

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

Literatur

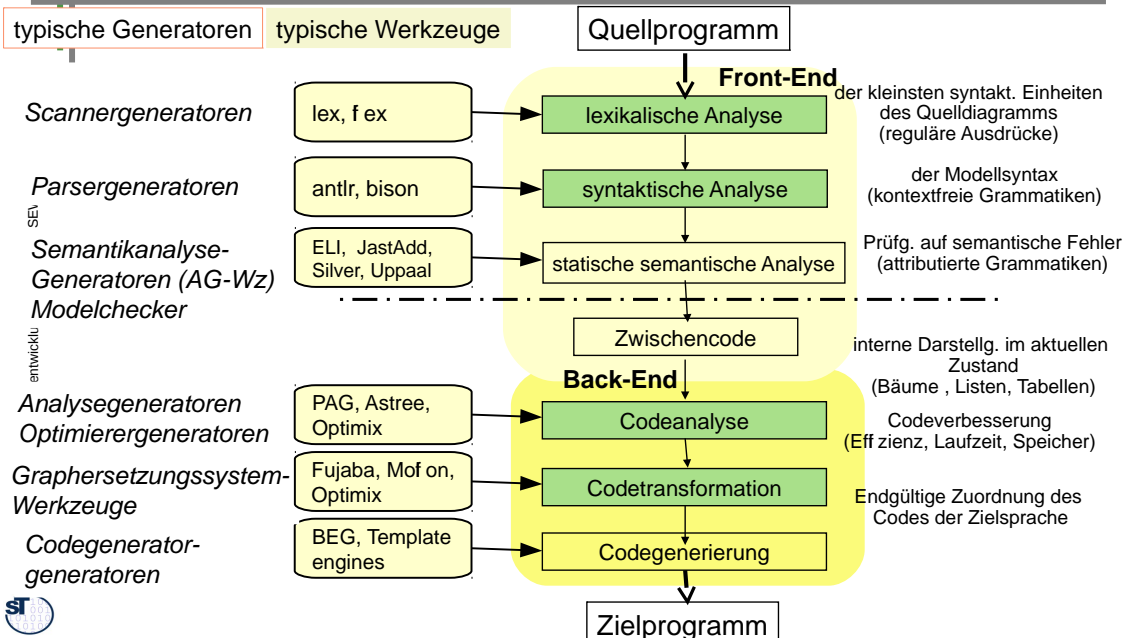
2

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

Prof. U. Aßmann, Softwareentwicklungswerkzeuge (SEW)

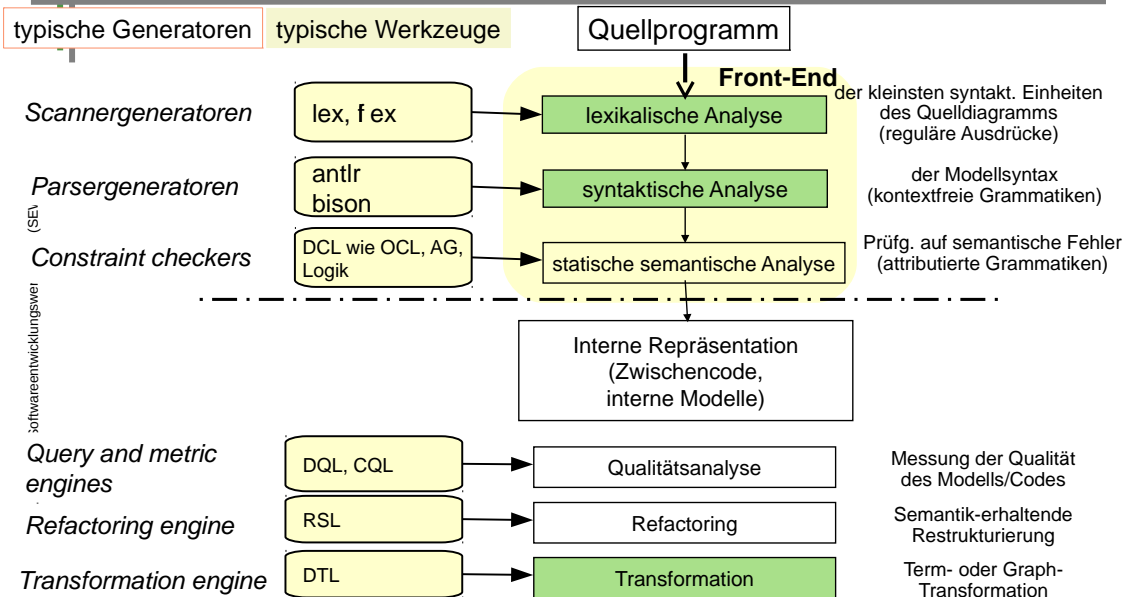
Phasen eines Compilers und die erzeugenden Werkzeuge

