

“2024. Año del Bicentenario de la Erección del Estado Libre y Soberano de México”

DIRECCIÓN ACADÉMICA
Formato de entrega de evidencias
FO-205P11000-14

División:	Ingeniería en Sistemas Computacionales				Grupo:	361-M
Asignatura:	Lenguajes y autómatas II		Docente:	M. en T.I Brian Antonio Mejía Díaz		
Nombre y número de control:			Alemán Díaz Mauricio Leonardo 213107223			
Fecha de entrega:			18 de marzo del 2024			
Competencia No.	2	Descripción:	Diseña las reglas para traducir el código fuente a un código intermedio.			
Indicador de alcance:		A) Se adapta a situaciones y contextos complejos. B) Hace aportaciones a las actividades académicas desarrolladas.				
Evidencia de aprendizaje:		Reporte de investigación				

“2024. Año del Bicentenario de la Erección del Estado Libre y Soberano de México”

DIVISIÓN DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

2



Reporte de investigación: Análisis semántico

Asignatura: Lenguajes y autómatas II

Elaborado por: Alemán Díaz Mauricio Leonardo
213107223

Docente: M. en T.I Brian Antonio Mejía Díaz

Grupo: 361-M

Período: 2024-1

Contenido

Introducción.....	4
Generación de código intermedio	5
2.1. Notaciones.....	5
2.1.1 Prefija	6
2.1.2 Infija	7
2.1.3 Postfija.....	8
2.2 Representaciones de código intermedio.....	9
Representaciones:.....	9
2.2.1 Notación polaca	10
2.2.2 Triplos	11
2.2.3 Cuádruplos	12
Ventajas	12
Limitaciones	13
Conclusión	14
Referencias.....	15
Hoja de sellos.....	16

Ilustración 1 : notaciones.....	5
Ilustración 2 : notación prefija	6
Ilustración 3 : notación infija.....	7
Ilustración 4 : notación postfija.....	8
Ilustración 5 : lenguaje de programación Lisp.....	10
Ilustración 6 : triplos	11
Ilustración 7 : cuádruplo	12

Introducción

4

En el presente reporte de investigación se abarcarán temas de suma importancia para poder entender el concepto de la generación de código intermedio en la materia leguajes y autómatas, específicamente para el desarrollo e implementación dentro de un compilador, como anteriormente revisamos los principales subtemas del análisis semántico, anteriormente había mencionado la analogía del cuando estamos pequeños nos enseñan las estructuras de lenguaje y reglas de ortografía para una comprensión mas clara al tema visto anteriormente. En el presente tema que es “Generación de código intermedio” quisiera hacer una nueva analogía para que se pueda entender un poco mas claro este concepto que puede llegar a ser complicado. Si imaginamos que tenemos un texto en español, nuestro idioma nativo, pero se necesita que el texto sea transcrito a un idioma diferente por alguna razón o algún objetivo, supongamos que se necesita transcribir al portugués. Entonces decimos que queremos traducir de un idioma a otro, afortunadamente estos dos idiomas son basados del latín. Por lo tanto, suponemos que tenemos tres estaciones, la primera que es la de entrada (el español), la estación de proceso (el latín) y la estación de salida (el portugués). Comenzamos a analizar el texto de entrada y vemos que tiene una estructura propia, diferencias en palabras y diferentes características que se pueden ver a simple vista, después de la primera estación donde se analiza el texto en español pasa por su segunda estación donde mediante esta traduce el texto original a uno del latín, algo simple y sencillo con su propia estructura donde se puede relacionar con la siguiente estación de una manera diferente. Al terminar la segunda estación pasaría a la tercera donde con la traducción intermedia se pude volver a traducir, a transformar a un nuevo idioma con sus reglas propias. Ya en el campo computacional este concepto es un poco diferente sin embargo sencillo de entender si lo vemos de esta forma, mediante el código de programación que intentamos compilar, este buscara una forma de relacionarse con el lenguaje de la computadora mediante estructuras definidas con anterioridad para poder comprenderse.

Generación de código intermedio

21. Notaciones

5

¿Qué son las notaciones? Las notaciones en la generación de código intermedio son formas específicas de escribir representando información o expresiones de diferentes maneras o contextos. Este tipo de notaciones son importantes cuando se implementan en la generación de código intermedio por que expresan y visualizan conceptos y operaciones de una manera clara y eficiente dependiendo el contexto. Así mismo cuando se utiliza una forma de notación ya sea la notación prefija, infija y postfija simplifican el lenguaje natural, con una notación bien definida para evitar ambigüedades y mal entendidos. En dado caso que no se utilice la forma de notación correcta puede afectar a la eficiencia y legibilidad dentro del código intermedio.



