Compilerbau

Martin Plümicke

WS 2021/22

Agenda

Überblick Vorlesung

- Literatur

Compiler Überblick

Compiler (I)

- Scanner
- Parser

Literatur


Bauer and Höllerer.
Übersetzung objektorientierter Programmiersprachen.
Springer-Verlag, 1998, (in german).


Alfred V. Aho, Ravi Lam, Monica S.and Sethi, and Jeffrey D. Ullman.
Compiler: Prinzipien, Techniken und Werkzeuge.
Pearson Studium Informatik. Pearson Education Deutschland, 2. edition, 2008.
(in german).


Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman.
Compilers Principles, Techniques and Tools.
Addison Wesley, 1986.


Reinhard Wilhelm and Dieter Maurer.
Übersetzerbau.
Springer-Verlag, 2. edition, 1992.
(in german).

Literatur II

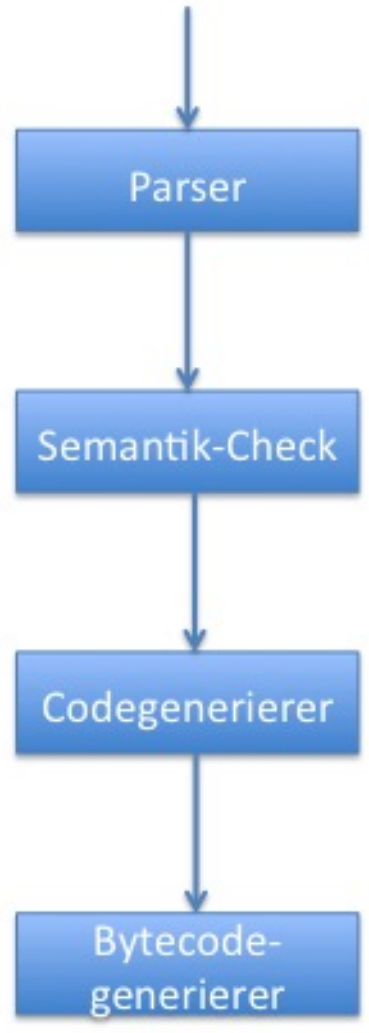

James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley.
The Java® Language Specification.
The Java series. Addison-Wesley, Java SE 8 edition, 2014.


Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley.
The Java® Virtual Machine Specification.
The Java series. Addison-Wesley, Java SE 8 edition, 2014.

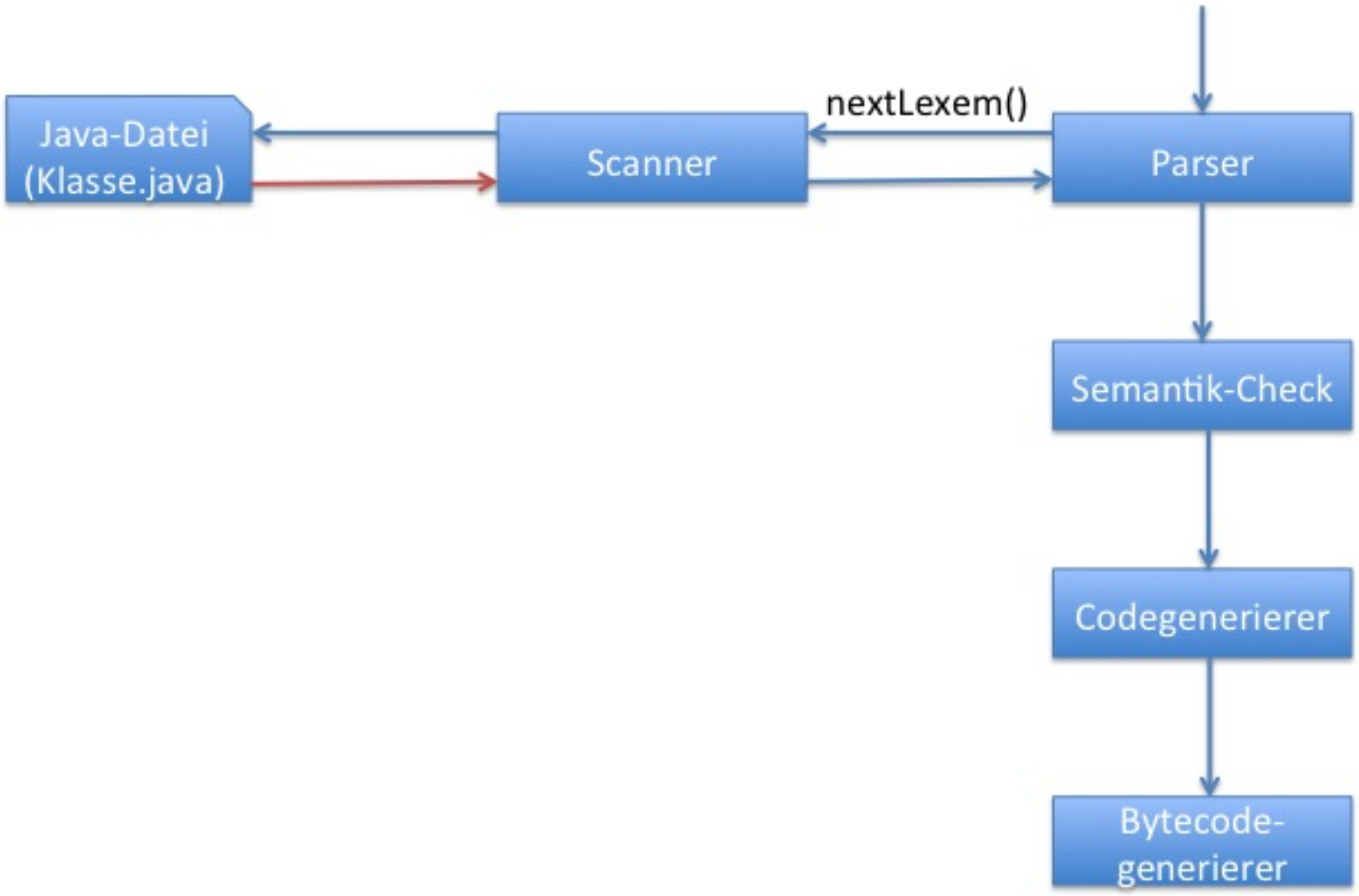

Bryan O'Sullivan, Donald Bruce Stewart, and John Goerzen.
Real World Haskell.
O'Reilly, 2009.


Peter Thiemann.
Grundlagen der funktionalen Programmierung.
Teubner, 1994.

