Einführung in den Compilerbau

Syntaktische Analyse

TECHNISCHE UNIVERSITÄT DARMSTADT

Andreas Koch
Fachgebiet Eingebettete Systeme und ihre Anwendungen
Fachbereich Informatik

Embedded Systems & Applications

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Kompilierung

Embedded Systems & Applications

Ablauf der Übersetzung 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Terminologie: Phase

- Transformationsschritte
  - Von Quellcode
  - . . . zum Maschinencode

Ablauf der Übersetzung 1

TECHNISCHE UNIVERSITÄT DARMSTADT

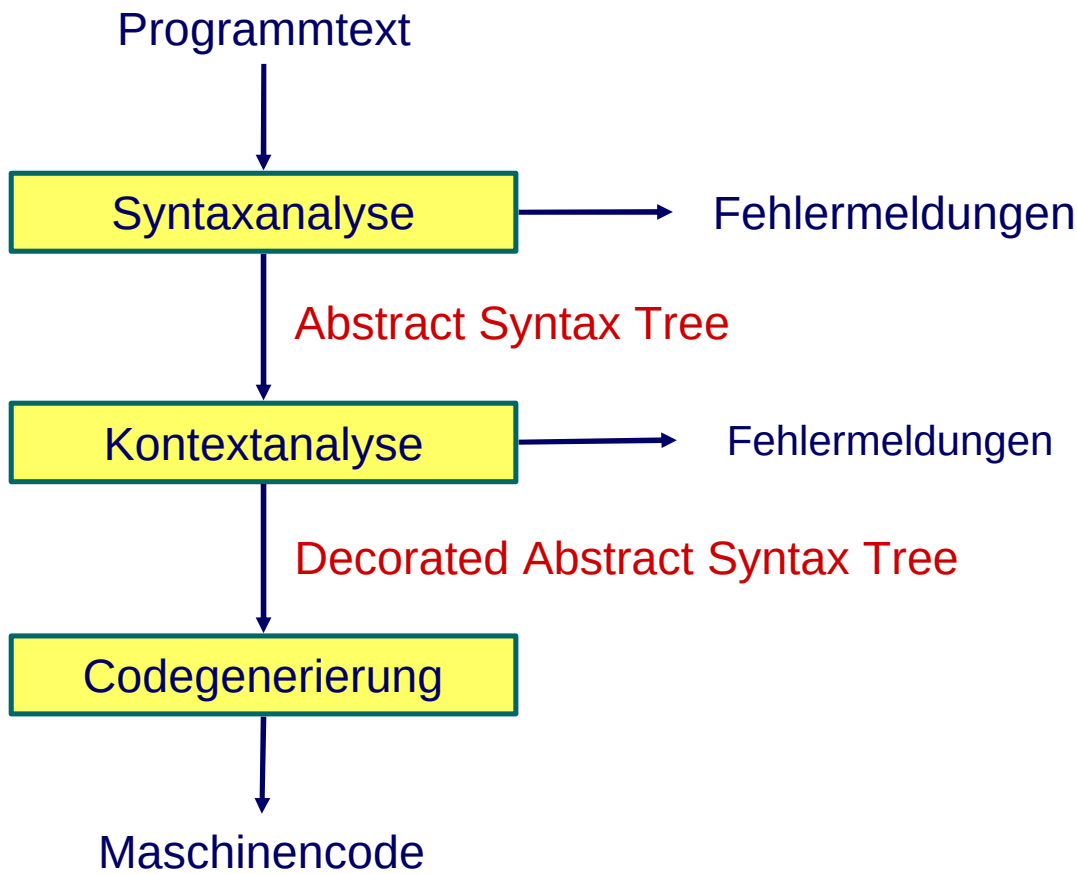
Terminologie: Phase

- Transformationsschritte
  - Von Quellcode
  - . . . zum Maschinencode

- Entspricht häufig den Teilen der Sprachspezifikation
  1. Syntax ➜ Syntaxanalyse
  2. Kontextuelle Einschränkungen ➜ Kontextanalyse
  3. Semantik ➜ Codegenerierung

Ablauf der Übersetzung 2

TECHNISCHE UNIVERSITÄT DARMSTADT



Embedded Systems & Applications

Ablauf der Übersetzung 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Terminologie: Durchgang (*pass*)

- Kompletter Durchgang des Programmes
- Läuft über Quelltext oder IR
- Pass *kann* Phase entsprechen
- . . . muss aber nicht!

Ablauf der Übersetzung 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Terminologie: Durchgang (*pass*)

- Kompletter Durchgang des Programmes
- Läuft über Quelltext oder IR
- Pass *kann* Phase entsprechen
- . . . muss aber nicht!
- Einzelner Pass kann mehrere Phasen durchführen
- Aufbau des Compilers wird von der Anzahl der Passes dominiert

Ein-Pass Compiler

TECHNISCHE UNIVERSITÄT DARMSTADT

- Macht nur einen Pass über den Quelltext
  - Baut in der Regel keine echte IR auf

Ein-Pass Compiler

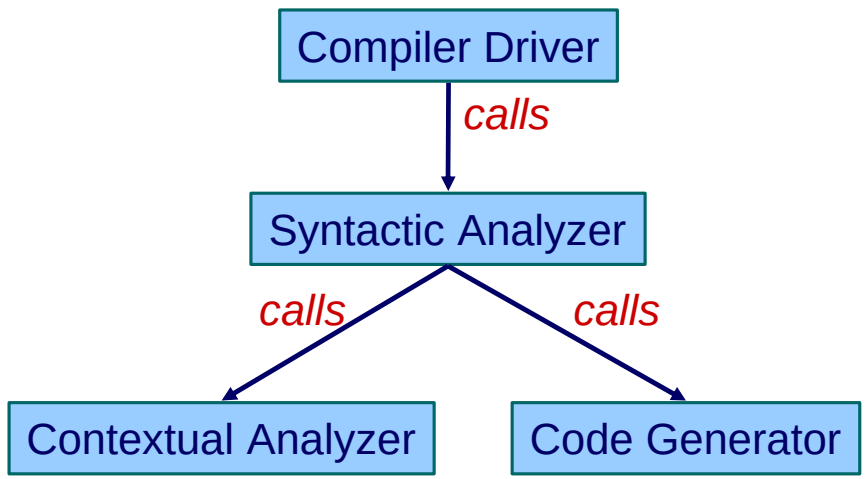
TECHNISCHE UNIVERSITÄT DARMSTADT

- Macht nur einen Pass über den Quelltext
  - Baut in der Regel keine echte IR auf
- Führt gleichzeitig aus
  - Syntaxanalyse (Parsing)
  - Kontextanalyse
  - Codegenerierung

Ein-Pass Compiler

TECHNISCHE UNIVERSITÄT DARMSTADT

- Macht nur einen Pass über den Quelltext
  - Baut in der Regel keine echte IR auf
- Führt gleichzeitig aus
  - Syntaxanalyse (Parsing)
  - Kontextanalyse
  - Codegenerierung
- Pascal Compiler haben häufig Ein-Pass-Struktur



Embedded Systems & Applications

Multi-Pass Compiler

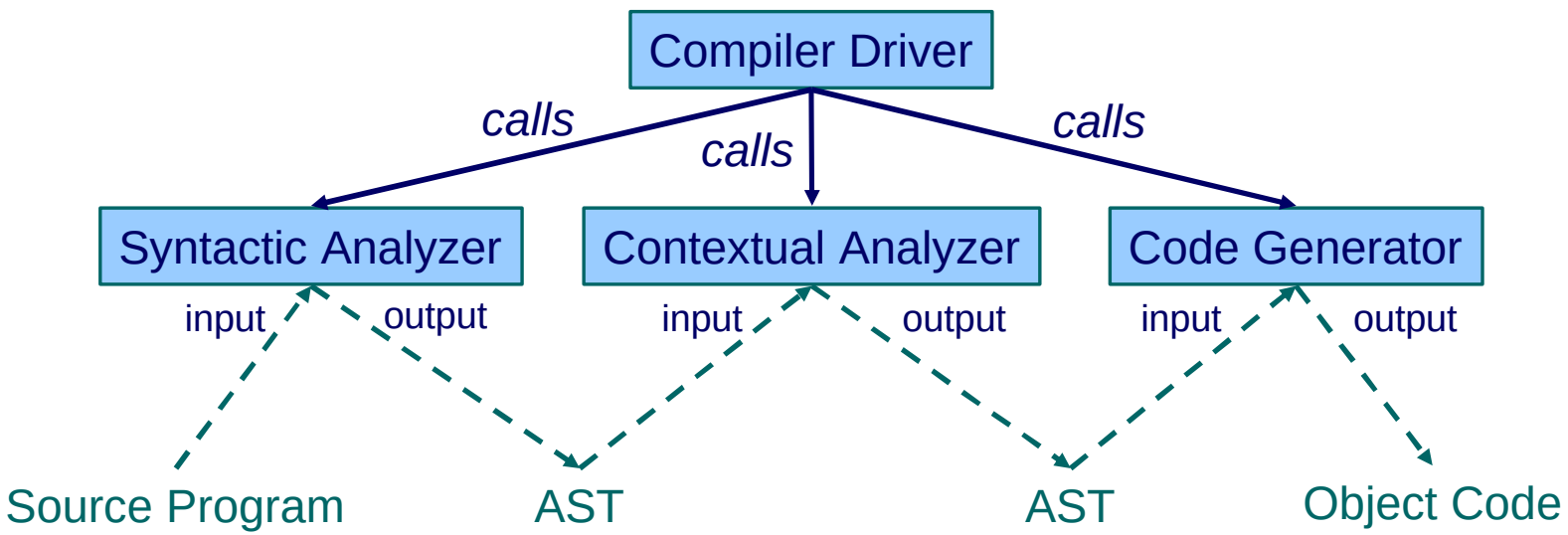
TECHNISCHE UNIVERSITÄT DARMSTADT

- Macht mehrere Passes über das Program
  - Quelltext und IR

Multi-Pass Compiler

TECHNISCHE UNIVERSITÄT DARMSTADT

- Macht mehrere Passes über das Program
  - Quelltext und IR
- Datenweitergabe zwischen Passes über IR



Embedded Systems & Applications

Vergleich Ein-Pass ./. Multi-Pass-Compiler

TECHNISCHE UNIVERSITÄT DARMSTADT

	Ein-Pass	Multi-Pass
Laufzeit	+	-
Speicher	+ für große Prog.	+ für kleine Prog.
Modularität	-	+
Flexibilität	-	+
Globale Optim.	- -	+
Eingabesprachen	Nicht für alle	

Embedded Systems & Applications

Beispiel Multi-Pass

TECHNISCHE UNIVERSITÄT DARMSTADT

Java-Compilierung erfordert mehrere Passes

```
class Example {
  void inc() { n = n + 1; }
  int n;
  void use() { n = 0; inc();}
}
```

Beachte Reihenfolge Verwendung/Bindung von **n**!

Aufbau des Triangle-Compilers

TECHNISCHE UNIVERSITÄT DARMSTADT

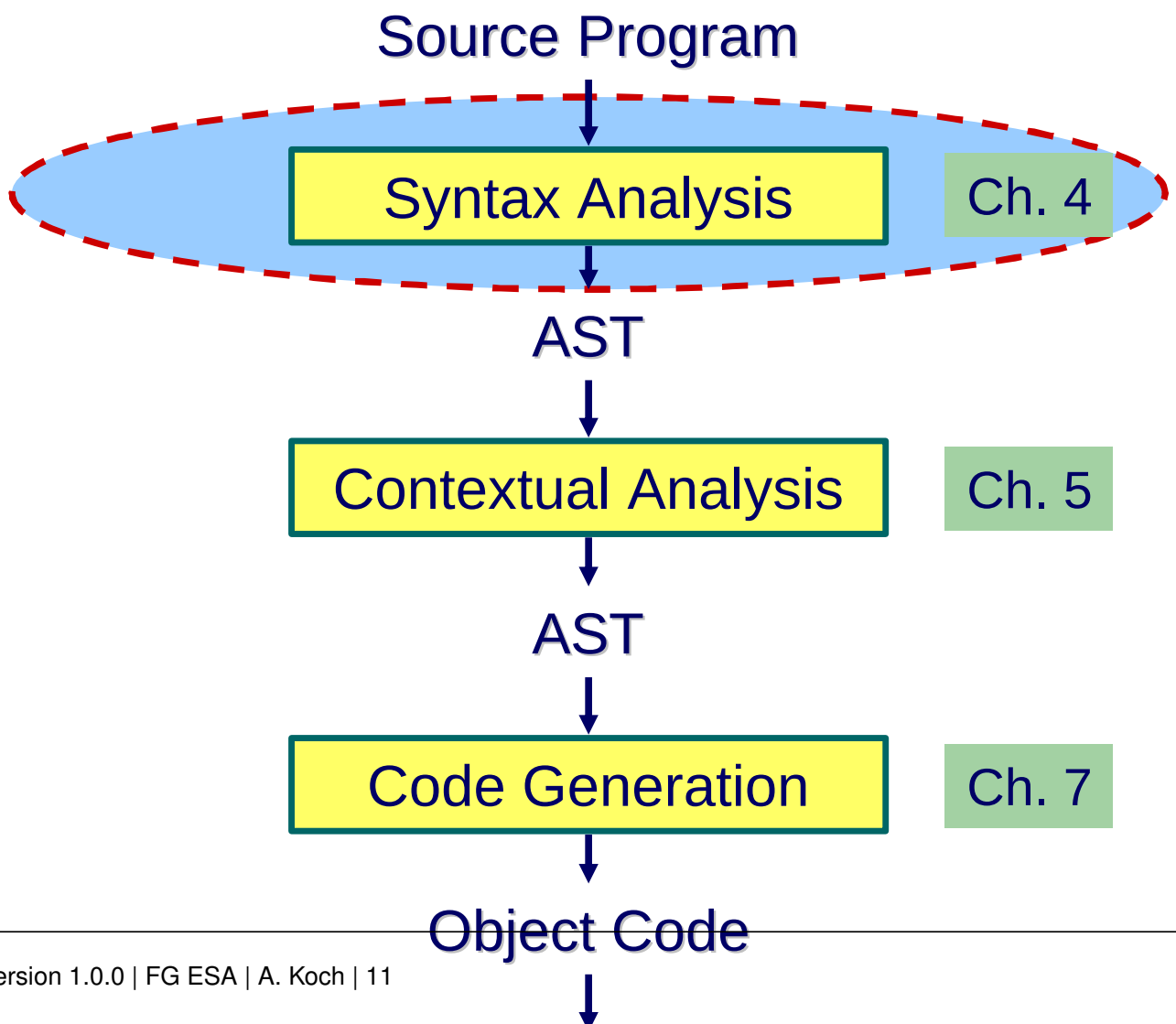
- Ein-Pass wäre für Triangle möglich
- Aus pädagogischen Gründen aber Multi-Pass

```
public class Compiler {
  public static boolean compileProgram(...) {
    Scanner scanner = new Scanner(...);
    Parser  parser  = new Parser(...);
    Checker checker = new Checker(...);
    Encoder encoder = new Encoder(...);
    ...
    Program theAST = parser.parse();
    checker.check(theAST);
    encoder.encode(theAST);
    ...
  }

  public static void main(String[] args) {
    ... compileProgram(...); ...
  }
}
```

Syntaxanalyse

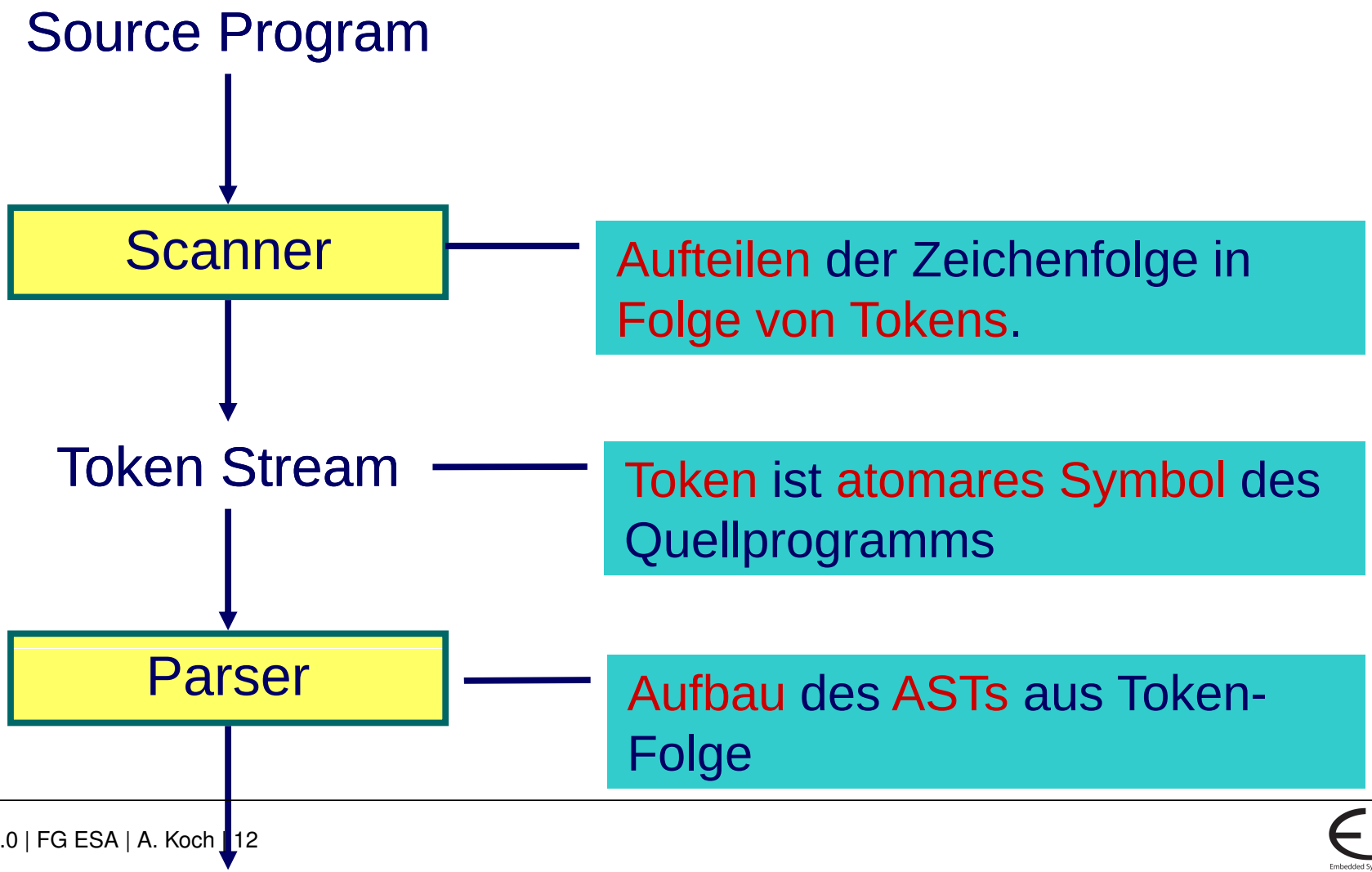
TECHNISCHE UNIVERSITÄT DARMSTADT



Embedded Systems & Applications

Subphasen der Syntaxanalyse

TECHNISCHE UNIVERSITÄT DARMSTADT



Embedded Systems & Applications

Token-Folge

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispielprogramm in Triangle

```
! Groesster Gemeinsamer Teiler
let func gcd(x: Integer, y: Integer) : Integer ~
    if x // y = 0                    ! // -> Modulo
    then y
    else gcd(y, x // y);
in  putint(gcd(321,81))
```

