Tecnológico de Monterrey

Desarrollo de aplicaciones avanzadas de ciencias computacionales (503)

Patito - Entrega #3

TC3002B.503

Mario Alberto González Méndez - A00832313

Prof. Carlos Acuña Ocampo
Prof. José Carlos Ortiz Bayliss
Profa. Elda Guadalupe Quiroga González

12 de Noviembre del 2025

Etapa 3 Generación de Codigo Intermedio (Cuadruplos)

Durante esta etapa 3, el compilador ya pasa de solo validar a ser un traductor. El objetivo fue tomar el codigo Patito que ya está correcto semanticamente, y traducirlo a una representación intermedia que son los cuadruplos.
Para lograr esto, necesitamos abandonar la logica de pasar tipos hacia arriba, (p[0] = tipo) y se implementó un sistema explícito de pilas stacks para gestionar las operaciones y operandos a medida que el parser los encuentra.

Estructura de datos Pilas y Fila

Para manejar la traducción de expresiones jerárquicas como (a+b) * c, a una lista lineal de instrucciones, se creó el archivo de quad_manager.py esto hace que el parser.py esté mas limpio y deja su logica dividida.

- Quad Manager contiene la siguiente estructura:
    - FIla de Cuádruplos:
        - Lista de python
        - Estructura para almacenar el resultado final, para First in first out se hace append y queda.
        - Operación agregar_cuadruplo(op, izq, der, res)
    - Pila de Operandos(PilaO):
        - Una lista de Python usada como Pila (append y pop)
        - Almacena operandos(variables como x o constantes como 5)
        - Operación: push_operando_tipo(operando, tipo.
    - Pila de Tipos(PilaT):
        - Lista de python como Pila
        - Sombra de PilaO. Almacena tipo de dato(entero o flotante de cada operando en pila O)
        - push_operando_tipo(operando, tipo)
        - Justo antes de generar un cuadruplo, sacamos los tipos de PilaT y los enviamos al cubo semántico para una ultima validación.
    - Pila de Operadores(POper):
        - Lista de python como pila.
        - Almacena operadores +*(=> para manejar precedencia. Un * se resuelve antes que +, el (como fondo para agrupar operaciones.
        - Operación: push_operador(operador)

Puntos Neurálgicos (PNs) en la Gramática

Para conectar el parser.py con el quad_manager.py se insertaron puntos Neuralgicos que son las reglas de gramatica vacías en lugares clave. Estos PNs actúan como gatillos que llaman a las funciones del quad_manager.

<factor> (La base de las expresiones)

- factor -> ID pn_factor_id
    - Acción (pn_factor_id): Busca el ID en el dir_general para obtener su tipo. Llama a quad_manager.push_operando_tipo(ID, tipo) para meterlos a PilaO y PilaT.
- factor -> cte pn_factor_cte
    - Acción (pn_factor_cte): La regla cte devuelve el valor y el tipo (ej. (5, 'entero')). Este PN llama a quad_manager.push_operando_tipo(5, 'entero') para meterlos a PilaO y PilaT.
- factor -> LPAREN pn_push_paren expresion RPAREN pn_pop_paren
    - Acción (pn_push_paren): Llama a quad_manager.push_operador('('). Esto actúa como un "muro" o "fondo falso" en la pila de operadores.
    - Acción (pn_pop_paren): Llama a quad_manager.pila_operadores.pop(). Esto elimina el "fondo falso" (después de que toda la expresion interna ya se ha resuelto.

<termino> y <exp> (Manejo de Precedencia)

- termino -> ... pn_check_op_mult
    - Acción (pn_check_op_mult): Se ejecuta después de leer un factor. Revisa el tope de POper. Si es * o /, genera el cuádruplo inmediatamente (ej. (op, izq, der, temp)).
- exp -> ... pn_check_op_aditivo
    - Acción (pn_check_op_aditivo): Se ejecuta después de leer un termino. Revisa el tope de POper. Si es + o -, genera el cuádruplo.

Esta combinación de PNs nos permite resolver 5 * 2 antes de 10 + ... en la expresión 10 + 5 * 2.

<expresion> (Operadores Relacionales)

- expresion -> exp pn_expresion_relacional
    - Acción (pn_expresion_relacional): Si existe un operador relacional (ej. >), esta regla lo detecta y llama a quad_manager.generar_cuadruplo_expresion() para crear el cuádruplo booleano (ej. (>, a, b, t1)).

<asigna> (Estatutos Lineales)

- asigna -> ID ASIG pn_push_operador expresion ... pn_gen_quad_asig
    - Acción (pn_push_operador): Mete el = a POper.
    - Acción (pn_gen_quad_asig): Se ejecuta al final. Para este punto, la expresion ya se resolvió y dejó su resultado (ej. t5) en PilaO. Este PN:
        - Saca t5 de PilaO.
        - Saca = de POper.
        - Valida los tipos (tipo(ID) vs tipo(t5)) con el Cubo Semántico.
        - Genera el cuádruplo final: (=, t5, None, ID).

<imprime> (Estatutos Lineales)

- item_imprime -> ... pn_gen_quad_imprime
    - Acción (pn_gen_quad_imprime): Se ejecuta después de cada expresion o LETRERO en la lista de escribe. Saca el resultado de PilaO (ej. t6 o "hola") y genera el cuádruplo (ESCRIBE, None, None, t6).

Link del github con todos los archivos del compilador y codigos de prueba:
https://github.com/elshavo/Compiladores-ML/tree/main/Compiladores/LenguajePatitoV1