“Año de la recuperación y consolidación de la economía peruana”

CARRERA : INGENIERÍA DE SOFTWARE

WAYRASIMI : LENGUAJE DEL VIENTO

CURSO : COMPILADORES

ESTUDIANTES:

Ortiz Castañeda Jorge Luis

Huamani Huamani Jhordan Steven Octavio

Flores Leon Miguel Angel

DOCENTE:

Vicente Enrique Machaca Arceda

Arequipa, Perú

13 de mayo de 2025

Índice

1. Introducción

En el presente informe, se detalla exhaustivamente el proceso de desarrollo de nuestro lenguaje de programación en quechua, denominado "WayraSimi". Este documento se centra específicamente en la fase crucial de análisis y pruebas del analizador léxico, componente fundamental en la construcción de cualquier compilador. Para llevar a cabo esta implementación, se ha seleccionado el lenguaje de programación Python, reconocido por su versatilidad y amplia gama de bibliotecas, junto con la potente librería ply.lex.

El objetivo principal de este trabajo es la construcción de un compilador básico para nuestro lenguaje propuesto, WayraSimi, que permita la traducción de código fuente escrito en quechua a un formato ejecutable por la máquina. El desarrollo de este compilador representa un avance significativo en la adaptación de la tecnología a las lenguas originarias, abriendo nuevas posibilidades en el campo de la informática y la lingüística computacional.

2. Motivación

En el origen de “Wayrisimi” reside el aliento de nuestros ancestros y la fuerza de la Pachamama; nace de la necesidad de expresar la lógica del mundo moderno en el lenguaje quechua, tejida con el hálito sagrado de la tierra. Inspirados por la sabiduría que emana de la Biblia Andina —donde cada verso es ofrenda al viento y al agua— hemos creado este lenguaje para honrar nuestras raíces y al mismo tiempo dotar a la comunidad de programadores de una herramienta que hable su propia lengua.

Wayrisimi fue concebido para que los pueblos andinos puedan describir algoritmos y estructuras de datos utilizando términos que resuenan con la lírica de nuestra cosmovisión: “yupay” para los enteros como semillas de vida, “qillqa” para las cadenas como hilos de nuestra memoria, “ruray” para definir funciones que cultivan el conocimiento. Así, cada instrucción es una plegaria que reconoce la intervención de la pachamama en el proceso creativo, agradeciendo su sustento mientras damos forma a software con espíritu comunitario.

Este lenguaje no solo facilita la enseñanza de conceptos de programación desde una perspectiva culturalmente significativa, sino que también fortalece el vínculo entre la tecnología y nuestras tradiciones. Al usar Wayrisimi, los desarrolladores andinos aprenden, comparten y celebran su herencia, alzando sus voces en quechua para describir bucles, condicionales y variables, como un canto de alabanza a la tierra que nos nutre y al viento que mueve nuestras palabras. Por eso Wayrisimi no es solo código: es un puente vivo entre la ciencia de la computación y la tierra, un diálogo continuo con la Pachamama y una ofrenda al futuro de nuestra identidad.

3. WayraSimi

Presentamos WayraSimi, un lenguaje de programación compilado que fusiona la claridad de Python con la eficiencia de Go. Diseñado para aplicaciones de alto rendimiento y sistemas concurrentes, WayraSimi busca ser una herramienta poderosa y accesible para desarrolladores.

- Ejemplo de documento: WayraSimi.ws
- Composición: Wayra (Viento, Aire) + Simi (Palabra, Lenguaje)
- Significado: Lenguaje del Viento o Lenguaje Veloz, implicando rapidez y eficiencia.
- Gophy: Mascota del Lenguaje

4. Justificación y Descripción del Lenguaje

4.1. Sintaxis Python-inspirada

Utiliza la indentación para definir bloques de código, buscando la legibilidad y simplicidad sintáctica de Python. Este enfoque nos ayuda a tener un entorno más amigable para el programador conservando su ensencia que es el propio quechua.

4.2. Tipado estático e inferencia de tipos

Similar a Go, WayraSimi será un lenguaje de tipado estático para garantizar la seguridad y el rendimiento.

