
Contenido

1 INTRODUCCIÓN 1

- 1.1 ¿Por qué compiladores? Una breve historia 2
- 1.2 Programas relacionados con los compiladores 4
- 1.3 Proceso de traducción 7
- 1.4 Estructuras de datos principales en un compilador 13
- 1.5 Otras cuestiones referentes a la estructura del compilador 14
- 1.6 Arranque automático y portabilidad 18
- 1.7 Lenguaje y compilador de muestra TINY 21
- 1.8 C-Minus: Un lenguaje para un proyecto de compilador 26
- Ejercicios 27
- Notas y referencias 29

2 RASTREO O ANÁLISIS LÉXICO 31

- 2.1 El proceso del análisis léxico 32
- 2.2 Expresiones regulares 34
- 2.3 Autómatas finitos 47
- 2.4 Desde las expresiones regulares hasta los DFA 64
- 2.5 Implementación de un analizador léxico TINY ("Diminuto") 75
- 2.6 Uso de Lex para generar automáticamente un analizador léxico 81
- Ejercicios 91
- Ejercicios de programación 93
- Notas y referencias 94

3 GRAMÁTICAS LIBRES DE CONTEXTO Y ANÁLISIS SINTÁCTICO 95

- 3.1 El proceso del análisis sintáctico 96
- 3.2 Gramáticas libres de contexto 97
- 3.3 Árboles de análisis gramatical y árboles sintácticos abstractos 106
- 3.4 Ambigüedad 114
- 3.5 Notaciones extendidas: EBNF y diagramas de sintaxis 123
- 3.6 Propiedades formales de los lenguajes libres de contexto 128
- 3.7 Sintaxis del lenguaje TINY 133
- Ejercicios 138
- Notas y referencias 142

4 ANÁLISIS SINTÁCTICO DESCENDENTE 143

- 4.1 Análisis sintáctico descendente mediante método descendente recursivo 144
- 4.2 Análisis sintáctico LL(1) 152
- 4.3 Conjuntos primero y siguiente 168
- 4.4 Un analizador sintáctico descendente recursivo para el lenguaje TINY 180
- 4.5 Recuperación de errores en analizadores sintácticos descendentes 183
 - Ejercicios 189
 - Ejercicios de programación 193
 - Notas y referencias 196

5 ANÁLISIS SINTÁCTICO ASCENDENTE 197

- 5.1 Perspectiva general del análisis sintáctico ascendente 198
- 5.2 Autómatas finitos de elementos LR(0) y análisis sintáctico LR(0) 201
- 5.3 Análisis sintáctico SLR(1) 210
- 5.4 Análisis sintáctico LALR(1) y LR(1) general 217
- 5.5 Yacc: un generador de analizadores sintácticos LALR(1) 226
- 5.6 Generación de un analizador sintáctico TINY utilizando Yacc 243
- 5.7 Recuperación de errores en analizadores sintácticos ascendentes 245
 - Ejercicios 250
 - Ejercicios de programación 254
 - Notas y referencias 256

6 ANÁLISIS SEMÁNTICO 257

- 6.1 Atributos y gramáticas con atributos 259
- 6.2 Algoritmos para cálculo de atributos 270
- 6.3 La tabla de símbolos 295
- 6.4 Tipos de datos y verificación de tipos 313
- 6.5 Un analizador semántico para el lenguaje TINY 334
 - Ejercicios 339
 - Ejercicios de programación 342
 - Notas y referencias 343

7 AMBIENTES DE EJECUCIÓN 345

- 7.1 Organización de memoria durante la ejecución del programa 346
- 7.2 Ambientes de ejecución completamente estáticos 349
- 7.3 Ambientes de ejecución basados en pila 352
- 7.4 Memoria dinámica 373
- 7.5 Mecanismos de paso de parámetros 381

- 7.6 Un ambiente de ejecución para el lenguaje TINY 386
 - Ejercicios 388
 - Ejercicios de programación 395
 - Notas y referencias 396

