

Contenido

Capítulo 1. Teoría de conjuntos.	1
1.1 Conjuntos.	3
1.1.1 Definiciones básicas.	3
1.1.2 Operaciones sobre conjuntos.	6
1.1.3 Diagrama de Venn.	7
1.1.4 Álgebra de conjuntos.	7
1.2 Relaciones.	9
1.2.1 Tipos de relación.	10
1.3 Funciones.	13
1.3.1 Imagen.	13
1.3.2 Grafo de una función.	14
1.3.3 Tipos de funciones.	15
1.4 Grafos.	16
1.4.1 Conceptos básicos.	16
1.4.2 Tipos de grafos.	18
1.5 Árboles.	21
1.5.1 Árboles con raíz.	21
1.5.2 Árboles binarios.	22
1.6 Métodos de demostración.	24
1.6.1 Teoremas.	24
1.6.2 Si... entonces.	24
1.6.3 Si, y sólo si.	26
1.6.4 y, o, no.	26
1.6.5 Tipos de teoremas.	27
1.6.6 Tipos de demostraciones en la teoría de conjuntos.	28
1.6.7 Inducción matemática.	29
Capítulo 2. Lenguaje.	39
2.1 Definiciones del autor.	40
2.2 Definiciones del diccionario.	40
2.3 Definición de lenguaje, orientada a la formalidad.	41
2.4 Conceptos básicos orientados a la teoría de compiladores.	43
2.4.1 Terminología asociada a la definición conceptual de lenguaje.	43
2.5 Lenguaje muy, pero muy básico para recetas de cocina.	44
2.5.1 Esquema de definición de lenguaje.	45
2.5.2 "Intentando" formalizar el lenguaje de recetas de cocina.	46
Capítulo 3. Lenguajes formales.	55
3.1 Definición del autor.	56
3.2 Operadores * y +.	56
3.3 Conceptos básicos.	56
3.4 Primera definición de lenguaje.	59

3.4.1 Lenguaje formal.	59
3.4.2 Lenguaje regular.	60
3.5 Expresiones regulares.	60
3.5.1 Precedencia en las expresiones regulares.	61
3.5.2 "Álgebra" de expresiones regulares.	62
3.5.3 Diseño de expresiones regulares.	63

Capítulo 4. Autómatas finitos. 67

4.1 Introducción.	68
4.2 Conceptos básicos.	68
4.2.1 Definición formal de autómata finito.	68
4.2.2 Autómatas finitos en representación gráfica.	69
4.2.3 Análisis de la entrada a través de un autómata finito.	69
4.2.4 Lenguaje aceptado por un autómata finito.	70
4.2.5 Ejemplos de autómatas orientados a diseñar la etapa de léxico de un lenguaje de programación.	70
4.3 Tipos de autómatas.	71
4.4 Autómata finito no determinista.	71
4.4.1 Análisis de la entrada a través de un autómata finito no determinista.	71
4.5 Autómata con transiciones- ϵ	72
4.5.1 Análisis de la entrada a través de un autómata finito con transiciones- ϵ	73
4.6 Autómata no determinista y con transiciones- ϵ	73
4.7 Conversión de autómata finito no determinista a autómata finito determinista.	73
4.8 Conversión de autómata finito con transiciones- ϵ a autómata finito determinista.	76
4.8.1 Otro ejemplo.	77
4.9 Aplicación de autómatas.	78
4.9.1 Analizador léxico 1.	78
4.9.2 Analizador léxico 2.	81
4.9.3 Analizador léxico 3.	81
4.9.4 Analizador léxico 4.	86
4.10 Conversión de expresión regular a autómata finito no determinista.	88
4.11 Conversión de expresión regular a autómata finito determinista.	90

Capítulo 5. Gramáticas. 95

5.1 Conceptos básicos.	96
5.1.1 Definición formal.	96
5.1.2 Convenciones de notación.	96
5.1.3 Notación simplificada.	97
5.1.4 Derivaciones.	97
5.1.5 Lenguaje generado por una gramática.	99
5.2 Diseño de gramáticas.	99
5.2.1 Técnicas para el diseño de gramáticas.	102
5.2.2 Modularidad de las gramáticas.	104
5.2.3 Límites de las gramáticas.	105
5.3 Diseño arbitrario de gramáticas.	106

