Universidad de Buenos Aires
Facultad de
Ciencias Exactas y Naturales
Departamento de Computación

Teoría de Lenguajes

Primer Cuatrimestre de 2013

Trabajo práctico

Micro HTML Prettyprint

Integrante	LU	Correo electrónico
Cammi, Martín	676/02	martincammi@gmail.com
De Sousa Bispo, Mariano	389/08	marian_sabianaa@hotmail.com
Felisatti, Ana	335/10	anafelisatti@gmail.com

1. Introducción

En el presente trabajo práctico presentaremos un parser para la gramática de la primer parte. Además, detallaremos las modificaciones realizadas a esta y las desiciones tomadas para implementar el parser. Finalmente, presentaremos una serie de htmls válidos e inválidos.

El problema consiste en validar un texto html simple y a su vez obtener una copia visualizable desde un *browser*, con el código HTML indentado y con atributos visuales distintivos. Para ello utilizamos la herramienta ANTLR para Java, que nos permitió definir una grámatica con acciones atribuidas a cada producción, junto con los tokens léxicos necesarios.

2. Gramática

Nuestra gramática, modificada según las correcciones y modificaciones de implementación, es la siguiente:

$G = \langle V_N, V_T, P, \text{BEGIN} \rangle$ donde

V_N = { BEGIN, HTML, HEAD, LEER_HEAD, TITLE, SCRIPT, BODY, LEER_BODY }

V_T = { initHtml, endHtml, initHead, endHead, initTitle, endTitle, initScript, endScript, initBody, endBody, initDiv, endDiv, initH1, endH1, initP, endP, br, noTags }

y P está dada por:

BEGIN	$\longrightarrow$	initHtml HTML endHtml
HTML	$\longrightarrow$	HEAD BODY
HEAD	$\longrightarrow$	λ | initHead LEER_HEAD endHead
LEER_HEAD	$\longrightarrow$	TITLE LEER_HEAD |
		SCRIPT LEER_HEAD | λ
TITLE	$\longrightarrow$	initTitle noTags endTitle
SCRIPT	$\longrightarrow$	initScript noTags endScripts
BODY	$\longrightarrow$	λ | initBody LEER_BODY endBody
LEER_BODY	$\longrightarrow$	λ | noTags LEER_BODY | br LEER_BODY |
		initDiv LEER_BODY endDiv LEER_BODY |
		initP LEER_BODY endP LEER_BODY |
		initH1 LEER_BODY endH1 LEER_BODY

Para resolver el problema haremos una traducción dirigida por sintaxis incorporando acciones en nuestras producciones.

Al comenzar, imprimiremos el comienzo de la estructura HTML que permitirá la visualización en un *browser*, junto con el estilo que le dará colores a las distintas clases de tags. Estos estarán contenidos dentro de elementos `span` cuya clase fijará los colores. Para la indentación, los tags que la requieran estarán contenidos dentro de bloques `div`. Esto es basicamente lo sugerido por la cátedra.

Vale mencionar que nuestros tags ahora serán el texto de otro documento HTML, es decir, el resultado de nuestro programa.

3. Expresiones regulares

Nuestros tokens ahora son:

Terminal	***Expresión regular***
initHtml	<html>
endHtml	</html>
initHead	<head>
endHead	</head>
initBody	<body>
endBody	</body>
initTitle	<title>
endTitle	</title>
initScript	<script>
endScript	</script>
initDiv	
endDiv	
initH1	<h1>
endH1	</h1>
initP	
endP	
br	
noTags	(~(< | >))*

El token *noTags* ahora se representa con una expresión regular que identifica cualquier texto que no contenga símbolos menor o mayor, es decir, apertura y cierre de tags. No conseguimos implementarla como propusimos en un principio, es decir, que identificara cualquier texto excepto los tags de la grámatica porque las expresiones regulares en ANTLR son distintas de las de Java y no encontramos manera de adaptarlo.

Esto mismo vale para lo contenido entre los tokens *initScript* y *endScript* que inicialmente era cualquier texto sin estos tokens. Por ello ahora también *noTags* es lo esperado entre tales tokens.

