

20. Parser-Generatoren im Technikraum

Grammarware

1

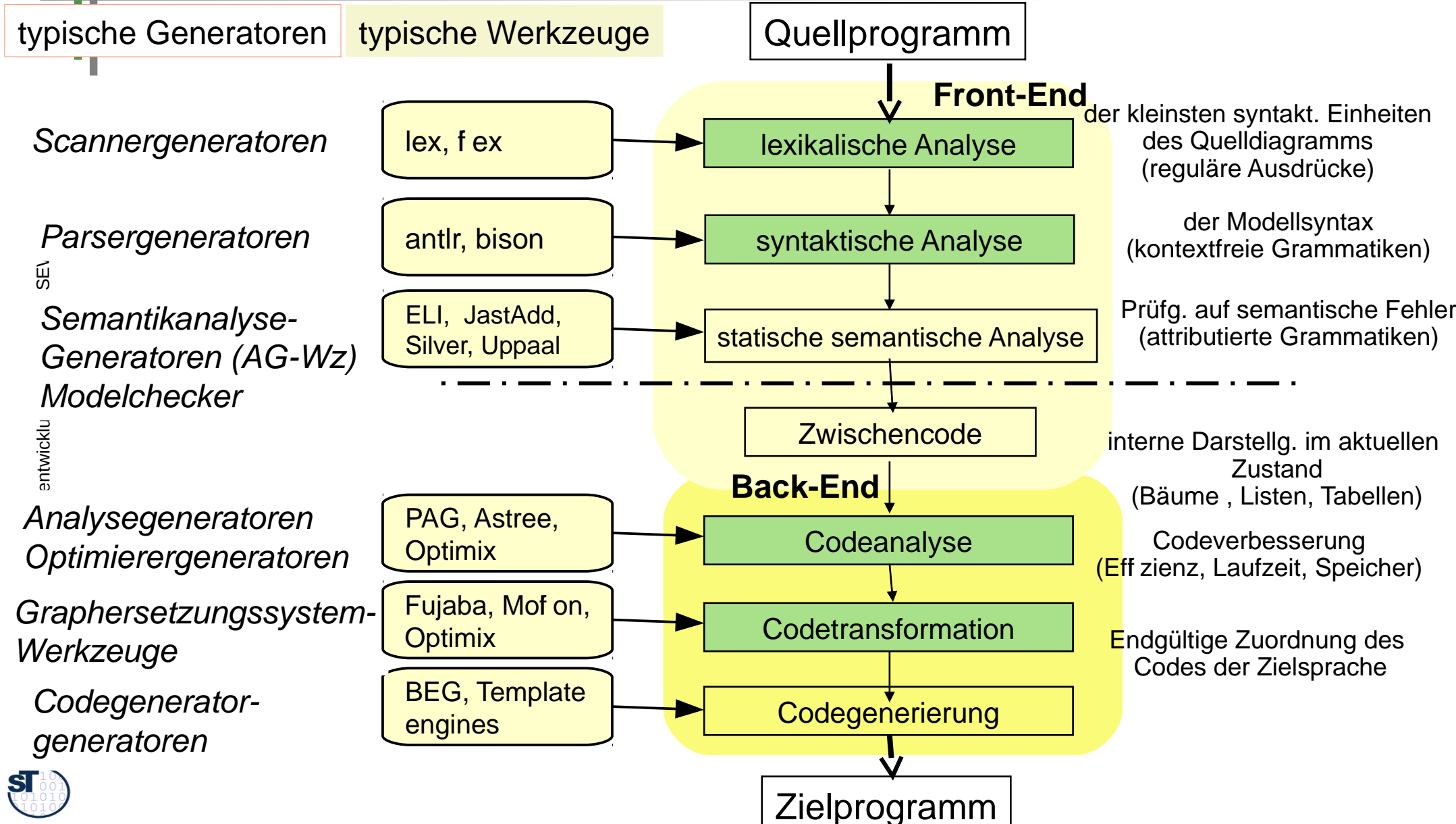
- 1) Grundlagen
- 2) Beispiel Taschenrechner