5.4 Primer acercamiento a la jerarquía de Chomsky para gramáticas.	108
5.5 Gramática regular implementada en un autómata finito no determinista.	109

Capítulo 6. Autómatas de pila. 115

6.1 Jerarquía de lenguajes.	116
6.2 Conceptos básicos.	117
6.2.1 Definición formal.	117
6.2.2 Reconocimiento de una cadena en un autómata de pila.	117
6.2.3 Seguimiento en formato de corrida de escritorio.	119
6.3 Gramáticas independientes del contexto, implementadas en autómatas de pila.	119
6.3.1 Seguimiento en formato de corrida de escritorio.	122
6.4 Comentarios finales.	125

Capítulo 7. Máquina de Turing. 127

7.1 Conceptos básicos.	128
7.2 Máquina de Turing como realizadora de cálculos.	128
7.2.1 Reconocimiento de una cadena de entrada.	129
7.3 Máquina de Turing como reconocedora de lenguajes.	129
7.4 Diseño de la máquina de Turing.	130
7.4.1 Técnicas para la construcción de las máquinas de Turing.	130
7.4.2 Ejemplo 1. Máquina de Turing para sumar.	130
7.4.3 Ejemplo 2. Máquina de Turing para restar.	135
7.4.4 Ejemplo 3. Máquina de Turing para multiplicar.	141
7.5 Modularización en la máquina de Turing.	142
7.5.1 Ejemplo 1. Máquina "primitiva" de Turing para copiar.	142
7.5.2 Ejemplo 2. Máquina de Turing para sumar.	143
7.5.3 Ejemplo 3. Máquina de Turing para multiplicar.	144

Capítulo 8. Expresiones, primer acercamiento. 147

8.1 Consideraciones para generar expresiones en lenguajes de programación.	148
8.2 Conceptos básicos.	148
8.2.1 Términos dentro de una expresión.	148
8.2.2 Tipos de operadores.	149
8.2.3 Prioridad o precedencia.	149
8.3 Notaciones.	150
8.4 Generación de notaciones a través de árboles binarios.	151
8.4.1 Recorrido del árbol.	152
8.5 Importancia de la notación posfija.	153
8.6 Algoritmo para convertir de notación fija a notación posfija.	153
8.7 Algoritmo para evaluar notaciones posfijas.	155
8.8 Aplicación de los algoritmos de conversión y evaluación en un programa.	157
8.9 Comentarios finales.	160

Capítulo 9. Análisis sintáctico, autómatas de pila determinísticos. 165

9.1 Gramáticas independientes del contexto.	166
--	-----

9.1.1 Formato de las gramáticas independientes del contexto.	166
9.1.2 Características de las gramáticas independientes del contexto.	166
9.1.3 Tips en el diseño de gramáticas pensando en la implementación.	167

Métodos sintácticos descendentes.

9.2 Método predictivo recursivo o método directo.	169
9.2.1 Recursividad por la izquierda.	171
9.2.2 Subcadenas válidas.	174
9.2.3 Generalización para eliminar recursividad por la izquierda.	175
9.2.4 Definición formal de gramática recursiva por la izquierda.	175
9.2.5 Gramáticas ambiguas.	176
9.3 Método predictivo no recursivo.	177
9.3.1 Reconocimiento de la entrada.	177
9.3.2 Construcción de la tabla predictiva.	179
9.3.2.1 Primero.	179
9.3.2.2 Siguierte.	180
9.3.2.3 Construcción de la tabla de análisis sintáctico predictivo no recursivo.	182
9.3.3 Seguimiento de cadenas en tablas predictivas.	185

Métodos sintácticos ascendentes.

9.4 Métodos por desplazamiento y reducción.	187
9.4.1 Autómata de pila no determinístico ascendente.	188
9.5 Método LR _(k)	188
9.5.1 Modelo de un analizador LR.	189
9.5.2 Reconocimiento de la entrada.	189
9.5.3 Construcción de la tabla LR.	191
9.5.4 Programa LR.	197
9.5.5 Otro ejemplo.	199
9.5.6 Ejemplo. Desarrollo de la tabla LR para una gramática con producciones ϵ	200
9.6 Método sencillo para el manejo de errores en la tabla LR.	202
9.7 Grafo de transiciones.	204
9.8 Afinando términos para los métodos LR(0) y SLR(1).	205
9.9 Algoritmo para la construcción de las tablas SLR.	206
9.10 Características de las gramáticas LR(0) y SLR(1).	206
9.11 Conflictos desplazamiento/reducción.	207
9.12 Método LR(1).	208
9.13 Conflictos reducción/reducción.	211
9.14 Método LALR(1).	213
9.15 Exclusividad en las transiciones.	215
9.16 Gramáticas ambiguas LR.	217

