DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

8 de Julio de 2015 — Teoria de Lenguajes

Grupo

Integrante	LU	Correo electrónico
Mancuso Emiliano	597/07	emiliano.mancuso@gmail.com
Mataloni Alejandro	706/07	amataloni@gmail.com
Serapio Noelia	871/03	noeliaserapio@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

0. Introducción

El objetivo del trabajo práctico es implementar un parser para un lenguaje orientado a la composición de piezas musicales, llamado Musileng, que luego será transformado al formato MIDI1 para su reproducción. Se deberá diseñar una gramática para Musileng e implementar a partir de ella el intérprete, utilizando algunas de las herramientas existentes para generar el analizador léxico y sintáctico. El objetivo final es, dada una pieza musical escrita en el lenguaje de entrada, poder escuchar el MIDI que ésta representa. Para la generación de los archivos MIDI finales se provee un programa que facilita su creación, cuya entrada consiste en un archivo de texto con determinado formato, que el parser deberá generar.

1. Grámatica Y Tokens

```
expression -> te co vars voices
te -> TEMPO FIGURE NUMBER
co -> COMPASS_V NUMBER DIV NUMBER
vars -> vars CONST NAME EQUAL cons_val SEMICOLON
vars ->

cons_val -> NUMBER
cons_val -> NAME
voices -> voice voices
voices ->

voice -> VOICE LPAREN cons_val RPAREN LCURLYBRACKET voice_content RCURLYBRACKET


voice_content -> compass_or_repeat voice_content
voice_content ->

compass_or_repeat -> COMPASS LCURLYBRACKET compass_content RCURLYBRACKET
compass_or_repeat -> REPEAT LPAREN cons_val RPAREN LCURLYBRACKET repeat_content RCURLYBRACKET
compass_or_repeat ->

repeat_content -> compass repeat_content
repeat_content ->

compass -> COMPASS LCURLYBRACKET compass_content RCURLYBRACKET
compass_content -> note compass_content
compass_content -> silence compass_content
compass_content ->

note -> NOTE LPAREN NOTE_ID COMMA cons_val COMMA figure_duration RPAREN SEMICOLON
silence -> SILENCE LPAREN figure_duration RPAREN SEMICOLON
figure_duration -> FIGURE
figure_duration -> DURATION
```

1.1. Tokens

Las expresiones regulares de los tokens se encuentran al final del informe, en el código del archivo *lexer_rules.py*.

2. Resolución

Según la documentación de **ply**, el *lexer* requiere los tokens en un orden específico. Decidimos definir funciones para determinar el orden de precedencia, que si bien puede ser un poco más tedioso de leer el código, nos acelero en el proceso de desarrollo.

Para las reglas de *parsing* decidimos definir una función por cada producción en lugar de usar la sintaxis del **or**. Intentamos usarla pero se prestaba a confusión y nos complicaba el desarrollo.

3. Conclusiones

Disfrutamos resolver este trabajo práctico más que el anterior, pues pudimos implementar lo aprendido durante la cursada y básicamente dimos los primeros pasos en la creación de nuestro propio lenguaje.

Si bien es un ejemplo didáctico y bastante divertido, el propósito final del parser es difícil de probar. Es decir, que al no saber de música poder disfrutar de un **midi** creado por nosotros es muy difícil obtener una salida apreciable (ej la canción de Mario, Tiburón u otras).

Tal vez apreciaríamos más un lenguaje de programación básico, donde tenemos que tomar más desiciones implementativas y definir nuestra propia grámatica y no una tan estructurada como esta.

4. Instrucciones

El programa principal se llama **musileng**, es un archivo binario y su modo de uso es el siguiente:

```
musileng [-h] [--ast AST] inputFile outputFile

Musileng - Compositor Musical.

required arguments:
  inputFile
  outputFile

optional arguments:
  -h, --help         show this help message and exit
  --ast AST, -a AST  Dump AST into a .dot file
```

4.1. Example

```
./musileng Ejemplos/ej1_input.txt output.txt --ast ast.out
AST dumped successfully
Syntax is valid.
```