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und
Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
<http://st.inf.tu-dresden.de>
Version 12-1.1, 15.11.12



- ▶ Obligatorisch:
- ▶ <http://www.antlr.org>
- ▶ Zusätzlich:
 - Cocktail www.cocolab.de, die Compiler-Toolbox für die schnellsten Compiler der Welt (kommerziell, Demoverversionen erhältlich)

Phasen eines Compilers und die erzeugenden Werkzeuge



Phasen eines Importers in ein Repository und die erzeugenden Werkzeuge

typische Generatoren

typische Werkzeuge

Quellprogramm

Scannergeneratoren

lex, flex

lexikalische Analyse

der kleinsten syntakt. Einheiten
des Quelldiagramms
(reguläre Ausdrücke)

Parsergeneratoren

antlr
bison

syntaktische Analyse

der Modellsyntax
(kontextfreie Grammatiken)

Constraint checkers

DCL wie OCL, AG,
Logik

statische semantische Analyse

Prüfg. auf semantische Fehler
(attributierte Grammatiken)

softwareentwicklungsweg

Interne Repräsentation
(Zwischencode,
interne Modelle)

Query and metric
engines

DQL, CQL

Qualitätsanalyse

Messung der Qualität
des Modells/Codes

Refactoring engine

RSL

Refactoring

Semantik-erhaltende
Restrukturierung

Transformation engine

DTL

Transformation

Term- oder Graph-
Transformation

Front-End

Problem

- ▶ Parsen eines Programms, Modells oder Artefakts bedeutet, seine kontextfreie Syntax zu erkennen
- ▶ Parser sind die ersten Phasen eines Werkzeugs, denn es muss ein Artefakt importieren und damit ihn parsen
- ▶ Parsen erzeugt einen *Syntaxbaum*
- ▶ Parser wurden ursprünglich von Hand geschrieben (Compilerbau), heute generiert man sie aus *Grammatiken in EBNF*

Wie arbeite ich flexibel mit mehreren Programmiersprachen oder DSL?

- ▶ Bidirektionale Abbildung zwischen Technikraum “Grammarware” und einem anderen Technikraum, wie z.B. “Treeware” oder “Modelware”

In dem ich aus Grammatiken Parser (Zerteiler) generiere
und
zusätzlich Prettyprinter

Beispiel EMFText

- ▶ Nutzt Parser-Generator ANTLR zur Generierung von Parsern
 - Parser und Metamodell werden aufeinander abgebildet (mapping), um konkrete auf abstrakte Syntax abzubilden
- ▶ Nutzt schablonengesteuerte Codegenerierung zur Erzeugung von Text und Programmen (siehe später).

Beispiel: ANTLR www.antlr.org

- ▶ In den 90er Jahren gab es für C viele Parsergeneratoren
 - Cocktail's lalr, ell, lark www.cocolab.de
 - fnc2
 - flex und bison (gnu)
- ▶ Für Java ist ANTLR populär geworden
 - LL(k)
 - Generierter Parser mit Algorithmus “rekursiver Abstieg”
 - Etwas “gefärbte” Seite mit Geschichte
http://www.bearcave.com/software/antlr/antlr_expr.html