Token-Folge

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispielprogramm in Triangle

```
! Groesster Gemeinsamer Teiler
let func gcd(x: Integer, y: Integer) : Integer ~
    if x // y = 0                    ! // -> Modulo
    then y
    else gcd(y, x // y);
in  putint(gcd(321,81))
```

Token-Folge: Ohne Leerzeichen, Zeilenvorschub und Kommentare

```
let func gcd ( x : Integer , y : Integer )
: Integer ~ Integer if x // y = 0 then y
else gcd ( y , x // y ) ; in putint ( gcd
( 321 , 81 ) )
```


Embedded Systems & Applications

Tokens

TECHNISCHE UNIVERSITÄT DARMSTADT

- Token ist atomares Symbol des Programms
- Verwendet zwischen Scanner und Parser
- Kann auch aus mehreren Zeichen bestehen

Tokens

TECHNISCHE UNIVERSITÄT DARMSTADT

- Token ist atomares Symbol des Programms
- Verwendet zwischen Scanner und Parser
- Kann auch aus mehreren Zeichen bestehen
- Zeichen selbst i.d.R. uninteressant, Ausnahmen:
  - Bezeichnernamen
  - Konstante Werte (Zahlen, Zeichen), sog. *Literale*

Tokens

TECHNISCHE UNIVERSITÄT DARMSTADT

- Token ist atomares Symbol des Programms
- Verwendet zwischen Scanner und Parser
- Kann auch aus mehreren Zeichen bestehen
- Zeichen selbst i.d.R. uninteressant, Ausnahmen:
  - Bezeichnernamen
  - Konstante Werte (Zahlen, Zeichen), sog. *Literale*
- . . . Parser ist nur an der Art des Tokens interessiert

```
class Token {
  TokenKind      kind;     // enum TokenKind {...}
  String         spelling;
  SourcePosition position; // Zeilennummer, Spalte
}
```

Aufzählung für Token-Arten (Auszug)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
enum TokenKind {
  // literals, identifiers, operators...
  INTLITERAL ( 0, "<int>"),
  IDENTIFIER ( 2, "<identifier>"),
  OPERATOR   ( 3, "<operator>"),

  // reserved words
  BEGIN      ( 5, "begin"),
  CONST      ( 6, "const"),
  DO         ( 7, "do"),
  ELSE       ( 8, "else"),
  END        ( 9, "end"),
  IF         (11, "if"),
  IN         (12, "in"),
  LET        (13, "let"),
  THEN       (17, "then"),
  VAR        (19, "var"),
  WHILE      (20, "while"),
}
```

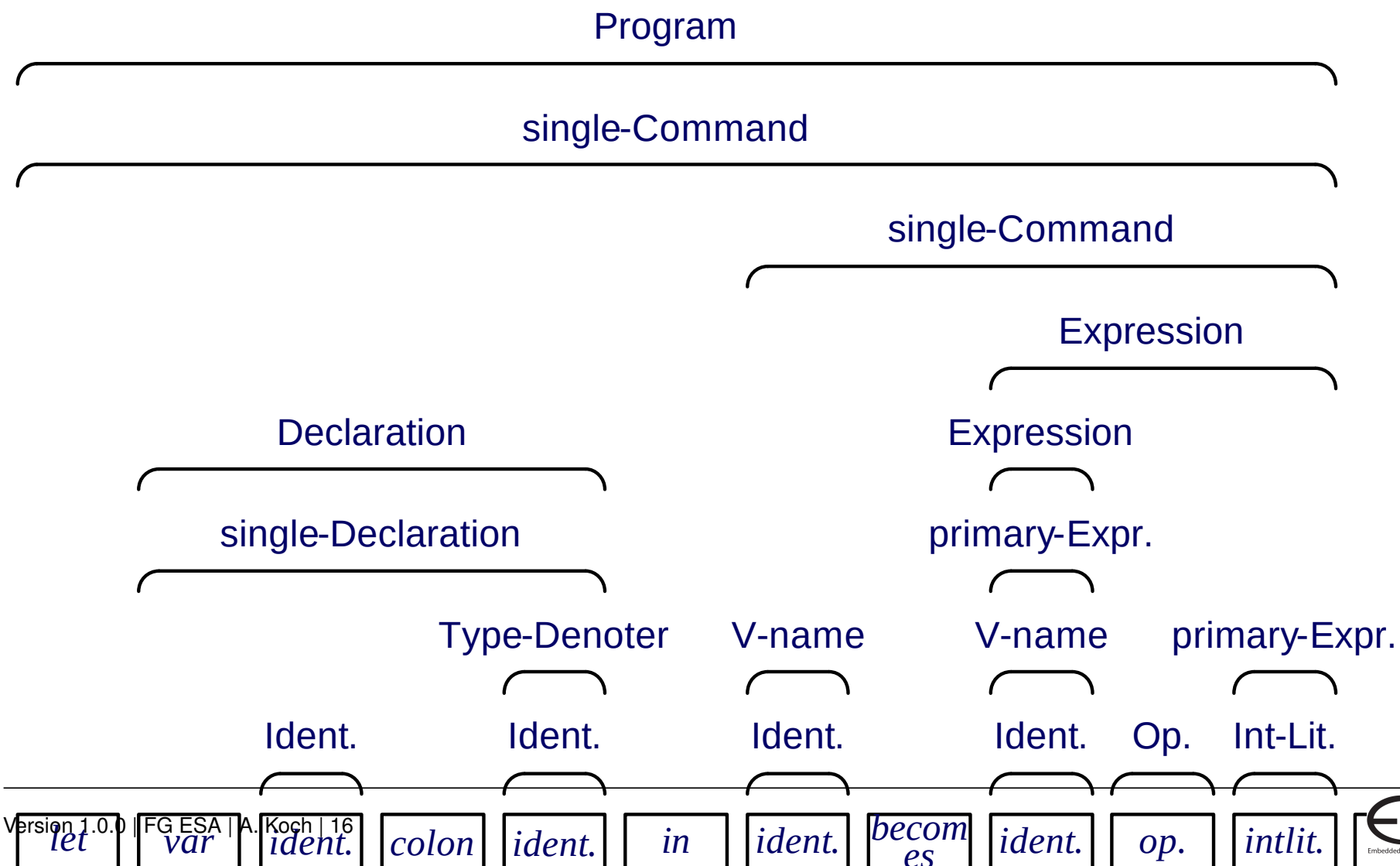
```
// punctuation and backets
SEMICOLON  (23, ";"),
COMMA      (24, ","),
BECOMES    (25, ":="),
IS         (26, "~"),
LPAREN     (27, "("),
RPAREN     (28, ")"),
...
  final int    id;
  final String spelling;
...
```

Beispiel: `t = new Token(TokenKind.OPERATOR, "+", position);`

Embedded Systems & Applications

Parsen der Token-Folge

TECHNISCHE UNIVERSITÄT DARMSTADT

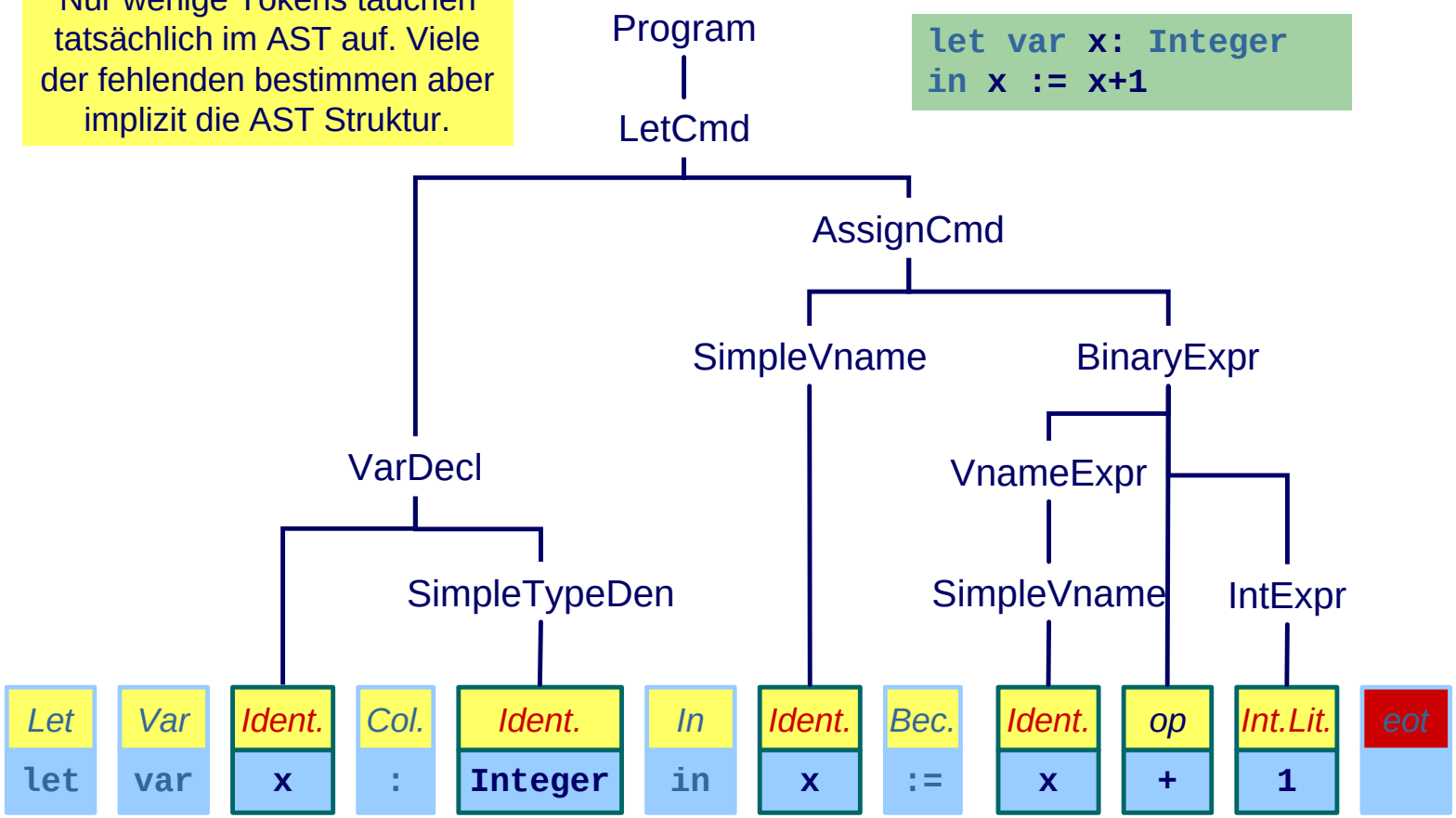


Aufbau des AST aus Token-Folge

TECHNISCHE UNIVERSITÄT DARMSTADT

Nur wenige Tokens tauchen tatsächlich im AST auf. Viele der fehlenden bestimmen aber implizit die AST Struktur.

```
let var x: Integer
in x := x+1
```



Embedded Systems & Applications

Kurzwiederholung Grammatiken

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kontextfreie Grammatiken (CFG)
- Spezifiziert durch (N, T, P, S)
- CFG definiert Menge von Zeichenketten
  - Elemente sind *Sätze* bestehend aus Terminalsymbolen
  - Gesamtmenge ist *Sprache* der CFG

Kurzwiederholung Grammatiken

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kontextfreie Grammatiken (CFG)
- Spezifiziert durch (N, T, P, S)
- CFG definiert Menge von Zeichenketten
  - Elemente sind *Sätze* bestehend aus Terminalsymbolen
  - Gesamtmenge ist *Sprache* der CFG
- Hier: Sätze haben eindeutige Phrasenstruktur
- P häufig in Backus-Naur-Form (BNF) angegeben
- Übersichtlicher: Extended BNF
  - BNF + Reguläre Ausdrücke auf rechter Seite der Produktionen

Beispiel: Produktionen in EBNF

TECHNISCHE UNIVERSITÄT DARMSTADT

BNF

```
Program      ::= single-Command
Command      ::= single-Command
              |  Command ; single-Command
....
Expression   ::= primary-Expression
              |  Expression operator primary-Expression
```

EBNF

```
Command      ::= single-Command (; single-Command)*

....

Expression   ::= primary-Expression
                    (operator primary-Expression)*
```


Embedded Systems & Applications

Reguläre Ausdrücke und Grammatiken

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch REs definieren eine Sprache
  - Reguläre Sprache
  - Weniger komplex als durch CFG beschreibbare Sprachen

Reguläre Ausdrücke und Grammatiken

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch REs definieren eine Sprache
  - Reguläre Sprache
  - Weniger komplex als durch CFG beschreibbare Sprachen
- CFG erlaubt Beschreibung von Selbsteinbettung
  - Ausdruck **`a*(b+c)/d`** bettet Ausdruck **`b+c`** ein
  - Vergleichbar dem Konzept der Rekursion
- REs erlauben keine Beschreibung von Selbsteinbettung

Reguläre Ausdrücke und Grammatiken

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch REs definieren eine Sprache
  - Reguläre Sprache
  - Weniger komplex als durch CFG beschreibbare Sprachen
- CFG erlaubt Beschreibung von Selbsteinbettung
  - Ausdruck `a*(b+c)/d` bettet Ausdruck `b+c` ein
  - Vergleichbar dem Konzept der Rekursion
- REs erlauben keine Beschreibung von Selbsteinbettung

Ziel: Systematische Herleitung von Parsern aus CFG

Transformation von Grammatiken

TECHNISCHE UNIVERSITÄT DARMSTADT

Hilfsmittel

- CFG kann transformiert (umgestellt) werden
- . . . unter Beibehaltung der beschriebenen Sprache

Grammatik-Transformation durch Gruppierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenfassen von Produktionen mit gleichem Nicht-Terminal auf linker Seite
  - *Left-Hand Side* (LHS), analog RHS

Grammatik-Transformation durch Gruppierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenfassen von Produktionen mit gleichem Nicht-Terminal auf linker Seite
  - *Left-Hand Side* (LHS), analog RHS

Vor Transformation

$$\begin{aligned} \mathbf{S} &::= \mathbf{X} + \mathbf{S} \\ \mathbf{S} &::= \mathbf{X} \\ \mathbf{S} &::= \varepsilon \end{aligned}$$

Grammatik-Transformation durch Gruppierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenfassen von Produktionen mit gleichem Nicht-Terminal auf linker Seite
  - *Left-Hand Side* (LHS), analog RHS

Vor Transformation

$$\begin{aligned} \mathbf{S} &::= \mathbf{X} + \mathbf{S} \\ \mathbf{S} &::= \mathbf{X} \\ \mathbf{S} &::= \varepsilon \end{aligned}$$

Nach Gruppierung

$$\mathbf{S} ::= \mathbf{X} + \mathbf{S} | \mathbf{X} | \varepsilon$$

Embedded Systems & Applications

Grammatik-Transformation durch Linksausklammern

TECHNISCHE UNIVERSITÄT DARMSTADT

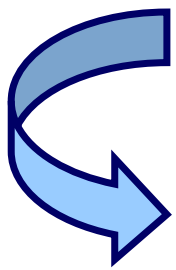
- Zusammenfassen von gleichen Anfängen in einer Produktion
- **X Y | X Z ➜ X(Y|Z)**

Grammatik-Transformation durch Linksausklammern

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenfassen von gleichen Anfängen in einer Produktion
- **X Y** | **X Z** ➜ **X**(**Y**|**Z**)

Beispiel:



```
cmd   := if Expr then cmd
       |  if Expr then cmd else cmd
```

```
cmd   := if Expr then cmd (ε  |  else cmd)
```


Embedded Systems & Applications

Beseitigung von Linksrekursion

TECHNISCHE UNIVERSITÄT DARMSTADT

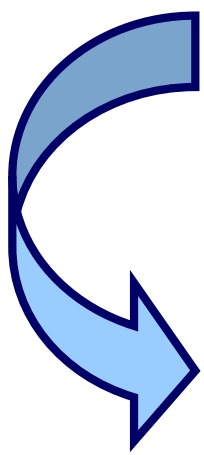
- Linksrekursion in Produktion
  - $\mathbf{N} ::= \mathbf{X} \mid \mathbf{N}\ \mathbf{Y}$
  - $L(\mathbf{N}) = \{\mathbf{X}, \mathbf{XY}, \mathbf{XYY}, \mathbf{XYYY}, \mathbf{XYYYY}, \ldots\}$
- Ersetzung durch
  - $\mathbf{N} ::= \mathbf{X}(\mathbf{Y})*$

Beseitigung von Linksrekursion

TECHNISCHE UNIVERSITÄT DARMSTADT

- Linksrekursion in Produktion
  - $\mathbf{N} ::= \mathbf{X} \mid \mathbf{N}\ \mathbf{Y}$
  - $L(\mathbf{N}) = \{\mathbf{X}, \mathbf{XY}, \mathbf{XYY}, \mathbf{XYYY}, \mathbf{XYYYY}, \ldots\}$
- Ersetzung durch
  - $\mathbf{N} ::= \mathbf{X}(\mathbf{Y})*$

Beispiel:



