20. Parser-Generatoren im Technikraum Grammarware

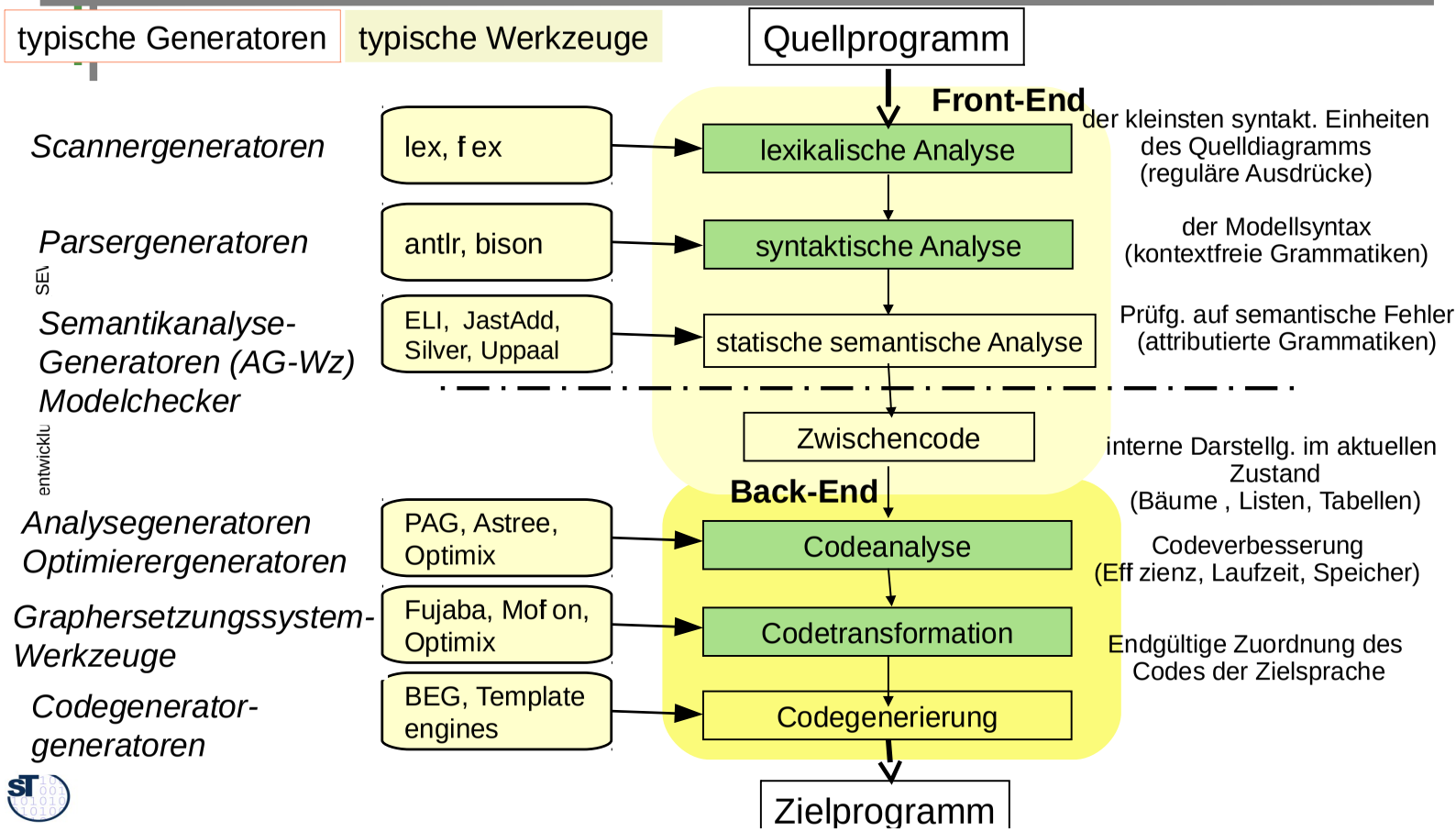
1) Grundlagen
2) Beispiel Taschenrechner