- parameter_declaration
- identifier_list
- initializer
- initializer_list
- type_name
- abstract_declarator
- direct_abstract_declarator
- typedef_name
- ▼ Statement
 - statement
 - labeled_statement
 - expression_statement
 - compound_statement
 - statement_list
 - selection_statement
 - iteration_statement
 - jump_statement
- Expression
- Lexer

```

compound_statement
: RCURLY declaration_list? statement_list? LCURLY
;

statement_list
: statement+
;

selection_statement
: 'if' LPAREN expression RPAREN statement ('else' statement)?
'switch' LPAREN expression RPAREN statement
;

iteration_statement
: 'while' LPAREN expression RPAREN statement
'do' statement 'while' LPAREN expression RPAREN statement
'for' LPAREN expression SEMI expression SEMI expression
;

jump_statement
: 'goto' identifier SEMI
'continue' SEMI
'break' SEMI
'return' expression SEMI
;

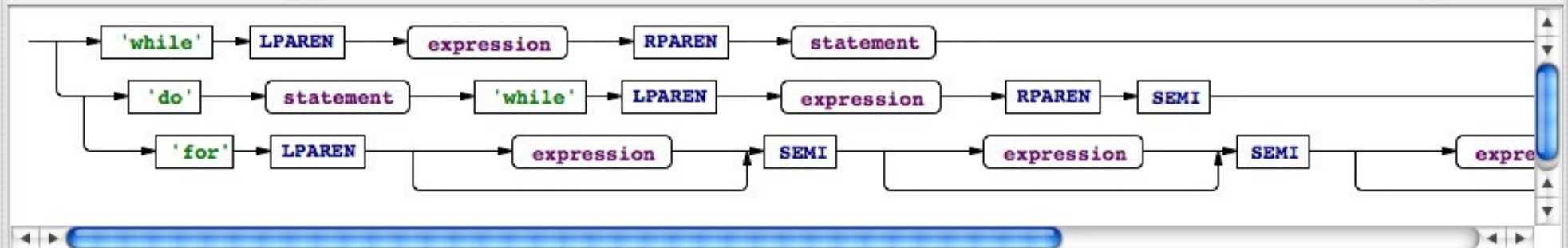
```

Enter rule name:

st

- struct_or_union_specifier
- storage_class_specifier
- struct_or_union
- struct_declaration_list
- struct_declaration
- struct_declarator_list
- struct_declarator
- statement
- statement_list
- string