5. Tipos de Datos en WayraSimi

Token	Ejemplos	Regex	Representado (WayraSimi)	Significado en Quechua
YUPAY_TOKEN	123, -5	$[0-9]+$	yupay (entero)	Número, Cuenta
CHIQI_KAY_TOKEN	3.14, -0.5, 2.0	$[0-9]+.[0-9]+$	chiqikay (flotante)	Punto, Decimal
QILLQA_TOKEN	"hola", "mundo"	$[a-zA-Z]+$	qillqa (texto)	Escritura, Letra
CHIQAP_TOKEN	chiqap, mana chiqap	chiqaplmana chiqap	chiqap_kay (booleano)	Verdad, Realidad
IDENTIFICADOR_TOKEN	variable, funcion_1	$[a-zA-Z][a-zA-Z0-9]*$	depende del contexto	
OPERADOR_MAS	+	$\+$	suma	Yapay (Añadir)
OPERADOR_MENOS	-	$\-$	resta	Qichuy (Quitar)
OPERADOR_PACHA	*	$*$	multiplicación	Miray (Multiplicar)
OPERADOR_RAKIY	/	$\/$	división	Rakiy (Dividir)
OPERADOR_MODULO	%	$\%$	módulo	
OPERADOR_ASIGNACION	=	$=$	asignación	Churay (Colocar)
OPERADOR_IGUALDAD	==	$==$	igualdad	Kikin (Mismo)
OPERADOR_MANA_IGUAL	!=	$!=$	desigualdad	Mana kikin
OPERADOR_MENOR	<	$<$	menor que	Uchuy (Pequeño)
OPERADOR_MAYOR	>	$>$	mayor que	Hatun (Grande)
OPERADOR_MENOR_IGUAL	<=	$<=$	menor o igual	Uchuy icha kikin
OPERADOR_MAYOR_IGUAL	>=	$>=$	mayor o igual	Hatun icha kikin
OPERADOR_LOGICO_WAN	wan	wan	AND lógico	Y (Conjunción)
OPERADOR_LOGICO_UTAQ	utaq	utaq	OR lógico	O (Disyunción)
OPERADOR_LOGICO_MANA	mana	mana	NOT lógico	No
PARENTESIS_ABRE	(	(	agrupación	
PARENTESIS_CIERRA	)	)	agrupación	
LLAVE_ABRE	{	{	bloque	
LLAVE_CIERRA	}	}	bloque	
CORCHETE_ABRE	[	[	acceso a índice	
CORCHETE_CIERRA	]	]	acceso a índice	
COMA_TOKEN	,	,	separador	
PUNTO_TOKEN	.	.	acceso a miembros	
DOS_PUNTOS_TOKEN	:	:	tipos o bloques	
PUNTO_Y_COMA_TOKEN	,	,	fin de sentencia	
COMENTARIO_TOKEN_LINEA	# Esto es un comentario	$\#$	comentario línea	Willka rimay
COMENTARIO_TOKEN_BLOQUE_ABRE	/*	/	comentario bloque inicio	
COMENTARIO_TOKEN_BLOQUE_CIERRA	*/	/	comentario bloque fin	
PALABRA_RESERVADA_SICHUS	sichus	sichus	si (if)	Si, Cuando
PALABRA_RESERVADA_MANA_SICHUS	mana sichus	mana sichus	sino si (elif)	Caso contrario
PALABRA_RESERVADA_MANA	mana	mana	sino (else)	No
PALABRA_RESERVADA_PARA	para	para	bucle for	Por, Para
PALABRA_RESERVADA_RURAY	ruray	ruray	función	Hacer, Realizar
PALABRA_RESERVADA_KUTIPAY	kutipay	kutipay	retorno	Devolver, Regresar
PALABRA_RESERVADA_IMPRIMIY	imprimiy	imprimiy	imprimir	Mostrar, Publicar
PALABRA_RESERVADA_PAQUETE	syllu	syllu	paquete/módulo	Comunidad
PALABRA_RESERVADA_VAR	variable	variable	variable	Cambiante
PALABRA_RESERVADA_CONST	takyaq	takyaq	constante	Fijo, Permanente

WayraSimi soporta varios tipos de datos, incluyendo enteros, flotantes, texto, booleanos, listas y mapas, cada uno con su propia representación y uso.