8 GENERACIÓN DE CÓDIGO 397

- 8.1 Código intermedio y estructuras de datos para generación de código 398
- 8.2 Técnicas básicas de generación de código 407
- 8.3 Generación de código de referencias de estructuras de datos 416
- 8.4 Generación de código de sentencias de control y expresiones lógicas 428
- 8.5 Generación de código de llamadas de procedimientos y funciones 436
- 8.6 Generación de código en compiladores comerciales: dos casos de estudio 443
- 8.7 TM: Una máquina objetivo simple 453
- 8.8 Un generador de código para el lenguaje TINY 459
- 8.9 Una visión general de las técnicas de optimización de código 468
- 8.10 Optimizaciones simples para el generador de código de TINY 481
 - Ejercicios 484
 - Ejercicios de programación 488
 - Notas y referencias 489

Apéndice A: PROYECTO DE COMPILADOR 491

- A.1 Convenciones léxicas de C— 491
- A.2 Sintaxis y semántica de C— 492
- A.3 Programas de muestra en C— 496
- A.4 Un ambiente de ejecución de la Máquina Tiny para el lenguaje C— 497
- A.5 Proyectos de programación utilizando C— y TM 500

Apéndice B: LISTADO DEL COMPILADOR TINY 502

Apéndice C: LISTADO DEL SIMULADOR DE LA MÁQUINA TINY 545

Bibliografía 558

Índice 562

Capítulo 3

Gramáticas libres de contexto y análisis sintáctico

- | | |
|---|--|
| 3.1 El proceso del análisis sintáctico | 3.4 Ambigüedad |
| 3.2 Gramáticas libres de contexto | 3.5 Notaciones extendidas: EBNF y diagramas de sintaxis |
| 3.3 Árboles de análisis gramatical y árboles sintácticos abstractos | 3.6 Propiedades formales de los lenguajes libres de contexto |
| | 3.7 Sintaxis del lenguaje TINY |

El análisis gramatical es la tarea de determinar la sintaxis, o estructura, de un programa. Por esta razón también se le conoce como **análisis sintáctico**. La sintaxis de un lenguaje de programación por lo regular se determina mediante las **reglas gramaticales** de una **gramática libre de contexto**, de manera similar como se determina mediante expresiones regulares la estructura léxica de los tokens reconocida por el analizador léxico. En realidad, una gramática libre de contexto utiliza convenciones para nombrar y operaciones muy similares a las correspondientes en las expresiones regulares. Con la única diferencia de que las reglas de una gramática libre de contexto son **recursivas**. Por ejemplo, la estructura de una sentencia if debe permitir en general que otras sentencias if estén anidadas en ella, lo que no se permite en las expresiones regulares. Este cambio aparentemente elemental para el poder de la representación tiene enormes consecuencias. La clase de estructuras reconocible por las gramáticas libres de contexto se incrementa de manera importante en relación con las de las expresiones regulares. Los algoritmos empleados para reconocer estas estructuras también difieren mucho de los algoritmos de análisis léxico, ya que deben utilizar llamadas recursivas o una pila de análisis sintáctico explícitamente administrada. Las estructuras de datos utilizadas para representar la estructura sintáctica de un lenguaje ahora también deben ser recursivas en lugar de lineales (como lo son para lexemas y tokens). La estructura básica empleada es por lo regular alguna clase de árbol, que se conoce como **árbol de análisis gramatical** o **árbol sintáctico**.

Como en el capítulo anterior, necesitamos estudiar la teoría de las gramáticas libres de contexto antes de abordar los algoritmos de análisis sintáctico y los detalles de los analizadores sintácticos reales que utilizan estos algoritmos. Sin embargo, al contrario de lo que sucede con los analizadores léxicos, donde sólo existe, en esencia un método algorítmico (representado por los autómatas finitos), el análisis sintáctico involucra el tener que elegir entre varios métodos diferentes, cada uno de los cuales tiene distintas propiedades y capacidades. De hecho existen dos categorías generales de algoritmos: de **análisis sintáctico descendente** y de **análisis sintáctico ascendente** (por la manera

en que construyen el árbol de análisis gramatical o árbol sintáctico). Pospondremos un análisis detallado de estos métodos de análisis sintáctico para capítulos subsiguientes. En este capítulo describiremos de manera general el proceso de análisis sintáctico y posteriormente estudiaremos la teoría básica de las gramáticas libres de contexto. En la sección final proporcionaremos la sintaxis del lenguaje TINY en términos de una gramática libre de contexto. El lector familiarizado con la teoría de las gramáticas libres de contexto y los árboles sintácticos puede omitir las secciones medias de este capítulo (o utilizarlas como repaso).