4. Parser

Utilizando ANTLR, definimos nuestra gramática y los tokens léxicos para generar así el lexer y el parser. Su implementación se encuentra en el archivo `PrettyPrint.g`. Notar que además de los tokens previamente nombrados, tenemos los que reconocen espacios y comentarios, ocultándolos.

La clase `PrettyPrinter` engloba todas las funciones de impresión que se utilizan en las producciones, por ejemplo, impresión del comienzo de un bloque de indentación, su final o el comienzo de un `span` que aporte estilo.

Cada producción define lo que debe imprimirse, por ejemplo, sabemos que en

$$\text{TITLE} \longrightarrow \quad \text{initTitle noTags endTitle}$$

debemos imprimir la apertura del tag `title`, el texto en medio y su clasura.

Para imprimir el texto utilizamos el atributo `text` del token *noTags*. Todas estas operaciones tienen un efecto colateral sobre la clase `PrettyPrinter`, la cual es un Singleton que es accedida por el parser y el main.

Para manejar los errores, sobreescribimos el método `reportError` de ANTLR (tanto en el lexer como en el parser) para que levante una excepción y la ejecución se detenga. De ocurrir tal excepción, es atrapada por el main y se imprime un HTML de error con el mensaje de la misma.

5. HTMLs

A modo de prueba, generamos archivos HTML de entrada que pueden encontrarse en la carpeta `Tests` adjunta al informe, junto con los `.html` resultantes para cada uno. A continuación detallamos que se testeo en cada uno.

5.1. Válidos

- emptyhtml: vacío, es decir, sin head ni body.
- emptyheadbody: sin head, únicamente con body.
- noscripts: head con sólo title, body sin tags internos.
- severalscriptsemptybody: tres scripts con title en medio, sin body.
- nobody: head con title, sin body.
- nestedtags: head con solo scripts, body con varios tags anidados.
- hugebody: title, script, body con varios tags internos y anidamiento.
- comments: sin head, body, dos comentarios.
- input: ejemplo de la cátedra.

5.2. Inválidos

- lexererrorhtml: token inicial incorrecto, es rechazado por el lexer.
- lexererrormismatch: tag en title, es rechazado por el lexer por contener <.
- parsererrorunclosedbody: token de cierre de body faltante, es rechazado por el parser.
- parsererrormisplacedtag: token p en head, es rechazado por el parser.

6. Modo de Uso

El programa entregado puede ejecutarse en cualquier sistema operativo que soporte Java desde una terminal con el siguiente comando: `java jar tleng_tp2.jar`. Recibe por `STDIN` el HTML a procesar y devuelve por `STDOUT` el ya procesado.

