Generación de código intermedio

La generación de código intermedio es una fase fundamental de los compiladores en la que se traduce el código fuente en una representación que no depende directamente de la máquina final, pero que conserva la estructura lógica del programa.

Dos tipos de representaciones intermedias

Existen dos formas principales de representaciones intermedias:

1. **Árboles**, en particular los árboles de análisis sintáctico y los árboles de sintaxis.
    - Los árboles de análisis sintáctico muestran la estructura del lenguaje y también son llamados **árboles de sintaxis concreta**.
    - Los árboles de sintaxis (abstracta) representan solo los elementos esenciales del programa y también se conocen como **árboles de sintaxis abstracta**.
    - Los árboles abstractos son una versión simplificada de los árboles de análisis, eliminando elementos puramente sintácticos.
2. **Representación lineal**, principalmente el **código de tres direcciones**, que se asemeja al lenguaje ensamblador pero sin registros explícitos.

Construcción de árboles de sintaxis abstracta

Los compiladores reales no generan árboles de análisis sintáctico completos porque son demasiado grandes y no aportan información adicional. En su lugar, producen **árboles de sintaxis abstracta (AST)**.

- La construcción de estos árboles se realiza reduciendo la producción gramatical a sus componentes esenciales.

- Por ejemplo, en una instrucción `while`, el árbol abstracto tendría un nodo raíz `while` con dos hijos:

    1. El árbol que representa la condición.

    2. El árbol que representa el cuerpo de la instrucción.

Los árboles de sintaxis también se emplean para representar **bloques** y **expresiones**:

- Un bloque `{ stmts }` se traduce en un nodo cuyo hijo directo es el árbol de sus sentencias.

- En expresiones, la **precedencia de operadores** queda reflejada en la jerarquía del árbol, sin necesidad de reglas adicionales.

El árbol abstracto se construye mediante **atributos sintetizados** en las producciones gramaticales, que generan nodos a medida que se reconocen las partes del programa.

Comprobación estática

La **comprobación estática** se realiza en tiempo de compilación y busca detectar errores antes de la ejecución. Se diferencia de la **comprobación dinámica**, que ocurre durante la ejecución.

Ejemplos de comprobaciones estáticas:

1. **Sintácticas**, como evitar que un identificador se declare más de una vez en el mismo ámbito.

2. **De tipos**, que aseguran que los operandos de las operaciones correspondan al tipo esperado.

Estas comprobaciones incluyen:

- **Coerciones**, que son conversiones automáticas de un tipo a otro cuando se requiere.

- **Sobrecarga**, donde un mismo operador puede tener diferentes significados según los tipos de los operandos.

En la implementación, la información de los símbolos suele almacenarse en una **tabla de símbolos**, aunque deben optimizarse técnicas para evitar el uso ineficiente de memoria.

Máquinas de pila abstractas

Otra representación intermedia común es usar una **máquina de pila abstracta**, que simula una arquitectura sencilla para ejecutar el código.

Características:

- Separación entre memoria de instrucciones y de datos.
- Aritmética realizada directamente sobre la pila.
- Instrucciones básicas para operaciones aritméticas, manipulación de la pila (push, pop, dup), y control de flujo.
- Dos registros implícitos: **tos** (top of stack) y **pc** (program counter).

En este modelo se distingue entre **lvalues** (ubicaciones de memoria que pueden aparecer en el lado izquierdo de una asignación) y **rvalues** (valores que aparecen en el lado derecho).

Código de tres direcciones

El **código de tres direcciones** es una forma lineal de representación intermedia.

- Cada instrucción tiene un operador y hasta tres operandos, de los cuales uno es el destino y los demás son fuentes.
- Puede representarse como **cuádruplos** (operador y tres argumentos) o en notación más familiar (asignaciones con operadores).

Ejemplo de formas:

- Cuádruplos: `ADD x y z`

- Notación familiar: `x = y + z`

Traducción de declaraciones

- Una instrucción `if expr then stmt` se traduce en bloques de código para evaluar la expresión, una instrucción condicional que determina si continuar o saltar, el bloque del `then` y finalmente una etiqueta de salida.
- Los árboles abstractos almacenan estos nodos, y un método, digamos `gen()`, de cada nodo se encarga de generar el código correspondiente.

Traducción de expresiones

- Se construye código evaluando recursivamente los operandos y generando instrucciones para cada operación.
- A menudo se introducen **temporales** para guardar resultados intermedios.

Optimización básica

Aunque la optimización es un tema amplio, existen simplificaciones locales como:

- Reducir asignaciones redundantes.
- Evitar el cálculo repetido de subexpresiones comunes.

Traducción en máquinas de pila

- Las expresiones se traducen en secuencias de instrucciones que siguen un orden posfijo.
- Por ejemplo, una asignación con operaciones aritméticas genera instrucciones para apilar los valores, aplicar las operaciones y finalmente almacenar el resultado.

Control de flujo

El flujo se gestiona con instrucciones de salto como `goto`, `gofalse`, `gotrue`, además de `label` y `halt`.

- Estas instrucciones permiten implementar estructuras como condicionales y ciclos.

Pseudocódigo para declaraciones

La traducción de declaraciones se implementa mediante procedimientos que:

- Detectan el tipo de sentencia (asignación, `if`, `while`, etc.).
- Emiten instrucciones correspondientes según la gramática y las acciones semánticas.

✅ En conclusión, la generación de código intermedio utiliza representaciones abstractas como **árboles de sintaxis** o **código de tres direcciones**, junto con modelos simplificados como la **máquina de pila abstracta**, para traducir el programa fuente en una forma que permita al compilador aplicar comprobaciones estáticas, optimizaciones y, posteriormente, producir el código final para la máquina destino.