"Teoría de Autómatas y Lenguajes Formales" (para Ingenieros Informáticos)

Domingo Gómez & Luis M. Pardo

February 15, 2015

Prólogo

Lo que sigue son una evolución de notas comenzadas a impartir en el curso 2007/08, para la asignatura "Teoría de Autómatas y Lenguajes Formales". La evolución de los contenidos ha sido siempre en función de las diversas promociones, su formación previa y su capacidad de adquirir nuevos conocimientos. Estos apuntes representan una versión definitiva de nuestra experiencia. Nuestro cuidado ha sido conseguir una adquisición de conocimientos teóricos, su aplicación con una formación de carácter matemático para que el alumno desarrolle una actitud rigurosa ante problemas de ésta y otras materias.

Este aspecto es de gran importancia en una materia como los autómatas y los lenguajes formales, considerados como uno de los fundamentos de las ciencias de computación. Dada la escasa formación previa en matemáticas de los alumnos que llegan a la Univerisdad española, se ha pretendido mantener el máximo de formalismo en definiciones y enunciados pero renunciando a las demostraciones en términos formales completos, y limitando éstas a aquellos casos en los que la prueba se basa en la existencia de un algoritmo. El objetivo del curso es que el alumno comprenda, aplique y asimile una serie de herramientas matemáticas que se han desarrollado para la teoría de autómatas y lenguajes formales.

Estos apuntes están también pensados para "aprender a resolver problemas algorítimicamente". Según nuestro punto de vista, la mejor forma para conseguir este objetivo es a través de adquirir experiencia en la resolución de problemas a través de algoritmos definidos con precisión.

Nuestro planteamiento es constructivo. Estos apuntes están dirigidos a futuros ingenieros, por lo que intentaremos que los algoritmos presentados sean eficientes y adecuados para implementación (al menos en los casos en que ésto sea posible) en un programa informático.

La teoría de la computación es un área abstracta de la ingeniería informática. Por ello incluimos ejemplos, cuestiones y problemas que van de preguntas triviales hasta retos que requerirán utilizar lo aprendido con sencillos argumentos semi–formales.

Los autores han utilizado este libro para una asignatura cuatrimestral, y su objetivo ha sido que los alumnos conocieran los lenguajes regulares y los libres de contexto. A partir de ellos, pudieran generar autómatas que reconocieran estos lenguajes y los algoritmos que permiten pasar de autámatas a gramáticas y viceversa. La última parte trata sobre dos clases de lenguajes especiales "LL" y "LR". Estos lenguajes tienen una importancia especial en el Análisis Sintáctico, como parte preliminar del poceso de compilación.

Las referencias bibliográficas tampoco pretenden ser exhaustivas, aunque sí contienen lo esencial de la literatura en estos temas.

Contents

1 Jerarquía de Chomsky **9**
- 1.1 Introducción . . . 9
- 1.2 Lenguajes Formales y Monoides . . . 13
  - 1.2.1 - . . . 14
  - 1.2.2 Operaciones Elementales con Lenguajes . . . 15
  - 1.2.3 Sistemas de Transición . . . 16
- 1.3 Gramáticas Formales . . . 16
  - 1.3.1 Sistema de Transición Asociado a una Gramática. . . . 17
  - 1.3.2 Otras Notaciones para las Producciones. . . . 18
    - 1.3.2.1 Notación BNF. . . . 18
    - 1.3.2.2 Notación EBNF. . . . 18
- 1.4 Jerarquía de Chomsky . . . 19
- 1.5 Disgresión: Problemas de Palabra . . . 21

2 Expresiones Regulares **25**
- 2.1 Las Nociones y Algoritmos Básicos . . . 25
  - 2.1.1 Las Nociones . . . 25
  - 2.1.2 La Semántica de las expresiones regulares. . . . 27
- 2.2 De RE's a RG's: Método de las Derivaciones . . . 28
  - 2.2.1 Derivación de Expresiones Regulares . . . 28
  - 2.2.2 Cómo no construir la Gramática . . . 29
  - 2.2.3 Derivadas Sucesivas: el Método de las derivaciones . . . 31
- 2.3 De RG's a RE's: Uso del Lema de Arden . . . 33
  - 2.3.1 Ecuaciones Lineales. Lema de Arden . . . 33
  - 2.3.2 Sistema de Ecuaciones Lineales Asociado a una Gramática. . . . 35
- 2.4 Problemas y Cuestiones. . . . 37
  - 2.4.1 Cuestiones Relativas a Lenguajes y Gramáticas. . . . 37
  - 2.4.2 Cuestiones Relativas a Expresiones Regulares. . . . 37
  - 2.4.3 Problemas Relativos a Lenguajes Formales y Gramáticas . . . 38
  - 2.4.4 Problemas Relativos a Expresiones Regulares . . . 40

3 Autómatas Finitos **43**
- 3.1 Introducción: Correctores Léxicos o Morfológicos . . . 43
- 3.2 La Noción de Autómata . . . 44
  - 3.2.1 Sistema de Transición de un autómata: . . . 45

3.2.1.1 Representación Gráfica de la Función de Transición. 47
3.2.1.2 Lenguaje Aceptado por un Autómata 48
3.3 Determinismo e Indeterminismo 49
3.3.1 El Autómata como Programa 49
3.3.2 Autómatas con/sin $\lambda-$Transiciones. 49
3.3.2.1 Grafo de $\lambda-$transiciones. 50
3.3.3 Determinismo e Indeterminismo en Autómatas 51
3.4 Lenguajes Regulares y Autómatas. 52
3.4.1 Teorema de Análisis de Kleene 52
3.4.2 Teorema de Síntesis de Kleene 53
3.5 Lenguajes que no son regulares 56
3.5.1 El Palíndromo no es un Lenguaje Regular. 59
3.6 Minimización de Autómatas Deterministas 60
3.6.1 Eliminación de Estados Inaccesibles. 61
3.6.2 Autómata Cociente 61
3.6.3 Algoritmo para el Cálculo de Autómatas Minimales. 62
3.7 Cuestiones y Problemas. 64
3.7.1 Cuestiones. 64
3.7.2 Problemas 66

4 Libres de Contexto **71**
4.1 Introducción 71
4.2 Árboles de Derivación de una Gramática 73
4.2.1 Un algoritmo incremental para la vacuidad. 75
4.3 Formas Normales de Gramáticas. 76
4.3.1 Eliminación de Símbolos Inútiles o Inaccesibles 76
4.3.1.1 Eliminación de Símbolos Inaccesibles. 78
4.3.1.2 Eliminación de Símbolos Inútiles. 78
4.3.2 Transformación en Gramáticas Propias. 79
4.3.2.1 Eliminación de $\lambda-$producciones. 79
4.3.2.2 Eliminación de Producciones Simples o Unarias . . 81
4.3.2.3 Hacia las Gramáticas Propias. 82
4.3.3 El Problema de Palabra para Gramáticas Libres de Contexto es Decidible. 84
4.3.4 Transformación a Formal Normal de Chomsky. 86
4.3.5 Forma Normal de Greibach 87
4.4 Cuestiones y Problemas 88
4.4.1 Cuestiones 88
4.4.2 Problemas 88

5 Autómatas con Pila **91**
5.1 Noción de Autómatas con Pila. 91
5.1.1 Las Pilas como Lenguaje (Stacks). 91
5.2 Sistema de Transición Asociado a un Autómata con Pila. 94
5.2.1 Modelo gráfico del sistema de transición. 95
5.2.2 Transiciones: Formalismo. 95

5.2.3 Codificación del Autómata con Pila. 97
5.3 Lenguaje Aceptado por un Autómata con Pila. 100
5.4 Equivalencia con Gramáticas Libres de Contexto. 105
5.5 Propiedades Básicas 107
5.6 Problemas 109
5.6.1 Problemas 109

6 Introducción a Parsing **115**
6.1 Introducción 116
6.1.1 El problema de parsing: Enunciado 119
6.2 Compiladores, Traductores, Intérpretes 119
6.2.1 Traductores, Compiladores, Intérpretes 120
6.2.1.0.1 Ventajas del Intérprete. 120
6.2.1.0.2 Inconvenientes de los Intérpretes. 120
6.2.1.1 Compiladores Interpretados. 121
6.2.2 Las etapas esenciales de la compilación. 121
6.2.2.1 La Compilación y su entorno de la programación. . 121
6.2.2.2 Etapas del Proceso de Compilación. 121
6.2.2.3 En lo que concierne a este Capítulo. 122
6.3 Conceptos de Análisis Sintáctico 122
6.3.1 El problema de la Ambigüedad en CFG 122
6.3.2 Estrategias para el Análisis Sintáctico. 124
6.4 Análisis CYK 126
6.4.1 La Tabla CYK y el Problema de Palabra. 126
6.4.2 El Árbol de Derivación con las tablas CYK. 128
6.4.3 El Algoritmo de Análisis Sintáctico CYK 129
6.5 Traductores Push–Down. 130
6.5.0.1 Sistema de Transición asociado a un PDT. 131
6.6 Gramáticas $LL(k)$: Análisis Sintáctico 132
6.6.1 FIRST & FOLLOW 132
6.6.2 Gramáticas $LL(k)$ 136
6.6.3 Tabla de Análisis Sintáctico para Gramáticas $LL(1)$ 138
6.6.4 Parsing Gramáticas $LL(1)$ 139
6.7 Cuestiones y Problemas 142
6.7.1 Cuestiones 142
6.7.2 Problemas 144

A Teoría Intuitiva de Conjuntos **149**
A.1 Introducción 150
A.2 Conjuntos. Pertenencia. 150
A.2.1 Algunas observaciones preliminares. 151
A.3 Inclusión de conjuntos. Subconjuntos, operaciones elementales. . . . 151
A.3.1 El retículo $\mathcal{P}(X)$. 153
A.3.1.1 Propiedades de la Unión. 153
A.3.1.2 Propiedades de la Intersección. 153
A.3.1.3 Propiedades Distributivas. 153

A.3.2 Leyes de Morgan. 153
A.3.3 Generalizaciones de Unión e Intersección. 154
A.3.3.1 Un número finito de uniones e intersecciones. 154
A.3.3.2 Unión e Intersección de familias cualesquiera de subconjuntos. 154
A.3.4 Conjuntos y Subconjuntos: Grafos No orientados. 154
A.4 Producto Cartesiano (`list`). Correspondencias y Relaciones. 155
A.4.1 Correspondencias. 156
A.4.2 Relaciones. 157
A.4.2.1 Relaciones de Orden. 158
A.4.2.2 Relaciones de Equivalencia. 159
A.4.3 Clasificando y Etiquetando elementos: Conjunto Cociente. . . 160
A.5 Aplicaciones. Cardinales. 161
A.5.1 Determinismo/Indeterminismo. 162
A.5.2 Aplicaciones Biyectivas. Cardinales. 164

Chapter 1

Jerarquía de Chomsky

Contents

1.1 Introducción 9
1.2 Lenguajes Formales y Monoides 13
1.2.1 - 14
1.2.2 Operaciones Elementales con Lenguajes 15
1.2.3 Sistemas de Transición 16
1.3 Gramáticas Formales 16
1.3.1 Sistema de Transición Asociado a una Gramática. 17
1.3.2 Otras Notaciones para las Producciones. 18
1.3.2.1 Notación BNF. 18
1.3.2.2 Notación EBNF. 18
1.4 Jerarquía de Chomsky 19
1.5 Disgresión: Problemas de Palabra 21

1.1 Introducción

La primera disquisición importante al fijar un modelo de cálculo hace referencia a los fundamentos de la comunicación y el lenguaje. Para ser precisos todo cálculo algorítmico consiste fundamentalmente en un proceso de comunicación: algo es emitido (input), manipulado (transmisión) y respondido (output). Es una comunicación entre hombre y máquina o una comunicación entre seres humanos. Pero el principio de esta discusión debe orientarse hacia lo que es susceptible de ser comunicado (tanto el input como el output son objetos comunicables).
Nos vamos directamente al Círculo de Viena, con precursores como K. Popper, y con actores activos como R. Carnap, H. Hahn y O. Neurath. En el ámbito de la lógica matemática son relevantes la pertenencia de miembros de la talla de K. Gödel o A. Tarski. Este influyente conjunto de filósofos, matemáticos y lógicos se ve roto por el nazismo en pedazos incomponibles, pero influyó muy notablemente la filosofía y la lógica de primeros de siglo (hasta finales de los 30).

Apliquemos el empirismo lógico como ideología provisional. Tomemos la disquisición inicial: ¿qué es susceptible de ser comunicado?.
Una respuesta razonable (empírica en nuestra aproximación) es que es comunicable todo aquello expresable en un alfabeto finito. Nuestra aproximación empírica se basa en la experiencia: no conozco ningún caso de información emitida o recibida por alguien, con contenido semántico no ambiguo, que no haya sido expresada sobre un alfabeto finito.
A partir de esta idea consideraremos como Σ un conjunto finito que denominaremos *alfabeto* y por Σ^* el conjunto de todas las *palabras* expresables sobre este alfabeto finito.
Dos precisiones importantes: Lo comunicado (el significante) es una palabra sobre un alfabeto finito, pero el significado (la componente semántica de la comunicación) no es tan claramente finito. Tomemos un ejemplo de las matemáticas. Sea D^1 el conjunto de números reales dado por:

$$\{(x, y) \in \mathbb{R}^2 \ : \ x^2 + y^2 - 1 \leq 0\}$$

El tal conjunto no es finito, ni contable. Podría quizá discutirse su existencia (uno de los problemas más difíciles de la filosofía de las matemáticas es determinar el significado de existencia: existe lo que es expresable –esto es seguro–, pero, ¿existe lo que no puedo expresar? [1]). Suponiendo que $\mathbb{R}$ exista, yo puedo expresar un conjunto cuyo cardinal no es numerable mediante una expresión sobre un alfabeto finito. Por lo tanto, los significantes caminan sobre una digitalización finita, sobre un alfabeto finito, no así los significados. No olvidemos, finalmente, que la modelización continua de la semántica es una de las corrientes de la moda última; pero tampoco olvidemos que la semántica (y la Semiótica) cuentan con los elementos adicionales de la subjetividad que son bastante "difusos".
La segunda consideración es que nosotros usaremos el lenguaje de la Teoría de la Recursividad y no el de la Lingística. Para referencias al asunto véase, por ejemplo, [Marcus, 67]. En este caso, la terminología se modifica del modo siguiente: el alfabeto se denomina vocabulario, las palabras son lo mismo, y el lenguaje es una cantidad, posiblemente infinita, de palabras sobre el vocabulario. Pero vayamos a nuestra definición:

Definición 1 (Alfabeto)**.** *Sea Σ un conjunto finito que llamaremos alfabeto.*

- *Una palabra sobre Σ es una lista finita de símbolos de Σ. Podemos formalmente identificar las listas $x = x_1 \cdots x_n$ de símbolos ($x_i \in \Sigma$) con los elementos del producto cartesiano Σ^n. Denotaremos por $\mid x \mid = n$ la longitud de la palabra $x_1 \cdots x_n$.*
- *El conjunto de todas las palabras sobre el alfabeto Σ se denotará mediante Σ^* y podemos identificarlo con la unión disjunta*

$$\Sigma^* = \dot{\bigcup_{n \in \mathbb{N}}} \Sigma^n$$

[1] Esto será lo máximo que nos introduciremos por los oscuros caminos de la filosofía. Paul Gordan, gran matemático del siglo XIX, amonestó a David Hilbert con su famosa frase "Das ist keine Matematik, Das ist Theologie" por "demostrar" la existencia de objetos, sin "mostrar" esos objetos.

- *Los subconjuntos L de Σ^* se denominan lenguajes.*

Insistamos en la notación $x_1 \cdots x_n$ para expresar la palabra $(x_1, \ldots, x_n) \in \Sigma^n$. Los ")" y las ", " pudieran ser (y suelen ser) elementos del alfabeto.
Nótese por ejemplo la identificación obvia, cuando $\Sigma = \{a\}$ es un alfabeto de una sola palabra, entre Σ^* y $\mathbb{N}$.

Nota 2. *La única observación relevante es que si Σ es un conjunto finito Σ^* es un conjunto numerable. No podemos expresar mucho más allá que una cantidad numerable de significantes (a lo sumo). La verdad, no es gran cosa: una sonata de Mozart no es sino una triste palabra de unas pocas numerables posibles.*
El considerar alfabetos numerables no cambiaría gran cosa, lo que se puede expresar sobre un alfabeto numerable es expresable sobre un alfabeto finito (por razones obvias). El punto duro comienza cuando uno se pregunta si es posible la comunicación (hombre–máquina u hombre–hombre) a través de lenguajes sobre alfabetos no numerables.

Otros ejemplos, "El Quijote", entendido como el libro completo, es, para nuestro contexto, una palabra sobre el alfabeto del castellano, i.e. $\{a, b, \ldots, z\}$, $\{A, B, \ldots, Z\}$, $\{?, \acute{A}, !, \text{“}, \text{”}, \text{“.”}, \vdots, \text{“}\sharp\text{”}, .\}$, donde las "," y "." son los obvios, $\vdots$ es el "punto-y-aparte" y $\sharp$ son los espacios entre palabras.
Uno podría muy bien argumentar porqué el "Otello" de Shakespeare no es una palabra del castellano y la respuesta es la obvia: es una palabra sobre el alfabeto castellano; pero el castellano no es solamente un alfabeto, sino un lenguaje $\mathcal{C} \subseteq \Sigma^*$ en el que se incluyen solamente las palabras formadas por sucesiones de símbolos del Diccionario de la Real Academia de la Lengua (ver autómatas finitos para más disquisiciones). El "Otello" pertenece al lenguaje inglés $\mathcal{I} \subseteq \Sigma^*$. Módulo traducciones (que no juegan por ahora) una versión original de la Ilíada[2], el Corán, El Idiota o los Vedas no pertenecerían a Σ^* por usar diferentes alfabetos (el griego, el árabe, el cirílico o el sánscrito). Por lo tanto, variarán tanto los alfabetos como los conjuntos llamados lenguajes, teniendo la comunicación occidental en común el alfabeto. Sin embargo, las traducciones muestran que no hay mucho que añadir aunque se cambie el alfabeto.
Esto muestra que alfabetos complicados o sencillos no tienen relación con la simplicidad del lenguaje. Aprovecharemos este momento para introducir un lenguaje que volverá a aparecer más adelante.

Ejemplo 1. *Sea $\Sigma = \{ A, C, G, T \}$, representando las cuatro bases que conforman el ADN, a saber: Adenina, Guanina, Citosina y Timina. Las cadenas de ADN forman un lenguaje que contiene la información genética y esta empaquetada de esta forma para su posible transmisión hereditaria. Curiosamente, este lenguaje es casi universal y se aplica a todos los seres vivos menos en excepciones contadas dentro de casos contados.*

Cada cadena de ADN guarda la codificación de varias proteínas. Dentro del ADN, cada secuencia de tres bases de las mencionadas corresponden a un aminoacido

[2] Aunque la tradición mantiene la ceguera de Homero y, por tanto, la transmisión oral de sus versos, aceptamos como "original" cualquier versión escrita durante el período helenístico.

concreto. Esto se conoce como el código genético. Por ejemplo, la combinación ATG representa el inicio de la secuencia y el final puede ser representada por TGA, TAG, TAA. [3]

Primeras concreciones sobre lo computable:
Un algoritmo es usado para resolver un problema. La entrada del algoritmo son los "datos del problema" que, convenientemente manipulados, aportan una "solución". Por lo tanto, todo algoritmo evalúa una correspondencia

$$f : D \longrightarrow S$$

donde D son los datos y S las soluciones. Como ya queda claro de lo discutido arriba (o eso esperamos), éste que escribe sólo puede discernir "datos" y "soluciones" como significantes de algo (el "algo" ya no corresponde a la disquisición). Luego

Definición 3. *Se definen:*

- *Un problema es una correspondencia $f : D \longrightarrow S$ entre dos conjuntos. Resolver un problema es evaluar f.*
- *Un problema $f : D \longrightarrow S$ es susceptible de ser resuelto algorítmicamente si y solamente si D y S son lenguajes expresables sobre un alfabeto finito. Uniendo alfabetos, uno podría suponer que son lenguajes sobre un alfabeto común Σ.*
- *Un problema es, por tanto, evaluar una correspondencia $f : \Sigma^* \longrightarrow \Sigma^*$. Los elementos del dominio (los datos) se suelen llamar inputs (también son susceptibles de ser llamados inputs aquellos $x \in \Sigma^*$ tales que no existe $f(x)$). Los elementos del rango de f son las soluciones y se denominan outputs.*

Entre los muchos problemas distinguimos una subclase de gran importancia: los PROBLEMAS DECISIONALES. Se trata de evaluar funciones parcialmente definidas $f : \Sigma^* \longrightarrow \{0, 1\}$. Claramente si $D(f)$ es el dominio de definición de f y definimos $L := f^{-1}(\{1\})$, la función f es del tipo restricción al dominio de f de la función característica de L (ver Ejemplo 37 en el Apéndice A). Los tales lenguajes L se denominarán lenguajes recursivamente enumerables cuando su función característica sea parcialmente computable, i.e. cuando $f : \Sigma^* \longrightarrow \{0, 1\}$ sea computable y:

- $L \subseteq D(f)$,
- $\chi_L \mid_{D(f)} = f$.

Para perfilar la noción de función computable y problema resoluble por un algoritmo debemos avanzar aún un largo trecho, que supera los estrechos márgenes de este curso. En todo caso, comencemos tratando de precisar cómo han de entenderse las manipulaciones de objetos de Σ^* que sirven para evaluar correspondencias f.

[3] El ADN siempre se ha considerado el "lenguaje de la vida" y parece que se cumple la máxima de Galileo: "La Naturaleza es un libro escrito con el lenguaje de las matemáticas".

1.2 Lenguajes Formales y Monoides

La operación esencial sobre Σ^* es la concatenación (también llamada adjunción) de palabras:

$$\cdot : \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$$

$$(x, y) \longmapsto x \cdot y$$

es decir, si $x = x_1 \cdots x_n$ e $y = y_1 \cdots y_m$, entonces

$$x \cdot y = x_1 \cdots x_n y_1 \cdots y_m.$$

Denotemos por $\lambda \in \Sigma^*$ la palabra vacía (para distinguirla del lenguaje vacío $\emptyset$, usando la notación estándar de Teoría de Conjuntos (cf. Apéndice A para más detalles).

Lema 4. *$(\Sigma^*, \cdot)$ es un monoide[4], donde λ es el elemento neutro. La longitud define un morfismo de monoides[5] entre Σ^* y el conjunto de los número naturales. El monoide Σ^* es abeliano[6] si y solamente si el cardinal de Σ es uno.*

Demostración. Ejercicio obvio. □

Lema 5. *Si Σ es un alfabeto finito, el conjunto Σ^* es numerable, esto es, es biyectable con el conjunto $\mathbb{N}$ de los números naturales.*

Demostración. Para dar una prueba de este enunciado basta con fijar un buen orden en Σ^*. Un ejercicio razonable consiste en definir el buen orden basado en "lexicográfico + longitud" que define la biyección. Recuérdese que el orden lexicográfico es el orden usual del "diccionario", i.e. basado en establecer un orden en el alfabeto Σ (en ocasiones lo llamarán alfabético). Es decir, sea $\ll$ un orden total en Σ (que existe por ser Σ finito). Supongamos que los elementos de Σ quedan ordenados mediante:

$$\Sigma := \{\alpha_1 \ll \alpha_2 \ll \cdots \ll \alpha_r\}.$$

Definimos para $x = x_1 \ldots x_n, y = y_1 \ldots y_m \in \Sigma^*$ la relación de orden siguiente:

$$x \leq y \Longleftrightarrow \begin{cases} \text{o bien} & n = |x| < |y| = m, \\ \text{o bien} & |x| = |y| = n = m, \quad \begin{cases} \exists k \leq n = m, \forall i \leq k-1, \\ x_i = y_i, \\ x_k \ll y_k \end{cases} \\ \text{o bien} & x = y. \end{cases}$$

[4] Recordemos que un monoide es un conjunto X con una operacón $* : X \times X \longrightarrow X$ que verifica la propiedad asociativa y de tal modo que X contiene un elementos neutro.

[5] Una transformación $f : (G, *) \longrightarrow (T, \perp)$, que verifica $f(\lambda) = \lambda$ y $f(x * y) = f(x) \perp f(y)$, es decir, respeta el elemento neutro y la operación entre dos elementos del primer monoide se transforma en la operación entre las imágenes.

[6] Todos sus elementos conmutan al operarlos.

Esta relación de orden es un buen orden en Σ^* y permite una identificación (biyección) entre Σ^* y $\mathbb{N}$, asociando a cada elemento de Σ^*, el número de elementos menores que el:

$$\begin{array}{ccc} \lambda & \longmapsto & 0 \\ \alpha_1 & \longmapsto & 1 \\ \alpha_2 & \longmapsto & 2 \\ & \cdots & \\ \alpha_1\alpha_1 & \longmapsto & r+1 \\ \alpha_1\alpha_2 & \longmapsto & r+2 \\ & \cdots & \end{array}$$

□

Nótese que como consecuencia (Corolario) se tienen las propiedades siguientes:

Corolario 6. *Sea Σ un alfabeto finito y $L \subseteq \Sigma^*$ un lenguaje. Entonces, L es un conjunto contable (i.e. es finito o numerable).*
Más aún, el cardinal de los posibles lenguajes $L \subseteq \Sigma^*$ coincide con el número de subconjuntos de $\mathbb{N}$ y, por tanto, verifica:

$$\sharp(\mathcal{P}(\Sigma^*)) = \sharp(\mathcal{P}(\mathbb{N})) = \sharp(\mathbb{R}) = 2^{\aleph_0}.$$

En particular, hay una cantidad infinita no numerable de lenguajes sobre un alfabeto finito (cf. el ejemplo 40).

1.2.1 -

Operaciones Básicas con palabras. Además de la concatenación de palabras, podemos destacar las siguientes:

- *Potencia de Palabras.* Se define recursivamente a partir de la concatenación. Así, dada una palabra $\omega \in \Sigma^*$ y un número natural $n \in \mathbb{N}$, definimos la potencia ω^n, mediante:
  - Definimos $\omega^0 = \lambda$,
  - Para $n \geq 1$, definimos
  $$\omega^n := \omega \cdot \omega^{n-1}.$$
- *Reverso de una Palabra:* Se trata de una biyección
  $$^R : \Sigma^* \longrightarrow \Sigma^*,$$
  dada mediante:
  - Si $\omega = \lambda$, $\lambda^R = \lambda$,
  - Si $\omega = x_1 \cdots x_n \in \Sigma^*$, con $x_i \in \Sigma$, definimos
  $$\omega^R := x_n x_{n-1} \cdots x_1 \in \Sigma^*.$$

Un lenguaje que tendrá cierta relevancia en nuestras discusiones posteriores es el *Palíndromo* $\mathcal{P} \subseteq \{0,1\}^*$ y que viene dado por la siguiente igualdad:

$$\mathcal{P} := \{\omega \in \{0,1\}^* \ : \ \omega^R = \omega\}.$$

1.2.2 Operaciones Elementales con Lenguajes

Vamos a considerar las siguientes operaciones básicas con lenguajes formales. Tendremos fijado un alfabeto Σ,

- *Unión de Lenguajes:* De la manera obvia como subconjuntos. Dados $L_1, L_2 \subseteq \Sigma^*$, definimos:

$$L_1 \cup L_2 := \{\omega \in \Sigma^* \ : \ [\omega \in L_1] \vee [\omega \in L_2]\}.$$

- *Concatenación de Lenguajes:* Dados $L_1, L_2 \subseteq \Sigma^*$, definimos su concatenación:

$$L_1 \cdot L_2 := \{\omega_1 \cdot \omega_2 \in \Sigma^* \ : \ \omega_1 \in L_1, \ \omega_2 \in L_2\}.$$

- *Potencia de Lenguajes:* Se define recursivamente.
  - Si $n = 0$, $L^0 = \{\lambda\}$.
  - Si $n \geq 1$, $L^n := L \cdot (L^{n-1})$.

Nota 7. *Obsérvese que $L_1 \cdot L_2$ no es, en general, igual a $L_2 \cdot L_1$. Tampoco es cierto que si $L_1 \cdot L_2 = L_2 \cdot L_1$ entonces se tiene $L_1 = L_2$. El ejemplo más sencillo de esto es $\Sigma = \{a\}, \ L_1 = \{a\}, \ L_2 = \{aa\}$.*

Proposición 8 (Distributivas)**.** *Con las anteriores notaciones, se tienen las siguientes propiedades para lenguajes L_1, L_2 y L_3 contenidos en Σ^*:*

$$L_1 \cdot (L_2 \cup L_3) = L_1 \cdot L_2 \cup L_1 \cdot L_3.$$

$$(L_1 \cup L_2) \cdot L_3 = L_1 \cdot L_3 \cup L_2 \cdot L_3.$$

Otras operaciones importantes entre lenguajes, hacen referencia al cálculo de la clausura transitiva por la operación de adjunción :

- *Clausura transitiva o monoide generado por un lenguaje:* Dado un lenguaje $L \subseteq \Sigma^*$ definimos el monoide L^* que genera mediante la igualdad siguiente:

$$L^* := \bigcup_{n \in \mathbb{N}} L^n.$$

- *Clausura positiva de un lenguaje:* Dado un lenguaje $L \subseteq \Sigma^*$ definimos la clausura positiva L^+ que genera mediante la igualdad siguiente:

$$L^+ := \bigcup_{n \geq 1} L^n.$$

Nota 9. *Es obvio que L^* es la unión (disjunta si $\lambda \notin L$) entre L^+ y $\{\lambda\}$. En otro caso (i.e. si $\lambda \in L$), ambos lenguajes coinciden.*

1.2.3 Sistemas de Transición

Una de las ideas esenciales en un proceso algorítmico es que "se van dando pasos hasta obtener un resultado". Lo del número finito de pasos lo dejamos para un poco después. Surge así la noción de
Sistema de Transición, Sistema Deductivo, Sistema de Producciones, Sistema de semi–Thue etc.

Definición 10. *Llamaremos sistema de transición a todo par $(S, \rightarrow)$, donde S es un conjunto (que se denomina espacio de configuraciones) y $\rightarrow \subseteq S \times S$ es una relación.*

Una *sucesión de computación* en el sistema de transición $(S, \rightarrow)$ es simplemente una sucesión finita de elementos de S:

$$s_1, \ldots, s_n$$

donde $(s_i, s_{i+1}) \in \rightarrow$ (se dice que la configuración s_{i+1} es deducible de s_i en un sólo paso deductivo). Normalmente, uno prefiere escribir $s_i \rightarrow s_{i+1}$ en lugar de $(s_i, s_{i+1}) \in \rightarrow$. Con ello, una computación en el sistema de transición $(S, \rightarrow)$ es simplemente:

$$s_1 \rightarrow \cdots \rightarrow s_n$$

Se dice que el sistema de transición es determinístico si cada $s \in S$ tiene un sólo sucesor a lo sumo y es indeterminista en caso contrario.

Definición 11. *Dada una configuración $s \in S$, diremos que una configuración $s' \in S$ es deducible de s y lo denotaremos por $s \vdash s'$, si existe una sucesión de computación*

$$s = s_1 \rightarrow \cdots \rightarrow s_n = s'$$

La relación que debe existir entre los datos de un problema y su resolución es de ser deducible para algún sistema de transición. En cada caso clarificaremos los sistemas de transición del modelo de cálculo introducido (es decir, la acción dinámica del modelo definido).