```
Identifier    ::=    Letter
                |    Identifier Letter
                |    Identifier Digit
```

```
Identifier    ::=    Letter (Letter | Digit)*
```


Embedded Systems & Applications

Kombinierter Fall

TECHNISCHE UNIVERSITÄT DARMSTADT

Vor Transformation

$$\mathbf{N} \quad ::= \quad \mathbf{X}_1 \mid \ldots \mid \mathbf{X}_m \mid \mathbf{N}\,\mathbf{Y}_1 \mid \ldots \mid \mathbf{N}\,\mathbf{Y}_n$$

Kombinierter Fall

TECHNISCHE UNIVERSITÄT DARMSTADT

Vor Transformation

$$\mathbf{N} \quad ::= \quad \mathbf{X}_1 \mid \ldots \mid \mathbf{X}_m \mid \mathbf{N}\,\mathbf{Y}_1 \mid \ldots \mid \mathbf{N}\,\mathbf{Y}_n$$

Nach Linksausklammern

$$\mathbf{N} \quad ::= \quad (\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_m) \mid (\mathbf{N}(\mathbf{Y}_1 \mid \ldots \mid \mathbf{Y}_n))$$

Kombinierter Fall

TECHNISCHE UNIVERSITÄT DARMSTADT

Vor Transformation

$$\mathbf{N} \quad ::= \quad \mathbf{X}_1 \mid \ldots \mid \mathbf{X}_m \mid \mathbf{N}\,\mathbf{Y}_1 \mid \ldots \mid \mathbf{N}\,\mathbf{Y}_n$$

Nach Linksausklammern

$$\mathbf{N} \quad ::= \quad (\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_m) \mid (\mathbf{N}(\mathbf{Y}_1 \mid \ldots \mid \mathbf{Y}_n))$$

Nach Beseitigen der Linksrekursion

$$\mathbf{N} \quad ::= \quad (\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_m)(\mathbf{Y}_1 \mid \ldots \mid \mathbf{Y}_n)*$$

Ersetzen von Nicht-Terminalsymbolen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wenn **N** ::= **X** einzige Produktion mit LHS **N** ist
- . . . **N** durch **X** in RHS aller Produktionen ersetzen

Ersetzen von Nicht-Terminalsymbolen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wenn **N** ::= **X** einzige Produktion mit LHS **N** ist
- . . . **N** durch **X** in RHS aller Produktionen ersetzen

Beispiel:

Vor Transformation

single-Declaration ::= **var** Identifier : Type-denoter | . . .
Type-denoter ::= Identifier

Ersetzen von Nicht-Terminalsymbolen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wenn **N** ::= **X** einzige Produktion mit LHS **N** ist
- . . . **N** durch **X** in RHS aller Produktionen ersetzen

Beispiel:

Vor Transformation

single-Declaration ::= **var** Identifier : Type-denoter | . . .
Type-denoter ::= Identifier

Nach Ersetzung

single-Declaration ::= **var** Identifier : Identifier | . . .

Ersetzen von Nicht-Terminalsymbolen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wenn **N** ::= **X** einzige Produktion mit LHS **N** ist
- . . . **N** durch **X** in RHS aller Produktionen ersetzen

Beispiel:

Vor Transformation

single-Declaration ::= **var** Identifier : Type-denoter | . . .
Type-denoter ::= Identifier

Nach Ersetzung

single-Declaration ::= **var** Identifier : Identifier | . . .

Aber . . .

Solche “überflüssigen” Nicht-Terminals können nützlichen Dokumentationscharakter für den menschlichen Leser haben!

Embedded Systems & Applications

Nutzen von Grammatiktransformationen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Hier auf den ersten Blick noch nicht erkennbar
- Erlauben kompaktere und lesbarere Beschreibung von CFGs
- Sehr nützlich bei der Konstruktion von Parsern für CFGs

Parsing Terminologie

TECHNISCHE UNIVERSITÄT DARMSTADT

Erkennung: Entscheidung, ob ein Eingabetext ein Satz der Grammatik *G* ist.

Parsing Terminologie

TECHNISCHE UNIVERSITÄT DARMSTADT

Erkennung: Entscheidung, ob ein Eingabetext ein Satz der Grammatik *G* ist.

Parsing: Erkennung und zusätzlich Bestimmung der Phrasen-Struktur

Parsing Terminologie

TECHNISCHE UNIVERSITÄT DARMSTADT

Erkennung: Entscheidung, ob ein Eingabetext ein Satz der Grammatik *G* ist.

Parsing: Erkennung und zusätzlich Bestimmung der Phrasen-Struktur

- Beispiel: Durch *konkreten* Syntaxbaum

Parsing Terminologie

TECHNISCHE UNIVERSITÄT DARMSTADT

Erkennung: Entscheidung, ob ein Eingabetext ein Satz der Grammatik *G* ist.

Parsing: Erkennung und zusätzlich Bestimmung der Phrasen-Struktur

- Beispiel: Durch *konkreten* Syntaxbaum

Eindeutigkeit: Eine Grammatik ist eindeutig falls jeder Eingabetext auf maximal eine Weise geparsed werden kann,

Parsing Terminologie

TECHNISCHE UNIVERSITÄT DARMSTADT

Erkennung: Entscheidung, ob ein Eingabetext ein Satz der Grammatik *G* ist.

Parsing: Erkennung und zusätzlich Bestimmung der Phrasen-Struktur

- Beispiel: Durch *konkreten* Syntaxbaum

Eindeutigkeit: Eine Grammatik ist eindeutig falls jeder Eingabetext auf maximal eine Weise geparsed werden kann,

- Ein syntaktisch korrekter Eingabetext hat genau einen eindeutigen Syntaxbaum

Parsing Strategien

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zwei wesentliche Verfahren

Parsing Strategien

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zwei wesentliche Verfahren
- Unterscheiden sich in der Art ihres Vorgehens

Parsing Strategien

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zwei wesentliche Verfahren
- Unterscheiden sich in der Art ihres Vorgehens
  - Top-Down Beispiel: Rekursiver Abstieg

Parsing Strategien

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zwei wesentliche Verfahren
- Unterscheiden sich in der Art ihres Vorgehens

  Top-Down Beispiel: Rekursiver Abstieg
  Bottom-Up Beispiel: Shift/Reduce

Beispiel: Micro-English

TECHNISCHE UNIVERSITÄT DARMSTADT

Produktionen

Sentence	::=	**Subject Verb Object.**
Subject	::=	**I** | **a Noun** | **the Noun**
Object	::=	**me** | **a Noun** | **the Noun**
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Beispiel: Micro-English

TECHNISCHE UNIVERSITÄT DARMSTADT

Produktionen

Sentence	::=	**Subject Verb Object .**
Subject	::=	**I** | **a Noun** | **the Noun**
Object	::=	**me** | **a Noun** | **the Noun**
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Beispiele der erzeugten Sprache

