

PLY (Python Lex-Yacc)

Maciej Nowak
Arkadiusz Wieczorek

09.01.2017

1 Problematyka

W przetwarzaniu języka naturalnego (ang. natural language processing, NLP) programista ma za zadanie wyprowadzenie reguł dla komputera, aby ten mógł analizować, tłumaczyć czy też generować język naturalny zrozumiały dla człowieka. Istotą tego zagadnienia jest najczęściej wydawanie zwykłych poleceń do komputera w języku naturalnym. Sposoby komunikowania się z komputerem:

- urządzenia wskazujące
- polecenia w terminalu
- panele dotykowe
- komendy głosowe
- komunikacja obrazowa
- kontrolery ruchu



Rysunek 1: Samsung EyeCan - sterowanie kursorem za pomocą oka

2 Wstęp do gramatyki

2.1 Definicja gramatyki

Gramatyka jest to metoda opisująca dopuszczalne sekwencje wyrazów należących do leksykonu.

2.1.1 Przykładowy leksykon

```
{ 'trzy', 'troje', 'siedem', 'jedna', 'chłopców', 'drużyny', 'pilek',  
'piłka', 'małych', 'duże', 'wielka', 'niebieska', 'owalna' }
```

2.1.2 Przykładowe sekwencje dopuszczalne

- trzy duże drużyny
- troje małych chłopców
- siedem małych piłek
- jedna wielka niebieska owalna piłka

2.1.3 Przykładowe sekwencje niedopuszczalne

- trzy
- troje chłopców małych
- siedem małych dużych piłek
- jedna wielka piłek

2.2 Gramatyka formalnie

2.2.1 Definicja

Gramatyka jest zbiorem zasad dla ciągów tekstowych w języku formalnym. Zasady opisują w jaki sposób tworzyć ciągi tekstowe z alfabetu języka w taki sposób, aby były one zgodne ze składnią języka. Gramatyka nie opisuje znaczenia ciągu tekstowego lecz tylko jego formę.

2.2.2 Zapis formalny

$$G = \langle N, V, P, S \rangle$$

N - zbiór symboli nieterminalnych (symbole pomocnicze)

V - zbiór symboli terminalnych (słowa leksykonu)

P - zbiór produkcji języka (zasady gramatyki)

S - symbol startowy (jeden z symboli nieterminalnych)

2.2.3 Przykład gramatyki

- Gramatyka $G = \langle N, V, P, S \rangle$
- Zbiór symboli nieterminalnych $N = \{cmd, art, color, size, number, shape, kind\}$
- Zbiór symboli terminalnych $V = \{trzy, troje, siedem, chlopcow, druzyny, pilek, pilka, malych, duze, wielka, niebieska, owalna\}$
- Symbol startowy $S = cmd$

Zbiór produkcji języka $P =$

```
{
  cmd: (number, art),
  art: (size, color, shape, kind),
  number: ("trzy", "troje", "jedna", "siedem"),
  size: ("malych", "duze", "wielka"),
  color: ("niebieska"),
  shape: ("owalna"),
  kind: ("chlopcow", "druzyny", "pilek", "pilka")
}
```

2.3 Analiza leksykalna

2.3.1 Definicja

Analiza leksykalna polega na podzieleniu tekstu na tokeny, a następnie przypisanie każdemu z nich typu z leksykonu. Program, który dokonuje tej analizy nazywany jest lexer (np.: LEX).

2.3.2 Przykłady

```
IN:      troje malych chlopcow
OUT:     troje:      NUMBER
         malych:     SIZE
         chlopcow:   KIND
```

```
IN:      trzy duze druzyny
OUT:     trzy:       NUMBER
         duze:       SIZE
         druzyny:    KIND
```

```
IN:      jedna wielka niebieska owalna pilka
OUT:     jedna:      NUMBER
         wielka:     SIZE
         niebieska:  COLOR
         owalna:     SHAPE
         pilka:      KIND
```

```
IN:      siedem malych pilek
OUT:     siedem:     NUMBER
         malych:     SIZE
         pilek:      KIND
```

2.4 Analiza składniowa

2.4.1 Definicja

Analiza składniowa sprawdza czy wejście spełnia zasady gramatyki. W przypadku poprawności, może zwrócić ciąg podejmowanych akcji. Program, który wykonuje tę analizę zwany jest parser (np.: YACC).

2.4.2 Przykłady

```
IN:
jedna:SIZE owalna:SHAPE niebieska:COLOR wielka:SIZE pilka:KIND
```

```
OUT: Failure
```

```
IN:
jedna:NUMBER wielka:SIZE niebieska:COLOR owalna:SHAPE pilka:KIND
```

```
OUT: Success
      Action: Save into local database
      Action: Send data to server
```

3 PLY, czyli LEX oraz YACC w użyciu

3.1 Czym jest PLY?

PLY jest biblioteką dla języka Python która umożliwia budowę parserów i kompilatorów. PLY wzoruje się na znanych bibliotekach Lex oraz Yacc dla języka C - jednak jest zaimplementowana od zera w języku Python. Bibliotekę PLY dzielimy na dwie składowe LEX oraz YACC. LEX służy do analizy leksykalnej. Z pomocą LEX budujemy zestaw tokenów naszego języka oraz wyrażenia regularne. Dla wybranych tokenów uwzględniamy także dodatkowo logikę za pomocą funkcji. W przypadku YACC mamy do czynienia z analizą składniową. Z pomocą YACC budujemy zestawy reguł dla naszego języka, możemy określać ich kolejność, a także wzbogacać reguły o dodatkową logikę.