Nota 12. *Nótese la obvia analogía entre sistemas de transición y grafos (potencialmente con un número infinito de nodos). De hecho, un grafo orientado es simplemente un sistema de transición con un conjunto de configuraciones finito.*

La siguiente sección introducirá el concepto de gramáticas formales.

1.3 Gramáticas Formales

A. Thue[7] fue un matemático noruego que en 1914 introdujo la noción de sistema de reescritura. El interés de Thue era el análisis de los problemas de palabra para grupos y semi-grupos. Habrá que esperar a los trabajos de Noam Chomsky a finales de los años 50 para tener una estructuración de los sistemas de transición en el formato de gramáticas formales que, inicialmente, intentaba utilizar para modelizar los lenguajes naturales.

[7] A. Thue. Probleme über Veränderungen von Zichereihen nach gegebenen reglen. *Regeln. Skr. Videnk. Selks. Kristiania I, Mat. Nat. Kl. :* **10** (1914).

Definición 13 (Gramáticas Formales)**.** *Una gramática formal es una cuaterna* $G = (V, \Sigma, Q_0, P)$*, donde:*

- V *es un conjunto finito llamado alfabeto de símbolos no terminales o, simplemente, alfabeto de variables.*
- Σ *es otro conjunto finito, que verifica* $V \cap \Sigma = \emptyset$ *y se suele denominar alfabeto de símbolos terminales.*
- $Q_0 \in V$ *es una "variable" distinguida que se denomina símbolo inicial.*
- $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ *es un conjunto finito llamado conjunto de producciones (o, simplemente, sistema de reescritura).*

1.3.1 Sistema de Transición Asociado a una Gramática.

Para poder definir la dinámica asociada a una gramática, necesitamos asociarle un sistema de transición.

Definición 14. *Sea* $G = (V, \Sigma, Q_0, P)$ *una gramática, definiremos el sistema de transición asociado* $(S_G, \to_G)$ *dado por las propiedades siguientes:*

- *El espacio de configuraciones será dado por:*

$$S_G := (V \cup \Sigma)^* .$$

- *Dadas dos configuraciones* $s_1, s_2 \in S_G$*, decimos que* $s_1 \to_G s_2$ *si se verifica la siguiente propiedad:*

$$\exists x, y, \alpha, \beta \in S_G = (V \cup \Sigma)^* , \text{ tales que}$$

$$s_1 := x \cdot \alpha \cdot y, \;\; s_2 := x \cdot \beta \cdot y, \;\; (\alpha, \beta) \in P.$$

Ejemplo 2. *Consideremos la gramática:* $G = (V, \Sigma, Q_0, P)$*, donde*

$$V := \{Q_0\}, \;\; \Sigma := \{a, b\}, \;\; , P := \{(Q_0, aQ_0), (Q_0, \lambda)\}.$$

El sistema de transición tiene por configuraciones $S := \{Q_0, a, b\}^*$ *y un ejemplo de una computación sería:*

$$aaQ_0bb \to aaaQ_0bb \to aaaaQ_0bb \to aaaa\lambda bb = aaaabb.$$

Nótese que las dos primeras veces hemos usado la regla de reescritura (Q_0, aQ_0) *y la última vez hemos usado* (Q_0, λ)*.*

Notación 15. *Por analogía con el sistema de transición, se suelen usar la notación* $A \mapsto B$ *en lugar de* $(A, B) \in P$*, para indicar una producción. Y, en el caso de tener más de una producción que comience en el mismo objeto, se suele usar* $A \mapsto B \mid C$*, en lugar de escribir* $A \mapsto B$*,* $A \mapsto C$*.*

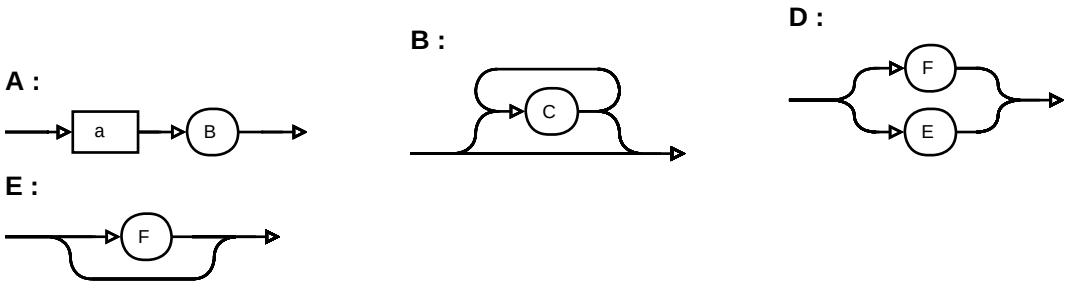


Figure 1.1: Representación EBNF A:"a"B, B:C*, D:F | E, E:F?

1.3.2 Otras Notaciones para las Producciones.

1.3.2.1 Notación BNF.

La notación de Backus-Naur, también conocida como **BNF** (de Backus–Naur Form), es una notación alternativa para las gramáticas y que remonta su origen a la descripción que, del sánscrito, hizo el gramático hindú Panini. No es una notación estandarizada, aunque está bien establecida. Entre otroas cosas porque los primeros usuarios de esta notación inisistieron en diversas notaciones para el símbolo $\mapsto$. Aquí usaremos el `‘‘estándar Wiki’’` por llamarlo de algún modo. Se trata de hacer los siguientes cambios

- Las variables $X \in V$ se representan mediante $\langle X \rangle$.
- Los símbolos terminales (del alfabeto Σ) se presentan entre comillas (“a”, “b”, “c”, ...)
- El símbolo asociado a las producciones $\mapsto$ es reemplazado por ::=.

Así, la gramática descrita en el Ejemplo 2 anterior vendría dada por:

$$V = \{\langle Q \rangle\}, \quad \Sigma = \{\text{“a”}, \text{“b”}\},$$

y las producciones estarían dadas por las reglas:

$$\langle Q \rangle = \text{“a”} \langle Q \rangle \mid \lambda$$

1.3.2.2 Notación EBNF.

Esta notación es extensión de la notación BNF. Es un estándar ISO-1497 y es utilizada (con algunas modificaciones) en los generadores de compiladores, como ANTLR.

Básicamente, añade funcionalidad a la notación BNF, permitiendo repeticiones o diferentes opciones. Varios ejemplos están dados en la figura encabezando la página (nótese la diferencia para los símbolos terminales y no terminales). Los siguientes son las principales modificaciones con respecto a la notación BNF,

- Las variables $X \in V$ no son modificadas.

- Los símbolos terminales (del alfabeto Σ) se representan entre comillas simples.
- El símbolo asociado a las producciones $\mapsto$ es reemplazado por :.
- Se introducen nuevos símbolos para representar repeticiones $*$ (ninguna, una o mas repeticiones) $+$ (una repetición al menos).
- ? indica que la expresión puede ocurrir o no.

Se deja como ejercicio al alumno hallar la expresión de la gramática

$$\begin{aligned} \langle Q \rangle = &\quad a\langle Q \rangle \\ \langle Q \rangle = &\quad \lambda. \end{aligned}$$

con notación EBNF.
Independiente de las notaciones, el elemento clave es la noción de lenguaje generado por una gramática. En lo que respecta a este manuscrito, utilizaremos la notación usada en páginas anteriores (equivalente a BNF) y evitaremos (por excesiva e innecesaria para nuestros propósitos) la notación EBNF.

Definición 16 (Lenguaje Generado por una gramática)**.** *Sea G una gramática definida como $G := (V, \Sigma, Q_0, P)$. Llamaremos lenguaje generado por la gramática G al lenguaje $L(G) \subseteq \Sigma^*$ dado por:*

$$L(G) := \{x \in \Sigma^* \ : \ Q_0 \vdash_G x\},$$

es decir, a las palabras sobre el alfabeto de símbolos terminales alcanzables (dentro del sistema de transición asociado) desde el símbolo inicial de la gramática.

1.4 Jerarquía de Chomsky

Chomsky pretende la modelización de los lenguajes (formales y naturales) mediante gramáticas en su trabajo [Chomsky, 57]. El uso de máquinas con un número finito de estados (autómatas) ya aparece en [Chomsky–Miller, 57].
Es en sus trabajos del año 59 (ca.[Chomsky, 59a] y [Chomsky, 59b]) donde aparece la clasificación que discutiremos en las páginas que siguen.

Definición 17 (Gramáticas Regulares o de Tipo 3)**.** *Definiremos las gramáticas con producciones lineales del modo siguiente:*

- *Llamaremos gramática lineal por la izquierda a toda $G := (V, \Sigma, Q_0, P)$ gramática tal que todas las producciones de P son de uno de los dos tipos siguientes:*
  - *$A \mapsto a$, donde $A \in V$ y $a \in \Sigma \cup \{\lambda\}$.*
  - *$A \mapsto aB$, donde $A, B \in V$ y $a \in \Sigma \cup \{\lambda\}$.*
- *Llamaremos gramática lineal por la derecha a toda gramática $G = (V, \Sigma, Q_0, P)$ tal que todas las producciones de P son de uno de los dos tipos siguientes:*
  - *$A \mapsto a$, donde $A \in V$ y $a \in \Sigma \cup \{\lambda\}$.*

- $A \mapsto Ba$, *donde* $A, B \in V$ *y* $a \in \Sigma \cup \{\lambda\}$.

- *Llamaremos* **gramáticas regulares** *a las gramáticas lineales por la izquierda o lineales por la derecha.*

La dualidad (y simetría) entre las garmáticas lineales a izquierda o lineales a derecha es obvia y nos quedaremos solamente con las gramáticas lineales a izquierda.

Definición 18 (Lenguajes Regulares)**.** *Un lenguaje* $L \subseteq \Sigma^*$ *se denomina un lenguaje regular si existe una gramática regular* $G = (V, \Sigma, Q_0, P)$ *que lo genera.*

Por definición una producción puede ser una transformación del tipo $\alpha A \beta \mapsto \omega$, donde $\alpha, \beta \in (\Sigma \cup V)^*$, $A \in V$. A las palabras α y β se las denomina *contexto* de la producción (o contexto de la variable A en esa producción). Así, una producción libre de contexto es una producción en la que ninguna variables tiene contexto, esto es, de la forma $A \mapsto \omega$, con $A \in V$.

Definición 19 (Gramáticas libres de contexto o de Tipo 2)**.** *Llamaremos gramática libre de contexto a toda* $G = (V, \Sigma, Q_0, P)$ *gramática tal que todas las producciones de* P *son del tipo siguiente:*

$$A \mapsto \omega, \text{ donde } A \in V \text{ y } \omega \in (\Sigma \cup V)^*.$$

Un lenguaje **libre de contexto** *es un lenguaje generado por una gramática libre de contexto.*

Definición 20 (Gramáticas sensibles al contexto o de Tipo 1)**.** *Llamaremos gramática sensible al contexto a toda gramática* $G = (V, \Sigma, Q_0, P)$ *tal que todas las producciones de* P *son del tipo siguiente:*

$$\alpha A \beta \mapsto \alpha \gamma \beta, \text{ donde } A \in V \text{ y } \alpha, \beta \in (\Sigma \cup V)^*,\ \gamma \in (\Sigma \cup V)^*,$$

es decir, en todas las producciones hay al menos una variable en la parte "izquierda" de la producción.
Un lenguaje **sensible al contexto** *es un lenguaje generado por una gramática sensible al contexto.*

Definición 21 (Gramáticas formales, sistemas de semi–Thue o de Tipo 0)**.** *Llamaremos gramática formal (o sistema de semi–Thue o sistema de reescritura finitamente generado y finitamente presentado) a toda gramática* $G = (V, \Sigma, Q_0, P)$ *que admite todo tipo de producciones, esto es, sus producciones son de la forma*

$$\alpha \mapsto \omega, \text{ donde } \alpha, \omega \in (\Sigma \cup V)^*, \alpha \neq \lambda.$$

En las gramáticas de tipo 0 (las más generales) admitimos que haya producciones sin ninguna variable en el lado izquierdo de la producción.

1.5 Sistemas de Thue: Problemas de Palabra

Las gramáticas de tipo 0 son también *Sistemas de Semi–Thue* (véase, por ejemplo, la referencia en [Davis-Weyuker, 94]) en honor del matemático que las introdujo. Hblaremos de sistemas de Semi–Thue finitamente generados y finitamente presentados cuando el alfabeto subyacente sea finito y las reglas de reescritura sean dadas en número finito. El objetivo de Thue era analizar el siguiente tipo de problemas.

Problema Motívico 1 (Problema de Palabra para Sistemas de Semi–Thue)**.** *Dado un sistema de semi–Thue* (Σ, R) *y dados* $x, y \in \Sigma^*$*, decidir si* $x \vdash_R y$*.*

Problema Motívico 2 (Problema de Palabra en Semigrupos)**.** *Dado* R *un sistema de semi–Thue sobre un alfabeto finito* Σ*, consideramos la estructura de semigrupo con unidad de* Σ^* *(monoide). Dos palabras* $x, y \in \Sigma^*$ *se dicen relacionadas mediante* R*, si* $x \vdash_R y$ *en el sistema de transición asociado (i.e. si* y *es deducible de* x*).*

Un sistema de Thue es un sistema de semi–Thue en el que R verifica la siguiente propiedad adicional :

$$\forall x, y \in \Sigma^*, \quad (x, y) \in R \Leftrightarrow (y, x) \in R$$

Entonces, R define una relación de equivalencia $\vdash_R$ en Σ^* y podemos considerar el conjunto cociente :

$$S(\Sigma, R) := \Sigma^* / \vdash_R$$

Claramente se tiene que $S(\Sigma, R)$ es un semigrupo, cuyos elementos son las clases $[x]$ definidas por elementos $x \in \Sigma^*$.
El problema de la palabra para semigrupos se define mediante :
Dados un sistema de Thue (Σ, R) *y dados* $x, y \in \Sigma^*$*, decidir si* $[x] = [y]$

Nota 22. *Esta versión del problema de la palabra está relacionada directamente con un hábito muy común en matemáticas. Supongamos que quiero trabajar con un semigrupo* S*, no necesariamente conmutativo. Para describirlo, todos pondríamos un conjunto de generadores (digamos* $\{\gamma_1, \ldots, \gamma_n\}$*). Sabidos los generadores, sabemos que los elementos son todos de la forma :*

$$\gamma_{s(1)} \cdots \gamma_{s(m)}$$

donde $s : \{1, \ldots, m\} \longrightarrow \{1, \ldots, n\}$ *es una aplicación, con* $m \in \mathbb{N}$*. El problema de una representación –tal cual ésta– es que uno no puede hacer cosas tan elementales como comparar dos elementos dados (obsérvese que nadie dijo que las cosas conmuten ni que la representación sea única). Por lo tanto, uno debería dar, al menos, las relaciones entre los generadores (que son inevitables). Estas relaciones tienen la pinta*

$$\gamma_{r_i(1)} \cdots \gamma_{r_i(m_i)} = \gamma_{s_i(1)} \cdots \gamma_{s_i(k_i)}$$

para $1 \leq i \leq N$*, siendo* r_i *y* k_i *aplicaciones con rango* $\{1, \ldots, n\}$*. Nos conformamos con que sólo haya un número finito de relaciones entre los generadores. Claramente, tenemos un sistema de reescritura sobre el alfabeto* $\Sigma = \{1, \ldots, n\}$*, siendo*

$$R := \{(r_i(1) \cdots r_i(m_i), s_i(1) \cdots s_i(k_i)) \ : \ 1 \leq i \leq N\}$$

Es obvio que nuestro semigrupo S inicial es justamente $S(\Sigma, R)$. Luego el problema de las palabras viene a decir si somos capaces de identificar o distinguir dos elementos de un semigrupo dado por sus generadores y sus relaciones. La respuesta , dada por E. Post[8] *en* 1947 *es que el problema de la palabra para semi–grupos finitamente presentados es indecidible (luego, insoluble).*

Teorema 23 (Post[9], 47)**.** *Los problemas de palabras para sistemas de semi–Thue, y semigrupos son insolubles algorítmicamente.*

El problema de palabra en grupos El problema anterior se sofistica un poco más, si en lugar de semigrupo hablamos de grupos. Un grupo finitamente generado (no necesariamente abeliano) no es sino un semigrupo asociado a un sistema de Thue (Σ, R) que, además verifica la propiedad siguiente :
existe una aplicación $\sigma : \Sigma \longrightarrow \Sigma$ *tal que :*

$$\forall a \in \Sigma, (a\sigma(a), \lambda) \in R)$$

donde λ *es la palabra vacía.* Escribamos $G(\Sigma, R)$ por el grupo cociente Σ^*/R
El problema de la palabra es también :
`Dado un sistema de grupo` (Σ, R) `y dadas` $x, y \in \Sigma^*$`, decidir si` $[x] = [y]$ `en` $G(\Sigma, R)$.
Tras mucho esfuerzo P. Novikov[10] (en 1955) y W.W. Boone[11] (con una demostración mucho más simple, en 1958) lograron demostrar que el enunciado siguente:

Teorema 24 (Novikov–Boone)**.** *El problema de palabra para grupos finitamente presentados y finitamente generados es insoluble algorítmicamente.*

Como aún no sabemos lo que es un algoritmo, dejemos la demostración para alguna referencia bibliográfica (cf. [Weihrauch, 97]). Nótese que los problemas de palabras de los sistemas de producciones también pueden interpretarse como una primera aproximación al problema de lo deducible en una teoría formal. Pero eso es otro asunto.
Problema de correspondencia de Post. Se trata de otro problema basado en los sistemas de reescritura y que resulta, también insoluble algorítmicamente (cf. E. Post[12] en 1946).

Problema Motívico 3 (Post Correspondence)**.** *Consideremos un sistema de semi–Thue* (Σ, R) *y sus elementos como piezas de dominó :*

$$R := \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

y las piezas

$$\mathcal{D}_i :=| \frac{x_i}{y_i} |$$

[8] E. Post. "Recursive unsolvability of a Problem of Thue". *J. of Symb. Logic* **12** (1947) 1–11.
[9] E. Post. "Recursive unsolvability of a Problem of Thue". *J. of Symb. Logic* **12** (1947) 1–11.
[10] P.S. Novikov. "On the algorithmic unsolvability of the word problem in group theory". *Proceedings of the Steklov Institute of Mathematics* **44** (1995), 1-143.
[11] William W. Boone. "The word problem". *Proceedings of the National Academy of Sciences* **44** (1958) 1061-1065.
[12] E. Post . "A variant of a recursively unsolvable problem." *Bull. A.M.S.* **52** (1946) 264–268.

Decidir si existe una secuencia de fichas

$$\mathcal{D}_{s(1)} \cdots \mathcal{D}_{s(n)}$$

tal que lo que aparece escrito en las partes superiores de los dominós coincide con lo escrito debajo.

Por ejemplo, sea R (Post prefiere Pairing Lists i.e. PL)

$$R := \{(a, aa), (bb, b), (a, bb)\}$$

para el alfabeto $\Sigma := \{a, b\}$. La siguiente es una solución :

$$| \frac{a}{aa} || \frac{a}{bb} || \frac{bb}{b} || \frac{bb}{b} |$$

Teorema 25 (Post, 46)**.** *El problema de la correspondencia es insoluble por métodos algorítmicos. En otras palabras, no existe (ni se puede encontrar) un algoritmo que resuelva el problema de correspondencia de Post.*

La prueba de la Indecidibilidad de este Problema puede verse en el [Weihrauch, 97] o en el [Davis-Weyuker, 94] , entre otros.

Chapter 2

Expresiones Regulares

Contents

2.1 Las Nociones y Algoritmos Básicos 25
2.1.1 Las Nociones 25
2.1.2 La Semántica de las expresiones regulares. 27
2.2 De RE's a RG's: Método de las Derivaciones 28
2.2.1 Derivación de Expresiones Regulares 28
2.2.2 Cómo no construir la Gramática 29
2.2.3 Derivadas Sucesivas: el Método de las derivaciones 31
2.3 De RG's a RE's: Uso del Lema de Arden 33
2.3.1 Ecuaciones Lineales. Lema de Arden 33
2.3.2 Sistema de Ecuaciones Lineales Asociado a una Gramática. 35
2.4 Problemas y Cuestiones. 37
2.4.1 Cuestiones Relativas a Lenguajes y Gramáticas. 37
2.4.2 Cuestiones Relativas a Expresiones Regulares. 37
2.4.3 Problemas Relativos a Lenguajes Formales y Gramáticas . 38
2.4.4 Problemas Relativos a Expresiones Regulares 40

El problema que se pretende resolver mediante la introducción de las expresiones regulares es el de obtener algún tipo de descriptores para los lenguajes generados por las gramáticas regulares (las gramáticas de Tipo 3 o Regulares en la jerarquía de Chomsky), además de utilizarlos en la notación EBNF.

2.1 Las Nociones y Algoritmos Básicos

2.1.1 Las Nociones

Siendo éste un curso de lenguajes formales, utilizaremos la metodología propia del área. Empezaremos definiendo las reglas de formación (la gramática) de las expresiones regulares. A continuación las dotaremos de significado (la semántica) y veremos los recursos que nos ofrece esta nueva herramienta.

Definición 26. *Sea Σ un alfabeto finito. Llamaremos expresión regular sobre el alfabeto Σ a toda palabra sobre el alfabeto Σ_1 definido por la siguiente igualdad:*

$$\Sigma_1 := \{'\emptyset', '\lambda', +, \cdot, (,), ^* \} \cup \Sigma,$$

conforme a las reglas siguientes:

- *Las siguientes son expresiones regulares:*
  - *El símbolo $'\emptyset'$ es una expresión regular,*
  - *el símbolo $'\lambda'$ es una expresión regular,*
  - *y el símbolo a es una expresión regular, para cualquier símbolo a en el alfabeto Σ,*
- *Si α y β son expresiones regulares, también lo son las construidas mediante las reglas siguientes:*
  - *$(\alpha + \beta)$ es una expresión regular,*
  - *$(\alpha \cdot \beta)$ es una expresión regular,*
  - *$(\alpha)^*$ es una expresión regular,*

Nota 27. *Por comodidad de la escritura (y sólo en el caso de que no haya ninguna posibilidad de ambigüedades) se suprimen los paréntesis y los símbolos de producto $(\cdot)$.*

Nota 28. *También por simplificación de la escritura escribiremos simplemente $\emptyset$ y λ en lugar de $'\emptyset'$ y $'\lambda'$ y siempre que no haya confusión entre su sentido como expresión regular y su uso habitual como conjunto vacío o como palabra vacía.*

Nota 29. *La anterior definición no es sino la definición de un lenguaje sobre el alfabeto Σ_1: el lenguaje formado por las expresiones regulares. Dicha gramática se pude representar mediante una única variable $\langle RE \rangle$, el alfabeto Σ_1 y las producciones siguientes:*

$$\langle RE \rangle \mapsto' \emptyset' \mid' \lambda' \mid a, \quad \forall a \in \Sigma.$$

$$\langle RE \rangle \mapsto (\langle RE \rangle + \langle RE \rangle) \mid (\langle RE \rangle \cdot \langle RE \rangle) \mid (\langle RE \rangle)^*.$$

Ejemplo 3. *Tomemos el alfabeto $\Sigma := \{a, b\}$. Son expresiones regulares las secuencias de símbolos (palabras) siguientes:*

$$a \cdot a + b^* a, ab^* ba, \ldots$$

No serán expresiones regulares cosas del tipo:

$$(+b^* \emptyset)^*, \ldots$$

2.1.2 La Semántica de las expresiones regulares.

A cada objeto sintáctico, como lo es una expresión regular, conviene añadirle el mecanismo de asignación de significado (semántica). En el caso de expresiones regulares asignaremos *un único significado* a cada expresión como *el lenguaje formal que describe*.

Definición 30. *Sea Σ un alfabeto finito. A cada expresión regular sobre el alfabeto α le asignaremos un lenguaje formal $L(\alpha) \subseteq \Sigma^*$ conforme a las siguientes reglas:*

- *En el caso de que α sea una palabra de longitud 1, seguiremos las reglas siguientes:*
  - *Si $\alpha = \emptyset$, entonces $L(\alpha) = \emptyset$,*
  - *Si $\alpha = \lambda$, entonces $L(\alpha) = \{\lambda\}$,*
  - *Si $\alpha = a \in \Sigma$, entonces $L(\alpha) = \{a\}$,*
- *Aplicando las reglas recursivas, si α y β son dos expresiones regulares sobre el alfabeto Σ usaremos las reglas siguientes:*
  - $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$,
  - $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$,
  - $L(\alpha^*) = L(\alpha)^*$.

Ejemplo 4. *A modo de ejemplo, sea $\alpha := 0^*10^*$ la expresión regular sobre el alfabeto $\Sigma := \{0, 1\}$. Entonces,*

$$L(0^*10^*) = L(0)^* \cdot L(1) \cdot L(0)^* = \{0^m 10^n \ : \ n, m \in \mathbb{N}\}.$$

Definición 31. *Diremos que dos expresiones regulares α y β son tautológicamente equivalentes (o, simplemente, equivalentes) si se verifica:*

$$L(\alpha) = L(\beta).$$

Escribamos $\alpha \equiv \beta$ para indicar equivalencia tautológica.

Algunas de las propiedades básicas de la asignación semántica de lenguajes a expresiones regulares se resumen en la siguiente Proposición, cuya demostración es completamente obvia.

Proposición 32 (Propiedades Básicas)**.** *Sea Σ un alfabeto finito, se verifican las siguientes propiedades para expresiones regulares α, β, γ sobre α:*

a. Asociativas.

$$\alpha \cdot (\beta \cdot \gamma) \equiv (\alpha \cdot \beta) \cdot \gamma, \qquad \alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma.$$

b. Conmutativa (sólo para $+$)[1]:

$$\alpha + \beta \equiv \beta + \alpha.$$

[1] Aunque la insistencia sea innecesaria, es común olvidar que $2 \cdot 3$ no es igual que $3 \cdot 2$. Cosas de malos hábitos.

c. Elementos Neutros:

$$\alpha + \emptyset \equiv \alpha, \quad \alpha \cdot \lambda \equiv \alpha, \quad \alpha \cdot \emptyset \equiv \emptyset.$$

d. Idempotencia:

$$\alpha + \alpha \equiv \alpha.$$

e. Distributivas:

$$\alpha \cdot (\beta + \gamma) \equiv \alpha \cdot \beta + \alpha \cdot \gamma.$$
$$(\alpha + \beta) \cdot \gamma \equiv \alpha \cdot \gamma + \beta \cdot \gamma.$$

f. Invariantes para *:

$$\lambda^* \equiv \lambda, \quad \emptyset^* \equiv \emptyset.$$

g. La notación α^+:

$$\alpha^* \cdot \alpha \equiv \alpha \cdot \alpha^* \equiv \alpha^+.$$

h. $\alpha^* = \lambda + \alpha^+$

i. Relación de * con la suma:

$$(\alpha + \beta)^* \equiv (\alpha^* \beta^*)^*.$$

2.2 De RE's a RG's: Método de las Derivaciones

2.2.1 Derivación de Expresiones Regulares

En esta Sección dedicaremos algún tiempo a fijar una de las operaciones básicas en el tratamiento de expresiones regulares: la derivación.

Definición 33. *Sea Σ un alfabeto finito, $a \in \Sigma$ un símbolo del alfabeto, y α una expresión regular sobre el alfabeto Σ. Llamaremos derivada de α con respecto al símbolo α a la expresión regular $\frac{\partial \alpha}{\partial a}$ definida mediante la regla recursiva siguiente:*

- *Para expresiones regulares de longitud 1, tenemos las definiciones siguientes:*

$$\frac{\partial \emptyset}{\partial a} = \emptyset, \quad \frac{\partial \lambda}{\partial a} = \emptyset, \quad \frac{\partial b}{\partial a} = \emptyset, \quad \forall b \in \Sigma,\ b \neq a.$$

$$\frac{\partial a}{\partial a} = \lambda.$$

- *Si α y β son dos expresiones regulares sobre Σ, definiremos:*

$$\frac{\partial(\alpha)^*}{\partial a} = \frac{\partial(\alpha)}{\partial a} \cdot \alpha^*,$$

$$\frac{\partial(\alpha + \beta)}{\partial a} = \frac{\partial \alpha}{\partial a} + \frac{\partial \beta}{\partial a},$$

$$\frac{\partial(\alpha \cdot \beta)}{\partial a} = \frac{\partial \alpha}{\partial a} \cdot \beta + t(\alpha) \frac{\partial \beta}{\partial a},$$

donde $t(\alpha)$ es la función dada por la identidad siguiente:

$$t(\alpha) := \begin{cases} \lambda & \text{si } \lambda \in L(\alpha), \\ \emptyset & \text{en caso contrario.} \end{cases}$$

Nota 34. *La derivada de una expresión regular con respecto a un símbolo de un alfabeto finito es, claramente, una derivada parcial y, por tanto, está perfectamente descrita mediante el símbolo $\frac{\partial \alpha}{\partial a}$. Sin embargo, el símbolo ∂ parece poner nerviosos a ciertos autores, por lo que también es costumbre (solamente entre los nerviosos) usar el símbolo menos correcto (pero menos enervante) $D_a(\alpha)$. Dejaremos que los alumnos reescriban la definición anterior con esta nueva notación. De ahora en adelante usaremos $D_a(\alpha)$.*

La propiedad fundamental por la cual derivar es una acción útil, viene dada por la siguiente Proposición (cuya prueba omitiremos por obvia).

Proposición 35. *Con las notaciones anteriores, para cada expresión regular α sobre un alfabeto Σ, la derivada $D_a(\alpha)$ es una expresión regular que verifica la siguiente propiedad:*

$$L(D_a(\alpha)) = \{\omega \in \Sigma^* \ : \ a\omega \in L(\alpha)\}.$$

Demostración. Como pista para la demostración, digamos que sale de manera inmediata a partir de la definición recursiva de expresiones regulares. □

Una identificación más clara de la relación de una palabra con sus derivadas viene dada por la siguiente Proposición (que resume la regla de Leibnitz para polinomios homogéneos multivariados).

Proposición 36 (Regla de Leibnitz para Expresiones Regulares)**.** *Dada una expresión regular α sobre un alfabeto finito Σ, supongamos que $\Sigma = \{a_1, \ldots, a_n\}$. Entonces,*

$$\alpha \equiv a_1 D_{a_1}(\alpha) + \cdots + a_n D_{a_n}(\alpha) + t(\alpha),$$

donde $t(\alpha)$ es la función definida anteriormente.