3.1 EL PROCESO DEL ANÁLISIS SINTÁCTICO

La tarea del analizador sintáctico es determinar la estructura sintáctica de un programa a partir de los tokens producidos por el analizador léxico y, ya sea de manera explícita o implícita, construir un árbol de análisis gramatical o árbol sintáctico que represente esta estructura. De este modo, se puede ver el analizador sintáctico como una función que toma como su entrada la secuencia de tokens producidos por el analizador léxico y que produce como su salida el árbol sintáctico.

secuencia de tokens $\xrightarrow{\text{analizador sintáctico}}$ *árbol sintáctico*

La secuencia de tokens por lo regular no es un parámetro de entrada explícito, pero el analizador sintáctico llama a un procedimiento del analizador léxico, como `getToken`, para obtener el siguiente token desde la entrada a medida que lo necesite durante el proceso de análisis sintáctico. De este modo, la etapa de análisis sintáctico del compilador se reduce a una llamada al analizador léxico de la manera siguiente:

```
syntaxTree = parse();
```

En un compilador de una sola pasada el analizador sintáctico incorporará todas las otras fases de un compilador, incluyendo la generación del código, y no es necesario construir ningún árbol sintáctico explícito (las mismas etapas del analizador sintáctico representarán de manera implícita al árbol sintáctico), y, por consiguiente, una llamada

```
parse();
```

lo hará. Por lo común, un compilador será de múltiples pasadas, en cuyo caso las pasadas adicionales utilizarán el árbol sintáctico como su entrada.

La estructura del árbol sintáctico depende en gran medida de la estructura sintáctica particular del lenguaje. Este árbol por lo regular se define como una estructura de datos dinámica, en la cual cada nodo se compone de un registro cuyos campos incluyen los atributos necesarios para el resto del proceso de compilación (es decir, no sólo por aquellos que calcula el analizador sintáctico). A menudo la estructura de nodos será un registro variante para ahorrar espacio. Los campos de atributo también pueden ser estructuras que se asignen de manera dinámica a medida que se necesite, como una herramienta adicional para ahorrar espacio.

Un problema más difícil de resolver para el analizador sintáctico que para el analizador léxico es el tratamiento de los errores. En el analizador léxico, si hay un carácter que no puede ser parte de un token legal, entonces es suficientemente simple generar un token de error y consumir el carácter problemático. (En cierto sentido, al generar un token de error, el analizador léxico transfiere la dificultad hacia el analizador sintáctico.) Por otra parte, el analizador sintáctico no sólo debe mostrar un mensaje de error, sino que debe recuperarse del error y continuar el análisis sintáctico (para encontrar tantos errores como sea posible). En

ocasiones, un analizador sintáctico puede efectuar **reparación de errores**, en la cual infiere una posible versión de código corregida a partir de la versión incorrecta que se le haya presentado. (Esto por lo regular se hace sólo en casos simples.) Un aspecto particularmente importante de la recuperación de errores es la exhibición de mensajes de errores significativos y la reanudación del análisis sintáctico tan próximo al error real como sea posible. Esto no es fácil, puesto que el analizador sintáctico puede no descubrir un error sino hasta mucho después de que el error real haya ocurrido. Como las técnicas de recuperación de errores dependen del algoritmo de análisis sintáctico que se haya utilizado en particular, se aplazará su estudio hasta capítulos subsiguientes.

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

1. Pero vea más adelante en el capítulo el análisis sobre reglas gramaticales como ecuaciones.