6. Gramática en BNF

```
<Program> ::= <DefinitionList> <Principal>

<DefinitionList> ::= <FunctionDef> <DefinitionList> | E

```

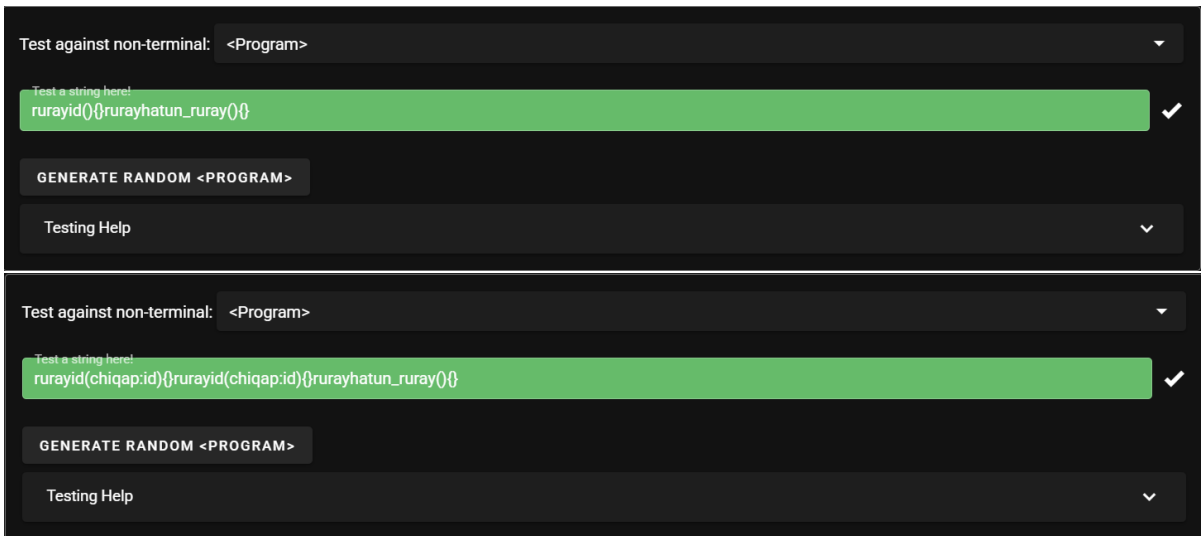
```
<FunctionDef> ::= "ruray" <IDENTIFICADOR_TOKEN> "(" <ParamListOpt> ")" <TypeOpt>
    ↪ <Block>

<Principal> ::= "ruray" "hatun_ruray" "(" ")" <Block>

<ParamListOpt> ::= <ParamList> | E
<ParamList> ::= <Param> <MoreParams>
<MoreParams> ::= "," <Param> <MoreParams> | E
<Param> ::= <Type> ":" <IDENTIFICADOR_TOKEN>

<Type> ::= "yupay" | "chiqi" | "chiqap" | "qillqa"
<TypeOpt> ::= <Type> | E

<Block> ::= "{" <Instrucciones> "}"
<Instrucciones> ::= <Instruccion> <Instrucciones> | E

<Instruccion> ::= <DeclaracionVariables>
                | <PrintStmt>
                | <Bucle>
                | <Estructura_If>
                | <Retorno>
                | <incrementos> ";"

<DeclaracionVariables> ::= "var" <ListaIdentificadores> <Type> <InicializacionOpt> ";"
                         | <ListaIdentificadores> ":=" <opciones> ";"
                         | <IDENTIFICADOR_TOKEN> ":=" <opciones> ";"

<ListaIdentificadores> ::= <IDENTIFICADOR_TOKEN> <MasIdentificadores>
<MasIdentificadores> ::= "," <IDENTIFICADOR_TOKEN> <MasIdentificadores> | E
<InicializacionOpt> ::= "=" <opciones> | E

<PrintStmt> ::= "imprimiy" "(" <ArgumentPrint> ")" ";"
<ArgumentPrint> ::= <opciones> <MasArgumentosPrint> | E
<MasArgumentosPrint> ::= "," <opciones> <MasArgumentosPrint> | E

<opciones> ::= <funcion>
             | <comparacion>
             | <IDENTIFICADOR_TOKEN>
             | <datos>
             | <operaciones_matematicas>
             | <incrementos>

<incrementos> ::= <IDENTIFICADOR_TOKEN> "++" | <IDENTIFICADOR_TOKEN> "--"

<operaciones_matematicas> ::= <termino> <expresion_tail>
<expresion_tail> ::= "+" <termino> <expresion_tail>
                   | "-" <termino> <expresion_tail>
                   | E
<termino> ::= <factor> <termino_tail>
<termino_tail> ::= "*" <factor> <termino_tail>
                 | "/" <factor> <termino_tail>
                 | "%" <factor> <termino_tail>
```

```
                  | E