7. Código fuente

7.1. Gramática

```
grammar PrettyPrint;

options {
  language = Java;
  output = AST;
}

@header {
  package com.tleng.grammar;
}

@lexer::header {
  package com.tleng.grammar;
}

@members {
  @Override
  public void reportError(RecognitionException e) {
      throw new IllegalArgumentException(e);
  }
}

begin : InitHTML {PrettyPrinter.GetInstance().Append("<html> &lt;html&gt;" +
                        " <head>" +
                         " <style type='text/css'>" +
                         " div.bloque {margin-left:2em;}" +
                         " span.tagHeadBody {color: blue;}" +
                         " span.tagTitleScript {color: orange;}" +
                         " span.Script {color: orange;}" +
                         " span.tagTexto {color:grey;}" +
                         " span.tagH1 {color: red;}" +
                         " span.tagBr {color: lightblue;}" +
                         " span.tagDiv{color: crimson;}" +
                         " span.tagP{color: purple;}" +
                         " </style>" +
                         " </head>");}
        html
      EndHTML {PrettyPrinter.GetInstance().Tag("/html");
              PrettyPrinter.GetInstance().Append("</html>");}
      ;

html : head body;

head :   InitHead {PrettyPrinter.GetInstance().StartBlock("tagHeadBody", "head");}
      leer_head
```

```
46        EndHead   {PrettyPrinter.GetInstance().CloseBlock("tagHeadBody", "/head" );}
47      | ;
48
49  body :    InitBody {PrettyPrinter.GetInstance().StartBlock("tagHeadBody", "body");
50                PrettyPrinter.GetInstance().StartBlock();}
51        leer_body
52        EndBody   {PrettyPrinter.GetInstance().CloseBlock();
53                PrettyPrinter.GetInstance().CloseBlock("tagHeadBody", "/body");}
54       | ;
55
56  leer_head :    script leer_head
57           | title leer_head
58           | ;
59
60  title : InitTitle        {PrettyPrinter.GetInstance().StartBlock("tagTitleScript",
        "title");}
61        content=NoTags     {PrettyPrinter.GetInstance().Text("tagTexto", $content.
            text);}
62        EndTitle        {PrettyPrinter.GetInstance().CloseBlock("tagTitleScript", "/
            title");}
63        ;
64
65  script : InitScript {PrettyPrinter.GetInstance().StartBlock("tagTitleScript", "
        script");
66                    PrettyPrinter.GetInstance().StartBlock();}
67         (content=NoTags |)
68                  {PrettyPrinter.GetInstance().Text("tagTexto", $content.text);}
69         EndScript   {PrettyPrinter.GetInstance().CloseBlock();
70                    PrettyPrinter.GetInstance().CloseBlock("tagTitleScript", "/script
                        ");}
71         ;
72
73  leer_body :    content=NoTags {PrettyPrinter.GetInstance().Text("tagTexto",
        $content.text);}
74             leer_body
75           | Br      {PrettyPrinter.GetInstance().LineSpan("tagBr", "br");}
76             leer_body
77           | InitDiv    {PrettyPrinter.GetInstance().LineSpan("tagDiv", "div");}
78             leer_body
79             EndDiv     {PrettyPrinter.GetInstance().LineSpan("tagDiv", "/div");}
80             leer_body
81           | InitP      {PrettyPrinter.GetInstance().LineSpan("tagP", "p");}
82             leer_body
83             EndP       {PrettyPrinter.GetInstance().LineSpan("tagP", "/p");}
84             leer_body
85           | InitH1     {PrettyPrinter.GetInstance().LineSpan("tagH1", "h1");}
86             leer_body
87             EndH1   {PrettyPrinter.GetInstance().LineSpan("tagH1", "/h1");}
88             leer_body
```

```
89            | ;
90
91 InitHTML : '<html>';
92 EndHTML: '</html>';
93 InitHead : '<head>';
94 EndHead : '</head>';
95 InitBody : '<body>';
96 EndBody : '</body>';
97 InitTitle : '<title>';
98 EndTitle : '</title>';
99 InitScript : '<script>';
100 EndScript : '</script>';
101 InitDiv : '<div>';
102 EndDiv : '</div>';
103 InitH1 : '<h1>';
104 EndH1 : '</h1>';
105 InitP : '<p>';
106 EndP : '</p>';
107 Br : '<br>';
108 NoTags : (~('<'|'>'))*;
109
110 WS
111     :   (' '              //Escapes spaces
112       | '\t'
113       | '\r'
114       | '\n'
115       | '<!--'(.*)'-->')    //Escapes comments
116       {$channel=HIDDEN;}
117     ;
```

