Proyecto 3 Analizador Semántico

Iker Garcia German

2025-05-14

Victor Manuel de la Cueva

Introducción

El analizador semántico es la fase del compilador encargada de garantizar que el código fuente respete las reglas de tipos y ámbitos del lenguaje. A continuación se presentan las reglas de inferencia de tipos formales y la descripción de la tabla de símbolos empleada.

Reglas Lógicas de Inferencia de Tipos

Reglas en post order:

$$\frac{}{\vdash n : \text{Integer}} \quad (\text{Const})$$

$$\frac{\text{lookup}(x) = \tau}{\vdash x : \tau} \quad (\text{Var})$$

$$\frac{\vdash e_1 : \text{Integer} \quad \vdash e_2 : \text{Integer}}{\vdash e_1 \oplus e_2 : \text{Integer}} \quad (\text{OpInteger})$$

$$\frac{\vdash e_1 : \text{Integer} \quad \vdash e_2 : \text{Integer}}{\vdash e_1 == e_2 : \text{Integer}} \quad (\text{OpEq})$$

$$\frac{\vdash e_1 : \text{Integer} \quad \vdash e_2 : \text{Integer}}{\vdash e_1 < e_2 : \text{Integer}} \quad (\text{OpLt})$$

$$\frac{\vdash x : \text{Integer} \quad \vdash e : \text{Integer}}{\vdash x = e : \text{Integer}} \quad (\text{Assign})$$

$$\frac{\vdash e : \text{Integer}}{\vdash \text{if } (e) \text{ then } s_1 \text{ else } s_2} \quad (\text{If})$$

$$\frac{\vdash e : \text{Integer}}{\vdash \text{while } (e) \text{ do } s} \quad (\text{While})$$

$$\frac{}{\vdash \text{input}() : \text{Integer}} \quad (\text{Input})$$

$$\frac{\vdash e : \text{Integer}}{\vdash \text{output}(e) : \text{Void}} \quad (\text{Output})$$

$$\frac{f : \tau_1 \times \cdots \times \tau_n \to \tau_r, \quad \vdash e_1 : \tau_1, \ \ldots, \vdash e_n : \tau_n}{\vdash f(e_1, \ldots, e_n) : \tau_r} \quad (\text{Call})$$

$$\frac{\vdash e : \tau_r}{\vdash \text{return } e : \tau_r} \quad (\text{Return})$$

Estructura de la tabla de símbolos

Se usa un stack de scopes donde cada scope es un dict que mapea nombres a entradas (`SymEntry`)

```
# symtab.py
_symtab_stack: List[Dict[str, SymEntry]]
location_counter: int  # asigna offsets de memoria a variables

class SymEntry:
    name: str
    def_lineno: int       # línea de definición
    loc: int              # ubicación de memoria (offset)
    is_func: bool
    return_type: ExpType
    param_types: List[ExpType]
    use_lines: Set[int]   # líneas donde se usa la variable
```

Operaciones principales:

- `st_enter_scope()` : abre un nuevo scope (`append({})`).
- `st_exit_scope()` : cierra el scope más interno (`pop()`).
- `st_insert(name, lineno)` : inserta variable o añade uso.
- `st_insert_func(name, lineno, return_type, param_types)` : registra la cabecera de la función en el scope global.
- `st_lookup(name)` : busca de adentro hacia afuera en la pila.

Proceso de construcción:

1. `st_reset()` borra la pila y reinicia `location_counter` .
2. Se entra en *scope* global y se insertan las funciones `input` y `output` .
3. Recorrido preorder del AST con `insertar_nodo` :
   - Funciones (`DeclK` con cuerpo) → creación de nuevo scope y parámetros.
   - Bloques `{}` → `st_enter_scope()` .
   - Variables → `st_insert` .
   - Usos de identificadores → registro único de línea.

Ejemplo de tabla impresa:

Nombre	Tipo	Clase	Scope	Líneas	Extras
gcd	INT	function	global	17	params: 2
u	INT	param	nivel1	17, 18, 19	
x	INT	variable	nivel4	11, 12, 13	
...	...	...	...	...	...