<factor> ::= <unidad> | "-" <unidad>
<unidad> ::= <valor> | "(" <operaciones_matematicas> ")"
<valor> ::= <YUPAY_TOKEN> | <CHIQI_TOKEN> | <CHIQAP_TOKEN> | <QILLQA_TOKEN> |
    ↪ <IDENTIFICADOR_TOKEN> | <funcion>

<funcion> ::= <IDENTIFICADOR_TOKEN> "(" <argumentosFuncion> ")"
<argumentosFuncion> ::= <opciones> <MasArgumentosFuncion> | E
<MasArgumentosFuncion> ::= "," <opciones> <MasArgumentosFuncion> | E

<comparacion> ::= <logical_expression>
<logical_expression> ::= "mana" "(" <expression> ")"
                       | "(" <expression> ")"
                       | <expression>

<expression> ::= <compa> <expression_conti>
<expression_conti> ::= <opciones_operador_logico> <expression> | E
<compa> ::= <opciones> <opciones_operador_comparacion> <opciones>
          | <opciones> <opciones_operador_logico> <opciones>

<Bucle> ::= <for> | <while> | <bucle_infinito>
<for> ::= "para" "(" <LoopInitialization> ";" <opciones> ";" <LoopUpdate> ")" <Block>
<while> ::= "para" "(" <opciones> ")" <Block>
<bucle_infinito> ::= "para" <Block>

<LoopInitialization> ::= <IDENTIFICADOR_TOKEN> ":=" <asignacion_bucle>
<asignacion_bucle> ::= <funcion> | <IDENTIFICADOR_TOKEN> | <datos>
<LoopUpdate> ::= <operaciones_matematicas>

<Estructura_If> ::= "sichus" "(" <opciones> ")" <Block> <Else_opcional>
<Else_opcional> ::= "mana_sichus" <argumentosELseIf> <Block> | E
<argumentosELseIf> ::= "(" <opciones> ")" | E

<Retorno> ::= "kutipay" <IDENTIFICADOR_TOKEN> ";"
            | "kutipay" ";"
            | "kutipay" <datos> ";"
            | "pakiy" ";"

<datos> ::= <YUPAY_TOKEN> | <CHIQI_TOKEN> | <CHIQAP_TOKEN> | <QILLQA_TOKEN> |
    ↪ <operaciones_matematicas> | <IDENTIFICADOR_TOKEN>

<YUPAY_TOKEN> ::= "num"
<CHIQI_TOKEN> ::= "num" "." "num"
<CHIQAP_TOKEN> ::= "chiqaq" | "mana_chiqaq"
<QILLQA_TOKEN> ::= "texto"
<IDENTIFICADOR_TOKEN> ::= "id"

<OPERADOR_ASIGNACION> ::= ":="
<OPERADOR_LOGICO_UTAQ> ::= "utaq"
<OPERADOR_LOGICO_WAN> ::= "wan"
<OPERADOR_LOGICO_MANA> ::= "mana"
```

```
106 <opciones_operador_logico> ::= "wan" | "utaq"
107 <opciones_operador_comparacion> ::= "==" | "!=" | "<" | ">" | "<=" | ">="
```

