DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico SLS: Un simple lenguaje de scripting

Informe y análisis de resultados.

Teoría de Lenguajes

Grupo Altokemono

Integrante	LU	Correo electrónico
Lebedinsky, Alan	802/11	alanlebe@gmail.com
Podavini Rey, Martín Gastón	483/12	marto.rey2006@gmail.com
Valdes Castro, Tobías	800/12	tobini2@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

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

Introducción

Se desea incorporar un lenguaje de scripting, denominado Simple Lenguaje de Scripting (SLS), a un sistema de software ya existente. Para ello se requiere desarrollar un analizador léxico y sintáctico para este lenguaje.

Así, se recibirá como entrada un código fuente, el cual se deberá chequear por si cumple la sintaxis y restricciones de tipado del lenguaje, para, finalmente, formatear el código con la 'indentación' adecuada para SLS. En caso de haberse detectado algún error, se deberá informar claramente cuáles son las características del mismo.

1. Gramática

```
program
    : statement_list

statement_list
    : COMENTARIO
    | statement_list COMENTARIO
    | statement
    | statement_list statement
    ;

statement
    : expression_statement
    | selection_statement
    | iteration_statement
    | jump_statement
    ;

single_statement
    : comment_list statement
    | statement
    ;

comment_list
    : COMENTARIO comment_list
    | COMENTARIO
    ;

block
    : single_statement
    | bracketed_statement_list
    ;

bracketed_statement_list
    : '{' statement_list '}'
    ;

expression_statement
    : ';'
    | expression ';'
    ;

expression
    : assignment_expression
    | expression ',' assignment_expression
    ;

primary_expression
    : NOMBRE_VARIABLE
    | NUMERO
    | CADENA
    | TRUE
    | FALSE
    | '(' expression ')'
    | '[' vector_expression ']'
    | '{' reg_expression '}'
    | function
```

```
    ;

vector_expression
    : conditional_expression
    | conditional_expression ',' vector_expression
    ;

reg_expression
    : NOMBRE_VARIABLE ':' expression
    | NOMBRE_VARIABLE ':' expression, reg_expression
    ;

function
    : MULTIPLICACION_ESCALAR
    | CAPITALIZAR
    | COLINEALES
    | PRINT
    | LENGTH
    ;

postfix_expression
    : primary_expression
    | postfix_expression '[' expression ']'
    | function '(' ')'
    | function '(' argument_expression_list ')'
    | postfix_expression '.' NOMBRE_VARIABLE
    | postfix_expression '++'
    | postfix_expression '--'
    ;

argument_expression_list
    : assignment_expression
    | argument_expression_list ',' assignment_expression
    ;

unary_expression
    : postfix_expression
    | '++' unary_expression
    | '--'  unary_expression
    | unary_operator unary_expression
    ;

unary_operator
    : '+'
    | '-'
    | 'NOT'
    ;

multiplicative_expression
    : unary_expression
    | multiplicative_expression '*' unary_expression
    | multiplicative_expression '/' unary_expression
    | multiplicative_expression '^' unary_expression
    ;

congruence_expression
    : multiplicative_expression
    | congruence_expression '%' multiplicative_expression
```

```
additive_expression
    : congruence_expression
    | additive_expression '+' congruence_expression
    | additive_expression '-' congruence_expression
    ;

relational_expression
    : additive_expression
    | relational_expression '<' additive_expression
    | relational_expression '>' additive_expression
    ;

equality_expression
    : relational_expression
    | equality_expression '==' relational_expression
    | equality_expression '!=' relational_expression
    ;

logical_and_expression
    : equality_expression
    | logical_and_expression 'AND' equality_expression
    ;

logical_or_expression
    : logical_and_expression
    | logical_or_expression 'OR' logical_and_expression
    ;

conditional_expression
    : logical_or_expression
    | logical_or_expression '?' expression ':' conditional_expression
    ;

assignment_expression
    : conditional_expression
    | unary_expression '=' assignment_expression
    | unary_expression '*=' assignment_expression
    | unary_expression '/=' assignment_expression
    | unary_expression '+=' assignment_expression
    | unary_expression '-=' assignment_expression
    ;

selection_statement
    : IF '(' expression ')' block
    | IF '(' expression ')' block ELSE block
    ;

iteration_statement
    : WHILE '(' expression ')' block
    | DO block WHILE '(' expression ')' ';'
    | FOR '(' expression_statement expression_statement ')' block
    | FOR '(' expression_statement expression_statement expression ')' block
    ;

jump_statement
    : RETURN ';'
    | RETURN expression ';'
    ;
```

2. Notas sobre el trabajo

El código tiene varios comentarios sobre cómo hicimos el trabajo, las asunciones que tomamos, incluso TODOs con dudas o elementos faltantes, etc.

Por favor lean todos los comentarios, lo esencial a tener en cuenta sobre nuestro código está ahí mismo.

3. Resumen y conclusión

Pudimos programar esta gramática gracias a PLY, hacer una TDS para checkear condiciones particulares, y con esto logramos parsear con *prettyprint* y checkeo de tipado básico este lenguaje SLS pseudo C-Python.

En una nota aparte, la programación fue trabajosa y tediosa, no parece nada escalable y no es tan sencillo de leer y seguir. Creemos que es culpa de falta de práctica con esto, un poco de falta de orientación o simplemente porque PLY no es la mejor herramienta y tal vez usar Java con la otra librería era más limpio. De todas formas, los resultados se dieron bien, y el parser se pudo terminar.