Ilustración 1 : notaciones

Las notaciones también varían según cómo se recorra el árbol sintáctico, ya sea en inorden, preorden o postorden. Esta relación tiene un mapeo directo con la notación de los operadores. Donde cada nodo del árbol es correspondió a una palabra, estructura gramatical, operador u operando donde las ramas del árbol sintáctico conectan los nodos y combinan las expresiones.

2.1.1 Prefija

¿Qué es la notación prefija? Este tipo de notación se refiere a una de las tres formas de representar expresiones matemáticas o lógicas, donde el operador como los signos que conocemos normalmente como la suma, resta, multiplicación, división precede a los operandos, los que conocemos como los números.

Esta notación lleva un orden predilecto que es gracias a lo que se diferencia de las demás formas de notación, en el caso de la notación prefija el orden es el siguiente: operador – primer operando – segundo operando.

+ 3 4

Ilustración 2 : notación prefija

Dentro de esta forma de notación no son necesarios los paréntesis para indicar el orden de precedencia de operadores y la evaluación se realiza de izquierda a derecha consecutivamente hasta que se encuentre el primer operador seguido de los operandos que se encuentren en la expresión.

Con respecto a la evaluación se evalúa la expresión binaria y el resultado se trata de un nuevo operando.

Expresión infija: “3 * (4 + 2)”

Notación prefija equivalente: “* 3 + 4 2” (evaluado como “(3 * (4 + 2))”)

Esta forma de notación es útil por que simplifica expresiones sin la necesidad de incorporar paréntesis ya que en la aritmética computacional esta puede verse afectada con tantos operandos que debería contemplar.

2.1.2 Infija

La notación infija es la mas conocida y popular entre las formas de representar expresiones aritméticas y lógicas que se pueden encontrar en artículos y libros. Comúnmente se le identifica debido a que el operador es colocado entre los operandos.

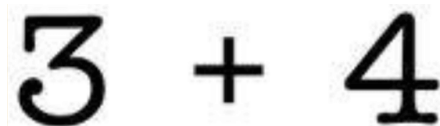
The image shows the mathematical expression 3 + 4 in a large, bold, black font. The numbers 3 and 4 are on either side of the plus sign (+), which is centered between them. This illustrates the infix notation where the operator is placed between the operands.

Ilustración 3 : notación infija

La legibilidad de esta forma de notación es más fácil de entender para las personas comunes ya que es la forma como comúnmente es expresada, sin embargo, para un ordenador esto no tiene que ser como a los humanos. Ya que durante el proceso de compilación fuente de un código puede ser conveniente ser representado de una forma aritmética diferente para una mejor optimización. Con el uso del árbol de sintaxis abstracta se obtiene una expresión intermedia dependiendo de su configuración. El árbol de sintaxis con la notación infija facilita el análisis sintáctico y la validación de la corrección de las expresiones dadas por este mismo.

Cuando se llega a la generación de código objeto el código intermedio obtenido de la notación infija es utilizado como una base o un pilar para que el compilador sea capaz de traducir las expresiones en la secuencia o pila de instrucciones maquina propuestas por el desarrollador y así ser ejecutadas y entendibles por el procesador.

Esta notación a diferencia de las otras dos es la mas utilizada en la generación de código intermedio, ya que es un estándar para generar el código objeto al finalizar el proceso, sin embargo, las otras pueden ser mas recomendables en procesos como la evaluación de lenguajes de programación funcionales o de análisis de expresiones de cálculos algebraicos.

2.1.3 Postfija

Llegando a la última de las notaciones tenemos la notación postfija, también conocida como la notación polaca inversa, esta, diferente a las anteriores tiene su forma de expresar las expresiones en la cual los operadores se ubican después de sus operandos correspondientes, además no repite el uso de paréntesis que indican la procedencia de los operadores ni su asociatividad, esto porque estas características determinan el orden en que son expresados los operandos y operadores.

El diagrama muestra la expresión '3 4 +' en una tipografía grande y clara. Los números '3' y '4' están separados por un espacio, y el símbolo '+' los sigue. Esto ilustra cómo en la notación postfija los operandos se listan primero, seguidos por el operador.

Ilustración 4 : notación postfija

La evaluación de las expresiones en esta notación se realiza con el uso de una única estructura de datos, esta estructura de datos puede ser una pila o “stack”. Donde cuenta con un algoritmo propio para evaluar una expresión postfija.

1. Recorrer la expresión de izquierda a derecha.
2. Si se encuentra un operando, se empuja (push) en la pila.
3. Si se encuentra un operador, se desapilan (pop) los operandos necesarios de la pila (en el orden correcto según la aridad del operador), se realiza la operación y se empuja el resultado en la pila.
4. Al finalizar la expresión, el valor en la cima de la pila es el resultado final.