Capítulo 10. Análisis semántico, traducción dirigida por la sintaxis. 225

10.1 Traducción dirigida por la sintaxis.	226
10.1.1 Ejemplo 1. Traductor de notación infija a posfija.	226
10.2 Generador de expresiones de 7 niveles de precedencia.	229

10.2.1	Tabla de precedencia.	229
10.2.2	Ejemplos de expresiones a resolver.	229
10.2.3	Desarrollo.	230
10.2.3.1	Léxico.	230
10.2.3.2	Sintaxis.	232
10.2.3.3	Semántica.	232

Capítulo 11. Intérprete MIBASIC. 241

11.1	Programas ejemplo.	242
11.2	Etapa de léxico.	247
11.3	Generador de expresiones.	250
11.4	Función main(), carga del programa y ciclo de ejecución de las instrucciones.	252
11.5	Instrucción de ASIGNACIÓN.	253
11.6	Instrucción INPUT.	253
11.7	Instrucción PRINT.	254
11.8	Instrucción IF.	255
11.9	Instrucción GOTO.	256
11.10	Instrucción FOR.	258
11.11	Instrucción GOSUB.	259

Apéndice A. Simbología. 267

Apéndice B. Ejercicios resueltos. 269

Bibliografía. 309

Índice alfabético. 311

Introducción

Desarrollar un compilador es una de las aplicaciones más complejas en el área de sistemas computacionales, debido a que se necesita integrar un conocimiento profundo en el área de matemáticas discretas, la programación de propósito general y las estructuras de datos.

Sin embargo, el desarrollo de un compilador puede facilitarse si se analizan detalladamente cada una de las etapas que lo integran.

El concepto de compilador está íntimamente ligado con el de lenguajes de programación, que en la actualidad son poderosas herramientas de desarrollo, los cuales integran un ambiente de programación que generalmente incluye un editor, ambientes visuales, depuradores, preprocesadores, un compilador, cargadores, funciones de biblioteca de propósito general, manejadores de bases de datos, controladores para sistemas distribuidos, instaladores, etc.

La responsabilidad del compilador dentro de los lenguajes de programación es la de convertir las instrucciones de alto nivel a código ensamblador, el resto de las responsabilidades en la generación de aplicaciones en un lenguaje de programación corren a cargo de las herramientas antes mencionadas.

Se reconocen casi universalmente seis etapas en el desarrollo de un compilador, las cuales son.

- Léxico.
- Sintaxis.
- Semántica.
- Generación de código intermedio.
- Optimización de código.
- Generación de código ensamblador.

Las tres primeras etapas tienen una alta carga matemática y las tres restantes una alta carga de algoritmos y estructura de datos. Por lo anterior es necesario dominar, realmente dominar, la parte matemática y de programación combinada con estructuras de datos para desarrollar un compilador.

El objetivo de esta obra es aplicar con profundidad, claridad, detalle y “desde cero” los modelos matemáticos que sustentan la teoría de compiladores, para esto es necesario que el programa de aprendizaje incluya temas periféricos o que se supone se deben tratar de manera especializada y por separado.

El camino del conocimiento necesario para desarrollar un compilador se encuentra plasmado en la gráfica que ilustra la portada.



Programación de sistemas
Compiladores
Intérpretes
Lenguajes y autómatas
Lenguajes formales
Teoría matemática de la computación
Matemáticas discretas

Si se tienen algunos antecedentes en los temas se puede observar que se dedican obras de manera individual y especializada a cada uno de los temas.

Para lograr el objetivo de esta obra, se sintetizan los temas con mayor relevancia en el área de compiladores, distribuidos en los capítulos de la siguiente forma.

