mehr „Else-Teile" der Form **else** *compound*

Dem Schlüsselwort **end**.

Hierbei ist jede *condition* ein Boole'scher Ausdruck, und jeder *compound* ist eine Verbunds-Instruktion.

Wieso Syntax formal beschreiben?

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.

Die Macht des menschlichen Gehirns

I cdnoult blvelee taht I cluod aulacity uesdnatnrd waht I was rdgnieg. The Paomnnehal Pweor of the Hmuan Mnid Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, is deosn't mttaer 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 mses 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?

Wieso Syntax formal beschreiben?

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

Formale Beschreibung der Syntax

Benutzen Sie eine formale Sprache, um Programmiersprachen zu beschreiben.

Sprachen, die andere Sprachen beschreiben, heissen Meta-Sprachen

Meta-Sprache, die Eiffel beschreibt:

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

Geschichte

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

Formale Beschreibung einer Sprache

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

Formale Beschreibung der Syntax

Eine Sprache ist eine Menge von Phrasen

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

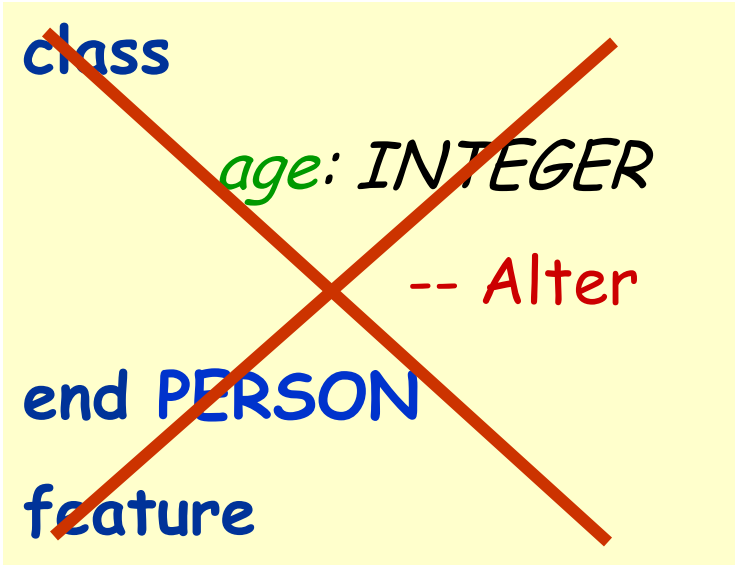
Nicht jede mögliche Sequenz ist eine Phrase der Sprache