5. Código

../src/ast.py

```
1  import lexer_rules
2  import parser_rules
3
4  from ply.lex import lex
5  from ply.yacc import yacc
6
7  class AST(object):
8    def __init__(self, input_file):
9      self._parse(input_file)
10
11   def get_tree(self):
12     return self.ast
13
14   # Dump the AST into a .dot file to see the tree as a digraph.
15   def dump_ast(self, output_file):
16     try:
17       edges = []
18       queue = [self.ast]
19       numbers = {self.ast: 1}
20       current_number = 2
21       node_str = 'node[width=1.5, height=1.5, shape="circle", label="%s"] n%d;\n'
22
23       f = open(output_file, 'w')
24       f.write("digraph {\n")
25
26       while len(queue) > 0:
27         node = queue.pop(0)
28         number = numbers[node]
29         f.write( node_str %(node.name(), number))
30
31         for child in node.children():
32           numbers[child] = current_number
33           edge = 'n%d -> n%d;\n' %(number, current_number)
34           edges.append(edge)
35           queue.append(child)
36           current_number += 1
37
38       f.write("".join(edges))
39       f.write("}")
40
41       f.close()
42     except IOError:
43       print "Error: can\'t find file or read data"
44     else:
45       print "AST dumped successfully"
46
47
48   # Reads input file and returns the parsed AST.
49   def _parse(self, input_file):
50     lexer = lex(module=lexer_rules)
51     parser = yacc(module=parser_rules)
52
53     try:
54       file = open(input_file, 'r')
```

```
55
56          self.ast = parser.parse(file.read(), lexer)
57
58          file.close()
59        except IOError:
60          print "Error: can\'t find file or read data"
```

../src/expressions.py

```
1  from helpers import figure_values
2
3  class Node(object):
4    def __init__(self, label, items, attrs = {}):
5      self.label = label
6      self.items = items
7      self.attributes = attrs
8
9    def name(self):
10     return str(self.label)
11
12   def _element(self, x):
13     if(isinstance(x, Element) or isinstance(x, Node)):
14       return x
15     else:
16       return Element(x)
17
18   def children(self):
19     return map(self._element, self.items)
20
21 class Initial(Node):
22   def __init__(self, items, attrs = {}):
23     Node.__init__(self, 'S', items, attrs)
24
25   def tempo(self):
26     return self.items[0]
27
28   def compass(self):
29     return self.items[1]
30
31   def voices(self):
32     v_array = []
33     voices_v = self.items[-1]
34
35     while isinstance(voices_v, Node):
36       v_array.append(voices_v.children()[0])
37       voices_v = voices_v.children()[-1]
38
39     return v_array
40
41   def constants(self):
42     return self.attributes['names']
43
44
45 class Tempo(Node):
46   def __init__(self, items, attrs = {}):
47     Node.__init__(self, 'te', items, attrs)
48
49   def microseconds(self):
50     f = figure_values(self.items[0])
51     n = self.items[1]
52
53     return 1000000*15*f/n
54
55
56 class DefCompass(Node):
57   def __init__(self, items, attrs = {}):
```

```
58      Node.__init__(self, 'co', items, attrs)
59      self.n = items[0]
60      self.d = items[1]
61
62    def figure_clicks(self, f):
63      if(f.endswith('.')):
64        mod = 1.5
65        f = f[0:-1]
66      else:
67        mod = 1
68
69      return 384 * self.d * mod / figure_values(f)
70
71
72  class Voice(Node):
73    def __init__(self, items, attrs = {}):
74      Node.__init__(self, 'voice', items, attrs)
75
76    def instrument(self, constants):
77      instr = self.items[0]
78
79      if(isinstance(instr, Constant)):
80        instr = constants[instr.name()]
81
82      if(isinstance(instr, Number)):
83        instr = instr.value
84
85      return instr
86
87    def compasses(self):
88      array = []
89      aux = self.items[1]
90
91      while isinstance(aux, Node):
92        child = aux.children()[0]
93        if isinstance(child, Repeat):
94          array = array + child.compasses()
95        else:
96          array.append(child)
97
98        aux = aux.children()[-1]
99
100     return array
101
102
103 class Compass(Node):
104   def __init__(self, items, attrs = {}):
105     Node.__init__(self, 'Compass', items, attrs)
106
107   def notes(self):
108     array = []
109     aux = self.items[0]
110
111     while isinstance(aux, Node):
112       array.append(aux.children()[0])
113       aux = aux.children()[-1]
114
115     return array
```

```
116
117
118 class Repeat(Node):
119   def __init__(self, items, attrs = {}):
120     Node.__init__(self, 'Repeat', items, attrs)
121     self.times = items[0].value
122
123   def compasses(self):
124     array = []
125     aux = self.items[1]
126
127     while isinstance(aux, Node):
128       array.append(aux.children()[0])
129       aux = aux.children()[-1]
130
131     return array * self.times
132
133
134 class Note(Node):
135   def __init__(self, items, attrs = {}):
136     Node.__init__(self, 'Note', items, attrs)
137
138     self.note = Note.translation_en(items[0])
139     self.octave = items[1]
140     self.duration = items[2]
141
142   def to_s(self):
143     return self.note + str(self.octave.value)
144
145   @staticmethod
146   def translation_en(to_translate):
147     translation = { 'do': 'c', 're': 'd', 'mi': 'e', 'fa': 'f',
148                     'sol': 'g', 'la': 'a', 'si': 'b' }
149     aux = to_translate[-1]
150
151     if( aux == '-' or aux == '+'):
152       to_translate = to_translate[:-1]
153     else:
154       aux = ''
155
156     return translation[to_translate] + aux
157
158
159 class Silence(Note):
160   def __init__(self, items, attrs):
161     Node.__init__(self, 'Silence', items, attrs)
162     self.duration = items[0]
163
164
165 class Element(object):
166   def __init__(self, value, attrs = {}):
167     self.value = value
168     self.attributes = attrs
169
170   def name(self):
171     return str(self.value)
172
173   def children(self):
```

```
174       return []
175
176
177   class Constant(Element):
178     def __init__(self, name, int_value):
179       self.var_name = name
180       self.value = int_value
181
182     def name(self):
183       return self.var_name
184
185
186   class Number(Element):
187     def name(self):
188       return "num: " + str(self.value)
```

