WayraSimi: Un Compilador de Lenguaje de Programación en Quechua

*Implementación de un compilador completo para el lenguaje WayraSimi

1st Jhordan Steven Octavio Huamani Huamani
Ingeniería de Software
Universidad La Salle
Arequipa, Perú
jhuamani@ulasalle.edu.pe

2nd Miguel Angel Flores Leon
Ingeniería de Software
Universidad La Salle
Arequipa, Perú
mflores@ulasalle.edu.pe

3rd Jorge Luis Ortiz Castañeda
Ingeniería de Software
Universidad La Salle
Arequipa, Perú
jortiz@ulasalle.edu.pe

***Resumen*—Este documento presenta WayraSimi, un lenguaje de programación compilado que combina la sintaxis clara de Python con la eficiencia de Go, desarrollado íntegramente en Quechua. El proyecto implementa un compilador completo que incluye análisis léxico, sintáctico, semántico y generación de código assembly. WayraSimi representa un avance significativo en la adaptación de la tecnología a las lenguas originarias, proporcionando una herramienta de programación que honra las raíces culturales andinas mientras ofrece capacidades modernas de desarrollo de software. El compilador procesa código fuente en cuatro etapas secuenciales y incluye un visualizador 3D interactivo del árbol sintáctico desarrollado con Three.js, además de una extensión personalizada para VS Code.**

***Index Terms*—compilador, lenguaje de programación, Quechua, análisis léxico, análisis sintáctico, análisis semántico, generación de código**

I. Introducción

En el presente informe se detalla exhaustivamente el proceso de desarrollo del lenguaje de programación WayraSimi, un compilador completo desarrollado íntegramente en Quechua. Este proyecto representa un esfuerzo significativo para crear una herramienta de programación que honre las tradiciones culturales andinas mientras proporciona capacidades modernas de desarrollo de software.

WayraSimi (que significa "Lenguaje del Viento.[en] Quechua) combina la claridad sintáctica de Python con la eficiencia de Go, creando un lenguaje compilado que busca ser tanto accesible como potente. El proyecto incluye un compilador de cuatro etapas: análisis léxico, sintáctico, semántico y generación de código assembly [1].

El desarrollo de este compilador aborda una necesidad importante en el campo de la informática: la adaptación de la tecnología a las lenguas originarias. Esto no solo facilita la enseñanza de conceptos de programación desde una perspectiva culturalmente significativa, sino que también fortalece el vínculo entre la tecnología moderna y nuestras tradiciones ancestrales [3].

I-A. Motivación

WayraSimi nace de la necesidad de expresar la lógica del mundo moderno en el lenguaje Quechua, honrando nuestras raíces ancestrales. El lenguaje fue concebido para que los pueblos andinos puedan describir algoritmos y estructuras de datos utilizando términos que resuenan con nuestra cosmovisión: ɀupay"para los enteros, "qillqa"para las cadenas de texto, ruray"para definir funciones. El espíritu del lenguaje, que combina la claridad de Python con la eficiencia de Go, está representado por su mascota, GoPhy (Figura 1).

Figura 1. GoPhy, la mascota de WayraSimi, una fusión que representa la sintaxis amigable de Python (serpiente) y el rendimiento de Go (gopher).

Este enfoque no solo facilita la comprensión de conceptos de programación, sino que también preserva y promueve el uso del Quechua en contextos tecnológicos modernos, creando un puente entre la sabiduría ancestral y la innovación contemporánea.

II. Especificación Léxica

La especificación léxica de WayraSimi define los tokens fundamentales del lenguaje, cada uno con su correspondiente expresión regular y significado en Quechua. La Tabla I presenta la especificación completa de tokens [1].

Los tokens están diseñados para reflejar conceptos familiares en Quechua, facilitando así la comprensión y el aprendizaje del lenguaje por parte de hablantes nativos.