Demostración. Mediante la proposición anterior, basta con verificar a que las palabras en $L(\alpha)$ son de los tipos (obvios) siguientes: o empiezan por algún símbolo de $a_1 \in \Sigma$ (y, por tanto, están en $a_1 D_{a_1}(\alpha)$) o es la palabra vacía (y queda sumida en la expresión $t(\alpha)$). El caso restante es que no haya ninguna palabra en $L(\alpha)$ lo cual también queda expresado por la identidad y por $t(\alpha)$. □

2.2.2 Cómo no construir la Gramática

En esta Sección demostraremos que el lenguaje descrito por una expresión regular es un lenguaje regular, es decir, que existe una gramática regular que lo genera. Más aún, daremos un algoritmo que transforma expresiones regulares en gramáticas regulares respetando los lenguajes que describen/generan: es el *Método de las Derivaciones.*

Lema 37. *Sea L_1 y L_2 dos lenguajes (regulares) sobre el alfabeto Σ generados respectivamente por gramáticas $G_1 = (V_1, \Sigma, Q_1, P_1)$ y $G_2 = (V_2, \Sigma, Q_2, P_2)$, entonces $L_1 \cup L_2$ es también un lenguaje (regular) generado por una gramática. La gramática que genera la unión es una nueva gramática $G = (V, \Sigma, Q_0, P)$ dada por las reglas siguientes:*

a. *Al precio de renombrar las variables, podemos suponer que $V_1 \cap V_2 = \emptyset$ (es decir, G_1, G_2 no poseen símbolos no terminales comunes) y $P_1 \cap P_2 = \emptyset$.*

b. *Introducimos una nueva variable $Q_0 \notin V_1 \cup V_2$.*

c. *Finalmente, definimos $V := V_1 \cup V_2 \cup \{Q_0\}$.*

d. *Y definimos $P := P_1 \cup P_2 \cup \{Q_0 \mapsto Q_1 \mid Q_2\}$.*

Demostración. Con esta definición de la nueva gramática G es un mero ejercicio de demostración por inducción en el número de pasos de cálculo. □

Lema 38. *En el caso de unión finita $L = L_1 \cup \cdots \cup L_m$, el Lema anterior se puede extender de la forma obvia. Por tanto, la unión finita de lenguajes generados por gramáticas (resp. regulares) es un lenguaje generado por una gramática (resp. regulares).*

Lema 39. *Sea $L \subseteq \Sigma^*$ un lenguaje sobre el alfabeto Σ generado por una gramática (regular) $G := (V, \Sigma, q_0, P)$. Sea $a \in \Sigma$ un símbolo del alfabeto. Entonces, la siguiente gramática $G_a = (V_a, \Sigma, Q_a, P_a)$ genera el lenguaje $a \cdot L$:*

- *Sea Q_a una nueva variable (no presente en V) y definamos $V_a := V \cup \{Q_a\}$.*
- *Definamos $P_a := P \cup \{Q_a \mapsto aQ_0\}$.*

Demostración. De nuevo un mero ejercicio de demostración por inducción. Es importante señalar que si la gramática G es regular, la nueva gramática también es regular. □

Combinando la Proposición 36 con los lemas 38 y 39, uno pensaría en un argumento inductivo para generar un lenguaje dado por una expresión regular α a partir de sus derivadas. La idea, grosso modo, sería la siguiente:

Sea $L(\alpha)$ un lenguaje dado por una expresión regular α sobre un alfabeto Σ, supongamos que $\Sigma = \{a_1, \ldots, a_n\}$. entonces, la Regla de Leibnitz para expresiones regulares nos da la siguiente identidad:

$$L(\alpha) = a_1 \cdot L(D_{a_1}(\alpha)) \cup \cdots \cup a_n \cdot L(D_{a_n}(\alpha)) \cup t(\alpha).$$

A partir de esta identidad, uno pretende generar un árbol entre expresiones regulares y podría tratar de argumentar como sigue:

- Supongamos dadas gramáticas $G_1, \ldots, G_n$ que generan (respectivamente) los lenguajes $L(D_{a_1}(\alpha)), \ldots, L(D_{a_n}(\alpha))$.
- Utilizado el Lema 39, uno podría construir gramáticas $G'_1, \ldots, G'_n$ de tal modo que G'_i es la gramática que genera el lenguaje $a_i L(D_{a_i}(\alpha))$.

- Finalmente, utilizando el Lema 38 uno concluiría exhibiendo la gramática que genera el lenguaje $L(\alpha)$ a través de la identidad dada por la Regla de Leibnitz (Proposición 36).

El problema en esta forma de pensamiento es la "gradación" de las gramáticas. En esta propuesta hay implícitamente una suposición de que las expresiones regulares asociadas a las derivadas son "más pequeñas" que la expresión regular original. El concepto de "más pequeño" es inevitable para poder dar un argumento recursivo con esta construcción. Sin embargo, la intuición sobre las propiedades de las derivadas no debe confundirnos. La derivada de una expresión regular puede ser "más grande" (o de mayor "grado") que la expresión original, debido justamente al papel del operador *. Veamos algunos ejemplos:

Ejemplo 5. *Sea $\Sigma = \{a, b\}$ y consideremos la expresión regular $a^* \subseteq \Sigma^*$. Consideramos las derivadas $D_a(a^*) = a^*$, $D_b(a^*) = \emptyset$. Tendremos, por Leibnitz,*

$$\{a\}^* = L(a^*) = a \cdot L(a^*) + \emptyset + \{\lambda\}.$$

Claramente, la inducción pretendida nos dice que para hallar la gramática asociada a la expresión a^* **necesitamos calcular previamente la gramática asociada a la expresión a^*!**. *La respuesta a este dilema en este caso, sería la gramática siguiente:*

- *Dado que $\lambda \in L(a^*)$ escribamos la producción $q \mapsto \lambda$,*
- *Dado que $D_a(a^*) \neq \lambda, \emptyset$, escribamos la producción $q \mapsto aq$.*

Nótese que, en este ejemplo, hemos identificado la variable q con la expresión regular a^* y, hemos escrito la producción $q \mapsto aq$ porque $D_a(a^*) = a^*$.

Ejemplo 6. *En el anterior ejemplo, la expresión regular obtenida tras derivar no "crece" con respecto a la expresión regular original (en todo caso, se estabiliza). Pero es posible que se produzca un crecimiento (al menos en la longitud como palabra) y eso se muestra a través del ejemplo $(abc)^*$ de una expresión regular sobre el alfabeto $\Sigma = \{a, b, c\}$. Al derivar observamos:*

$$D_a((abc)^*) = bc(abc)^*,$$

cuya longitud es mayor que la longitud de la expresión regular original.

2.2.3 Derivadas Sucesivas: el Método de las derivaciones

Para resolver este problema acudiremos al análisis de las derivadas sucesivas de una expresión regular.

Definición 40 (Derivadas sucesivas (de una RE))**.** *Sea $\Sigma = \{a_1, \ldots, a_n\}$ un alfabeto finito, $\omega \in \Sigma^*$ una palabra sobre el alfabeto y α una expresión regular. Definiremos la derivada $D_\omega(\alpha)$ mediante el proceso siguiente:*

- *Si $\omega = \lambda$ es la palabra vacía, $D_\lambda(\alpha) = \alpha$.*

- *Si* $|\omega| = 1$ *(es una palabra de longitud* 1*) y, por tanto,* $\omega = a_i \in \Sigma$*, definimos* $D_\omega(\alpha) = D_{a_i}(\alpha)$*, conforme a la definición de derivada anterior.*

- *Si* $|\omega| = n \geq 2$ *(es una palabra de longitud* n*) y, por tanto, existe* $a_i \in \Sigma$ *y existe* $\omega_1 \in \Sigma^*$*, con* $|\omega_1| = n - 1$*, tal que*

$$\omega = a_i\omega_1,$$

definimos

$$D_\omega(\alpha) = D_{a_i}(D_{\omega_1}(\alpha)),$$

conforme a la definición recursiva para palabras de longitud $n - 1$*.*

Nota 41. *De nuevo la intuición puede hacer estragos, nótese que no hay conmutatividad de las derivadas (como sí ocurría en el caso de las derivadas parciales habituales). Es decir,* $D_{ab} \neq D_{ba}$*. Por poner un ejemplo, consideremos la expresión* $\alpha = aa^*bb^*$*. Tendremos,*

$$D_a(\alpha) = a^*bb^*, D_b(\alpha) = \emptyset.$$

Por tanto,

$$D_{ba}(\alpha) = D_b(D_a(\alpha)) = D_b(a^*bb^*) = b^*,$$

mientras que

$$D_{ab}(\alpha) = D_a(D_b(\alpha)) = D_a(\emptyset) = \emptyset.$$

El resultado crucial es el siguiente:

Proposición 42. *Sea* α *una expresión regular sobre un alfabeto finito* Σ *y sea* $Der(\alpha)$ *el conjunto de todas las derivadas sucesivas de* α *con respecto a palabras en* Σ^**. Esto es,*

$$Der(\alpha) := \{\beta \ : \ \exists\omega \in \Sigma^*, \ \beta = D_\omega(\alpha)\}.$$

Entonces, $Der(\alpha)$ *es un conjunto finito.*

Demostración. Se demostraría por inducción en la definición recursiva de la expresión regular. □

Nuestro propósito es construir un grafo con pesos asociado al conjunto de todas las derivadas de la expresión regular. Esto va a constituir la gramática buscada.

Proposición 43. *El algoritmo siguiente transforma toda expresión regular* α *en una gramática finita* G *que genera el lenguaje* $L(\alpha)$ *descrito por la expresión. En particular, los lenguajes descritos por expresiones regulares son lenguajes regulares.*

Demostración. La idea principal para realizar este algoritmo es la Regla de Leibnitz, combinando las gramáticas con los Lemas 38 y 39. □

Consideremos el siguiente algoritmo:

---

begin

INPUT: *Una expresión regular* α *sobre un alfabeto finito* Σ

- Hallar todos los elementos del conjunto $Der(\alpha) := \{D_\omega(\alpha) \ : \ \omega \in \Sigma^*\}$.
- Definir un conjunto finito V de variables, biyetable al conjunto $Der(\alpha)$. Sea $Q_0 \in V$ un elemento de ese conjunto de variables.
- Definir una biyección $E : Der(\alpha) \longrightarrow V$, tal que $E(\alpha) = Q_0$.
- Definir $P_1 := 1$ y

$$P_2 := \begin{cases} \{Q_0 \mapsto \lambda\}, & \text{si } \lambda \in L(\alpha) \\ \emptyset, & \text{en caso contrario} \end{cases}$$

while $P_2 \neq P_1$ **do**
$P_1 := P_2$
Para cada $\beta \in Der(\alpha)$ **do**
Para cada $a \in \Sigma$ **do**
Hallar $\gamma := D_a(\beta)$, $Q_1 := E(\gamma)$ y $Q_2 := E(\beta)$ en V.
Si $\lambda \in L(\gamma)$, hacer $P_2 := P_2 \cup \{Q_2 \mapsto a\}$.
Si $\gamma \neq \emptyset, \lambda$, hacer $P_2 := P_2 \cup \{Q_2 \mapsto aQ_1\}$.
next a
od
next β
od
od

- OUTPUT: La lista $[V, \Sigma, Q_0, P_2]$.

end

2.3 De RG's a RE's: Uso del Lema de Arden

2.3.1 Ecuaciones Lineales. Lema de Arden

La ecuaciones lineales en los lenguajes regulares juegan un papel muy importante. Estás nos posibilitarán probar que las palabras generadas por una gramática regular forman un lenguaje dado por una expresión regular. Empecemos con la definición.

Definición 44. *Llamaremos sistema de ecuaciones lineales en expresiones regulares a toda ecuación del tipo siguiente:*

$$\begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} + \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \tag{2.1}$$

donde los $\alpha_{i,j}$ y los β_k son expresiones regulares sobre un alfabeto Σ.

Una solución de uno de tales sistemas de ecuaciones es una lista $(\omega_1, \ldots, \omega_n)$ de expresiones regulares sobre el mismo alfabeto, tales que

$$\omega_i \equiv \alpha_{i,1} \cdot \omega_1 + \cdots + \alpha_{i,n} \cdot \omega_n + \beta_i,$$

donde $\equiv$ es la igualdad entre los lenguajes que describen (i.e. la igualdad tautológica de las expresiones regulares).
El objetivo de esta Subsección es la discusión del método obvio de resolución de este tipo de ecuaciones lineales. La clave para poder establecer lo obvio es un clásico resultado de Arden:

Definición 45. *Se denomina ecuación lineal fundamental en expresiones regulares a la ecuación lineal en una variable X siguiente:*

$$X = \alpha X + \beta,$$

donde α y β son expresiones regulares sobre un alfabeto finito Σ.

Lema 46 (Lema de Arden)**.** *Dada la ecuación fundamental siguiente:*

$$X = \alpha X + \beta,$$

donde α, β son expresiones regulares sobre un alfabeto Σ. Se verifican las propiedades siguientes:

a. *La ecuación fundamental anterior posee una solución única si y solamente si $\lambda \notin L(\alpha)$.*

b. *La expresión regular $\alpha^* \cdot \beta$ es siempre solución de la ecuación fundamental anterior.*

c. *Si $\lambda \in L(\alpha)$, para cualquier expresión regular γ, la expresión $\alpha^* \cdot (\beta + \gamma)$ es una solución de la ecuación fundamental*

Demostración. Aunque no se pretende dar una demostración completa del Lema, al menos señalaremos los hechos fundamentales.
El alumno puede ver fácilmente que cualquier expresión regular que sea solución debe contener al lenguaje $L(\alpha^*\beta)$. Otra observación trivial es que cualquier palabra del lenguaje generado por una solución debe de estar en el lenguaje generado por β o es la concatenación de dos palabras, la primera en el lenguaje generado por α y la segunda en el lenguaje generado por X.
También nótese que si α es una expresión regular, se tiene que

$$L(\alpha \cdot \alpha^*) = L(\alpha)^+.$$

Es decir, $\alpha \cdot \alpha^* \equiv \alpha^+$. Ahora bien, nótese que $L(\alpha^*) = L(\alpha^+)$ si y solamente si $\lambda \in L(\alpha)$.
Del mismo modo, consideremos una expresión regular γ cualquiera, tendremos:

$$\alpha \cdot \alpha^* \cdot (\beta + \gamma) + \beta \equiv \alpha^+ \cdot \beta + \beta + \alpha^+ \cdot \gamma \equiv (\alpha^+ + \lambda) \cdot \beta + \alpha^+ \cdot \gamma \equiv \alpha^* \cdot \beta + \alpha^+ \cdot \gamma.$$

Por su parte, $\alpha^* \cdot (\beta + \gamma) = \alpha^* \cdot \beta + \alpha^* \cdot \gamma$. Esto nos da inmediatamente que si $\alpha^* \equiv \alpha^+$ o si $\gamma = \emptyset$ tenemos la equivalencia. □
Este simple Lema es la base para el algoritmo obvio de sustitución, es decir, eligiendo una variable y sustituyéndola en las demás ecuaciones. Formalmente, esto se expresa de la siguiente manera.

Proposición 47. *Toda ecuación lineal en expresiones regulares del tipo de la Definición 44 posee solución, que es una lista de expresiones regulares sobre el mismo alfabeto.*

Demostración. El algoritmo se divide en las dos fases obvias: triangulación/reducción (a través del Lema de Arden) y levantamiento (invirtiendo las expresiones ya despejadas). Los detalles del algoritmo se dejan como ejercicio al alumno.

- **Triangulación:** Seguiremos la notación 2.1 y procederemos utilizando inducción. El caso $n = 1$ se resuelve mediante el Lema de Arden. Para el caso $n > 1$, usaremos un doble paso:
  - *Despejar.* Podemos despejar X_n en la última ecuación, mediante la expresión siguiente:
    $$X_n := \alpha_{n,n}^* R_n, \tag{2.2}$$
    donde $R_n := \sum_{j=1}^{n-1} \alpha_{n,j} X_j + \beta_n$.
  - *Sustituir.* Podemos sustituir la expresión anterior en el resto de las ecuaciones obteniendo un nuevo sistema de $(n-1)$ ecuaciones en $(n-1)$ variables. Este sistema viene dado, obviamente, por las expresiones siguientes para $1 \leq i \leq n-1$:
    $$X_i := \left(\sum_{j=1}^{n-1} \left(\alpha_{i,j} + \alpha_{i,n} \alpha_{n,n}^* \alpha_{n,j} \right) X_j \right) + \left(\beta_i + \alpha_{n,n}^* \beta_n \right).$$
- **Levantamiento.** Una vez llegados al caso $n = 1$, se obtiene una expresión regular válida para X_1 y se procede a levantar el resto de las variables usando las expresiones obtenidas en la fase de despejado (expresiones (2.2)).

□

2.3.2 Sistema de Ecuaciones Lineales Asociado a una Gramática.

Comenzaremos asociando a cada gramática regular $G = (V, \Sigma, Q_0, P)$ un sistema de ecuaciones lineales en expresiones regulares mediante la regla siguiente:

- Supongamos $V = \{Q_0, \ldots, Q_n\}$ es el conjunto de los símbolos no terminales, que supondremos de cardinal $n + 1$. Definamos un conjunto de variables $\{X_0, \ldots, X_n\}$ con el mismo cardinal y con la asignación $q_i \longmapsto X_i$ como biyección.

- Definamos para cada i, $0 \leq i \leq n$, la expresión regular β_i mediante la construcción siguiente. Consideremos todas las producciones que comienzan en la variable q_i y terminan en elementos de $\Sigma \cup \{\lambda\}$. Supongamos que tales producciones sean
$$Q_i \mapsto a_1 \mid \ldots \mid a_r.$$
Definimos[2]
$$\beta_i := a_1 + \ldots + a_r.$$
Si no hubiera ninguna producción del tipo $Q_i \mapsto a \in \Sigma \cup \{\lambda\}$, definiremos $\beta_i := \emptyset$.

- Para cada i y para cada j, definiremos el coeficiente $\alpha_{i,j}$ del modo siguiente. Consideremos todas las producciones que comienzan en el símbolo no terminal Q_i e involucran al símbolo no terminal Q_j. Supongamos que tales producciones sean:
$$Q_i \mapsto a_1 Q_j \mid \cdots \mid a_r Q_j,$$
con $a_k \in \Sigma \cup \{\lambda\}$. Entonces definiremos
$$\alpha_{i,j} := a_1 + \cdots + a_r.$$
Si no hubiera ninguna de tales producciones, definiremos $\alpha_{i,j} := \emptyset$.

Definición 48 (Sistema asociado a una gramática)**.** *Dada una gramática $G = (V, \Sigma, Q_0, P)$ llamaremos sistema asociado a G y lo denotaremos por $\mathcal{S}(G)$ al sistema:*
$$\mathcal{S}(G) := \left\{ \begin{pmatrix} X_0 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} \alpha_{0,1} & \cdots & \alpha_{0,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,0} & \cdots & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} X_0 \\ \vdots \\ X_n \end{pmatrix} + \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix} \right\},$$
dado por las anteriores reglas de construcción.

Proposición 49. *Con las anteriores notaciones, sea $(\alpha_0, \ldots, \alpha_n)$ una solución del sistema $S(G)$ asociado a una gramática G. Entonces, $L(\alpha_0)$ es el lenguaje generado por la gramática G.*

Demostración. La idea de la demostración es que estamos asociando una expresión regular a cada variable. La variable X_i es la expresión regular de las palabras que se pueden generar a través de derivaciones empezando por la variable Q_i. Por esa razón la solución de nuestro problema es encontrar X_0. A partir de esta idea, la demostración se realiza por inducción. □

Teorema 50. *Los lenguajes regulares son los descritos por las expresiones regulares. Es decir, todo lenguaje descrito por una expresión regular es el lenguaje generado por alguna gramática regular y, recíprocamente, todo lenguaje generado por alguna gramática regular puede ser descrito por alguna expresión regular. Además, existen algoritmos que transforman RE's en RG's y recíprocamente.*

Demostración. Basta con combinar los algoritmos descritos en las Proposiciones 49 y 43. □

[2]Note el lector que algún a_i puede ser λ.

2.4 Problemas y Cuestiones.

2.4.1 Cuestiones Relativas a Lenguajes y Gramáticas.

Cuestión 1. *Se considera una gramática sobre el alfabeto* $\Sigma := \{a, b\}$*, cuyas producciones vienen dadas por*

$$Q_0 \longrightarrow \lambda \mid aQ_0a \mid bQ_0b.$$

Decidir si el lenguaje generado por esa gramática es el conjunto de los palíndromos sobre Σ*.*

Cuestión 2. *Demostrar la falsedad de las afirmaciones dando el código java sobre las siguientes afirmaciones (se deja a un lado la funcionalidad del programa, por ahora sólo se requiere si el compilador devolverá un error al tratar de compilarlo):*

- *Cambiar una orden por otra correcta no provoca errores de compilación.*
- *Trabajando con paréntesis () y corchetes [], no hay que tener más cuidado que cuando abramos alguno, se cierre en la misma orden.*

Cuestión 3. *Si el sistema de producciones de una gramática no posee ninguna transformación del tipo* $A \longrightarrow a$*, ¿ podemos asegurar que no es una gramática regular?.*

Cuestión 4. *El lenguaje sobre el alfabeto* $\{0,\ 1\}$ *de las palabras que no contienen a* 00 *como subpalabra, ¿es un lenguaje regular?.*

Cuestión 5. *Dados dos lenguajes* L_1 *y* L_2 *sobre el alfabeto* $\{a, b\}$*, ¿ podemos asegurar que se verifica la siguiente igualdad*

$$(L_1 \cdot L_2)^R = L_1^R \cdot L_2^R?$$

Cuestión 6. *Dar una definición inductiva (recursiva) de la transformación* $w \longmapsto w^R$ *que revierte las palabras.*

2.4.2 Cuestiones Relativas a Expresiones Regulares.

Cuestión 7. *Se dice que una expresión regular* α *está en forma normal disyuntiva si*

$$\alpha = \alpha_1 + \cdots + \alpha_n,$$

donde las expresiones regulares $\alpha_1, \ldots,\ \alpha_n$ *no involucran el operador* $+$*. Decidir si la siguiente expresión regular esta en forma disyuntiva ó encontrar una forma de ponerla en forma disyuntiva:*

$$(0 + 00 + 10)^*,$$

con $\Sigma = \{0,\ 1\}$*.*

Cuestión 8. *Decidir si es verdadera la siguiente igualdad de expresiones regulares:*

$$(a + b)^* = (a^* + b^*)^*.$$

Cuestión 9. *¿Pertenece la palabra acdcdb al lenguaje descrito por la expresión regular siguiente:*

$$\alpha = (b^* a^* (cd)^* b)^* + (cd)^*?.$$

Cuestión 10. *Sea L el lenguaje sobre el alfabeto $\{a, b\}$ formado por todas las palabras que contienen al menos una aparición de la palabra b. ¿Es L el lenguaje descrito por la expresión regular siguiente*

$$\alpha := a^* (ba^*)^* bb^* (b^* a^*)^*?.$$

Cuestión 11. *Dada cualquier expresión regular α, ¿Se cumple $\alpha^* \alpha = \alpha^*$?.*

Cuestión 12. *Dadas tres expresiones regulares α, β, γ, ¿Es cierto que $\alpha + (\beta \cdot \gamma) = (\alpha + \beta) \cdot (\alpha + \gamma)$?.*

Cuestión 13. *¿Es siempre la derivada de una expresión regular otra expresión regular?.*

2.4.3 Problemas Relativos a Lenguajes Formales y Gramáticas

Problema 1. *Sea $L := \{\lambda, a\}$ un lenguaje sobre el alfabeto $\Sigma := \{a, b\}$. Hallar L^n para los valores $n = 0, 1, 2, 3, 4$. ¿Cuántos elementos tiene L^n?.*

Problema 2. *Dados los lenguajes $L_1 := \{a\}$ y $L_2 := \{b\}$ sobre el mismo alfabeto anterior, describir $(L_1 \cdot L_2)^*$ y $(L_1 \cdot L_2)^+$. Buscar coincidencias.*

Problema 3. *Probar que la concatenación de los lenguajes no es distributiva con respecto a la intersección de lenguajes.*

Problema 4. *Probar que la longitud $|\cdot| : \Sigma^* \longrightarrow \mathbb{N}$ es un morfismo de monoides suprayectivo, pero no es un isomorfismo excepto si $\sharp(\Sigma) = 1$.*

Problema 5. *Dado el alfabeto $\Sigma = \{0, 1\}$, se consideran los siguientes dos lenguajes:*

$$L_1 := \{\omega \in \Sigma^* \ : \ \sharp \text{ (ceros en } \omega) \in 2\mathbb{Z}\}.$$

$$L_2 := \{\omega \in \Sigma^* \ : \ \exists\, n \in \mathbb{N},\ \omega = 01^n\}.$$

Demostrar que $L_1 \cdot L_2$ es el lenguaje L_3 siguiente:

$$L_3 := \{\omega \in \Sigma^* \ : \ \sharp \text{ (ceros en } \omega) \in 2\mathbb{Z} + 1\}.$$

Problema 6. *Sea $G = (\{Q_0\}, \{a, b\}, Q_0, P)$ una gramática libre de contexto dada por las producciones:*

$$Q_0 \longrightarrow aQ_0b \ \mid \ \lambda.$$

Probar que $L(G)$ es el lenguaje definido por

$$L := \{a^n b^n \ : \ n \in \mathbb{N}\}.$$

Problema 7. *Sea* $L := \{a^n b^n c^n \ : \ n \in \mathbb{N}\}$ *un lenguaje sobre el alfabeto* $\Sigma = \{a, b, c\}$. *Hallar una gramática* G *tal que* $L(G) = L$. *Clasificar* G *dentro de la jerarquía de Chomsky.*

Problema 8. *Hallar una gramática libre de contexto (no regular) y otra equivalente regular para cada uno de los dos lenguajes siguientes:*

$$L_1 := \{ab^n a \ : \ n \in \mathbb{N}\},$$

$$L_2 := \{0^n 1 \ : \ n \in \mathbb{N}\}.$$

Problema 9. *Hallar gramáticas que generen los siguientes lenguajes:*

$$L_1 := \{0^m 1^n \ : \ [m, n \in \mathbb{N}] \wedge [m \geq n]\},$$

$$L_2 := \{0^k 1^m 2^n \ : \ [n, k, m \in \mathbb{N}] \wedge [n = k + m]\}.$$

Problema 10. *Dado el lenguaje* $L := \{z \in \{a, b\}^* \ : \ \exists w \in \{a, b\}^*, \text{ con } z = ww\}$, *hallar una gramática que lo genere.*

Problema 11. *Clasificar las siguientes gramáticas en términos de la jerarquía de Chomsky. Tratar de analizar los lenguajes generados por ellas y definirlos por compresión.*

a. $P := \{Q_0 \to \lambda \mid A, A \to c \mid AA\}$, $V := \{Q_0, A\}$, $\Sigma := \{c\}$.

b. $P := \{Q_0 \to \lambda \mid A, A \to Ad \mid cA \mid c \mid d\}$, $V := \{Q_0, A\}$, $\Sigma := \{c, d\}$.

c. $P := \{Q_0 \to c \mid Q_0 c Q_0\}$, $V := \{Q_0\}$, $\Sigma := \{c\}$.

d. $P := \{Q_0 \to c \mid AcA, A \to cc \mid cAc, cA \to cQ_0\}$, $V := \{Q_0, A\}$, $\Sigma := \{c\}$.

e. $P := \{Q_0 \to AcA, A \to 0, Ac \to AAcA \mid ABc \mid AcB, B \to B \mid AB\}$, $V := \{Q_0, A, B\}$, $\Sigma := \{0, c\}$.

Problema 12. *Sea* G *la gramática dada por las siguientes producciones:*

$$Q_0 \to 0B \mid 1A,$$

$$A \to 0 \mid 0Q_0 \mid 1AA,$$

$$B \to 1 \mid 1Q_0 \mid 0BB.$$

Siendo $V := \{Q_0, A, B\}$ *y* $\Sigma := \{0, 1\}$, *probar que*

$$L(G) := \{\omega \in \{0,1\}^* \ : \ \sharp(\text{ceros en } \omega) = \sharp(\text{unos en } \omega) \ \wedge \ |\omega| \geq 0\}.$$

Problema 13. *Probar que si* L *es el lenguaje dado por la siguiente igualdad:*

$$L := \{\omega \in \{0,1\}^* \ : \ \sharp(\text{ceros en } \omega) \neq \sharp(\text{unos en } \omega)\},$$

entonces $L^* = \{0, 1\}^*$.

Problema 14. *Sea* $L \subseteq \{a, b\}^*$ *el lenguaje dado por la siguiente definición:*

- $\lambda \in L$,
- *Si* $\omega \in L$, *entonces* $a\omega b \in L$ *y* $b\omega a \in L$,
- *Si* $x, y \in L$, *entonces* $xy \in L$.

Describir el lenguaje y definirlo por comprensión.

Problema 15. *Probar que si* L *es generado por una gramática regular a izquierda, entonces* L^R *es generado por una gramática regular a derecha.*

2.4.4 Problemas Relativos a Expresiones Regulares

Problema 16. *Dadas* α, β *dos expresiones regulares. Probar que si* $L(\alpha) \subseteq L(\beta)$, *entonces* $\alpha + \beta \equiv \beta$.

Problema 17. *Dada la expresión regular* $\alpha = a + bc + b^3a$, *¿Cuál es el lenguaje regular* $L(\alpha)$ *descrito por ella?. ¿Cuál es la expresión regular que define el lenguaje* $\{a, b, c\}^*$*?*

Problema 18. *Simplificar la expresión regular* $\alpha = a + a(b + aa)(b^*(aa)^*)^*b^* + a(aa + b)^*$, *usando las equivalencias (semánticas) vistas en clase.*

Problema 19. *Calcular la derivada* $D_{ab}(\alpha) = D_a(D_b(\alpha))$, *siendo* $\alpha := a^*ab$.

Problema 20. *Comprobar que* $x := \alpha^*\beta$ *es una solución para la ecuación fundamental* $\alpha \equiv \alpha x + \beta$.

Problema 21. *Simplificar la expresión regular* $\alpha := 1^*01^* \left(01^*01^*0 + 1\right)^* 01^* + 1^*$.

Problema 22. *Hallar la expresión regular* α *asociada a la siguiente gramática (por el método de las ecuaciones lineales):*

$$Q_0 \to aA \mid cA \mid a \mid c,$$

$$A \to bQ_0.$$

Aplicar el método de las derivadas a α *y comparar los resultados.*

Problema 23. *Hallar la gramática que genera el lenguaje descrito por la siguiente expresión regular:*

$$\alpha := (b + ab^+a)^*ab^*.$$

Problema 24. *Comprobar la equivalencia tautológica*

$$(b + ab^*a)^*ab^* \equiv b^*a(b + ab^*a)^*.$$

Problema 25. *Dada la expresión regular* $\alpha := (ab + aba)^*$, *hallar una gramática que genere el lenguaje* $L(\alpha)$.

Problema 26. *Dada la expresión regular* $\alpha := a(bc)^*(b + bc) + a$, *hallar una gramática* G *que genere el lenguaje* $L(\alpha)$. *Construir el sistema* $S(G)$ *asociado a la gramática calculada, resolverlo y comparar los resultados.*

Problema 27. *Hallar la expresión regular α asociada a la siguiente gramática:*

$$Q_0 \to bA \mid \lambda,$$

$$A \to bB \mid \lambda,$$

$$B \to aA.$$

Aplicar el método de las derivadas a α y comparar los resultados.