Zoom Show NFA



Syntax Diagram Interpreter Debugger Console



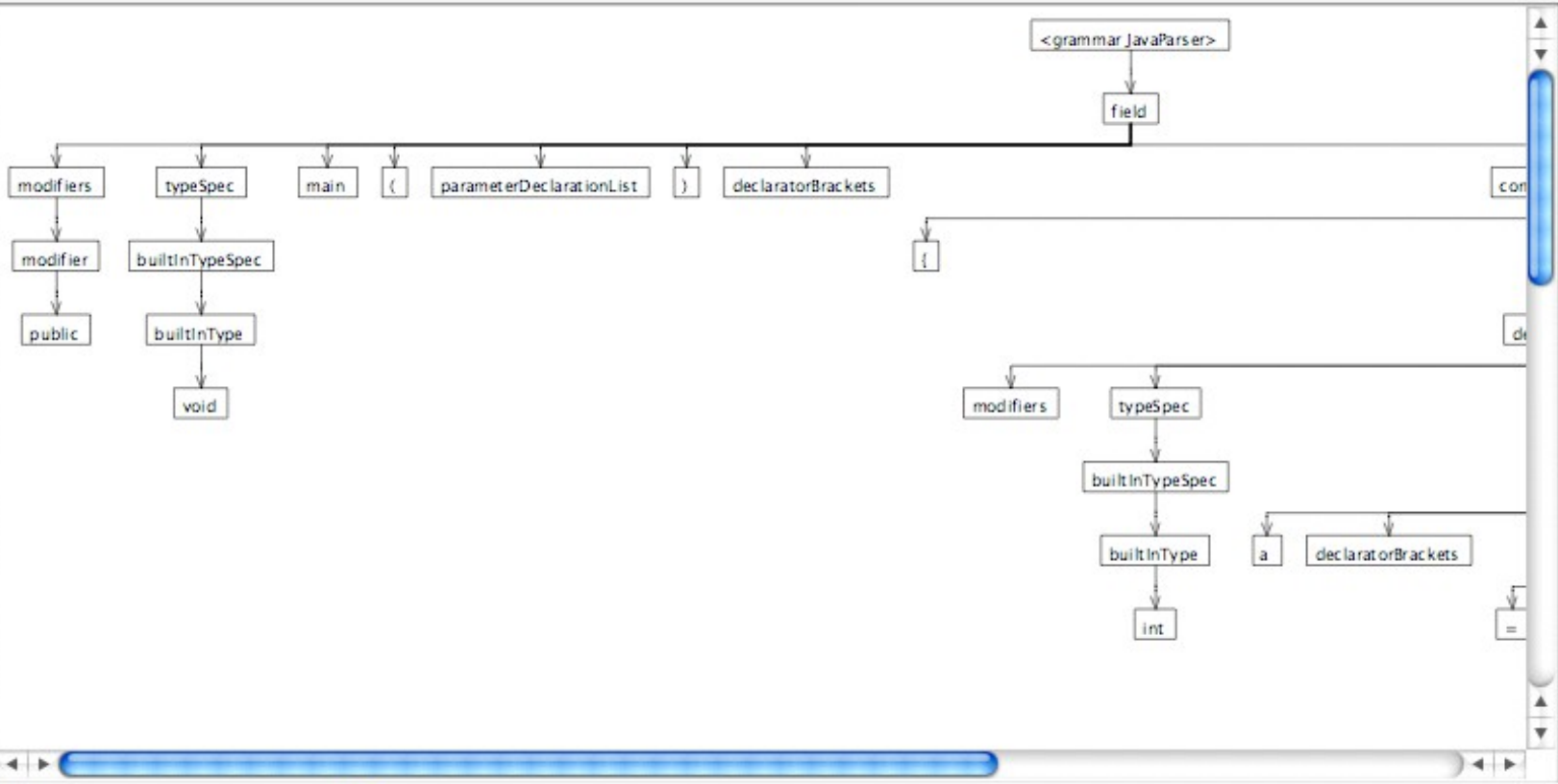


- P handler
- P expression
- P expressionList
- P assignmentExpression
- P conditionalExpression

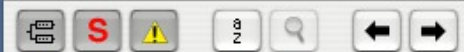
```
// the mother of all expressions  
expression  
: assignmentExpression  
;
```

field

```
public void main() {  
  int a = 2+3;  
}
```



Syntax Diagram Interpreter Debugger Console



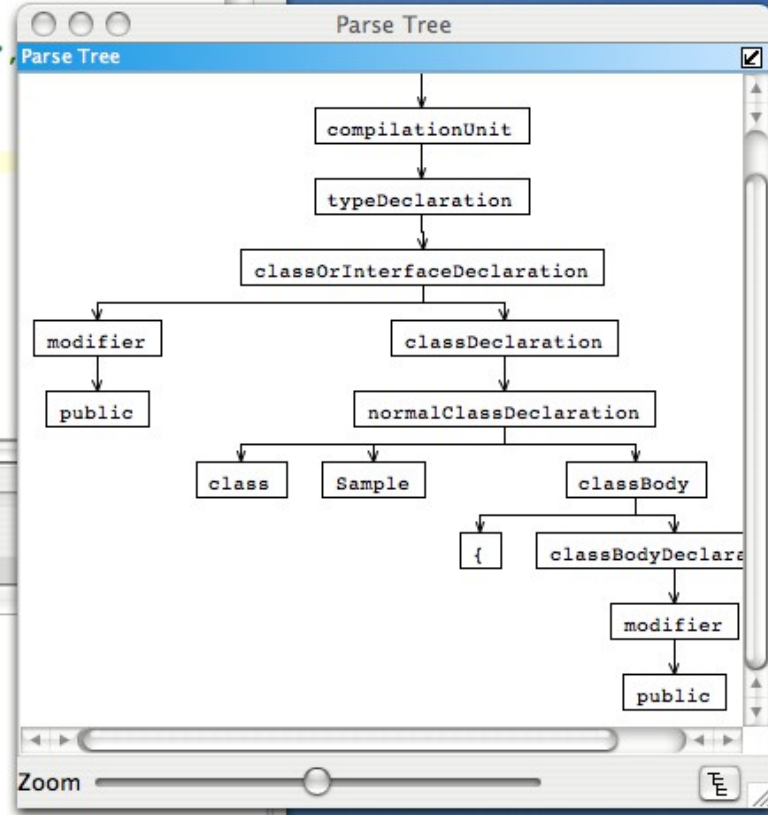
- interfaceBodyDeclaration
- interfaceMemberDecl
- interfaceMethodOrFieldDecl
- interfaceMethodOrFieldRest
- methodDeclaratorRest
- voidMethodDeclaratorRest
- interfaceMethodDeclaratorRest
- interfaceGenericMethodDecl
- voidInterfaceMethodDeclaratorRest
- constructorDeclaratorRest
- constantDeclarator
- variableDeclarators
- variableDeclarator
- variableDeclaratorRest
- constantDeclaratorsRest
- constantDeclaratorRest
- variableDeclaratorId
- variableInitializer
- arrayInitializer
- modifier

```
variableDeclaratorId
    : Identifier ('[' '])*
    ;

variableInitializer
    : arrayInitializer
    | expression
    ;

arrayInitializer
    : '{' (variableInitializer ',' variableInitializer)* ','
    ;

modifier
    : annotation
    | 'public'
    | 'protected'
    | 'private'
    | 'static'
    | 'abstract'
    | 'final'
    | 'native'
    | 'synchronized'
    | 'transient'
    | 'volatile'
    | 'strictfp'
    ;
```