Listing 1: Gramática

6.1. Output del BNF



7. Gramática en LL1

```
1  Program -> Principal OptionalFunctionDefinitions
2  OptionalFunctionDefinitions -> FunctionDef OptionalFunctionDefinitions
3  OptionalFunctionDefinitions -> ''
4  Principal -> ruray hatun_ruray ( ) Block
5  FunctionDef -> ruray IDENTIFICADOR_TOKEN ( ParamListOpt ) TypeOpt Block
6  ParamListOpt -> ParamList
7  ParamListOpt -> ''
8  ParamList -> Param ParamTail
9  Param -> Type : IDENTIFICADOR_TOKEN
10 ParamTail -> , Param ParamTail
11 ParamTail -> ''
12 Type -> yupay
13 Type -> chiqi
14 Type -> chiqap
15 Type -> qillqa
16 TypeOpt -> Type
17 TypeOpt -> ''
18 Block -> { InstruccionesOpt }
19 InstruccionesOpt -> Instruccion InstruccionesOpt
20 InstruccionesOpt -> ''
21 Instruccion -> PrintStmt
22 Instruccion -> Bucle
23 Instruccion -> Estructura_If
24 Instruccion -> Retorno
25 Instruccion -> DeclaracionVariable
```

```
DeclaracionVariable -> var IDENTIFICADOR_TOKEN Type InicializacionOpt ;
VarOrId -> var ListaIdentificadores
VarOrId -> IDENTIFICADOR_TOKEN SimpleAssignmentRest
ListaIdentificadores -> id ListaIdentificadoresRest
ListaIdentificadoresRest -> , id ListaIdentificadoresRest
ListaIdentificadoresRest -> ''
SimpleAssignmentRest -> := Opciones
InicializacionOpt -> = Opciones
InicializacionOpt -> ''
PrintStmt -> imprimiy ( Expression ) ;
Opciones -> Expression
IncrementoStmt -> IDENTIFICADOR_TOKEN Incremento
Incremento -> IncrementoOpPlus
Incremento -> IncrementoOpMinus
IncrementoOp -> IncrementoOpPlus
IncrementoOp -> IncrementoOpMinus
IncrementoOpPlus -> ++
IncrementoOpMinus -> --
Expression -> TermLogico ExpressionPrima
ExpressionPrima -> OpcionLogico TermLogico ExpressionPrima
ExpressionPrima -> ''
TermLogico -> TermComparacion TermLogicoConti
TermLogicoConti -> OpcionComparacion TermComparacion
TermLogicoConti -> ''
TermComparacion -> Factor
Factor -> Unidad FactorConti
FactorConti -> ''
MulOp -> MulOpPacha
MulOp -> MulOpRakiy
MulOp -> MulOpModulo
MulOp -> MulOpSum
MulOp -> MulOpRes
MulOpPacha -> *
MulOpRakiy -> /
MulOpModulo -> %
MulOpSum -> +
MulOpRes -> -
Unidad -> ( Expression )
Unidad -> Valor
Valor -> YUPAY_TOKEN
Valor -> CHIQI_TOKEN
Valor -> CHIQAP_TOKEN