../src/lexer_rules.py

```
1  tokens = [
2     'TEMPO',
3     'COMPASS_V',
4     'FIGURE',
5     'CONST',
6     'COMPASS',
7     'NOTE',
8     'SILENCE',
9     'REPEAT',
10    'DURATION',
11    'NOTE_ID',
12    'VOICE',
13    'NAME',
14    'DIV',
15    'COMMA',
16    'RCURLYBRACKET',
17    'LCURLYBRACKET',
18    'EQUAL',
19    'NUMBER',
20    'LPAREN',
21    'RPAREN',
22    'SEMICOLON'
23 ]
24
25
26 def t_COMMENT(t):
27     "//.*"
28 def t_TEMPO(t):
29     "\#tempo"
30     return t
31 def t_COMPASS_V(t):
32     "\#compas"
33     return t
34 def t_DURATION(t):
35     "(redonda|blanca|negra|corchea|semicorchea|fusa|semifusa)[\.]"
36     return t
37 def t_FIGURE(t):
38     "redonda|blanca|negra|corchea|semicorchea|fusa|semifusa"
39     return t
40 def t_CONST(t):
41     "const"
42     return t
43 def t_COMPASS(t):
44     "compas"
45     return t
46 def t_NOTE(t):
47     "nota"
48     return t
49 def t_SILENCE(t):
50     "silencio"
51     return t
52 def t_REPEAT(t):
53     "repetir"
54     return t
55 def t_NOTE_ID(t):
56     "(do|re|mi|fa|sol|la|si)[\+|\-]?"
57     return t
```

```
58  def t_VOICE(t):
59      "voz"
60      return t
61  def t_DIV(t):
62      "/"
63      return t
64  def t_COMMA(t):
65      ","
66      return t
67  def t_LCURLYBRACKET(t):
68      "{"
69      return t
70  def t_RCURLYBRACKET(t):
71      "}"
72      return t
73  def t_EQUAL(t):
74      "="
75      return t
76  def t_LPAREN(t):
77      "\("
78      return t
79  def t_RPAREN(t):
80      "\)"
81      return t
82  def t_SEMICOLON(t):
83      ";"
84      return t
85  def t_NUMBER(token):
86      r"[0-9][0-9]*"
87      token.value = int(token.value)
88      return token
89  def t_NAME(t):
90      "\w+"
91      return t
92
93
94  t_ignore = " \t"
95
96  def t_error(token):
97      message = "Token desconocido:"
98      message = "\ntype:" + token.type
99      message += "\nvalue:" + str(token.value)
100     message += "\nline:" + str(token.lineno)
101     message += "\nposition:" + str(token.lexpos)
102     raise Exception(message)
103
104 def t_NEWLINE(token):
105     r"\n+"
106     token.lexer.lineno += len(token.value)
```