Break on: All Location Consume LT Exception

```
Input
public class Sample {
    public void main() {
        System.out.println("Hello, world");
    }
}
```

#	Rule
0	compilationUnit
1	typeDeclaration
2	classOrInterfaceDeclaration
3	classDeclaration
4	normalClassDeclaration
5	classBody
6	classBodyDeclaration
7	modifier

Input Output Parse Tree AST Stack Events

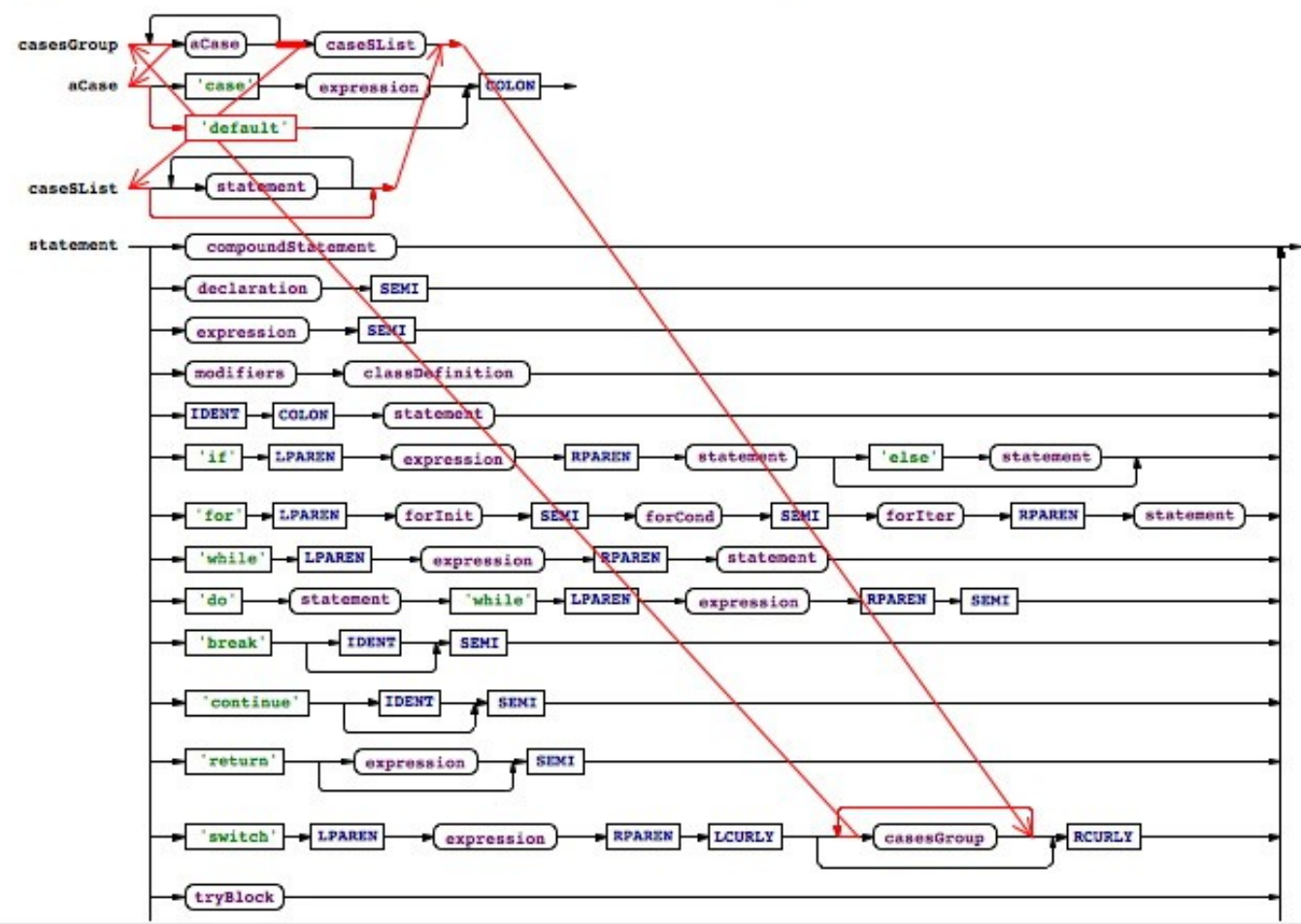
Syntax Diagram Interpreter Debugger Console





Zoom Show NFA

(1/2) Decision can match input such as "default" using multiple alternatives Alternatives: 1 2



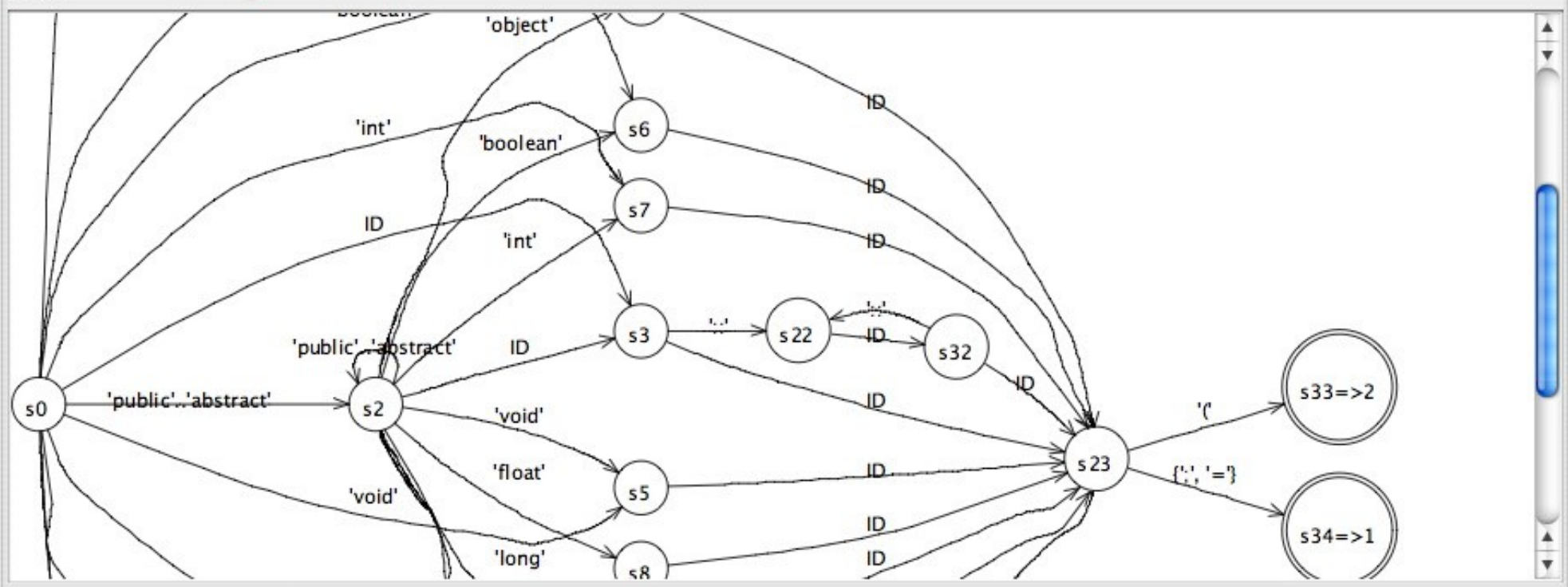
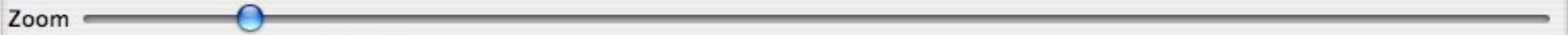
Syntax Diagram Interpreter Debugger Console