Cuadro I
ESPECIFICACIÓN DE TOKENS EN WAYRASIMI

Token	Ejemplos	Regex	Representación	Significado en Quechua
YUPAY_TOKEN	123, -5	[0-9]+	yupay (entero)	Número, Cuenta
CHIQI_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	chiqap|mana chiqap	chiqap_kay (booleano)	Verdad, Realidad
IDENTIFICADOR_TOKEN	variable, funcion_1	[a-zA-Z][a-zA-Z0-9]*	identificador	Nombre
OPERADOR_MAS	+	\+	suma	Yapay (Añadir)
OPERADOR_MENOS	-	\-	resta	Qichuy (Quitar)
OPERADOR_PACHA	*	*	multiplicación	Miray (Multiplicar)
OPERADOR_RAKIY	/	\/	división	Rakiy (Dividir)
OPERADOR_MODULO	%	\%	módulo	Phuyu
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)
PALABRA_RESERVADA_SICHUS	sichus	sichus	si (if)	Si, Cuando
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

III. GRAMÁTICA

La gramática de WayraSimi está definida en formato BNF (Backus-Naur Form) y ha sido verificada para asegurar que no sea ambigua y esté factorizada por la izquierda [2]. La gramática completa se presenta a continuación:

```
<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>
                             ↪ ";"

<PrintStmt> ::= "imprimiy" "(" <ArgumentPrint>
    ↪ ")" ";"

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

<operaciones_matematicas> ::= <termino>
    ↪ <expresion_tail>
<expresion_tail> ::= "+" <termino>
    ↪ <expresion_tail>
                   | "-" <termino>
                       ↪ <expresion_tail>
                   | E

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

<Estructura_If> ::= "sichus" "(" <opciones> ")"
    ↪ <Block> <Else_opcional>

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

<datos> ::= <YUPAY_TOKEN> | <CHIQI_TOKEN> |
    ↪ <CHIQAP_TOKEN> | <QILLQA_TOKEN>
```

Listing 1. Gramática BNF de WayraSimi

III-A. Gramática LL(1)

Para la implementación del analizador sintáctico, la gramática ha sido transformada a formato LL(1), eliminando la recursión por la izquierda y factorizando las reglas. La Figura **??** muestra la interfaz utilizada para probar la gramática y verificar la factorización. Ejemplos de entrada válidos incluyen:

- `ruray hatun_ruray ( ) { var id yupay ; }` *(Declaración simple)*
- `ruray hatun_ruray ( ) { sichus ( id <id ( ) utaq id == id ( ) ) { imprimiy ( id ) ; } }` *(Condicional compuesto)*

- `ruray hatun_ruray ( ) { para ( id = id ; id <id ; id ++ ) { } }` *(Bucle for)*
- `ruray hatun_ruray ( ) { sichus ( id <id ) { sichus ( id >id ) { imprimiy ( id ) ; } } }` *(input2: condicionales anidados)*
- `ruray factorial ( n yupay ) yupay { sichus ( n == 0 ) { kutipay 1 ; } mana { kutipay n * factorial ( n - 1 ) ; } }` *(input7: recursividad, factorial)*
- `ruray hatun_ruray ( ) { var x yupay ; x := ( 5 + 3 ) * ( 2 - 1 ) / 4 ; imprimiy ( x ) ; }` *(input8: operaciones complejas)*

IV. Implementación

IV-A. Arquitectura del Compilador

El compilador WayraSimi está implementado en Python y consta de cuatro componentes principales que se ejecutan secuencialmente:

1. **Analizador Léxico** (`AnalizadorLexico.py`): Tokeniza el código fuente y identifica los elementos léxicos del lenguaje.
2. **Analizador Sintáctico** (`analizadorSintactico.py`): Construye el árbol de sintaxis abstracta (AST) verificando que el código cumple con la gramática.
3. **Analizador Semántico** (`AnalizadorSemantico.py`): Verifica la consistencia semántica, tipos de datos y alcance de variables.
4. **Generador de Código Assembly** (`creadorAssembly.py`): Produce código assembly optimizado a partir del AST verificado.

IV-B. Herramientas Adicionales

El proyecto incluye herramientas complementarias que enriquecen la experiencia de desarrollo:

- **Visualizador 3D**: Aplicación web desarrollada con Three.js que muestra el árbol sintáctico de manera interactiva.
- **Extensión VS Code**: Extensión personalizada que proporciona resaltado de sintaxis y características específicas para archivos `.wasi`.
- **Generador de Tablas**: Utilidad para crear tablas de transiciones LL(1) a partir de la gramática.