Problema 28. *Idem con la gramática:*

$$Q_0 \to 0A \mid 1B \mid \lambda,$$

$$A \to 1A \mid 0B,$$

$$B \to 1A \mid 0B \mid \lambda.$$

Problema 29. *Probar que si α es una expresión regular tal que $\alpha^2 \equiv \alpha$, entonces $\alpha^* = \alpha + \lambda$.*

Problema 30. *Probar que si α es una expresión regular se cumple $\alpha^* \equiv \alpha^*\alpha + \lambda$.*

Problema 31. *Hallar dos expresiones regulares distintas que sean solución de la siguiente ecuación lineal*

$$(a + \lambda)X = X.$$

Problema 32. *Las Expresiones Regulares Avanzadas son expresiones regulares añadiendo diferentes operadores. Se utilizan en lenguajes de programación como Perl para búsquedas dentro de texto. Los operadores añadidos son los siguientes:*

- *operador de rango: Para las letras [a..z], significa que cualquier letra del rango es correcta.*
- *operador ?: Este operador aplicado a una expresión regular entre paréntesis indica que necesariamente encaja en este esquema.*
- *operador /i: Este operador, indica el símbolo en la posición i de la palabra, por ejemplo /1 indica el primer símbolo de la palabra.*

Mostrar como transformar las expresiones regulares avanzadas a expresiones regulares. Aplicarlo al siguiente caso,

$$[a..c]([C..E]^*)?/1.$$

Chapter 3

Autómatas Finitos

Contents

3.1 Introducción: Correctores Léxicos o Morfológicos 43
3.2 La Noción de Autómata 44
3.2.1 Sistema de Transición de un autómata: 45
3.2.1.1 Representación Gráfica de la Función de Transición. 47
3.2.1.2 Lenguaje Aceptado por un Autómata 48
3.3 Determinismo e Indeterminismo 49
3.3.1 El Autómata como Programa 49
3.3.2 Autómatas con/sin $\lambda-$Transiciones. 49
3.3.2.1 Grafo de $\lambda-$transiciones. 50
3.3.3 Determinismo e Indeterminismo en Autómatas 51
3.4 Lenguajes Regulares y Autómatas. 52
3.4.1 Teorema de Análisis de Kleene 52
3.4.2 Teorema de Síntesis de Kleene 53
3.5 Lenguajes que no son regulares 56
3.5.1 El Palíndromo no es un Lenguaje Regular. 59
3.6 Minimización de Autómatas Deterministas 60
3.6.1 Eliminación de Estados Inaccesibles. 61
3.6.2 Autómata Cociente 61
3.6.3 Algoritmo para el Cálculo de Autómatas Minimales. 62
3.7 Cuestiones y Problemas. 64
3.7.1 Cuestiones. 64
3.7.2 Problemas 66

3.1 Introducción: Correctores Léxicos o Morfológicos

La siguiente etapa, que constituye un buen entrenamiento para las máquinas de Turing, son los autómatas finitos. Los autómatas finitos corresponden a correctores

ortográficos. Se trata de la vieja tarea del maestro de primaria, corrigiendo los dictados, esto es, evaluando la presencia de errores ortográficos. El maestro no se ocupa de la corrección sintáctica del dictado (es él quien ha dictado las palabras y su secuencia, incluyendo signos ortográficos) sino solamente de los errores de transcripción y, por tanto, errores en la escritura morfológica o léxica. El otro ejemplo son los populares `Spell Checkers`, sobre todo si no tienen en cuenta elementos sintácticos del texto que corrigen (concordancias de género, número, subordinadas...).
En términos informáticos, los autómatas finitos se usan para corregir (o señalar) los lugares en los que la morfología de un lenguaje de programación no ha sido respetada. Si, por ejemplo, alguien elabora un pequeño programa en C, Maple, Matlab o, simplemente, un documento Textures, como parte del proceso de compilación existe un autómata finito que detecta la presencia de errores y, si encuentra alguno, salta mostrando dónde aparece.
El gran impulsor de la Teoría de Autómatas fue J. von Neumann. Este matemático, gastó buena parte de los últimos años de su vida en el desarrollo de la teoría de autómatas y, durante la Segunda Guerra Mundial, en el desarrollo de los computadores electrónicos de gran tamaño que fructificó en la aparición del ENIAC (un ordenador para calcular rápidamente trayectorias balísticas que fue financiado por el ejército de los Estados Unidos y finalizado en 1948 [1]).

3.2 La Noción de Autómata

Formalmente, se definen como sigue:

Definición 51. *Llamaremos autómata finito indeterminístico a todo quíntuplo* $A := (Q, \Sigma, q_0, F, \delta)$ *donde:*

- Σ *es un conjunto finito (alfabeto),*
- Q *es un conjunto finito cuyos elementos se llaman estados y que suele denominarse espacio de estados,*
- q_0 *es un elemento de* Q *que se denomina estado inicial,*
- F *es un subconjunto de* Q*, cuyos elementos se denominan estados finales aceptadores,*
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow Q$ *es una correspondencia que se denomina función de transición.*

Si δ *es aplicación, el autómata se denomina determinístico.*

Nota 52. *En el caso indeterminístico, uno podría considerar la transición* δ *no como una correspondencia*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow Q$$

[1] La historia del diseño y puesta en marcha del ENIAC y las personas involucradas puede seguirse en la página web *http://ftp.arl.mil/ ~mike/comphist/eniac-story.html* .

sino como una aplicación

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q),$$

donde $\mathcal{P}(Q)$ es el conjunto de todos los subconjuntos del espacio de estados. Así, por ejemplo, si (q, a) no está en correspondencia (vía δ) con ningún estado, podríamos haber escrito $\delta(q, a) = \emptyset$.
Sin embargo, mantendremos la notación (incorrecta, pero cómoda) $\delta(q, a) = p$ para indicar que el estado $p \in Q$ están en correspondencia con el par (q, a) a través de la correspondencia δ. Así, por ejemplo, escribiremos $\neg\exists p \in Q,\ \delta(q, a) = p$ (o, simplemente, $\neg\exists\delta(q, a)$) para denotar que no hay ningún estado de Q en correspondencia con (q, a) a través de δ. Del mismo modo, $\exists p \in Q,\ \delta(q, a) = p$ (o, simplemente, $\exists\delta(q, a)$) en el caso contrario.

Para ver la acción dinámica asociada a un autómata, definamos su sistema de transición.

3.2.1 Sistema de Transición de un autómata:

Sea dado el autómata $A := (Q, \Sigma, q_0, F, \delta)$

- $S := Q \times \Sigma^*$ es el espacio de configuraciones,
- La transición $\rightarrow_A \subseteq S \times S$ se define por las reglas siguientes:

$$(q, x) \rightarrow_A (q', x') \ \Leftrightarrow\ \exists\alpha \in \Sigma \cup \{\lambda\},\ \ x = \alpha x',\ \ q' = \delta(q, x)$$

Para interpretar mejor el proceso, hagamos nuestra primera descripción gráfica. Las palabras del alfabeto Σ^* se pueden imaginar como escritas en una cinta infinita, dividida en celdas en cada una de las cuales puedo escribir un símbolo de Σ.

$$\underline{\overline{\mid x_1 \mid x_2 \mid x_3 \mid \cdots}}$$

Hay una unidad de control que ocupa diferentes posiciones sobre la cinta y que dispone de una cantidad finita de memoria en la que puede recoger un estado de Q:

$$\underline{\overline{\mid x_1 \mid x_2 \mid x_3 \mid \cdots}}$$
$$\uparrow$$
$$\underline{\overline{\mid q \mid}}$$

Las configuraciones de S sólo representan el momento instantáneo (*snapshot*) de cálculo correspondiente. Así, dada una palabra $x = x_1 \cdots x_n \in \Sigma^*$ el autómata A computa sobre esta palabra de la manera siguiente:

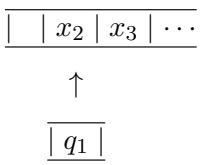
- Inicializa $(q_0, x) \in S$, es decir

$$\underline{\overline{\mid x_1 \mid x_2 \mid x_3 \mid \cdots}}$$
$$\uparrow$$
$$\underline{\overline{\mid q_0 \mid}}$$

- $q_1 := \delta(q_0, x_1)$, $x^{(1)} := x_2 \cdots x_n \in \Sigma^*$,

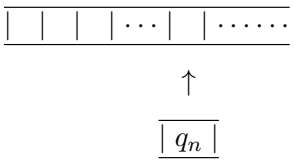
$$(q_0, x) \to_A (q_1, x^{(1)})$$

Gráficamente, borramos el contenido de la primera celda, cambiamos el estado en la unidad de control de q_0 (estado inicial) a q_1 y movemos la unidad de control un paso a la derecha:



- El proceso continúa hasta que nos quedamos sin palabra, i.e. llegamos a la configuración $(q_{n-1}, x^{(n)}) \in S$, donde $x^{(n)} := x_n$ es una palabra de longitud 1. Sea $q_n := \delta(q_{n-1}, x_n)$ y λ la palabra vacía y tenemos la sucesión de computación:

$$(q_0, x) \to_A (q_1, x^{(1)}) \to_A \cdots \to_A (q_{n-1}, x^{(n)}) \to_A (q_n, \lambda)$$



Ejemplo 7. *Consideremos el siguiente autómata $A = (Q, \Sigma, q_0, F, \delta)$. Donde,*

- $\Sigma = \{a, b\}$.
- $Q := \{q_0, q_1, q_2, q_3\}$.
- $F := \{q_2\}$.

Para la función de transición δ elegiremos una representación a través de una tabla:

δ	**a**	**b**
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_3	q_3

Esta tabla debe interpretarse como $\delta(q_i, x)$ es el estado que aparece en la fila q_i y columna x. Revisemos la computación del autómata A sobre un par de entradas:

Sea $x = aabbb \in \Sigma^*$ y veamos cómo funciona nuestro autómata:

$$(q_0, aabbb) \to_A (q_1, abbb) \to_A (q_1, bbb) \to_A$$

$$\to_A (q_2, bb) \to_A (q_2, b) \to_A (q_2, \lambda)$$

Y la palabra aabbb es aceptada.
Tomemos la palabra $y = baba \in \Sigma^*$ y tratemos de seguir los cálculos de nuestro autómata:

$$(Q_0, baba) \to_A (Q_3, aba) \to_A (Q_3, ba) \to_A (Q_3, a) \to_A (Q_3, \lambda)$$

y la palabra baba no es aceptada por nuestro autómata.

El autómata ya presenta una primera aproximación a las máquinas de Turing. Procesa listas y va corrigiendo la palabra dada. Si, al final del proceso, el autómata alcanza una configuración final aceptadora, es porque la palabra dada en la configuración inicial era correcta. En caso contrario rechaza. Así surge la primera noción de problema susceptible de ser tratado computacionalmente.

Definición 53. *Dado un autómata A en las notaciones anteriores y una palabra $\omega \in \Sigma^*$, definimos la correspondencia*

$$\delta^* : Q \times \Sigma^* \longrightarrow \mathcal{P}(Q)$$

dada por:

- $\delta^*(q, a) = \delta(q, a)$ *si* $a \in \Sigma \cup \{\lambda\}$.
- $\delta^*(q, aw) = \{\delta^*(q', \omega) \quad \forall q' \in \delta(q, a)\ \}$.

3.2.1.1 Representación Gráfica de la Función de Transición.

Una forma estética, pero no siempre conveniente a la hora de manipular autómatas relativamente grandes, es la representación de sistemas de transición mediante grafos con aristas etiquetadas (pesos), un ejemplo se puede ver en la Figura 3.1. Las reglas son las siguientes:

- Los nodos del grafo están dados por los estados del grafo. Cada nodo está rodeado de, al menos, una circunferencia.
- Los nodos finales aceptadores del grafo son aquellos que están rodeados por dos circunferencias, el resto de los nodos aparecen rodeados de una sola circunferencia.
- Dada una transición $\delta(q, z) = p$, asignaremos la arista del grafo (q, p) con etiqueta z.
- Hay una arista sin entrada, cuya salida es el nodo asociado al estado inicial.

Usaremos más habitualmente la representación de las funciones de transición bien mediante listas o bien mediante tablas.

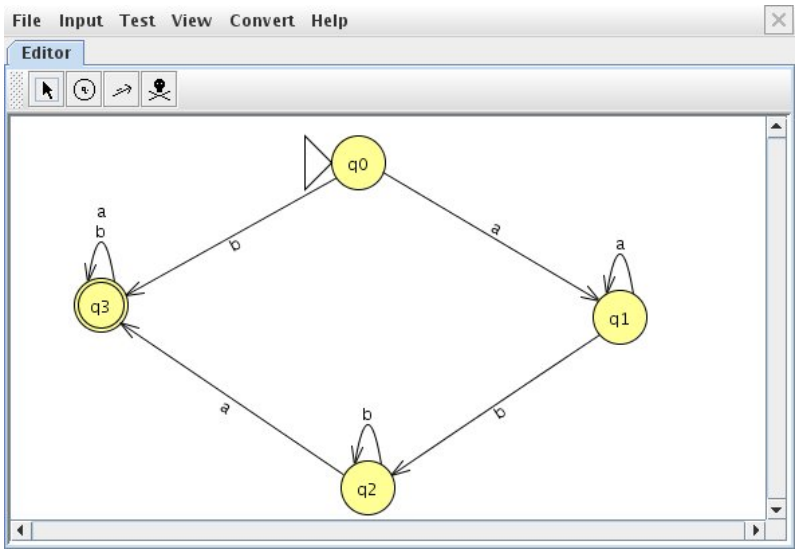


Figure 3.1: Representación gráfica del autómata.

3.2.1.2 Lenguaje Aceptado por un Autómata

Definición 54. *Llamaremos lenguaje aceptado por un autómata A al conjunto de palabras $\omega \in \Sigma^*$ tales que $\delta(q_0, \omega) \in F$, es decir al conjunto de palabras tales que se alcanza alguna configuración final aceptadora.*

En términos de la Definición 53, podremos también escribir:

$$L(A) := \{x \in \Sigma^* \ : \ \delta^*(q, x) \cap F \neq \emptyset\}.$$

Podemos interpretar un autómata como un evaluador de la función característica de un subconjunto de $L \subseteq \Sigma^*$:

$$\chi_L : \Sigma^* \longrightarrow \{0, 1\}$$

Los autómatas deterministas directamente sirven para evaluar χ_L y la interpretación es la obvia en términos de pregunta respuesta:

INPUT: Una palabra $\omega \in \Sigma^*$
OUTPUT:

- 1 si el autómata llega a una configuración final aceptadora (i.e., $\delta(q_0, \omega) \in F$).
- 0 si el autómata llega a una configuración final no aceptadora (i.e., $\delta(q_0, \omega) \in Q \setminus F$).

Una buena referencia sobre autómatas es el texto [Davis-Weyuker, 94], donde también se pueden encontrar ejemplos sencillos que ayuden al alumno a asimilar la noción.

3.3 Determinismo e Indeterminismo

3.3.1 El Autómata como Programa

Una manera bastante natural de interpretar el autómata finito es usar un pseudo–código para expresar un autómata como un programa con un **while** basado en el sistema de transición anterior. Informalmente, sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata. El programa (algoritmo) que define es el dado por la siguiente descripción:

INPUT: $x \in \Sigma^*$ (una palabra sobre el alfabeto).

Initialize: $I := (q_0, x)$ (la configuración inicial sobre x)

while $I \notin F \times \{\lambda\}$ **do**
    if $I = (q, x_1x')$, $x_1 \in \Sigma \cup \{\lambda\}$, $x_1x' \neq \lambda$, **then** $I := (\delta(q, x_1), x')$
        else OUPUT **NO**
    fi
od
OUTPUT **YES**

Nótese que hemos introducido deliberadamente un pseudo–código que no necesariamente termina en todos los inputs. Esto es por analogía con las máquinas de Turing y el estudio de los lenguajes recursivamente enumerables y recursivos. Aquí, el pseudo–código tiene una interpretación directa y natural en el caso determinístico y genera una forma imprecisa en el caso indeterminístico. Esta interpretación como programa (determinístico) de este pseudo–código depende esencialmente de la ausencia de dos obstrucciones:

- La presencia de $\lambda-$transiciones, esto es, de transiciones de la forma $\delta(q, \lambda)$ que pueden hacer que caigamos en un ciclo infinito.
- La indefinición de $I = (\delta(q, x_1), x')$ por no estar definido $\delta(q, x_1)$ o por tener más de un valor asociado.

Ambas obstrucciones se resuelven con los algoritmos que se describen a continuación.

3.3.2 Autómatas con/sin $\lambda-$Transiciones.

Se denominan $\lambda-$transiciones a las transiciones de una autómata $A := (Q, \Sigma, q_0, F, \delta)$ de la forma:

$$\delta(q, \lambda) = p,$$

done hemos mantenido la notación válida para el caso determinístico e indeterminístico, a pesar de la notación incorrecta del segundo caso. Un autómata se dice *libre de $\lambda-$transiciones* si no hay ninguna de tales transiciones.
En un sentido menos preciso, las $\lambda-$transiciones son meras transformaciones de los estados conforme a reglas que no dependen del contenido de la cinta.

En términos del sistema de transición, para cada una configuración (q, x) en el sistema de transición asociado al autómata y supuesto que existe una $\lambda-$transición $\delta(q, \lambda) = p$, entonces la transición será de la forma $(q, x) \to (p, x)$, donde x no es modificado y sólo hemos modificado el estado.
En términos de operaciones de lecto–escritura, nuestra $\lambda-$transición realiza las siguientes tareas:

- *NO lee* el contenido de la cinta.
- *Modifica* el estado en la unidad de control.
- *NO borra* el contenido de la celda señalada por la unidad de control.
- *NO se mueve* a la izquierda.

3.3.2.1 Grafo de $\lambda-$transiciones.

A partir de las $\lambda-$transiciones de un autómata podemos construir un grafo. Dado un autómata $A := (Q, \Sigma, q_0, F, \delta)$, definimos el grafo de las $\lambda-$transiciones de A mediante $G := (V, E)$, donde las reglas son:

- $V = Q$.
- Dados $p, q \in V$, decimos que $(p, q) \in E$ si $q \in \delta(p, \lambda)$, i.e.

$$E := \{(p, q) \ : \ q \in \delta(p, \lambda)\}.$$

Si miramos el grafo asociado al autómata (cf. 3.2.1.1), podemos extraer el grafo de $\lambda-$transiciones, dejando los mismos nodos (o vértices) y suprimiendo todas las aristas que estén etiquetadas con algún símbolo del alfabeto (y dejando solamente las que están etiquetadas con λ).
A partir del grafo de las $\lambda-$transiciones podemos considerar la *clausura transitiva* de un nodo (estado), definiéndola del modo siguiente:

$$\lambda - cl(p) := \{q \in V \ : \ (p, \lambda) \vdash (q, \lambda)\}.$$

Obsérvese que la $\lambda-$clausura de un nodo p está determinada por las configuraciones (con palabra vacía λ) alcanzables desde la configuración (p, λ) dentro del sistema de transición asociado al autómata.
Obsérvese también que la palabra vacía λ está en el lenguaje aceptado $L(A)$ si y solamente si la clausura $\lambda - cl(q_0)$ del estado inicial contiene algún estado final aceptado (i.e. $\lambda - cl(q_0) \cap F \neq \emptyset$).
Del mismo modo, dados $p \in Q$ y $a \in \Sigma$, definiremos la $\lambda-$clausura de p y a mediante:

$$\lambda - cl(p, a) := \{q \in V \ : \ (p, \lambda) \vdash (q, \lambda), \ \exists \delta(q, a)\}.$$

Nuestro objetivo es probar el siguiente enunciado:

Proposición 55. *Dado cualquier lenguaje L que sea aceptado por un autómata con $\lambda-$transiciones, entonces existe un autómata libre de $\lambda-$transiciones que acepta el mismo lenguaje. Más aún, la transformación de un autómata a otra se puede realizar algorítmicamente.*

Demostración. Como en el resto de los casos, nos basta con tomar como dado de entrada un autómata $A := (Q, \Sigma, q_0, F, \delta)$ y definir un nuevo autómata que elimina las $\lambda-$transiciones. El nuevo autómata no ha de ser determinista, pero éso es irrelevante como veremos en la Proposición 58.
Construiremos un nuevo autómata $\bar{A} := (\bar{Q}, \Sigma, \overline{q_0}, \bar{F}, \bar{\delta})$ definido conforme al algoritmo siguiente:

INPUT: Autómata $A := (Q, \Sigma, q_0, F, \delta)$.

Initialize: $\bar{Q} := Q$ y $\overline{q_0} := q_0$.

for each $p \in Q$ **do find** $\lambda - cl(p)$ **od**

$\bar{F} := F \cup \{p \ : \ \lambda - cl(p) \cap F \neq \emptyset\}$.

for each $p \in Q$ **do**
  if $\lambda - cl(p, a) \neq \emptyset$, **then** $\bar{\delta}(p, a) := \bigcup_{q \in \lambda - cl(p)} \lambda - cl(\delta(q, a))$.
  fi
od
OUTPUT $\bar{A} := (\bar{Q}, \Sigma, \overline{q_0}, \bar{F}, \bar{\delta})$

Nótese que $\bar{\delta}(p, \lambda)$ no está definida para ningún $p \in Q$. Dejamos como ejercicio la comprobación de que el autómata $\bar{A}$ acepta L. □

Nota 56. *Obsérvese que el resultado de eliminar $\lambda-$transiciones puede ser un autómata indeterminista.*

Nota 57. *Nótese que en el caso en que $\lambda \in L(A)$ (i.e. $\lambda - cl(q_0) \cap F \neq \emptyset$), el estado inicial pasa a ser también estado final aceptador.*

3.3.3 Determinismo e Indeterminismo en Autómatas

Una primera preocupación técnica podría ser el papel que juega el indeterminismo en la clase de lenguajes aceptados por autómatas. Los siguientes resultados tranquilizan mostrando que el indeterminismo es irrelevante en cuanto a la clase de lenguajes aceptados.

Proposición 58. *Si un lenguaje $L \subseteq \Sigma^*$ es aceptado por un autómata finito indeterminista, entonces, existe un autómata finito determinista que lo acepta.* [2]

Demostración. La idea es simple, sea $A = (Q, \Sigma, q_0, F, \delta)$ un autómata indeterminista sin $\lambda-$transiciones que acepta un lenguaje $L \subseteq \Sigma^*$. Definamos el siguiente autómata determinista $\overline{A}$ dado por:

[2] Una característica del indeterminismo es que no modifica la clase de lenguajes aceptados; aunque sí podría modificar los tiempos de cálculo. Esto no afecta a los autómatas finitos, según se prueba en este enunciado, pero sí está detrás de la Conjetura de Cokk **P** = **NP**?.

- $\overline{Q} := \mathcal{P}(Q)$ (el espacio de estados es el conjunto de las partes de Q).
- $\overline{F} := \{X \in \overline{Q} \ : \ X \cap F \neq \emptyset\}$ (las configuraciones finales aceptadoras son aquellas que contienen algún estado del espacio F de estados finales aceptadores).
- $\overline{q_0} := \{q_0\}$ (el conjunto formado por la antigua configuración inicial).
- La función de transición
$$\overline{\delta} : \overline{Q} \times \Sigma \longrightarrow \overline{Q}$$
definida mediante:
$$\overline{\delta}(X, a) := \{q \in Q \ : \ \exists q' \in X, \ q = \delta(q', a)\}.$$

Dejamos el asunto de la comprobación como ejercicio. □

Nota 59. *A partir de ahora usaremos autómatas deterministas e indeterministas sin la preocupación sobre el indeterminismo, dado que podemos reemplazar unos por otros sin mayores problemas.*

3.4 Lenguajes Regulares y Autómatas.

Como indica el título, el objetivo de esta sección es mostrar que los lenguajes aceptados por los autómatas son los lenguajes regulares. Para ello, mostraremos dos procedimientos de paso conocidos como Teorema de Análisis y Teorema de Síntesis de Kleene (cf. [Kleene, 56]).

3.4.1 Teorema de Análisis de Kleene

Nuestra primera duda que cualquier lenguaje aceptado por un autómata finito esta generado por una expresión regular. El teorema siguiente afirma eso y además da un algoritmo para calcularlo. Se deja al alumno el ejercicio de demostrar la complejidad del algoritmo.

Teorema 60. *Sea $L \subseteq \Sigma^*$ un lenguaje aceptado por un autómata finito determinista. Entonces, existe una expresión regular α sobre el alfabeto Σ tal que $L = L(\alpha)$. Más aún, mostraremos que existe un procedimiento tratable que permite calcular la expresión regular asociada al lenguaje aceptado por un autómata.*

Demostración. Nos limitaremos con mostrar el procedimiento, que casi viene prefigurado por las definiciones.
Para ello construiremos un sistema de ecuaciones lineales en expresiones regulares con las reglas siguientes:

- Supongamos que $Q := \{q_0, \ldots, q_n\}$. Introducimos un conjunto de variables biyectable con Q dado por $\{X_0, \ldots, X_n\}$. La biyección será dada por $q_i \longmapsto X_i$.

- Definimos un sistema de ecuaciones lineales en expresiones regulares:

$$\begin{pmatrix} X_0 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} \alpha_{0,0} & \cdots & \alpha_{0,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,0} & \cdots & \alpha_{n,n} \end{pmatrix} \begin{pmatrix} X_0 \\ \vdots \\ X_n \end{pmatrix} + \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix},$$

  Conforme a las reglas siguientes:

  - Para cada i, $0 \leq i \leq n$, definamos $\beta_i = \lambda$ si $q_i \in F$ y $\beta_i = \emptyset$ si $q_i \notin F$.
  - Para cada i, j, $0 \leq i, j \leq n$, definamos $A_{i,j}$ mediante:

$$A_{i,j} := \{z \in \Sigma \ : \ \delta(q_i, z) = q_j\}.$$

    Definiremos

$$\alpha_{i,j} := \sum_{z \in A_{i,j}} z,$$

    notando que si $A_{i,j} = \emptyset$, entonces, $\alpha_{i,j} = \emptyset$.

Entonces, si $(\alpha_0, \ldots, \alpha_n)$ es una solución del anterior sistema lineal, $L(\alpha_0)$ es el lenguaje aceptado por el autómata. La idea de la demostración es la siguiente: Empecemos por calcular el lenguaje de las palabras que empezando en q_0 son aceptadas por el autómata y llamemos a este lenguaje X_0. De la misma forma, para cada uno de los estados ponemos un lenguaje X_1, $X_2 \ldots$ Hay una clara relación entre estos lenguajes, que esta dada por las ecuaciones lineales dadas más arriba. El lenguaje X_0 está claramente formado por la unión de los lenguajes X_i correspondientes, con prefijo dado por el símbolo de la transición. Además, si el estado es final hay que añadir la palabra λ. □

Definición 61 (Sistema Característico de un Autómata)**.** *Se denomina sistema de ecuaciones característico de un autómata al sistema de ecuaciones lineales en expresiones regulares obtenido conforme a las reglas descritas en la demostración del Teorema anterior.*

Nota 62. *Nótese que, a partir del Sistema característico de un autómata A uno podría reconstruir una gramática regular G que genera el mismo lenguaje $L(G)$ que el aceptado por A, i.e. $L(G) = L(A)$.*

3.4.2 Teorema de Síntesis de Kleene

En esta segunda parte, vamos a mostrar el recíproco. Esto es, que para cualquier lenguaje descrito por una expresión regular se puede encontrar un autómata determinista que lo acepta. Para ello haremos como en el caso del paso de expresiones regulares a gramáticas: usaremos el árbol de formación de la expresión regular. Comenzaremos por un sencillo Lema.

Lema 63. *Dado un lenguaje L aceptado por un autómata, existe un autómata $A := (Q, \Sigma, q_0, F, \delta)$ que acepta L y que verifica las siguientes propiedades:*

a. $\sharp(F) = 1$, *es decir, sólo hay una configuración final aceptadora. Supondremos* $F := \{f\}$.

b. $\delta(q, x)$ *está definida para todo* $q \in Q$ *y todo* $x \in \Sigma$.

c. *Las únicas* $\lambda-$*transiciones entran en* f. *Es decir,*

$$\text{Si } \delta(p, \lambda) = q \Leftrightarrow q = f.$$

Demostración. Dado el autómata $A := (Q, \Sigma, q_0, F, \delta)$, que podemos suponer determinista, definamos el nuevo autómata $\bar{A} := (\bar{Q}, \Sigma, q_0, \bar{F}, \bar{\delta})$ conforme a las reglas siguientes:

- Sea $f, ERROR$ dos nuevos estados tal que $f, ERROR \notin Q$. Definamos $\bar{Q} := Q \cup \{f\} \cup \{ERROR\}$.
- Definamos $\bar{F} := \{f\}$.
- Para cada $p \in Q$ y para cada $a \in \Sigma$, definamos para los nuevos estados

$$\bar{\delta}(ERROR, a) := ERROR, \qquad \bar{\delta}(f, a) = ERROR.$$

  y extendamos la función de transición para los antiguos estados si $a \in \Sigma$

$$\bar{\delta}(p, a) := \begin{cases} \delta(p, a), \text{ si } \delta(p, a) \text{ está definida,} \\ ERROR, \text{ en otro caso.} \end{cases}$$

- Para cada $p \in F$, definamos $\bar{\delta}(p, \lambda) := f$.

Es claro que $\bar{A}$ acepta el mismo lenguaje que aceptaba A. La razón es simple: la única manera de alcanzar el nuevo estado f es llegar a un estado final con la cinta vacía. □

Teorema 64. *Sea* Σ *un alfabeto finito y* α *una expresión regular sobre* Σ. *Entonces, existe un autómata finito* A *que reconoce el lenguaje* $L(\alpha)$ *descrito por* α. *Más aún, el proceso de obtención del autómata a partir de la expresión regular se puede lograr de manera algorítmica.*

Demostración. De nuevo nos limitaremos a describir un proceso algorítmico que transforma expresiones regulares en autómatas, usando los operadores de definición de la expresión (i.e., el procedimiento es recursivo en la construcción de la expresión regular).

- **El caso de los símbolos primarios:**
  - *El caso* $\emptyset$*:* Bastará un autómata con $Q := \{q_0, f\}$, $F := \{f\}$ tal que la función de transición no esté definida en ningún caso.
  - *El caso* λ*:* De nuevo usaremos $Q := \{q_0, f\}$, $F := \{f\}$, pero la función de transición está definida solamente para $\delta(q_0, \lambda) = f$ y no definida en el resto de los casos.

– *El caso constante $a \in \Sigma$:* Igual que en el caso anterior, usaremos $Q := \{q_0, f\}$, $F := \{f\}$, pero la función de transición está definida solamente para $\delta(q_0, a) = f$ y no definida en el resto de los casos.

- **Siguiendo los operadores:**

  – *El autómata de la unión ($\alpha + \beta$):* Si tenemos $A_1 := (Q_1, \Sigma, q_1, F_1, \delta_1)$ un autómata determinista que acepta $L(\alpha) \subseteq \Sigma^*$ y un segundo autómata también deterministas $A_2 := (Q_2, \Sigma, q_2, F_2, \delta_2)$ un autómata que acepta $L(\beta) \subseteq \Sigma^*$, definimos un nuevo autómata[3] $A := (Q, \Sigma, q_0, F, \delta)$ que acepta $L_1 \cup L_2$ y viene dado por las reglas siguientes:

  * $Q := Q_1 \times Q_2$,
  * $F := (F_1 \times Q_2) \cup (Q_1 \times F_2)$
  * $Q_0 := (q_1, q_2)$
  * $\delta((p, q), z) = (\delta_1(p, z), \delta_2(q, z))$, $\forall p \in Q_1, q \in Q_2$ y $\forall z \in \Sigma \cup \{\lambda\}$.