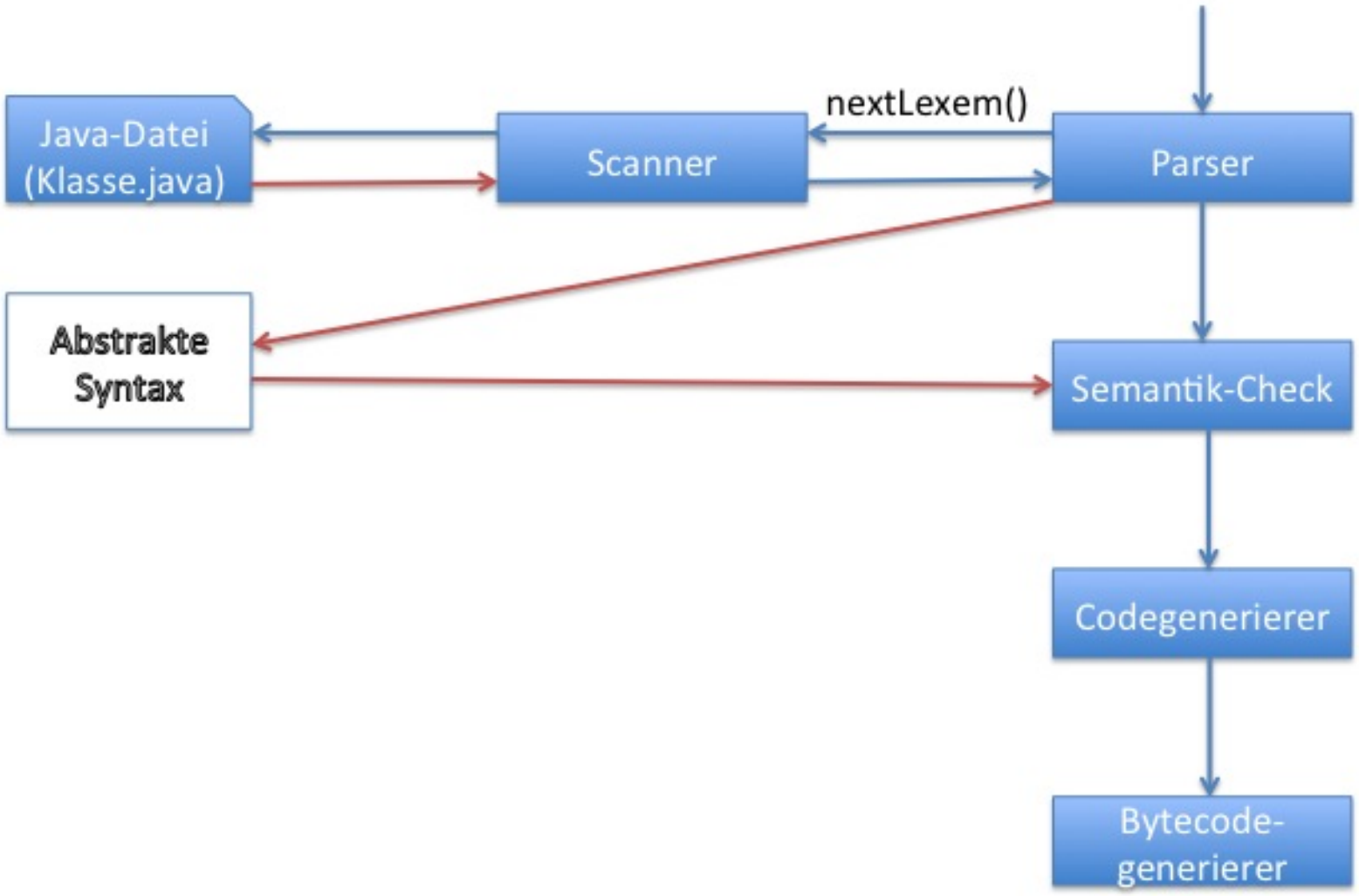
Compiler Überblick



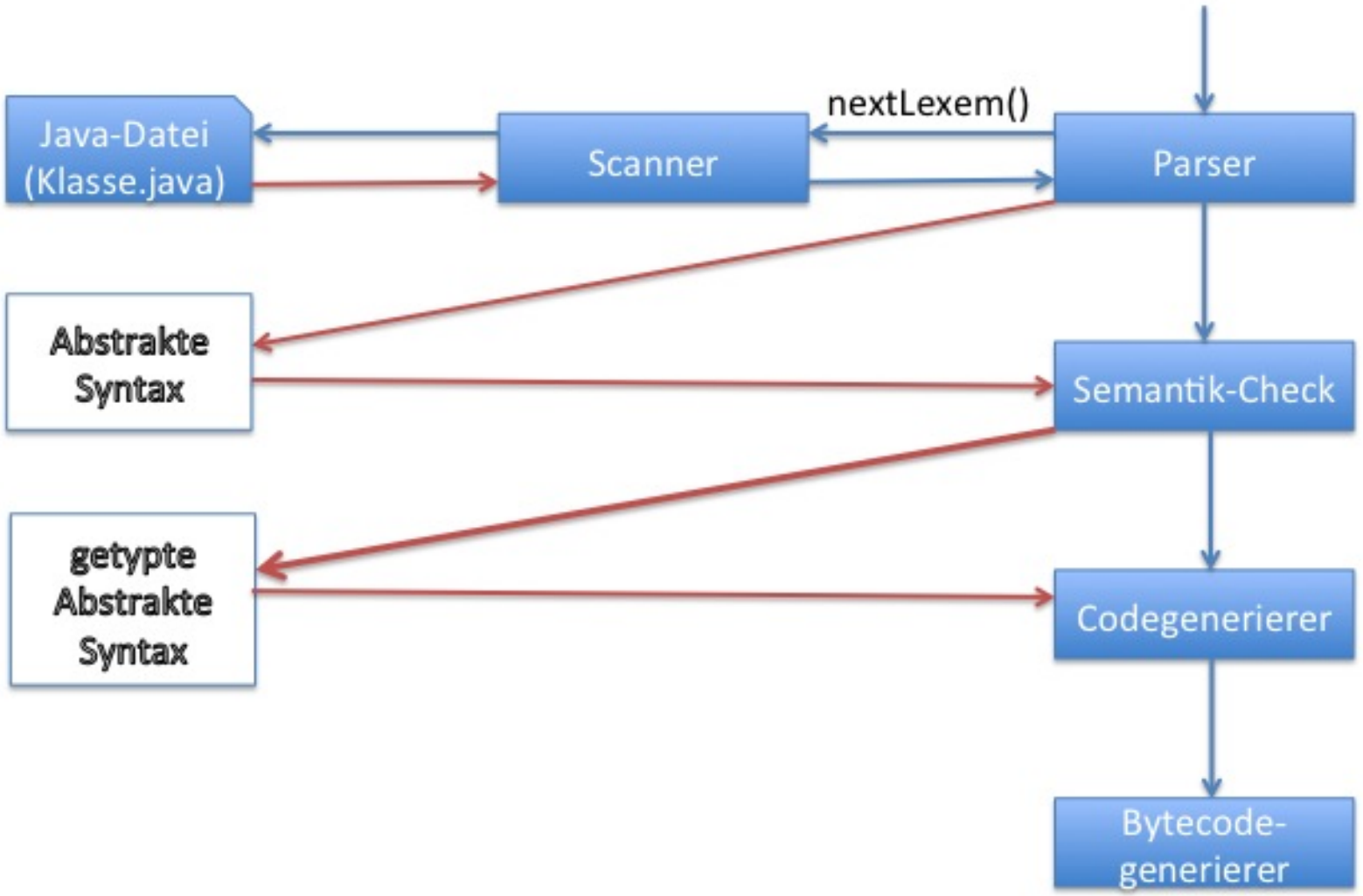
Compiler Überblick



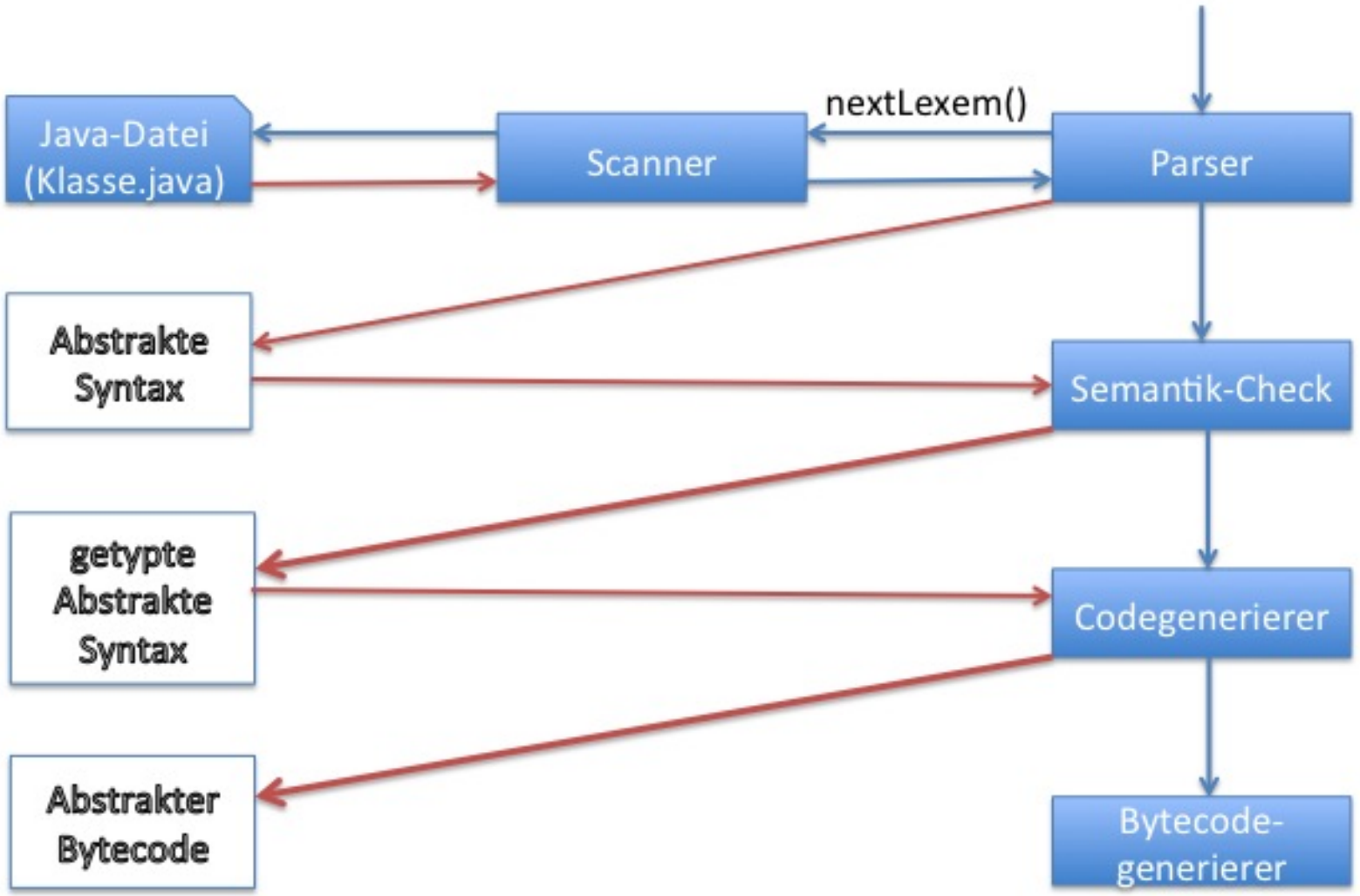
Compiler Überblick



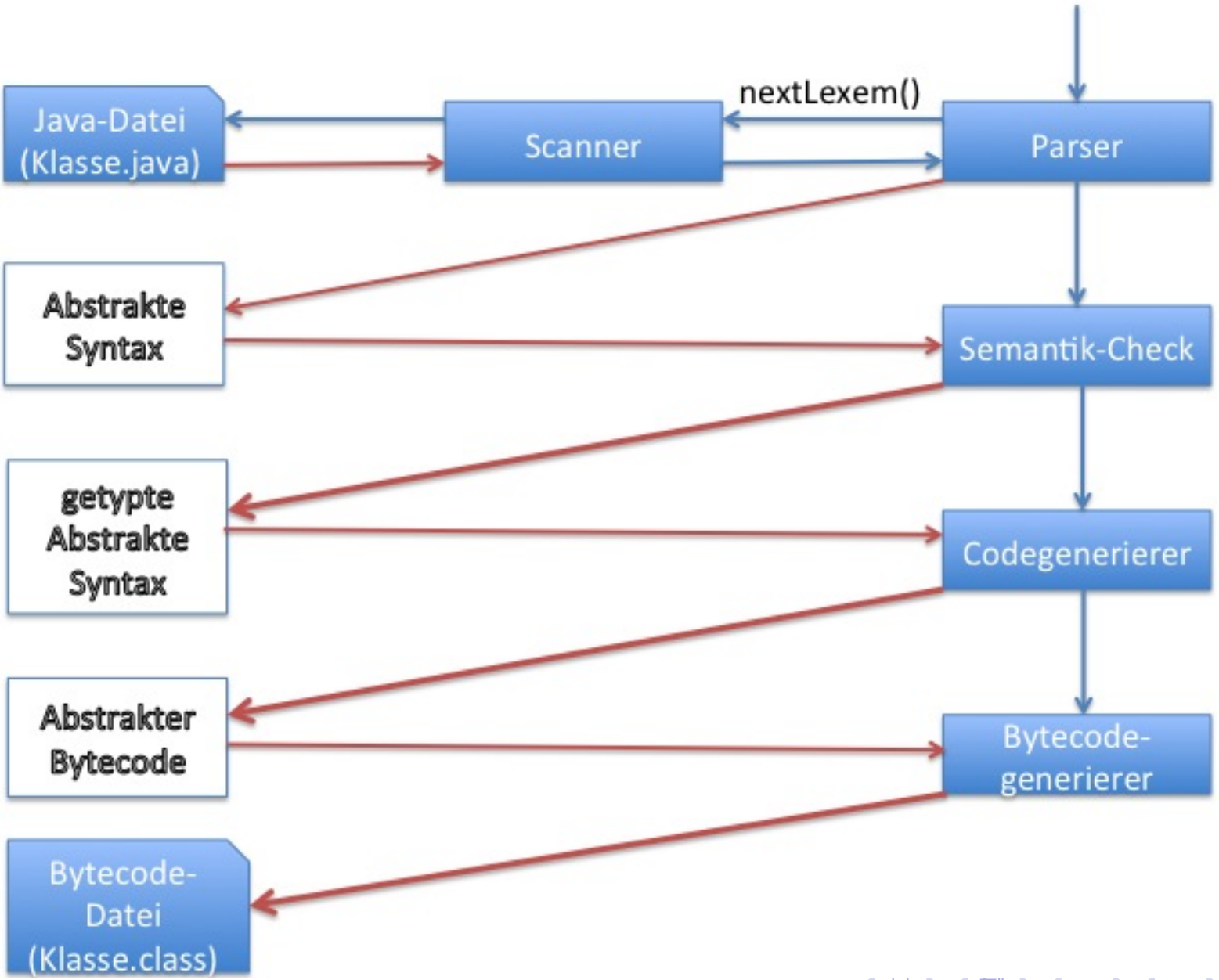
Compiler Überblick



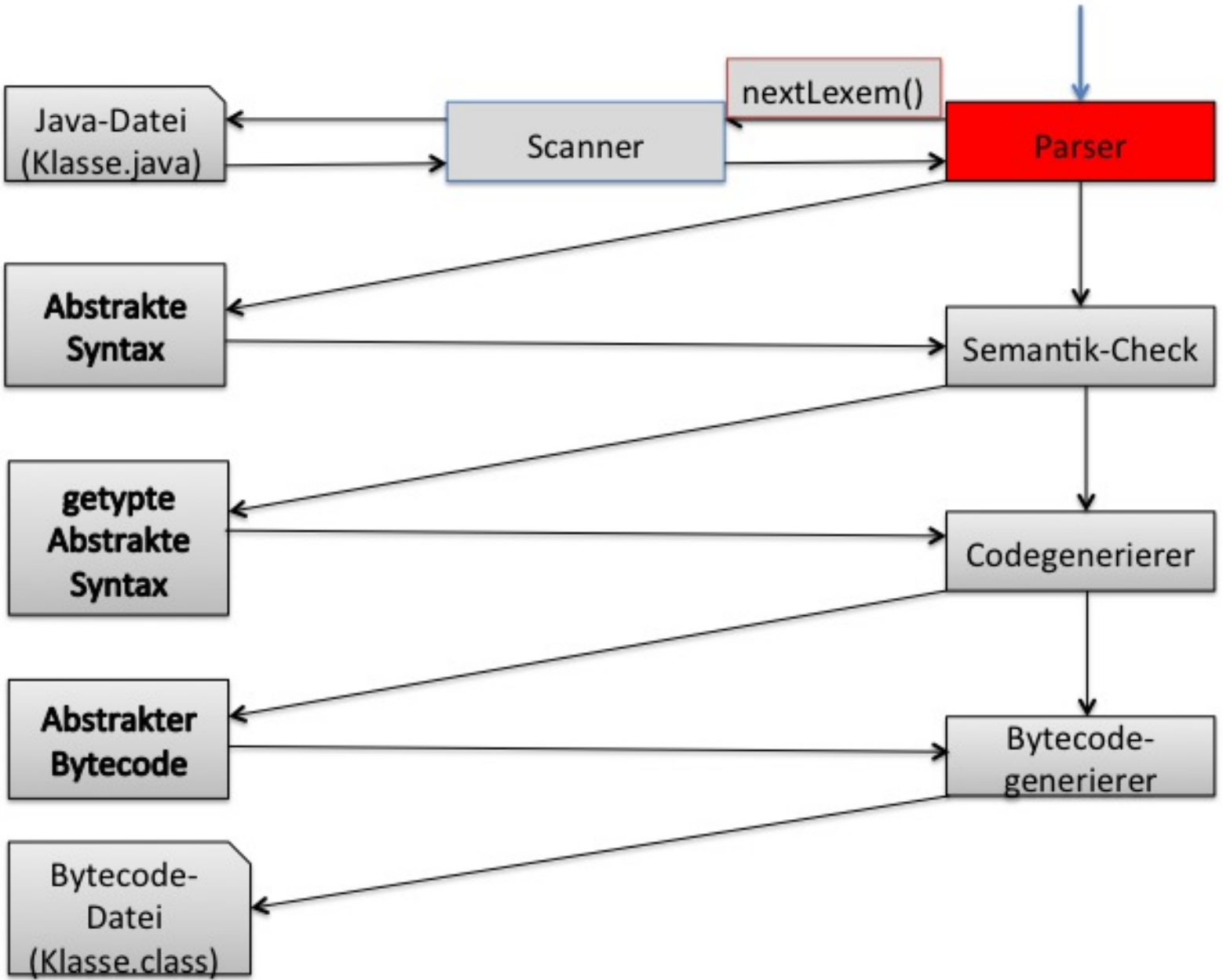
Compiler Überblick



Compiler Überblick



Parser


Java-Datei
(Klasse.java)
Scanner
nextLexem()
Parser
Abstrakte
Syntax
Semantik-Check
getypte
Abstrakte
Syntax
Codegenerierer
Abstrakter
Bytecode
Bytecode-
generierer
Bytecode-
Datei
(Klasse.class)

Programmiersprachen

Programmiersprachen werden als formale Sprachen über einem Alphabet von Tokens definiert.