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
```



Grammatik

Definition

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

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.

Elemente von Grammatiken: Terminale

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 "**:=**"

Elemente einer Grammatik: Nonterminale

Nonterminale

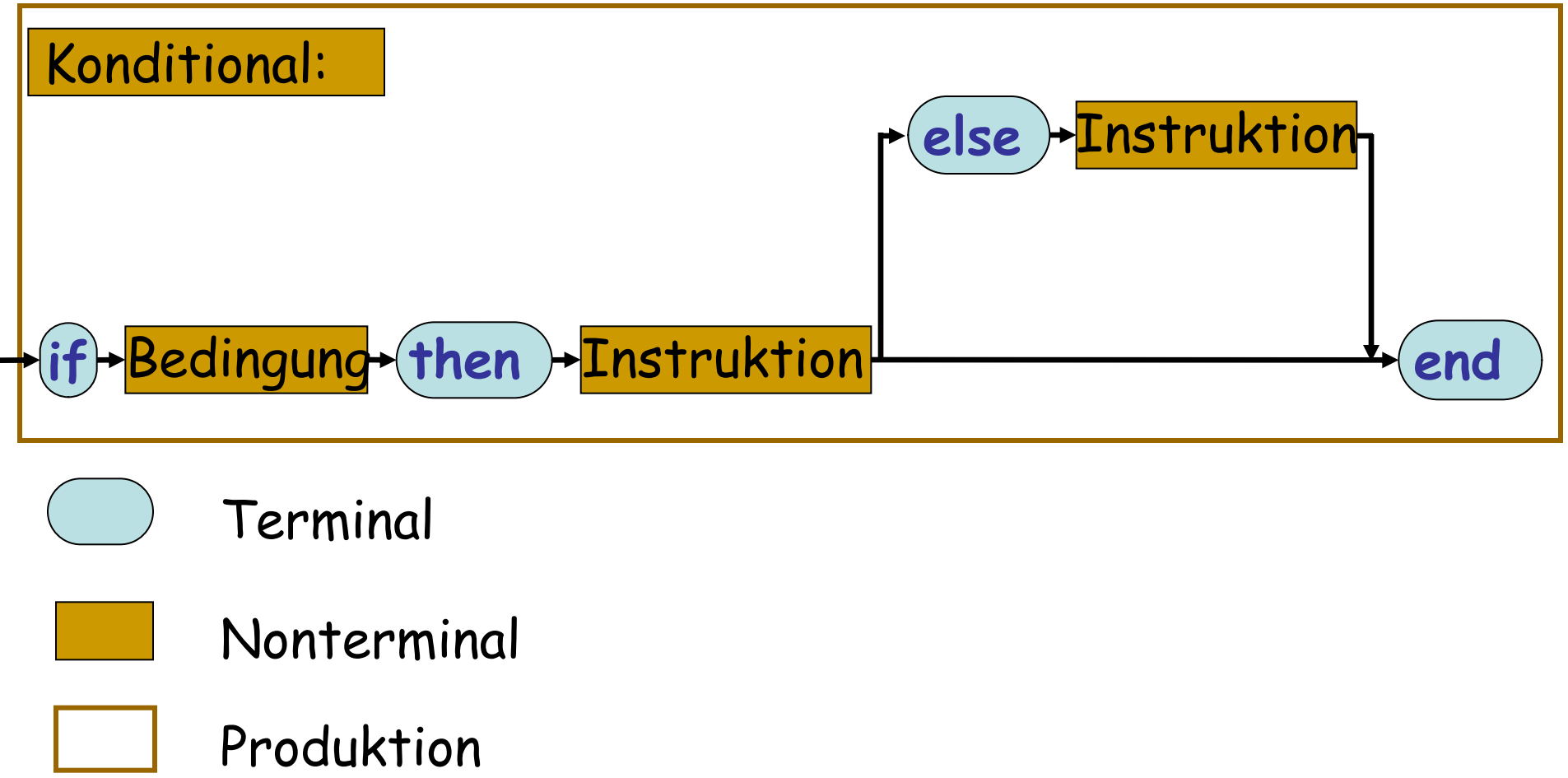
Namen von syntaktischen Strukturen oder Unterstrukturen, die benutzt werden, um Phrasen zu erstellen.

Elemente einer Grammatik: Produktionen

Produktionen

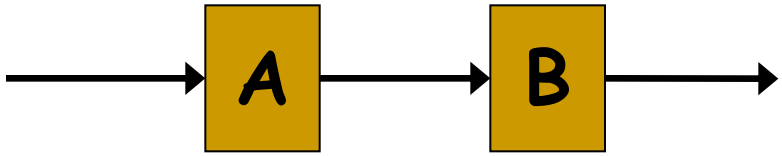
Regeln, die durch eine Kombination von Terminalen und (anderen) Nonterminalen, die Nonterminale einer Grammatik definieren

Eine Beispielsproduktion