  – *El autómata de la concatenación ($\alpha \cdot \beta$):* Supongamos $A_1 := (Q_1, \Sigma, q_1, F_1, \delta_1)$ un autómata que acepta $L(\alpha) \subseteq \Sigma^*$ y un segundo autómata $A_2 := (Q_2, \Sigma, q_2, F_2, \delta_2)$ un autómata que acepta $L(\beta) \subseteq \Sigma^*$. Supongamos que A_1 verifica las condiciones descritas en el Lema 63 y sea $F_1 := \{f\}$.
  Definimos un nuevo autómata $A := (Q, \Sigma, q_0, F, \delta)$ que acepta $L(\alpha\beta)$ y viene dado por las reglas siguientes:

  * $Q := (Q_1 \times \{1\}) \cup (Q_2 \times \{2\})$.
  * $F := F_2 \times \{2\}$.
  * $q_0 := (q_1, 1)$
  * La función de transición $\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow Q$, viene dada por:

$$\delta((q,i),z) := \begin{cases} (\delta_1(q,z),1)\,, & \text{si } q \in Q_1,\ i = 1 \\ (Q_2,2)\,, & \text{si } q = f \in F_1,\ i = 1,\ z = \lambda \\ (\delta_2(q,z),2)\,, & \text{si } q \in Q_2,\ i = 2 \end{cases} \tag{3.1}$$

  – *El autómata del monoide generado (α^*):* De nuevo suponemos que tenemos un autómata $A := (Q, \Sigma, q_0, F, \delta)$ que acepta el lenguaje $L(\alpha)$. Podemos suponer que dicho autómata verifica las condiciones del Lema 63 anterior. Supongamos $F = \{f\}$. Definamos un nuevo autómata $A^* := (Q, \Sigma, q_0, F, \bar{\delta})$ conforme a las reglas siguientes:

  * Para cada $q \in Q \setminus F$ y para cada $z \in \Sigma \cup \{\lambda\}$, definamos $\bar{\delta}(q, z) := \delta(q, z)$.
  * Finalmente, definamos:

$$\bar{\delta}(f, \lambda) := q_0.$$

y

$$\bar{\delta}(q_0, \lambda) := f.$$

[3] Esta construcción se la conoce como *Autómata Producto.*

Es claro que este autómata acepta el lenguaje previsto.

Con esto acabamos la demostración, ya que cualquier expresión regular esta formada por concatenación, suma de expresiones regulares o es estrella de una expresión regular. □

3.5 Lenguajes que no son regulares

La tradición usa el Lema de Bombeo para mostrar las limitaciones de los lenguajes regulares. El resultado es debido a Y. Bar-Hillel, M. Perles, E. Shamir[4]. Este Lema se enuncia del modo siguiente:

Teorema 65 (Pumping Lemma)**.** *Sea L un lenguaje regular. Entonces, existe un número entero positivo $p \in \mathbb{N}$ $(p \geq 1)$ tal que para cada palabra $\omega \in L$, con $|\omega| \geq p$ existen $x, y, z \in \Sigma^*$ verificando los siguientes propiedades:*

- $|y| \geq 1$ *(i.e. $y \neq \lambda$),*
- $|xy| \leq p$,
- $\omega = xyz$,
- *Para todo $\ell \in \mathbb{N}$, las palabras $xy^\ell z \in L$*

El Lema de Bombeo simplemente dice que hay prefijos y una lista finita de palabras tal que, bombeando esas palabras, permaneceremos en el mismo lenguaje regular.

Nota 66. *Hay varias razones por las que éste es un enunciado insuficiente. La primera es estética: un exceso de fórmulas cuantificadas hace desagradable su lectura.*
Adicionalmente, debemos señalar que el Lema de Bombeo da una condición necesaria de los lenguajes regulares, pero no es una condición suficiente. Es decir, hay ejemplos de lenguajes que no son regulares (ver Corolario 69) pero que sí satisfacen el Lema de Bombeo (ver ejemplo 8 más abajo)

Ejemplo 8. *Los lenguajes regulares satisfacen el Lema de Bomeo y también lo satisface el siguiente lenguaje:*

$$L := \{a^i b^j c^k \ : \ [i = 0] \vee [j = k]\} \subseteq \{a, b, c\}^*.$$

Veamos que satisface el Teorema 65 anterior con $p = 1$. Para ello, sea $\omega \in \Sigma^*$ y tendremos tres casos:

- *Caso 1: $i = 0$ con $j = 0$ o, lo que es lo mismo, $\omega = c^k$, con $k \geq 1$. En ese caso, tomando $x = \lambda$, $y = c$, $z = c^{k-1}$, tenemos que $xy^\ell z \in L$, $\forall \ell \in \mathbb{N}$.*

[4] Y. Bar-Hillel, M. Perles, E. Shamir. "On formal properties of simple phrase structure grammars". *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* **14** (1961) 143–172.

- *Caso 2:* $i = 0$ *con* $j \geq 1$ *o, lo que es lo mismo,* $\omega = b^j c^k$, $j \geq 1$. *En ese caso, tomando* $x = \lambda$, $y = b$, $z = b^{j-1}c^k$, *tenemos que* $xy^\ell z \in L$, $\forall \ell \in \mathbb{N}$.
- *Caso 3:* $i \geq 1$ *o, lo que es lo mismo,* $\omega = a^i b^j c^j$. *En ese caso, tomando* $x = \lambda$, $y = a$, $z = a^{i-1}b^j c^j$, *tenemos que* $xy^\ell z \in L$, $\forall \ell \in \mathbb{N}$.

Veremos más adelante (Corolario 69) que este lenguaje no es regular.

Definición 67 (Prefijos)**.** *Sea* Σ *un alfabeto finito y sea* $L \subseteq \Sigma^*$ *un lenguaje cualquiera. Definimos la siguiente relación de equivalencia sobre* Σ^**:*
dados $x, y \in \Sigma^*$, $x \sim_L y$ *si y solamente si:*

$$\forall w \in \Sigma^*, xw \in L \Leftrightarrow yw \in L.$$

Verificar que estamos ante una relación de equivalencia es un mero ejercicio. Lo que pretendemos es caracterizar los lenguajes aceptados por un autómata mediante una caracterización del conjunto cociente: $\Sigma^*/\sim_L$.

Teorema 68 (Myhill–Nerode)**.** [5] *Si* $L \subseteq \Sigma^*$ *es un lenguaje, entonces* L *es regular si y solamente si* $\Sigma^*/\sim_L$ *es finito.*

Demostración. Comencemos con una de las implicaciones. Supongamos que L es el lenguaje aceptado por un autómata determinista $A := (\Sigma, Q, q_0, F, \delta)$. Consideremos el conjunto de los estados alcanzables por alguna computación de A:

$$\overline{Q} := \{q \in Q \ : \ \exists y \in \Sigma^*, \ (q_0, y) \vdash (q, \lambda)\}$$

es claro que $\overline{Q} \subseteq Q$ es un conjunto finito. Para cada $q \in \overline{Q}$, sea $y_q \in \Sigma^*$ un elemento cualquiera tal que $(q_0, y_q) \vdash (q, \lambda)$. Sea

$$S := \{y_q \ : \ q \in \overline{Q}\}$$

Claramente S es un conjunto finito y vamos a probar que

$$\Sigma^*/\sim_L = \{[y_q] \ : \ y_q \in S\},$$

donde $[y_q]$ es la clase de equivalencia definida por y_q y tendremos la afirmación. Ahora, tomemos $x \in \Sigma^*$ y sea $(q_0, x) \vdash (q, \lambda)$, $q \in \overline{Q}$. Para cualquier $w \in \Sigma^*$, el sistema de transición asociado al autómata A, trabajando sobre xw realiza algo como lo siguiente:

$$(q_0, xw) \to_A \cdots \to_A (q, w)$$

mientras vamos borrando la x. Ahora bien, si tomamos $y_q w \in \Sigma^*$, el cálculo hará también el camino:

$$(q_0, y_q w) \to_A \cdots \to_A (q, w)$$

Lo que pase a partir de (q, w) es independiente de por dónde hayamos empezado, luego xw es aceptado por A si y solamente si $y_q w$ es aceptado por A. Con esto hemos demostrado una de las direcciones del enunciado, esto es, si el lenguaje es

[5] Rabin, M. and Scott, D.."Finite automata and their decision problems". *IBM Journal of Research & Development* **3** (1959), 114-125.

regular y, por ende, aceptado por un autómata finito, entonces, el conjunto cociente $\Sigma^*/\sim_L$ es finito.
Para el recíproco, supongamos que $\Sigma^*/\sim_L$ es finito y supongamos:

$$\Sigma^*/\sim_L = \{[y_1], \ldots, [y_m]\},$$

donde $y_i \in \Sigma^*$. Podemos suponer que $y_1 = \lambda$ (la palabra vacía estará en alguna clase de equivalencia). Además, observemos que la clase de equivalencia $[y_1] = [\lambda]$ está formada por los elementos de L. Ahora definamos un autómata $A = (Q, \Sigma, q_0, F, \delta)$ con las reglas siguientes:

- Los estados están definidos mediante:

$$Q := \{[y_1], \ldots, [y_m]\}.$$

- El estado inicial es dado por $q_0 = [y_1] = [\lambda]$.
- El espacio de estados finales aceptadores es $F := \{[\lambda]\}$.
- La función de transición es dada para cada $x \in \Sigma \cup \{\lambda\}$, mediante:

$$\delta([y], x) = [yx].$$

Veamos que esta autómata realiza la tarea indicada. La configuraciíon inicial es $([\lambda], \omega)$ para cualquier palabra $\omega \in \Sigma^*$. Conforme va avanzando a partir de esta configuración, el autómata alcanza $([\omega], \lambda)$ y, por tanto, acepta si y solamente si $[\omega] = [\lambda]$, lo cual es equivalente a $\omega \in L$. □

Corolario 69. *El lenguaje L descrito en el Ejemplo 8 no es un lenguaje regular.*

Demostración. Basta con verificar que no satisface las propiedades descritas en el Teorema de Myhill–Nerode. Para ello supongamos que el conjunto cociente $\Sigma^*/\sim_L$ es finito, es decir,

$$\Sigma^*/\sim_L := \{[y_1], \ldots, [y_r]\}.$$

Consideremos la sucesión infinita $x_n := ab^n$. Como sólo hay un número finito de clases de equivalencia, hay una clase de equivalencia (digamos $[y_1]$) que contiene una infinidad de términos de esa sucesión. En otras palabras, existe una sucesión infinita y creciente:

$$1 < n_1 < n_1 < \cdots < n_k < n_{k+1} < \cdots, \tag{3.2}$$

de tal modo que

$$\{x_{n_i} \ : \ i \in \mathbb{N}, i \geq 1\} \subseteq [y_1].$$

En este caso, se ha de tener, además, la siguiente propiedad:

$$\forall \omega \in \Sigma^*, \ y_1\omega \notin L. \tag{3.3}$$

Para probarlo, nótese que si existiera naluna palabra $\omega \in \Sigma^*$ tal que $y_1\omega \in L$, entonces podemos suponer que esa palabra es de longitud finita. Supongamos $p :=$

$|\omega| \in \mathbb{N}$ esa longitud. Como la sucesión de los n_i's es inifniota y creciente, entonces, ha de existir algún n_t tal que $n_t > p + 3$.
Pero, además, $ab^{n_t} \sim_L y_1$, luego, como $y_1\omega \in L$, entonces también se ha de tener

$$ab^{n_t}\omega \in L.$$

Por la definición de L tendremos, entonces que

$$ab^{n_t}\omega = ab^j c^j,$$

para algún j. Obviamente ésto significa que ω es de la forma $\omega = b^r c^j$ y, necesariemente, $j = n_t + r \geq n_t$. Por lo tanto, $p + 3 = |\omega| + 3 \geq j + 3 = n_t + r + 3 > n_t$, contraviniendo nuestra elección de $n_t \geq p + 3$.
Con esto hemos probado la veracidad de la afirmación (3.3) anterior. Pero, de otro lado, $ab_{n_1}c_{n_1} \in L$ y $ab^{n_1} \sim_L y_1$ luego $y_1 c^{n_1} \in L$, lo que contradice justamente la afirmación probada. La hipótesis que no se sostiene es que el conjunto cociente $\Sigma^*/\sim_L$ sea finito y, por tanto, L no es un lenguaje regular. □

3.5.1 El Palíndromo no es un Lenguaje Regular.

Se trata del ejemplo clásico y común que "deben" contemplar todos los cursos de Introducción a los lenguajes regulares: *el Palíndromo* o, en buen catalán, el problema de la detección de los *"cap–i–cua"*, del que veremos que no es un lenguaje regular, como consecuencia del resultado de Myhill y Nerode anterior.
Comencemos recordando la definición del Palíndromo ya presentado en Secciones anteriores. Dado un alfabeto finito Σ, y una palabra $\omega = x_1 \cdots x_n \in \Sigma^*$, denominamos el reverso de ω, ω^R a la palabra: $\omega^R = x_n \cdots x_1$.
El lenguaje del Palíndromo es dado por las palabras que coinciden con su reverso, esto es,

$$\mathcal{P} := \{x \in \Sigma^* \ : \ x^R = x\}.$$

Daremos una demostración del resultado siguiente usando la finitud de lso prefijos.

Corolario 70. *El Palíndromo no es un lenguaje regular si el alfabeto tiene al menos dos dígitos distintos.*

Demostración. Por simplicidad supongamos $\Sigma = \{0, 1\}$. Para cada número natural $n \in \mathbb{N}$, consideremos la palabra de longitud $n + 2$ siguiente:

$$x_n := 0^n 10.$$

Supongamos que el palíndromo es un lenguaje regular y será finito el conjunto cociente siguiente:

$$\Sigma^*/\sim_{\mathcal{P}} = \{[y_1], \ldots, [y_m]\}.$$

De otro lado, consideremos las clases definidas por los elementos de la sucesión anterior:

$$S := \{[x_n] \ : \ n \in \mathbb{N}\}.$$

Como el conjunto cociente es finito, el anterior conjunto S es finito y, por tanto, habrá alguna clase $[y]$ en la que estará una infinitud de elementos de la sucesión $\{x_n \ : \ n \in \mathbb{N}\}$. Es decir, que existe una sucesión infinita creciente de índices:

$$n_1 < n_2 < n_3 < \cdots < n_k < \cdots$$

de tal modo que $x_{n_j} \in [y]$. Supongamos n_j suficientemente grande (por ejemplo, $n_j \geq 2|y| + 3$).
Ahora obsérvese que $x_{n_j} x_{n_j}^R \in \mathcal{P}$ es un palíndromo. Como $x_{n_j} \sim_{\mathcal{P}} y$ (están en la misma clase de equivalencia), tendremos que $y x_{n_j}^R \in \mathcal{P}$. Por tanto,

$$y x_{n_j}^R = x_{n_j} y^R. \tag{3.4}$$

Como la longitud de x_{n_j} es mayor que la de y, tendremos que y debe coincidir con los primeros $\ell = |y|$ dígitos de x_{n_j}. Por tanto, $y = 0^\ell$.
Ahora bien, el único dígito 1 de la palabra $y x_{n_j}^R$ en la identidad (3.4) ocupa el lugar $\ell + 2$, mientras que el único dígito 1 de la palabra $x_{n_j} y^R$ ocupa el lugar $n_j + 1$, como $n_j \geq 2\ell + 3$ no es posible que ambas palabra sean iguales, contradiciendo la igualdad (3.4) y llegando a contradicción. Por tanto, el palídromo no puede ser un lenguaje regular. □

Ejemplo 9. *Los siguientes son también ejemplos de lenguajes no regulares:*

- $\Sigma = \{0, 1\}$ *y el lenguaje* L *dado por la condición el número de* 1*'s es mayor que el número de* 0*'s.*

- *Para el mismo alfabeto el lenguaje:*

$$L := \{0^m 1^m \ : \ m \in \mathbb{N}\}$$

- *Para el alfabeto* $\Sigma = \{0, 1, \ldots, 9\}$ *sea* $\pi \subseteq \Sigma^*$ *el lenguaje formado por las palabras que son prefijos de la expansión decimal de* $\pi \in \mathbb{R}$*, es decir:*

$$L := \{3, 31, 314, 3141, 31415, \ldots\}$$

3.6 Minimización de Autómatas Deterministas

En ocasiones uno puede observar que el autómata que ha diseñado (usando algunas de las propiedades o métodos ya descritos) es un autómata con demasiados estados (y, por tanto, el código del programa es excesivo para el programador). Para resolver esta situación se utiliza un proceso de minimización de los autómatas que pasaremos a describir a continuación.
Comenzaremos observando que las computaciones que realizan varios estados pueden ser esencialmente las mismas, que los efectos que producen ciertos estados podrían ser los mismos. Esto se caracteriza mediante la relación de equivalencia siguiente:

3.6.1 Eliminación de Estados Inaccesibles.

En ocasiones se presentan autómatas en los que se han incluido estados innaccesibles, es decir, estados a los que no se puede llegar de ningún modo desde el estado inicial. Para describir esta noción, definiremos la siguiente estructura de grafo asociada a un autómata.
Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata determinista. Consideremos el grafo de estados siguiente: $\mathcal{G}_A := (V, E)$, donde

- El conjunto de vértices o nodos V es el conjunto de estados Q (i.e. $V = Q$).
- Las aristas del grafo (que será orientado) son los pares (p, q) tales que existe $x \in \Sigma$ verificando $\delta(p, x) = q$.

Nótese que el grafo coincide con el grafo subyacente a la descripción del autómata como grafo con pesos.

Definición 71. *Dado un autómata $A := (Q, \Sigma, q_0, F, \delta)$, un estado $q \in Q$ se denomina accesible si está en la clausura transitiva (componente conexa) del estado inicial q_0. Se llaman estados inaccesibles aquellos estados que no son accesibles.*

Proposición 72. *Dado un autómata $A := (Q, \Sigma, q_0, F, \delta)$, existe un autómata A' que acepta el mismo lenguaje y que no contiene estados inaccesibles.*

Demostración. Para definir A' basta con eliminar los estados inaccesibles del autómata A, es decir, definimos $A' := (Q', \Sigma, q_0{}', F', \delta')$ mediante

- $Q' := \{q \in Q \ : \ q \text{ es accesible desde } q_0 \text{ en } \mathcal{G}_A\}$.
- $q'_0 = q_0$.
- $F' := F \cap Q$.
- La función de transición δ' es la restricción a Q' de δ:

$$\delta' := \delta \mid_{Q' \times \Sigma} .$$

□

3.6.2 Autómata Cociente

Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata determinista. Supongamos que todos los estados son accesibles. Dos estados $p, q \in Q$ se dicen equivalentes si se verifica la siguiente propiedad:

$$\forall z \in \Sigma^*, \ \text{Si } \big(\big((p, z) \vdash (p', \lambda)\big) \wedge \big((q, z) \vdash (q', \lambda)\big)\big) \Longrightarrow \big((p' \in F) \Leftrightarrow (q' \in F)\big) .$$

En otras palabras, dos estados son equivalentes si para cualquier palabra el efecto de la computación que generan es el mismo (en términos de alcanzar o no un estado final aceptador).
Denotaremos por $p \sim_A q$ en el caso de que p y q sean equivalentes. Para cada estado $q \in Q$, denotaremos por $[q]_A$ la clase de equivalencia definida por q y denotaremos por $Q/\sim_A$ al conjunto cociente. Definiremos autómata minimal al autómata que tiene el menor número de estados y que acepta un lenguaje.

Teorema 73 (Autómata Cociente)**.** *Sea L un lenguaje aceptado por un autómata determinista A sin estados inaccesibles. Entonces, existe un autómata minimal que lo acepta. Dicho autómata $(\tilde{Q}, \Sigma, \widetilde{Q_0}, \tilde{F}, \tilde{\delta})$ viene dado en los términos siguientes:*

- $\tilde{Q} := Q/\sim_A$,
- $\tilde{F} := \{[q]_A \ : \ q \in F\}$.
- $\widetilde{q_0} := [q_0]_A$.
- $\tilde{\delta}([q]_A, z) := [\delta(q, a)]$.

Demostración. Lo dejamos para la reflexión de los alumnos. □

3.6.3 Algoritmo para el Cálculo de Autómatas Minimales.

De la definición del autómata cociente, concluimos la dificultad (aparente) del cálculo de las clases de equivalencia no puede hacerse de manera simple (porque habríamos de verificar todas las palabras $z \in \Sigma^*$). Por eso se plantean algoritmos alternativos como el que se describe a continuación (tomado de [Eilenberg, 74]).
Para construir nuestro autómata cociente, tomaremos una cadena de relaciones de equivalencia. Las definiremos recursivamente del modo siguiente:
Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata. Definamos las siguientes relaciones:

- *La relación E_0:* Dados $p, q \in Q$, diremos que pE_0q (p y q están relacionados al nivel 0) si se verifica:
$$p \in F \Leftrightarrow q \in F.$$
Es claramente una relación de equivalencia. El conjunto cociente está formado por dos clases:
$$Q/E_0 := \{F, Q \setminus F\}.$$
Definamos $e_0 := \sharp\,(Q/E_0) = 2$.

- *La relación E_1:* Dados $p, q \in Q$, diremos que pE_1q (p y q están relacionados al nivel 1) si se verifica:
$$pE_1q \Leftrightarrow \begin{cases} pE_0q, \\ \wedge \\ \delta(p,z)E_0\delta(q,z), \quad \forall z \in \Sigma \cup \{\lambda\} \end{cases}$$
Es, de nuevo, una relación de equivalencia. El conjunto cociente ya no es tan obvio, y definimos:
$$e_1 := \sharp\,(Q/E_1)\,.$$

- *La relación E_n:* Para $n \geq 2$, definimos la relación del modo siguiente: Dados $p, q \in Q$, diremos que pE_nq (p y q están relacionados al nivel n) si se verifica:
$$pE_nq \Leftrightarrow \begin{cases} pE_{n-1}q, \\ \wedge \\ \delta(p,z)E_{n-1}\delta(q,z), \quad \forall z \in \Sigma \cup \{\lambda\} \end{cases}$$

Es, de nuevo, una relación de equivalencia. El conjunto cociente ya no es tan obvio, y definimos:

$$e_n := \sharp\,(Q/E_n)\,.$$

Lema 74. *Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata y sean $\{E_n : n \in \mathbb{N}\}$ la cadena de relaciones de equivalencia definidas conforme a la regla anterior. Se tiene:*

a. Para cada $n \in \mathbb{N}$, $e_n \leq e_{n+1}$.

b. Si existe $n \in \mathbb{N}$, tal que $e_n = e_{n+1}$, entonces

$$e_m = e_n,\ \forall m \geq n.$$

Demostración. Es claro que si dos estados están relacionados a nivel n entonces, están relacionados a nivel $n-1$. Esto es así por pura construcción (por definición). Por tanto, la relación E_{n+1} lo más que puede hacer es partir en más de una clase de equivalencia alguna de las clases de equivalencia del conjunto cociente anterior. Por tanto,

$$e_n = \sharp(Q/E_n) \leq \sharp(Q/E_{n+1}) = e_{n+1}.$$

Como, además, la relación E_{n+1} se define inductivamente a partir de la relación E_n, si $e_n = e_{n+1}$, entonces, las clases a nivel n siguen siendo las clases a nivel $n+1$. En otras palabras, si $e_n = e_{n+1}$, entonces para todo par $p, q \in Q$, pE_nq si y solamente si $pE_{n+1}q$. En particular, $E_n = E_{n+1}$ y ambas relaciones de equivalencia son la misma. Inductivamente, para $n+2$ se tendrá

$$pE_{n+2}q \Leftrightarrow \left\{ \begin{array}{c} pE_{n+1}q, \\ \\ \delta(p,z)E_{n+1}\delta(q,z), \quad \forall z \in \Sigma \cup \{\lambda\} \end{array} \right\} \Leftrightarrow$$

$$\Leftrightarrow \left\{ \begin{array}{c} pE_nq, \\ \\ \delta(p,z)E_n\delta(q,z), \quad \forall z \in \Sigma \cup \{\lambda\} \end{array} \right\} \Leftrightarrow pE_{n+1}q \Leftrightarrow pE_nq.$$

Por tanto $E_{n+2} = E_{n+1} = E_n$ y, en consecuencia, $e_{n+2} = e_{n+1} = e_n$.
Para cualquier $m \geq n+3$, aplique inducción para concluir $E_m = E_{n+1} = E_n$ y, además, $e_m = e_n$. □

Proposición 75. *Con las notaciones del Lema anterior, para cada autómata A existe $n \in \mathbb{N}$, con $n \leq \sharp(Q) - 2$, tal que para todo $m \geq n$ se verifica:*

a. $pE_mq \Leftrightarrow pE_nq$, $\forall p, q \in Q$.

b. $e_m = e_n$.

Demostración. Por el Lema anterior, concluimos:

$$2 = e_0 \leq e_1 \leq e_2 \leq \cdots \leq e_n \leq \cdots$$

Ahora consideremos $n = \sharp(Q) - 2$. Pueden ocurrir dos cosas:

- *Caso I:* Que $e_i \neq e_{i+1}$ para todo $i \leq n$. Es decir, que tengamos (con $n = \sharp(Q) - 2$):
$$2 = e_0 < e_1 < e_2 < \cdots < e_n.$$
En este caso, tendríamos
$$e_1 \geq e_0 + 1 = 3,$$
$$e_2 \geq e_1 + 1 \geq 4,$$
$$\vdots$$
$$e_{n-1} \geq e_{n-2} + 1 \geq \cdots \geq e_1 + (n-2) \geq 3 + (n-1) = n + 2 = \sharp(Q).$$
$$e_n \geq e_{n-1} + 1 \geq n + 3 = \sharp(Q).$$
Recordemos que $e_n = \sharp(Q/E_n)$ y el número de clases de equivalencia no puede ser mayor que el número de elementos de Q, es decir, habremos logrado que $n + 3 \leq e_n \leq \sharp(Q) = n + 2$ y eso es imposible. Así que este caso no se puede dar.

- *Caso II:* La negación del caso anterior. Es decir, existe un i, con $0 \leq i \leq n$ (y $n = \sharp(Q) - 2$) tal que $e_i = e_i + 1$. Entonces, por el Lema anterior, se tendrá:
$$2 = e_0 < e_1 < e_2 < \cdots < e_i = e_{i+1} = \cdots = e_m = \cdots$$

A partir de que sólo se puede dar el caso II, es obvio que se tienen las propiedades del enunciado. □

Teorema 76. *Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata sin $\lambda-$transiciones y sean p, q dos estados. Entonces, tomando $n = \sharp(Q) - 2$, se tendrá que*
$$p \sim_A q \Leftrightarrow pE_nq.$$

Demostración. Lo dejamos como ejercicio para el alumno. □

En particular, el algoritmo que calcula el autómata minimal funciona como sigue:

- *Hallar el conjunto cociente (Q/E_0) y su cardinal e_0.*
- *(Siguiendo los E_i's) Mientras el conjunto cociente "nuevo" sea alterado con respecto al anterior, hallar el conjunto cociente siguiente.*
- *Parar cuando el cardinal del "nuevo" conjunto cociente coincida con el último calculado.*

3.7 Cuestiones y Problemas.

3.7.1 Cuestiones.

Cuestión 14. *Sea $A := (Q, \Sigma, q_0, F, \delta)$ un autómata indeterminista que verifica la siguiente propiedad: Para todo estado q y para todo símbolo $z \in \Sigma \cup \{\lambda\}$,*
$$\sharp\left(\{p \ : \ \delta(q, z) = p\}\right) \leq 1,$$
donde $\sharp$ significa cardinal. Dar un procedimiento inmediato para hallar uno equivalente que sea determinista.

Cuestión 15. *Hallar una expresión regular α sobre el alfabeto $\{a, b\}$ que describa el lenguaje aceptado por el autómata siguiente. Sea $Q := \{q_0, q_1\}$ y $F = \{q_1\}$. Siendo la función de transición dada por la tabla:*

δ	**a**	**b**	λ
q_0	q_0	q_1	**N.D.**
q_1	**N.D.**	**N.D.**	**N.D.**

Donde **N.D.** *significa "No Definido".*

Cuestión 16. *Considerar el autómata $A := (Q, \Sigma, q_0, F, \delta)$, donde*

- $\Sigma := \{a\}$,
- $Q := \{q_0, q_1, q_2\}$,
- $F := \{q_2\}$.

Y la función de transición es dada por la Tabla siguiente:

δ	**a**	λ
q_0	q_1	**N.D.**
q_1	q_2	**N.D.**
q_2	q_0	**N.D.**

Probar que $L(A) = \{(aaa)^n aa \ : \ n \in \mathbb{N}\}$.

Cuestión 17. *Describir un autómata que acepta el siguiente lenguaje:*

$$L(A) := \{\omega \in \{a, b\}^* \ : \ \sharp(\text{apariciones de } b \text{ en } \omega) \in 2\mathbb{N}\}.$$

Cuestión 18. *Considérese el autómata siguiente:*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $F := \{q_2, q_3, q_4\}$.

Cuya función de transición es dada por la tabla siguiente:

δ	**0**	**1**	λ
q_0	q_1	q_2	**N.D.**
q_1	q_0	q_3	**N.D.**
q_2	q_4	q_5	**N.D.**
q_3	q_4	q_4	**N.D.**
q_4	q_4	q_5	**N.D.**
q_5	q_5	q_5	**N.D.**

a. *Dibuja el grafo que describe al autómata.*

b. Probar que q_0 y q_1 son equivalentes.

c. Probar que q_2, q_3, q_4 son equivalentes.

d. Hallar el autómata mínimo correspondiente.

Cuestión 19. *Sea G una gramática sobre el alfabeto $\{a, b\}$ cuyas reglas de producción son las siguientes:*

$$\begin{array}{rcl} Q_0 & \mapsto & bA \mid \lambda \\ A & \mapsto & bB \mid \lambda \\ B & \mapsto & aA \end{array}$$

Hallar un autómata que acepte el lenguaje generado por esa gramática. Hallar el autómata mínimo que acepte ese lenguaje. Hallar una expresión regular que describa ese lenguaje.

Cuestión 20. *Dado un autómata A que acepta el lenguaje L, ¿hay un autómata que acepta el lenguaje L^R?, ¿cómo le describirías?.*

Cuestión 21. *Dado un autómata A que acepta el lenguaje descrito por una expresión regular α y dado un símbolo a del alfabeto, ¿Cómo sería el autómata finito que acepta el lenguaje $L(D_a(\alpha))$?*

3.7.2 Problemas

Problema 33. *Construir autómatas que acepten los siguientes lenguajes:*

a. $L_1 := \{\omega \in \{a, b\}^* \ : \ abab \text{ es una subcadena de } \omega\}$.

b. $L_2 := \{\omega \in \{a, b\}^* \ : \ \text{ni } aa \text{ ni } bb \text{ son subcadenas de } \omega\}$.

c. $L_3 := \{\omega \in \{a, b\}^* \ : \ ab \text{ y } ba \text{ son subcadenas de } \omega\}$.

d. $L_4 := \{\omega \in \{a, b\}^* \ : \ bbb \text{ no es subcadena de } \omega\}$.

Problema 34. *Hallar un autómata determinista equivalente al autómata indeterminista $A := (Q, \{0, 1\}, q_0, F, \delta)$, donde*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4\}$,
- $F := \{q_4\}$.

Y δ es dado por la tabla siguiente:

δ	**a**	**b**	λ
q_0	**N.D.**	q_2	q_1
q_1	q_0, q_4	**N.D.**	q_2, q_3
q_2	**N.D.**	q_4	**N.D.**
q_3	q_4	**N.D.**	**N.D.**
q_4	**N.D.**	**N.D.**	q_3

Problema 35. *Minimizar el autómata sobre el alfabeto* $\{0,1\}$ *dado por las propiedades siguientes:*

- $\Sigma := \{0,1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $F := \{q_3, q_4\}$.

δ	**0**	**1**	λ
q_0	q_1	q_2	**N.D.**
q_1	q_2	q_3	**N.D.**
q_2	q_2	q_4	**N.D.**
q_3	q_3	q_3	**N.D.**
q_4	q_4	q_4	**N.D.**
q_5	q_5	q_4	**N.D.**

Hallar su grafo, una gramática que genere el mismo lenguaje y una expresión regular que lo describa.

Problema 36. *Construir una expresión regular y un autómata finito asociados al lenguaje siguiente:*

$$L := \{\omega \in \{a,b\}^* \ : \ \exists z \in \{a,b\}^*,\ \omega = azb\}.$$

Problema 37. *Hallar una expresión regular y una gramática asociadas al lenguaje aceptado por el autómata* $A := (Q, \Sigma, q_0, F, \delta)$*, dado por las propiedades siguientes*

- $\Sigma := \{a,b\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4\}$,
- $F := \{q_3, q_4\}$.