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

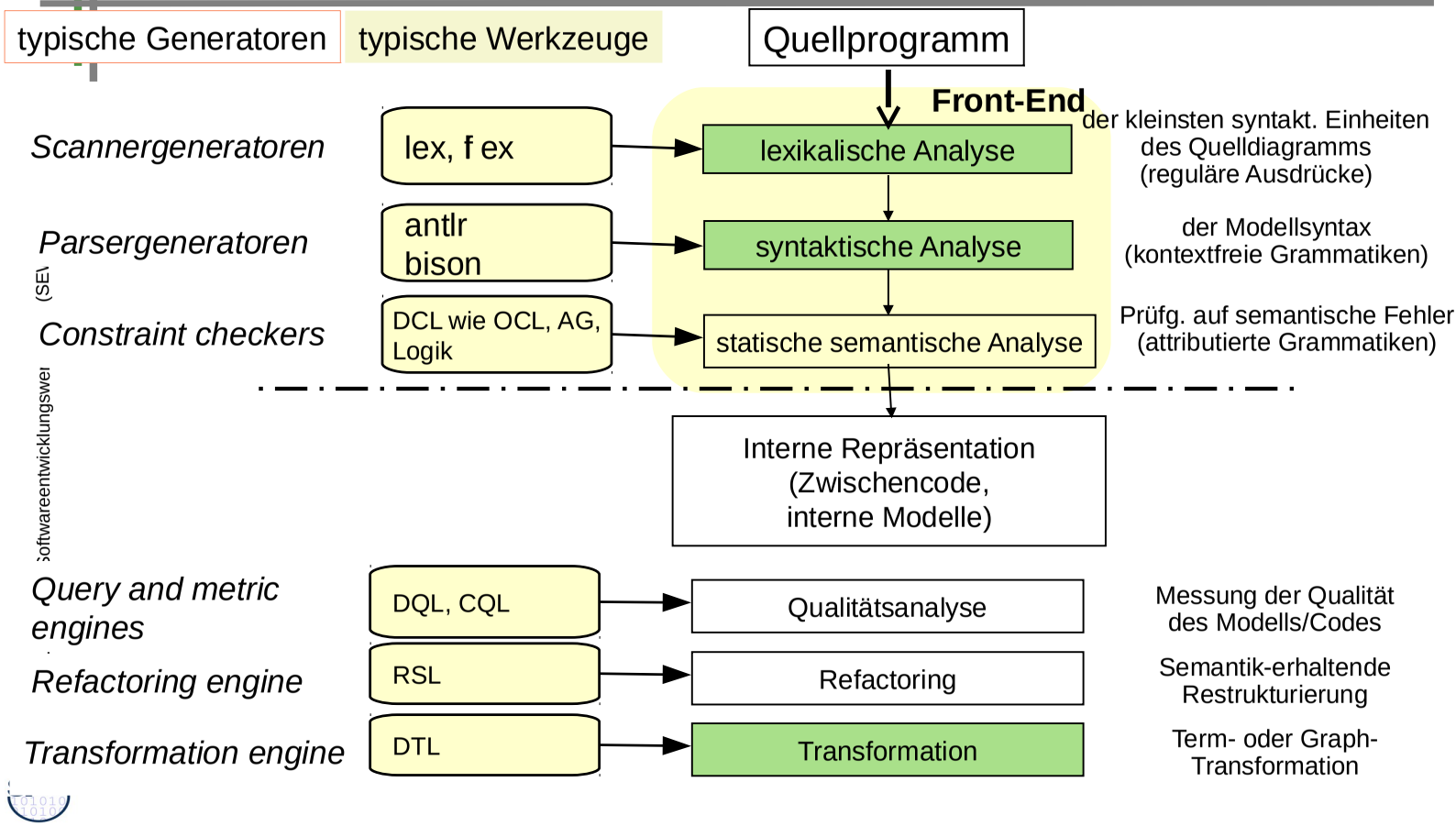
Literatur

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

Phasen eines Compilers und die erzeugenden Werkzeuge



Phasen eines Importers in ein Repository und die erzeugenden Werkzeuge



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?

