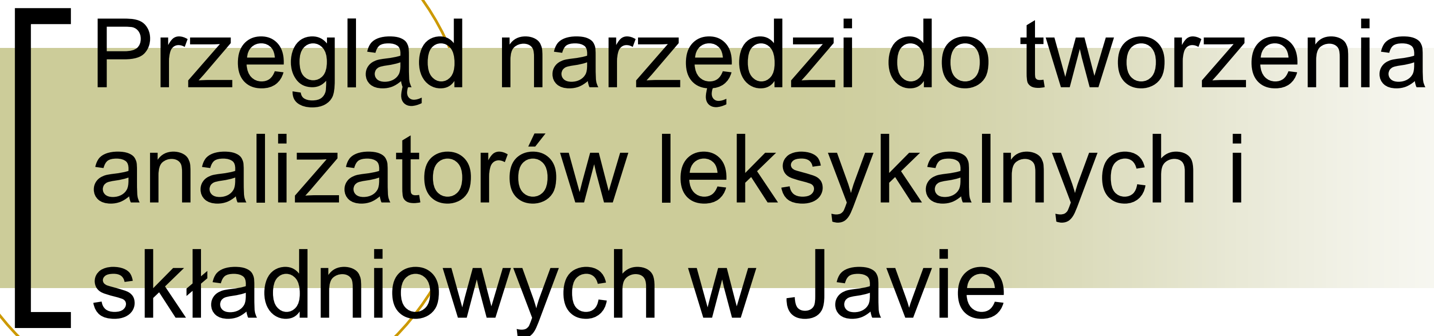
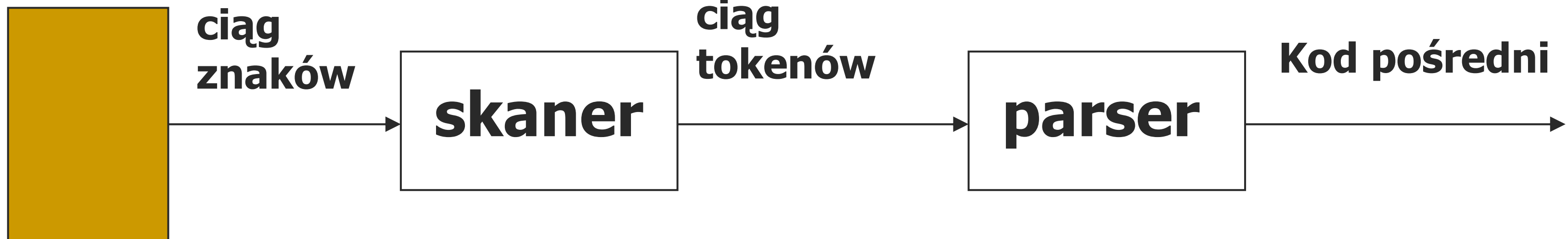


Przegląd narzędzi do tworzenia analizatorów leksykalnych i składniowych w Javie

Autorzy:
Jakub Lortz
Jakub Kapusta

Znany wszystkim schemat



O czym będziemy mówić?

- ANTLR
- Grammatica
- JLex/JFlex
- JavaCC
- Rats
- SableCC
- I inne…

ANTLR
ANother Tool for Language Recognition

- Generator parserów typu LL(k)
- Bardzo popularny
- Kod wynikowy w Javie, C# i C++
- Bogata dokumentacja, duża ilość tutoriali i gotowych gramatyk
- Open Source na licencji BSD
- Opis gramatyki – EBNF z możliwoscią wstawiania kodu w języku docelowym
- Wbudowany analizator leksykalny

http://www.antlr.org

ANTLR
Budowa leksera i parsera

Lekser

```
class ExprLexer extends Lexer;
 options {
            k=2 ;
}
 LPAREN  :          '(' ;
 RPAREN  :          ')' ;
 PLUS    :          '+' ;
 STAR    :          '*' ;
 INT     :          ('0'..'9')+ ;
 WS      :          ( ' '
                    | '\r' '\n'
                    | '\n'
                    )
            {$setType(Token.SKIP) ; }
            ;
```