Y δ *es dado por la tabla siguiente:*

δ	**a**	**b**	λ
q_0	q_1	**N.D.**	**N.D.**
q_1	q_2	q_4	**N.D.**
q_2	q_3	q_4	**N.D.**
q_3	q_3	q_4	**N.D.**
q_4	**N.D.**	q_4	**N.D.**

Problema 38. *Hallar un autómata determinista que acepta el lenguaje descrito por la siguiente expresión regular:*

$$a(bc)^*(b+bc)+a.$$

Minimiza el resultado obtenido.

Problema 39. *Haz lo mismo que en el problema anterior para la expresión regular:*

$$(a\,(ab)^*)^*\,da\,(ab)^*\,.$$

Problema 40. *Haz lo mismo que en el problema anterior para la expresión regular:*

$$0(011)^*0 + 10^*\,(1\,(11)^*\,0 + \lambda) + \lambda.$$

Problema 41. *Calcula un autómata finito determinista minimal, una gramática regular y una expresión regular para el lenguaje siguiente:*

$$L := \{\omega \in \{0,1\}^* \ : \ [\sharp\,(0\text{'s en } \omega) \in 2\mathbb{N}] \vee [\sharp\,(1\text{'s en } \omega) \in 3\mathbb{N}]\}.$$

Problema 42. *Obtener una expresión regular para el autómata descrito por las siguientes propiedades: $A := (Q, \Sigma, q_0, F, \delta)$, y*

- $\Sigma := \{a, b\}$,
- $Q := \{q_0, q_1, q_2\}$,
- $F := \{q_2\}$.

Y δ es dado por la tabla siguiente:

δ	**a**	**b**	λ
q_0	q_0, q_2	**N.D.**	q_1
q_1	q_2	q_1	**N.D.**
q_2	**N.D.**	**N.D.**	q_1

Problema 43. *Dada la expresión regular $(ab)^*(ba)^* + aa^*$, hallar:*

a. El autómata determinista minimal que acepta el lenguaje que describe esa expresión regular.

b. Una gramática regular que genere dicho lenguaje.

Problema 44. *Considera un tipo de datos* real *de algún lenguaje de programación. Halla una expresión regular que describa este lenguaje. Halla un autómata que los reconozca y una gramática que los genere.*

Problema 45. *Se considera el autómata descrito por la información siguiente $A := (Q, \Sigma, q_0, F, \delta)$, y*

- $\Sigma := \{0, 1\}$,
- $Q := \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $F := \{q_1, q_3, q_5\}$.

Y δ es dado por la tabla siguiente:

δ	**0**	**1**	λ
q_0	**N.D.**	**N.D.**	q_1
q_1	q_2	**N.D.**	**N.D.**
q_2	**N.D.**	q_3	**N.D.**
q_3	q_4	**N.D.**	q_1
q_4	q_5	**N.D.**	**N.D.**
q_5	**N.D.**	**N.D.**	q_3, q_1

Se pide:

a. *Dibujar el grafo de transición del autómata.*

b. *Decidir si* 0101 *es aceptado por al autómata y describir la computación sobre esta palabra.*

c. *Hallar un autómata determinista que acepte el mismo lenguaje.*

d. *Minimizar el autómata determinista hallado.*

e. *Hallar una expresión regular que describa el lenguaje aceptado por ese autómata.*

f. *Hallar una gramática que genere dicho lenguaje.*

Problema 46. *Hallar un autómata que acepte las expresiones matemáticas con sumas y restas y sin paréntesis. Añadir a este una cinta donde escribir la traducción al español acabando en punto. Ejemplo:*

$$4 + 3 - 5 \mapsto \textit{ cuatro mas tres menos cinco.}$$

Problema 47. *Suponer que al autómata anterior se le quisiera añadir expresiones con paréntesis. Para hacer esto toma la expresión regular del autómata anterior α y se añade considera la siguiente expresión regular $(^*\alpha)^*$. Demostrar que el autómata no comprueba que todos los paréntesis que se abren son cerrados.*

Problema 48. *Otro de los problemas de los autómatas finitos es que no tienen en cuenta el orden entre los distintos elementos. Utilicemos una expresión regular α mencionada en el ejercicio anterior. Hallar el autómata que acepte el lenguaje generado por la siguiente expresión regular*

$$('('+'\{')^* alpha(')'+'\}')^*.$$

Demostrar que el autómata no tiene en cuenta el orden de aparición de las llaves y los paréntesis.

Problema 49. *Este ejercicio demuestra el problema de traducción para las estructuras condicionales. Suponemos que EXPRESION es conocido y los bucles condicionales están dados por la siguiente expresión regular:*
((if EXPRESION then BUCLE) (else if EXPRESION then BUCLE)) Hallar un autómata finito que acepte el lenguaje dado por la expresión regular y discutir como añadir una cinta de traducción.*

Chapter 4

Gramáticas Libres de Contexto

Contents

4.1 Introducción 71
4.2 Árboles de Derivación de una Gramática 73
4.2.1 Un algoritmo incremental para la vacuidad. 75
4.3 Formas Normales de Gramáticas. 76
4.3.1 Eliminación de Símbolos Inútiles o Inaccesibles 76
4.3.1.1 Eliminación de Símbolos Inaccesibles. 78
4.3.1.2 Eliminación de Símbolos Inútiles. 78
4.3.2 Transformación en Gramáticas Propias. 79
4.3.2.1 Eliminación de $\lambda-$producciones. 79
4.3.2.2 Eliminación de Producciones Simples o Unarias . 81
4.3.2.3 Hacia las Gramáticas Propias. 82
4.3.3 El Problema de Palabra para Gramáticas Libres de Contexto es Decidible. 84
4.3.4 Transformación a Formal Normal de Chomsky. 86
4.3.5 Forma Normal de Greibach 87
4.4 Cuestiones y Problemas 88
4.4.1 Cuestiones 88
4.4.2 Problemas 88

4.1 Introducción

El proceso de compilación es el proceso que justifica la presencia de un estudio teórico de lenguajes y autómatas como el que se desarrolla en esta asignatura. Sin embargo, el objetivo de un curso como éste no es, ni debe ser, el del diseño de un compilador, ni siquiera el del análisis de todos los procesos que intervienen en la compilación. Para culminar este proceso existe una asignatura dedicada a "Compiladores" (la asignatura llamada Procesadores de Lenguajes) en el curso cuarto de la titulación dentro del plan de estudios vigente.

Sin embargo, la pérdida de la motivación puede suponer la pérdida del interés (y del potencial atractivo) de una materia teórica y densa como la presente. En compensación, podemos recuperar el esquema básico de un clásico como los dos volúmenes (que han influido intensamente en el diseño del presente manuscrito) como son [Aho-Ullman, 72a] y [Aho-Ullman, 72b]. Por tanto, dedicaremos una parte del curso al *parsing* (*Análisis Sintático*) sin más pretensiones que las de ubicar este proceso dentro del contexto de la compilación. En términos bibliográficos, alcanzaremos los tópicos del volumen I ([Aho-Ullman, 72a]) dejando el resto de los temas de [Aho-Ullman, 72b] para la asignatura correspondiente.
Antes de comenzar señalemos que los lenguajes de programación modernos son típicamente lenguajes dados por gramáticas libres de contexto. El uso de gramáticas en niveles más elevados de la jerarquía de Chomsky hace que el problema de palabra se vuelva un problema indecidible:

- En la Sección 1.5 ya hemos observado que el Problema de Palabra para Gramáticas Formales cualesquiera es indecidible, esto es, no puede existir ningún algoritmo que lo resuelva.
- Para gramáticas sensibles al contexto, el problema de decidir si el lenguaje que genera es vacío o no también es un problema indecidible.
- En cuanto al Problema de Palabra para gramáticas sensibles al contexto, el problema es **PSPACE**–completo para ciertas subclases, con lo que se hace impracticable para su uso en compilación.

De ahí la restricción a Gramáticas Libres de Contexto para los lenguajes de programación. Veremos en este Capítulo (véase la Subsección 4.3.3) que el problema de palabra para gramáticas libres de contexto es decidible. Esto significa que es posible diseñar un algoritmo/programa que realice la tarea siguiente:

Problema 50 (Detección de errores sintácticos CFG.)**.** *Dado un lenguaje de programacion $L \subseteq \Sigma^*$, mediante una gramática libre de contexto G que lo genera, y dada una palabra $\omega \in \Sigma^*$ (un fichero) decidir si ω es un programa sintácticamente válido (i.e. una palabra aceptada) para ese lenguaje de programación.*

Recordemos que una Gramática Libre de Contexto (CFG) o de Tipo 2 es dada por la siguiente definición.

Definición 77 (Gramáticas libres de contexto o de Tipo 2)**.** *Llamaremos gramática libre de contexto a toda gramática $G = (V, \Sigma, q, P)$ tal que todas las producciones de P son del tipo siguiente:*

$$A \mapsto \omega, \text{ donde } A \in V \text{ y } \omega \in (\Sigma \cup V)^*.$$

Un lenguaje **libre de contexto** *es un lenguaje generado por una gramática libre de contexto.*

El sistema de transición asociado a una gramática libre de contexto es el mismo que asociamos a una gramática cualquiera. Usaremos el símbolo $C \vdash_G C'$ para indicar que la configuración C' es deducible de la configuración C mediante computaciones de G.
Esta sería la primera acción asociada a un compilador: resolver el problema de palabra para un lenguaje de programación fijado a priori. En otras palabras, ser capaz de decidir si un fichero es un programa o devolver al programador un mensaje de error.
El problema de palabra en general admite también como input la gramática que lo genera. Sin embargo, la situación usual es que nuestro lenguaje de programación está fijado. Por tanto, el problema a resolver no tiene a la gramática como input sino, simplemente, la palabra. Esto es,

Problema 51 (Errores sintácticos con lenguaje pre–fijado.)**.** *Fijado un lenguaje de programación L decidir si una palabra $\omega \in \Sigma^*$ (un fichero) es un programa sintácticamente válido (i.e. una palabra aceptada) para ese lenguaje de programación.*

En este caso de lenguaje fijado no se pide generar "el programa que decide" como parte del problema. Al contrario, se pre-supone que se dispone de ese programa (parser) y se desea que sea eficiente. El modelo de algoritmo natural que aparece en este caso es el *Autómata con Pila* (Pushdown Automata, PDA) que discutiremos en el Capítulo siguiente.
En el presente Capítulo nos ocupamos de resolver el Problema 50 y mostraremos cómo reducir a formas normales las Gramáticas Libres de Contexto, lo que simplificará el análisis de la equivalencia con los PDA's.
En el Capítulo próximo mostraremos los aspectos relativos a la equivalencia entre ambas concepciones: la gramática libre de contexto (como *generador del lenguaje*) y los PDA's (como *reconocedor/algoritmo de decisión*), resolviendo de paso el Problema 51.
Dejaremos para el Capítulo último el problema de la traducción que acompaña el proceso de compilación de manera esencial.

4.2 Árboles de Derivación de una Gramática

Definición 78 (Formas Sentenciales y Formas Terminales)**.** *Llamamos formas sentenciales a todos los elementos ω de $(V \cup \Sigma)^*$. Llamaremos formas terminales a las formas sentenciales que sólo tienen símbolos en el alfabeto de símbolos terminales, es decir, a los elementos de Σ^*.*

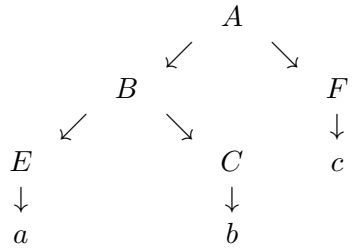
Definición 79 (Árbol de Derivación)**.** *Sea $G := (V, \Sigma, Q_0, P)$ una gramática libre de contexto, sea $A \in V$ una variable. Diremos que un árbol $\mathcal{T}_A := (V, E)$ etiquetado es un árbol de derivación asociado a G si verifica las propiedades siguientes:*

- *La raíz del árbol es un símbolo no terminal (i.e. una variable).*
- *Las etiquetas de los nodos del árbol son símbolos en $V \cup \Sigma \cup \{\lambda\}$.*

- *Cada nodo interior está etiquetado con un símbolo en V (i.e. un símbolo no terminal).*
- *Cada hoja está etiquetada con una variable, un símbolo terminal o λ*
- *Si un nodo está etiquetado con una variable X y sus descendientes (leídos de izquierda a derecha) en el árbol son $X_1, \ldots, X_k$ entonces, hay una producción $X \mapsto X_1 \cdots X_k$ en G.*

Ejemplo 10. *Hallar árboles de derivación para las gramáticas siguientes:*

- *$A \mapsto BF, B \mapsto EC, E \mapsto a, C \mapsto b, F \mapsto c$, de tal manera que la raíz sea A y las hojas estén etiquetadas con a, b, c en este orden.*



(4.1)

- *$Q_0 \mapsto zABz, B \mapsto CD, C \mapsto c, D \mapsto d, A \mapsto a$, de tal manera que la raíz sea Q_0 y las hojas estén etiquetadas (de izquierda a derecha) mediante z, a, c, d, z.*
- *$Q_0 \mapsto aQ_0bQ_0 \mid bQ_0aQ_0 \mid \lambda$. Escribe árboles de derivación cuyas hojas tengan la lectura siguiente:*
  - *Una sóla hoja con λ.*
  - *Un árbol con varias hojas, tales que leyendo la palabra se obtenga abab.*
  - *Un árbol distinto, con varias hojas tales que leyendo la palabra se obtenga abab.*

Proposición 80. *Sea $G := (V, \Sigma, Q_0, P)$ una gramática libre de contexto, sea $A \in V$ una variable. Sea $\mathcal{T}_A$ un árbol asociado a la gramática con raíz A. Sea $\omega \in (V \cup \Sigma)^*$ la forma sentencial obtenida leyendo de izquierda a derecha los símbolos de las hojas de $\mathcal{T}_A$. Entonces, $A \vdash_G \omega$. En particular, las formas sentenciales alcanzables desde el símbolo inicial Q_0 está representados por los árboles de derivación de Q_0.*

Demostración. Obvio a partir de la definición. □

Corolario 81. *Las palabras sobre el alfabeto Σ están en el lenguaje $L(G)$ generado por una gramática G si y solamente si existe un árbol de derivación cuyas hojas (leídas de izquierda a derecha) expresan la palabra ω.*

Demostración. Obvio. □

4.2.1 Un algoritmo incremental para la vacuidad.

Comenzaremos con un ejemplo de un algoritmo que será reutilizado, por analogía en los demás algoritmos de esta subsección. Su objeitvo consiste en mostrar que el problema de decidir si es o no vacío el lenguaje generado por una gramática libre de contexto.

Teorema 82 (Vacuidad de un lenguaje libre de contexto)**.** *El problema de la vacuidad de los lenguajes generados por gramáticas libres de contexto es decidible. Es decir, existe un algoritmo que toma como input una gramática libre de contexto G y devuelve una respuesta afirmativa si $L(G) \neq \emptyset$ y negativa en caso contrario.*

Demostración. Definimos el algoritmo siguiente:

INPUT: Una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$.
$M := \emptyset$
$N := \{A \in V \ : \ (A \mapsto a) \in P, \ a \in \Sigma^*\}$
while $N \neq M$ **do**
$M := N$
$N := \{A \in V \ : \ (A \mapsto a) \in P, \ a \in (N \cup \Sigma)^*\} \cup N.$
endwhile
if $Q_0 \in N$, **then** OUPUT SI
else OUTPUT NO
fi

Obsérvese que este algoritmo tiene la propiedad de que los sucesivos conjuntos M y N construidos en su recorrido son siempre subcojuntos del conjunto de símbolos no terminales V. Por tanto, es un algoritmo que acaba sus cálculos en todos los datos de entrada.
Veamos que este algoritmo realiza las tareas prescritas. Para ello, consideremos una cadena de subconjuntos N_i de V que reflejan los sucesivos pasos por el ciclo while. Escribamos $N_0 = \emptyset$ y denotemos por N_i al conjunto obtenido en el $i-$ésimo paso por el ciclo while, sin considerar la condición de parada. Esto es,

$$N_i := \{A \in V \ : \ (A \mapsto a) \in P, \ a \in (N_{i-1} \cup \Sigma)^*\} \cup N_{i-1}.$$

Está claro que tenemos una cadena ascendente

$$N_0 \subseteq N_1 \subseteq N_2 \subseteq \cdots \subseteq N_n \subseteq \cdots$$

Por construción observamos, además, que si existe un paso i tal que $N_i = N_{i+1}$, entonces, $N_i = N_m$ para todo $m \geq i + 1$.
Analicemos qué propiedades han de verificar las variables en N_i. Por inducción se probará lo siguiente:

Una variable $X \in V$ verifica que $X \in N_i$ si y solamente si existe un árbol de derivación de G de altura[1] *$i+1$ que tiene X como raíz y cuyas hojas están etiquetadas con símbolos en $\Sigma \cup \{\lambda\}$.*

Una vez probada esta propiedad, es claro que se tiene:

[1] Medimos altura por el número de nodos atravesados en el camino más largo.

- Sea i tal que nuestro algoritmo detiene sus cálculos tras i pasos por el ciclo while. Sea N el conjunto de variables calculado en ese paso. Entonces, $N = N_m$, $\forall m \geq i+1$.
- Si $Q_0 \in N$, entonces, $N = N_{i+1}$ y existe un árbol de derivación de la gramática de altura $i+2$ cuyas hojas son todo símbolos en $\Sigma \cup \{\lambda\}$ y cuya raíz es Q_0. Sea $\omega \in \Sigma^*$, la palabra descrita mediante la lectura (de izquierda a derecha) de las hojas del árbol. Entonces, $Q_0 \vdash_G \omega \in \Sigma^*$, luego $\omega \in L(G) \neq \emptyset$.
- Por otro lado, si $\omega \in L(G) \neq \emptyset$ habrá un árbol de derivación de G cuya raíz es Q_0 y cuyas hojas tienen sus etiquetas en $\Sigma \cup \{\lambda\}$ producen, leyendo de izquierda a derecha, ω. Sea m la altura de tal árbol ($m \geq 1$, obviamente) y, por tanto, $Q_0 \in N_{m-1} \subseteq N_{i+1} = N$ para cualquier m.

□

4.3 Formas Normales de Gramáticas.

El objetivo de esta Sección es la de realizar una serie de reducciones algorítmicas (transformaciones de gramáticas) hasta reducir una gramática libre de contexto a una gramática en Forma Normal de Chomsky. Las diferentes subsecciones están basadas en las progresivas reducciones y simplificaciones.

Definición 83. *Dos gramáticas libres de contexto G y G' se dicen equivalentes, si generan el mismo lenguajes, esto es, si $L(G) = L(G')$.*

De ahora en adelante, todas las transformaciones de gramáticas serán transformaciones que preserven la considión de "ser equivalentes".

4.3.1 Eliminación de Símbolos Inútiles o Inaccesibles

Definición 84 (Símbolos Inútiles)**.** *Sea $G := (V, \Sigma, Q_0, P)$ una gramática libre de contexto. Llamamos símbolos útiles de G a todos los símbolos (terminales o no) $X \in V \cup \Sigma$ tales que existen $\alpha, \gamma \in (V \cup \Sigma)^*$ y $\omega \in \Sigma^*$ de tal modo que:*

$$Q_0 \vdash_G \alpha X \gamma, \quad \text{y} \quad \alpha X \gamma \vdash_G \omega.$$

Los símbolos inútiles son los que no son útiles.

Ejemplo 11. *Consideremos la gramática $G := (\{Q_0, A, B\}, \{a, b\}, Q_0, P)$, donde las producciones de P son dadas por:*

$$P := \{Q_0 \mapsto a \mid A, A \mapsto AB, B \mapsto b\}.$$

Obsérvese que A, B, b son símbolos inútiles en esta gramática. La razón es que el lenguaje aceptado es $\{a\}$. Si, por el contrario, añadiéramos la producción $A \mapsto a$, entonces, todos ellos serían símbolos útiles.

Definición 85. *Sea $G := (V, \Sigma, Q_0, P)$ una gramática libre de contexto.*

- *Llamamos símbolos productivos (o fecundos) de G a todos los símbolos no terminales $X \in V$ tales que existe $\omega \in \Sigma^*$ tal que $X \vdash_G \omega$. Son improductivos (o infecundos) a los que no satisfacen esta propiedad.*

- *Llamamos símbolos accesibles de G a todos los símbolos (terminales o no) $X \in V \cup \Sigma$ tales que existen $\alpha, \gamma \in (V \cup \Sigma)^*$ de tal modo que:*

$$Q_0 \vdash_G \alpha X \gamma.$$

  Se llaman inaccesibles a los que no son accesibles.

Ejemplo 12. *Nótese que si X es un símbolo útil, se han de producir dos propiedades. De una parte, la propiedad $Q_0 \vdash_G \alpha X \gamma$ que nos dice que X es accesible. De otra parte, por estar en una gramática libre de contexto, ha de existir $\beta \in \Sigma^*$ tal que $X \vdash_G \beta$. Esto es necesario porque, al ser libre de contexto, todas las producciones se basan en reemplazar una variales por formas sentenciales. Si la variable X no alcanzara nunca una forma terminal en el sistema de transición, entonces, $\alpha X \beta$ tampoco alcanzaría una forma terminal contradiciendo el hecho de ser X útil. La existencia de $\beta \in \Sigma^*$ tal que $X \vdash_G \beta$ nos dice que X es un símbolo fecundo o productivo. En el siguiente ejemplo:*

$$P := \{Q_0 \mapsto AB \mid CD, A \mapsto AQ_0, B \mapsto b, C \mapsto Cc \mid \lambda, D \mapsto d\}.$$

El símbolo B es fecundo y accesible, pero es un símbolo inútil.

Proposición 86. *Si $G := (V, \Sigma, Q_0, P)$ es una gramática libre de contexto, entonces los símbolos útiles son productivos y accesibles. El recíproco no es cierto.*

Demostración. Es obvia la implicación enunciada. En cuanto a ejemplos que muestran que el recíproco no es en general cierto, baste con ver las gramáticas expuestas en los Ejemplos 11 y 12 anteriores. □

Proposición 87. *Si $G := (V, \Sigma, Q_0, P)$ es una gramática libre de contexto, y libre de símbolos infecundos, entonces todo símbolo es útil si y solamente si es accesible.*

Demostración. En ausencia de símbolos infecundos accesibilidad es sinónimo de utilidad. La prueba es la obvia. □

Proposición 88 (Eliminación de símbolos infecundos)**.** *Toda gramática libre de contexto es equivalente a una gramática libre de contexto sin símbolos infecundos. Además, dicha equivalencia puede hacerse de manera algorítmica.*

Demostración. El algoritmo es esencialmente el propuesto en el Teorema 82 anterior:

INPUT: Una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$.
$M := \emptyset$
$N := \{A \in V \ : \ (A \mapsto a) \in P, \ a \in \Sigma^*\}$
    while $N \neq M$ **do**
        $M := N$
        $N := \{A \in V \ : \ (A \mapsto a) \in P, \ a \in (N \cup \Sigma)^*\} \cup N.$

endwhile
if $Q_0 \notin M$, **then** OUTPUT $(\{Q_0\}, \Sigma, Q_0, \emptyset)$
else OUTPUT $\bar{G} := (V \cap N, \Sigma, Q_0, \bar{P})$, donde $\bar{P}$ son las producciones de P que involucran solamente símbolos en $(V \cap N) \cup \Sigma \cup \{\lambda\}$
fi

Por la prueba del Teorema 82, sabemos que N es justamente el conjunto de variables productivas y el algoritmo realiza la tarea pretendida. □

4.3.1.1 Eliminación de Símbolos Inaccesibles.

Teorema 89. *[Eliminación de Símbolos Inaccesibles] Toda gramática libre de contexto es equivalente a una gramática libre de contexto sin símbolos inaccesibles. Además, dicha equivalencia puede hacerse de manera algorítmica.*

Demostración. El siguiente algoritmo elimina símbolos inaccesibles de una gramática libre de contexto. La demostración de que es un algoritmo y de que realiza la tarea prevista es análoga a la demostración del Teorema 82 anterior. Nótese que, de facto, el algoritmo calcula los símbolos que sí son accesibles.

INPUT: Una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$.
$M := \{Q_0\}$
$N := \{X \in V \cup \Sigma \ : \ \exists A \in M, \ \exists \alpha, \beta \in (V \cup \Sigma)^*, \text{ con } A \mapsto \alpha X \beta \text{ en } P\}$.

while $N \neq M$ **do**
$M := N$
$N := \{X \in V \cup \Sigma \ : \ \exists A \in M, \ \exists \alpha, \beta \in (V \cup \Sigma)^*, \text{ con } A \mapsto \alpha X \beta \text{ en } P\}$.
endwhile

OUTPUT: La gramática $\bar{G} = (\bar{V}, \bar{\Sigma}, Q_0, \bar{P})$, con
$\bar{V} := N \cap V, \ \ \bar{\Sigma} := N \cap \Sigma$,
$\bar{P} :=$ {Las producciones de P que sólo contienen los elementos de $\bar{V} \cup \bar{\Sigma}$}.

□

4.3.1.2 Eliminación de Símbolos Inútiles.

Teorema 90. *[Eliminación de Símbolos Inútiles] Toda gramática libre de contexto es equivalente a una gramática sin símbolos inútiles. Además, esta equivalencia es calculable algorítmicamene.*

Demostración. Utilizaremos un algoritmo que combina los dos algoritmos descritos anteriormente y el enunciado de la Proposición 87. Primero eliminamos los símbolos infecundos y luego los inaccesibles.

INPUT: Una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$.

Eliminar Símbolos Infecundos

$M := \emptyset$
$N := \{A \in V \ : \ (A \mapsto a) \in P,\ a \in \Sigma^*\}$

while $N \neq M$ **do**
$M := N$
$N := \{A \in V \ : \ (A \mapsto a) \in P,\ a \in (N \cup \Sigma)^*\} \cup N.$
endwhile

$G_1 := (V_1, \Sigma, Q_0, P_1)$, donde
$V_1 := V \cap N$,
$P_1 := \{$Las producciones en P que no involucran símbolos fuera de $V_1 \cup \Sigma\}$.

Eliminar Símbolos Inaccesibles de G_1

$M := \{Q_0\}$
$N := \{X \in V_1 \cup \Sigma \ : \ \exists A \in M,\ \exists \alpha, \beta \in (V \cup \Sigma)^*,\ \text{con } A \mapsto \alpha X \beta \ \text{ en } P\}$.

while $N \neq M$ **do**
$M := N$
$N := \{X \in V_1 \cup \Sigma \ : \ \exists A \in M,\ \exists \alpha, \beta \in (V \cup \Sigma)^*,\ \text{con } A \mapsto \alpha X \beta \ \text{ en } P\}$.
endwhile

OUTPUT: La gramática $\bar{G} = (\overline{V_1}, \overline{\Sigma}, Q_0, \overline{P_1})$, con
$\overline{V_1} := N \cap V_1$, $\overline{\Sigma} := N \cap \Sigma$,
$\overline{P} := \{$Las producciones de P que sólo contienen los elementos de $\overline{V_1} \cup \overline{\Sigma}\}$.

□

4.3.2 Transformación en Gramáticas Propias.

En nuestro camino hasta la forma normal de Chomsky, continuaremos con transformaciones de las gramáticas libes de contexto hasta obtener gramáticas propias.

4.3.2.1 Eliminación de $\lambda-$producciones.

Definición 91. *Sea $G = (V, \Sigma, Q_0, P)$ una gramática libre de contexto.*

a. *Llamaremos $\lambda-$producciones en G a todas las producciones de la forma $X \mapsto \lambda$, donde $X \in V$ es un símbolo no terminal.*

b. *Diremos que la gramática G es $\lambda-$libre si verifica una de las dos propiedades siguientes:*

- *O bien no posee $\lambda-$producciones,*
- *o bien la única $\lambda-$producción es de la forma $Q_0 \mapsto \lambda$ y Q_0 no aparece en el lado derecho de ninguna otra producción de P (es decir, no existe ninguna producción de la forma $X \mapsto \alpha Q_0 \beta$, con $\alpha, \beta \in (V \cup \Sigma)^*$).*

Ejemplo 13. *Consideremos la gramática cuyas producciones son:*

$$Q_0 \mapsto aQ_0bQ_0 \mid bQ_0aQ_0 \mid \lambda.$$

No es una gramática $\lambda-$libre.

Teorema 92 (Transformación a Gramática $\lambda-$libre)**.** *Toda gramática libre de contexto es equivalente a una gramática $\lambda-$libre. Además, dicha equivalencia es calculable algorítmicamente.*

Demostración. El algoritmo comienza con una tarea que repite esencialmente lo hecho en algoritmos anteriores. Se trata de hacer desaparecer las variables que van a parar a la palabra vacía λ; pero de manera selectiva. No las eliminamos completamente porque podrían ir a parar a constantes o formas terminales no vacías.
Hallar $V_\lambda := \{A \in V \ : \ A \vdash_G \lambda\}$.
A partir del cálulo de V_λ procedemos de la forma siguiente:

a. Calculamos el nuevo sistema de producciones $\overline{P}$ del modo siguiente:

- Consideremos todas las producciones de la forma siguiente:

$$A \mapsto \alpha_0 B_1 \alpha_1 \cdots B_k \alpha_k,$$

donde $\alpha_i \notin V_\lambda^*$, $B_i \in V_\lambda$ y no todos los α_i son iguales a λ. Definamos

$$\overline{P} := \overline{P} \cup \{A \mapsto \alpha_0 X_1 \alpha_1 \cdots X_k \alpha_k \ : \ X_i \in \{B_i, \lambda\}\}.$$

- Consideremos todas las producciones de la forma siguiente:

$$A \mapsto B_1 \cdots B_k,$$

donde $B_i \in V_\lambda$. Definamos:

$$\overline{P} := \overline{P} \cup (\{A \mapsto X_1 \cdots X_k \ : \ X_i \in \{B_i, \lambda\}\} \setminus \{A \mapsto \lambda\}).$$

b. Eliminamos todas las $\lambda-$producciones restantes.

c. Finalmente, si $Q_0 \in V_\lambda$ sea $\overline{V} := V \cup \{Q_0'\}$, con $Q_0' \notin V$. Y añadamos

$$\overline{P} = \overline{P} \cup \{Q_0' \mapsto Q_0 \mid \lambda\}.$$

En otro caso, $\overline{V} = V$.

El output será la gramática $\overline{G} := (\overline{V}, \Sigma, Q_0', \overline{P})$ y satisface las propiedades pretendidas. □

Nota 93. *La eliminación de $\lambda-$producciones puede tener un coste exponencial en el máximo de las longitudes de las formas sentenciales (en $(\Sigma \cup V)^*$) que aparecen a la derecha de las producciones de la gramática dada.*

4.3.2.2 Eliminación de Producciones Simples o Unarias

Definición 94 (Producciones Simples o Unarias)**.** *Se llaman producciones simples (o unarias) a las producciones de una gramática libre de contexto de la forma $A \mapsto B$, donde A y B son símbolos no terminales.*

Teorema 95. *[Eliminación de Producciones Simples] Toda gramática $\lambda-$libre es equivalente a una gramática $\lambda-$libre y sin producciones simples. Esta equivalencia es calculable algorítmicamente.*

Demostración. El algoritmo tiene dos partes. La primera parte sigue el mismo esquema algorítmico usado en resultados anteriores. La segunda parte se dedica a eliminar todas las producciones simples.