```
the cat sees a rat .
  I like the cat .
  the cat see me .
    I like me .
  a rat like me .
```


Embedded Systems & Applications

Bottom-Up Parsing 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorgehensweise

- Untersuche Eingabetext zeichenweise, von links nach rechts

Bottom-Up Parsing 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorgehensweise

- Untersuche Eingabetext zeichenweise, von links nach rechts
- Baue Syntaxbaum von unten nach oben auf
  - Von den Terminalzeichen in den Blättern
  - . . . zum *S* Nicht-Terminal in der Wurzel

Bottom-Up Parsing 2

Zwei Arten von Aktionen

Shift Lese Zeichen ein

- Zusätzlich: Und lege es auf dem Stack ab

Bottom-Up Parsing 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Zwei Arten von Aktionen

Shift Lese Zeichen ein

- Zusätzlich: Und lege es auf dem Stack ab

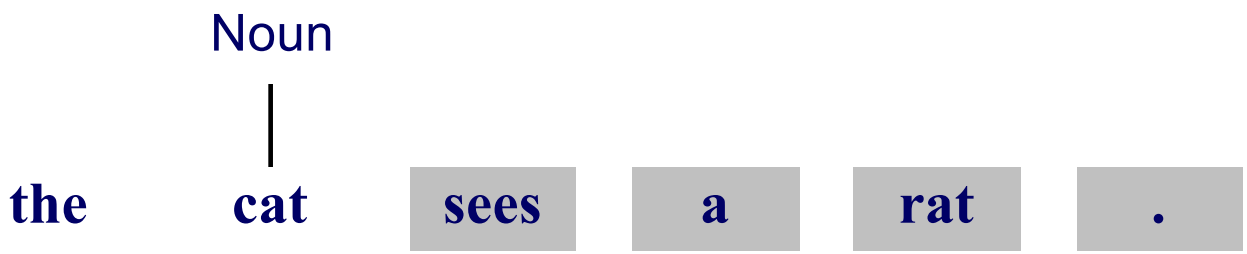
Reduce Erkenne ein Nicht-Terminal LHS der Produktion *p*

- Zusätzlich: Oberste Elemente des Stapels müssen RHS von *p* entsprechen, ersetze durch LHS von *p* (Zusammenfassen)
- Ende wenn Startsymbol *S* erreicht und Eingabetext komplett gelesen

Beispiel Bottom-Up Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT



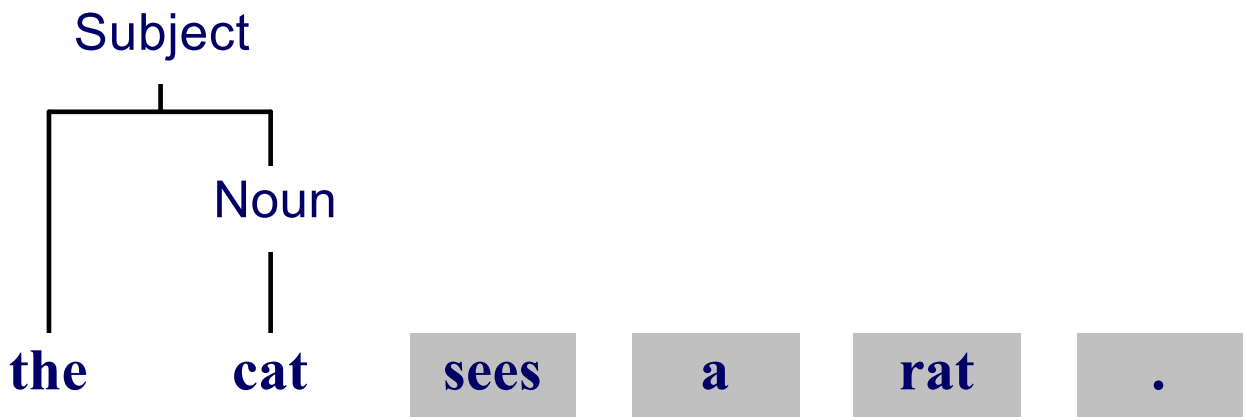
Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Embedded Systems & Applications

Beispiel Bottom-Up Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT



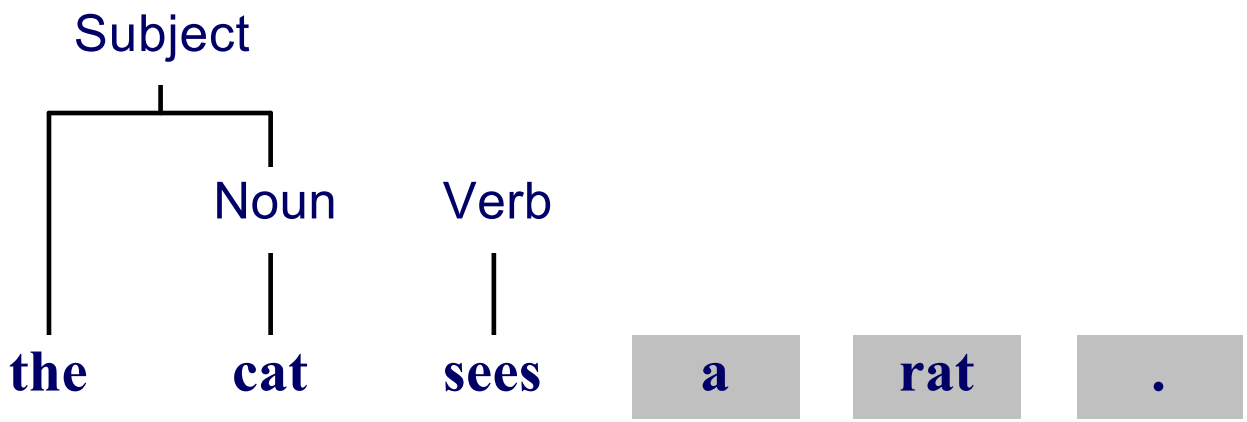
Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Embedded Systems & Applications

Beispiel Bottom-Up Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT



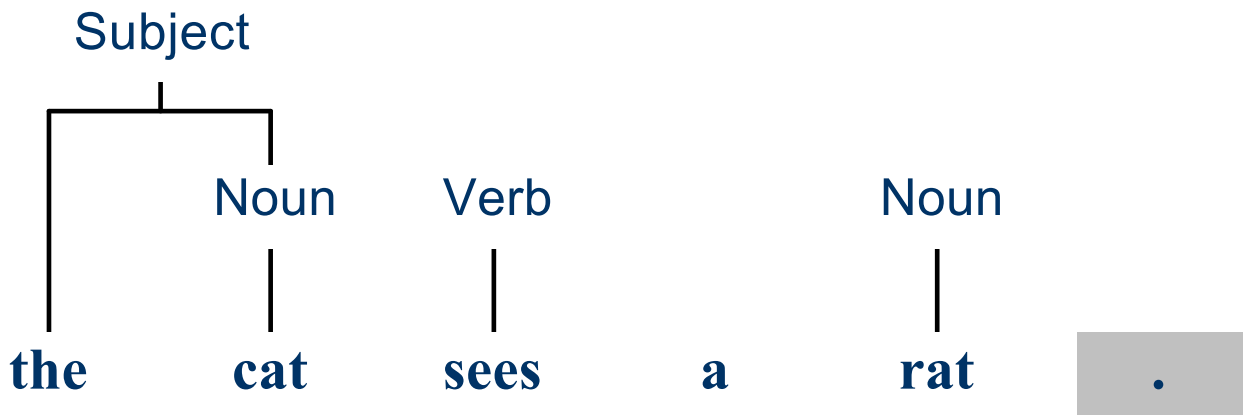
Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Embedded Systems & Applications

Beispiel Bottom-Up Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT



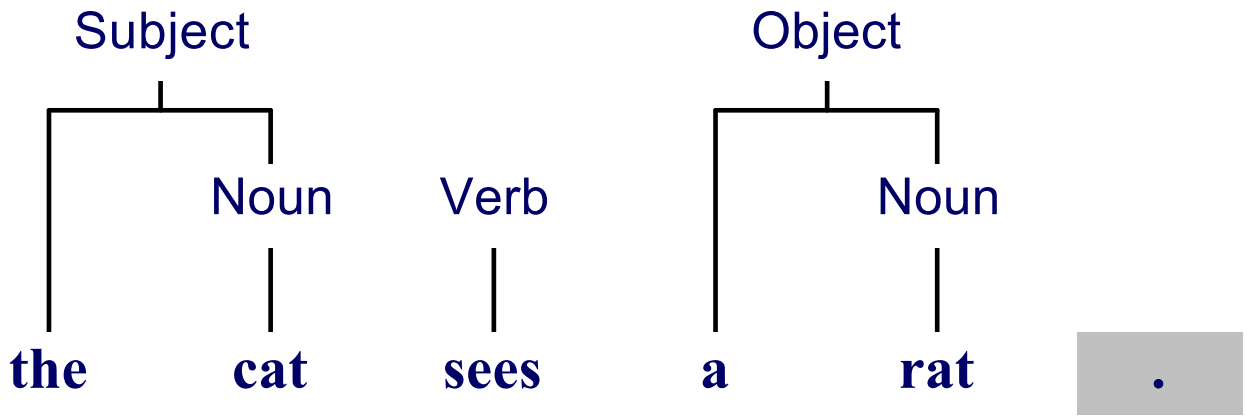
Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Embedded Systems & Applications

Beispiel Bottom-Up Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT



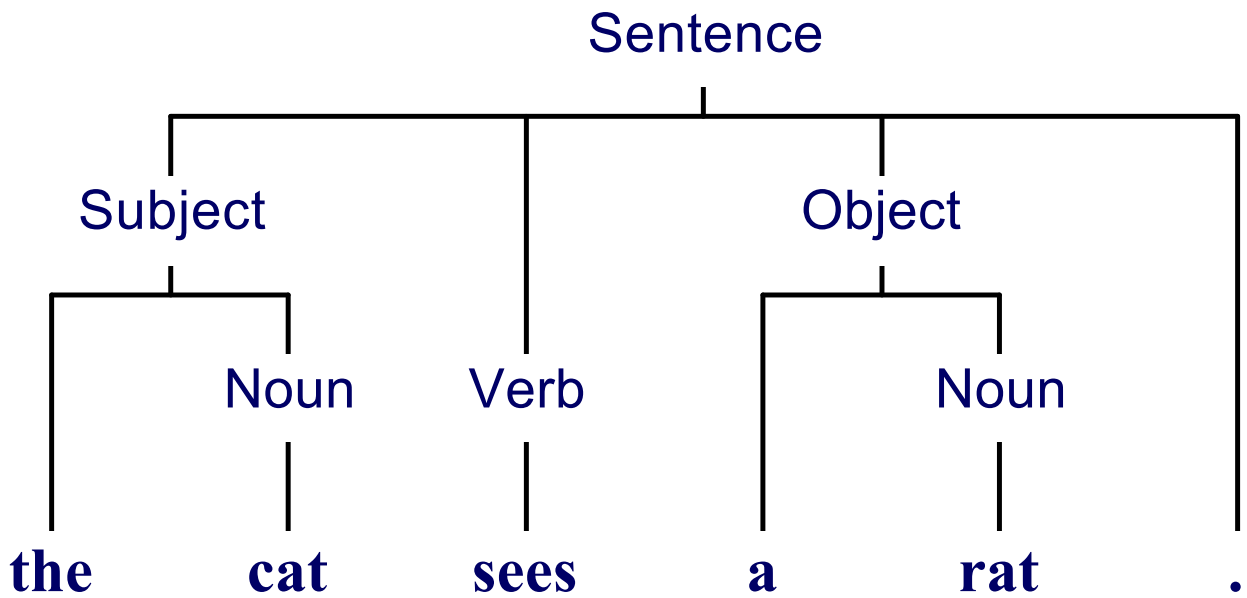
Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Embedded Systems & Applications

Beispiel Bottom-Up Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT



Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**

Embedded Systems & Applications

Schwierigkeit bei Bottom-Up Parsing

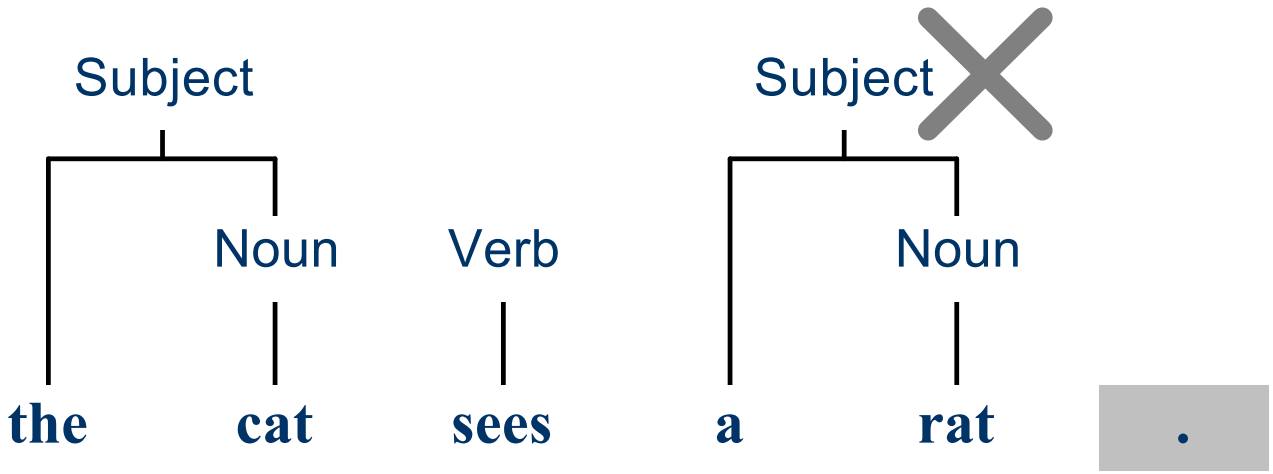
TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Produktion beim Zusammenfassen anwenden?

Schwierigkeit bei Bottom-Up Parsing

TECHNISCHE UNIVERSITÄT DARMSTADT

Welche Produktion beim Zusammenfassen anwenden?



Lösung: Nicht nur bekannte Zeichen betrachten, sondern auch noch Zustand (“schon Subject gesehen”) einbeziehen.
. . . aber hier nicht weiter vertieft!

Embedded Systems & Applications

Top-Down Parsing 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorgehensweise

- Untersuche Eingabetext zeichenweise, von links nach rechts

Top-Down Parsing 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorgehensweise

- Untersuche Eingabetext zeichenweise, von links nach rechts
- Baue Syntaxbaum von oben nach unten auf
  - Vom Start-Nicht-Terminal *S* in der Wurzel
  - . . . zu den Terminalzeichen in den Blättern

Top-Down Parsing 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Aktion

- Expandiere jeweils das am weitesten links gelegene Nicht-Terminal **N**

Top-Down Parsing 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Aktion

- Expandiere jeweils das am weitesten links gelegene Nicht-Terminal **N**
- . . . durch Anwendung einer Produktion **N** ::= **X**

Top-Down Parsing 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Aktion

- Expandiere jeweils das am weitesten links gelegene Nicht-Terminal **N**
- . . . durch Anwendung einer Produktion **N** ::= **X**
- Wähle Produktion aus durch Betrachten der nächsten n Zeichen des Eingabetextes (Annahme hier: $n = 1$)

Top-Down Parsing 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Aktion

- Expandiere jeweils das am weitesten links gelegene Nicht-Terminal **N**
- . . . durch Anwendung einer Produktion **N** ::= **X**
- Wähle Produktion aus durch Betrachten der nächsten n Zeichen des Eingabetextes (Annahme hier: $n = 1$)
- Falls keine Produktion auf Zeichen passt ➜Fehler!

Top-Down Parsing 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Aktion

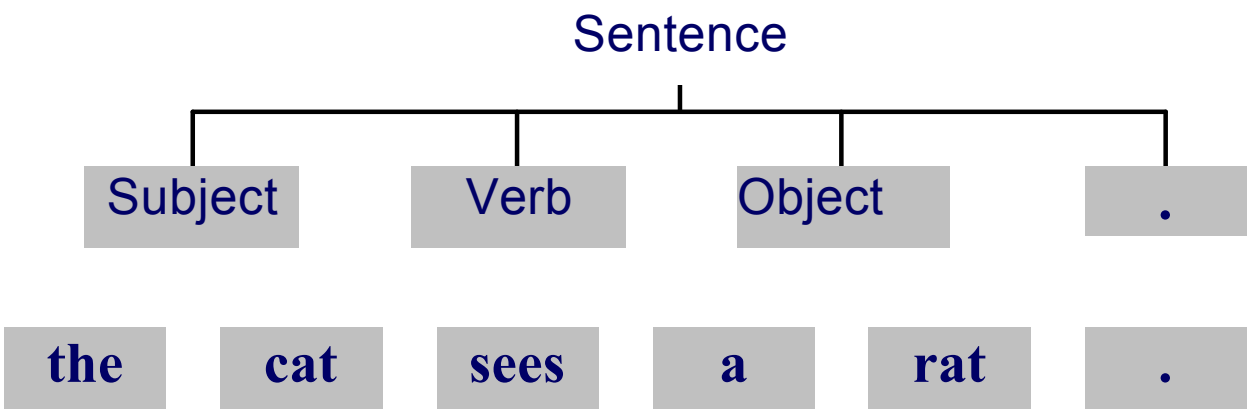
- Expandiere jeweils das am weitesten links gelegene Nicht-Terminal **N**
- . . . durch Anwendung einer Produktion **N** ::= **X**
- Wähle Produktion aus durch Betrachten der nächsten n Zeichen des Eingabetextes (Annahme hier: $n = 1$)
- Falls keine Produktion auf Zeichen passt ➜Fehler!
- Ende wenn Eingabetext komplett gelesen und kein unexpandiertes Nicht-Terminal mehr existiert

Beispiel Top-Down Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**



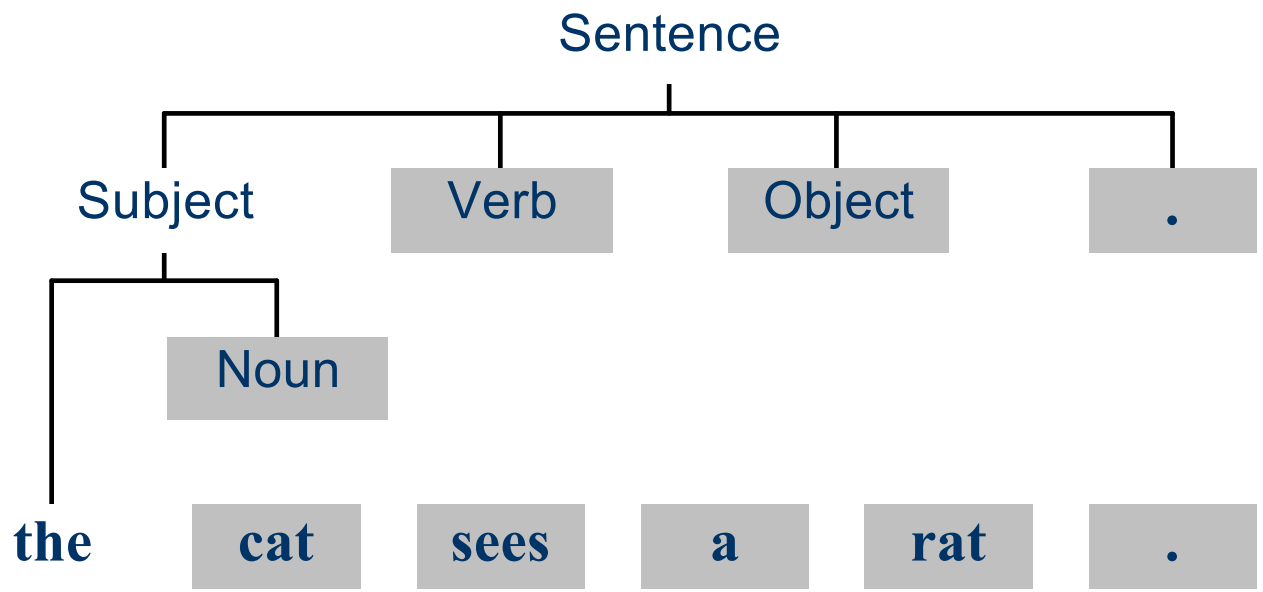
Embedded Systems & Applications

Beispiel Top-Down Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**



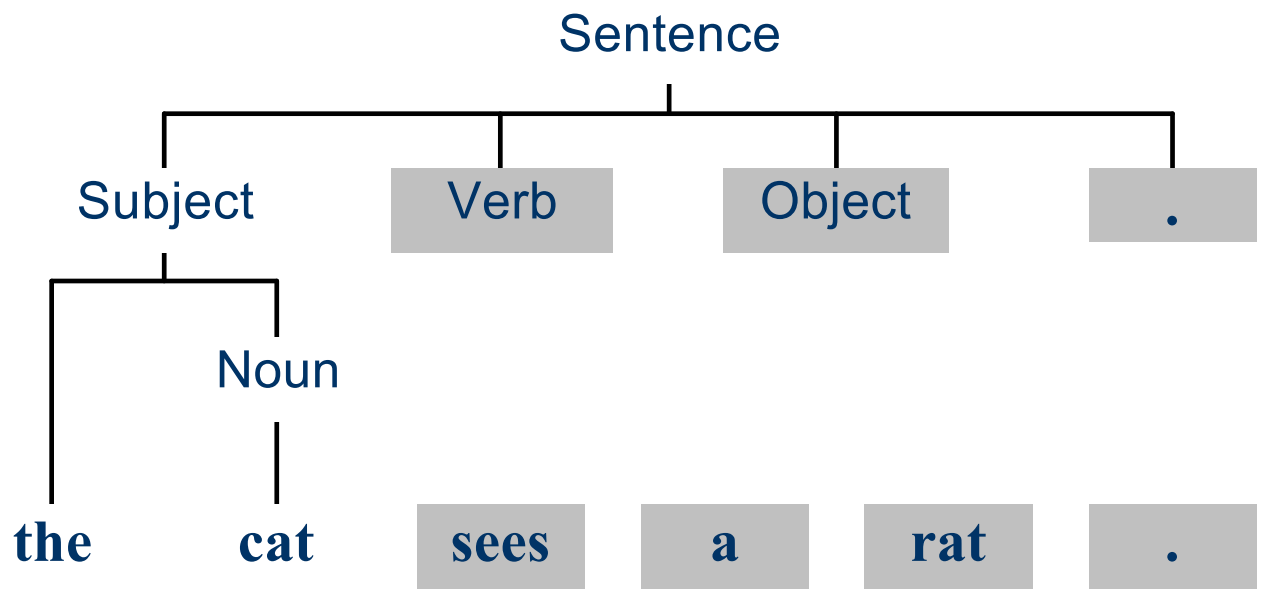
Embedded Systems & Applications

Beispiel Top-Down Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**



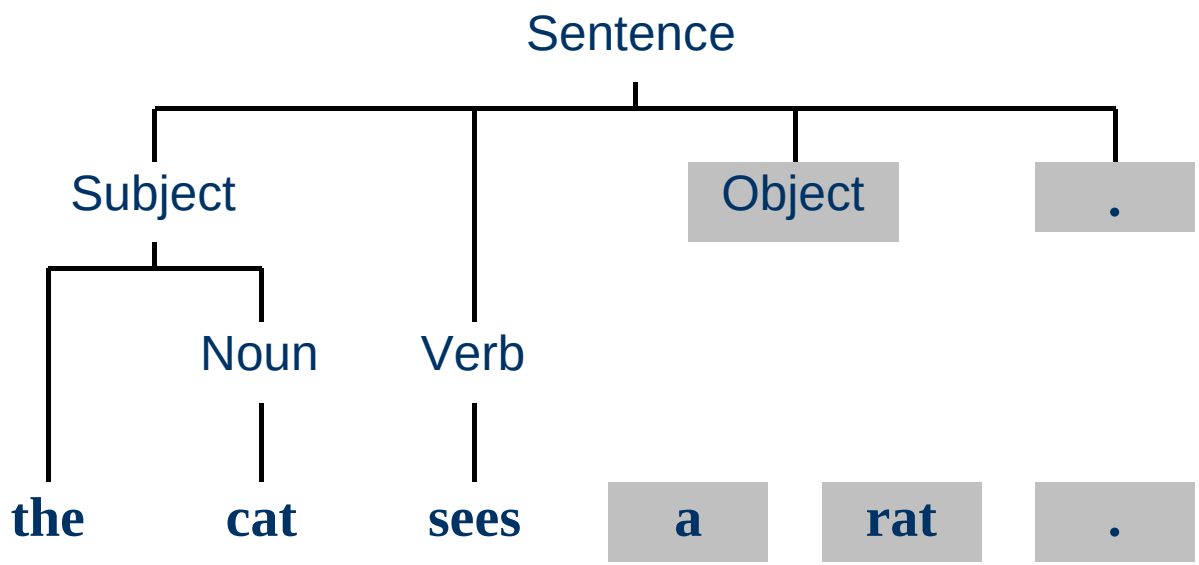
Embedded Systems & Applications

Beispiel Top-Down Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence	::=	Subject Verb Object **.**
Subject	::=	**I** | **a** Noun | **the** Noun
Object	::=	**me** | **a** Noun | **the** Noun
Noun	::=	**cat** | **mat** | **rat**
Verb	::=	**like** | **is** | **see** | **sees**



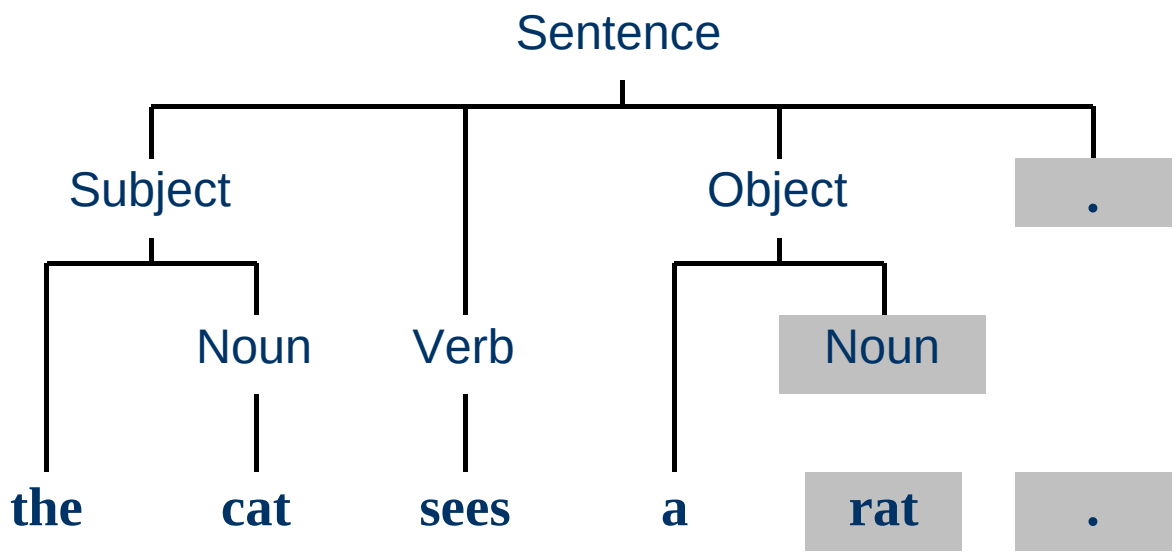
Embedded Systems & Applications

Beispiel Top-Down Parsing

`the cat sees a rat .`

TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence	::=	Subject Verb Object **`.`**
Subject	::=	**`I`** | **`a`** Noun | **`the`** Noun
Object	::=	**`me`** | **`a`** Noun | **`the`** Noun
Noun	::=	**`cat`** | **`mat`** | **`rat`**
Verb	::=	**`like`** | **`is`** | **`see`** | **`sees`**



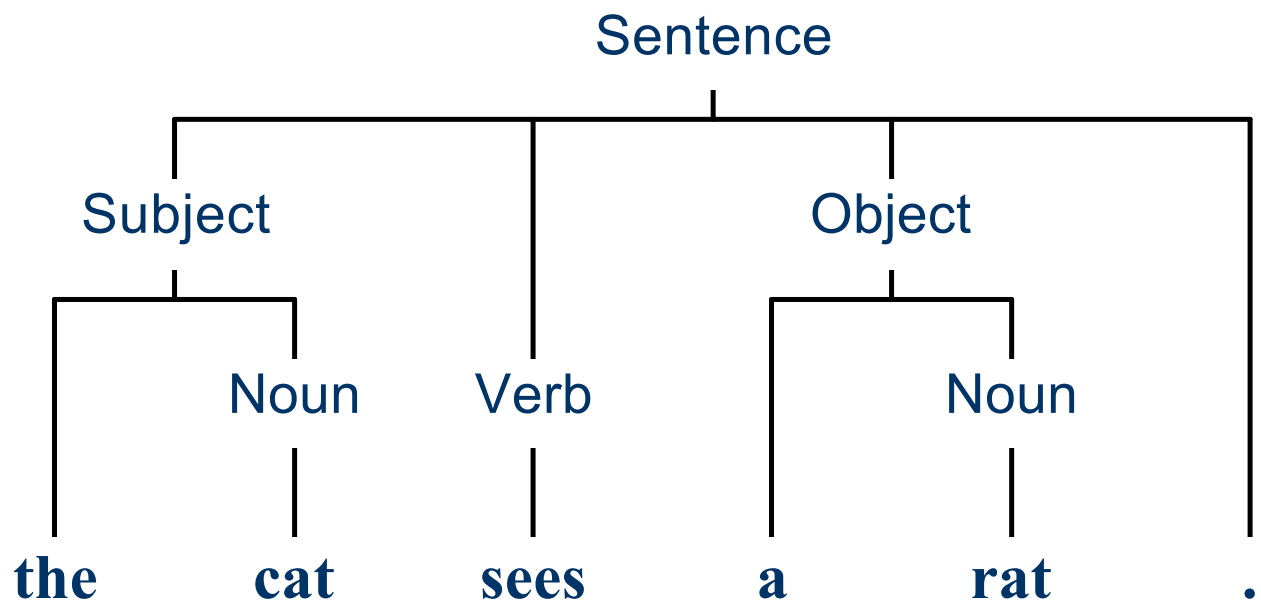
Embedded Systems & Applications

Beispiel Top-Down Parsing

`the cat sees a rat .`

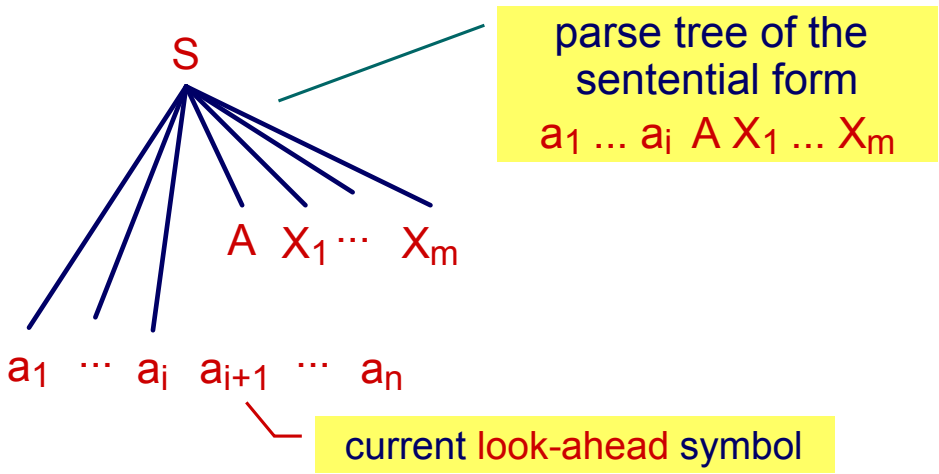
TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence	::=	Subject Verb Object **`.`**
Subject	::=	**`I`** | **`a`** Noun | **`the`** Noun
Object	::=	**`me`** | **`a`** Noun | **`the`** Noun
Noun	::=	**`cat`** | **`mat`** | **`rat`**
Verb	::=	**`like`** | **`is`** | **`see`** | **`sees`**



Embedded Systems & Applications

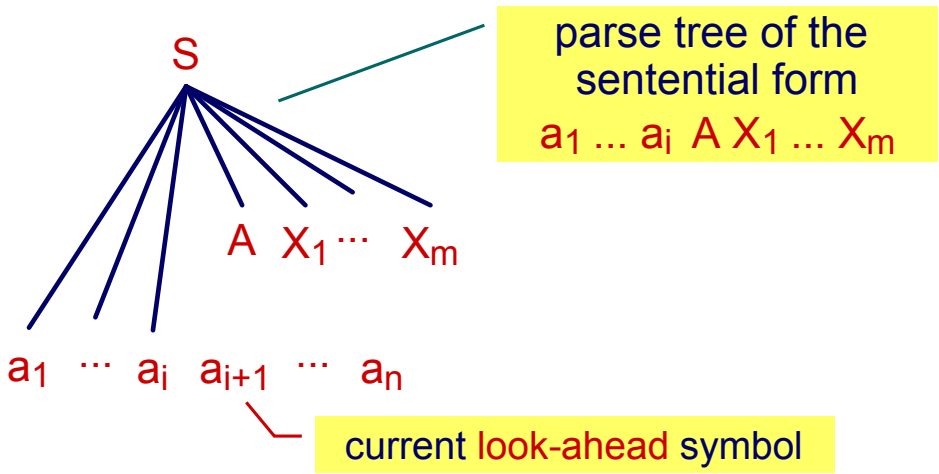
Hintergrund Top-Down Parsing



Falls es möglich ist,

- . . . bei Betrachten der nächsten *k* Zeichen des Textes
- . . . immer die richtige Produktion zu finden

Hintergrund Top-Down Parsing



Falls es möglich ist,

- . . . bei Betrachten der nächsten *k* Zeichen des Textes
- . . . immer die richtige Produktion zu finden

dann ist die Grammatik LL(*k*)

- L: Lese Eingabetext von links nach rechts
- L: Leite immer vom am weitesten links stehenden Nicht-Terminal ab.

Vergleich Top-Down mit Bottom-Up Parsing

TECHNISCHE UNIVERSITÄT DARMSTADT

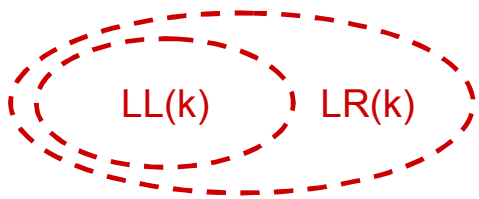
- Probleme mit Top-Down-Parsing
  - Konstruktion einer LL(k) Grammatik für die gewünschte Sprache gelegentlich mühsam
  - Linksausklammern und Beseitigen von Linksrekursion können Lesbarkeit der Grammatik erschweren

Vergleich Top-Down mit Bottom-Up Parsing

TECHNISCHE UNIVERSITÄT DARMSTADT

- Probleme mit Top-Down-Parsing
  - Konstruktion einer LL(k) Grammatik für die gewünschte Sprache gelegentlich mühsam
  - Linksausklammern und Beseitigen von Linksrekursion können Lesbarkeit der Grammatik erschweren

- Lösung: Bottom-Up-Parsing mit LR(k)-Techniken
  - L: Lese Eingabetext von links nach rechts
  - R: Fasse die am weitesten rechts stehenden Terminal-Symbole zusammen und baue den Baum rückwärts auf
  - Mächtigeres Beschreibungsinstrument als LL(k)
  - Nachteil: Parsing-Vorgang komplexer und schlechter verständlich



Embedded Systems & Applications

Parsen mit rekursivem Abstieg

TECHNISCHE UNIVERSITÄT DARMSTADT

Einfache Implementierung der Top-Down Strategie, Idee:

- Struktur des konkreten Syntaxbaumes (Parse-Baum) entspricht
- . . . Aufrufmuster von sich wechselseitig aufrufenden Prozeduren
- Für jedes Nicht-Terminal **XYZ** existiert
- . . . Parse-Prozedur **`parseXYZ`**, die genau dieses Nicht-Terminal parst

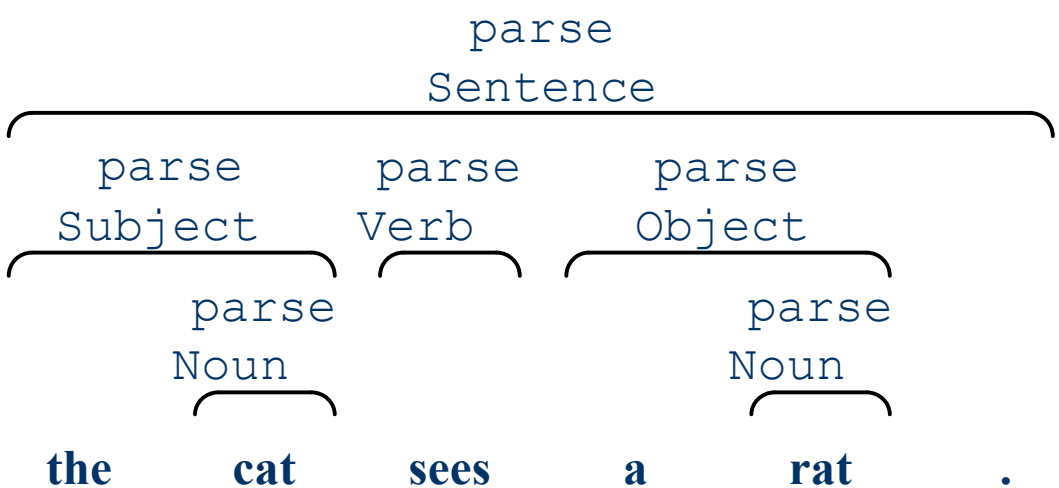
Parsen mit rekursivem Abstieg

TECHNISCHE UNIVERSITÄT DARMSTADT

Einfache Implementierung der Top-Down Strategie, Idee:

- Struktur des konkreten Syntaxbaumes (Parse-Baum) entspricht
- . . . Aufrufmuster von sich wechselseitig aufrufenden Prozeduren
- Für jedes Nicht-Terminal **XYZ** existiert
- . . . Parse-Prozedur **`parseXYZ`**, die genau dieses Nicht-Terminal parst

Beispiel:



Embedded Systems & Applications

Beispiel für Micro-English 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Sentence ::= Subject Verb Object .