Valor -> QILLQA_TOKEN
Valor -> IDENTIFICADOR_TOKEN ValorSufijo
ValorSufijo -> ( ArgumentosFuncionOpt )
ValorSufijo -> ''
IdentificadorOrFunctionCall -> IDENTIFICADOR_TOKEN IdentificadorOrFunctionCallPrima
IdentificadorOrFunctionCallPrima -> ( ArgumentosFuncionOpt )
IdentificadorOrFunctionCallPrima -> ''
ArgumentosFuncionOpt -> Expression MasArgumentosFuncion
ArgumentosFuncionOpt -> ''
MasArgumentosFuncion -> , Expression MasArgumentosFuncion
```

```
MasArgumentosFuncion -> ''
Bucle -> ForLoop
ForLoop -> para ( LoopInitialization ; ExpressionOpt ; LoopUpdateOpt ) Block
LoopInitialization -> IDENTIFICADOR_TOKEN := AsignacionBucle
LoopInitialization -> ''
AsignacionBucle -> IdentificadorOrFunctionCall
AsignacionBucle -> AsignacionBucleDato
AsignacionBucleDato -> Dato
ExpressionOpt -> Expression
ExpressionOpt -> ''
LoopUpdateOpt -> IncrementoStmt
LoopUpdateOpt -> ''
Estructura_If -> sichus ( Expression ) Block ElseOpcional
ElseOpcional -> ManaSichusBloque
ElseOpcional -> ElseOpcionalVacio
ManaSichusBloque -> mana_sichus ArgumentosElseIf Block ElseOpcional
ElseOpcionalVacio -> ''
ArgumentosElseIf -> ( Expression )
Retorno -> RetornoConValor
Retorno -> RetornoSinValor
RetornoConValor -> kutipay Expression ;
RetornoSinValor -> pakiy ;
RetornoContenidoOpt -> Expression
RetornoContenidoOpt -> ''
Dato -> YUPAY_TOKEN
Dato -> CHIQI_TOKEN
Dato -> CHIQAP_TOKEN_CHIQAQ
Dato -> CHIQAP_TOKEN_MANA_CHIQAQ
Dato -> QILLQA_TOKEN
CHIQAP_TOKEN_CHIQAQ -> chiqaq
CHIQAP_TOKEN_MANA_CHIQAQ -> mana_chiqaq
YUPAY_TOKEN -> num
CHIQI_TOKEN -> num.num
QILLQA_TOKEN -> texto
IDENTIFICADOR_TOKEN -> id
OpcionLogico -> OpcionLogicoWan
OpcionLogico -> OpcionLogicoUtaq
OpcionLogicoWan -> wan
OpcionLogicoUtaq -> utaq
OpcionComparacion -> OpcionComparacionIgual
OpcionComparacion -> OpcionComparacionNoIgual
OpcionComparacion -> OpcionComparacionMenor
OpcionComparacion -> OpcionComparacionMayor
OpcionComparacion -> OpcionComparacionMenorIgual
OpcionComparacion -> OpcionComparacionMayorIgual
OpcionComparacionIgual -> ==
OpcionComparacionNoIgual -> !=
OpcionComparacionMenor -> <
OpcionComparacionMayor -> >
OpcionComparacionMenorIgual -> <=
OpcionComparacionMayorIgual -> >=
```