- *Clausura Transitiva de símbolos no terminales.* Se trata de calcular, para cada $A \in V$, el conjunto siguiente:

$$W_A := \{B \in V \ : \ A \vdash B\} \cup \{A\}.$$

Nótese que se trata de la clausura transitiva en el grafo $(V, \rightarrow)$, inducido por el sistema de transición sobre el conjunto de variables. El algoritmo obvio funciona del modo siguiente:

INPUT: Una gramática libre de contexto $G := (V, \Sigma, Q_0, P)$ y $\lambda-$libre.

Para cada $A \in V$ calcular

$M_A := \emptyset$

$N_A := \{A\}$

while $N_A \neq M$ **do**

$M := N_A$

$N_A := \{C \in V \ : \ B \mapsto C \text{ está en P, y } B \in N_A\} \cup N_A$

endwhile

OUTPUT: Para cada $A \in V$, N_A.

- También podemos definir el conjunto de los antepasados de una variable

$$V_A := \{B \in V \ : \ B \vdash A\} := \{B \in V \ : \ A \in W_B\}.$$

Es calculable por el algoritmo anterior del modo obvio.

- *Eliminar las producciones simples.* Para cada variable B tal que existe una producción simple $A \mapsto B$ en P, procederemos como sigue:
  - Hallar todos los X's tales que $B \in W_X$ (o, equivalentemente, los X's en V_B) .
  - Para cada producción $B \mapsto \alpha$ que no sea producción simple, añadir a P la producción $X \mapsto \alpha$.
  - Eliminar toda producción del tipo $X \mapsto B$.

Nótese que cada iteración de la parte segunda del proceso añade producciones no simples y elimina al menos una producción simple. Con ello se alcanza el objetivo buscado. □

4.3.2.3 Hacia las Gramáticas Propias.

Definición 96. *Diremos que una gramática libre de contexto $G := (Q, \Sigma, Q_0, F, \delta)$ es acíclica (o libre de ciclos) si no existe ningún símbolo no terminal $A \in V$ tal que existe una computación no trivial (en el sistema de transición asociado):*

$$A \to \omega_1 \to \cdots \to \omega_k = A.$$

Definición 97 (Gramáticas Propias)**.** *Diremos que una gramática libre de contexto $G := (Q, \Sigma, Q_0, F, \delta)$ es propia si verifica las siguientes propiedades:*

- *G es acíclica,*
- *G es $\lambda-$libre,*
- *G es libre de símbolos inútiles.*

Lema 98. *[Interacción entre los algoritmos expuestos] Se dan las siguientes propiedades:*

a. Si G es una gramática libre de contexto que es $\lambda-$libre y está libre de producciones simples, entonces G es acíclica.

b. Sea G es una gramática libre de contexto, $\lambda-$libre. Sea $\bar{G}$ la gramática obtenida después de aplicar a G el algoritmo de eliminación de producciones simples descrito en la demostración del Teorema 95. Entonces, $\bar{G}$ sigue siendo $\lambda-$libre.

c. Sea G es una gramática libre de contexto, libre de producciones simples y $\lambda-$libre. Sea $\bar{G}$ la gramática obtenida después de aplicar a G el algoritmo de eliminación de símbolos inútiles descrito en la demostración del Teorema 90. Entonces, $\bar{G}$ sigue siendo libre de producciones simples y $\lambda-$libre.

Demostración.

a. Supongamos que la gramática fuera $\lambda-$libre y libre de producciones simples, pero hubiera un estado que generara un ciclo. Es decir, supongamos que existe:

$$A \to \omega_1 \to \cdots \to \omega_k \to A,$$

con $k \geq 1$. Entonces, puedo suponer que

$$\omega_k := \alpha_0 X_1 \alpha_1 \cdots \alpha_{n-1} X_n \alpha_n,$$

donde $\alpha_i \in \Sigma^*$, $X_i \in V$. En primer lugar, obsérvese que, como estamos hablando de gramáticas libres de contexto, las únicas producciones que se aplican son de la forma $B \to \gamma$. Si alguno de los α_i fuera distinto de la palabra vacía λ, no habría forma de "borrarlos" usando las producciones libre de contexto (para quedarnos solamente con un símbolo no terminal como A). Por tanto, $\alpha_i = \lambda$ para cada i, $0 \leq i \leq n$. En conclusión, sólo tenemos (como última acción):

$$\omega_k = X_1 \cdots X_n \to A.$$

Si, además, $n \geq 2$, con una única producción libre de contexto, no podríamos eliminar nada más que un símbolo no terminal. Con lo cual no podríamos obtener A. Por tanto, tenemos, en realidad, $n \leq 2$. Tenemos dos casos $n = 1$ o $n = 2$. Si $n = 1$ tendremos:

$$\omega_k = X_1 \to A.$$

En el este caso debería existir la producción simple $X_1 \mapsto A$, pero hemos dicho que nuestra gramática es libre de producciones simples. Por tanto, el segundo caso ($n = 1$) no puede darse. Nos queda un único caso $n = 2$ y tendremos:

$$\omega_k = X_1 X_2 \to A.$$

En este caso hay dos opciones simétricas, con lo que discutiremos sólo una de ellas: $X_1 = A$ y $X_2 \mapsto \lambda$ es una producción. Pero nuestra gramática es $\lambda-$libre. Si hay una producción $X_2 \mapsto \lambda$ es sólo porque X_2 es el símbolo inicial $X_2 = Q_0$. Por tanto, tenemos una computación:

$$A \to \omega_1 \to \cdots \to \omega_{k-1} \to AQ_0 \to A,$$

Pero, para llegar a AQ_0 desde A tiene que ocurrir que Q_0 esté en la parte derecha de alguna producción (lo cual es imposible por la propia defínción de $\lambda-$libre. Luego no uede haber ciclos.

b. Retomemos el algoritmo descrito en la prueba del Teorema 95. Una vez hemos calculado V_A para cada A, y tratamos las producciones unarias del tipo $A \mapsto B$ del modo siguiente:

- Hallar todos los X's tales que $B \in W_X$.
- Para cada producción $B \mapsto \alpha$ que no sea producción simple, añadir a P la producción $X \mapsto \alpha$.
- Eliminar la producción $A \mapsto B$.

Ahora bien, si G es una gramática $\lambda-$libre, y si Q_0 es el estado inicial, Q_0 no puede estar en la derecha de ninguna producción. En particular, no existen producciones simples de la forma $A \mapsto Q_0$. Por tanto, toda producción simple

$A \mapsto B$ de P verifica que $B \neq Q_0$. De otro lado, como G es $\lambda-$ libre, para todo $B \neq Q_0$, las producciones de la forma $B \mapsto \alpha$ verifican que $\alpha \neq \lambda$. Por tanto, ninguna de las producciones que añadimos en este proceso es una $\lambda-$producción. Y, por tanto, $\bar{G}$ sigue siendo $\lambda-$libre.

c. Basta con observar que el algoritmo de eliminación de símbolos inútiles solamente elimina símbolos y producciones que los involucran. Pero no añade producciones. Por ello, si no había en G ninguna producción simple, tampoco la habrá en $\bar{G}$. Si G era $\lambda-$libre, también lo será $\bar{G}$ puesto que no añadimos $\lambda-$ producciones.

□

Teorema 99. *Toda gramática libre de contexto es equivalente a una gramática propia. Dicha equivalencia es calculable algorítmicamente.*

Demostración. Basta con unir los algoritmos antes expuestos en el orden adecuado que indica el Lema 98 anterior. El proceso será el siguiente:

INPUT: Una gramática G libre de contexto.

- *Hallar* G_1 la gramática $\lambda-$libre obtenida aplicando a G, mediante el algoritmo del Teorema 92.
- *Hallar* G_2 la gramática obtenida de aplicar a G_1 el algoritmo del Teorema 95.
- *Hallar* G_3 la gramática obtenida de aplicar a G_2 el algoritmo del Teorema 90.

OUTPUT: G_3

El algoritmo anterior realiza la tarea prescrita. La gramática G_1 es claramente $\lambda-$libre como consecuencia del Teorema 92. Como consecuencia del apartado 2 del Lema 98, como la gramática G_1 es $\lambda-$libre, también es $\lambda-$libre la gramática G_2. De otro lado, el Teorema 95 nos garantiza que G_2 es libre de producciones simples.
Como consecuencia del apartado 3 del Lema 98, la gramática G_3 sigue siendo una gramática $\lambda-$libre y libre de producciones simples. Asimismo, el Teorema 90 nos garantiza que G_3 es libre de símbolos inútiles.
Finalmente, el apartado 1 del Lema 98, nos garantiza que G_3 es acíclica. Por tanto, verifica todas las propiedades para ser una gramática propia. □

4.3.3 El Problema de Palabra para Gramáticas Libres de Contexto es Decidible.

En la Sección 1.5 hemos descrito algunos resultados relativos a la indecidibilidad del Problema de palabra de sistemas de semi–Thue (o gramáticas de Tipo 0). En esta subsección mostaremos que este problema es decidible para gramáticas libres de contexto. Es decir, se trata de mostrar un algoritmo que resuelve el problema siguiente:

Problema de Palabra para Gramáticas libres de Contexto. Dada una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$ y dada una palabra $\omega \in \Sigma^*$, decidir si $\omega \in L(G)$.

Lema 100. *Sea $G = (V, \Sigma, Q_0, P)$ una gramática libre de contexto y $\lambda-$libre. Sea $\omega \in L(G)$ una palabra aceptada por la gramática y sea:*

$$Q_0 \to \alpha_1 \to \alpha_2 \to \cdots \to \alpha_n = \omega,$$

una derivación aceptadora de ω, donde $\alpha_i \in (V \cup \Sigma)^*$ son formas sentenciales de la gramática. Entonces, la longitud de cada una de estas formas sentenciales verifica:

$$|\alpha_i| \leq |\omega|, \ \forall i.$$

Demostración. De hecho, basta con observar que si tenemos dos configuraciones (i.e. dos formas sentenciales) $c \to_G c'$ y si la gramática es $\lambda-$libre, entonces o bien $c' = \lambda$ (en cuyo caso $c = Q_0$ forzosamente) o bien $|c| \leq |c'|$ (dado que no suprimimos ninguna variable, al reemplazarla nos sale, al menos, un símbolo y la longitud no puede disminuir). □

Teorema 101. *El problema de palabra es decidible para gramáticas libres de contexto.*

Demostración. Basta con usar el Lema anterior. El procedimiento es el siguiente: En primer lugar, trasnformamos nuestra gramática original en una gramática $\lambda-$libre. posteriormente, dado ω, construiremos un grafo $G_\omega := (N_\omega, E_\omega)$ con ls reglas siguientes:

- Los vértices del grafo N_ω son todas las palabras de $(V \cup \Sigma)^*$ de longitud menor o igual que la longitud de ω.
- Las aristas del grafo E_ω son los pares $(c, c') \in N_\omega$ tales que $c \to_G c'$.

A partir de del grafo G_ω, calculamos la clausura transitiva de Q_0: $CT_G(Q_0)$. Entonces, usando el Lema anterior, ω está en $L(G)$ si y solamente si está en la clausura transitiva de Q_0. □

Nota 102. *Decidibilidad no significa eficiencia. Es decir, el hecho de la existencia de un algoritmo para el problema de palabra no singifica de modo inmediato que se pueda usar ese algoritmo para la detección de errores. De hecho, debe haber un pre–proceamiento para el cálculo del autómata (con pila, como veremos) que decide el problema de palabra y luego sólo debe usarse el autómata para cada palabra concreta. En otro caso estaríamos consumiendo una enormidad de tiempo de cálculo para cada verificación de correctitud de una palabra.*

4.3.4 Transformación a Formal Normal de Chomsky.

Definición 103 (Forma Normal de Chomsky)**.** *Una gramática libre de contexto $G := (Q, \Sigma, Q_0, F, \delta)$ se dice que está en forma normal de Chomsky si es $\lambda-$libre y las únicas producciones (exceptuando, eventualmente, la única $\lambda-$ producción $Q_0 \mapsto \lambda$), son exclusivamente de uno de los dos tipos siguientes.*

- $A \mapsto b$, *con* $A \in V$ *y* $b \in \Sigma$,
- $A \mapsto CD$, *con* $A, C, D \in V$.

Nótese que es acíclica porque carece de producciones unarias. En la definición, bien podrímos haber supuesto que es propia sin cambiar la esencia de la definición. Hemos dejado la habitual.
De otro lado, debe señalarse que el modelo se corresponde al modelo de codificación de polinomios (en este caso sobre anillos no conmutativos) llamado *"straight–line program"*.

Teorema 104 (Transformación a forma normal de Chomsky)**.** *Toda gramática libre de contexto es equivalente a una gramática libre de contexto en forma normal de Chomsky. Además, esta equivalencia es algorítmicamente computable.*

Demostración. Bastará con que demos un algoritmo que transforma gramáticas propias en gramáticas en forma normal de Chomsky. Así, supongamos que tenemos una gramática $G = (V, \Sigma, Q_0, P)$ propia. Procederemos del modo siguiente:
Definamos un par de clases $\bar{V}$ y $\bar{P}$ de símbolos no terminales y producciones, conforme a las reglas siguientes:

- Inicializar con $\bar{V} := V$, $\bar{P} = \emptyset$.
- Si $Q_0 \mapsto \lambda$ está en P, añadir $Q_0 \mapsto \lambda$ a $\bar{P}$ sin modificar $\bar{V}$.
- Si en P hay una producción del tipo $A \mapsto a \in \Sigma$ entonces, añadir $A \mapsto a$ a $\bar{P}$ sin modificar $\bar{V}$.
- Si en P hay una producción del tipo $A \mapsto CD$, con $C, D \in V$, está en P, entonces, añadir $A \mapsto CD$ a $\bar{P}$ sin modificar $\bar{V}$.
- Finalmente, Para cada producción en P del tipo

  $$A \mapsto X_1 \cdots X_k,$$

  con $X_i \in V \cup \Sigma$ que no sea de ninguno de los tres tipos anteriores[2] realizar las tareas siguientes:

  - Para cada i tal que $X_i \in V$, no modificar $\bar{V}$

[2]Obsérvese que $k \geq 2$ puesto que no hay producciones simples. Si $k = 2$, al no ser de ninguno de los tipos anteriores, son o bien dos símbolos en Σ o bien uno es un símbolos en Σ y el otro está en V. En cualquier caso se aplica el mismo método.

- Para cada i tal que $X_i \in \Sigma$, añadir a $\bar{V}$ una nueva variable $\bar{X}_i$, distinta a todas las que ya estuvieran en $\bar{V}$. Añadir a $\bar{P}$ la producción $\bar{X}_i \mapsto X_i$ en este caso. (Obs'ervese que, en este caso, aparece una producción del Tipo 1).
- Añadir $\bar{P}$ la producción $A \mapsto X'_1 \cdots X'_k$, donde X'_i viene dada por:

$$X'_i := \begin{cases} X_i, & \text{si } X_i \in V \\ \bar{X}_i, & \text{en otro caso} \end{cases}$$

- Si $k = 2$, no modificar.
- Si $k > 2$, reemplazar en $\bar{P}$, la producción $A \mapsto X'_1 \cdots X'_k$ por una cadena de producciones:

$$A \mapsto X'_1 Y_2, Y_2 \mapsto X'_2 Y_3, \cdots, Y_{k-1} \mapsto X'_{k-1} X'_k,$$

añadiendo a $\bar{V}$ las variables $\{Y_2, \ldots, Y_{k-1}\}$.

- OUTPUT: $(\bar{V}, \Sigma, Q_0, \bar{P})$.

Es claro que el algoritmo descrito verifica las propiedades deseadas. □

Nota 105. *Obsérvese que los árboles de derivación asociados a gramáticas en forma normal de Chomsky son árboles binarios cuyas hojas vienen de nodos con salida unaria.*

4.3.5 Forma Normal de Greibach

Es otra "normalización" de las gramáticas libres de contexto que hace referencia al trabajo de Sheila A. Greibach en Teoría de Autómatas. Si la Forma Normal de Chomsky se corresponde con la presentación de polinomios como straight–line programs, la forma de Greibach se corresponde a la codificación mediante monomios.

Definición 106 (Producciones Monomiales)**.** *Una gramática $G := (V, \Sigma, Q_0, P)$ se dice en forma normal de Greibach si es $\lambda-$libre y las únicas producciones (exceptuando, eventualmente, la única $\lambda-$ producción $Q_0 \mapsto \lambda$) pertenecen al tipo siguiente:*

$$A \mapsto \alpha X_1 \cdots X_k,$$

donde $A \in V$ es una variable, $\alpha \in \Sigma$ es un símbolo terminal (posiblemente λ) y $X_1, \ldots, X_k \in V^*$ es una lista (posiblemente vacía) de símbolos no terminales (variables en V) entre los cuales no está el símbolo inicial Q_0.

Obviamente toda gramática libre de contexto es equivalente a una gramática en Forma Normal de Greibach.

4.4 Cuestiones y Problemas

4.4.1 Cuestiones

Cuestión 22. *Comprobar, utilizando las siguientes producciones de una gramática G, que al convertir una gramatica a $\lambda-$libre, puede quedar con símbolos inútiles:*

$$S \mapsto a \mid aA, A \mapsto bB, B \mapsto \lambda.$$

Cuestión 23. *Decidir si existe un algoritmo que realice la tarea siguiente:*
Dada una gramática libre de contexto G y dadas dos formas sentenciales de la gramática c y c', el algoritmo decide si $c \vdash_G c'$.

Cuestión 24. *Sean L_1 y L_2 dos lenguajes libres de contexto. Decidir si es libre de contexto el lenguaje siguiente:*

$$L := \bigcup_{n \geq 1} (L_1)^n (L_2)^n.$$

Cuestión 25. *Hallar una estimación del número de pasos necesarios para generar una palabra de un lenguaje libre de contexto, en el caso en que la gramática que lo genera esté en forma normal de Chomsky.*

Cuestión 26. *Discutir si alguno de los siguientes lenguajes es un lenguaje incontextual:*

a. $\{\omega \in \{a,b\}^* \ : \ \omega = \omega^R, \ \forall x, y \in \{a,b\}^*, \omega \neq xabay\}$.

b. $\{a^i b^j c^k \ : \ i = j \vee j = k\}$.

c. $\{a^i b^j c^k d^l \ : \ (i = j \wedge k = l) \vee (i = l \wedge j = k)\}$.

d. $\{xcy \ : \ x, y \in \{a,b\}^* \wedge \sharp_a(x) + \sharp_b(y) \in 2\mathbb{Z} \wedge |x| = |y|\}$.

4.4.2 Problemas

Problema 52. *Dada una gramática libre de contexto G, con las siguientes producciones:*

$$Q_0 \mapsto AB \mid 0Q_01 \mid A \mid C, A \mapsto 0AB \mid \lambda, B \mapsto B1 \mid \lambda.$$

Se pide:

- *Eliminar los símbolos inútiles*
- *Convertirla en $\lambda-$libre*
- *Eliminar las producciones unitarias*

Problema 53. *Eliminar las variables improductivas en la gramática G con las siguientes producciones:*

$$Q_0 \mapsto A \mid AA \mid AAA, \ A \mapsto ABa \mid ACa \mid a, B \mapsto ABa \mid Ab \mid \lambda,$$

$$C \mapsto Cab \mid CC, D \mapsto CD \mid Cd \mid CEa, E \mapsto b.$$

Eliminar los símbolos inaccesibles en la gramática resultante.

Problema 54. *Hallar una gramática $\lambda-$libre equivalente a la siguiente:*

$$Q_0 \mapsto aQ_0a \mid bQ_0b \mid aAb \mid bAa, A \mapsto aA \mid bA \mid \lambda.$$

¿Es una gramática propia?.

Problema 55. *Hallar una gramática propia equivalente a la siguiente:*

$$Q_0 \mapsto XY, X \mapsto aXb \mid \lambda, Y \mapsto bYc \mid \lambda.$$

Problema 56. *Sea $G = (V, \Sigma, Q_0, P)$ la gramática libre de contexto dada por las propiedades siguientes:*

- $V := \{Q_0, X, Y, Z, T\}$,
- $\Sigma := \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, \times, (,)\}$,
- *Las producciones en P están dadas por:*

$$Q_0 \mapsto X \mid X + Q_0, X \mapsto T \mid Y \times Z,$$

$$Y \mapsto T \mid (X + Q_0), Z \mapsto Y \mid Y \times Z,$$

$$T \mapsto 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.$$

Se pide:

a. *Hallar la clase de formas terminales de esta gramática.*

b. *Hallar el lenguaje generado por esta gramática.*

c. *Eliminar producciones unarias.*

d. *Eliminar producciones inútiles.*

e. *Convertirla en una gramática propia.*

f. *Transformarla en forma Normal de Chomsky.*

Problema 57. *Hacer los mismos pasos 3 a 6 del problema anterior con la gramática siguiente:*

$$Q_0 \mapsto A \mid B, A \mapsto aA \mid aAb \mid a, B \mapsto Bd \mid ABa \mid b.$$

Problema 58. *Eliminar $\lambda-$producciones y hacer los mismos pasos del problema anterior con la gramática siguiente:*

$$Q_0 \mapsto ABQ_0 \mid BAQ_0 \mid \lambda, A \mapsto bAA \mid a, B \mapsto aBB \mid b.$$

Problema 59. *Dar un algoritmo que decida si el lenguaje generado por una gramática libre de contexto es finito o infinito.*

Chapter 5

Autómatas con Pila.

Contents

5.1 Noción de Autómatas con Pila. 91
5.1.1 Las Pilas como Lenguaje (Stacks). 91
5.2 Sistema de Transición Asociado a un Autómata con Pila. 94
5.2.1 Modelo gráfico del sistema de transición. 95
5.2.2 Transiciones: Formalismo. 95
5.2.3 Codificación del Autómata con Pila. 97
5.3 Lenguaje Aceptado por un Autómata con Pila. 100
5.4 Equivalencia con Gramáticas Libres de Contexto. 105
5.5 Propiedades Básicas 107
5.6 Problemas 109
5.6.1 Problemas 109

5.1 Noción de Autómatas con Pila.

Antes de pasar a definir un autómata con pila (o *Pushdown Automata*, PDA) recordemos (superficialmente) la estructura de datos pila (stack) vista como lenguaje formal y las funciones y relaciones que la caracterizan.

5.1.1 Las Pilas como Lenguaje (Stacks).

Podemos identificar las pilas con ciertos lenguajes formales sobre un nuevo alfabeto Γ. Comenzaremos añadiendo un nuevo símbolo Z_0 que no está en Γ. Las pilas (stacks) son elementos del lenguaje: $Z_0 \cdot \Gamma^*$. El símbolo Z_0 se identificará con el significado *Fondo de la Pila*[1]
Tendremos unas ciertas funciones sobre pilas:

[1] El ímbolo de 'fondo de la pila' no está suficientemente estandarizado. Diversos autores usan diveras variantes de símbolos como $\S, \sharp, \$, []$ y otros. Usaremos Z_0 como la simplificación menos molesta de todas esas variaciones.

- `empty`: Definimos la aplicación

$$\texttt{empty} : Z_0 \cdot \Gamma^* \longrightarrow \{0, 1\},$$

dada mediante:

$$\texttt{empty}(Z_0 Z) := \begin{cases} 1, & \text{si } Z = \lambda \\ 0, & \text{en otro caso} \end{cases}$$

- `top`: Definimos la aplicación

$$\texttt{top} : Z_0 \cdot \Gamma^* \longrightarrow \Gamma \cup \{\lambda\},$$

mediante la regla siguiente:
Dada una pila $Z_0 \cdot Z \in Z_0 \cdot \Gamma^*$ (con $Z = z_1 \cdots z_n \in \Gamma^*$),

$$\texttt{top}(Z_0 Z) := \begin{cases} z_n \in \Gamma, & \text{si } Z = z_1 \cdots z_n \in \Gamma^*,\ Z \neq \lambda \\ Z_0, & \text{en caso contrario} \end{cases}$$

Obsérvese que hemos elegido leer Z_0 cuando la pila está vacía[2]

- `push`: Apilar (empujar) una pila encima de otra. Definimos la aplicación

$$\texttt{push} : Z_0 \cdot \Gamma^* \times \Gamma^* \longrightarrow Z_0 \cdot \Gamma^*,$$

mediante la regla siguiente:
Dada una pila $Z_0 \cdot Z \in Z_0 \cdot \Gamma^*$ (con $Z = z_1 \cdots z_n \in \Gamma^*$), y una palabra $x \in \Gamma^*$, dada mediante: $x := x_1 \cdots x_r$, definimos

$$\texttt{push}(Z_0 Z, x) := Z_0 z_1 \cdots z_n x_1 \cdots x_r \in Z_0 \cdot \Gamma^*.$$

- `pop` *(Pull Out the toP):* Definimos la aplicación

$$\texttt{pop} : Z_0 \cdot \Gamma^* \longrightarrow Z_0 \cdot \Gamma^*,$$

mediante la regla siguiente:
Dada una pila $Z_0 \cdot Z \in Z_0 \cdot \Gamma^*$, definimos $\texttt{pop}(Z_0 Z)$ como el resultado de eliminar $\texttt{top}(Z_0 Z)$, esto es

$$\texttt{pop}(Z_0 Z) := \begin{cases} Z_0 z_1 \cdots z_{n-1} \in Z_0 \cdot \Gamma^*, & \text{si } Z = z_1 \cdots z_n \in \Gamma^*,\ Z \neq \lambda \\ Z_0, & \text{en caso contrario} \end{cases}$$

Obsérvese que el símbolo "fondo de pila" no se borra al hacer la operación `pop`.

Nota 107. *Una de las propiedades básicas de las operaciones es, obviamente, la siguiente:*

$$\texttt{push}(\texttt{pop}(Z_0 Z), \texttt{top}(Z_0 Z)) = Z_0 Z.$$

[2] Podríamos también haber definido $\texttt{top}(Z_0) = \lambda$. Esta corrección subjetiva la hemos hecho para enfatizar el hecho de que la pila está vacía. En caso de hacer la elección $\texttt{top}(Z_0) = \lambda$, deberían modificarse todas las notaciones que siguen de modo conforme.

Definición 108 (Non–Deterministic Pushdown Automata)**.** *Un autómata con pila (o Pushdown Automata) indeterminista es una lista* $A := (Q, \Sigma, \Gamma, q_0, F, Z_0, \delta)$ *donde:*

- Q *es un conjunto finito cuyos elementos se llaman estados y que suele denominarse espacio de estados,*
- Σ *es un conjunto finito (alfabeto),*
- Γ *es un conjunto finito llamado "alfabeto de la pila".*
- q_0 *es un elemento de* Q *que se denomina estado inicial,*
- F *es un subconjunto de* Q*, cuyos elementos se denominan estados finales aceptadores,*
- Z_0 *es un símbolo que no está en el alfabeto* Γ *y que se denomina "fondo de la pila".*
- δ *es una correspondencia:*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\}) \longrightarrow Q \times \Gamma^*,$$

que se denomina función de transición y que ha de verificar la propiedad siguiente[3] *para cada lista* $(q, x, A) \in Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\})$*:*

$$\sharp\left(\{(q, \omega) \in Q \times \Gamma^* \ : \ \delta(q, x, A) = (q, \omega)\}\right) < \infty.$$

Es decir, sólo un número finito de elementos de $Q \times \Gamma^*$ *estarán relacionados con cada elemento* (q, x, A) *mediante la función de transición.*

Además, impondremos la condición siguiente[4]*:*

- $\delta(q, x, Z_0) \neq (q', \lambda)$ *para cualesquiera* $q, q' \in Q$ *y* $x \in \Sigma \cup \{\lambda\}$ *(i.e. no se "borra" el 'fondo de la pila').*

Nota 109. *De nuevo, como en el caso de Autómatas Finitos indeterministas, hemos preferido usar una notación funcional menos correcta del tipo*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\}) \longrightarrow Q \times \Gamma^*,$$

para representar correspondencias, que la notación como aplicación más correcta:

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\}) \longrightarrow \mathcal{P}\left(Q \times \Gamma^*\right).$$

La notación elegida pretende, sobre todo, enfatizar la diferencia entre el caso determinístico (δ *es aplicación) frente al indeterminístico (*δ *es correspondencia).*

[3] Recuérdese que si X es un conjunto finito, denotamos por $\sharp(X)$ su cardinal (i.e. el número de sus elementos).

[4] La primera de ellas indica que no podemos "borrar" el símbolo de "fondo de pila", aunque sí podemos leerlo. La pila "vacía es una pila que comienza en el símbolo Z_0.

Nota 110. *Nótese que hemos supuesto que la función de transición δ tiene su rango (el conjunto hacia el que va a parar) en $Q \times \Gamma^*$. Esta condición nos dirá (más adelante) que no podemos escribir en la pila el símbolo de "fondo de pila" nada más que cuando se escriba en la configuración inicial. Podremos, sin embargo, leerlo. No estará, en ningún caso, "en medio" de la pila.*

El determinismo en autómatas con pila difiere del caso de autómatas finitos. No vamos a exigir que δ sea aplicación sino algo más delicado.

Definición 111 (Autómata con Pila Determinista)**.** *Un autómata con pila indeterminista $A := (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ se denomina determinista si verifica las siguientes dos propiedades:*

- *La imagen de cualquier lectura contiene a lo sumo 1 elemento. Es decir, para cualesquiera $(q, x, A) \in Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\})$, el conjunto de los elementos relacionados con él a través de δ tiene, a lo sumo, 1 elemento:*

$$\sharp\left(\{(p,\omega) \in Q \times \Gamma^* \ : \ \delta(q,x,A) = (p,\omega)\}\right) \leq 1.$$

- *Si dados $q \in Q$ y $A \in \Gamma$, existieran $(p, \omega) \in Q \times \Gamma^*$ tales que $\delta(q, \lambda, A) = (q, \omega)$, entonces, ninguno de los elementos de $Q \times \Sigma \times (\Gamma \cup \{Z_0\})$ tiene imagen por δ. Es decir, si*

$$\sharp\left(\{(p,\omega) \in Q \times \Gamma^* \ : \ \delta(q,\lambda,A) = (p,\omega)\}\right) = 1,$$

entonces

$$\sharp\left(\bigcup_{x\in\Sigma}\{(p,\omega) \in Q \times \Gamma^* \ : \ \delta(q,x,A) = (p,\omega)\}\right) = 0.$$

Nota 112. *No es cierto, en el caso de autómatas con pila, que todo autómata con pila indeterminista sea equivalente a un autómata con pila determinista. Así, el siguiente lenguaje es aceptado por un autómata con pila indeterminista, pero no puede ser aceptado por un atómata con pila determinista:*

$$L := \{a^n b^m c^n \ : \ n, m \geq 1\} \bigcup \{a^n b^m c^m \ : \ n, m \geq 1\} \subseteq \{a, b, c\}^*.$$

5.2 Sistema de Transición Asociado a un Autómata con Pila.

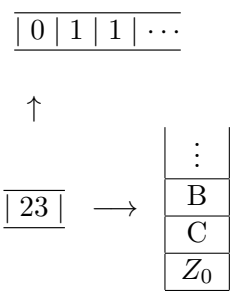
Sea dado un autómata $A := (Q, V, \Sigma, q_0, F, Z_0, \delta)$. El espacio de configuraciones es el producto $\mathcal{S}_A := Q \times \Sigma^* \times Z_0 \cdot V^*$. Dada una palabra $\omega \in \Sigma^*$, la configuración inicial vendrá dada por:

$$I_A(\omega) := (q_0, \omega, Z_0),$$

esto es, escribimos ω en la cinta de trabajo , escribimos el estado inicial en la unidad de control y escribimos Z_0 en la pila como tareas de inicialización.