Parser

```
class ExprParser extends Parser;

expr: mexpr (PLUS mexpr)* ;

mexpr :   atom (STAR atom)* ;

atom:       INT
          | LPAREN expr RPAREN ;
```

ANTLR
Uruchamianie

- Klasy parsera i leksera znajdują się w tym samym pliku o rozszerzeniu *.g
- Aby wygenerować program w javie należy wykonać

  $ java antlr.Tool gramatyka.g
- W programie możemy korzystać z naszych zdefiniowanych klas parsera i leksera, przykład najprostszego programu wykorzystującego gramatykę znajdziecie już na następnym slajdzie.

ANTLR
Wykorzystanie w programie

Program parsuje standardowe wejście sprawdzając zgodność z gramatyką
Zdefiniowaną wcześniej

```
import antlr.*;

public class Main {
		public static void main(String[] args) throws Exception {
		ExprLexer lexer = new ExprLexer(System.in);
		ExprParser parser = new ExprParser(lexer);
		parser.expr();
		}
}
```

Grammatica

- Generator parserów typu LL(k)
- Kod wynikowy w Javie i C#
- Sporo dostępnej dokumentacji, również do API udostępnianego przez wygenerowane parsery
- Oddzielona gramatyka od kodu parsera
- Automatycznie przeprowadzany proces error recovery
- Open source na licencji GPL

http://grammatica.percederberg.net/index.html

Grammatica
Ogólna budowa pliku gramatyki

- Grammatica używa własnego formatu gramatyki, przechowywana jest ona w plikach o rozszerzeniu *.grammar*

```
%header%
 GRAMMARTYPE = "LL"
...
%tokens%
...
%productions%
 ...
```

Grammatica Tokeny

- Wzory tokenów mogą być wyrażone jako stringi (``..``) lub wyrażenia regularne(<<..>>)
- Składnia wyrażeń regularnych jest praktycznie taka sama jak ta zawarta w Java API (patrz java.util.regexp.Pattern)

```
STRING_TOKEN = "Value"
WHITESPACE      = <<[ \t\n\r]+>> %ignore%
UNKN_CHAR       = <<.>> %error unexpected token%
```

Grammatica
Produkcje i generowanie kodu parsera

- Produkcje mają składnie niemal identyczną jak w EBNF

```
Prod = "TokenString" OtherProd
        | TOKEN_NAME OtherProd ;
```

- Wygenerowanie kodu parsera:

```
# java -jar lib/grammatica-1.4.jar
testgram.grammar --javaoutput test
```

Grammatica
Wykorzystanie w kodzie

- Aby użyć gramatyki testgram do przeparsowania jakiegoś kodu wystarczy w swojej klasie dodać:

```
Parser parser = null;
parser = new TestGramParser(new StringReader(input));
return parser.parse();
```

- Jeżeli chcemy wykonywać określone działania przy parsowaniu należy posłużyć się inną metodą:

```
class ArithmeticCalculator extends ArithmeticAnalyzer {
        protected Node exitProd(Production  node) {
                ArrayList values = getChildValues(node);
                …//dowolny kod
                return node;
        }
}
```

Grammatica a ANTLR Porównanie

- Rozdzielenie gramatyki od kodu źródłowego napisanego w Javie (ułatwia zrozumienie kodu i zwiększa uniwersalność)
- Oddzielenie gramatyki od generatora parsera (umożliwia wykorzystanie pliku gramatyki przez inne generatory)
- Automatyczny error recovery, w ANTLR można to uzyskać przez dodawanie klauzul catch
- Możliwość parsowania bez generowania kodu (w locie)
- Bardzo przejrzysty kod wynikowy
- Wolniejsze niż w ANTLR generowanie tokenów
- Mniejsza dostępność gotowych gramatyk