Spezifikation eines Parser

Eingabe: Grammatik $G = (N, \Sigma, \Pi, S)$, $w \in \Sigma^*$

Ausgabe: $erg \in \{True, False\}$

Nachbedingung: $erg = (w \in \mathcal{L}(G))$

Mit anderen Worten: Es muss eine Ableitung $S \stackrel{*}{\rightarrow} w$ gefunden werden.

Beispiel

$G = (N, T, \Pi, S)$ mit

$N = \{S, A\}$
$T = \{a, b, c\}$
$\Pi = \{S \to cAb$
$\quad A \to ab \mid a\}$

Beispiel

$G = (N, T, \Pi, S)$ mit

$N = \{S, A\}$
$T = \{a, b, c\}$
$\Pi = \{S \to cAb$
$\quad A \to ab \mid a\}$

$w = cab$

Beispiel

$G = (N, T, \Pi, S)$ mit

$N = \{S, A\}$
$T = \{a, b, c\}$
$\Pi = \{S \to cAb$
$\quad A \to ab \mid a\}$

$w = cab$

$\Rightarrow Ableitung : S \to cAb \to cab$

Beispiel

$G = (N, T, \Pi, S)$ mit

$N = \{S, A\}$
$T = \{a, b, c\}$
$\Pi = \{S \to cAb$
$\quad A \to ab \mid a\}$

$w = cab$

$\Rightarrow Ableitung : S \to cAb \to cab$

Ergebnis: $erg = True$

Ansatz für einen Parser

Programmiersprachen werden als kontextfreie Grammatiken mit kontextsensitiven Nebenbedingungen beschrieben

Ansatz für einen Parser

Programmiersprachen werden als kontextfreie Grammatiken mit kontextsensitiven Nebenbedingungen beschrieben

- Cocke-Younger-Kasami–Algorithmus
  Nachteil:
  - Voraussetzung: Grammatik Chomsky-Normalform
  - Aufwand $O(n^3)$

Ansatz für einen Parser

Programmiersprachen werden als kontextfreie Grammatiken mit kontextsensitiven Nebenbedingungen beschrieben

- Cocke-Younger-Kasami–Algorithmus
  Nachteil:
  - Voraussetzung: Grammatik Chomsky-Normalform
  - Aufwand $O(n^3)$
- (Nicht deterministische) Push–Down–Automaten
  Nachteil:
  - Polynomialer Aufwand

$\Rightarrow$ Betrachtung einer Teilmenge der Chomsky-2–Sprachen (LR–Sprachen) (sind äquivalent zu den Sprachen, die durch deterministische Push–Down–Automaten erkannt werden.

$\Rightarrow$ Effiziente Implementierung möglich.

Ableitungsbaum

Man kann die Ableitung eines Wortes als Baum betrachten.

Ableitungsbaum

Man kann die Ableitung eines Wortes als Baum betrachten.

Aufbau:

Top-down: Linksableitungen (man erhält eine Ableitung, bei der immer das am weitesten links stehende Nichtterminal abgeleitet wird)

Ableitungsbaum

Man kann die Ableitung eines Wortes als Baum betrachten.

Aufbau:

Top-down: Linksableitungen (man erhält eine Ableitung, bei der immer das am weitesten links stehende Nichtterminal abgeleitet wird)

Bottom–Up: Rechtsableitungen (man erhält (rückwärts) eine Ableitung bei der immer das am weitesten rechts stehende Nichtterminal abgeleitet wird)

Linksableitungen (top–down)

$G = (N, T, \Pi, S)$ mit $N = \{S, A, B, C\}$ $T = \{a, b, c\}$ und
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$

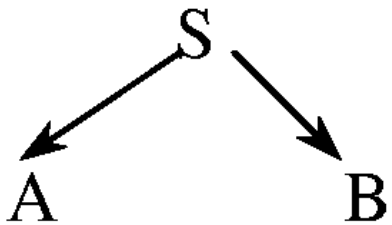
Eingabewort: $w = abc$

Linksableitungen (top–down)

$G = (N, T, \Pi, S)$ mit $N = \{S, A, B, C\}$ $T = \{a, b, c\}$ und
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$

Eingabewort: $w = abc$

$nexttoken() = a$

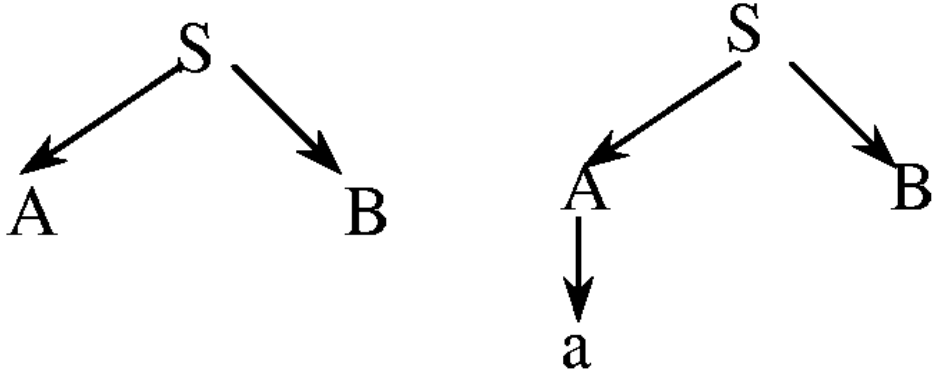


Linksableitungen (top–down)

$G = (N, T, \Pi, S)$ mit $N = \{S, A, B, C\}$ $T = \{a, b, c\}$ und
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$

Eingabewort: $w = abc$

$nexttoken() = a$

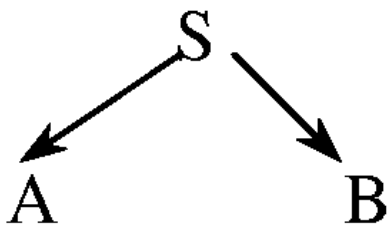


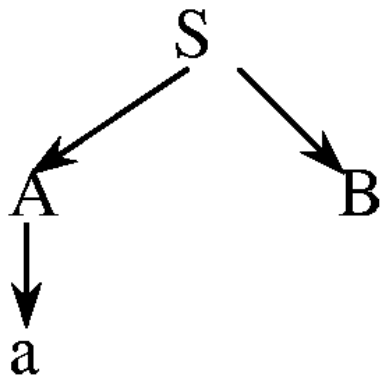
Linksableitungen (top–down)

$G = (N, T, \Pi, S)$ mit $N = \{S, A, B, C\}$ $T = \{a, b, c\}$ und
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$

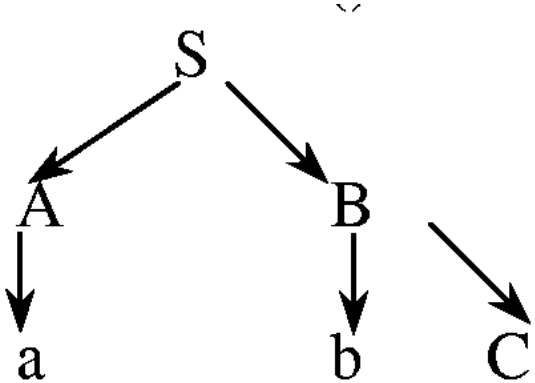
Eingabewort: $w = abc$

nexttoken() = *a*





nexttoken() = *b*

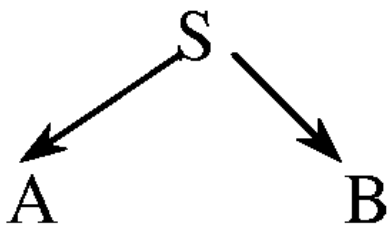


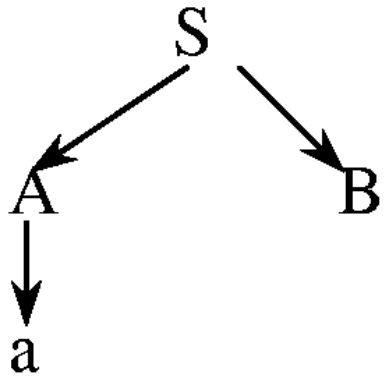
Linksableitungen (top–down)

$G = (N, T, \Pi, S)$ mit $N = \{S, A, B, C\}$ $T = \{a, b, c\}$ und
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$