Antwort

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

In dem ich aus Grammatiken Parser (Zerteiler) generiere

und

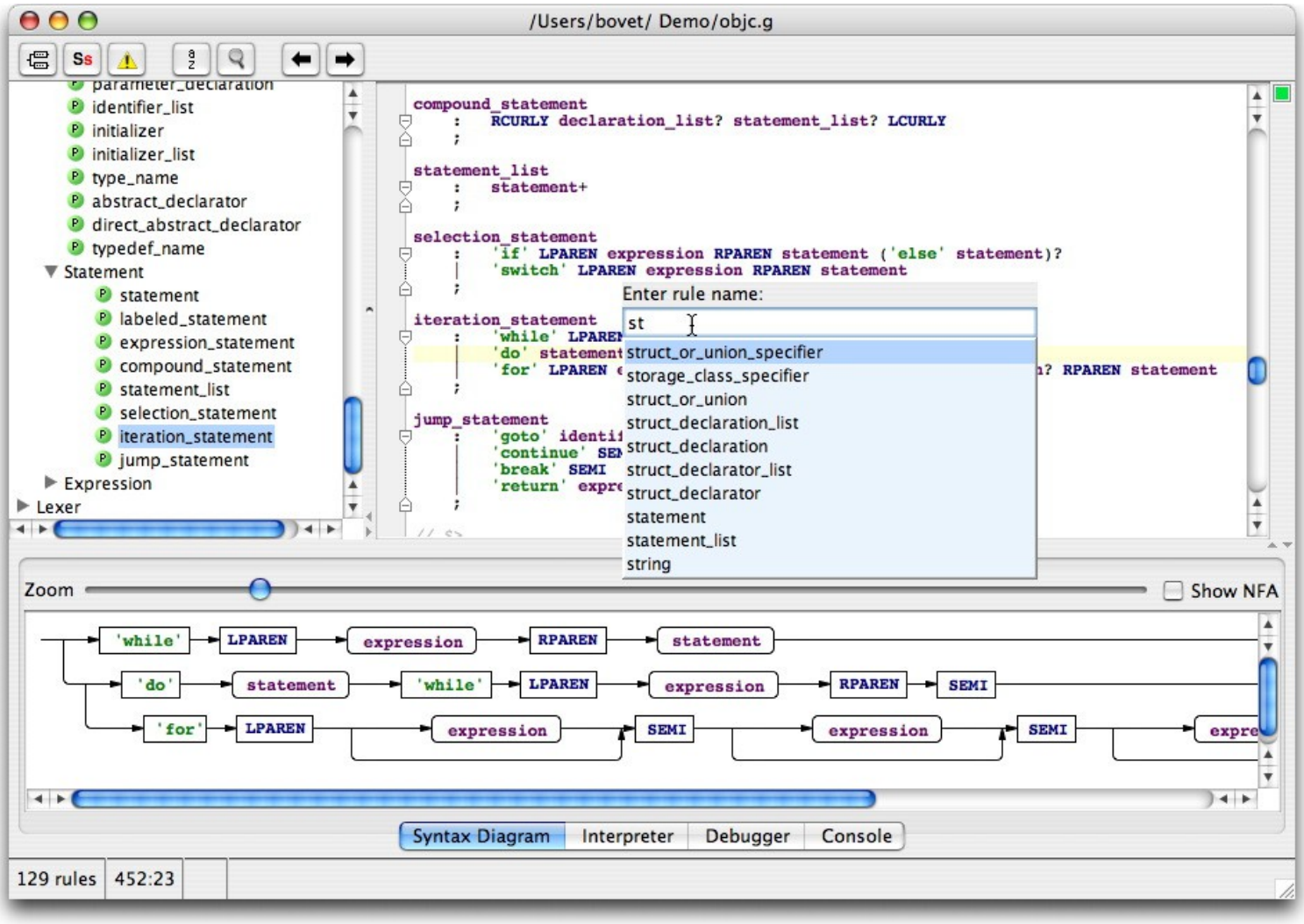
zusätzlich Prettyprinter

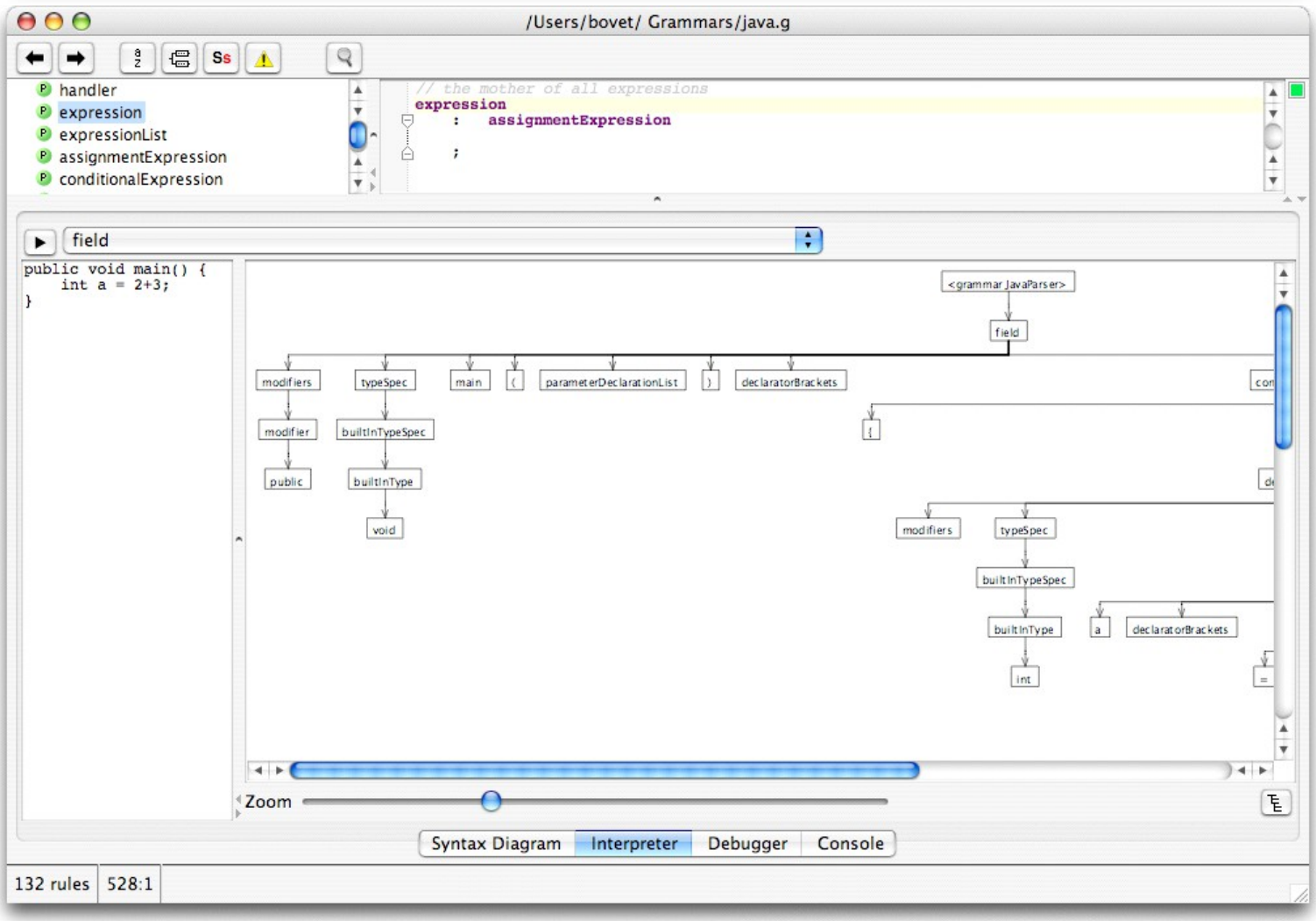
Beispiel EMFText

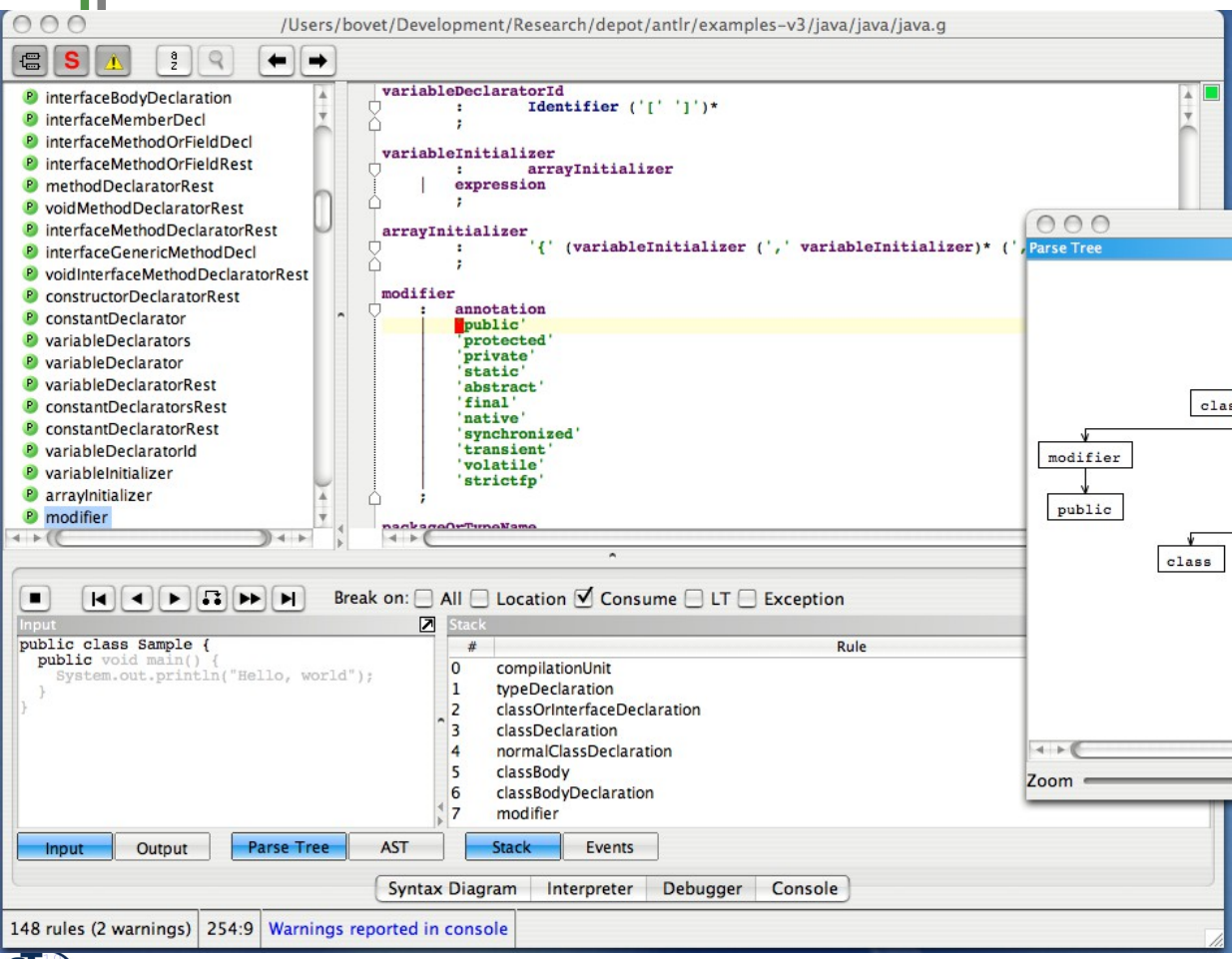
- Nutzt Parser-Generator ANTLR zur Generierung von Parsern
  - Parser und Metamodell werden aufeinander abgebildet (mapping), um konkrete auf abstrakte Syntax abzubilden
- Nutzt schablonengesteuerte Codegenerierung zur Erzeugung von Text und Programmen (siehe später).

