Compilador *Compiscript*: Documentación (TAC + MIPS + IDE)

Andy Fuentes 22944
Davis Roldán 22672
Diederich Solis 22952

15 de noviembre de 2025

1. Resumen General

Implementamos un compilador funcional para *Compiscript*, un subconjunto de TypeScript, que traduce programas desde su gramática hasta código MIPS ejecutable. El pipeline completo es:

$$\text{ANTLR4} \rightarrow \text{AST} \rightarrow \text{Análisis semántico} \rightarrow \text{TAC} \rightarrow \text{MIPS}$$

Incluye un IDE interactivo en Streamlit para visualizar AST, diagnósticos, tabla de símbolos, TAC y MIPS. El backend MIPS genera código compatible con MARS/SPIM, y las pruebas finales se ejecutan directamente en MARS (ver sección de pruebas y video demostrativo).

2. Enlaces Importantes

2.1. Repositorio GitHub

URL del repositorio: https://github.com/DiederichSolis/Compilador

2.2. Video en YouTube (Demostración Completa)

URL del video: https://youtu.be/MxyiSILFzOw

3. Ejecución

3.1. CLI

```
PYTHONPATH=src python -m src.cli program/program.cps
# Genera: AST, diagnosticos, tabla de s mbolos, .tac y .s
```

4. Arquitectura del Compilador

Módulos principales:

- **parsing/antlr**: gramática Compiscript.g4 y artefactos generados por ANTLR4.
- **semantic/**: checker, tabla de símbolos, tipos y diagnósticos.
- **ir/tac/**: ISA TAC, builder (emitter) y representación del programa.
- **ir/backend/tac_generator.py**: visitor AST → TAC.
- **ir/backend/mips/**: generador MIPS + asignador de registros.
- **src/cli.py**: orquestación en terminal.

Decisiones de diseño: TAC textual y estable; optimización peephole local; backend MIPS minimalista y consistente; convención uniforme de pila.

5. Tabla de Símbolos

Soporta:

- Ámbitos: GLOBAL, FUNCTION, CLASS, BLOCK.
- Funciones: parámetros y tipo de retorno.
- Clases: campos, métodos y lookup con herencia.
- Constantes no reasignables.
- `foreach`: inferencia del tipo del elemento del arreglo.

Errores implementados (E101–E500): tipos incompatibles, símbolo no definido, retorno inválido, break/continue fuera de ciclo, switch incompatible, código inalcanzable.

6. Representación TAC

6.1. Operandos

- Temporales: `tN`
- Locales: `%x`
- Globales: `@g`
- Literales: `#5`, `#"str"`, `#null`

6.2. Formato de Función

```
.func <name>(params): <ret>
.locals N
<instrucciones y labels>
.endfunc
```

6.3. ISA TAC

Instrucciones principales: Move, Unary, Binary, Goto, If/IfFalse, Param, Call, Ret, NewObj, NewArr, ALoad, AStore, GetF, SetF, Print.

7. Generación TAC (Visitor)

Incluye:

- Evaluación izquierda→derecha.
- Operadores aritméticos, relacionales y lógicos.
- Corto-circuito en `&&`, `||`.
- Ternario traducido a saltos.
- Accessos encadenados: llamadas, campos y arreglos.
- Control estructurado: if/else, while, do-while, for.
- Manejo de pila de bucles para break/continue.
- Epílogo único por función con `Lret`.

8. Optimización Peephole

Regla segura:

```
goto L
L:
# El goto se elimina
```

Aplica principalmente tras saltos triviales o ramas que saltan directamente a Lret.

9. Backend MIPS

9.1. Convenciones

- Stack descendente.
- Parámetros vía stack.
- Retorno en `$v0`.
- Layout: locales → $ra → parámetros.

9.2. Prolog/Epilog Generado Automáticamente

- Prolog: reserva espacio + guarda `$ra`.
- Epilog: restaura `$ra`, libera pila, `jr $ra`.

9.3. Asignación de Registros

SimpleRegAllocator:

- Pool circular de `$t0{$t7`.
- Sin análisis de liveness (spills automáticos).

9.4. Strings y Literales

- Pool de literales en `.data`: `str_0`, `str_1`, ...
- Carga inmediata: `li`.
- Carga de string: `la reg, str_N`.

9.5. Mapping TAC → MIPS

- `Move`: cargar → `sw`.
- Aritméticas: `add, sub, mul, div`.
- Llamadas: push params → `jal` → `$v0`.
- Concatenación: llamada a `f_concat`.

10. IDE (Streamlit)

El proyecto incluye un IDE interactivo desarrollado con `Streamlit` para visualizar de forma integral cada etapa del compilador.

10.1. Características

- Editor de código con resaltado.
- Ejecución completa: sintaxis, semántica, TAC, MIPS.
- Bloqueo seguro si hay errores.
- Descarga de TAC y MIPS.
- Vista de AST con Graphviz, tabla de símbolos, tokens y diagnósticos.

10.2. Pestañas

- Diagnósticos.
- Árbol Sintáctico (AST).
- Tokens.
- Tabla de Símbolos.
- Código Intermedio (TAC).
- Código MIPS.

10.3. Ejecución

```
streamlit run src/ide/app.py
```

11. Pruebas

Las pruebas unitarias cubren semántica, generación TAC, control de flujo, arrays, objetos, operadores, ternario, break/continue y retorno.

11.1. Pruebas en MIPS

El backend produce un archivo `.s` compatible con MARS. **Las pruebas MIPS se ejecutaron directamente en MARS durante el video demostrativo.** Por ello, los resultados se documentan en dicho video.

```
$ pytest -q
18 passed
```

12. Ejemplo Final (Smoke Test)

```
.func max(a, b): integer
t1 = %a > %b
ifFalse t1 goto Lelse
t0 = %a
goto Lret
Lelse:
t0 = %b
Lret:
ret t0
.endfunc
```