IV-C. Proceso de Compilación

El proceso de compilación sigue estos pasos:

1. El analizador léxico lee el archivo fuente `.wasi` y genera tokens.
2. El analizador sintáctico procesa los tokens y construye el AST.
3. El analizador semántico verifica la consistencia del programa.
4. El generador de código produce el assembly final.
5. Opcionalmente, el visualizador 3D muestra el árbol sintáctico.

IV-D. Repositorio

El código fuente completo del proyecto está disponible en el repositorio del curso, incluyendo todos los analizadores, herramientas de visualización, casos de prueba, y documentación técnica.

Repositorio:

- https://github.com/JhordanSir/Compiladores.git

V. Resultados

V-A. Visualización 3D

El visualizador 3D desarrollado con Three.js permite una representación interactiva del árbol sintáctico (ver Figura 2), utilizando:

- Esferas azules para símbolos no terminales
- Esferas verdes para símbolos terminales
- Esferas naranjas para producciones epsilon

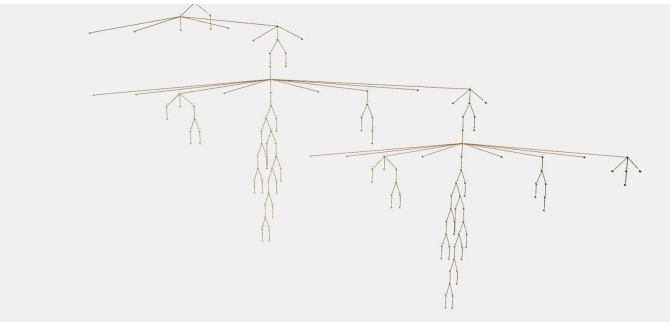

Figura 2. Visualización 3D interactiva de árboles sintácticos generados por el compilador WayraSimi.

La interfaz incluye controles de órbita, zoom, navegación por teclado, selector de temas y función de captura de pantalla.

V-B. Casos de Prueba

El compilador ha sido probado exitosamente con ocho casos de prueba diferentes (`input1.wasi` a `input8.wasi`), demostrando su capacidad para procesar diversos constructos del lenguaje incluyendo declaraciones de variables, estructuras de control, funciones y operaciones matemáticas. Ejemplos destacados:

- **input2**: Condicionales anidados

```
ruray hatun_ruray ( ) {
  sichus ( x < 10 ) {
    sichus ( y > 5 ) {
      imprimiy ( y ) ;
    }
  }
}
```

- **input7**: Recursividad (factorial)

```
ruray factorial ( n yupay ) yupay {
  sichus ( n == 0 ) {
    kutipay 1 ;
  } mana {
    kutipay n * factorial ( n - 1 ) ;
  }
}
```

- **input8**: Operaciones matemáticas complejas

```
ruray hatun_ruray ( ) {
  var x yupay ;
  x = ( 5 + 3 ) * ( 2 - 1 ) / 4 ;
  imprimiy ( x ) ;
}
```

VI. Conclusiones

WayraSimi representa un avance significativo en la integración de lenguas originarias con la tecnología moderna. El proyecto ha logrado:

1. Desarrollar un compilador completo funcional para un lenguaje de programación en Quechua.
2. Implementar todas las etapas de compilación: léxico, sintáctico, semántico y generación de código [1].
3. Crear herramientas visuales innovadoras para la comprensión del proceso de compilación.
4. Proporcionar una extensión de VS Code para mejorar la experiencia de desarrollo.
5. Demostrar la viabilidad de desarrollar herramientas de programación en lenguas indígenas [3].

Este trabajo abre nuevas posibilidades para la preservación y promoción de lenguas originarias en contextos tecnológicos, mientras proporciona una herramienta educativa valiosa para la enseñanza de conceptos de compiladores y programación.

El proyecto WayraSimi no solo cumple con los objetivos técnicos planteados, sino que también contribuye a la diversidad lingüística en el ámbito de la programación, honrando las tradiciones culturales andinas mientras abraza las innovaciones tecnológicas contemporáneas [2].

Referencias

[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, techniques, and tools*. Addison-Wesley.

[2] Appel, A. W. (2002). *Modern compiler implementation in Java*. Cambridge University Press.

[3] Grune, D., Van Reeuwijk, K., Bal, H. E., Jacobs, C. J., & Langendoen, K. (2012). *Modern compiler design*. Springer Science & Business Media.