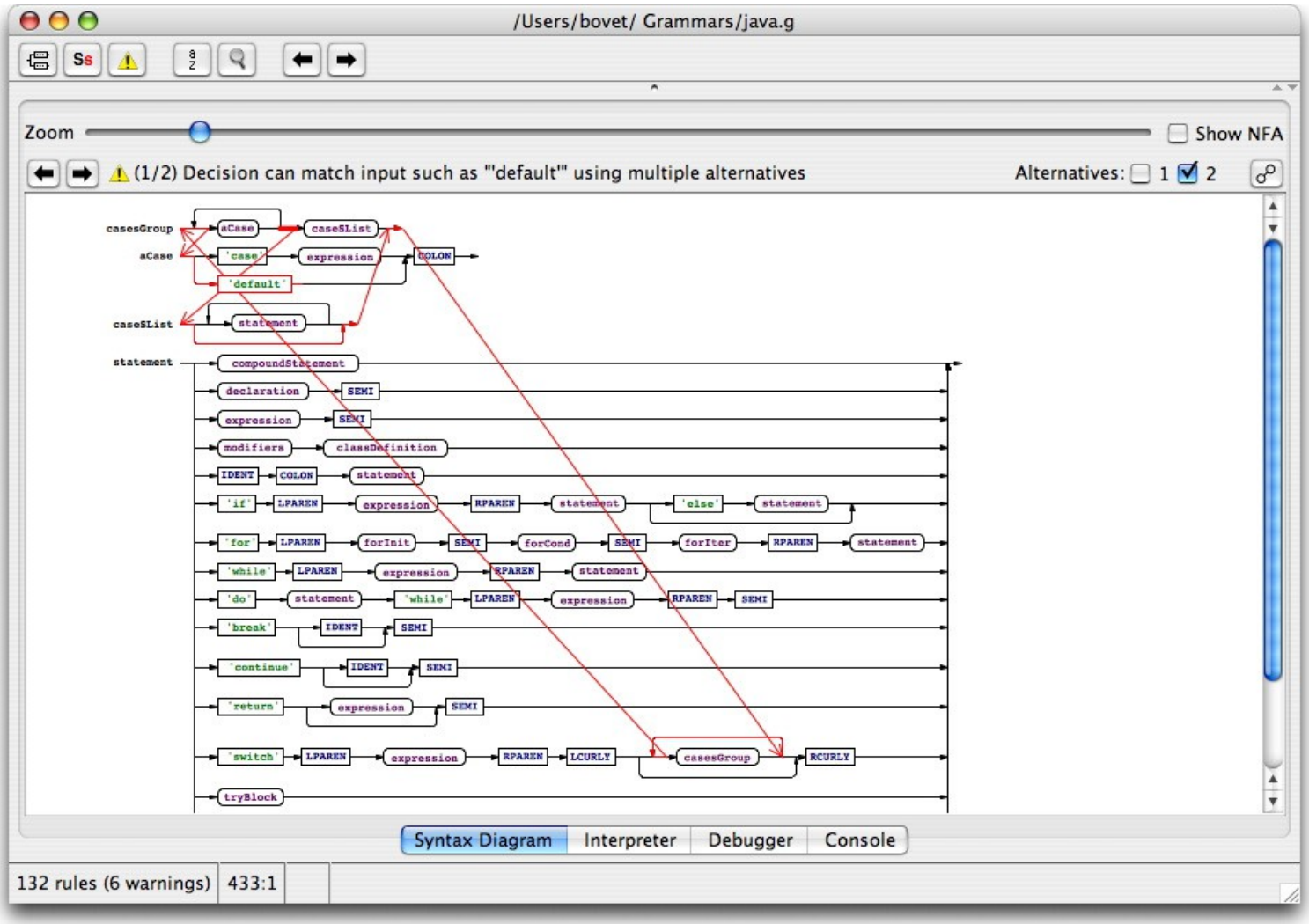
Beispiel: ANTLR www.antlr.org

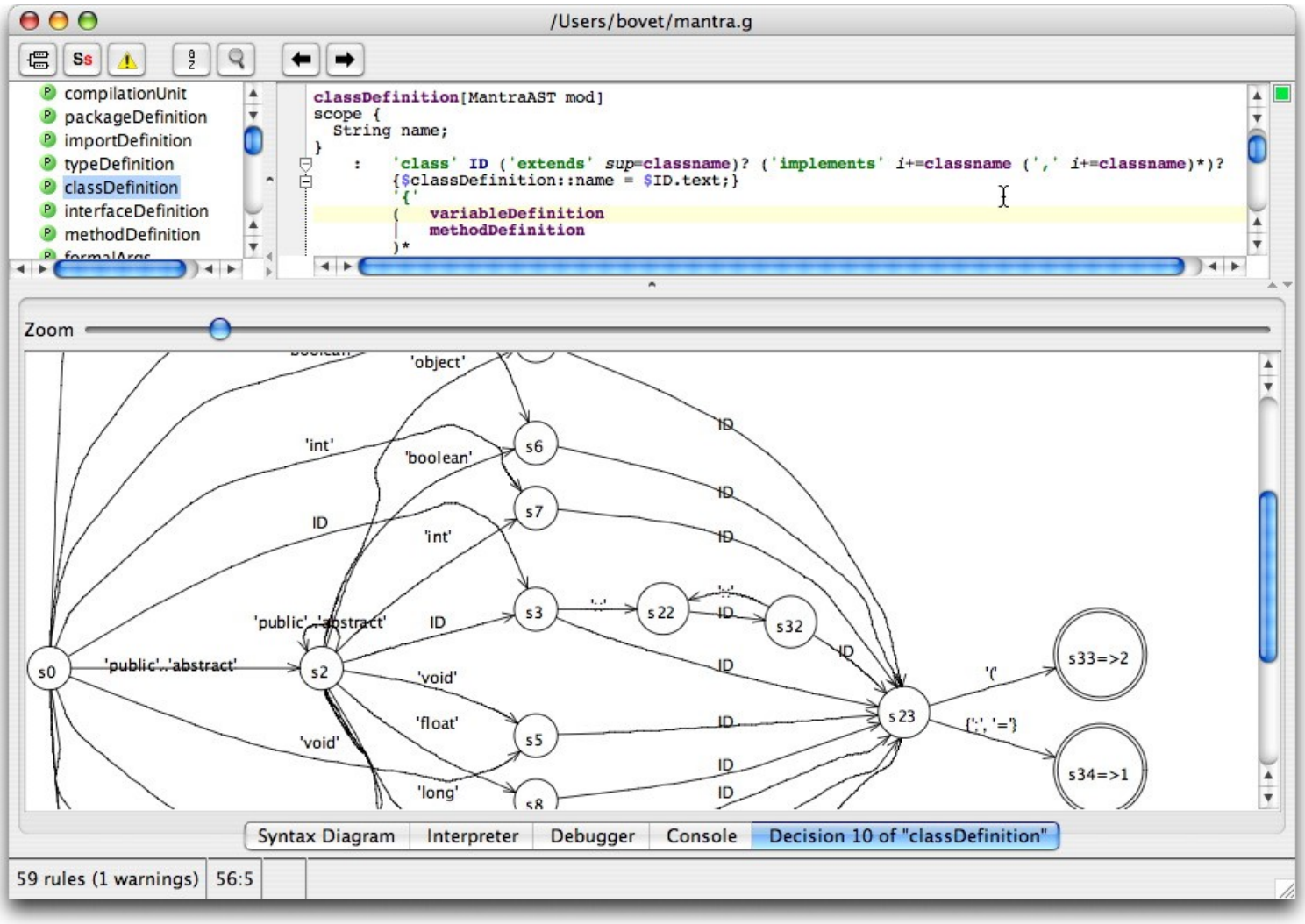
- In den 90ger Jahren gab es für C viele Parsergeneratoren
  - Cocktail's lalr, ell, lark www.cocolab.de
  - fnc2
  - flex und bison (gnu)
- Für Java ist ANTLR populär geworden
  - LL(k)
  - Generierter Parser mit Algorithmus “rekursiver Abstieg”
  - Etwas “gefärbte” Seite mit Geschichte http://www.bearcave.com/software/antlr/antlr_expr.html