La notación postfija tiene una característica que se podría decir atractiva para su uso, ya que es simple y eficiente por que no requiere considerar reglas de procedencia ni asociatividad de operadores.

2.2 Representaciones de código intermedio

La representación de código intermedio son estructuras que son utilizadas durante un proceso de compilación de programas, las cuales son ejecutadas para hacerse pasar como un puente o una conexión del código fuente que es escrito por un programador hacia el código objeto que es el que ejecuta y entiende la computadora, tiene como propósito general simplificar la traducción de código fuente a código objeto.

Representaciones:

Árbol de Sintaxis Abstracta (AST): Es una representación jerárquica en forma de árbol que captura la estructura lógica del programa fuente. Cada nodo del árbol representa una construcción del lenguaje (expresiones, declaraciones, bucles, etc.) y las relaciones entre ellos reflejan las reglas gramaticales del lenguaje.

Grafo de Flujo de Control (CFG): Representa el flujo de control de un programa mediante un grafo dirigido, donde los nodos representan bloques básicos de código (secuencias de instrucciones sin bifurcaciones) y las aristas representan las transferencias de control entre ellos (saltos, llamadas, etc.).

Código de Tres Direcciones (Cuádruplas): Es una representación lineal en la que cada instrucción tiene a lo sumo dos operandos fuente y un operando destino.

Representación Intermedia de LLVM (LLVM IR): LLVM es un proyecto de código abierto que proporciona una infraestructura de compilación y un lenguaje intermedio propio llamado "LLVM IR".

Bytecode: Es una representación de código intermedio utilizada por máquinas virtuales o intérpretes, como la Máquina Virtual Java (JVM) o la Máquina Virtual de Python (CPython).

2.2.1 Notación polaca

La notación polaca es conocida también como la notación prefija, y como ya lo revisamos anterior mente es una forma para representar expresiones aritméticas y lógicas, para complementar con el punto visto anteriormente expondremos algunas características importantes de la notación polaca.

Eliminación de paréntesis: Cuando se colocan los operadores antes de los operandos si se requiere que sean agregados paréntesis para indicar precedencias y asociatividad de los operadores.

Evaluación de la recursividad: Cuando se representan las expresiones en la notación polaca pueden ser evaluadas sin tener que usar funciones o métodos recursivos, mediante el uso de una pila estructural, el algoritmo es similar al de la notación postfija pero los operadores son procesados antes de empujar los operadores a la pila.

Aplicaciones en lenguajes funcionales: La notación polaca es aplicada en diferentes lenguajes de programación funcionales como Lisp y Forth, donde las funciones y operaciones son representadas de una forma prefija.



Ilustración 5 : lenguaje de programación Lisp

Las ventajas de la notación polaca son algunas como, por ejemplo, al ser menos intuitiva y difícil de leer en comparación con la infija comúnmente es menos utilizada por lo que habrá menos algoritmos de integración o documentación para comprender su funcionamiento a mayores rasgos.

2.2.2 Triplos

Son una representación de código intermedio dentro de los compiladores. Al ser una representación de código intermedio se refiere a estructuras lineales de instrucciones donde cada instrucción se compone de tres o más campos o componentes.

11

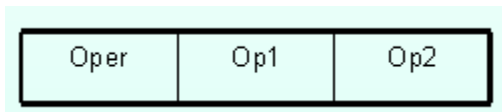


Ilustración 6 :triplos

Donde:

- Operador: Representa la operación o instrucción a realizar, como una asignación, operación aritmética, salto condicional, etc.
- Operando1: Es el primer operando de la instrucción, que puede ser una variable, constante, etiqueta o dirección de memoria.
- Operando2: Es el segundo operando de la instrucción, con el mismo rango de posibilidades que operando1.

En este caso se representa el código intermedio de un árbol de tres direcciones, para dos operadores y uno para un resultado. La ventaja primordial de los triplos es que tienen mayor consistencia, sin embargo, dependen de un índice de posición, por ello tienen una capacidad de optimización presente en el manejo del código complejo. La notación de triplos necesita un menor espacio y el compilador no es necesario que genere nombres temporales según la posición del índice que es calculado. Una forma de solución se basa en listar las posiciones a las tripletas donde en lugar de listar las tripletas mismas. Existiendo un optimizador este cumplirá con mover una instrucción reordenando la lista, sin tener que mover las tripletas en sí.

2.2.3 Cuádruplos

Los cuádruplos son una forma de representación de código intermedio en la que cada instrucción se describe mediante una tupla (cuádrupla) que consta de cuatro campos.