BNF Elemente: Konkatenation

Graphische Repräsentation:

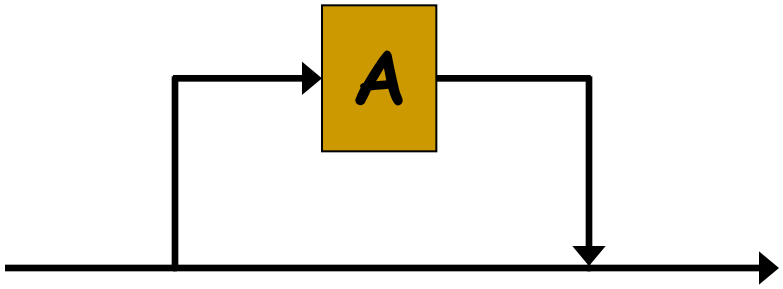


BNF: A B

Bedeutung: A gefolgt von B

BNF Elements: Optional

Graphische Repräsentation:

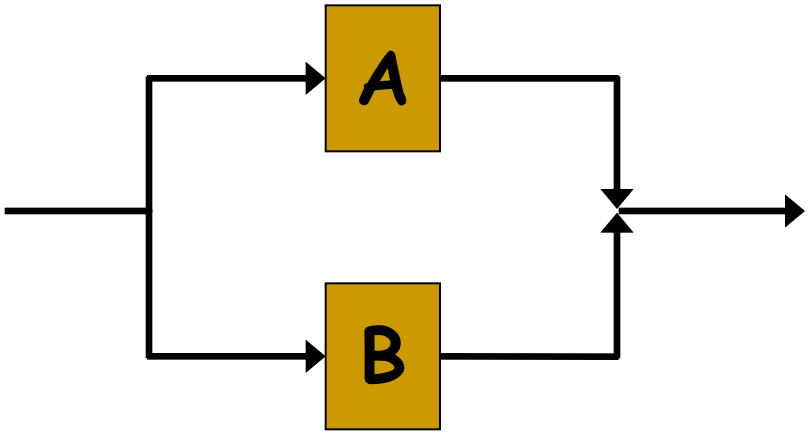


BNF: [A]

Bedeutung: A oder nichts

BNF Elements: Wahl

Graphische Repräsentation:

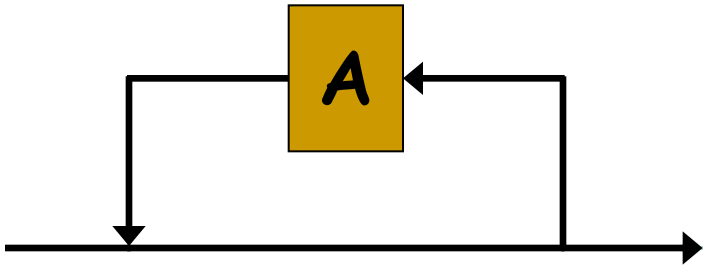


BNF: A | B

Bedeutung: entweder A oder B

BNF Elements: Repetition

Graphische Repräsentation:

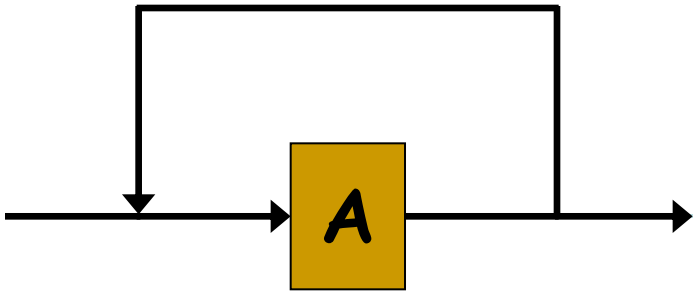


BNF: { *A* }*

Bedeutung: Sequenz von null oder mehreren *A*

BNF Elements: Repetition, einmal oder mehr

Graphische Repräsentation:



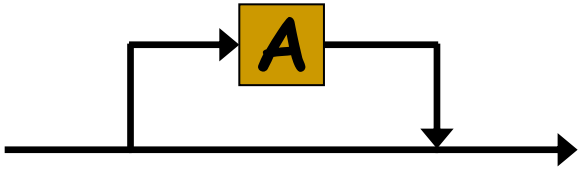
BNF: $\{ A \}^{+}$

Bedeutung: Sequenz von einem oder mehreren *A*

