Especificación Léxica del Lenguaje SKRN

Hugo Alonso Youzzueff Diaz Chavez

Juan José Huamaní Vásquez

Melvin Jarred Yabar Carazas

Gabriel Frank Krisna Zela Flores

Universidad La Salle

Escuela Profesional de Ingeniería de Software

Curso: Compiladores

Docente: Vicente Enrique Machaca Arceda

Compiladores - 13 de Mayo de 2025

1. Introducción

Este informe presenta el diseño y la implementación de un lenguaje de programación educativo inspirado en C, en el que se ha invertido el orden de las palabras reservadas para promover el entendimiento de los analizadores sintácticos LL(1). Se aborda el proceso completo, desde el análisis léxico hasta la generación de árboles sintácticos, con el objetivo de brindar una base sólida para estudiantes de compiladores.

2. Especificación Léxica

Tokens y Expresiones Regulares

- **ID**: [a-zA-Z_][a-zA-Z0-9_]*
- **INTEGER**: \d+
- **FLOATING**: \d+\.\d+([eE][+-]?\d+)?
- **STRING**: ".*?"
- **CHARACTER**: '.'
- **Operadores**: =, +, -, *, /,
- **Delimitadores**: (,), {, }, [,], ;, ,
- **Palabras reservadas**: tni, taolf, diov, fi, esle, elihw, od, rof, nruter, tnirp, entre otras.

3. Gramática del Lenguaje (completa)

```
Program    -> Stmt Program |
Stmt       -> FuncDecl | ExprStmt | PrintStmt | ForStmt | IfStmt
            | WhileStmt | DoWhileStmt | VarDecl | ReturnStmt
Block       -> { Program }
DoWhileStmt -> od Block elihw ( E )
WhileStmt   -> elihw ( E ) Block
ForStmt     -> rof ( OptExpr ; OptExpr ; OptExpr ) Block
OptExpr     -> E |
IfStmt      -> fi ( E ) Block IfStmtail
IfStmtail   -> esle Block |
FuncDecl    -> fed id ( Params ) { Program }
Params      -> Param ParamsTail |
ParamsTail  -> , Param ParamsTail |
Param       -> id
ExprStmt    -> E
PrintStmt   -> tnirp( Args )
Args        -> E ArgsTail | Type E |
ArgsTail    -> , E ArgsTail |
VarDecl     -> Type E
```

```
Type        -> tni | taolf | diov
ReturnStmt  -> nruter E
E           -> T E'
E'          -> + T E' | - T E' |
T           -> G T'
T'          -> * G T' | / G T' |
G           -> F G'
G'          -> >= F G' | % F G' | < F G' | <= F G' | > F G'
             | = F G' | += F G' | -= F G' | == F G' | != F G' | && F G' |
F           -> ( E ) | id | " id " | id( Args ) B | num | eurt | eslaf
B           ->  | Block
```

Anexo: Fragmento de la Tabla LL(1)

A continuación, se muestra un extracto representativo de la tabla LL(1) generada automáticamente a partir de la gramática:

No Terminal	`tni`	`id(`	`fed`	`diov`	`od`
Program	Stmt Program	Stmt Program	Stmt Program	Stmt Program	Stmt Program
Stmt	VarDecl	ExprStmt	FuncDecl	VarDecl	DoWhileStmt
FuncDecl			fed id (Params) { Program }		
DoWhileStmt					od Block elihw (E)
VarDecl	Type E			Type E	

Cuadro 1: Fragmento de la Tabla LL(1) para símbolos iniciales clave

4. Implementación del Analizador Sintáctico

4.1 Análisis Léxico

El módulo `lexer.py` fue desarrollado usando `ply.lex`. Se encarga de procesar el archivo fuente `entrada_lex.txt` y generar la secuencia de tokens.

4.2 Tabla Sintáctica LL(1)

`tabla.py` genera la tabla `tabla.csv` a partir de una definición textual de la gramática.

4.3 Análisis Sintáctico Predictivo

`parser.py` contiene la lógica del algoritmo LL(1), construye el árbol sintáctico y lo exporta en formato Graphviz:

```
input_tokens = ['tni', 'id(', ')', '{', 'tnirp(', 'id', ')', 'nruter', 'num', '}']
root = predictive_parser(input_tokens, csv_file='producciones.csv')
generar_arbol_graphviz(root)
```

5. Ejemplos de Código

5.1 Ejemplo 1: Hola Mundo

```
tni niam() {
    tnirp("Hola, Mundo!")
    nruter 0
}
```

5.2 Ejemplo 2: Bucles Anidados

```
tni niam() {
    tni i = 0
    elihw (i < 5) {
        tni j = 0
        elihw (j < 5) {
            tnirp(i, " ", j)
            j = j + 1
        }
        i = i + 1
    }
    nruter 0
}
```

5.3 Ejemplo 3: Recursividad - Factorial

```
diov factorial(tni n) {
    fi (n == 0) {
        nruter 1
    }
    nruter n * factorial(n - 1)
}

tni niam() {
    tni resultado = factorial(5)
    tnirp(resultado)
    nruter 0
}
```

6. Conclusiones

Se logró implementar exitosamente un analizador sintáctico LL(1) completo, desde la etapa léxica hasta la generación de árboles, acompañado de visualización en Graphviz.