12

oper	op1	op2	res
*	C	D	T1
+	B	T1	T2
:=	A		T2

Ilustración 7 : cuádruplo

- **operador:** Representa la operación o instrucción a realizar, como una asignación, operación aritmética, salto condicional, llamada a función, etc.
- **operando1 y operando2:** Son los operandos de la instrucción, que pueden ser variables, constantes, etiquetas o direcciones de memoria.
- **resultado:** Es el lugar donde se almacena el resultado de la operación, que puede ser una variable temporal, un registro o una dirección de memoria.

Ventajas

Simplicidad: Su estructura de cuatro campos es sencilla y fácil de generar y analizar.

Linealidad: Los cuádruplos forman una secuencia lineal de instrucciones, lo que facilita su procesamiento y optimización.

“2024. Año del Bicentenario de la Erección del Estado Libre y Soberano de México”

Independencia de la máquina: Al ser una representación intermedia, los cuádruplos son independientes de la arquitectura de la máquina objetivo, lo que promueve la portabilidad del compilador.

13

Optimizaciones: Los cuádruplos permiten realizar diversas optimizaciones de código, como la eliminación de código redundante, propagación de constantes, asignación de registros, etc.

Limitaciones

Variables temporales adicionales: Pueden requerir la introducción de variables temporales adicionales para almacenar resultados intermedios, lo que puede aumentar el consumo de memoria.

Representación de control de flujo: Pueden tener dificultades para representar algunas construcciones de control de flujo complejas de manera eficiente.

Entonces cada instrucción se codifica en una tupla de cuatro campos que son representadas con el operador, los operandos y el resultado de la operación, esto nos permite una representación pequeña y eficiente de expresiones y operaciones.

Conclusión

14

In the process of compilation, it is important to have appropriate representations of the source code that facilitate the different stages of analysis, transformation and object code generation. Intermediate code notations and representations play a key role in this respect. Notation infested, commonly used in programming languages, is the most natural and legible form for humans to represent expressions. It allows efficient synthetic analysis and construction of abstract syntax trees (AST), which capture the logical structure of the program. On the other hand, post-fixed and Polish notations, although less intuitive for humans, offer significant technical advantages, such as the removal of the need for parentheses, the ease of evaluation by batteries and their intermediate use. Linear intermediate representations, such as the code of three directions (triplos) and quadruples, are simple, compact and independent of the architecture of the target machine. They allow code optimizations efficiently and facilitate the final object code generation. In addition, there are more complex representations such as control flow graphs (CFG), which model the program's performance flow and are used for analysis and optimizations of data flow, and intermediate LV representations. The choice of intermediate code representation and the notation of expressions depends on several factors, such as the design of the compiler, source language, optimization objectives and portability requirements. Some representations, such as IR LLVM, have the advantage of being independent of source language and admitting multiple input sources. The notations and representations of intermediate code are key tools in the design and construction of compilers, as they facilitate analysis, optimization and code generation in an efficient and portable manner. Each performance and notation has its strengths and weaknesses, and the right choice is crucial for a successful compilation process and optimal performance of the generated code.

Referencias

de, G. (2020). *Unidad 2.- Generación de Código Intermedio*. [online] Lenguajes Y Automatas II -Yellow-. Available at: <https://5d5eec0cb105c.site123.me/unidades/unidad-2-generaci%C3%B3n-de-c%C3%B3digo-intermedio> [Accessed 19 Apr. 2024].

II, A. (2017). *2.1.- Notaciones*. [online] Lenguajes y Autómatas II. Available at: <https://5e344735705b1.site123.me/unidad-ii-generaci%C3%B3n-de-c%C3%B3digo-intermedio/21-notaciones> [Accessed 19 Apr. 2024].

Carlos, J. and Rojas, O. (n.d.). *Unidad VI Generación de Código Intermedio*. [online] Available at: <https://hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r97373.PDF>.

Laura, V. (2024). *Triplos*. [online] Blogspot.com. Available at: <https://vilmauramoralessunun.blogspot.com/2019/09/triplos.html> [Accessed 19 Apr. 2024].

"2024. Año del Bicentenario de la Erección del Estado Libre y Soberano de México"

Hoja de sellos

D	M	A
---	---	---



Tabla de triple → a partir de 3 elementos se genera un código de operación, un operando de entrada y salida para almacenar el resultado

Cuadruplos → se extiende la tabla con dos operandos de entrada y uno de salida

Esquemas de generación Diagramas de procesos y funcionalidad

Variables y constante con conversión

Expresiones (Operadores)

Instrucciones de asignación, instrucciones por expresión

Instrucciones de control

Ciclos y más

Funciones (+, -, *, /) entre métodos

Estructuras → Desarrollo de la pila semántica del compilador



L.S.C. Brian A. Mejía Díaz

[Handwritten signature]