- P compilationUnit
- P packageDefinition
- P importDefinition
- P typeDefinition
- P classDefinition
- P interfaceDefinition
- P methodDefinition
- P formalArgs

```
classDefinition[MantraAST mod]
scope {
  String name;
}
: 'class' ID ('extends' sup=classname)? ('implements' i+=classname (',' i+=classname)*)?
  { $classDefinition::name = $ID.text; }
  {
    variableDefinition
    methodDefinition
  }*
```



Syntax Diagram Interpreter Debugger Console **Decision 10 of "classDefinition"**



13.2 Ein Taschenrechner



14

```
grammar Expr;
@header {
package test;
import java.util.HashMap;
}
@lexer::header {package test;}
@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}
prog:  stat+ ;

stat:  expr NEWLINE {System.out.println($expr.value);}
      | ID '=' expr NEWLINE
        {memory.put($ID.text, new Integer($expr.value));}
      | NEWLINE
      ;

expr returns [int value]
:  e=multExpr {$value = $e.value;}
  ( '+' e=multExpr {$value += $e.value;}
  | '-' e=multExpr {$value -= $e.value;}
  )*
  ;

multExpr returns [int value]
:  e=atom {$value = $e.value;} ('*' e=atom {$value *= $e.value;}) *
  ;

atom returns [int value]
:  INT {$value = Integer.parseInt($INT.text);}
  | ID
    {
    Integer v = (Integer)memory.get($ID.text);
    if ( v!=null ) $value = v.intValue();
    else System.err.println("undefined variable "+$ID.text);
    }
  | '(' e=expr ')' {$value = $e.value;}
  ;

ID : ('a'..'z'|'A'..'Z')+ ;
INT : '0'..'9'+ ;
NEWLINE: '\r'? '\n' ;
WS : (' |\t')+ {skip();} ;
```

```
import org.antlr.runtime.*;
public class Test {
    public static void main(String[] args) throws Exception
    {
        ANTLRInputStream input = new
ANTLRInputStream(System.in);
        ExprLexer lexer = new ExprLexer(input);
        CommonTokenStream tokens = new
CommonTokenStream(lexer);
        ExprParser parser = new ExprParser(tokens);
        parser.prog();
    }
}
```