```
protected void parseSentence() {
    parseSubject();
    parseVerb();
    parseObject();
    accept(".");
}
```

`accept(t)` prüft, ob aktuelles Token das erwartete Token `t` ist.

Embedded Systems & Applications

Beispiel für Micro-English 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Subject ::= **I** | **a** Noun | **the** Noun

```
protected void parseSubject() {
   if (currentToken matches "I") {
      accept("I");
   } else if (currentToken matches "a") {
      accept("a");
      parseNoun();
   } else if (currentToken matches "the") {
      accept("the");
      parseNoun();
   } else
      report a syntax error
}
```

Die Methode muß immer anhand von currentToken die passende Alternative auswählen können.

Embedded Systems & Applications

Beispiel für Micro-English 3

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MicroEnglishParser {
   protected Token currentToken;

   public void parse() {
      currentToken = first token;
      parseSentence();
      check that no token follows the sentence
   }

   protected void accept(Token expected) { ... }
   protected void parseSentence() { ... }
   protected void parseSubject() { ... }
   protected void parseObject() { ... }
   protected void parseNoun() { ... }
   protected void parseVerb() { ... }

   ...
}
```


Embedded Systems & Applications

Beispiel für Micro-English 3

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MicroEnglishParser {
  protected Token currentToken;

    public void parse() {
        currentToken = first token;
        parseSentence();
        check that no token follows the sentence
    }

    protected void accept(Token expected) { ... }
    protected void parseSentence() { ... }
    protected void parseSubject() { ... }
    protected void parseObject() { ... }
    protected void parseNoun() { ... }
    protecte