../src/parser_rules.py

```
1  from __future__ import division
2  from lexer_rules import tokens
3  from expressions import *
4  from helpers import figure_values
5
6  class SemanticException(Exception):
7    pass
8
9  names = {}
10 util_vars = { 'voices': 0 }
11
12 def p_expression_initial(se):
13   'expression : te co vars voices'
14   se[0] = Initial(se[1:], {'names': names, 'util_vars': util_vars})
15
16 def p_expression_tempo(se):
17   'te : TEMPO FIGURE NUMBER'
18   se[0] = Tempo([se[2], se[3]])
19
20 def p_expression_compass_v(se):
21   'co : COMPASS_V NUMBER DIV NUMBER'
22   util_vars['compass'] = se[2] / se[4]
23   se[0] = DefCompass([se[2], se[4]])
24
25 def p_vars(se):
26   'vars : vars CONST NAME EQUAL cons_val SEMICOLON'
27   name = se[3]
28   cons_val = se[5]
29
30   if name in names:
31     raise SemanticException("const '" + name + "' is already defined")
32
33   if isinstance(cons_val, Number):
34     names[name] = cons_val.value
35   elif cons_val.name() in names:
36     names[name] = names[cons_val.name()]
37
38   se[0] = Node('vars', [se[1], se[3], se[5]])
39
40 def p_vars_empty(se):
41   'vars :'
42
43 def p_cons_val_number(se):
44   'cons_val : NUMBER'
45   se[0] = Number(se[1])
46
47 def p_cons_val_name(se):
48   'cons_val : NAME'
49
50   if se[1] in names:
51     se[0] = Constant(se[1], names[se[1]])
52   else:
53     raise SemanticException("const '" + se[1] + "' is undefined")
54
55 def p_expression_voices(s):
56   'voices : voice voices'
57   util_vars['voices'] = util_vars['voices'] + 1
```

```
58    s[0] = Node('voices', s[1:])
59
60  def p_expression_voices_empty(s):
61    'voices :'
62
63  def p_expression_voice(se):
64    'voice : VOICE LPAREN cons_val RPAREN LCURLYBRACKET voice_content RCURLYBRACKET'
65    cons_val = se[3]
66    if (cons_val.__class__ == Element and not(cons_val.value in names)):
67      raise SemanticException("const '" + cons_val.value + "' is undefined")
68
69    se[0] = Voice([se[3], se[6]])
70
71  def p_expression_voice_content(se):
72    'voice_content : compass_or_repeat voice_content'
73
74    if se[1].attributes['sum'] != util_vars['compass']:
75      error = 'compass not valid. '
76      error += 'Sum: ' + str(se[1].attributes['sum'])
77      error += " expected: " + str(util_vars['compass'])
78
79      raise SemanticException(error)
80
81    se[0] = Node('voice_content', se[1:], {'sum': se[1].attributes['sum']})
82
83  def p_expression_voice_content_empty(s):
84    'voice_content :'
85
86  def p_expression_compass(se):
87    'compass_or_repeat : COMPASS LCURLYBRACKET compass_content RCURLYBRACKET'
88    se[0] = Compass([se[3]], {'sum': se[3].attributes['sum']} )
89
90  def p_expression_compass_repeat(se):
91    'compass_or_repeat : REPEAT LPAREN cons_val RPAREN LCURLYBRACKET repeat_content RCUI
92    se[0] = Repeat([se[3], se[6]], {'sum': se[6].attributes['sum']})
93
94  def p_expression_repeat_content(se):
95    'repeat_content : compass repeat_content '
96    se[0] = Node('repeat_content', se[1:], {'sum': se[1].attributes['sum']} )
97
98  def p_expression_compass_only(se):
99    'compass : COMPASS LCURLYBRACKET compass_content RCURLYBRACKET'
100   se[0] = Compass([se[3]], {'sum': se[3].attributes['sum']} )
101
102 def p_expression_repeat_content_empty(se):
103   'repeat_content :'
104
105 def p_expression_compass_empty(se):
106   'compass_or_repeat :'
107
108 def p_expression_voice_compass_content_note(se):
109   'compass_content : note compass_content '
110
111   sum_aux = se[1].attributes['sum']
112   if(se[2] is not None):
113     sum_aux += se[2].attributes['sum']
114
115   se[0] = Node('compass_content', se[1:], {'sum': sum_aux })
```

```
116
117 def p_expression_compass_content_empty(s):
118   'compass_content :'
119
120 def p_expression_voice_compass_content_silence(se):
121   'compass_content : silence compass_content'
122
123   sum_aux = se[1].attributes['sum']
124   if(se[2] is not None):
125     sum_aux += se[2].attributes['sum']
126
127   se[0] = Node('compass_content', se[1:], {'sum': sum_aux})
128
129 def p_expression_note(se):
130   'note : NOTE LPAREN NOTE_ID COMMA cons_val COMMA figure_duration RPAREN SEMICOLON'
131   se[0] = Note([se[3], se[5], se[7]], {'sum': se[7].attributes['fig_val']})
132
133 def p_expression_silence(se):
134   'silence : SILENCE LPAREN figure_duration RPAREN SEMICOLON'
135   se[0] = Silence([se[3]], {'sum': se[3].attributes['fig_val']})
136
137 def p_expression_figure(se):
138   'figure_duration : FIGURE'
139   se[0] = Element(se[1], {'fig_val': 1 / figure_values(se[1])})
140
141 def p_expression_duration(se):
142   'figure_duration : DURATION'
143   se[0] = Element(se[1], {'fig_val': (1 / figure_values(se[1][0:-1])) * 1.5})
144
145 def p_error(subexpressions):
146   values = (subexpressions.lineno, subexpressions.type)
147   raise Exception("at line: %s, token: %s" % values)
```