/Users/bovet/ Grammars/Demo/Expr.g

```
grammar Expr;

@header {
package test;
import java.util.HashMap;
}

@lexer::header {package test;}

@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}

prog: stat+ ;

stat: expr NEWLINE [System.out.println($expr.value);
| ID '=' expr NEWLINE
{memory.put($ID.text, new Integer($expr.value));}
| NEWLINE
:
;

expr returns [int value]
: e=multExpr {$value = $e.value;}
( '+' e=multExpr {$value += $e.value;}
| '-' e=multExpr {$value -= $e.value;}
)*
```

prog

Line Endings: Unix (LF) Ignore rules: WS

2+3*4

Zoom

Syntax Diagram Interpreter Debugger Console

9 rules 1:1 Writable

/Users/bovet/ Grammars/Demo/Expr.g

- prog
- stat
- expr
- multExpr
- atom
- ID
- INT
- NEWLINE
- WS

```

expr returns [int value]
: e=multExpr {$value = $e.value;}
( '+' e=multExpr {$value += $e.value;}
| '-' e=multExpr {$value -= $e.value;}
)*

multExpr returns [int value]
: e=atom {$value = $e.value;} (* e=atom {$value *= $e.value;})*

atom returns [int value]
: INT {$value = Integer.parseInt($INT.text);}
| ID
{
Integer v = (Integer)memory.get($ID.text);
if (v!=null) $value = v.intValue();
else System.err.println("undefined variable "+$ID.text);
}
| '{ e=expr '}' {$value = $e.value;}
:
;

ID : ('a'..'z'|'A'..'Z')+ ;
INT : '0'..'9'+ ;
NEWLINE : '\r'? '\n' ;
WS : '\s'+ ;

```

Break on: All Location Consume LT Exception

#	Rule
0	prog
1	stat
2	expr
3	multExpr
4	atom

Input: 2

Parse Tree:

```

graph TD
  root --> prog
  prog --> stat
  stat --> expr
  expr --> multExpr
  multExpr --> atom
  atom --> 2

```

Stack: 2

Input Output Parse Tree AST Stack Events

Syntax Diagram Interpreter Debugger Console

9 rules 35:13 Writable

The screenshot displays a Java IDE window titled `/Users/bovet/ Grammars/Demo/Expr.g`. The editor shows a grammar definition for `Expr.g` with the following content:

```
grammar Expr;
@header {
package test;
import java.util.HashMap;
}
@lexer::header {package test;}
@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}
prog: stat+ ;
stat: expr NEWLINE [System.out.println($expr.value);
| ID '=' expr NEWLINE
{memory.put($ID.text, new Integer($expr.value));}
| NEWLINE
;
expr returns [int value]
: e=multExpr {$value = $e.value;}
( '+' e=multExpr {$value += $e.value;}
| '-' e=multExpr {$value -= $e.value;}
)*
```

The `prog: stat+` line is highlighted in yellow. Below the editor, the `Parse Tree` view shows the following structure:

```
graph TD
    root --> prog
    prog --> stat
    stat --> expr
    stat --> empty[ ]
    expr --> multExpr1[multExpr]
    expr --> plus[+]
    expr --> multExpr2[multExpr]
    multExpr1 --> atom1[atom]
    atom1 --> 2[2]
    multExpr2 --> atom2[atom]
    atom2 --> 3[3]
    multExpr2 --> star[*]
    star --> atom3[atom]
    atom3 --> 4[4]
```

The `Stack` view is empty. The `Output` view shows the number `14`. The `Debugger` tab is active, showing `9 rules`, `15:15`, and `Writable`. The `Stack` tab is also visible.

Was haben wir gelernt?

- ▶ Parsergeneratoren gehören heute zum Werkzeugset jeden Softwareingenieurs
- ▶ Neben Cocktail gibt es freie Initiativen, z.B. ANTLR
- ▶ Leider erfasst der Parser nur die kontextfreie Struktur des Programms oder Dokuments; Kontextbedingungen und Integritätsbedingungen bleiben der *statischen semantischen Analyse* vorbehalten.

The End