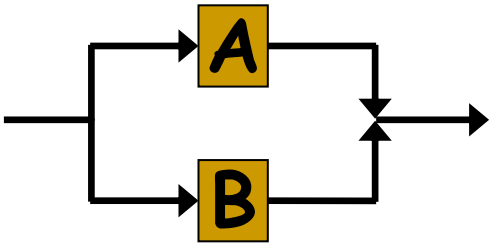
BNF Elemente: Übersicht

Konkatenation: A B

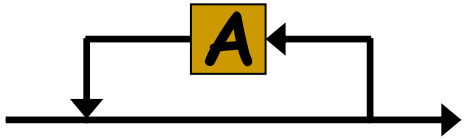
Optional: [A]



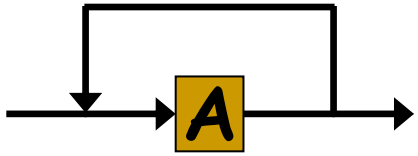
Wahl: A | B



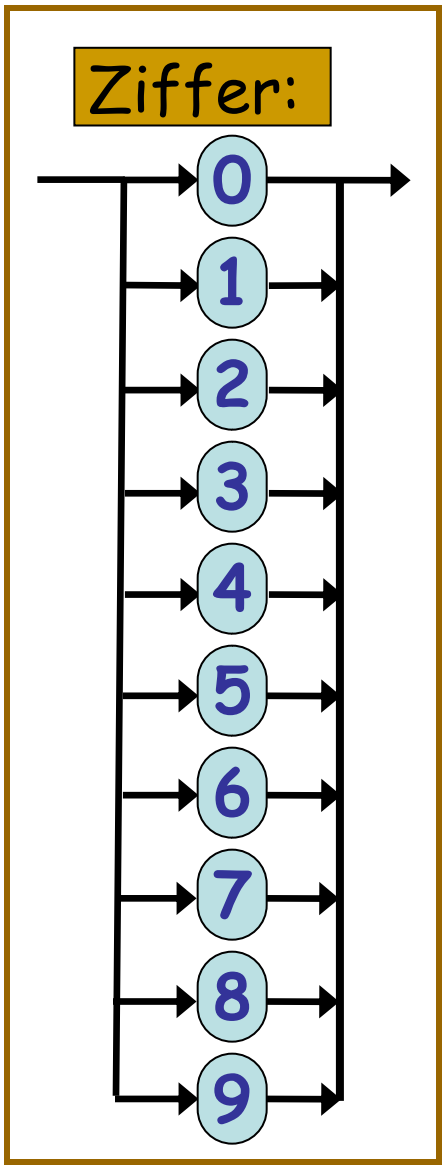
Repetition (0 oder mehr): { A }*

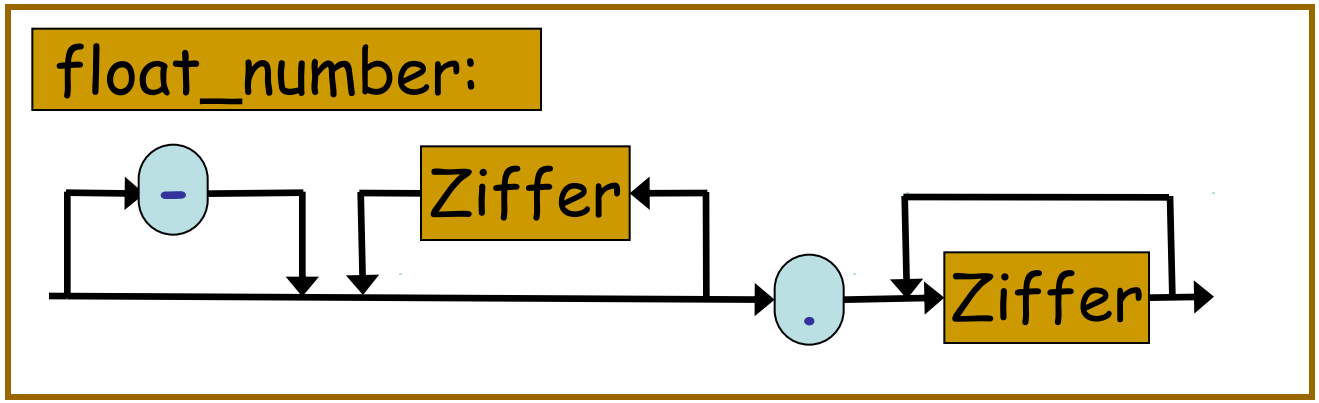


Repetition (mind. einmal): { A }+



Ein einfaches Beispiel





Beispielphrasen:
.76
-.76
1.56
12.845
-1.34
13.0

Übersetzen Sie es in die schriftliche Form!

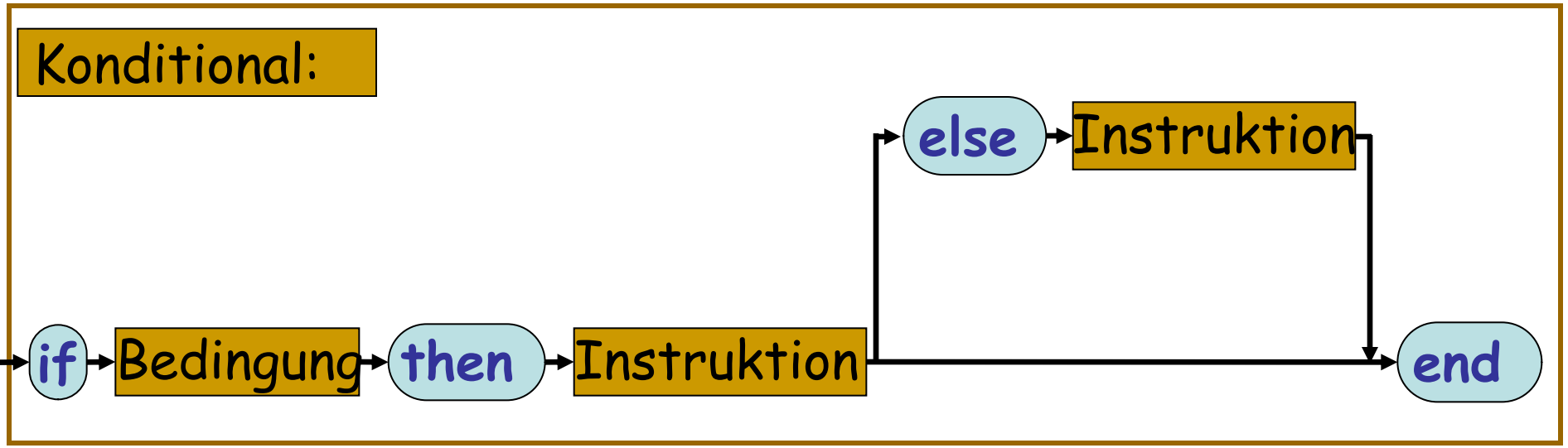
Ein einfaches Beispiel

In BNF:

Ziffer ≙ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

float_number ≙ [-] { Ziffer }* . { Ziffer }+

BNF Elemente kombiniert

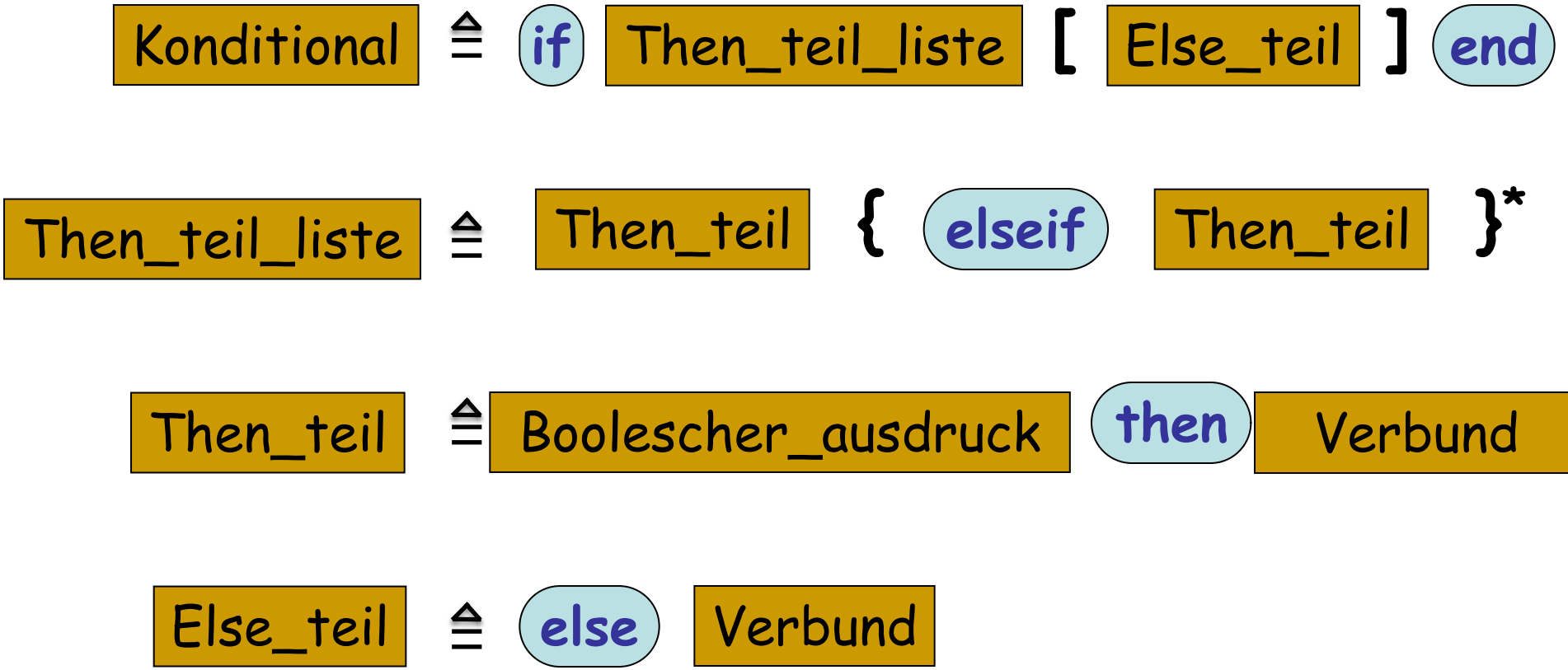