3.2 Przykładowa aplikacja

Klub piłkarski zamawia z hurtowni stroje sportowe do szatni klubowej. Zamówienie dotyczy podkoszulek, koszulek meczowych oraz spodenek w różnych rozmiarach, a także w dwóch wariantach kolorystycznych: niebieskim i czerwonym. Program ma za zadanie zebrać pełne zamówienie jako zestawienie różnych wariantów.

Na przykładzie tej aplikacji zostanie przedstawiony schemat budowania programu w PLY.

3.3 Budowanie aplikacji z użyciem PLY

3.3.1 Definicja zbioru tokenów

```
1 tokens = (  
2     'NUMBER', 'SIZE', 'COLOR', 'KIND', 'ACTION'  
3 )
```

3.3.2 Wyznaczenie zbioru wyrazów (leksykon)

```
1 t_ACTION = r'db|list'  
2  
3 def t_NUMBER(t):  
4     r'[\+]{0,1}[0-9]+[\.][0-9]+|[\+]{0,1}[0-9]+'  
5     print(t.type, t.value)  
6     return t  
7  
8 def t_SIZE(t):  
9     r'[SML]|XL'  
10    print(t.type, t.value)  
11    return t  
12  
13 def t_COLOR(t):  
14    r'blue|red'  
15    print(t.type, t.value)  
16    return t  
17  
18 def t_KIND(t):  
19    r'jerse(y|ys)|shorts|shir(t|ts)'  
20    print(t.type, t.value, '\n')  
21    return t  
22  
23 # Ignored characters  
24 t_ignore = " \t"  
25  
26 def t_newline(t):  
27    r'\n+'  
28    t.lexer.lineno += t.value.count("\n")  
29  
30 def t_error(t):  
31    t.lexer.skip(1)
```

3.3.3 Uruchomienie lex

```
1 import ply.lex as lex  
2 lexer = lex.lex()
```

3.3.4 Ustalenie kolejności zasad

```
1 def p_expression(p):  
2     '''expression : actionRule  
3                   | mainRule'''
```

3.3.5 Definicja zasad parsowania

```
1 def p_mainRule(p):  
2     '''mainRule : NUMBER SIZE COLOR KIND'''  
3     db.addElement(p)  
4
```

```

5 def p_actionRule(p):
6     '''actionRule : ACTION'''
7     if p[1] == "db":
8         db.showDatabase()
9     else:
10        db.showDatabaseHumanReadable()
11
12 def p_error(p):
13     print("_____ Failure")

```

3.3.6 Uruchomienie yacc

```

1 import ply.yacc as yacc
2 parser = yacc.yacc()

```

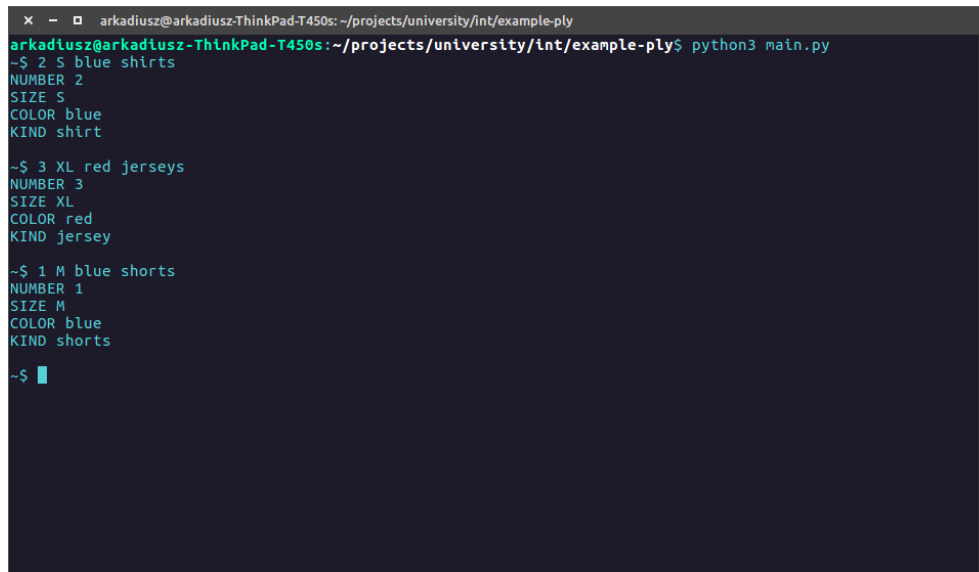
3.3.7 Start parsera

```

1 while True:
2     try:
3         s = input('~$ ') # Use raw_input on Python 2
4     except EOFError:
5         break
6     parser.parse(s)

```

4 Działanie przykładowej aplikacji



```

arkadiusz@arkadiusz-ThinkPad-T450s: ~/projects/university/int/example-ply
arkadiusz@arkadiusz-ThinkPad-T450s:~/projects/university/int/example-ply$ python3 main.py
~$ 2 S blue shirts
NUMBER 2
SIZE S
COLOR blue
KIND shirt

~$ 3 XL red jerseys
NUMBER 3
SIZE XL
COLOR red
KIND jersey

~$ 1 M blue shorts
NUMBER 1
SIZE M
COLOR blue
KIND shorts

~$ █

```

Rysunek 2: Wykonanie reguły mainRule

5 Podsumowanie

Podsumowując biblioteka PLY w dużej mierze ułatwia implementowanie kompilatorów oraz parserów. Dzięki niej możemy budować zaawansowane gramatyki, które bardzo dobrze sprawdzą się w zagadnieniach przetwarzania języka naturalnego.

Adres repozytorium git przedstawionej w referacie aplikacji:
`git@github.com:arkadiusz-wieczorek/example-ply.git`