JLex

- Napisany w Javie dla Javy – dostępny kod źródłowy
- Licencja open source
- Oparty na Lexie
- Współpracuje z generatorem parserów CUP
- Niezbyt bogata dokumentacja

http://www.cs.princeton.edu/~appel/modern/java/JLex/

JLex – budowa pliku

user code

%%

JLex directives

%%

regular expression rules

JLex – dyrektywy

- kod w klasie skanera - %{ ... %}
- inicjalizacja
- deklaracje stanów
- opcje

JLex – przykład

```
package Example;


import java_cup.runtime.Symbol;
%%
%cup
%%
";" { return new Symbol(sym.SEMI); }
"+" { return new Symbol(sym.PLUS); }
"*" { return new Symbol(sym.TIMES); }
"(" { return new Symbol(sym.LPAREN); }
")" { return new Symbol(sym.RPAREN); }
[0-9]+ { return new Symbol(sym.NUMBER, new Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
. { System.err.println("Illegal character: "+yytext()); }
```

CUP

- Popularny generator parserów
- Tworzy parsery LALR
- Oparty na YACC
- Pozwala na stosowanie gramatyk niejednoznacznych przez zastosowanie priorytetów tokenów

http://www.cs.princeton.edu/~appel/modern/java/CUP/

CUP

- Wywołanie:

  java java_cup.Main < parser.cup

- Pliki wyjściowe:

  sym.java - deklaracje tokenów

  parser.java - właściwy parser

CUP – budowa pliku gramatyki

- Lista importowanych pakietów
- Dodatkowy kod

  action code {: … :}

  init with {: … :}

  scan with {: … :}
- Deklaracje symboli terminalnych i nieterminalnych
- Deklaracje priorytetów terminali
- Specyfikacja gramatyki

CUP
Przykładowy plik

```
import java_cup.runtime.*;

/* Preliminaries to set up and use the
scanner.  */
init with {: scanner.init();              :};
scan with {: return
scanner.next_token(); :};

/* Terminals (tokens returned by the
scanner). */
terminal           SEMI, PLUS, MINUS,
TIMES, DIVIDE, MOD;
terminal           UMINUS, LPAREN,
RPAREN;
terminal Integer   NUMBER;

/* Non-terminals */
non terminal            expr_list,
expr_part;
non terminal Integer    expr;
```

```
/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;

/* The grammar */
expr_list ::= expr_list expr_part
      |
              expr_part;

expr_part ::= expr:e
      {: System.out.println("= " + e); :}
              SEMI
      ;
```

CUP
Przykładowy plik

```
expr      ::= expr:e1 PLUS expr:e2
      {: RESULT = new
Integer(e1.intValue() + e2.intValue());
:}
      |
            expr:e1 MINUS expr:e2
            {: RESULT = new
Integer(e1.intValue() - e2.intValue());
:}
      |
            expr:e1 TIMES expr:e2
      {: RESULT = new
Integer(e1.intValue() * e2.intValue());
:}
      |
            expr:e1 DIVIDE expr:e2
      {: RESULT = new
Integer(e1.intValue() / e2.intValue());
:}
 |
            expr:e1 MOD expr:e2
      {: RESULT = new
Integer(e1.intValue() %
e2.intValue()); :}
      |
            NUMBER:n
      {: RESULT = n; :}
      |
            MINUS expr:e
      {: RESULT = new Integer(0 -
e.intValue()); :}
      %prec UMINUS
      |
            LPAREN expr:e RPAREN
      {: RESULT = e; :}
      ;
```

JavaCC

- Jeden z najpopularniejszych generatorów parserów w Javie
- generuje parsery LL(1)
- specyfikacja skanera i parsera w jednym pliku
- możliwość generowania drzew AST (JJTree)
- notacja EBNF

https://javacc.dev.java.net/

JavaCC – budowa pliku gramatyki

- *Opcje i deklaracja klasy*

```
PARSER_BEGIN(Simple)

public class Simple {

  public static void main(String args[]) throws ParseException {
    Simple3 parser = new Simple(System.in);
    parser.Input();
  }

}

PARSER_END(Simple)
```