/Users/bovet/ Demo/objc.g
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
| 'do' statement
| 'for' LPAREN
;
jump_statement
: 'goto' identi
| 'continue' SE
| 'break' SEMI
| 'return' expr
;
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
'while'
LPAREN
expression
RPAREN
statement
'do'
statement
'while'
LPAREN
expression
RPAREN
SEMI
'for'
LPAREN
expression
SEMI
expression
SEMI
Syntax Diagram
Interpreter
Debugger
Console
129 rules
452:23


/Users/bovet/ Grammars/java.g
handler
expression
expressionList
assignmentExpression
conditionalExpression
// the mother of all expressions
expression
: assignmentExpression
;
field
public void main() {
int a = 2+3;
}
<grammar JavaParser>
field
modifiers
typeSpec
main
(
parameterDeclarationList
)
declaratorBrackets
modifier
builtInTypeSpec
public
builtInType
void
{
modifiers
typeSpec
builtInTypeSpec
builtInType
int
a
declaratorBrackets
Zoom
Syntax Diagram
Interpreter
Debugger
Console
132 rules
528:1


/Users/bovet/Development/Research/depot/antlr/examples-v3/java/java/java.g
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'
;
Parse Tree
Parse Tree
compilationUnit
typeDeclaration
classOrInterfaceDeclaration
modifier
classDeclaration
public
normalClassDeclaration
class
Sample
classBody
{
modifier
public
Zoom
Break on: All Location Consume LT Exception
Input
public class Sample {
public void main() {
System.out.println("Hello, world");
}
}
Stack
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
148 rules (2 warnings)
254:9
Warnings reported in console


/Users/bovet/ Grammars/java.g
Zoom
Show NFA
(1/2) Decision can match input such as "'default'" using multiple alternatives
Alternatives: 1 2
casesGroup
aCase
caseSList
statement
compoundStatement
declaration
SEMI
expression
modifiers
classDefinition
IDENT
COLON
'if'
LPAREN
RPAREN
'else'
'for'
forInit
forCond
forIter
'while'
'do'
'break'
'continue'
'return'
'switch'
LCURLY
RCURLY
tryBlock
Syntax Diagram
Interpreter
Debugger
Console
132 rules (6 warnings)
433:1