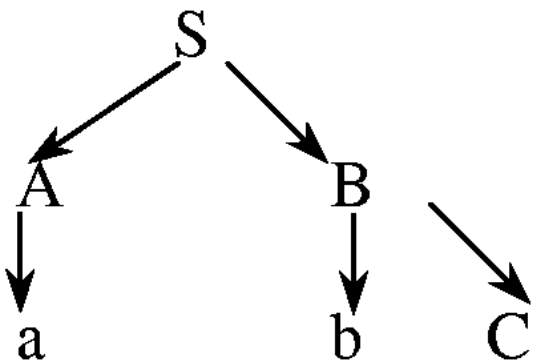
Eingabewort: $w = abc$

nexttoken() = *a*

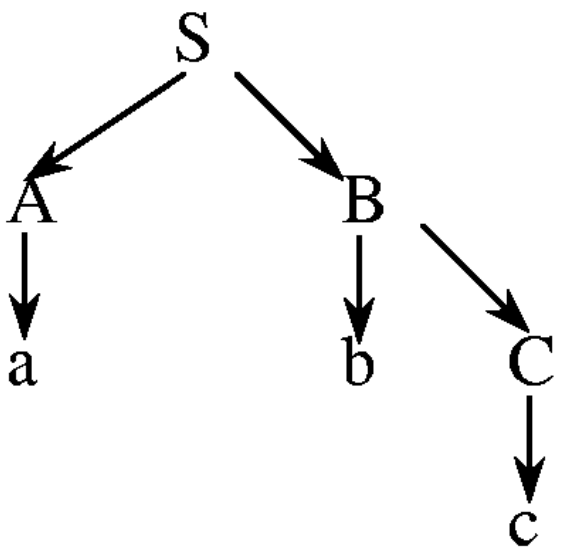




nexttoken() = *b*



nexttoken() = *c*



Recursive Decent–Syntaxanalyse

- Eingabe wird durch eine Menge rekursiver Funktionen abgebarbeitet.
- Jedem Nichtterminal der Grammatik entspricht eine Funktion.
- Die Folge der Funktionsaufrufe bestimmt implizit den Ableitungsbaum.

Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$

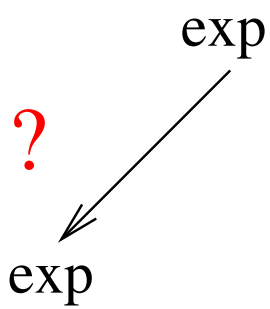
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



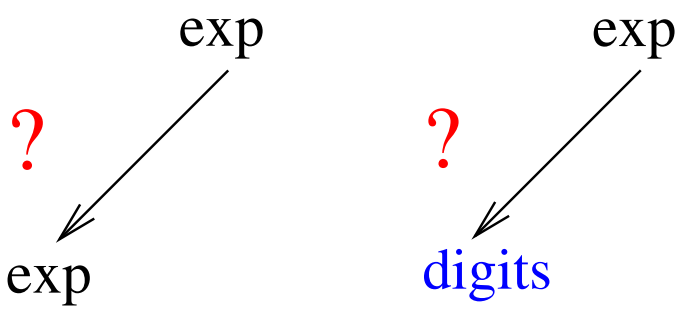
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



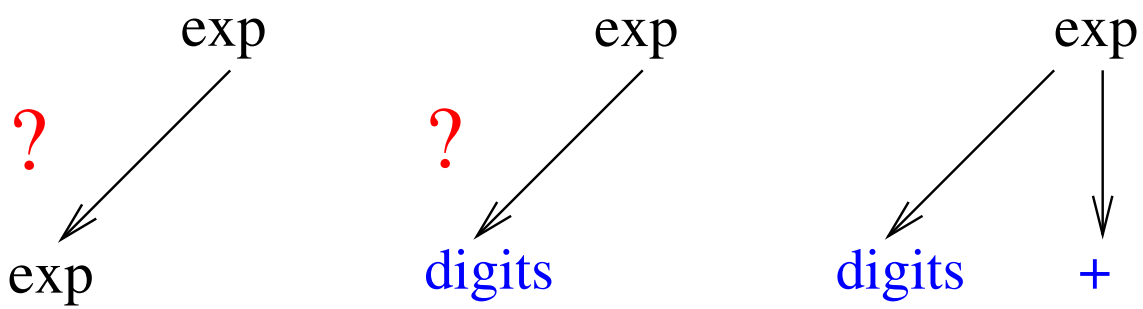
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



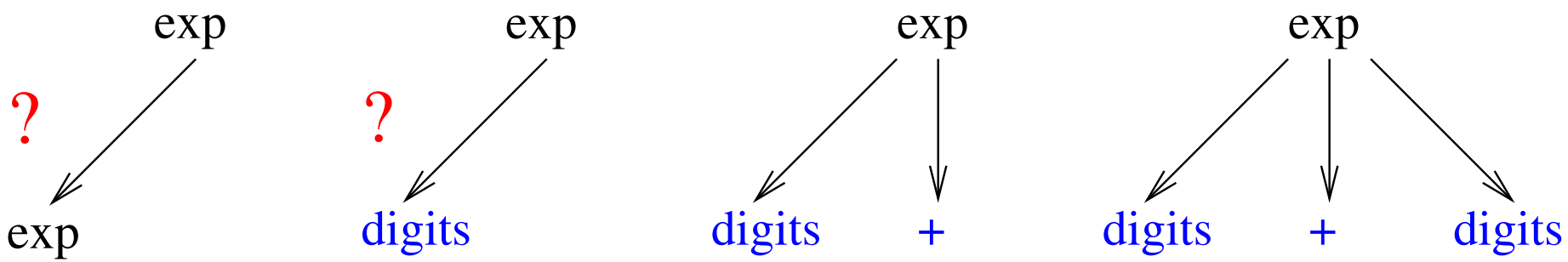
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow \mathit{let\ var\ =\ Exp\ in\ Exp} \mid \mathit{Exp\ +\ Exp} \mid \mathit{var} \mid \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



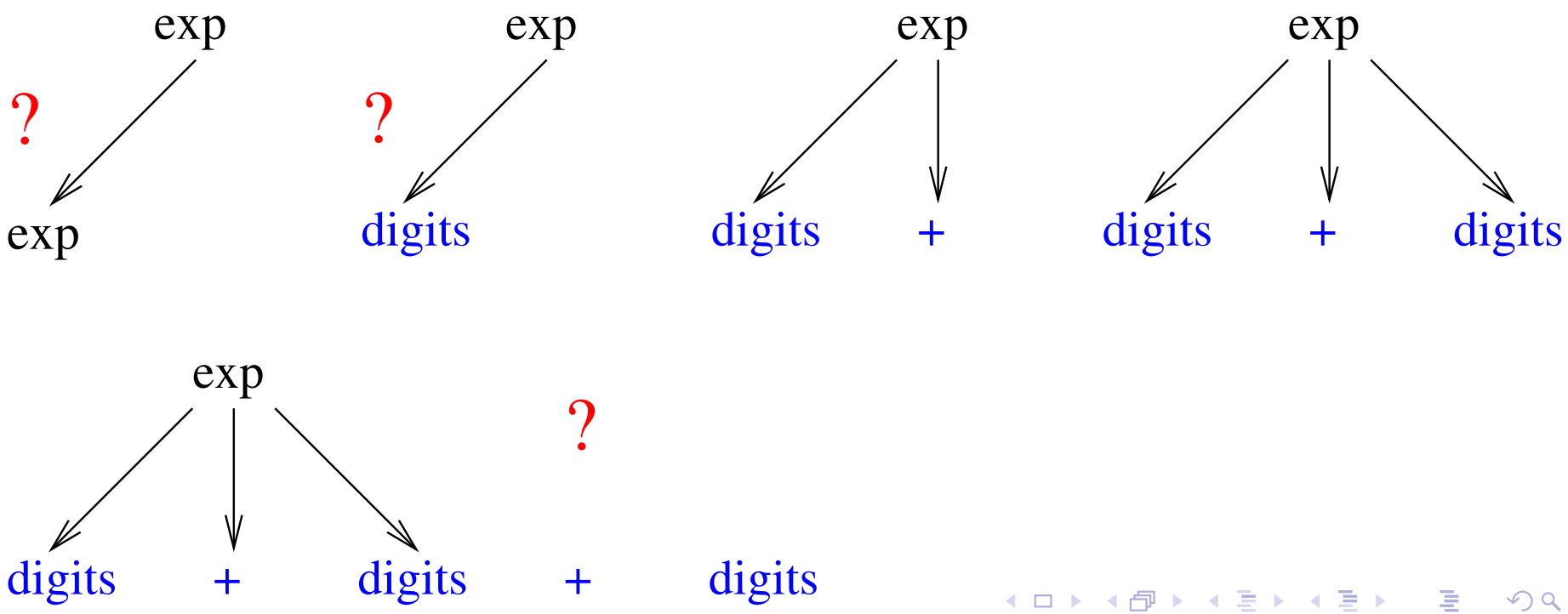
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$

