



Tecnológico de Monterrey

**Desarrollo de aplicaciones avanzadas de ciencias computacionales
(503)**

Patito - Entrega #2

TC3002B.503

Mario Alberto González Méndez - A00832313

Prof. Carlos Acuña Ocampo

Prof. José Carlos Ortiz Bayliss

Profa. Elda Guadalupe Quiroga González

07 de Noviembre del 2025

En esta Etapa 2, el objetivo es implementar el Análisis Semántico. Esto significa que el compilador pueda entender el significado del código, verificar coherencia lógica, y validar los tipos de datos.

La implementación se basa en 3 cosas claves:

- 1- Estructura de Datos Jerárquicas
- 2- Cubo Semántico
- 3- Puntos Neurálgicos

1- Estructuras de datos (FuncDirectory y VarTable)

Es importante saber los scopes de una variable global no es lo mismo que en una función. Por eso, se implementaron dos clases en el archivo de [directory.py](#)

Clase VarTable (Tabla de Variables)

- Contenedor de variables globales y de función.
- Uso de estructura:
 - ({ 'nombre_var' : {...}})
- Operaciones:
 - add_var(nombre, tipo)
 - Manejo de error incluido

Clase FuncDirectory (Directorio de Funciones)

- Estructura principal que gestiona todas las funciones del programa.
- Usa diccionario en Python cada llave es nombre de una función 'global', 'calcular'
- Valor asociado de cada llave es un objeto que contiene el tipo de retorno de la función y crea una instancia de VarTable para sus variables locales.

2- Cubo Semántico (Validador de Tipos)

Sirve como una matriz enfocada en la validación.

El cubo es un diccionario anidado (cubo[tipo_izq] [tipo_der] [operador]) que define el tipo de resultado de cada operación válida.

Ventaja: Esta decisión mantiene el archivo parser.py limpio. El parser no necesita saber por qué 'entero' = 'flotante' es un error; simplemente "pregunta" al cubo.

Ejemplo de Operación (lookup):

- El parser encuentra `mi_entero = 5.5;`.
- Llama a `cubo_semantico.lookup('entero', 'flotante', '=')`.
- El cubo busca esta combinación, la encuentra definida como 'error' (para prevenir truncamiento) y lanza una excepción.
- El parser atrapa la excepción y reporta el error al usuario:
 - Error Semántico: Operación inválida. No se puede hacer 'entero = flotante'.

3. Puntos Neurálgicos: Integración en el Parser

Finalmente, se conectaron estas estructuras al analizador sintáctico mediante Puntos Neurálgicos (PN). Estas son acciones semánticas (fragmentos de código Python) que se "enganchan" a las reglas de la gramática en parser.py.

Se implementaron dos tipos de Puntos Neurálgicos:

1. PN de Declaración (para construir la "memoria"):
 - Ejemplo: Al final de la regla `p_decl_var` (que reconoce `ids : tipo ;`), se añadió el código para iterar sobre la lista de IDs y llamar a `dir_general.add_var_to_func(...)` por cada uno. Esto registra las variables en la `VarTable` del ámbito actual.
2. PN de Expresión (para validar con las "reglas"):
 - Implementación: Modificación de las reglas de expresión (`p_factor_id`, `p_cte`, `p_exp`, etc.) para que devuelvan el tipo de dato que representan (`p[0] = 'entero'`).
 - Ejemplo: Al analizar `(y + 5.5)` (con `y` siendo flotante):
 1. `p_factor_id` (para `y`) busca `y` en el `dir_general` y devuelve `p[0] = 'flotante'`.
 2. `p_factor_cte` (para `5.5`) ve un `CTE_FLOT` y devuelve `p[0] = 'flotante'`.
 3. `p_exp` (para `+`) recibe `'flotante'` de un lado y `'flotante'` del otro.
 4. Llama a `cubo_semantico.lookup('flotante', 'flotante', '+')`, que le devuelve `'flotante'`.
 5. `p_exp` pasa ese resultado (`p[0] = 'flotante'`) hacia arriba en el árbol.

Este mecanismo de pasar tipos hacia arriba me permite, en la regla final `p_asigna` o `p_condicion`, tener el tipo resultante de toda la expresión para realizar la validación final.

Link del github con todos los archivos del compilador:

<https://github.com/elshavo/Compiladores-ML/tree/main/Compiladores/LenguajePatitoV1>