7.2. Main

```
1  package com.tleng.grammar;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.UnsupportedEncodingException;
7
8  import org.antlr.runtime.ANTLRStringStream;
9  import org.antlr.runtime.CharStream;
10 import org.antlr.runtime.CommonTokenStream;
11 import org.antlr.runtime.RecognitionException;
12 import org.antlr.runtime.TokenStream;
13
14 public class TP2 {
15    public static void main(String[] args) throws RecognitionException,
          UnsupportedEncodingException, IOException
16    {
17       String input = "";
18        try
19        {
20            input = Read();
21        }
22        catch (IOException e)
23        {
24          System.out.println("Error reading input.\n" + e.getMessage());
25        }
26
27       CharStream stream = new ANTLRStringStream(input);
28       PrettyPrintLexer lexer = new PrettyPrintLexer(stream);
29       TokenStream tokenStream = new CommonTokenStream(lexer);
30       PrettyPrintParser parser = new PrettyPrintParser(tokenStream);
31
32       try
33       {
34          parser.begin();
35       }
36       catch(IllegalArgumentException exception)
37       {
38          PrettyPrinter.GetInstance().LogError(exception);
39       }
40       finally
41       {
42          PrettyPrinter.GetInstance().PrintParsedHTML();
43       }
44    }
45
46    private static String Read() throws IOException
```

```
47    {
48        StringBuilder sb = new StringBuilder();
49        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
50         String readString;
51
52         while ((readString = in.readLine()) != null && readString.length() != 0)
53           sb.append(readString);
54
55         return sb.toString();
56    }
57 }
```

7.3. Printer

```
1  package com.tleng.grammar;
2
3  public class PrettyPrinter {
4     public static final PrettyPrinter prettyPrinter = new PrettyPrinter();
5     private static final String CLOSE_DIV = "</div>";
6     private static final String BEGIN_DIV = "<div class=\"bloque\">";
7     private static final String BEGIN_SPAN = "<span class=\"";
8     private static final String END_SPAN = "</span>";;
9     private static final String LESS = "&lt;";
10    private static final String GREATER = "&gt;";
11    private StringBuilder _stream;
12    private boolean _existsParsingError;
13
14    private PrettyPrinter()
15    {
16       _stream = new StringBuilder();
17       _existsParsingError = false;
18    }
19
20    public static PrettyPrinter GetInstance()
21    {
22       return prettyPrinter;
23    }
24
25    public void PrintParsedHTML()
26    {
27       System.out.println(_stream.toString());
28    }
29
30    public void Append(String stream)
31    {
32       if(!_existsParsingError)
33          _stream.append(stream);
34    }
35
36    private String CreateLineSpan(String cssClass, String tag)
37    {
38       String lineSpan = "";
39       lineSpan += BEGIN_SPAN + cssClass + "\">" + LESS + tag + GREATER +END_SPAN;
40       return lineSpan;
41    }
42
43    public void Tag(String tag)
44    {
45       Append(LESS + tag + GREATER);
46    }
47
```

```
48      public void Text(String cssClass, String text)
49      {
50          if(text == null)
51              text ="";
52          String prettyText = BEGIN_SPAN + cssClass + "\">" + text + END_SPAN;
53          Append(prettyText);
54      }
55
56      public void LineSpan(String cssClass, String tag )
57      {
58          String span = CreateLineSpan(cssClass, tag);
59          Append(span);
60      }
61
62      public void StartBlock()
63      {
64          Append(BEGIN_DIV);
65      }
66
67      public void StartBlock(String cssClass, String tag)
68      {
69          String span = CreateLineSpan(cssClass, tag);
70          Append(BEGIN_DIV + span);
71      }
72
73      public void CloseBlock()
74      {
75          Append(CLOSE_DIV);
76      }
77
78      public void CloseBlock(String cssClass, String tag)
79      {
80          String span = CreateLineSpan(cssClass, tag);
81          Append(span + CLOSE_DIV);
82      }
83
84      public void LogError(IllegalArgumentException exception)
85      {
86          _stream = new StringBuilder("<html> &lt;html&gt;");
87          StartBlock();
88          Append("No se pudo parsear correctamente el input. A continuacion se detalla
                el error:<br>"+
89                exception.getMessage());
90          CloseBlock();
91          Append("&lt;html&gt; </html>");
92          _existsParsingError = true;
93      }
94  }
```

8. Conclusión

La utilización de gramáticas para el parsing de un lenguaje simplifica y abstrae la tarea. ANTLR es una herramienta muy poderosa que permite al desarrollador concentrarse en la aplicación de la gramática y no en los por menores de la implementación del parser. Si bien sólo hemos explorado una pequeña parte de lo que las gramáticas, expresiones regulares y esta herramienta permiten crear, es evidente su potencial y utilidad.