    ...
}
```

Schnittstelle zum Scanner, der die Tokens liefert

In Watt & Brown sind die Parse-Methoden als **`private`** deklariert. Ungeschickt, da es die Anpassung des Verhaltens durch Vererbung verhindert.

Embedded Systems & Applications

Erläuterung

TECHNISCHE UNIVERSITÄT DARMSTADT

- **`currentToken`** enthält nacheinander die Tokens des Eingabetextes

Erläuterung

TECHNISCHE UNIVERSITÄT DARMSTADT

- **`currentToken`** enthält nacheinander die Tokens des Eingabetextes
- Ablauf einer Methode **`parseN`**
  - Bei Eintritt enthält **`currentToken`** eines der Token, mit denen **N** beginnen kann
  - . . . sonst wäre eine andere Parse-Methode aufgerufen werden (oder Syntaxfehler)
  - Bei Austritt enthält **`currentToken`** das auf die **N**-Phrase folgende Token

Erläuterung

TECHNISCHE UNIVERSITÄT DARMSTADT

- **`currentToken`** enthält nacheinander die Tokens des Eingabetextes
- Ablauf einer Methode **`parseN`**
  - Bei Eintritt enthält **`currentToken`** eines der Token, mit denen **N** beginnen kann
  - . . . sonst wäre eine andere Parse-Methode aufgerufen werden (oder Syntaxfehler)
  - Bei Austritt enthält **`currentToken`** das auf die **N**-Phrase folgende Token
- Ablauf der Methode **`accept(`***t***`)`**
  - Bei Eintritt muß **`currentToken`** $=$ *t* sein
  - . . . sonst Syntaxfehler
  - Bei Austritt enthält **`currentToken`** das auf *t* folgende Token

Systematische Konstruktion von Parsern

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwicklung von Parsern mit rekursivem Abstieg

1. Formuliere Grammatik (CFG) in EBNF
   - Eine Produktion pro Nicht-Terminal
   - Beseitige immer Linksrekursion
   - Klammere gemeinsame Teilausdrücks nach links aus wo möglich

Systematische Konstruktion von Parsern

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwicklung von Parsern mit rekursivem Abstieg

1. Formuliere Grammatik (CFG) in EBNF
   - Eine Produktion pro Nicht-Terminal
   - Beseitige immer Linksrekursion
   - Klammere gemeinsame Teilausdrücks nach links aus wo möglich
2. Erstelle Klasse für den Parser mit
   - **`protected`** Variable **`currentToken`**
   - Schnittstellenmethoden zum Scanner
     - **`accept(`***t*`)`** und **`acceptIt()`**
   - **`public`** Methode **`parse`**, welche ...
     - erstes Token via Scanner aus dem Eingabetext liest
     - die Parse-Methode des Start Nicht-Terminals *S* der CFG aufruft

Systematische Konstruktion von Parsern

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwicklung von Parsern mit rekursivem Abstieg

1. Formuliere Grammatik (CFG) in EBNF
   - Eine Produktion pro Nicht-Terminal
   - Beseitige immer Linksrekursion
   - Klammere gemeinsame Teilausdrücks nach links aus wo möglich
2. Erstelle Klasse für den Parser mit
   - **`protected`** Variable **`currentToken`**
   - Schnittstellenmethoden zum Scanner
     - **`accept(`***t*`)`** und **`acceptIt()`**
   - **`public`** Methode **`parse`**, welche ...
     - erstes Token via Scanner aus dem Eingabetext liest
     - die Parse-Methode des Start Nicht-Terminals *S* der CFG aufruft
3. Implementiere **`protected`** Parsing-Methoden
   - Methode **`parseN`** für jedes Nicht-Terminalsymbol **`N`**

Anfangsmengen

TECHNISCHE UNIVERSITÄT DARMSTADT

starters[[**X**]] mit EBNF-Ausdruck **X**

Menge aller Terminal-Symbole, die am Anfang einer aus **X** herleitbaren Zeichenkette stehen können.

Anfangsmengen

TECHNISCHE UNIVERSITÄT DARMSTADT

starters$[[\mathbf{X}]]$ mit EBNF-Ausdruck **X**

Menge aller Terminal-Symbole, die am Anfang einer aus **X** herleitbaren Zeichenkette stehen können.

Beispiele

$$\begin{aligned} \text{starters}[[\mathbf{ab}]] &= \{\mathbf{a}\} \\ \text{starters}[[\mathbf{a}|\mathbf{b}]] &= \{\mathbf{a}, \mathbf{b}\} \\ \text{starters}[[(\mathbf{re}) * \mathbf{set}]] &= \{\mathbf{r}, \mathbf{s}\} \end{aligned}$$

Berechnungsregeln für starters$[[\mathbf{X}]]$

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{aligned}
\text{starters}[[\varepsilon]] &= \{\} \\
\text{starters}[[\mathbf{t}]] &= \{\mathbf{t}\} \\
\text{starters}[[\mathbf{XY}]] &= \begin{cases} \text{starters}[[\mathbf{X}]]\text{: falls aus } \mathbf{X} \text{ kein } \varepsilon \text{ herleitbar} \\ \text{starters}[[\mathbf{X}]] \cup \text{starters}[[\mathbf{Y}]]\text{: sonst} \end{cases} \\
\text{starters}[[\mathbf{X}|\mathbf{Y}]] &= \text{starters}[[\mathbf{X}]] \cup \text{starters}[[\mathbf{Y}]] \quad \text{noch nicht ganz richtig!} \\
\text{starters}[[\mathbf{X}*]] &= \text{starters}[[\mathbf{X}]] \quad \text{dito!} \\
\text{starters}[[\mathbf{N}*]] &= \text{starters}[[\mathbf{X}]]\text{, wenn } \mathbf{N} ::= \mathbf{X} \quad \text{dito!}
\end{aligned}
$$

Embedded Systems & Applications

Berechnungsregeln für starters$[[\mathbf{X}]]$

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{aligned}
\text{starters}[[\varepsilon]] &= \{\} \\
\text{starters}[[\mathbf{t}]] &= \{\mathbf{t}\} \\
\text{starters}[[\mathbf{XY}]] &= \begin{cases} \text{starters}[[\mathbf{X}]]\text{: falls aus } \mathbf{X} \text{ kein } \varepsilon \text{ herleitbar} \\ \text{starters}[[\mathbf{X}]] \cup \text{starters}[[\mathbf{Y}]]\text{: sonst} \end{cases} \\
\text{starters}[[\mathbf{X}|\mathbf{Y}]] &= \text{starters}[[\mathbf{X}]] \cup \text{starters}[[\mathbf{Y}]] \quad \text{noch nicht ganz richtig!} \\
\text{starters}[[\mathbf{X}*]] &= \text{starters}[[\mathbf{X}]] \quad \text{dito!} \\
\text{starters}[[\mathbf{N}*]] &= \text{starters}[[\mathbf{X}]], \text{ wenn } \mathbf{N} ::= \mathbf{X} \quad \text{dito!}
\end{aligned}
$$

Ausbügeln der Ungenauigkeiten später (siehe Folie 52)

Embedded Systems & Applications

Inhalt der `parseN`-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: **N** ::= **X**, nun schrittweise Zerlegung von **X**

ε **;** (=leere Anweisung)

Embedded Systems & Applications

Inhalt der `parseN`-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: **N** ::= **X**, nun schrittweise Zerlegung von **X**

ε `;` (=leere Anweisung)

t `accept(`t`);`

Inhalt der `parseN`-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: **N** ::= **X**, nun schrittweise Zerlegung von **X**

ε	`;` (=leere Anweisung)
t	`accept(`*t*`);`
P	`parseP();`

Inhalt der `parseN`-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: **N** ::= **X**, nun schrittweise Zerlegung von **X**

ε	`;` (=leere Anweisung)
t	`accept(`*t*`);`
P	`parseP();`
P Q	`parseP();` `parseQ();`

Inhalt der `parseN`-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: **N** ::= **X**, nun schrittweise Zerlegung von **X**

ε `;` (=leere Anweisung)

t `accept(`t`);`

P `parseP();`

P Q
```
parseP();
parseQ();
```

P|**Q**
```
if (currentToken ∈ starters[[P]])   was bei P = ε?
   parseP();
else if (currentToken ∈ starters[[Q]])
   parseQ();
else
   melde Syntaxfehler
```


Embedded Systems & Applications

Inhalt der `parseN`-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme: **N** ::= **X**, nun schrittweise Zerlegung von **X**

ε `;` (=leere Anweisung)

t `accept(`t`);`

P `parseP();`

P Q
```
parseP();
parseQ();
```

P|Q
```
if (currentToken ∈ starters[[P]])   was bei P = ε?
   parseP();
else if (currentToken ∈ starters[[Q]])
   parseQ();
else
   melde Syntaxfehler
```

P$*$
```
while (currentToken ∈ starters[[P]])
      parseP();
```


Embedded Systems & Applications

Folgemengen

TECHNISCHE UNIVERSITÄT DARMSTADT

Analog: $\mathrm{follow}[[\mathbf{X}]]$ ist Menge der Tokens, die in der CFG nach **X** folgen können.

Folgemengen

TECHNISCHE UNIVERSITÄT DARMSTADT

Analog: $\mathrm{follow}[[\mathbf{X}]]$ ist Menge der Tokens, die in der CFG nach **X** folgen können.

1. Ausdruck innerhalb rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})\ \mathbf{c} \quad \rightarrow \mathrm{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{c}\}$

Folgemengen

TECHNISCHE UNIVERSITÄT DARMSTADT

Analog: $\text{follow}[[\mathbf{X}]]$ ist Menge der Tokens, die in der CFG nach **X** folgen können.

1. Ausdruck innerhalb rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})\ \mathbf{c} \quad \rightarrow \text{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{c}\}$
2. Ausdruck am Ende rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})$
   $\mathbf{B} ::= \mathbf{A}\ \mathbf{u}$
   $\mathbf{C} ::= \mathbf{A}\ \mathbf{v}$
   $\rightarrow \text{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{u}, \mathbf{v}\}$

Folgemengen

TECHNISCHE UNIVERSITÄT DARMSTADT

Analog: $\text{follow}[[\mathbf{X}]]$ ist Menge der Tokens, die in der CFG nach **X** folgen können.

1. Ausdruck innerhalb rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})\ \mathbf{c} \quad \rightarrow \text{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{c}\}$
2. Ausdruck am Ende rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})$
   $\mathbf{B} ::= \mathbf{A}\ \mathbf{u}$
   $\mathbf{C} ::= \mathbf{A}\ \mathbf{v}$
   $\rightarrow \text{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{u}, \mathbf{v}\}$
3. Nichtterminal auf linker Seite von Produktion:
   Vereinigung der Folgemenge aller Vorkommen auf rechten Seiten:
   $\text{follow}[[\mathbf{A}]] = \{\mathbf{u}, \mathbf{v}\}$

Folgemengen

TECHNISCHE UNIVERSITÄT DARMSTADT

Analog: $\text{follow}[[\mathbf{X}]]$ ist Menge der Tokens, die in der CFG nach **X** folgen können.

1. Ausdruck innerhalb rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})\ \mathbf{c} \quad \rightarrow \text{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{c}\}$
2. Ausdruck am Ende rechter Seite von Produktion:
   $\mathbf{A} ::= (\mathbf{a} \mid \mathbf{b})$
   $\mathbf{B} ::= \mathbf{A}\ \mathbf{u}$
   $\mathbf{C} ::= \mathbf{A}\ \mathbf{v}$
   $\rightarrow \text{follow}[[\mathbf{a} \mid \mathbf{b}]] = \{\mathbf{u}, \mathbf{v}\}$
3. Nichtterminal auf linker Seite von Produktion:
   Vereinigung der Folgemenge aller Vorkommen auf rechten Seiten:
   $\text{follow}[[\mathbf{A}]] = \{\mathbf{u}, \mathbf{v}\}$
4. Startsymbol: leere Menge

Anwendbarkeit der Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktionieren nur dann, wenn in Grammatik *G* gilt:

Anwendbarkeit der Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktionieren nur dann, wenn in Grammatik *G* gilt:

- Falls *G* **X**|**Y** enthält und sich weder X noch Y zu ε ableiten lassen:
  $\text{starters}[[\mathbf{X}]] \cap \text{starters}[[\mathbf{Y}]] = \emptyset$

Anwendbarkeit der Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktionieren nur dann, wenn in Grammatik *G* gilt:

- Falls *G* **X**|**Y** enthält und sich weder X noch Y zu ε ableiten lassen:
  $\text{starters}[[\mathbf{X}]] \cap \text{starters}[[\mathbf{Y}]] = \emptyset$
- Falls *G* **X**|**Y** enthält und sich beispielsweise **Y** zu ε ableiten lässt:
  $\text{starters}[[\mathbf{X}]] \cap (\text{starters}[[\mathbf{Y}]] \cup \text{follow}[[\mathbf{X}|\mathbf{Y}]]) = \emptyset$

Anwendbarkeit der Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktionieren nur dann, wenn in Grammatik *G* gilt:

- Falls *G* **X**|**Y** enthält und sich weder X noch Y zu ε ableiten lassen:
  $\text{starters}[[\mathbf{X}]] \cap \text{starters}[[\mathbf{Y}]] = \emptyset$
- Falls *G* **X**|**Y** enthält und sich beispielsweise **Y** zu ε ableiten lässt:
  $\text{starters}[[\mathbf{X}]] \cap (\text{starters}[[\mathbf{Y}]] \cup \text{follow}[[\mathbf{X}|\mathbf{Y}]]) = \emptyset$
- Falls *G* **X**$*$ enthält: $\text{starters}[[\mathbf{X}]] \cap \text{follow}[[\mathbf{X}*]] = \emptyset$

Embedded Systems & Applications

Anwendbarkeit der Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktionieren nur dann, wenn in Grammatik *G* gilt:

- Falls *G* **X**|**Y** enthält und sich weder X noch Y zu ε ableiten lassen:
  $\text{starters}[[\mathbf{X}]] \cap \text{starters}[[\mathbf{Y}]] = \emptyset$
- Falls *G* **X**|**Y** enthält und sich beispielsweise **Y** zu ε ableiten lässt:
  $\text{starters}[[\mathbf{X}]] \cap (\text{starters}[[\mathbf{Y}]] \cup \text{follow}[[\mathbf{X}|\mathbf{Y}]]) = \emptyset$
- Falls *G* **X**$*$ enthält: $\text{starters}[[\mathbf{X}]] \cap \text{follow}[[\mathbf{X}*]] = \emptyset$

Embedded Systems & Applications

Anwendbarkeit der Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Funktionieren nur dann, wenn in Grammatik *G* gilt:

- Falls *G* **X**|**Y** enthält und sich weder X noch Y zu ε ableiten lassen:
  $\text{starters}[[\mathbf{X}]] \cap \text{starters}[[\mathbf{Y}]] = \emptyset$
- Falls *G* **X**|**Y** enthält und sich beispielsweise **Y** zu ε ableiten lässt:
  $\text{starters}[[\mathbf{X}]] \cap (\text{starters}[[\mathbf{Y}]] \cup \text{follow}[[\mathbf{X}|\mathbf{Y}]]) = \emptyset$
- Falls *G* **X**$*$ enthält: $\text{starters}[[\mathbf{X}]] \cap \text{follow}[[\mathbf{X}*]] = \emptyset$

➥Wenn alles gilt: *G* ist LL(*k*) mit $k = 1$

Hinweis: Definition in PLPJ, p. 104 ist nicht ausreichend!

Embedded Systems & Applications

Nachweis der LL(1)-Eigenschaft

TECHNISCHE
UNIVERSITÄT
DARMSTADT

$$
\begin{array}{lcl}
\mathbf{S} & ::= & \mathbf{Xd} \\
\mathbf{X} & ::= & \mathbf{A} \mid \mathbf{B} \mid \mathbf{c} \\
\mathbf{A} & ::= & \mathbf{a}* \\
\mathbf{B} & ::= & \mathbf{b}
\end{array}
$$

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

- **S** ::= **Xd**, **B** ::= **b**
  - Nichts zu prüfen

S ::= **Xd**
X ::= **A** | **B** | **c**
A ::= **a** $*$
B ::= **b**

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{array}{lcl}
\mathbf{S} & ::= & \mathbf{Xd} \\
\mathbf{X} & ::= & \mathbf{A} \mid \mathbf{B} \mid \mathbf{c} \\
\mathbf{A} & ::= & \mathbf{a}* \\
\mathbf{B} & ::= & \mathbf{b}
\end{array}
$$

- $\mathbf{S} ::= \mathbf{Xd}$, $\mathbf{B} ::= \mathbf{b}$
  - Nichts zu prüfen
- $\mathbf{X} ::= \mathbf{A} \mid \mathbf{B} \mid \mathbf{c}$

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{aligned}
\mathbf{S} &::= \mathbf{Xd} \\
\mathbf{X} &::= \mathbf{A} \mid \mathbf{B} \mid \mathbf{c} \\
\mathbf{A} &::= \mathbf{a}* \\
\mathbf{B} &::= \mathbf{b}
\end{aligned}
$$

- $\mathbf{S} ::= \mathbf{Xd}$, $\mathbf{B} ::= \mathbf{b}$
  - Nichts zu prüfen
- $\mathbf{X} ::= \mathbf{A} \mid \mathbf{B} \mid \mathbf{c}$
  - Produktion enthält 3 Alternativen $\rightarrow$ paarweise prüfen!

Embedded Systems & Applications

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{array}{lcl}
\mathbf{S} & ::= & \mathbf{Xd} \\
\mathbf{X} & ::= & \mathbf{A} \mid \mathbf{B} \mid \mathbf{c} \\
\mathbf{A} & ::= & \mathbf{a}* \\
\mathbf{B} & ::= & \mathbf{b}
\end{array}
$$

- **S** ::= **Xd**, **B** ::= **b**
  - Nichts zu prüfen
- **X** ::= **A** | **B** | **c**
  - Produktion enthält 3 Alternativen → paarweise prüfen!
  - **B** | **c**
    $\text{starters}[[B]] \cap \text{starters}[[c]] = \emptyset$ ✓

Embedded Systems & Applications

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{array}{lcl}
\mathbf{S} & ::= & \mathbf{Xd} \\
\mathbf{X} & ::= & \mathbf{A} \mid \mathbf{B} \mid \mathbf{c} \\
\mathbf{A} & ::= & \mathbf{a}* \\
\mathbf{B} & ::= & \mathbf{b}
\end{array}
$$

- **S** ::= **Xd**, **B** ::= **b**
  - Nichts zu prüfen
- **X** ::= **A** | **B** | **c**
  - Produktion enthält 3 Alternativen → paarweise prüfen!
  - **B** | **c**
    $\text{starters}[[B]] \cap \text{starters}[[c]] = \emptyset$ ✓
  - **A** | **B**
    A kann zu ε abgeleitet werden!
    $(\text{starters}[[A]] \cup \text{follow}[[A \mid B]]) \cap \text{starters}[[B]] = \emptyset$ ✓

Embedded Systems & Applications

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

S ::= **Xd**
X ::= **A** | **B** | **c**
A ::= **a** $*$
B ::= **b**

- **S** ::= **Xd**, **B** ::= **b**
  - Nichts zu prüfen
- **X** ::= **A** | **B** | **c**
  - Produktion enthält 3 Alternativen $\rightarrow$ paarweise prüfen!
  - **B** | **c**
    $\text{starters}[[B]] \cap \text{starters}[[c]] = \emptyset$ ✓
  - **A** | **B**
    A kann zu ε abgeleitet werden!
    $(\text{starters}[[A]] \cup \text{follow}[[A \mid B]]) \cap \text{starters}[[B]] = \emptyset$ ✓
  - **A** | **c** analog

Embedded Systems & Applications

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

S ::= **Xd**
X ::= **A** | **B** | **c**
A ::= **a** $*$
B ::= **b**

- **S** ::= **Xd**, **B** ::= **b**
  - Nichts zu prüfen
- **X** ::= **A** | **B** | **c**
  - Produktion enthält 3 Alternativen $\rightarrow$ paarweise prüfen!
  - **B** | **c**
    $\text{starters}[[B]] \cap \text{starters}[[c]] = \emptyset$ ✓
  - **A** | **B**
    A kann zu ε abgeleitet werden!
    $(\text{starters}[[A]] \cup \text{follow}[[A \mid B]]) \cap \text{starters}[[B]] = \emptyset$ ✓
  - **A** | **c** analog
- **A** ::= **a**$*$

Embedded Systems & Applications

Nachweis der LL(1)-Eigenschaft

TECHNISCHE UNIVERSITÄT DARMSTADT

$$
\begin{array}{lcl}
\mathbf{S} & ::= & \mathbf{Xd} \\
\mathbf{X} & ::= & \mathbf{A} \mid \mathbf{B} \mid \mathbf{c} \\
\mathbf{A} & ::= & \mathbf{a}* \\
\mathbf{B} & ::= & \mathbf{b}
\end{array}
$$

- **S** $::=$ **Xd**, **B** $::=$ **b**
  - Nichts zu prüfen
- **X** $::=$ **A** $\mid$ **B** $\mid$ **c**
  - Produktion enthält 3 Alternativen $\rightarrow$ paarweise prüfen!
  - **B** $\mid$ **c**
    $\text{starters}[[B]] \cap \text{starters}[[c]] = \emptyset$ ✓
  - **A** $\mid$ **B**
    A kann zu ε abgeleitet werden!
    $(\text{starters}[[A]] \cup \text{follow}[[A \mid B]]) \cap \text{starters}[[B]] = \emptyset$ ✓
  - **A** $\mid$ **c** analog
- **A** $::=$ **a**$*$
  - **a**$*$
    $\text{starters}[[a]] \cap \text{follow}[[a*]] = \emptyset$ ✓

Embedded Systems & Applications

Verfeinerte Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Bisher gezeigt für **P**|**Q**

```
if (currentToken ∈ starters[[P]])
   parseP();
else if (currentToken ∈ starters[[Q]])
   parseQ();
else
   melde Syntaxfehler
```

Verfeinerte Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Bisher gezeigt für **P**|**Q**

```
if (currentToken ∈ starters[[P]])
   parseP();
else if (currentToken ∈ starters[[Q]])
   parseQ();
else
   melde Syntaxfehler
```

Problematisch, wenn ε aus **P** oder **Q** ableitbar.

Verfeinerte Zerlegungsregeln

TECHNISCHE UNIVERSITÄT DARMSTADT

Bisher gezeigt für **P**|**Q**

