Translator języka Pascal do języka Python

Mateusz Trzopek, Olaf Byrdy

AiR, EAIiIB, AGH

Spis treści

Wstęp i opis użytego narzędzia

Wykorzystanym narzędziem jest ANTLR – ang. ANother Tool for Language Recognition. Jest to narzędzie służące do analizy leksykalnej przeprowadzanej na podstawie opisu gramatyki, która znajduje się w pliku *.g. ANTLR rozpoznaje gramatyki klasy LL(k) oraz używa parsingu top-down. Analiza rozpoczynana jest od symbolu początkowego, a następnie stosowana jest produkcja dla pierwszego napotkanego z lewej symbolu nieterminalnego. Domyślnie ANTLR generuje lekser i parser w języku Java. Rozprowadzany jest na zgodnej z zasadami wolnego oprogramowania licencji BSD.

Analiza leksykalna znajduje się wszędzie tam, gdzie wczytywane są dane o określonej składni. Podczas wczytywania tych danych należy rozpoznać ich składnię w celu dalszego ich przetwarzania. Początkowym etapem jest podzielenie wczytywanego ciągu znaków na tzw. *leksemy*. Leksem jest ciągiem znaków, który stanowi semantycznie niepodzielną całość. Następnie skaner przyporządkowuje leksemom *tokeny* i (opcjonalnie) *atrybuty*. Token niesie informację o rodzaju leksemu. Z kolei, gdy leksem danego rodzaju przenosi ze sobą pewną wartość, to obok tokenu występuje również atrybut, który równy jest tej wartości. Typowymi przykładami leksemów są:

- identyfikatory,
- napisy,
- słowa kluczowe,
- liczby,
- operacje arytmetyczne i relacje.

Przykład analizy leksykalnej ciągu znaków:

$$E := m * c\text{^}2$$

Z tego ciągu znaków wyodrębnione zostały następujące leksemy:

$$E, \quad :=, \quad m, \quad *, \quad c, \quad \text{^}, \quad 2, \quad ;$$

Z kolei ich reprezentacja za pomocą tokenów i atrybutów jest następująca:

Leksem	Token	Atrybut
E	identyfikator	„E”
:=	przypisanie	
m	identyfikator	„M”
*	mnożenie	
c	identyfikator	„C”
^	potęgowanie	
2	liczba całkowita	2
;	średnik	

Gramatyka w ANTLR

Ogólny opis

ANTLR przyjmuje gramatyki napisane zgodnie z Rozszerzoną notacją Backusa-Naura (EBNF) zapisane w pliku o rozszerzeniu .g. Notacja EBNF określa reguły produkcji w następujący sposób:

```
a : b;
```

Oznacza to, że symbol `a`, znajdujący się po lewej stronie produkcji, może być zastąpiony symbolem `b`, znajdującym się po prawej stronie produkcji. Jeśli `b` jest produkcją, zostaje zastąpiony prawą stroną produkcji `b`. Ilość symboli po prawej stronie produkcji (tokenów) jest nieograniczona. Możliwe jest wskazanie alternatywnych tokenów przy użyciu symbolu `|`:

```
a : b | c;
```

Ogólna forma pliku .g wygląda następująco:

```
grammar NazwaGramatyki;

options {…}

import …;

tokens {…}

@actionName {…}

produkcja1 : token1 | … | tokenN;

…

produkcjaN: token1 | … | tokenN;
```

Plik zawierający gramatykę X musi mieć nazwę X.g. Możliwa jest specyfikacja opcji, importów, listy tokenów. Każda gramatyka musi mieć zdefiniowaną nazwę gramatyki oraz co najmniej jedną produkcję. Pozostałe elementy są opcjonalne. Nazwy produkcji dla parsera muszą zaczynać się od małej litery, natomiast nazwy reguł dla leksera muszą zaczynać się z wielkiej litery.

W pliku gramatyki możliwe jest także umieszczenie kodu programu (domyslnie w języku Java) w nawiasach { }.