../src/midicomp_exporter.py

```
1  from expressions import Silence
2
3  # Class to export our AST to Midicomp format.
4  class MidicompExporter(object):
5    def __init__(self, ast):
6      self.ast = ast.get_tree()
7      self.clicks_por_pulso = 384
8
9    def export(self, output_file):
10     try:
11       self.stream = open(output_file, 'w')
12     except IOError:
13       print 'cannot open', output_file
14     else:
15       try:
16         self._export_header()
17
18         self.channel = 0
19         self.constants = self.ast.constants()
20
21         # Recorrer las voces y crear una por una
22         for voice in self.ast.voices():
23           self._export_voice(voice)
24
25       except Exception as exception:
26         print 'something went wrong', exception
27       finally:
28         self.stream.close()
29
30
31   def _export_header(self):
32     ast = self.ast
33     values = (ast.attributes['util_vars']['voices'] +1)
34     self.stream.write("MFile 1 %d 384\n" % values)
35
36     self.stream.write("MTrk\n")
37     values = (ast.tempo().microseconds())
38     self.stream.write("000:00:000 Tempo %d\n" % values)
39     values = (ast.compass().n, ast.compass().d)
40     self.stream.write("000:00:000 TimeSig %d/%d 24 8\n" % values)
41     self.stream.write("000:00:000 Meta TrkEnd\n")
42     self.stream.write("TrkEnd\n")
43
44
45   def _export_voice(self, voice):
46     self.compass_counter = 0
47     self.channel = self.channel + 1
48     self.pulse = 0
49     self.click = 0
50
51     self.stream.write("MTrk\n")
52     values = self.channel
53     self.stream.write("000:00:000 Meta TrkName \"Voz %d\"\n" % values)
54     values = (self.channel, voice.instrument(self.constants))
55     self.stream.write("000:00:000 ProgCh ch=%d prog=%d\n" % values)
56
57     for compass in voice.compasses():
```

```
58          self._export_compass(compass)
59
60        values = (self.compass_counter, self.pulse, self.click)
61        self.stream.write("%03d:%02d:%03d Meta TrkEnd\n" % values)
62        self.stream.write("TrkEnd\n")
63
64
65      def _export_compass(self, compass):
66        self.pulse = 0
67        self.click = 0
68
69        for note in compass.notes():
70          if isinstance(note, Silence):
71            self._export_silence(note)
72          else:
73            self._export_note(note)
74
75
76      def _export_note(self, note):
77        str_aux = "%03d:%02d:%03d %s ch=%d note=%s  vol=%d\n"
78
79        values = (self.compass_counter, self.pulse, self.click,
80                  'On ', self.channel, note.to_s(), 70)
81        self.stream.write(str_aux % values)
82
83        self._increase_clicks(note)
84
85        values = (self.compass_counter, self.pulse, self.click,
86                  'Off', self.channel, note.to_s(), 0)
87        self.stream.write(str_aux % values)
88
89
90      def _export_silence(self, silence):
91        self._increase_clicks(silence)
92
93
94      def _increase_clicks(self, note_or_silence):
95        note_clicks = self.ast.compass().figure_clicks(note_or_silence.duration.value)
96
97        self.click += note_clicks
98
99        if (self.click >= 384):
100         self.pulse += self.click / 384
101         self.click = self.click % 384
102
103       if (self.pulse >= self.ast.compass().n):
104         self.pulse = 0
105         self.compass_counter += 1
```

../src/helpers.py

```
1  def figure_values(figure):
2    figure_values = {
3      'redonda': 1, 'blanca': 2, 'negra': 4, 'corchea': 8,
4      'semicorchea': 16, 'fusa': 32, 'semifusa': 64
5    }
6
7    return figure_values[figure]
```