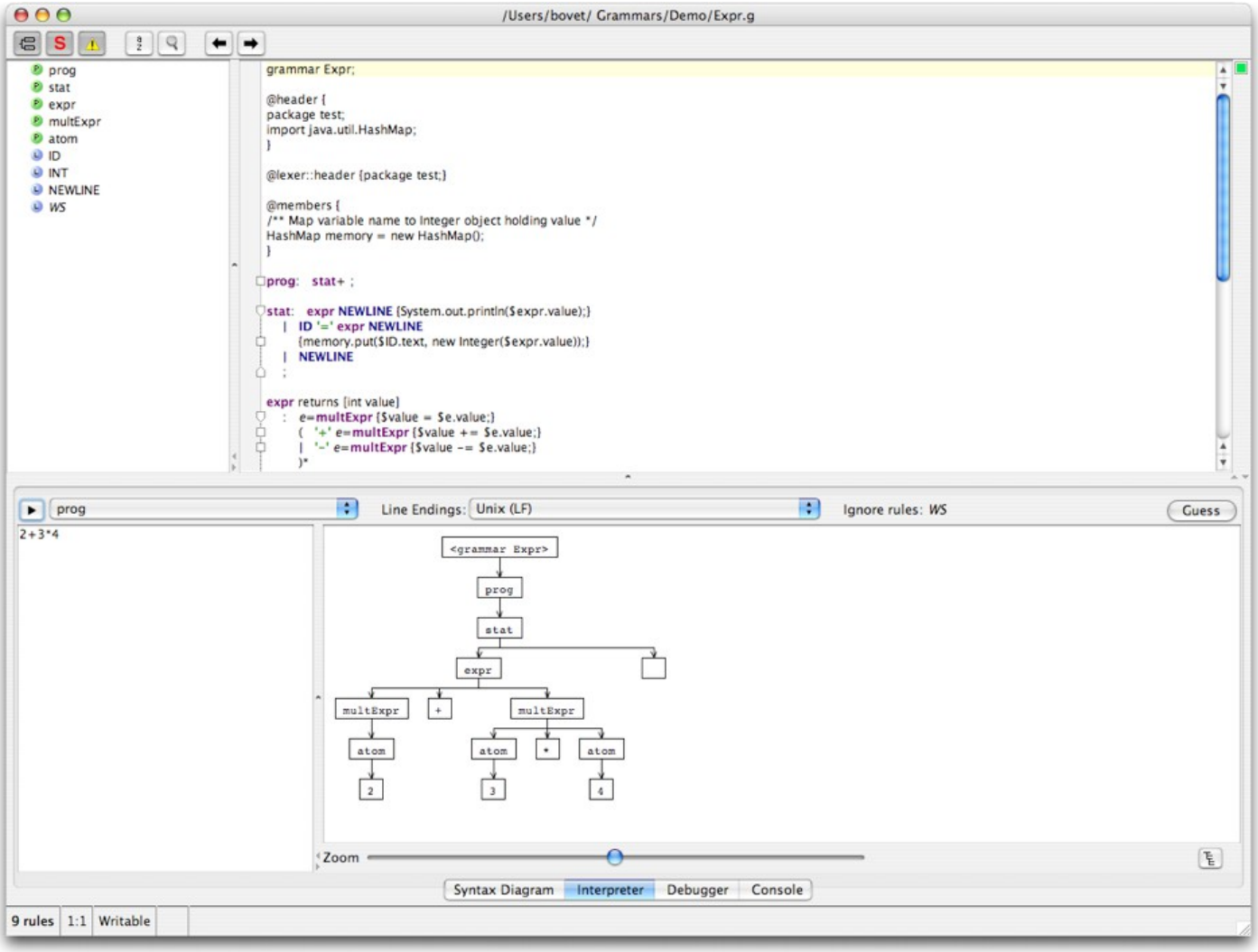


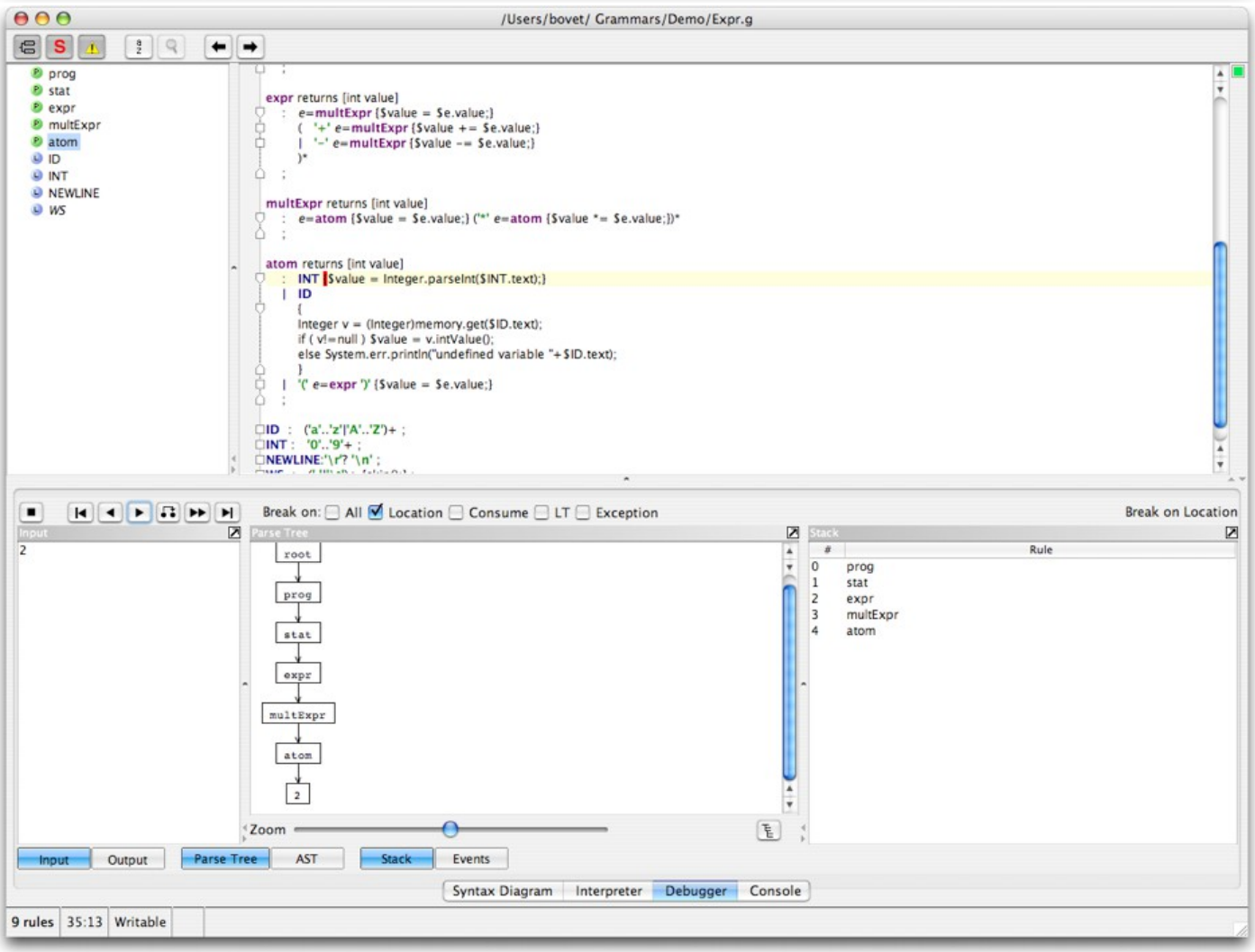
13.2 Ein Taschenrechner

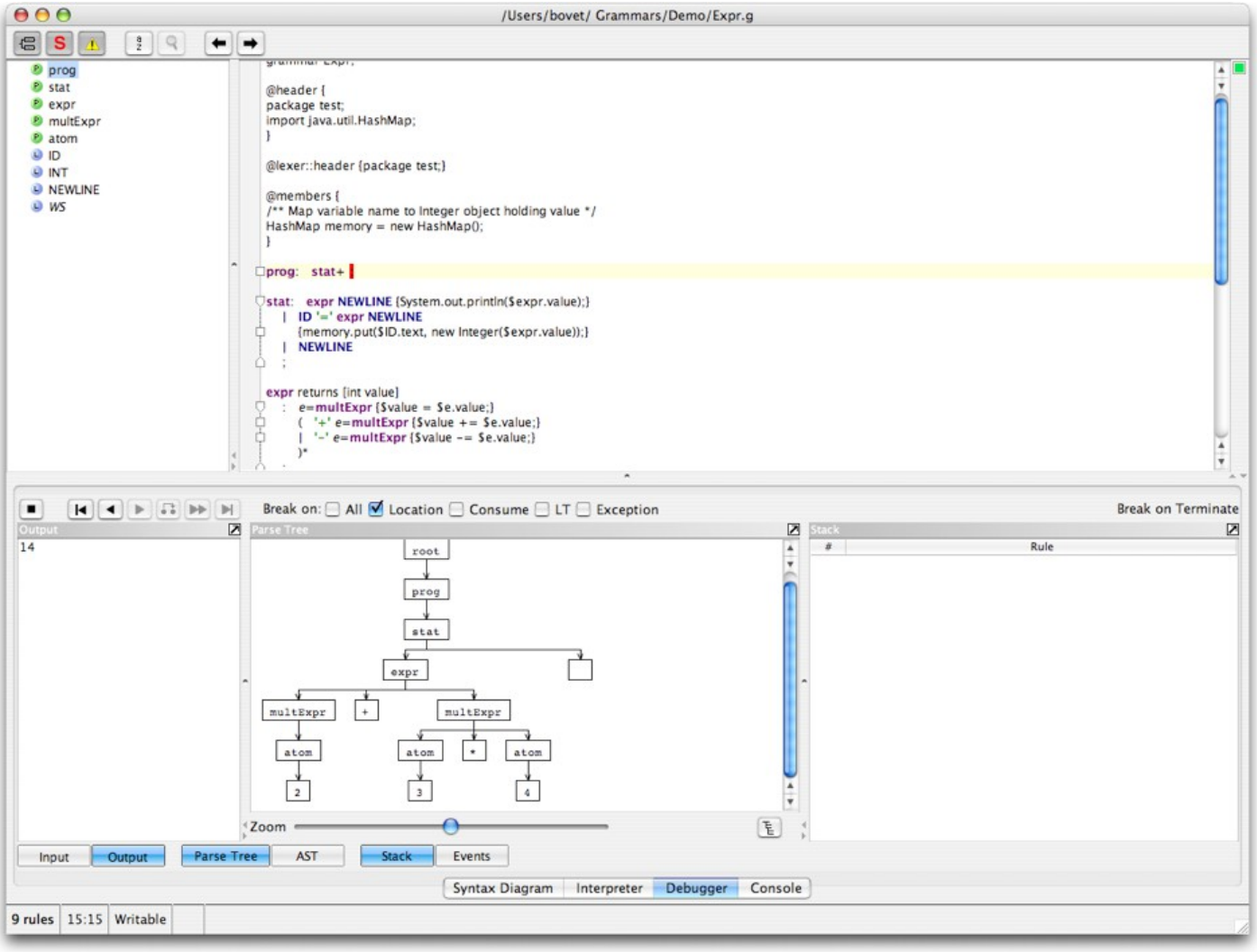
```
grammar Expr;
@header {
package test;
import java.util.HashMap;
}
@lexer::header {package test;}
@members {
/** Map variable name to Integer object holding value */
HashMap memory = new HashMap();
}
prog:   stat+ ;

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();} ;
```


Ansteuerung

```
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
prog
stat
expr
multExpr
atom
ID
INT
NEWLINE
WS
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
Guess
2+3*4
<grammar Expr>
prog
stat
expr
multExpr
+
multExpr
atom
atom
*
atom
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' ;
Break on: All Location Consume LT Exception
Break on Location
Input
2
Parse Tree
root
prog
stat
expr
multExpr
atom
2
Stack
Rule
0 prog
1 stat
2 expr
3 multExpr
4 atom
Zoom
Input
Output
Parse Tree
AST
Stack
Events
Syntax Diagram
Interpreter
Debugger
Console
9 rules
35:13
Writable



Was haben wir gelernt?

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

Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

The End