Na podstawie gramatyki ANTLR generuje parser i lekser, które są niezbędne przy procesie translacji.

Gramatyka pascal.g wykorzystana do stworzenia translatora

Gramatyka wykorzystana w translatorze z języka Pascal na język Python nosi nazwę pascal.g. Przykładowe fragmenty gramatyki zaprezentowane są poniżej.

Fragment sekcji tokens {...}

```
tokens {
   AND                = 'and'              ;
   BEGIN              = 'begin'            ;
   BOOLEAN            = 'boolean'          ;
   CASE               = 'case'             ;
   CHAR               = 'char'             ;
   CHR                = 'chr'              ;
   EXIT               = 'exit'             ;
   CONST              = 'const'            ;
   DIV                = 'div'              ;
   DO                 = 'do'               ;
   DOWNTO             = 'downto'           ;
   ELSE               = 'else'             ;
   END                = 'end'              ;
   FOR                = 'for'              ;
   FUNCTION           = 'function'         ;
   IF                 = 'if'               ;
   IN                 = 'in'               ;
   INTEGER            = 'integer'          ;
   LABEL              = 'label'            ;
   MOD                = 'mod'              ;
   NIL                = 'nil'              ;
   NOT                = 'not'              ;
   OR                 = 'or'               ;
   PROCEDURE          = 'procedure'        ;
   PROGRAM            = 'program'          ;
   REAL               = 'real'             ;
   REPEAT             = 'repeat'           ;
   THEN               = 'then'             ;
   TO                 = 'to'               ;
   TYPE               = 'type'             ;
   UNTIL              = 'until'            ;
   VAR                = 'var'              ;
   WHILE              = 'while'            ;
   STRING             = 'string'           ;

(...)
}
```

Fragment akcji @members

```
@members{
  List fnames = new ArrayList();
  int depth = 0;

  PrintWriter writer;
  File f;

  public pascalParser(CommonTokenStream input, String fileName) {
    super(input);

    try {
      f = new File(fileName);
      if(!f.exists()){
        new File("output").mkdir();
        f.createNewFile();
        System.out.println("Output file not found, new file
                                    created");
      }
      writer = new PrintWriter(fileName);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  (…)
}
```

W przedstawionym fragmencie widać dodanie do klasy parsera kilku pól oraz konstruktora umożliwiającego stworzenie parsera z możliwością obsługi plików.

Przykładowa produkcja zawierająca kod w języku Java umieszczony w nawiasach { }

```
procedureDeclaration
    : PROCEDURE^ pname=identifier
      {
      GenerateTabs(depth);
      writer.print("def " + $pname.text);
      }
      ( formalParameterList )? SEMI!
      ( block )
    ;
```

Produkcja ta określa deklarację procedury np.:

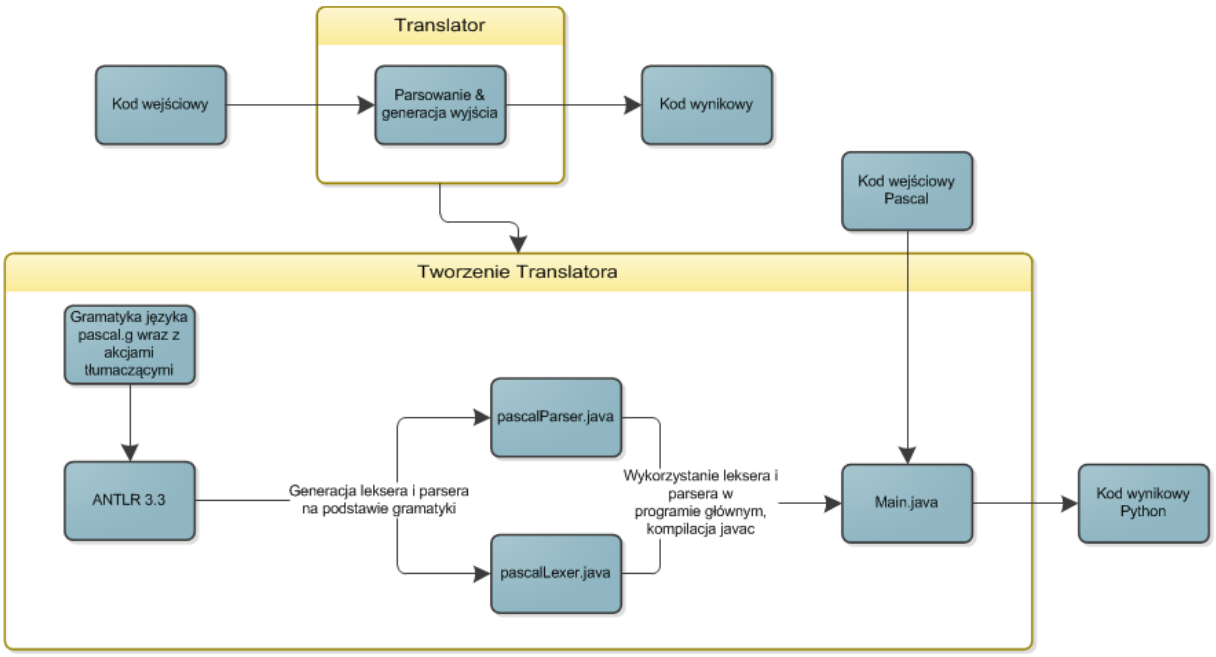
```
procedure nazwa_procedury(lista_argumentow)
```

i generuje częściowe tłumaczenie tej deklaracji na język Python:

```
def nazwa_procedury
```

Translator

Schemat działania translatora zaprezentowany jest poniżej:



Translator rozpoznaje poniższe struktury języka Pascal:

- Operacje arytmetyczne
- Operacje logiczne
- Instrukcje przypisania
- Instrukcje inkrementacji, dekrementacji
- Deklaracja stałych i zmiennych
- Wybór prosty wyrażeniami warunkowymi if
- Pętle for do while oraz repeat
- Instrukcje sterowania
- Standardowe procedury we/wy
- Standardowe procedury matematyczne
- Deklaracje i definicje funkcji
- Deklaracje i definicje procedur
- Wywołania procedur, funkcji

Instrukcja użytkowania

Aby skompilować plik wykonywalny niezbędna jest instalacja pakietu Java Development Kit, który można pobrać ze strony Oracle:
http://www.oracle.com/technetwork/java/javase/downloads/index.html (Download JDK).

Po zainstalowaniu pakietu należy upewnić się, czy zmienna środowiskowa Path jest poprawnie ustawiona. Można tego dokonać wpisując w konsoli systemowej polecenie `javac`. W przypadku, gdy zostanie zwrócony wydruk dostępnych funkcji, zmienna środowiskowa jest poprawnie ustawiona. W przeciwnym wypadku można ustawić zmienną w zaawansowanych ustawieniach systemu, dodając do zmiennej Path po średniku ścieżkę do katalogu bin JDK (przykładowo: C:\Program Files\Java\jdk1.7.0_21\bin) lub wpisać w konsoli polecenie:

```
>SET PATH=%path%;ŚCIEŻKA_DO_KATALOGU_BIN_JDK
```

Konieczne jest również ustawienie zmiennej Classpath ze ścieżką do pliku antlr-3.3-complete.jar. Można tego dokonać, wpisując w konsoli systemowej polecenie:

```
>SET CLASSPATH=%CLASSPATH%;SCIEZKA_DO_PLIKU_antlr-3.3-complete.jar
```

lub dopisując do zmiennej CLASSPATH w zaawansowanych ustawieniach sysemu ścieżkę do pliku antlr-3.3-complete.jar.

Gdy pakiet JDK jest poprawnie zainstalowany oraz ścieżki Path i Classpath są skonfigurowane, należy w systemowej konsoli przejść do katalogu, w którym znajdują się pliki: Main.java oraz pascal.g.

