Teoria de Lenguajes

Primer Cuatrimestre de 2018

Conversor de JSON a YAML

Integrante	LU	Correo electrónico
Regnier, Ezequiel	836/13	eze_regnier@hotmail.com
Zamboni, Gianfranco	219/13	gianfranco375@gmail.com
Pesaresi, Natalia	636/14	natalia.pesaresi@gmail.com

Reservado para la ctedra

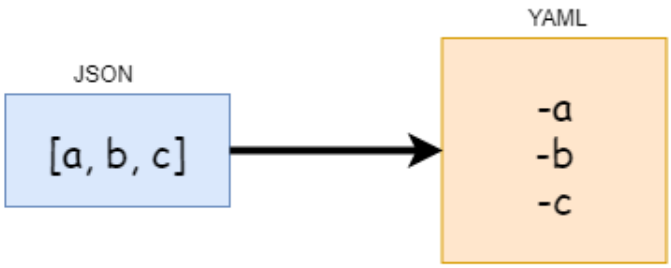
Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Contents

1 Introducción

En este trabajo, se propone realizar un conversor de código JSON a código YAML. Ambos lenguajes poseen las mismas estructuras, por lo que hay equivalencias sintácticas entre ellos.

Por ejemplo en el caso de los arrays:



Para realizar esta conversión, se necesita un Lexer que reconozca las cadenas que pertenecen al lenguaje JSON. Y un parser que traduzca sintácticamente la escritura a YAML.

2 Gramática

La siguiente gramática G, genera el lenguaje aceptado por el lenguaje JSON. G={ {Value, Object, Array, Members, Pair, Elements}, {string, number, true, false, null, [,], {, }, : }, Value, P }

Donde P:

Value -> Object | Array | string | number | true | false | null
Object -> {} | {Members}
Members -> Pair | Pair,Members
Pair -> string:Value
Array -> [] | [Elements]
Elements -> Value | Value,Elements

Una caraterística importante que se necesitaconocer, a la hora de hacer el parser, es su tipo. Se quizo analizar si la gramática es LALR(1). Para ello, se utoilizó una herramienta que genera para una grámatica su tabla LALR. [-1-]

LR table

State	ACTION													GOTO						
	{	}	,	string	:	[	]	number	Object	Array	true	false	null	$	S'	OBJECT	MEMBERS	PAIR	ARRAY	ELEMENTS	VALUE
0				s2				s3	s4	s5	s6	s7	s8								1
1														acc							
2														r_{10}							
3														r_{11}							
4														r_{12}							
5														r_{13}							
6														r_{14}							
7														r_{15}							
8														r_{16}							

Como la tabla no presenta conflictos, se concluye que la gramática es LALR(1),

3 Lexer

Para generar el Lexer, se utilizó la biblioteca de python Lex.

La estructura del lexer tiene las siguientes características:

- Definición de los tokens que posee la gramática
- Definición de las funciones t_NUMBER y t_STRING donde se especifica la forma de los números y cadenas de caracteres que acepta la gramática. Por ejemplo: '1.0' será un numero válido, pero '1.0ee' no.
- Definición de las funciones t_error y t_newline que son necesarias por la biblioteca para mostrar un mensaje de error cuando sea requerido e identificar un salto de línea respectivamente.

Por último, se define la instanciación del lexer de la siguiente manera: lexer = lex.lex().

4 Parser

Para generar el Parser, se utilizó la biblioteca de python Yacc. Éste toma el árbol de tokens credo por el Lexer.

Se crean tantas funciones como no terminales existan. Cada una de ellas acepta un único argumento 'p' que es un arreglo de elementos p asociados a los terminales y no teminales de la producción que tiene como parte derecha, que

corresponde a la función.
Por elemplo en la función p_object:

```
def p_object(p):
    '''object : LLAVEIZQ LLAVEDER'''
    ...
```

Aquí p = [p_0,p_1,p_2].

Un dato relevante es que 'p' tiene un sólo atributo 'value' que debe ser 'sintetizado'.

Durante la ejecución del parser se genera lo que llamamos 'Parsing Tree', esto es el árbol de objetos 'p' que se genera a partir de las sucesivas llamadas a función. Por ejemplo, al parsear: [a, b, c]. El 'Parsing Tree' será el árbol: p_Value → p_Object → [p_Members] →Value , p_Elements → Value , p_Elements → Value

En este trabajo, por las características del código YAML, era deseable contar con un atributo TABS 'heredado', y un atributo CADENA 'sintetizado'. Por lo expresado anteriormente, esto no era posible, por lo que se debió modificar la estrategia de diseño del parser.

Se procedió a crear la Clase TokkenWithAttributes:

```
classTokenWithAttributes:
    def__init__(self, tipo, yaml):
        self.tipo = tipo
        self.yaml = yaml
```

Por lo tanto ahora los elementos 'p', son del tipo TokkenWithAttributes. El atributo Tipo, es un string y el atributo 'Yaml' es una función que toma como parámetro los tabs correspondientes a la posición del objeto actual.

La estructura final del parser es:

- Se generó el 'Parsing Tree'. $parsedValue = p[1]$
- Se utilizó el atributo 'Yaml' del parsedValue, para imprimir la traducción.

5 Pruebas

Corrimos el parser para los archivos de prueba jasonObjet1.txt y jasonObjet2.txt con código JSON, y sus respectivas traducciones a código YAML fueron

res1.txt y res2.txt.

6 Requerimientos de Software

- Python 2.6
- PLY Version: 3.5

7 Comando de ejecución

- Por ejemplo: python3 tptleng.py jasonObjet1.txt

8 Referencias

[-1-] Página:

`http://jsmachines.sourceforge.net/machines/lalr1.html`

.