3



Phasen eines Importers in ein Repository und die erzeugenden Werkzeuge

4



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 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

/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

```

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 storage_class_specifier? struct_or_union specifier? RPAREN statement

jump_statement
: 'goto' ident;
: 'continue' SEMI;
: 'break' SEMI;
: 'return' expr;

```

Enter rule name: st

Zoom

Syntax Diagram Interpreter Debugger Console

129 rules 452:23

/Users/bovet/ Grammars/java.g

- handler
- expression
- expressionList
- assignmentExpression
- conditionalExpression

```

expression
: assignmentExpression
: ;

```

field

```

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

```

field

modifiers typeSpec main { parameterDeclarationList } declaratorBrackets

modifier builtinTypeSpec builtinType void

modifiers typeSpec builtinTypeSpec builtinType 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
- constantDeclaratorRest
- variableDeclaratorId
- variableInitializer
- arrayInitializer
- modifier

```

variableDeclaratorId
: identifier ('[' ']')*

variableInitializer
: expression

arrayInitializer
: '{' (variableInitializer (';' variableInitializer)* ';' )? '}'

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

```

Parse Tree

```

graph TD
  compilationUnit --> typeDeclaration
  compilationUnit --> classOrInterfaceDeclaration
  typeDeclaration --> modifier
  typeDeclaration --> classDeclaration
  classOrInterfaceDeclaration --> classDeclaration
  classDeclaration --> normalClassDeclaration
  normalClassDeclaration --> class
  normalClassDeclaration --> classBody
  classBody --> classBodyDeclarator
  classBodyDeclarator --> modifier
  classBodyDeclarator --> public

```

Stack

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

Input

```

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

```

Break on: All Location Consume LT Exception

Syntax Diagram Interpreter Debugger Console

148 rules (2 warnings) 254:9 Warnings reported in console

/Users/bovet/ Grammars/java.g

Zoom

Alternatives: 1 2

Decision can match input such as "default" using multiple alternatives

casesGroup

```

graph TD
  casesGroup --> eCase
  casesGroup --> caseList
  eCase --> case
  eCase --> expression
  caseList --> statement
  caseList --> default
  statement --> compoundStatement
  compoundStatement --> declaration
  compoundStatement --> expression
  declaration --> SEMI
  expression --> SEMI
  modifiers --> classDefinition
  classDefinition --> IDENT
  classDefinition --> COLON
  classDefinition --> statement
  'if' --> LPAREN
  'if' --> expression
  'if' --> RPAREN
  'if' --> statement
  'if' --> 'else'
  'if' --> statement
  'for' --> LPAREN
  'for' --> forInit
  'for' --> SEMI
  'for' --> forCond
  'for' --> SEMI
  'for' --> forIter
  'for' --> RPAREN
  'for' --> statement
  'while' --> LPAREN
  'while' --> expression
  'while' --> RPAREN
  'while' --> statement
  'do' --> statement
  'do' --> 'while'
  'do' --> LPAREN
  'do' --> expression
  'do' --> RPAREN
  'do' --> SEMI
  'break' --> IDENT
  'break' --> SEMI
  'continue' --> IDENT
  'continue' --> SEMI
  'return' --> expression
  'return' --> SEMI
  'switch' --> LPAREN
  'switch' --> expression
  'switch' --> RPAREN
  'switch' --> LCURLY
  'switch' --> casesGroup
  'switch' --> RCURLY
  tryLock

```

Syntax Diagram Interpreter Debugger Console

132 rules (6 warnings) 433:1

1

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"

59 rules (1 warnings) 56.5

13.2 Ein Taschenrechner

14

15

```

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();} ;

```

16 Ansteuerung

16

```

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();
    }
}

```



17

The screenshot shows a code editor with the following grammar rules:

```

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

```

The parse tree for the expression "2+3*4" is shown below:

```

graph TD
    root[<grammar Expr>] --> prog
    prog --> stat
    stat --> expr
    expr --> multExpr1[multExpr]
    expr --> plus["+"]
    expr --> multExpr2[multExpr]
    multExpr1 --> atom1[atom]
    atom1 --> 2
    multExpr2 --> atom2[atom]
    atom2 --> 3
    multExpr2 --> plus2["+"]
    multExpr2 --> atom3[atom]
    atom3 --> 4

```

9 rules 1:1 Writable

18

The screenshot shows the same grammar as in slide 17. The parse tree is identical. The stack is shown on the right:

```

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

```

9 rules 35:13 Writable

19

The screenshot shows the same grammar as in slide 17. The parse tree is identical. The stack is shown on the right:

```

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

```

9 rules 15:15 Writable

20

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 kontextfreie Struktur des Programms oder Dokuments; Kontextbedingungen und Integritätsbedingungen bleiben der *statischen semantischen Analyse* vorbehalten.