In BNF geschrieben:

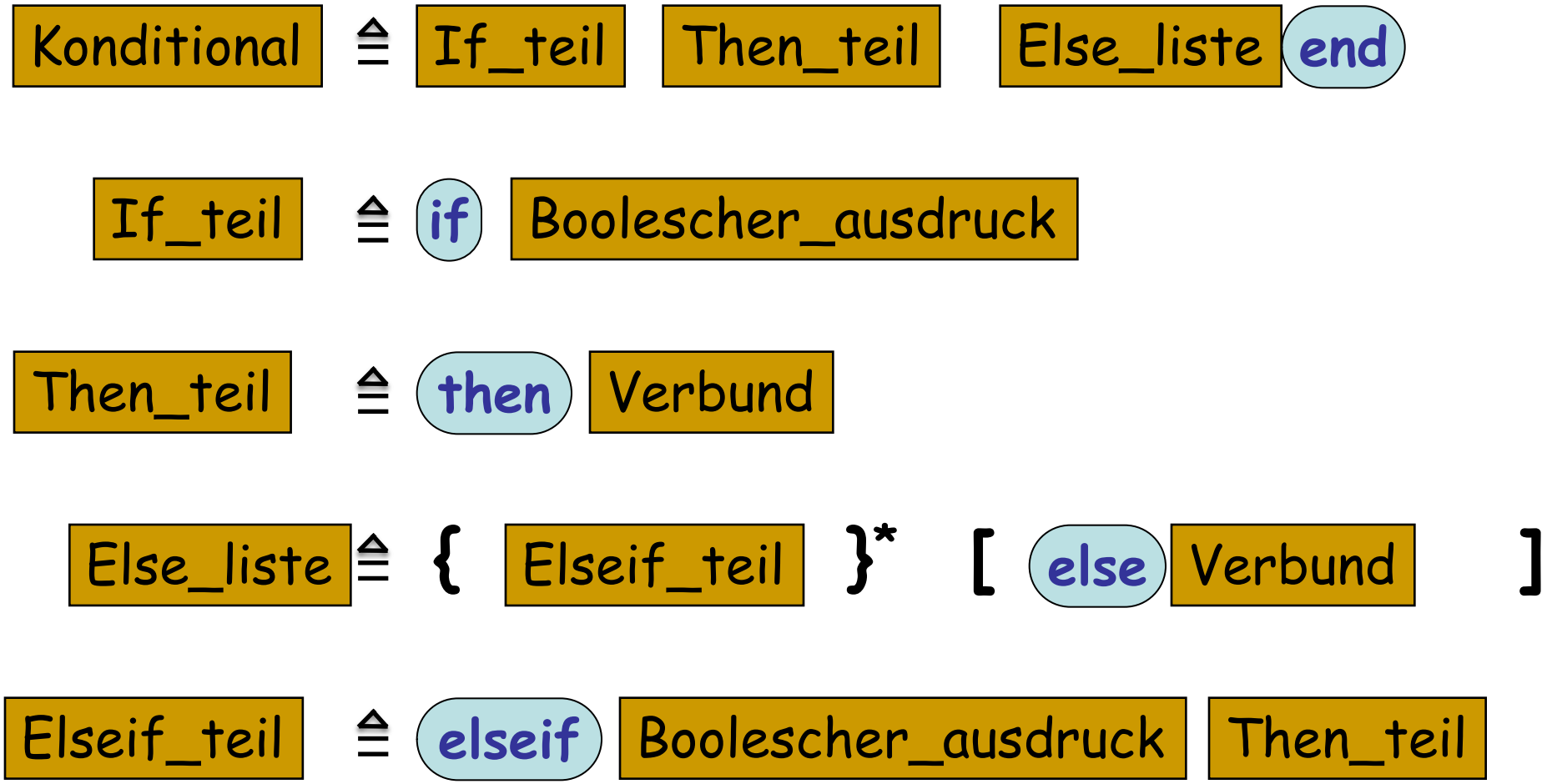
Konditional ≙

if Bedingung then Instruktion [else Instruktion] end

BNF: Konditional mit **elseif**



Andere Grammatik für Konditional



Einfaches BNF-Beispiel

Satz ≙ **I** [**don't**] Verb Namen Quant
Namen ≙ Name {**and** Name}*
Name ≙ **tomatoes** | **shoes** | **books** | **football**
Verb ≙ **like** | **hate**
Quant ≙ **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

Einfaches BNF-Beispiel (Lösung)

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 ≙ **I** [**don't**] Verb Namen [Quant]
Namen ≙ Name [{, Name}* **and** Name]
Name ≙ **tomatoes** | **shoes** | **books** | **football**
Verb ≙ **like** | **hate**
Quant ≙ **a lot** | **a little**