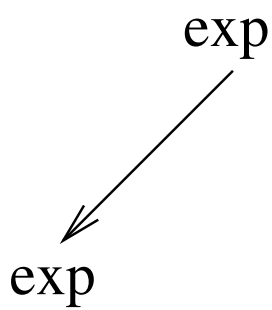
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow \mathit{let\ var\ =\ Exp\ in\ Exp} \mid \mathit{Exp\ +\ Exp} \mid \mathit{var} \mid \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



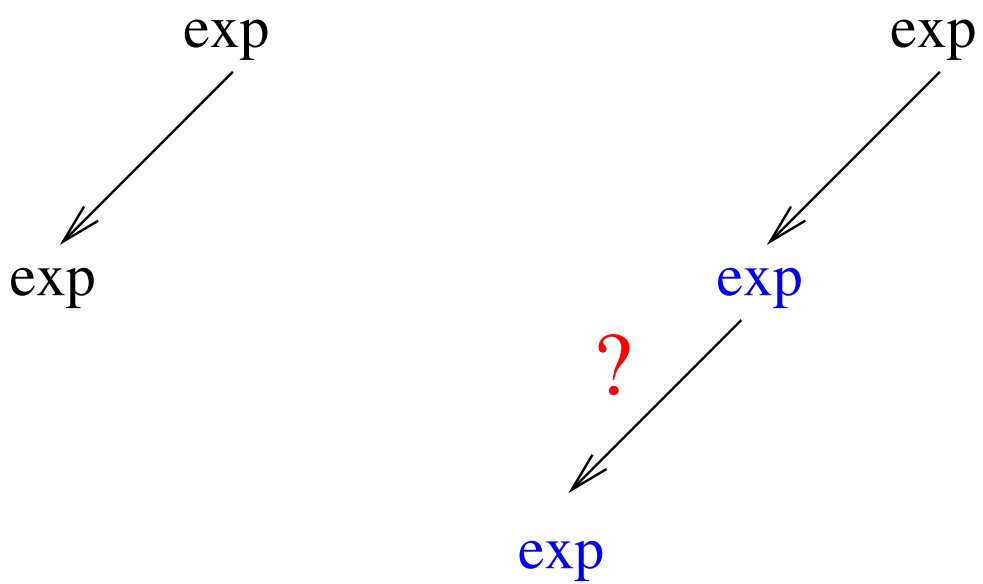
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



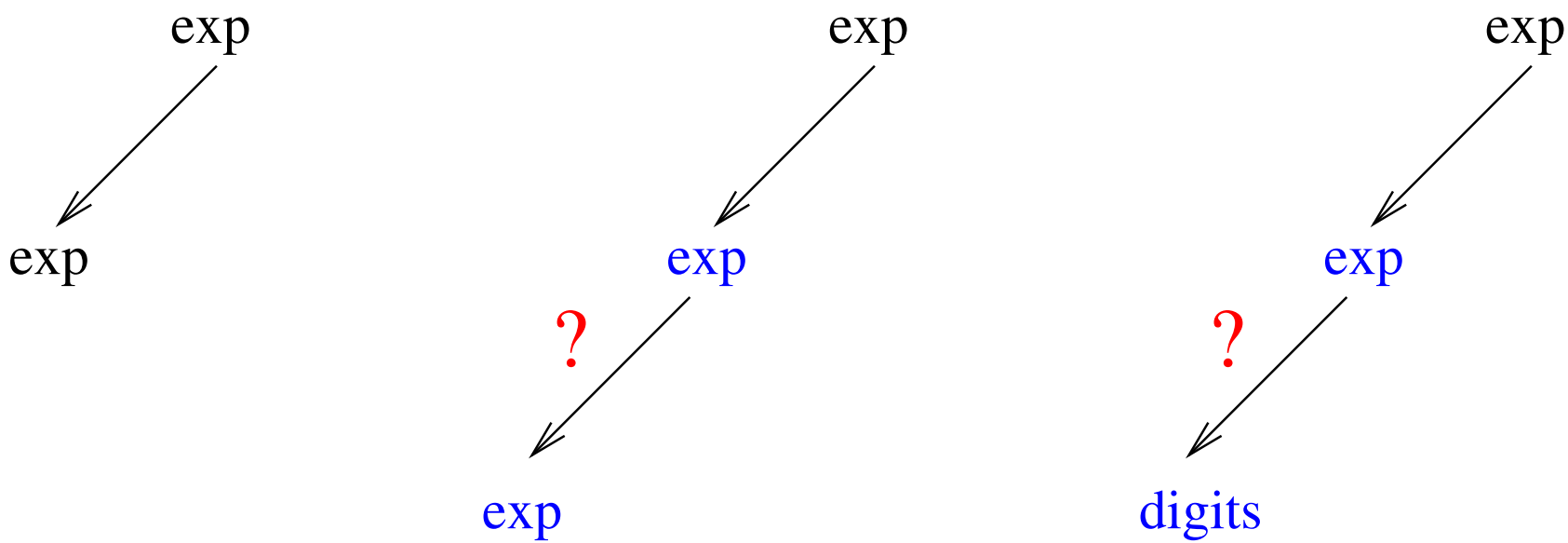
Linksrekursive Grammatik

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{\ \mathit{Exp} \rightarrow\ \mathit{let\ var\ =\ Exp\ in\ Exp}$$
$$\mid\ \ \mathit{Exp\ +\ Exp}$$
$$\mid\ \ \mathit{var}$$
$$\mid\ \ \mathit{digits}\}$$

Problem: Linksrekursion

Eingabewort $1 + 1 + 1$



Elimination der Linksrekursion

$G = (N, T, \Pi, S)$ mit

$$\Pi = \{ \begin{array}{ll} Exp \rightarrow & let\ var\ =\ Exp\ in\ Exp \\ \mid & Exp\ +\ Exp \\ \mid & var \\ \mid & digits \} \end{array}$$

Elimination der Linksrekursion

$G = (N, T, \Pi, S)$ mit

$$\begin{aligned}\Pi = \{\ Exp \rightarrow\ & let\ var\ =\ Exp\ in\ Exp \\ \mid\ & Exp\ +\ Exp \\ \mid\ & var \\ \mid\ & digits\}\end{aligned}$$

$$\begin{aligned}\Pi = \{\ Exp\ \ \rightarrow\ & TExp\ Exp' \\ Exp'\ \rightarrow\ & +\ TExp\ Exp' \\ \mid\ & \epsilon \\ TExp \rightarrow\ & let\ var\ =\ Exp\ in\ Exp \\ \mid\ & var \\ \mid\ & digits\ \}\end{aligned}$$

Rechtsableitungen (bottom–up)

$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$

Eingabewort: $w = abc$

Rechtsableitungen (bottom–up)

$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$
Eingabewort: $w = abc$

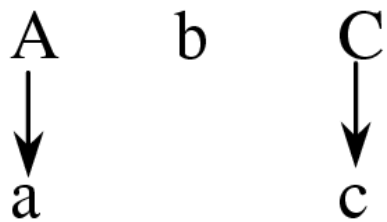
nexttoken$() = a$
wende an: $A \to a$

Rechtsableitungen (bottom–up)

$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$
Eingabewort: $w = abc$

nexttoken$() = a$
wende an: $A \to a$

nexttoken$() = b$
nexttoken$() = c$
wende an: $C \to c$

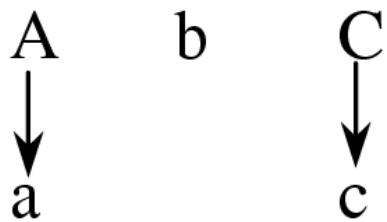


Rechtsableitungen (bottom–up)

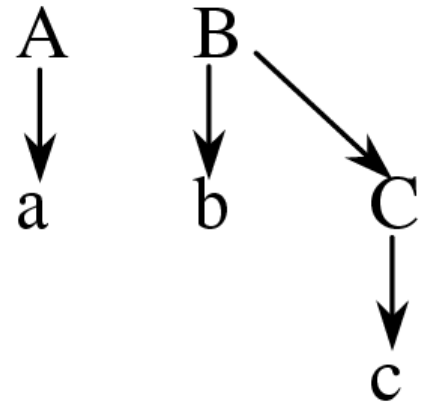
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$
Eingabewort: $w = abc$

nexttoken$() = a$
wende an: $A \to a$

nexttoken$() = b$
nexttoken$() = c$
wende an: $C \to c$



wende an: $B \to bC$

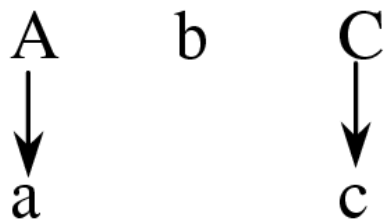


Rechtsableitungen (bottom–up)