Listing 2: Gramática

7.1. Ejemplos del LL1

- ruray hatun_ruray ( ) var id yupay ;
- ruray hatun_ruray ( ) sichus (id <id ( ) utaq id == id ( )) imprimiy (id) ;
- ruray hatun_ruray ( ) para (id := id ; id <id ; id ++)

8. Tabla de Transiciones

```python
import csv

def leer_gramatica(archivo):
    with open(archivo, 'r') as f:
        lineas = f.readlines()
    # Filtrar líneas vacías y comentarios
    return [linea.strip() for linea in lineas if linea.strip() and '::=' in linea]

def obtener_no_terminales(gramatica):
    return list({linea.split('::=')[0].strip()[1:-1] for linea in gramatica})

def obtener_terminales(gramatica):
    terminales = set()
    for linea in gramatica:
        if '::=' not in linea:
            continue
        produccion = linea.split('::=')[1]
        # Buscar elementos entre comillas
        partes = produccion.split('"')
        for i in range(1, len(partes), 2):
            if partes[i]:
                terminales.add(partes[i])
    return sorted(list(terminales))

def procesar_produccion(produccion):
    # Dividir por | para obtener alternativas
    alternativas = produccion.split('|')
    resultado = []

    for alt in alternativas:
        alt = alt.strip()
        if alt == 'E':
            resultado.append(['epsilon'])
        else:
            elementos = []
            partes = alt.split('"')
            for i, parte in enumerate(partes):
```

```
                parte = parte.strip()
                if i % 2 == 0:  # No está entre comillas
                    # Buscar no terminales
                    tokens = [t.strip() for t in parte.split() if t.strip()]
                    for token in tokens:
                        if token.startswith('<') and token.endswith('>'):
                            elementos.append(token[1:-1])
                else:  # Está entre comillas
                    elementos.append(parte)
            resultado.append(elementos)

    return resultado

def crear_tabla_transiciones(gramatica):
    no_terminales = obtener_no_terminales(gramatica)
    terminales = obtener_terminales(gramatica)

    # Crear diccionario para la tabla
    tabla = {nt: {t: '' for t in terminales + ['$']} for nt in no_terminales}

    # Procesar cada regla
    for linea in gramatica:
        no_terminal, produccion = [p.strip() for p in linea.split('::=')]
        no_terminal = no_terminal[1:-1]  # Quitar < >

        alternativas = procesar_produccion(produccion)

        for alt in alternativas:
            if alt == ['epsilon']:
                tabla[no_terminal]['$'] = 'epsilon'
                for t in terminales:
                    if not tabla[no_terminal][t]:
                        tabla[no_terminal][t] = 'epsilon'
            else:
                # Si el primer elemento es terminal
                if alt[0] in terminales:
                    tabla[no_terminal][alt[0]] = ' '.join(alt)
                else:
                    # Si el primer elemento es no terminal
                    tabla[no_terminal][terminales[0]] = ' '.join(alt)

    return tabla, terminales, no_terminales

def guardar_csv(tabla, terminales, no_terminales, archivo_salida):
    with open(archivo_salida, 'w', newline='') as f:
        writer = csv.writer(f)
        # Escribir encabezados
        writer.writerow([''] + terminales + ['$'])
        # Escribir filas
        for nt in no_terminales:
            fila = [nt]
            for t in terminales + ['$']:
```

```
                fila.append(tabla[nt][t])
            writer.writerow(fila)

def main(archivo_entrada, archivo_salida):
    # Leer la gramática
    gramatica = leer_gramatica(archivo_entrada)

    # Crear la tabla de transiciones
    tabla, terminales, no_terminales = crear_tabla_transiciones(gramatica)

    # Guardar la tabla en CSV
    guardar_csv(tabla, terminales, no_terminales, archivo_salida)

if __name__ == '__main__':
    archivo_entrada = "GramaticaBNF.txt"
    archivo_salida = "tablaTransiciones.csv"
    main(archivo_entrada, archivo_salida)
```

Listing 3: Gramática

8.1. Explicación del código: Tabla de Transiciones

- **import csv**: Se importa el módulo `csv` para manejar la escritura de archivos en formato CSV.
- **leer_gramatica(archivo)**: Abre y lee el archivo de la gramática, filtrando líneas vacías y aquellas que no contienen la cadena `::=`. Devuelve una lista con las producciones válidas.
- **obtener_no_terminales(gramatica)**: Extrae los no terminales desde la parte izquierda de las producciones, eliminando los signos <>. Devuelve una lista sin duplicados.
- **obtener_terminales(gramatica)**: Recorre la parte derecha de cada producción buscando terminales encerrados entre comillas. Los almacena en un conjunto para evitar duplicados y devuelve la lista ordenada.
- **procesar_produccion(produccion)**: Dada la parte derecha de una producción, divide las alternativas separadas por `|`. Cada alternativa se analiza en partes entre comillas para distinguir terminales de no terminales. El resultado es una lista de listas, donde cada sublista representa una alternativa de producción.
- **crear_tabla_transiciones(gramatica)**:
  - **no_terminales** y **terminales** se obtienen usando las funciones anteriores.
  - Se crea una tabla (diccionario anidado) inicializando cada celda vacía para todas las combinaciones de no terminales y terminales más `$`.
  - Para cada producción, se procesan sus alternativas. Si una alternativa es `epsilon`, se llena la fila con `epsilon` en las columnas vacías. Si la alternativa empieza con un terminal, se asigna directamente. Si empieza con un no terminal, se asigna arbitrariamente en la primera columna terminal (esto debería mejorarse usando `FIRST`).

- **guardar_csv(tabla, terminales, no_terminales, archivo_salida)**: Crea un archivo CSV con la tabla. La primera fila contiene los terminales y $. Luego, cada fila corresponde a un no terminal, seguido de sus transiciones.
- **main(archivo_entrada, archivo_salida)**: Ejecuta el proceso completo: lee la gramática, crea la tabla de transiciones y la guarda en un archivo.
- **if __name__ == '__main__'**: Indica que el bloque solo debe ejecutarse cuando el archivo se corre directamente. Define las rutas de entrada y salida, y llama a `main`.

9. Árboles impresos con Three.js