Matemáticas discretas.	Capítulo 1. Teoría de conjuntos El material está orientado a manejar la terminología necesaria en el desarrollo de compiladores y se considera como antecedente a la teoría de compiladores.
Teoría matemática de la computación.	Capítulo 3. Lenguajes formales. Capítulo 4. Autómatas finitos. Capítulo 5. Gramáticas. Capítulo 6. Autómatas de pila. Capítulo 7. Máquina de Turing. Teoría matemática de la computación es el término moderno de lo con anterioridad se conoce como lenguajes formales y teoría de autómatas.
Tema periférico, lenguaje natural.	Capítulo 2. Lenguajes. El material incluye un tema de lenguajes desde el punto de vista no formal, muestra soluciones a problemas que involucran lenguajes, sin fundamentarlos matemáticamente. Este tipo de soluciones se implementarían si el diseñador no tiene conocimientos en la teoría matemática de computación.
Lenguajes formales.	Capítulo 3. Lenguajes formales. El material establece la primer relación, la cual está dada en la definición de que un lenguaje formal es un tipo especial de conjunto.
Lenguajes y autómatas.	Capítulo 4. Autómatas finitos. El material ilustra la forma en que los lenguajes formales con una alta carga conceptual se pueden implementar en máquinas que pueden programarse.

Tema periférico, manejo de expresiones.	<p>Capítulo 8. Expresiones, primer acercamiento.</p> <p>Las expresiones requieren de un manejo especial y cuidadoso. El material incluye un capítulo que muestra algoritmos no formales en la solución de expresiones. Por ser métodos no formales tienen poca aplicación práctica en un lenguaje de programación robusto, pero ilustran de manera muy clara conceptos útiles en los métodos formales.</p>
Intérpretes.	<p>Capítulo 4. Autómatas finitos. Capítulo 5. Gramáticas. Capítulo 9. Análisis sintáctico, autómatas de pila determinísticos. Capítulo 10. Análisis semántico, traducción dirigida por la sintaxis. Capítulo 11. Intérprete MIBASIC.</p> <p>Para poder desarrollar un lenguaje de programación completo se necesitan las etapas de léxico, sintaxis y semántica. Con éstas es posible desarrollar una aplicación que ejecute instrucciones.</p> <p>Existen dos formas de implementar un lenguaje de programación, una en formato compilador y la otra en formato intérprete.</p> <p>Formato compilador. Se traducen las instrucciones de alto nivel a código ensamblador, posteriormente se procesan en un compilador para ensamblador y se obtiene el código objeto, posteriormente se procesa en un cargador (linker) y se obtiene el código ejecutable que es independiente de las plataformas en que se desarrollaron los procesos de conversión del código.</p> <p>Formato intérprete. Se ejecutan las instrucciones conforme se están generando, la construcción de un lenguaje a nivel intérprete se puede llevar a cabo con las tres etapas iniciales de un compilador y es el objetivo de este libro, que el lector pueda desarrollar su propio lenguaje de programación con fundamento matemático, orden, control, eficacia, dominio y elegancia.</p>
Compiladores.	<p>Capítulo 4. Autómatas finitos. Capítulo 5. Gramáticas. Capítulo 9. Análisis sintáctico, autómatas de pila determinísticos. Capítulo 10. Análisis semántico, traducción dirigida por la sintaxis.</p> <p>En esta obra no considera el desarrollo final de la construcción de un compilador, considera las tres primeras etapas. Debido a que resultados palpables en una aplicación tomarán un poco más de tiempo, lo cual didácticamente no es aconsejable.</p> <p>Por lo anterior, esta obra aplica todo el fundamento matemático en el desarrollo de un lenguaje implementado en formato intérprete, lo que permite ver resultados en el mediano plazo de forma clara, práctica, contundente e inclusive divertida, aunque a mis estudiantes nunca los he visto muy divertidos en mis cursos.</p>
Programación de sistemas.	<p>No tratado en este libro.</p> <p>La programación de sistemas involucra herramientas paralelas al compilador o</p>

	<p>intérprete en los lenguajes de programación, algunas ya mencionadas anteriormente como son el preprocesador, el compilador de código ensamblador a código objeto, el cargador o linkeador, la interacción del lenguaje con el sistema operativo y la arquitectura de la computadora.</p> <p>La programación de sistemas es el último escalón en la construcción robusta de lenguajes de programación.</p>
--	--

Las etapas restantes de un compilador; generación de código intermedio, optimización y generación de código junto con la programación de sistemas serán tratados en la obra que le dará seguimiento a “Compiladores, el comienzo ...”.