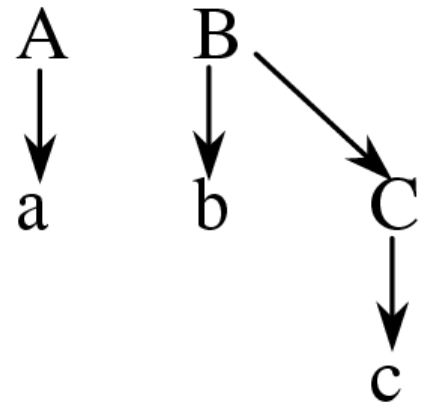
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$
Eingabewort: $w = abc$

nexttoken$() = a$
wende an: $A \to a$

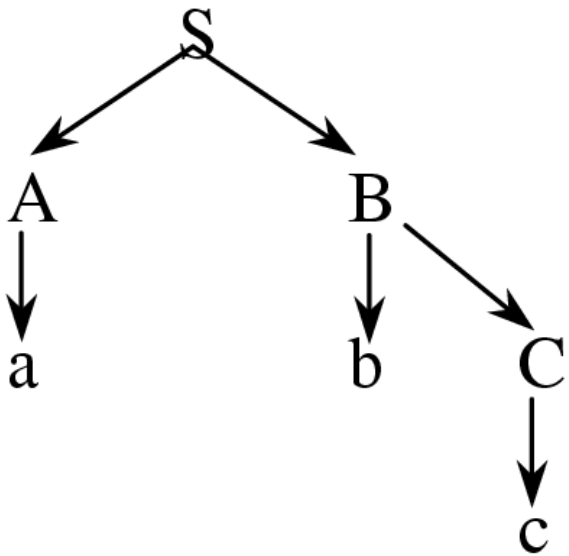
nexttoken$() = b$
nexttoken$() = c$
wende an: $C \to c$



wende an: $B \to bC$



wende an: $S \to AB$

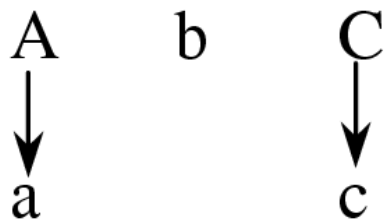


Rechtsableitungen (bottom–up)

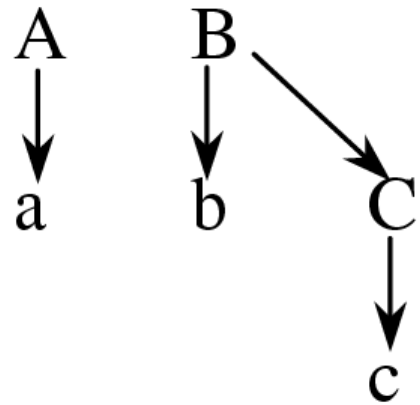
$\Pi = \{S \to AB, A \to a, B \to bC, C \to c\}$
Eingabewort: $w = abc$

nexttoken$() = a$
wende an: $A \to a$

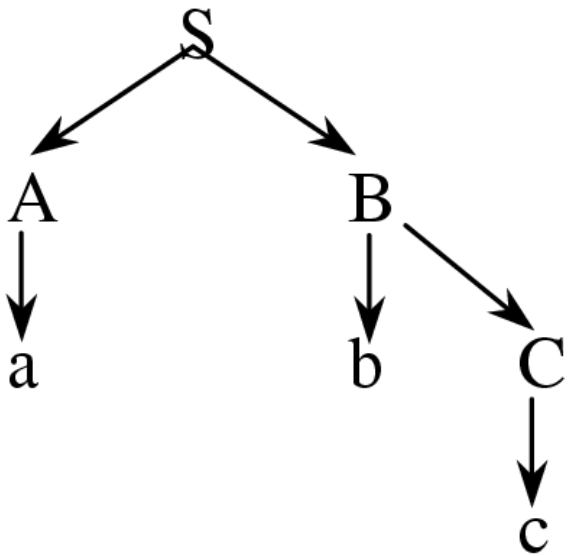
nexttoken$() = b$
nexttoken$() = c$
wende an: $C \to c$



wende an: $B \to bC$



wende an: $S \to AB$



Betrachtet man die Konstruktion rückwärts:

$$\underline{S} \to A\underline{B} \to Ab\underline{C} \to \underline{A}bc \to abc,$$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde \longleftarrow aAbc.de$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde \longleftarrow aAbc.de$

Shift–Reduce–Konflikt

$aAA.cde$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde \longleftarrow aAbc.de \longleftarrow$

Shift–Reduce–Konflikt

$aAA.cde$

$aA.de \longleftarrow aAd.e$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde \longleftarrow aAbc.de \longleftarrow$

Shift–Reduce–Konflikt

$aAA.cde$

$aA.de \longleftarrow aAd.e \longleftarrow aAB.e$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde \longleftarrow aAbc.de \longleftarrow$

Shift–Reduce–Konflikt

$aAA.cde$

$aA.de \longleftarrow aAd.e \longleftarrow aAB.e$

Reduce–Reduce–Konflikt

$aAA.e$

Weiteres Beispiel:

$G = (N, T, \Pi, S)$ mit
$N = \{S, A, B\}$,
$T = \{a, b, c, d, e\}$ und
$\Pi = \{S \to aABe, A \to Abc \mid b \mid d, B \to d\}$

Eingabestring: $w = abbcde$

Bottom–Up Syntaxanalyse:

$a.bbcde \longleftarrow ab.bcde \longleftarrow aA.bcde \longleftarrow aAb.cde \longleftarrow aAbc.de \longleftarrow$

Shift–Reduce–Konflikt

$aAA.cde$

$aA.de \longleftarrow aAd.e \longleftarrow aAB.e \longleftarrow aABe \longleftarrow S$

Reduce–Reduce–Konflikt

$aAA.e$

Konfliktlösungsansätze

LR(0): Es muss ohne Vorausschau möglich sein zu entscheiden, ob geshiftet oder reduziert wird.

SLR(1): An Hand der Bildung der Menge aller möglichen folgenden Terminalsymbole auf ein Nichtterminal, wird entschieden ob reduziert wird.

LR(1): Für jeden möglichen Ableitungsschritt in einer Produktion wird die Menge der darauffolgenden Terminalsymbolde zur Unterscheidung betrachtet.

LALR(1): Es werden alle Mengen von LR(1)–Elementen zusammengefasst, die den gleichen Ableitungsschritt vollziehen.

$$LR(0) \subset SLR(1) \subset LALR(1) \subset LR(1) (= \mathcal{L}(\mathcal{DPDA}))$$

Parsergenerator–Tool jaooy

Tokens

- Lexer/Scanner fasst die Strings zusammen, die an einer Stelle als Terminalsymbol in einer Grammatik stehen dürfen:

  Lexeme mit gleicher Bedeutung: z.B.: [<html>, <HTML>]
  Lexeme unterschiedlicher Bedeutung: z.B.: Text zwischen Tags

Ziel: Definition einer Datenstruktur, die diesem Rechnung trägt.

Klasse yyTokenclass

```
class yyTokenclass {
   public int tokennr;
   public Object value;

   yyTokenclass () {
      this.tokennr=-1;
   }
   yyTokenclass (Object o) {
      this.value = o;
   }
}
```

Alle Tokens erben von yyTokenclass.