BNF-E

Wird in der offiziellen Beschreibung von Eiffel benutzt.

Jede Produktion ist eine der folgenden

Konkatenation

$A \triangleq B\ C\ [\ D\ ]$

Wahl

$A \triangleq B \mid C \mid D$

Repetition

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

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

Interpretiert als

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

Interpretiert als

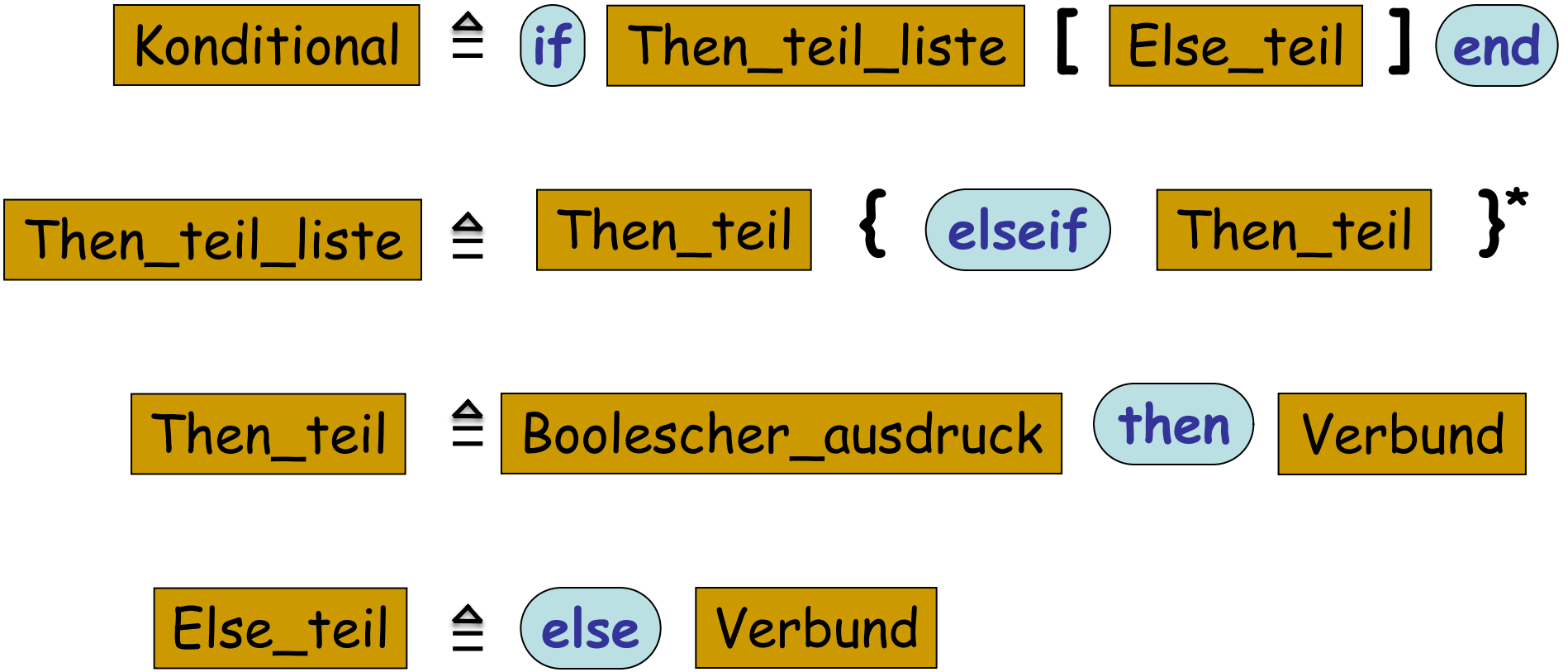
$A \triangleq B\ \{\ \text{delimiter}\ B\ \}^*$

BNF-E Regeln

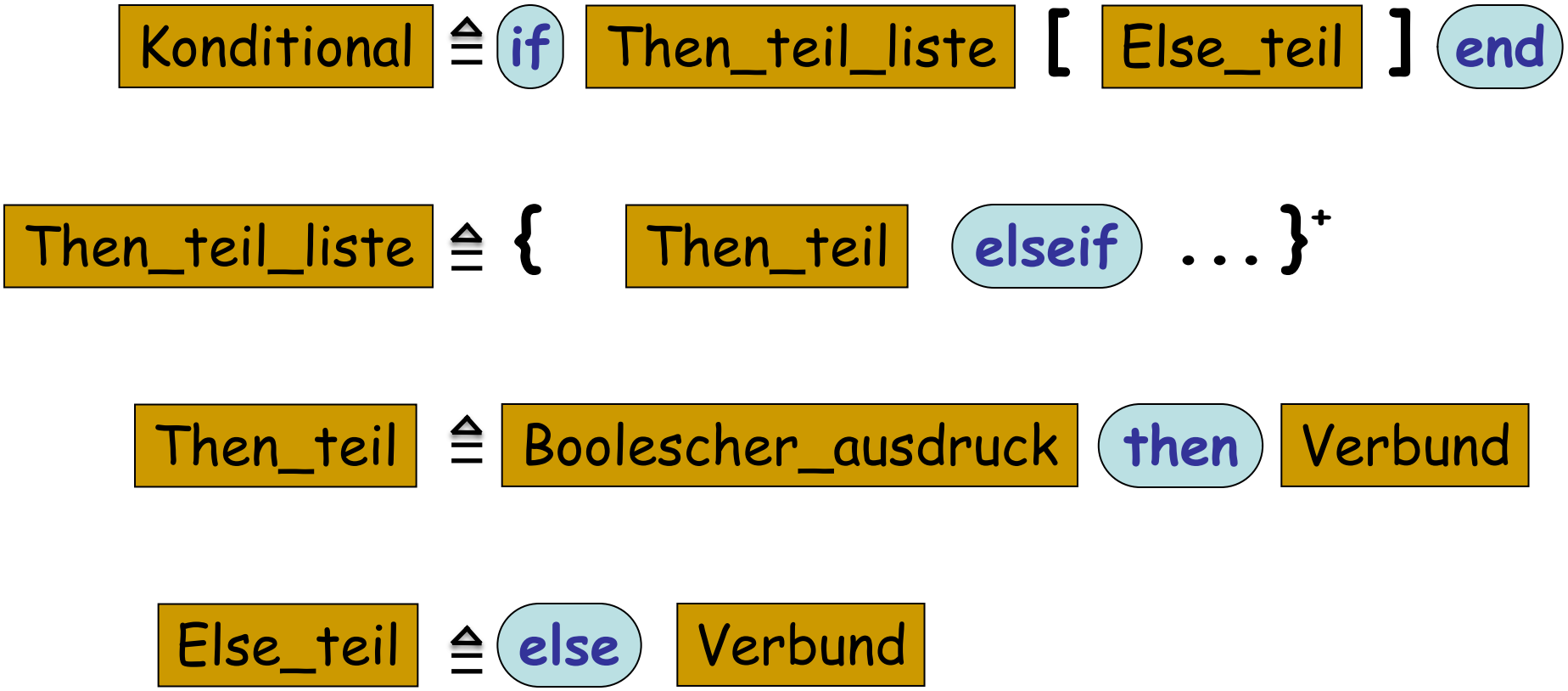
- Jedes Nonterminal muss auf der linken Seite von genau einer Produktion auftreten. Diese Produktion ist seine definierende Produktion.

- Jede Produktion ist von einer Art:
  Konkatenation, Wahl oder Repetition

BNF: Konditional mit **elseif**



BNF-E: Konditional



Rekursive Grammatiken

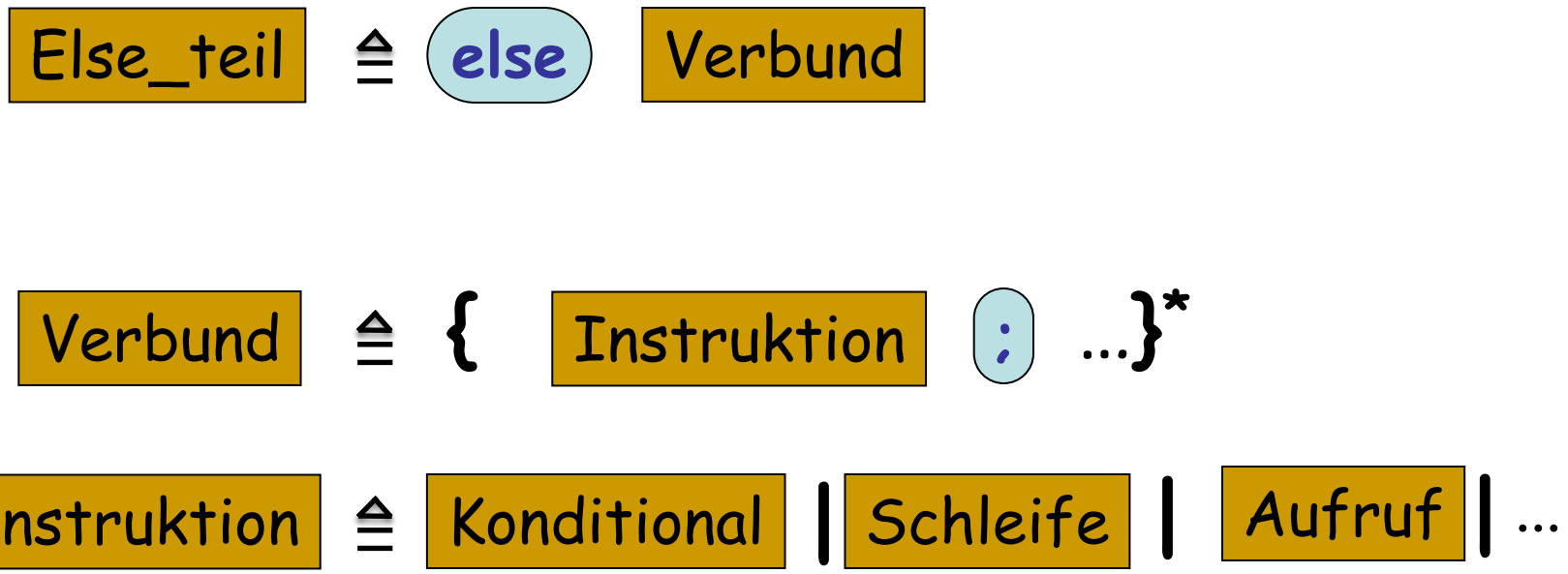
Konstrukte können verschachtelt sein

In BNF wird dies mit rekursiven Grammatiken ausgedrückt.

Rekursion: zirkuläre Abhängigkeiten von Produktionen

Rekursive Grammatiken

Konditionale können in anderen Konditionalen verschachtelt sein:



Rekursive Grammatiken

Der Produktionsname kann in der eigenen Definition vorkommen

Definition von Then_teil_liste mit Repetition:

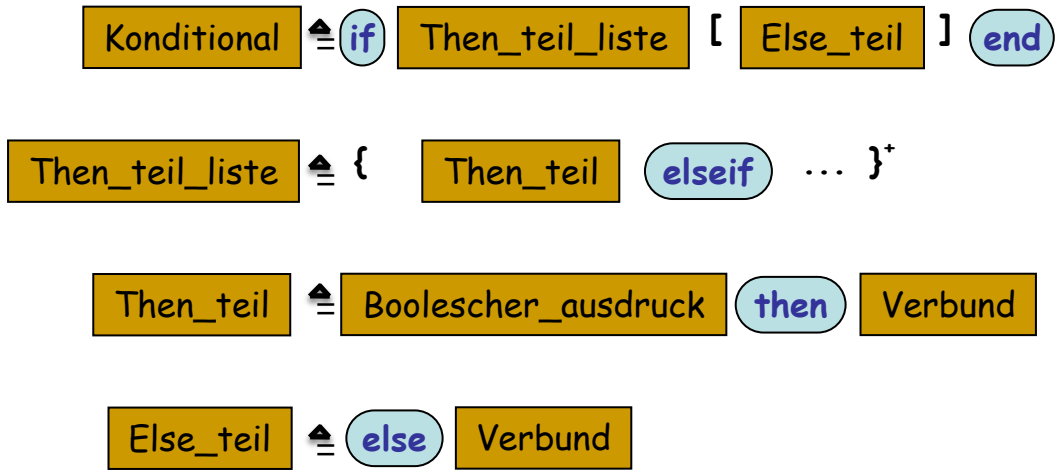
Then_teil_liste ≙ { Then_teil elseif ...}*

Rekursive Definition von Then_teil_liste:

Then_teil_liste ≙ 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
```