JavaCC – budowa pliku gramatyki

- *Specyfikacja skanera*

```
SKIP :
{ " " | "\t" | "\n" | "\r" }

TOKEN :
{
  <LBRACE: "{">
| <RBRACE: "}">
}
```

JavaCC – budowa pliku gramatyki

- *Gramatyka*

```
void Input() :
{ int count; }
{
  count=MatchedBraces() <EOF>
  { System.out.println("The levels of nesting is " + count); }
}
int MatchedBraces() :
{ int nested_count=0; }
{
  <LBRACE> [ nested_count=MatchedBraces() ] <RBRACE>
  { return ++nested_count; }
}
```

SableCC

- *Generuje parsery LALR(1)*
- *Wbudowany generator skanerów*
- *Gramatyka w notacji EBNF*
- *Automatyczna generacja drzew AST i klas tree-walker*
- *Separacja kodu wygenerowanego od kodu wpisanego przez użytkownika*

http://sablecc.org/

SableCC – budowa pliku gramatyki

- *Plik .grammar*
- *Nazwa pakietu*
- *Deklaracje tokenów*
- *Produkcje gramatyki*

SableCC - przykład

Plik .grammar:

```
Package postfix;

Tokens
 number = ['0' .. '9']+;
 plus = '+';
 minus = '-';
 mult = '*';
 div = '/';
 mod = '%';
 l_par = '(';
 r_par = ')';
 blank = (' ' | 13 | 10)+;

Ignored Tokens
 blank;
```

```
Productions
expr =
 {factor} factor |
 {plus} expr plus factor |
 {minus} expr minus factor;

factor =
 {term} term |
 {mult} factor mult term |
 {div} factor div term |
 {mod} factor mod term;

term =
 {number} number |
 {expr} l_par expr r_par;
```

SableCC - przykład

Klasa tree walker:

```
package postfix;
import postfix.analysis.*;
import postfix.node.*;
class Translation extends
DepthFirstAdapter {
 public void caseTNumber(TNumber
node) {
  System.out.print(node);
 }
 public void outAPlusExpr(APlusExpr
node) {
System.out.print(node.getPlus());
 }
public void
outAMinusExpr(AMinusExpr node) {
System.out.print(node.getMinus());
 }
 public void
outAMultFactor(AMultFactor node)
 {
  System.out.print(node.getMult());
 }
 public void outADivFactor(ADivFactor
node)
 {
  System.out.print(node.getDiv());
 }
 public void
outAModFactor(AModFactor node)
 {
  System.out.print(node.getMod());
 }
}
```

SableCC - przykład

Program zmieniający wyrażenie arytmetyczne w notacji infixowej na notację postfixową:

```
package postfix;
import postfix.parser.*;
import postfix.lexer.*;
import postfix.node.*;
import java.io.*;

public class Compiler {
 public static void main(String[] arguments) {
  try {
   System.out.println("Type an arithmetic expression:");
   Parser p = new Parser(
    new Lexer(
    new PushbackReader(
    new InputStreamReader(System.in), 1024)));
```

```
 // Parse the input.
   Start tree = p.parse();

   // Apply the translation.
   tree.apply(new Translation());
  }
  catch(Exception e)
  {
   System.out.println(e.getMessage());
  }
 }
}
```

Inne…

- **COCO/JAVA**

http://www.ssw.uni-linz.ac.at/Projects/Coco/Coco.html/#Java

- **GENERIC INTERPRETER** – narzędzie do tworzenia interpreterów i kompilatorów

http://www.csupomona.edu/~carich/gi/

- **JB** – zamienia pliki wynikowe GNU Bisona na pliki Javy

http://serl.cs.colorado.edu/~serl/misc/jb.html

- **OOPS** – parser LL(1)

http://luna2.informatik.uni-osnabrueck.de/alumni/bernd/oops/

Koniec