```
if (currentToken ∈ starters[[P]])
   parseP();
else if (currentToken ∈ starters[[Q]])
   parseQ();
else
   melde Syntaxfehler
```

Problematisch, wenn ε aus **P** oder **Q** ableitbar.
Korrekt: Verwende statt $\text{starters}[[\mathbf{X}]]$

$$\text{dirset}[[\mathbf{X}]] = \begin{cases} \text{starters}[[\mathbf{X}]]\text{: falls aus } \mathbf{X} \text{ kein } \varepsilon \text{ herleitbar} \\ \text{starters}[[\mathbf{X}]] \cup \text{follow}[[\mathbf{X}]]\text{: sonst} \end{cases}$$

Analog für $\mathbf{P}*$. Korrigiere so Folie 48.

Embedded Systems & Applications

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

- Prüfe Regel für **X**$*$

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

- Prüfe Regel für **X**$*$
  - starters$[[$**`;` Declaration**$]] = \{$**`;`**$\}$

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

- Prüfe Regel für **X**$*$
  - starters$[[$**`;` Declaration**$]] = \{$**`;`**$\}$
  - follow$[[($**`;` Declaration**$)*]] = \{$**`;`**$\}$

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

- Prüfe Regel für **X**$*$
  - starters$[[$**`;` Declaration**$]] = \{$**`;`**$\}$
  - follow$[[($**`;` Declaration**$)*]] = \{$**`;`**$\}$
  - starters$[[$**`;` Declaration**$]] \cap$ follow$[[($**`;` Declaration**$)*]]$
    $\neq \emptyset$

Embedded Systems & Applications

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

- Prüfe Regel für **X**$*$
  - starters$[[$**`;` Declaration**$]] = \{$**`;`**$\}$
  - follow$[[($**`;` Declaration**$)*]] = \{$**`;`**$\}$
  - starters$[[$**`;` Declaration**$]] \cap$ follow$[[($**`;` Declaration**$)*]]$ $\neq \emptyset$

- Produktion ist aber transformierbar
  Block ::= **`begin` Declaration `;`** (**Declaration `;`**)$*$ **Command `end`**

Embedded Systems & Applications

Beispiel für nicht-LL(1) Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aus Algol Grammatik
  Block ::= **`begin` Declaration** (**`;` Declaration**)$*$ **`;` Command `end`**

- Prüfe Regel für **X**$*$
  - starters$[[$**`;` Declaration**$]] = \{$**`;`**$\}$
  - follow$[[$(**`;` Declaration**)$*]] = \{$**`;`**$\}$
  - starters$[[$**`;` Declaration**$]] \cap$ follow$[[$(**`;` Declaration**)$*]]$
    $\neq \emptyset$

- Produktion ist aber transformierbar
  Block ::= **`begin` Declaration `;`** (**Declaration `;`**)$*$ **Command `end`**
- Annahme: starters$[[$**Declaration`;`**$]] \cap$starters$[[$**Command**$]] = \emptyset$

LL(k)-Parser

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme bis 1992

Rekursiver Abstieg funktioniert sinnvoll nur für $k = 1$, exponentieller Worst-Case-Aufwand bei $k > 1$.

LL(k)-Parser

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme bis 1992

Rekursiver Abstieg funktioniert sinnvoll nur für $k = 1$, exponentieller Worst-Case-Aufwand bei $k > 1$.

Gegenbeispiel 1992: PCCTS (jetzt ANTLR)

Worst-case kann für Grammatiken typischer Programmiersprachen in der Regel vermieden werden, sogar bei $k = \infty$.

LL(k)-Parser

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahme bis 1992

Rekursiver Abstieg funktioniert sinnvoll nur für $k = 1$, exponentieller Worst-Case-Aufwand bei $k > 1$.

Gegenbeispiel 1992: PCCTS (jetzt ANTLR)

Worst-case kann für Grammatiken typischer Programmiersprachen in der Regel vermieden werden, sogar bei $k = \infty$.

- Konstruktion von Top-Down-Parsern gut automatisierbar
- Für Java beispielsweise
  - ANTLR: LL(k) bis LL($*$)
  - JavaCC: LL(k)

Parser für Mini-Triangle: `parseCommand`

TECHNISCHE UNIVERSITÄT DARMSTADT

Command ::= single-Command (; single-Command)*

Embedded Systems & Applications

Parser für Mini-Triangle: `parseCommand`

TECHNISCHE UNIVERSITÄT DARMSTADT

Command ::= single-Command (; single-Command)*

```
Command parseCommand() {
  commandAST = parseSingleCommand();
  while (currentToken.kind ==
      TokenKind.SEMICOLON) {
    acceptIt();
    parseSingleCommand();
  }
}
```

`acceptIt()`

- Könnte auch **`accept(TokenKind.SEMICOLON)`** sein
- Würde aber überflüssige Fehlerüberprüfung vornehmen
  - Token wurde schon vorher in **`while(...)`** geprüft
- Also ohne weitere Bearbeitung akzeptieren

Embedded Systems & Applications

Parser für Mini-Triangle: `parseSingleCommand`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
single-Command  ::= Identifier  ( :=  Expression
                                | (   Expression  ) )
                  |   ...
```


Embedded Systems & Applications

Parser für Mini-Triangle: `parseSingleCommand`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
single-Command  ::= Identifier  ( := Expression
                                | ( Expression ) )
                  |  ...
```

```
Command parseSingleCommand() {
  switch (currentToken.kind) {
  case IDENTIFIER: {
    parseIdentifier();
    switch (currentToken.kind) {
      case BECOMES:
        acceptIt();
        parseExpression();
        break;
      case LPAREN:
        acceptIt();
        parseExpression();
        accept(TokenKind.RPAREN);
        break;
      default: /* melde Syntaxfehler */
    }
    break; // case IDENTIFIER
  }
  ...
```

Weitere Beispiele in PLPJ.

Embedded Systems & Applications

Sonderfälle

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Aufpassen bei
  - `parseIdentifier`
  - `parseIntegerLiteral`
  - `parseOperator`

Sonderfälle

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aufpassen bei
  - **`parseIdentifier`**
  - **`parseIntegerLiteral`**
  - **`parseOperator`**
- . . . hier nicht nur Art des Tokens relevant
- sondern tatsächlicher Text
  - **`TokenKind.IDENTIFIER`**: foo, bar, pi, k9, . . .
  - **`TokenKind.INTLITERAL`**: 23, 42, 2006, . . .
  - **`TokenKind.OPERATOR`**: +, -, /, . . .

Sonderfälle

TECHNISCHE UNIVERSITÄT DARMSTADT

- Aufpassen bei
  - **`parseIdentifier`**
  - **`parseIntegerLiteral`**
  - **`parseOperator`**
- . . . hier nicht nur Art des Tokens relevant
- sondern tatsächlicher Text
  - **`TokenKind.IDENTIFIER`**: foo, bar, pi, k9, . . .
  - **`TokenKind.INTLITERAL`**: 23, 42, 2006, . . .
  - **`TokenKind.OPERATOR`**: +, -, /, . . .

➥Eingabetext nicht nur auf Token-Art reduzieren, Text selbst muß erhalten bleiben

Häufige Fehler: Grammatik ist nicht LL(1)

Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

Auszug aus Grammatik

single-Command ::= V-name **:=** Expression

| Identifier **(** Expression **)**

| **if** Expression **then** single-Command

else single-Command

| …

Embedded Systems & Applications

Häufige Fehler: Grammatik ist nicht LL(1)

Grammatik

TECHNISCHE UNIVERSITÄT DARMSTADT

Auszug aus Grammatik

single-Command ::= V-name **:=** Expression
| Identifier **(** Expression **)**
| **if** Expression **then** single-Command
else single-Command
| …

Anfangsmengen

starters[[V-name **:=** Expression]] = starters[[V-name]]
= { Identifier }
starters[[Identifier **(** Expression **)**]] = { Identifier }
starters[[**if** Expression **then** …]] = { **if** }

Embedded Systems & Applications

Häufige Fehler: Grammatik ist nicht LL(1)

Implementierung des Parsers

TECHNISCHE UNIVERSITÄT DARMSTADT

Durch Zerlegung gewonnener Java-Code

```
Command parseSingleComand() {
  switch (currentToken.kind) {
  case IDENTIFIER:
    parseVname();
    accept(TokenKind.BECOMES);
    parseExpression();
    break;

  case IDENTIFIER:
    parseIdentifier();
    accept(TokenKind.LPAREN);
    parseExpression();
    accept(TokenKind.RPAREN);
    break;

  case IF: ...
```

Häufige Fehler: Linksausklammern vergessen

TECHNISCHE UNIVERSITÄT DARMSTADT

Auszug aus Grammatik nach Ersetzen von **V-name** durch **Identifier**

single-Command ::= Identifier **:=** Expression
| Identifier **(** Expression **)**
| **if** Expression **then** single-Command
else single-Command

Häufige Fehler: Linksausklammern vergessen

TECHNISCHE UNIVERSITÄT DARMSTADT

Auszug aus Grammatik nach Ersetzen von **V-name** durch **Identifier**

single-Command ::= Identifier **:=** Expression
| Identifier **(** Expression **)**
| **if** Expression **then** single-Command
else single-Command

Anfangsmengen

starters[[Identifier **:=** Expression]] = { Identifier }

starters[[Identifier **(** Expression **)**]] = { Identifier }

Häufige Fehler: Linksausklammern vergessen

TECHNISCHE UNIVERSITÄT DARMSTADT

Jetzt mit Linksausklammern

single-Command ::= Identifier (**:=** Expression | **(** Expression **)**)
| **if** Expression **then** single-Command
else single-Command

Häufige Fehler: Linksausklammern vergessen

TECHNISCHE UNIVERSITÄT DARMSTADT

Jetzt mit Linksausklammern

single-Command ::= Identifier (**:=** Expression | **(** Expression **)**)
| **if** Expression **then** single-Command
else single-Command

Neue Anfangsmengen

starters[[**:=** Expression]] = { := }

starters[[**(** Expression **)**]] = { (}

Embedded Systems & Applications

Häufige Fehler: Linksrekursion nicht beseitigt

TECHNISCHE UNIVERSITÄT DARMSTADT

Auszug aus Grammatik vor Korrektur

```
Command ::= single-Command
          |   Command ; single-Command
```

Häufige Fehler: Linksrekursion nicht beseitigt

TECHNISCHE UNIVERSITÄT DARMSTADT

Auszug aus Grammatik vor Korrektur

Command ::= single-Command
| Command **;** single-Command

Anfangsmengen

starters[[single-Command]]

= { Identifier, **`if`**, **`while`**, **`let`**, **`begin`** }

starters[[Command **;** single-Command]]

= { Identifier, **`if`**, **`while`**, **`let`**, **`begin`** }

Häufige Fehler: Linksrekursion nicht beseitigt

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Command parseCommand() {
  switch (currentToken.kind) {
  case IDENTIFIER:
  case IF:  case WHILE:
  case LET: case BEGIN:
    parseSingleCommand();
    break;

  case IDENTIFIER:
  case IF:  case WHILE:
  case LET: case BEGIN:
    parseCommand();
    accept(";")
    parseSingleCommand();
    break;

  default: /* melde Syntaxfehler */
  }
}
```

Parser für Mini-Triangle: Grammatikanpassung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Program          ::= single-Command

Command          ::= single-Command
                 |   Command ; single-Command

single-Command   ::= V-name := Expression
                 |   Identifier ( Expression )
                 |   ...
```

Parser für Mini-Triangle: Grammatikanpassung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Program          ::= single-Command
Command          ::= single-Command
                   |   Command ; single-Command
single-Command   ::= V-name := Expression
                   |   Identifier ( Expression )
                   |   ...
```

Linksrekursion

Linksausklammern

Embedded Systems & Applications

Parser für Mini-Triangle: Grammatikanpassung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Program          ::= single-Command

Command          ::= single-Command
                         ( ; single-Command)*

single-Command   ::= Identifier ( := Expression
                                 |  ( Expression )  )
                   |  ...
```

Abstrakte Syntaxbäume 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Parser mit rekursivem Abstieg baut impliziten Syntaxbaum auf
  - Durch den Aufrufgraph der Parse-Methoden

Abstrakte Syntaxbäume 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Parser mit rekursivem Abstieg baut impliziten Syntaxbaum auf
  - Durch den Aufrufgraph der Parse-Methoden
- In einem Ein-Pass-Compiler unproblematisch

Abstrakte Syntaxbäume 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Parser mit rekursivem Abstieg baut impliziten Syntaxbaum auf
  - Durch den Aufrufgraph der Parse-Methoden
- In einem Ein-Pass-Compiler unproblematisch
- Reicht nicht für Multi-Pass Compiler
  - Weitergabe der Daten zwischen Passes erforderlich

Abstrakte Syntaxbäume 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beobachtung: Jedes Nicht-Terminalsymbol **XYZ** wird durch eine Parse-Methode **`parseXYZ`** bearbeitet
  `protected void parseXYZ ( )`
  - Bisher nicht benutzt: Funktionsergebnis und Parameter

Abstrakte Syntaxbäume 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beobachtung: Jedes Nicht-Terminalsymbol **XYZ** wird durch eine Parse-Methode **`parseXYZ`** bearbeitet
  `protected void parseXYZ ( )`
  - Bisher nicht benutzt: Funktionsergebnis und Parameter
- Idee: Ausnutzung der Möglichkeiten zum Aufbau eines AST

AST Knoten von Mini-Triangle

TECHNISCHE UNIVERSITÄT DARMSTADT

Program	::=	Command	Program
Command	::=	Command **;** Command	SequentialCmd
	|	V-name **:=** Expression	AssignCmd
	|	Identifier **(** Expression **)**	CallCmd
	|	**if** Expression **then** single-Command **else** single-Command	IfCmd
	|	**while** Expression **do** single-Command	WhileCmd
	|	**let** Declaration **in** single-Command	LetCmd
Expression	::=	Integer-Literal	IntegerExpr
	|	V-name	VnameExpr
	|	Operator Expression	UnaryExpr
	|	Expression Operator Expression	BinaryExpr
V-name	::=	Identifier	SimpleVname
Declaration	::=	Declaration **;** Declaration	SeqDecl
	|	**const** Identifier **~** Expression	ConstDecl
	|	**var** Identifier **:** Type-denoter	VarDecl
Type-denoter	::=	Identifier	SimpleTypeDen

Embedded Systems & Applications

AST Knoten von Mini-Triangle

TECHNISCHE UNIVERSITÄT DARMSTADT

Program	::=	Command	Program
Command	::=	Command **;** Command	SequentialCmd
	|	V-name **:=** Expression	AssignCmd
	|	Identifier **(** Expression **)**	CallCmd
	|	**if** Expression **then** single-Command **else** single-Command	IfCmd
	|	**while** Expression **do** single-Command	WhileCmd
	|	**let** Declaration **in** single-Command	LetCmd
Expression	::=	Integer-Literal	IntegerExpr
	|	V-name	VnameExpr
	|	Operator Expression	UnaryExpr
	|	Expression Operator Expression	BinaryExpr
V-name	::=	Identifier	SimpleVname
Declaration	::=	Declaration **;** Declaration	SeqDecl
	|	**const** Identifier **~** Expression	ConstDecl
	|	**var** Identifier **:** Type-denoter	VarDecl
Type-denoter	::=	Identifier	SimpleTypeDen

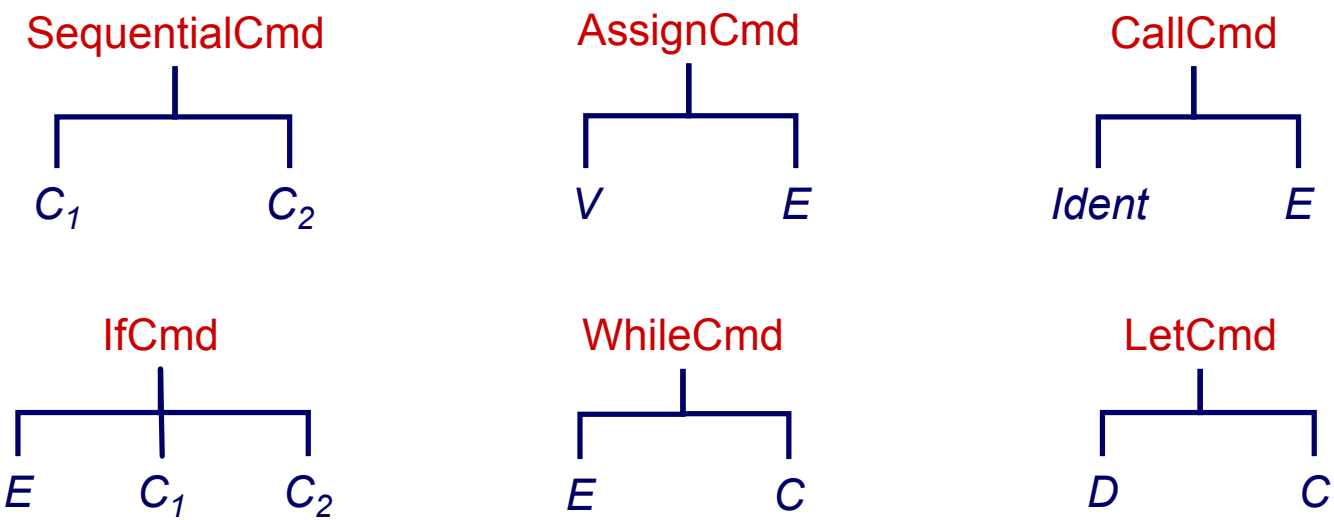
AST Knoten von Mini-Triangle

Embedded Systems & Applications

Sub-ASTs von Mini-Triangle

TECHNISCHE UNIVERSITÄT DARMSTADT

Command	::=	Command `;` Command	SequentialCmd
	|	V-name `:=` Expression	AssignCmd
	|	Identifier `(` Expression `)`	CallCmd
	|	`if` Expression `then` single-Command `else` single-Command	IfCmd
	|	`while` Expression `do` single-Command	WhileCmd
	|	`let` Declaration `in` single-Command	LetCmd



Embedded Systems & Applications

Java-Implementierung der ASTs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Abstrakte Basisklasse
  ```
  public abstract class AST { ... }
  ```
- Eigene Subklassen für alle Arten von AST-Knoten

Java-Implementierung der ASTs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Abstrakte Basisklasse
  `public abstract class AST { ... }`
- Eigene Subklassen für alle Arten von AST-Knoten

Jede Subklasse hat Instanzvariablen für ihre Unterknoten

```
public class Program extends AST {
  public Command C;
  ...
}
```

Java-Implementierung der ASTs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Abstrakte Basisklasse
  `public abstract class AST { ... }`
- Eigene Subklassen für alle Arten von AST-Knoten

Jede Subklasse hat Instanzvariablen für ihre Unterknoten

```
public class Program extends AST {
  public Command C;
  ...
}
```

Abstrakte Basisklasse aller Command AST-Knoten

```
public abstract class Command extends AST {
```

Unterklassen der `Command`-Klasse

TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
abstract class Command
  extends AST { ... }
```

Unterklassen der `Command`-Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract class Command
  extends AST { ... }
```

```
Command
      ::=   Command ; Command                          SequentialCmd
      |     V-name := Expression                       AssignCmd
      |     Identifier ( Expression )                  CallCmd
      |     if Expression then single-Command          IfCmd
                  else single-Command
      |     while Expression do single-Command         WhileCmd
      |     let Declaration in single-Command          LetCmd
```

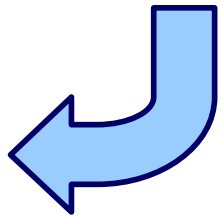
Unterklassen der `Command`-Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
abstract class Command
  extends AST { ... }
```

```
Command
    ::=  Command ; Command                          SequentialCmd
    |    V-name := Expression                       AssignCmd
    |    Identifier ( Expression )                  CallCmd
    |    if Expression then single-Command          IfCmd
              else single-Command
    |    while Expression do single-Command         WhileCmd
    |    let Declaration in single-Command          LetCmd
```

```
public class SequentialCmd extends Command {
   public Command c1, c2;
   ...
}
public class AssignCmd extends Command {
   public Vname      v;
   public Expression  e;
   ...
}
public class CallCmd extends Command {
   public Identifier  i;
   public Expression  e;
   ...
}
public class IfCmd extends Command {
   public Expression  e;
   public Command     c1, c2;
   ...
}
```



Die AST Subklassen haben auch entsprechende Konstruktoren zur korrekten Initialisierung der Objekte.

etc.