Pierwszym krokiem jest wygenerowanie parsera i leksera. Realizuje się to przy pomocy komendy:

```
>java org.antlr.Tool pascal.g
```

Następnie należy kompilować wygenerowane pliki oraz Main.java:

```
>javac *.java
```

Ostatecznie uruchomienie plików wykonywalnych odbywa się komendą:

```
>java Main input
```

gdzie `input` jest nazwą pliku (wraz z rozszerzeniem), który zawiera kod w języku Pascal. Translator zapisuje rezultat pracy, czyli przetłumaczony kod z języka Pascal na język Python w pliku: /output/result.py

Przykłady translacji kodu z języka Pascal na język Python

Przykład 1

W przykładzie zostanie przedstawiony wynik translacji prostego programu, deklarującego zmienne, którym następnie nadane są wartości z wykorzystaniem operacji arytmetycznych:

```
program expressionTest (input, output);
      var
            a, b : integer;
            c : real;
      begin
            a := 3;
            b := a * 4;
            c := (b + a)/ 2
      end.
```

Wynik translacji – kod w języku Python:

```
a = 3
b = a * 4
c = (b + a)/ 2
```

Drzewo syntaktyczne powyższego przykładu znajduje się w załączniku (`expressionTestTree.pdf`).

Przykład 2

Kolejny przykład jest bardziej rozbudowany i wykorzystuje większość możliwości translatora. Kod programu w języku Pascal:

```
program extendedTest(input, output);
var
        i, j, k, n:integer;
        a,b :real;

procedure Initiate(a,b:real);
begin
        i := 5 + int(a);
        randomize;
        j := random(10);
        k := sin(cos(20));
        inc(k);
        dec(j);
        writeln(i);
        writeln(j + k);
        readln(n);
end;

function Sum(a,b:integer): integer;
        begin
                if n = 1 then
                        begin
                                a := 5;
                                i := 5+b;
                                Sum := a + b - i;
                        end
                        else
                        begin
                                a := exp(5);
                                i := 5+ln(a);
                        end;
        for counter := 1 to 5 do
                begin
                        a := a+1;
                        i := counter - 1;
                end;
        repeat
                begin
                        a := 5;
                        i := 5+a;
                        k := Sum(i,j) * 2;
                end
        until (S=N);
        Sum := a + b;
end;
```

```
begin
        Initiate(2.4,1.53);
        if n = 1 then
                begin
                        a := 5;
                        i := 5+a;
                        if n = 1 then
                                begin
                                        a := Sum(a,i);
                                        i := 5+a;
                                end
                        else
                                begin
                                        a := 5;
                                        i := 5+a;
                                end;
                end
        else
                begin
                        a := 5;
                        i := 5+a;
                end;
end.
```

Wynik translacji – kod w języku Python:

```
def Initiate(a,b):
        i = 5 + math.floor(a)
        random.seed()
        j = random.random(10)
        k = math.sin(math.cos(20))
        k += 1
        j -= 1
        print i
        print j + k
        n =  raw_input

def Sum(a,b):
        if ( n == 1 ):
                a = 5
                i = 5+b
                return a + b - i

        else:
                a = math.exp(5)
                i = 5+math.log(a)

        for counter in range(1, 5 + 1):
                a = a+1
                i = counter - 1
```

```
        while (true):
                a = 5
                i = 5+a
                k = Sum(i,j) * 2

                if (S==N) break
        return a + b


Initiate(2.4,1.53)
if ( n == 1 ):
        a = 5
        i = 5+a
        if ( n == 1 ):
                a = Sum(a,i)
                i = 5+a

        else:
                a = 5
                i = 5+a


else:
        a = 5
        i = 5+a
```

Wnioski

Translator dobrze radzi sobie ze strukturami języka Pascal, które zostały wymienione wyżej. Ma jednak ograniczenia. Problemy przy translacji pojawiają się dla struktur `case` oraz `else if`. Inne problemy są związane z obsługą stringów i tablic oraz w przypadku, gdy parametry funkcji są różnego typu.