BNF für einfache arithmetische Ausdrücke

Nehmen Sie an, Number ist als positiver Integer definiert, und Variable ist ein alphabetisches Wort, das aus einem Zeichen besteht

Ist dies eine rekursive Grammatik?

Wie würde die gleiche Grammatik in BNF-E aussehen?

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	≙	Term { Add_op Term }*
Term	≙	Factor { Mult_op Factor}*
Factor	≙	Number | Variable | Nested
Nested	≙	(Expr)
Add_op	≙	+ | -
Mult_op	≙	* | /

BNF für einfache arithmetische Ausdrücke (Lösung)

Ist dies eine rekursive Grammatik? Ja (siehe Nested)

Wie würde die gleiche Grammatik in BNF-E aussehen? (siehe gelbe Box unten)

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

Richtlinien für Grammatiken

Halten Sie Produktionen kurz.

Einfacher zu lesen

Bessere Bewertung der Sprachengrösse

Conditional ≙

if Boolean_expression then Compound

{ elseif Boolean_expression then Compound }*

[else Compound] end

Richtlinien für Grammatiken

Behandeln Sie lexikale Konstrukte wie Terminale

Bezeichner

Konstante Werte

Identifier ≙ Letter {Letter | Digit | "_"}*

Integer_constant ≙ [-]{Digit}+

Floating_point ≙ [-] {Digit}* "." {Digit}+

Letter ≙ "A" | "B" | ... | "Z" | "a" | ... | "z"

Digit ≙ "0" | "1" | ... | "9"

Richtlinien für Grammatiken

Benutzen Sie eindeutige Produktionen.

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

Konditional ≙ **if** Then_teil_liste [Else_teil] **end**

Verbund ≙ { Instruktion }*

Instruktion ≙ Konditional | Schleife | Aufruf | ...

Syntaxbeschreibung von Eiffel

Die Syntax von Eiffel ist in BNF-E geschrieben

- Eine Produktion pro Nonterminal
- Jede Produktion ist entweder eine Konkatenation, 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*)

Syntaxbeschreibung von Eiffel (lexikale Ebene)

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 Konkatenation müssen die Elemente nicht (lexikalisch) getrennt sein

Noch eine Übung

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)

Lösung

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

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

Einen Parser schreiben

Ein Feature pro Produktion

Konkatenation:
Sequenz von Feature-Aufrufen für Nonterminale, überprüft auf Terminale

Wahl:
Konditional mit Verbund pro Alternative

Repetition:
Schleife

Einen Parser schreiben: EiffelParse

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

Einen Parser schreiben: Werkzeuge

Yooc:

Übersetzt BNF-E zu EiffelParse-Klassen

Yacc / Bison:

Übersetzt BNF zu C-Parser

BNF ähnliche Syntaxbeschreibungen

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)>
```

BNF ähnliche Syntaxbeschreibungen

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 ...
```

Eiffel-Syntax

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

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

Was wir gesehen haben

Eine Art Syntax zu beschreiben: BNF

3 Varianten: BNF, BNF-E, graphisch

Einen Blick ins Thema der Rekursion