Embedded Systems & Applications

Sonderfall: Terminal-Knoten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Blätter des ASTs, hier ist Text des Tokens relevant
- Bezeichner, Zahlen, Operatoren

Sonderfall: Terminal-Knoten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Blätter des ASTs, hier ist Text des Tokens relevant
- Bezeichner, Zahlen, Operatoren

Abstrakte Superklasse aller Terminal-Knoten

```
public abstract class Terminal extends AST {
  public String spelling;
  ...
}
```

Sonderfall: Terminal-Knoten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Blätter des ASTs, hier ist Text des Tokens relevant
- Bezeichner, Zahlen, Operatoren

Abstrakte Superklasse aller Terminal-Knoten

```
public abstract class Terminal extends AST {
  public String spelling;
  ...
}
```

Konkrete Unterklasse für Bezeichner

```
public class Identifier extends Terminal {
  public Identifier(String spelling) {
    this.spelling = spelling;
  }
}
```


Embedded Systems & Applications

Aufbau des ASTs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Während des rekursiven Abstiegs
- Idee: **`parseN`**-Methode liefert AST für **N**-Phrase
- AST für **N**-Phrase wird durch Zusammensetzen der ASTs der Subphrasen erstellt

Aufbau des ASTs

TECHNISCHE UNIVERSITÄT DARMSTADT

- Während des rekursiven Abstiegs
- Idee: **`parseN`**-Methode liefert AST für **N**-Phrase
- AST für **N**-Phrase wird durch Zusammensetzen der ASTs der Subphrasen erstellt

Beispiel für Produktion **N** ::= **X**

```
protected ASTN parseN () {
  ASTN itsAST;
  Parse X, sammele Subphrasen-ASTs in itsAST
  return itsAST
}
```

Zusammensetzen von Subphrasen ASTs 1

TECHNISCHE UNIVERSITÄT DARMSTADT

EBNF

Command ::= **single-Command** (**;** **single-Command**)$*$

Embedded Systems & Applications

Zusammensetzen von Subphrasen ASTs 1

TECHNISCHE UNIVERSITÄT DARMSTADT

EBNF

Command ::= single-Command (; single-Command)∗

AST

Command ::= Command ; Command SequentialCmd

Embedded Systems & Applications

Zusammensetzen von Subphrasen ASTs 1

TECHNISCHE UNIVERSITÄT DARMSTADT

EBNF

Command ::= **single-Command** (**;** **single-Command**)∗

AST

Command ::= **Command** **;** **Command** SequentialCmd

```
Command parseCommand() {
  Command c1AST = parseSingleCommand();
  while (currentToken.kind == TokenKind.SEMICOLON)
      {
    acceptIt();
    Command c2AST = parseSingleCommand();
    c1AST = new SequentialCmd(c1AST, c2AST);
  }
  return c1AST;
}
```


Embedded Systems & Applications

Zusammensetzen von Subphrasen ASTs 1

TECHNISCHE UNIVERSITÄT DARMSTADT

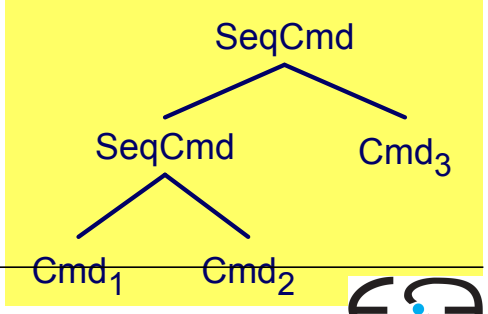
EBNF

Command ::= **single-Command** (**;** **single-Command**)∗

AST

Command ::= **Command** **;** **Command** SequentialCmd

```
Command parseCommand() {
  Command c1AST = parseSingleCommand();
  while (currentToken.kind == TokenKind.SEMICOLON)
      {
    acceptIt();
    Command c2AST = parseSingleCommand();
    c1AST = new SequentialCmd(c1AST, c2AST);
  }
  return c1AST;
}
```



Embedded Systems & Applications

Zusammensetzen von Subphrasen ASTs 2

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Declaration parseSingleDeclaration() {
  Declaration declAST;
  switch (currentToken.kind) {
  case CONST: // single-Declaration ::= const Identifier ~ Expression
    acceptIt();
    Identifier iAST = parseIdentifier();
    accept(TokenKind.IS);
    Expression eAST = parseExpression();
    declAST = new ConstDeclaration(iAST, eAST);
    break;

  case VAR:   // single-Declaration ::= var Identifier : Type-denoter
    acceptIt();
    Identifier iAST = parseIdentifier();
    accept(TokenKind.COLON);
    TypeDenoter tAST = parseTypeDenoter();
    declAST = new VarDeclaration(iAST, tAST);
    break;

  default: /* melde Syntaxfehler */
  }
  return declAST; }
```


Embedded Systems & Applications

Scanning - Woher kommen die Tokens?

TECHNISCHE UNIVERSITÄT DARMSTADT

Zwei relevante Methoden im Parser

```
public class Parser {
  Scanner scanner;
  Token currentToken;

  void accept(TokenKind tokenExpected) {
    if (currentToken.kind == tokenExpected)
      currentToken = scanner.scan();
    else
      /* melde Syntaxfehler */
  }
  void acceptIt() {
    currentToken = scanner.scan();
  }
}
```

Scanner

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch genannt lexikalische Analyse oder Lexer

Scanner

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch genannt lexikalische Analyse oder Lexer
- Ähnlich Parsing, aber auf einer Ebene feinerer Details
  - Parser: Arbeitet mit Tokens, die zu Phrasen gruppiert werden
  - Scanner: Arbeitet mit Zeichen, die zu Tokens gruppiert werden

Scanner

TECHNISCHE UNIVERSITÄT DARMSTADT

- Auch genannt lexikalische Analyse oder Lexer
- Ähnlich Parsing, aber auf einer Ebene feinerer Details
  - Parser: Arbeitet mit Tokens, die zu Phrasen gruppiert werden
  - Scanner: Arbeitet mit Zeichen, die zu Tokens gruppiert werden
- Aufgaben des Scanners
  - Bilde Tokens aus Zeichen
  - Entferne unerwünschte Leerzeichen, Zeilenvorschübe, etc. (white space)
  - Führe Buch über Zeilennummern und Eingabedateinamen

Scanner-Sicht auf Tokens

TECHNISCHE UNIVERSITÄT DARMSTADT

Tokens werden durch REs definiert, bestehend aus:

- Einzelzeichen
- Operatoren
  - Konkatenation: **A B**
  - Alternative: **A** | **B**
  - Optionalität: **A**?
  - Wiederholung: **A**$*$
  - Vordefinierte REs (sog. Macros)
- aber: keine rekursiven Definitionen

Darstellung von Scannern als endliche Automaten

TECHNISCHE UNIVERSITÄT DARMSTADT

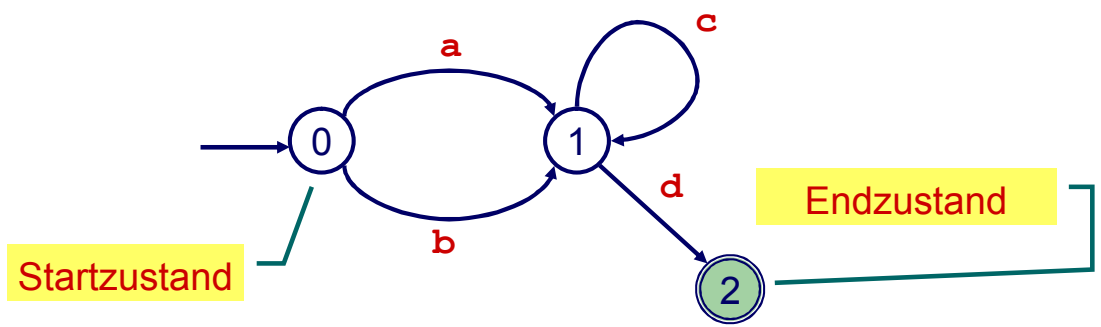
- Reguläre Ausdrücke können durch Übergangsdiagramme dargestellt werden
  - Endliche Automaten
  - Kanten/Transitionen beschriftet mit Eingabesymbolen
  - Zustände/Knoten
    - Genau ein Startzustand
    - Beliebig viele Endzustände (akzeptierende Zustände)

Darstellung von Scannern als endliche Automaten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Reguläre Ausdrücke können durch Übergangsdiagramme dargestellt werden
  - Endliche Automaten
  - Kanten/Transitionen beschriftet mit Eingabesymbolen
  - Zustände/Knoten
    - Genau ein Startzustand
    - Beliebig viele Endzustände (akzeptierende Zustände)

Beispiel: (**a** | **b**) **c**∗ **d**



Embedded Systems & Applications

Alternative: Rekursiver Abstieg

TECHNISCHE UNIVERSITÄT DARMSTADT

Systematische Konstruktion von Scannern

1. Formuliere lexikalische Grammatik in EBNF
   - Falls nötig: Transformiere für rekursiven Abstieg

Alternative: Rekursiver Abstieg

TECHNISCHE UNIVERSITÄT DARMSTADT

Systematische Konstruktion von Scannern

1. Formuliere lexikalische Grammatik in EBNF
   - Falls nötig: Transformiere für rekursiven Abstieg
2. Implementiere Scan-Methoden `scanN` für jede Produktion **N** ::= **X**, mit Rumpf passend zu **X**

Alternative: Rekursiver Abstieg

TECHNISCHE UNIVERSITÄT DARMSTADT

Systematische Konstruktion von Scannern

1. Formuliere lexikalische Grammatik in EBNF
   - Falls nötig: Transformiere für rekursiven Abstieg
2. Implementiere Scan-Methoden **`scanN`** für jede Produktion **N** ::= **X**, mit Rumpf passend zu **X**
3. Implementiere Scanner-Klasse, bestehend aus
   - **`protected`** Instanzvariable **`currentChar`**
   - **`protected`** Methoden **`take`** und **`takeIt`**
     - Analog zu **`accept`**/**`acceptIt`** im Parser
     - Lesen diesmal aber zeichenweise in **`currentChar`**
   - **`protected`** Scan-Methoden aus 2., erweitert um Erstellen von Token-Objekten
   - Eine **`public`** Methode **`scan`**, die den nächsten Token liefert
     - Überspringt dabei white space und Kommentare

Struktur des Java-Codes

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Scanner {
  char          currentChar;
  StringBuilder currentSpelling;

  public Token scan() {
    ... // Kommentare und Whitespace ueberlesen

    currentSpelling = new StringBuilder();
    TokenKind currentKind = scanToken();

    return new Token(currentKind, currentSpelling.toString());
  }

  TokenKind scanToken() {
    switch (currentChar) {
      ...
    }
  }

  void take(char expectedChar) { ... }
  void takeIt() { ... }
}
```

Struktur des Java-Codes

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Scanner {
  char          currentChar;
  StringBuilder currentSpelling;

  public Token scan() {
    ... // Kommentare und Whitespace ueberlesen

    currentSpelling = new StringBuilder();
    TokenKind currentKind = scanToken();

    return new Token(currentKind, currentSpelling.toString());
  }

  TokenKind scanToken() {
    switch (currentChar) {
      ...
    }
  }

  void take(char expectedChar) { ... }
  void takeIt() { ... }
}
```

Hänge **currentChar** an **currentSpelling** und lese nächstes Zeichen in **currentChar**

Embedded Systems & Applications

Scanner für Mini-Triangle

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Lexikalische Grammatik in EBNF verfassen

```
Token ::= Identifier | Integer-Literal | Operator |
          ; | : | := | ~ | ( | ) | eot
Identifier ::= Letter (Letter | Digit)*
Integer-Literal ::= Digit Digit*
Operator ::= + | - | * | / | < | > | =
Separator ::= Comment | space | eol
Comment ::= ! Graphic* eol
```

Scanner für Mini-Triangle

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Lexikalische Grammatik in EBNF verfassen

```
Token ::= Identifier | Integer-Literal | Operator |
          ; | : | := | ~ | ( | ) | eot
Identifier ::= Letter (Letter | Digit)*
Integer-Literal ::= Digit Digit*
Operator ::= + | - | * | / | < | > | =
Separator ::= Comment | space | eol
Comment ::= ! Graphic* eol
```

2. Umstellen für rekursiven Abstieg: Ersetzung und Linksausklammern

```
Token ::= Letter (Letter | Digit)*
        | Digit Digit*
        | + | - | * | / | < | > | =
        | ; | : (=|ε) | ~ | ( | ) | eot
Separator ::= ! Graphic* eol | space | eol
```

Hier eigentlich nicht nötig. Aber: Schneller!

Embedded Systems & Applications

Feinheiten am Rande

TECHNISCHE UNIVERSITÄT DARMSTADT

- EBNF kann nicht trennen zwischen

Feinheiten am Rande

TECHNISCHE UNIVERSITÄT DARMSTADT

- EBNF kann nicht trennen zwischen
  - Schlüsselworten

Feinheiten am Rande

TECHNISCHE UNIVERSITÄT DARMSTADT

- EBNF kann nicht trennen zwischen
  - Schlüsselworten
  - Bezeichnern

Feinheiten am Rande

TECHNISCHE UNIVERSITÄT DARMSTADT

- EBNF kann nicht trennen zwischen
  - Schlüsselworten
  - Bezeichnern
- Wird beides als **Identifier** beschrieben

Feinheiten am Rande

TECHNISCHE UNIVERSITÄT DARMSTADT

- EBNF kann nicht trennen zwischen
  - Schlüsselworten
  - Bezeichnern
- Wird beides als **Identifier** beschrieben

Feinheiten am Rande

TECHNISCHE UNIVERSITÄT DARMSTADT

- EBNF kann nicht trennen zwischen
  - Schlüsselworten
  - Bezeichnern
- Wird beides als **Identifier** beschrieben

➥während des Scannens reparieren.

Zeichenweises Einlesen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Scanner {
  private char          currentChar = ... // hole erstes Zeichen
  private StringBuilder currentSpelling;

  private void take(char expectedChar) {
    if (currentChar == expectedChar)
      takeIt();
    else
      /* melde lexikalischen Fehler */
  }

  private void takeIt() {
    currentSpelling.append(currentChar);
    currentChar = ... // hole naechstes Zeichen
  }
  ...
```

Scan-Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
...
public Token scan() {
  while (currentChar == '!'
      || currentChar == ' '
      || currentChar == '\n')
    scanSeparator();

  currentSpelling = new StringBuilder();
  TokenKind currentKind = scanToken();

  return new Token(kind, currentSpelling.toString(),
                   /* Position */);
}

private void scanSeparator() { ... }
private void scanToken() { ... }

...
```

Entwicklung sehr ähnlich zu Parse-Methode

Embedded Systems & Applications

Beispiel `scanToken`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
private TokenKind scanToken() {
  switch (currentChar) {
    case 'a': case 'b': ... case 'z':
    case 'A': case 'B': ... case 'Z':
      ... // Token ::= Letter (Letter | Digit)*
      return TokenKind.IDENTIFIER;
    case '0': ... case '9':
      ... // Token ::= Digit Digit*
      return TokenKind.INTLITERAL;
    case '+': case '-': ... case '=':
      takeIt();
      return TokenKind.OPERATOR;
    ...
  }
}
```


Embedded Systems & Applications

Genauer: Scannen von Identifier

TECHNISCHE UNIVERSITÄT DARMSTADT

```
case 'a': case 'b': ... case 'z':
case 'A': case 'B': ... case 'Z':
  takeIt();
  while (isLetter(currentChar) || isDigit(currentChar))
    takeIt();
  return TokenKind.IDENTIFIER;

case '0': ... case '9':
...
```

Hauptmethode `scan()`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
...
public Token scan() {
  while (currentChar == '!'
      || currentChar == ' '
      || currentChar == '\n')
    scanSeparator();

  currentSpelling = new StringBuilder();
  TokenKind currentKind = scanToken();

  return new Token(kind, currentSpelling.toString(),
                   /* Position */);
}
```

Wo nun Unterscheidung zwischen Bezeichnern und Schlüsselworten?

Ändern von Token-Art während der Konstruktion

TECHNISCHE UNIVERSITÄT DARMSTADT

```
enum TokenKind {
  ...
  static final Map<String, TokenKind> reservedWords;
  static {
    // Trage Schluesselwoerter in Hash-Tabelle ein
    reservedWords = Stream.of(
      ARRAY, BEGIN, CONST, DO, ELSE, END, FUNC, IF, IN,
      LET, OF, PROC, RECORD, THEN, TYPE, VAR, WHILE
    ).collect(toMap(t -> t.spelling, identity()));
  }
}

final class Token {
  ...
  Token(TokenKind kind, String spelling) {
  if (kind == TokenKind.IDENTIFIER)
    if (TokenKind.reservedWords.contains(spelling))
      kind = TokenKind.reservedWords.get(spelling)
  ...
  }
}
```


Embedded Systems & Applications

Automatisierbarkeit

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sehr mechanischer Ablauf
- Gut automatisierbar
- Beispiele
  - JLex/JFlex: Scanner basiert auf endlichem Automaten
  - Eingebaute Scanner in Parser-Generatoren ANTLR/JavaCC