JLex–Spezifikation mit Tokens

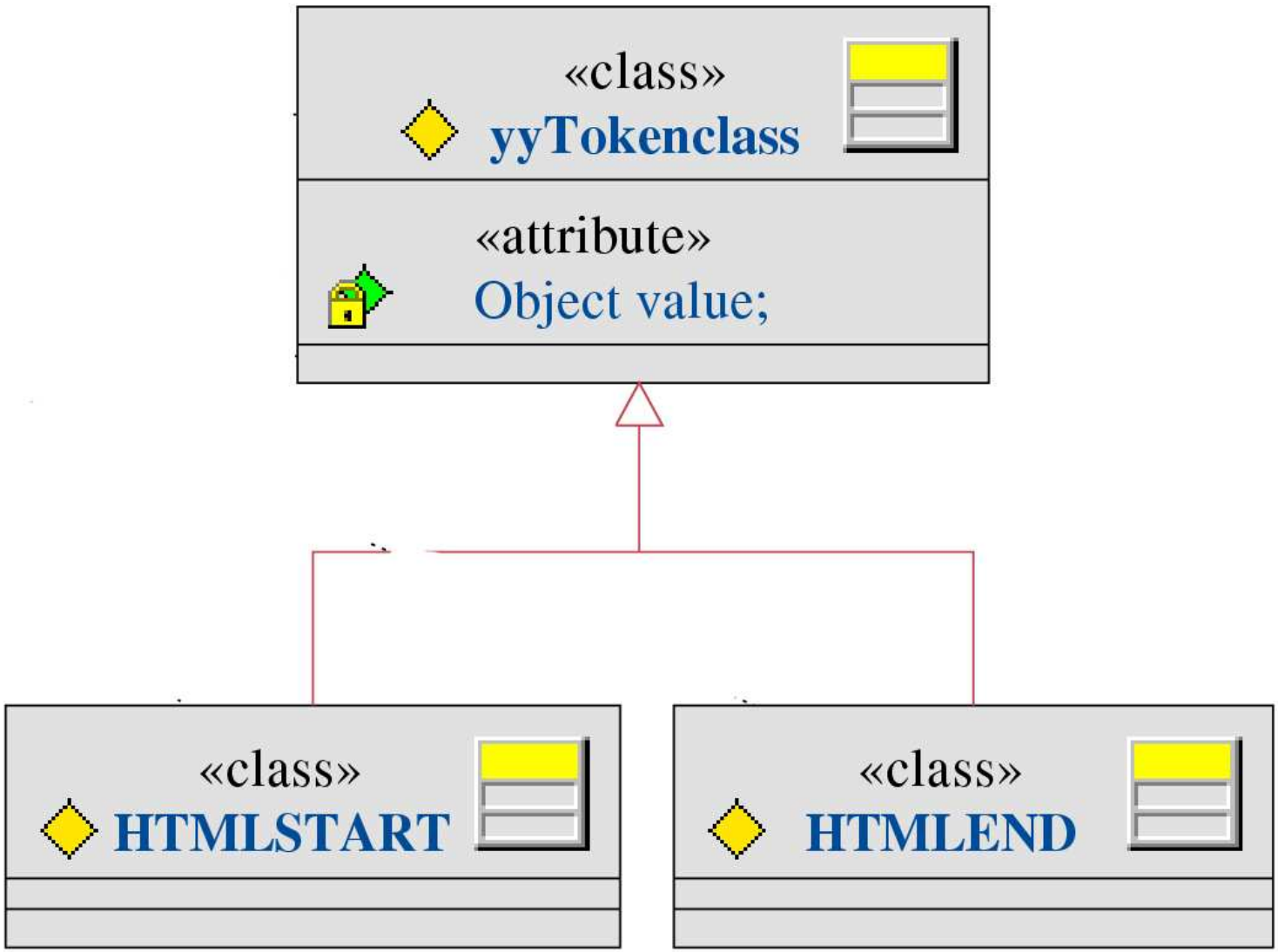
```
%%
%class browserlexer
%eofval{
  return new EOF();
%eofval}

ws = [ \t\r\n]+

%%
"<"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLSTART(yytext());}
"</"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLEND(yytext());}
[^\<]+          { System.out.println(yytext());
                          return new WORT(yytext());}
```

Wo kommen die Tokens (Java–Klassen) her?

Token–Klasse



jay–Tool (jaooy)

Eingabe (Grammatik)

- Namen der Tokenklassen (Terminale)
- Produktionen

Ausgabe

- Kellerautomat, der die Sprache der eingegebenen Grammatik erkennt als Java–Programm

jay–Spezifikation

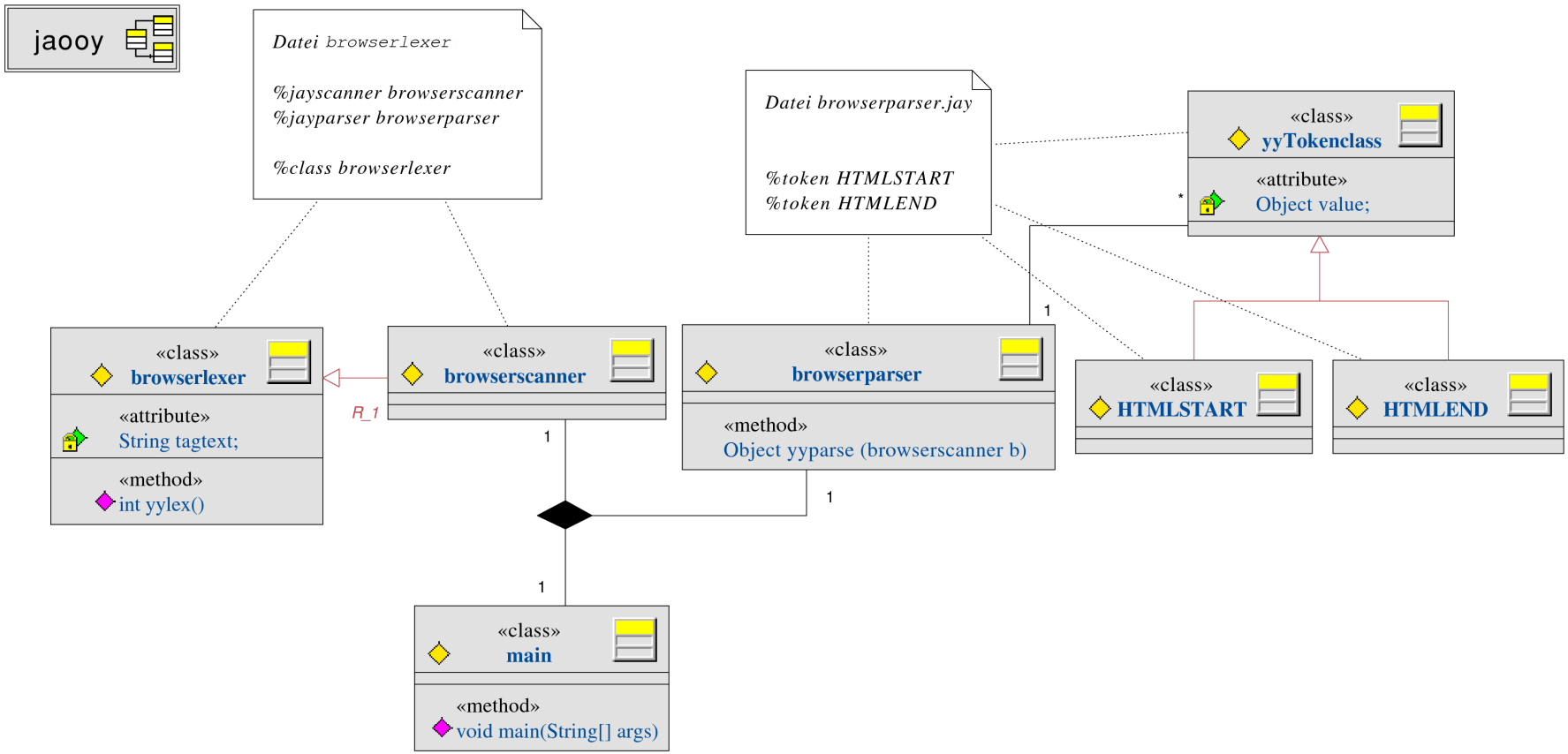
```
%{
class browserparser {
%}

%token HTMLSTART
%token HTMLEND
...

%%
S : HTMLSTART Head Body HTMLEND { }
Head : HEADSTART  Title   HEADEND { }
...

%%
}
```

Klassendiagramm JLex und jay



JLex–Spezifikation mit Klassendeklarationen

```
%%
%jayscanner browserscanner
%jayparser browserparser
%class browserlexer
%eofval{
  return new EOF();
%eofval}

ws = [ \t\r\n]+

%%
"<"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLSTART(yytext());}
"</"(h|H)(t|T)(m|M)(l|L)">" { return new HTMLEND(yytext());}
[^\<]+          { System.out.println(yytext());
                          return new WORT(yytext());}
```

Java–Klasse für den Aufruf

```
public class main {
    public static void main (String [] args) {
        browserscanner scanner =
            new browserscanner
                (new java.io.InputStreamReader (System.in));
        browserparser parser = new browserparser() ;
        try {
            parser.yyparse(scanner);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Makefile

```
Main.class: Main.java yyTokenclass.class browserscanner.class
	javac Main.java

yyTokenclass.class: browserparser.java
	javac browserparser.java

browserscanner.class: browserlexer.java
	javac browserlexer.java

browserlexer.java: browserlexer
	java -cp JLex2.jar JLex2.Main browserlexer

browserparser.java: browserparser.jay skeleton.jaooy
	jaooy -v browserparser.jay < skeleton.jaooy > browserparser.java
clean:
	rm -f *.class browserlexer.java browserparser.java
```

Aufgabe

Entwickeln Sie eine Mini–html–Parser als Grundlage für einen Browser.

- Definieren Sie eine Grammatik für Mini-html.
- Ergänzen Sie das JLex– und das Jay–File dementsprechend.