5.2.1 Modelo gráfico del sistema de transición.

Gráficamente podemos dibujar el autómata con pila mediante los siguientes elementos. El alfabeto de la "cinta" será $\Sigma = \{0, 1\}$. El alfabeto de la pila será $\Gamma = \{A, B, C\}$. El espacio de estados serán los primeros cien números naturales, i.e. $Q = \{0, 1, 2, 3, \ldots, 98, 99, 100\}$. Una representación gráfica de la configuración será:



En este dibujo, el estado es 23 y la unidad de control "lee" el primer símbolo de la cinta de entrada 0 y también "lee" el símbolo B en la pila. Supondremos que, en la pila, "lee" siempre el símbolo que se encuentra más arriba en la pila (el `top`), lo que supone que, en el dibujo, por encima de B no hay nada en la pila.

5.2.2 Transiciones: Formalismo.

Las transiciones son de los tipos siguientes:

- **Transiciones Read/Push.** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 Z) \to_A (q', x', Z_0 Z'),$$

  donde $q \in Q$ y

$$x := x_1 \cdots x_r, \quad Z_0 Z = Z_0 z_1 \cdots z_n.$$

  Realizamos las siguientes operaciones:

  - Read Leemos la información (q, x_1, z_n) (es decir, el estado, el primer símbolo de la palabra y el `top` de la pila). Supondremos $x_1 \neq \lambda$.
  - Transition Aplicamos la función de transición obteniendo $\delta(q, x_1, z_n) = (q', Y)$, con $Y \in \Gamma^*$, $Y \neq \lambda$.
  - Push and Move Entonces,
    - $q' = q$ (cambia el estado de la transición).
    - $x' = x_2 \cdots x_r$ ("borramos" un símbolo de la palabra a analizar (move).
    - $Z' := Z_0 z_1 \cdots z_{n-1} Y$, es decir, $Z_0 Z' := \texttt{push}(\texttt{pop}(Z_0 Z), Y)$.

- **Transiciones Read/Pop:** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 Z) \to_A (q', x', Z_0 Z'),$$

  donde $q \in Q$ y

$$x := x_1 \cdots x_r, \quad Z_0 Z = Z_0 z_1 \cdots z_n.$$

  Realizamos las siguientes operaciones:

- Read Leemos la información (q, x_1, z_n) (es decir, el estado, el primer símbolo de la palabra y el `top` de la pila). Supondremos $x_1 \neq \lambda$.
- Transition Aplicamos la función de transición obteniendo $\delta(q, x_1, z_n) = (q', \lambda)$ (ésta es la caracterización de las acciones Read/Pop). Indica que debemos hacer `pop`.
- Pop and Move Entonces,
  - $q' = q$ (cambia el estado de la transición).
  - $x' = x_2 \cdots x_r$ ("borramos" el primer símbolo de la palabra a analizar (move).
  - $Z_0 Z' := Z_0 z_1 \cdots z_{n-1} := \texttt{pop}(Z_0 Z) = \texttt{push}(\texttt{pop}(Z_0 Z), \lambda)$.

- **Transiciones Lambda/Push.** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 Z) \to_A (q', x', Z_0 Z'),$$

donde $q \in Q$ y

$$x := x_1 \cdots x_r, \quad Z_0 Z = Z_0 z_1 \cdots z_n.$$

Realizamos las siguientes operaciones:

- Read Lambda En este caso, no se lee la cinta a unque sí se lee la pila. Leeremos (q, λ, z_n) (es decir, el estado, la palabra vacía y el `top` de la pila).
- Transition Aplicamos la función de transición obteniendo $\delta(q, \lambda, z_n) = (q', Y)$, con $Y \in \Gamma^*$, $Y \neq \lambda$.
- Push Entonces,
  - $q' = q$ (cambia el estado de la transición).
  - $x' = x$ ("No borramos" un símbolo de la palabra a analizar).
  - $Z_0 Z' := Z_0 z_1 \cdots z_{n-1} Y := \texttt{push}(\texttt{pop}(Z_0 Z), Y)$.

- **Transiciones Lambda/Pop:** Son transiciones entre dos configuraciones:

$$(q, x, Z_0 Z) \to_A (q', x', Z_0 Z'),$$

donde $q \in Q$ y

$$x := x_1 \cdots x_r, \quad Z_0 Z = Z_0 y z_1 \cdots z_n, \text{con } z_n \neq Z_0.$$

Realizamos las siguientes operaciones:

- Read Lambda De nuevo, no se lee la cinta aunque sí se lee la pila. Tendremos (q, λ, z_n) (es decir, el estado, la palabra vacía y el `top` de la pila).
- Transition Aplicamos la función de transición obteniendo $\delta(q, \lambda, z_n) = (q', \lambda)$. Obsérvese que, en este caso, se ha obtenido λ como símbolo de la pila, ésto indica que debemos hacer `pop`.
- Pop and Move Entonces,

* $q' = q$ (cambia el estado de la transición).
* $x' = x$ ("No borramos" un símbolo de la palabra a analizar).
* $Z_0Z' := Z_0z_1 \cdots z_{n-1} := \texttt{pop}(Z_0Z) = \texttt{push}(\texttt{pop}(Z_0Z), \lambda)$.

Nota 113. *Es importante señalar que la diferencia entre "instrucciones"* **/Push** *y* **/Pop** *es "artificial". La mantenemos por razones didácticas. Nótese que*

$$\texttt{pop}(Z_0Z) = \texttt{push}(\texttt{pop}(Z_0Z), \lambda)$$

y

$$\texttt{push}(Z_0Z, Y) = \texttt{push}(\texttt{pop}(Z_0Z), Y),$$

con $\omega \in \Gamma^*$ *no tienen diferencias semánticas significativas porque, obviamente,* $\lambda \in \Gamma^*$. *Las distinguimos para que el lector pueda ver la operación "borrar el último dígito de la pila" como una operación distinguida.*

Nota 114. *Los autómatas finitos del Capítulo anterior se pueden releer como autómatas con pila del modo siguiente: Suponemos que la función de transición* δ *verifica que*

$$\delta(q, x, z) = (q', \lambda), \forall (q, x, z) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^*.$$

En este caso, todas las instrucciones pasan por hacer $\texttt{push}(\texttt{pop}(Z_0), \lambda)$ *que no cambia el contenido de la pila desde la configuración inicial.*

Proposición 115. *Si* A *es un autómata con pila determinista, su sistema de transición* $(S_A, \to_A)$ *es determinista. Es decir, dada una configuración* $c \in S_A$, *existirá a lo sumo una única configuración* $c' \in S_A$ *tal que* $c \to_A c'$.

Demostración. Siguiendo las dos hipótesis de la Definición de Autómata Determinista: Dada una configuración, uno pude hacer una transición **Read/...** o una **Lambda/...**. Si cabe la posibilidad de hacer una transición **Lambda/...**, no habrá ninguna transición que permita hacer lectura (por la segunda de las condiciones impuestas). Si, por el contrario, no hay ninguna transición **Lambda/...**, entonces es forzoso hacer una transición de lectura y ésta es, a lo sumo, única. □

Nota 116. *Esta propiedad nos garantiza que la ejecución de un autómata con pila determinista es posible dado que a cada configuración le sigue o bien una única configuración siguiente o bien una salida (hacia estado final aceptador o de rechazo) por no tener ninguna opción de continuar.*

5.2.3 Codificación del Autómata con Pila.

La introducción que hemos hecho de la noción de Autómata con Pila se basa en varios principios básicos. El primero es que los Autómatas con Pila son "expresables" sobre un alfabeto finito, del mismo modo que lo fueron los autómatas. Una vez visto que expresable sobre un alfabeto finito y visto el sistema de transición podemos admitir (espero) que se trata de un programa que es ejecutable en algún tipo de intérprete que sea capaz de seguir las "instrucciones" asociadas a la transición. Es, obviamente, simple diseñar un programa (en Java, C++, C o cualquier lenguaje) asociado a

un autómata (con la salvedad de los problemas de determinismo/indeterminismo: no tiene sentido "programar" autómatas no deterministas porque su sistema de transición no es determinista).
Sin embargo, podemos utilizar diversas representaciones del autómata. La más simple es la de tabla. Para ello, podemos usar dos entradas. De una parte, el producto cartesiano $E := Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\})$ que será el conjunto de las entradas de la función de transición. De otro lado tenemos un conjunto infinito de las "salidas" $O := Q \times \Gamma^*$, pero nuestra hipótsis nos dice que sólo un número finito de elementos de $Q \times \Gamma^*$ van a entrar en la relación. Así tendremos una tabla como la siguiente:

Ejemplo 14. *El PDA viene dado por:*

- $Q := \{q_0, q_1, r, s\}$,
- $\Sigma := \{0, 1\}$,
- $\Gamma := \{A\}$,
- *Estado Inicial* q_0.
- *Símbolo de fondo de pila* Z_0.
- $F := \{r\}$.

Tabla de transición:

E	**O**
$(q_0, 0, Z_0)$	(q_0, A)
$(q_0, 0, A)$	(q_0, AA)
$(q_0, 1, A)$	(q_1, λ)
$(q_1, 1, A)$	(q_1, λ)
(q_1, λ, Z_0)	(r, λ)
(q_1, λ, A)	(s, λ)
$(q_1, 1, Z_0)$	(s, λ)

Tomemos como entrada la palabra $0^3 1^3 \in \Sigma^*$.
Inicializamos
$I := (q_0, 000111, Z_0)$
Ahora las sucesivas computaciones serán dadas por la secuencia siguiente:

- *Read/Push* $I \to_A c_1 = (q_0, 00111, Z_0 A)$
- *Read/Push* $c_1 \to_A c_2 = (q_0, 0111, Z_0 AA)$
- *Read/Push* $c_2 \to_A c_3 = (q_0, 111, Z_0 AAA)$
- *Read/Pop* $c_3 \to_A c_4 = (q_1, 11, Z_0 AA)$
- *Read/Pop* $c_4 \to_A c_5 = (q_1, 1, Z_0 A)$

- *Read/Pop* $c_5 \to_A c_6 = (q_1, \lambda, Z_0)$
- *Lambda/Pop* $c_6 \to_A c_6 = (r, \lambda, Z_0)$

Si, por el contrario, escojo la palabra $0^3 1^2 \in \Sigma^*$ se produce el efecto siguiente:
Inicializamos
$I := (q_0, 00011, Z_0)$

- *Read/Push* $I \to_A c_1 = (q_0, 0011, Z_0 A)$
- *Read/Push* $c_1 \to_A c_2 = (q_0, 011, Z_0 AA)$
- *Read/Push* $c_2 \to_A c_3 = (q_0, 11, Z_0 AAA)$
- *Read/Pop* $c_3 \to_A c_4 = (q_1, 1, Z_0 AA)$
- *Lambda/Pop* $c_4 \to_A c_5 = (q_1, \lambda, Z_0 A)$
- *Lambda/Pop* $c_5 \to_A c_6 = (s, \lambda, Z_0)$

Finalmente, escojo la palabra $0^2 1^3 \in \Sigma^*$ se produce el efecto siguiente:
Inicializamos
$I := (q_0, 00111, Z_0)$

- *Read/Push* $I \to_A c_1 = (q_0, 0111, Z_0 A)$
- *Read/Push* $c_1 \to_A c_2 = (q_0, 111, Z_0 AA)$
- *Read/Pop* $c_2 \to_A c_3 = (q_1, 11, Z_0 A)$
- *Read/Pop* $c_3 \to_A c_4 = (q_1, 1, Z_0)$
- *Lambda/Pop* $c_4 \to_A c_5 = (s, \lambda, Z_0)$

Obsérvese que:

- Si tengo más ceros que unos, llegaré a leer (q_1, λ, A) con lo que acabaré en el estado s
- Si tengo más unos que ceros, llegaré a leer $(q_0, 1, z_0 y)$ con lo que acabaré en s.
- La única forma de llegar a r sería tener el mismo número de ceros que de unos. Más aún, las únicas palabras que llegan al estado r son las dadas por

$$L := \{0^n 1^n \ : \ n \in \mathbb{N}\}.$$

5.3 Lenguaje Aceptado por un Autómata con Pila.

Hay dos maneras de interpretar el lenguaje aceptado por un autómata con pila: por estado final aceptador y por pila y cinta vacías. Veremos además que ambas nociones de lenguajes son equivalentes, aunque, posiblemente, con diferentes autómatas. La razón de usar ambas nociones se justifica por la orientación de cada una. Así, el lenguaje aceptado por estado final aceptador extiende la noción de Autómata Finito y es extensible a nociones más abstractas de máquinas como los autómatas bi-direccionales o las máquinas de Turing. Por su parte, ver los lenguajes como lenguajes aceptados pr cinta y pila vacías nos simplificarán (en la Sección **??**) probar la relación con los lenguajes libres de contexto y, como veremos más adelantem estarán mejor adaptados al diseño de procesos de Análisis Sintáctico (porque simplifica el formalismo).
Sea $A := (Q, \Sigma, \Gamma, q_0, Z_0, F)$ un autómata con pila y sea $\mathcal{S}_A$ el sistema de transición asociado. Escribiremos $c \vdash_A c'$ cuando la configuración c' es alcanzable desde c en el sistema de transición $\mathcal{S}_A$.

Definición 117 (Lenguaje aceptado mediante estado final aceptador)**.**
Sea $A := (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ un autómata con pila y sea $\mathcal{S}_A$ el sistema de transición asociado. Para cada palabra $\omega \in \Sigma^*$, definimos la configuración inicial en ω a la configuración:

$$I_A(\omega) := (q_0, \omega, Z_0) \in \mathcal{S}_A.$$

Llamaremos lenguaje aceptado (mediante estado final final aceptador) por el autómata A (y lo denotaremos por $L_f(A)$) al conjunto siguiente:

$$L_f(A) := \{\omega \in \Sigma^* \ : \ I_A(\omega) \vdash_A (f, \lambda, z) \in \mathcal{S}_A, \ f \in F\}.$$

Este concepto nos lleva a un modelo de programa en el que la condición de parada viene dada por los estados finales aceptadores y por tener la cinta vacía. Es decir, un "programa" con la estructura siguiente:

INPUT: $\omega \in \Sigma^*$
Initialize: $I := (q_0, \omega, Z_0)$.
while $I \notin F \times \{\lambda\} \times Z_0\Gamma^*$ **do**
    Hallar $c' \in S_A$ tal que $I \to_A c'$
    Com.: Realiza un paso en el sistema de transición.
    $I := c'$
od
OUTPUT: ACEPTAR
end

Definición 118 (Lenguaje aceptado mediante pila y cinta vacías)**.** *Sea $A := (Q, \Sigma, \Gamma, q_0, Z_0, F)$ un autómata con pila y sea $\mathcal{S}_A$ el sistema de transición asociado. Llamaremos*

lenguaje aceptado (mediante pila y cinta vacías) por el autómata A (y lo denotaremos por $L_\emptyset(A)$) al conjunto siguiente:

$$L_\emptyset(A) := \{\omega \in \Sigma^* \ : \ I_A(\omega) \vdash_A (f, \lambda, Z_0) \in \mathcal{S}_A, f \in Q \setminus \{q_0\}^{5}\}.$$

La diferencia entre aceptar por pila vacía o por estado final es que en el caso de aceptar por estado final, la pila puede estar o no vacía mientras que en el caso de aceptación por pila vacía, la pila queda vacía, pero no nos preocupamos de cuál es el estado alcanzado.
En este segundo caso el "programa" tendrá la pinta siguiente:

INPUT: $\omega \in \Sigma^*$
Initialize: $I := (q_0, \omega, z_0 y)$.
while $I \notin Q \times \{\lambda\} \times \{Z_0\}$ **do**
    Hallar $c' \in S_A$ tal que $I \to_A c'$
    Com.: Realiza un paso en el sistema de transición.
    $I := c'$
od
OUTPUT: ACEPTAR
end

Nota 119. *Nótese que las palabras no aceptadas (i.e.* OUTPUT: *RECHAZAR) no se han incluido en una salida del bucle sino que se admite que puedan continuar indefinidamente dentro del bucle. Trataremos de clarificar el significado de este proceso.*
Nótese también que los "programas" anteriores no tienen el sentido usual cuando el autómata es indeterminista. Lo cual es, parcialmente, causa de este formalismo admitiendo bucles infinitos.

La siguiente Proposición muestra la equivalencia entre ambas formas de aceptación, aunque será más cómodo utilizar el caso de pila vacía.

Proposición 120. *Un lenguaje es aceptado por algún autómata con pila mediante pila y cinta vacías si y solamente si es aceptado por algún autómata con pila (posiblemente otro distinto) mediante estado final aceptador. Es decir, Sea A un autómata con pila solbre un alfabeto Σ y sean $L_1 := L_f(A) \subseteq \Sigma^*$ y $L_2 := L_\emptyset(A) \subseteq \Sigma^*$, respectivamente los lenguajes aceptados por A mediante cinta y pila vacías o mediante estado final aceptador. Entonces, se tiene:*

a. Existe un autómata con pila B_1 tal que $L_1 = L_\emptyset(B_1)$,

b. Existe un autómata con pila B_2 tal que $L_2 = L_f(B_2)$.

[5] Si admitimos q_0 entonces, todo lenguaje aceptado por pila vacía debería contener la palabra vacía.

Demostración. Mostraremos un mecanismo de paso, construyendo para cada lenguaje $L_f(A)$ aceptado por un autómata con pila A mediante estado final aceptador un autómata con pila B_1 que acepta el mismo lenguaje mediante pila y cintas vacías y lo mismo para la segunda de las afiramciones Esto es, $L_f(A) = L_\emptyset(B_1)$.

- Dado un autómata con pila $A := (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ que acepta el lenguaje $L_f(A)$ mediante estado final aceptador, construyamos el nuevo autómata que aceptará el mismo lenguaje mediante pila vacía $B_1 := (\bar{Q}, \Sigma, \bar{\Gamma}, \bar{q_0}, \bar{Z_0}, \bar{F}, \bar{\delta})$ del modo siguiente:
  - Sea $p_0, p_f \notin Q$ dos nuevos estados y definamos $\bar{Q} := Q \cup \{p_0, p_f\}$.
  - $\bar{q_0} := p_0, \bar{Z_0} := Z_0$.
  - La idea clave consiste en introducir un nuevo símbolo en el alfabeto de la pila X_0 que "protegerá" el símbolo de fondo de la pila. Así, elegiremos $X_0 \notin \Gamma$ y $\bar{\Gamma} := \Gamma \cup \{X_0\}$.
  - $\bar{F} := F$, dejamos el mismo conjunto de estados finales aceptadores[6].
  - Definamos

$$\bar{\delta} : \bar{Q} \times \left(\bar{\Sigma} \cup \{\lambda\}\right) \times \left(\bar{\Gamma} \cup \{z_0 y\}\right) \to \bar{Q} \times \bar{\Gamma}^*,$$

    mediante:
    - $\bar{\delta}(p_0, w, Z_0) = (q_0, Z_0 X_0)$. Es decir, inicializamos "protegiendo" Z_0 con una variable X_0. La transformación será:

$$I_{B_1}(\omega) \to_A (q_0, \omega, Z_0 X_0).$$

    - Mientras "vivamos" en el "viejo" autómata no cambiamos la función de transición, es decir:

$$\bar{\delta} \mid_{Q \times (\Sigma \cup \{\lambda\}) \times \Gamma} = \delta,$$

      Aquí nos garantizamos que la variable protectora X_0 no será nunca añadida después de haberla usado por vez primera.
    - Para una transición $\bar{\delta}(q, w, X_0)$ hacemos lo que hubiera hecho el viejo autómata si estuviera leyendo la pila vacía. Así, si $\delta(q, w, Z_0) = (p, z)$ y $z \neq \lambda$ haremos:

$$\bar{\delta}(q, w, X_0) := \delta(q, w, Z_0) = (p, z).$$

      Varios elementos importantes a describir aquí:
      - Por nuestra definición de la funci'on `pop`, Z_0 no puede borrarse, por eso, si la segunda coordenada de $\delta(q, w, Z_0) = (q, \lambda)$, entonces definiremos:

$$\bar{\delta}(q, w, X_0) := (p, X_0).$$

[6] Aunque bien podríamos haber añadido p_f a F y tendríamos el todo.

· La idea de esta transformación es que, durante los cálculos del "viejo autómata", pudiera ser que, en una etapa intermedia, se vaciase la pila sin haber acabado con la palabra. En ese caso, seguiríamos apilando información que podría ser útil en el resto de la computación.

* Si $q \in F$, definimos $\bar{\delta}(q, \lambda, z) := (p_f, \lambda)$, para $z \in \Gamma$, $z \neq Z_0, X_0$.
* Si $q \in F$, definimos $\bar{\delta}(q, \lambda, X_0) := (p_f, X_0)$.
* Finalmente, definimos para cada $z \neq Z_0$, $\bar{\delta}(p_f, \lambda, z) = (p_f, \lambda)$.

Para alcanzar el estado p_f, debemos alcanzar un estado final aceptador de F. Además, las configuraciones de B_1 tienen la forma siguiente para $q \neq p_0, p_f$:

$$(q, x, Z_0 X_0 Z), \text{ con } x \in \Sigma^*, Z \in \Gamma^*,$$

y están identificadas con las configuraciones de A dadas mediante:

$$(q, x, Z_0 Z), \text{ con } x \in \Sigma^*, Z \in \Gamma^*,$$

Las demás configuraciones son o bien la configuración inicial (q_0, x, Z_0) o configuraciones cuyo estado es p_f. Para una palabra $x \in \Sigma^*$, tendremos:

$$I_A(x) \vdash_A (q, x, Z_0 Z).$$

Si $q \in F$, el autómata B_1 habría calculado también:

$$I_{B_1}(x) \vdash_{\bar{A}} (q, \lambda, Z_0 X_0 Z).$$

Y, en la siguiente fase, procedería a vaciar la pila, usando $\bar{\delta}(q, \lambda, X_0) := (p_f, X_0)$ y $\bar{\delta}(p_f, \lambda, z) = (p_f, \lambda)$, para todo $z \neq Z_0$. Esto nos da $L_f(A) \subseteq L_\emptyset(B_1)$.

De otro lado, dada $x \in \Sigma^*$ si $x \in L_\emptyset(B_1)$, entonces, habremos realizado una computación que produce el efecto siguiente:

$$I_{B_1}(x) \vdash_{\bar{A}} (p_f, \lambda, Z_0).$$

Ahora bien, p_f sólo se alcanza tras una configuración final aceptadora de A, por lo que deberíamos haber calculado:

$$I_{B_1}(x) \vdash_{B_1} (q, x', Z_0 X_0 Z) \vdash_{B_1} (p_f, \lambda, Z_0),$$

con $q \in F$ en algún momento intermedio. Entonces, la acción del autómata B_1 nos permite garantizar que se tiene:

$$I_{B_1}(x) \vdash_{B_1} (q, x', Z_0 X_0 Z) \vdash_{B_1} (q, x', Z_0 X_0) \vdash_{B_1} (p_f, \lambda, Z_0),$$

Si $x' \neq \lambda$, las transiciones asociadas al borrado de la pila ($\bar{\delta}(q, \lambda, \alpha) := (p_f, \lambda)$ $\bar{\delta}(q, \lambda, X_0) := (p_f, X_0)$ y $\bar{\delta}(p_f, \lambda, z) = (p_f, Z_0)$) no nos permiten borrar contenido en la cinta. Por tanto, la única configuración final alcanzable sería:

$$I_{B_1}(x) \vdash_{B_1} (q, x', Z_0 X_0 Z) \vdash_{B_1} (q, x', Z_0 X_0) \vdash_{B_1} (p_f, x', Z_0).$$

Por tanto, sólo cabe la posibilidad de que $x' = \lambda$ con lo cual habremos hecho la computación mediante:

$$I_{B_1}(x) \vdash_{B_1} (q, \lambda, Z_0 X_0 Z) \vdash_{B_1} (p_f, \lambda, Z_0).$$

Y el autómata A habría seguido la computación:

$$I_A(x) \vdash_A (q, \lambda, Z_0 Z),$$

con lo que $L_\emptyset(B_1) \subseteq L(A)$.

- Recíprocamente, dado un autómata con pila $A := (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ que acepta el lenguaje $L_\emptyset(A)$ mediante pila y cinta vacías, construyamos el nuevo autómata que aceptará el mismo lenguaje mediante estados finales aceptadores $B_2 := (\bar{Q}, \bar{\Sigma}, \bar{\Gamma}, \bar{q_0}, \bar{Z_0}, \bar{F}, \bar{\delta})$ del modo siguiente. Introduciremos un estado final aceptador nuevo p_f y definimos $\bar{F} := \{p_f\}, \bar{Q} := Q \cup \{p_f\}$. Introducimos un nuevo símbolo inicial para la pila $\bar{Z_0} := X_0$ y definimos $\bar{\Gamma} := \Gamma \cup \{Z_0\}$. Ahora introducimos una nueva función de transición $\bar{\delta}$ definida del modo siguiente:

$$\bar{\delta} \mid_{Q \times (\Sigma \cup \{\lambda\}) \times \bar{\Gamma}} = \delta.$$

$$\bar{\delta}(q, \lambda, Z_0) := (p_f, \lambda).$$

$$\bar{\delta}(\bar{q_0}, \lambda, X_0) := (q_0, Z_0).$$

Es clara la identificación entre las configuraciones de A y las configuraciones de B_2 que poseen un estado de A:

$$(q, x, \alpha) \leftrightarrow (q, x, X_0 \alpha).$$

Ahora cosideramos $L_f(B_2)$ el lenguaje aceptado mediante estado final aceptador por B_2. Una palabra $x \in \Sigma^*$ es aceptada si se ha producido una computación cuyos extremos son:

$$I_{B_2}(x) \vdash_{B_2} (p_f, \lambda, Z_0 Z).$$

Ahora observamos que el estado p_f sólo se alcanza mediante $\lambda-$transiciones que leen el símbolo Z_0 en la pila (i.e. $\bar{\delta}(q, \lambda, Z_0) := (p_f, \lambda)$). Pero el símbolo Z_0 sólo se lee cuando la pila "original" está vacía (i.e. $Z_0 Z = Z_0$). Así, nuestra computación debe tener la forma:

$$I_{B_2}(x) \vdash_{B_2} (q, x', X_0 Z_0) \vdash_{B_2} (p_f, \lambda, X_0).$$

De otro lado, estas $\lambda-$transiciones no borran información en la cinta. Por tanto, con los mismos argumentos que en el apartado anterior, necesariamente ha de darse $x' = \lambda$ y existirán:

$$I_{B_2}(x) \vdash_{B_2} (q, \lambda, X_0 Z_0) \vdash_{B_2} (p_f, \lambda, X_0 Z_0),$$

que se corresponde a la computación en A

$$I_A(x) \vdash_A (q, \lambda, Z_0).$$

Con ello concluimos que $x \in L_\emptyset(A)$, es aceptado por A mediante pila y cinta vacías, y $L_f(B_2) \subseteq L_\emptyset(A)$.
De otro lado, supongamos que x es aceptado por A mediante pila y cinta vacías. En ese caso, tendremos una computación en A de la forma:

$$I_A(x) \vdash_A (q, \lambda, Z_0).$$

Esto se transforma en una computación en B_2 de la forma:

$$I_{B_2}(x) \vdash_{B_2} (q, \lambda, X_0 Z_0).$$

Aplicando la transición $\delta(q, \lambda, Z_0) := (p_f, \lambda)$ obtendremos:

$$I_{B_2}(x) \vdash_{B_2} (q, \lambda, X_0 Z_0) \vdash_{B_2} (p_f, \lambda, X_0)$$

y habremos probado que $L_\emptyset(A) \subseteq L_f(B_2)$ como pretendíamos.

□

5.4 Equivalencia con Gramáticas Libres de Contexto.

Teorema 121. *Los lenguajes libres de contexto son exactamente los lenguajes aceptados por los autómatas con pila mediante cinta y pila vacías. Es decir, se verifican las siguiente dos propiedades:*

a. *Para cada gramática libre de contexto G sobre un alfabeto Σ de símbolos terminales, existe un autómata con pila A tal que $L(G) = L_\emptyset(A)$.*

b. *Para cada autómata A con alfabeto de cinta Σ existe una gramática libre de contexto G tal que el lenguaje generado por G coincide con $L_\emptyset(A)$.*

Más aún, daremos procedimientos de construcción en ambos sentidos.

Demostración. Dividiremos la prueba en las dos afirmaciones.

a. Bastará con lo probemos para gramáticas en forma normal de Chomsky. El resto se obtiene en las progresivas transformaciones de gramáticas. Así, supongamos que G es dada mediante $G := (V, \Sigma, q_0, P)$, donde q_0 es el símbolo inicial. Defniremos un autómata con pila $A := (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ de la forma siguiente:

   - $Q := \{q_0\}$ posee un único estado (que es también el estado inicial).
   - El símbolo de fondo de la pila es un símbolo auxiliar.
   - El alfabeto de la pila reúne a todos los símbolos (terminales o no) de la gramática $\Gamma := V \cup \Sigma$.
   - La función de transición δ estará dada del modo siguiente:
     - $\delta(q_0, \lambda, Z_0) := (q_0, Q_0)$ (al comenzar pongamos Q_0 justo encima del fondo de la pila).

- Si la gramática tiene una producción del tipo $A \mapsto a \in \Sigma \cup \{\lambda\}$, escribamos[7]:
$$\delta(q_0, \lambda, A) := (q_0, a).$$
- Si la gramática tiene una producción del tipo $A \mapsto CD$, con $C, D \in V$, pongamos:
$$\delta(q_0, \lambda, A) := (q_0, DC).$$
- Finalmente, para cada $a \in \Sigma$, pongamos:
$$\delta(q_0, a, a) := (q_0, \lambda).$$

Para ver la demostración de la igualdad bastará con observar que la pila ejecuta un árbol de derivación de la gramática. Por tanto, basta con seguir (borrando) las hojas para ir borrando en la cinta. El vaciado de la cinta y de la pila se produce conforme vamos verificando las hojas.

b. Para la segunda de las afirmaciones, consideremos dado un autómata con pila $A := (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ que acepta un lenguaje $L_\emptyset(A)$. Construyamos la gramática $G := (V, \Sigma, Q_0, P)$ mediante las definiciones siguientes:

- $V := Q \times (\Gamma \cup \{Z_0\}) \times Q \cup \{Q_0\}$. Utilizaremos la notación $\langle qAp \rangle$ para representar el símbolo no terminal $(q, A, p) \in V$[8].
- El símbolo inicial Q_0 lleva acompañada unas producciones del tipo siguiente:
$$Q_0 \mapsto \langle q_0 Z_0 p \rangle,$$
para cada $p \in Q$.
- Si la función de transición δ satisface $\delta(p, a, A) = (q, \lambda)$ con $a \in \Sigma \cup \{\lambda\}$ y $A \in \Gamma \cup \{Z_0\}$, escribiremos la producción:
$$\langle pAq \rangle \mapsto a.$$
- Si la función de transición δ satisface $\delta(p, a, A) = (q, B_1 \cdots B_n)$ con $a \in \Sigma \cup \{\lambda\}$ y $B_1, \ldots, B_n \in \Gamma \cup \{Z_0\}$, escribiremos las producciones siguientes:
$$\langle pAq \rangle \mapsto \langle pB_n s_1 \rangle \langle s_1 B_{n-1} s_2 \rangle \langle s_2 B_{n-2} s_3 \rangle \cdots \langle s_{n-1} B_1 q \rangle a,$$
para todos los estados $(s_1, \ldots, s_{n-1}) \in Q^{n-1}$.

□

Nota 122. *Nótese que la construcción de la gramática asociada a un autómata con pila introduce un número exponencial (en el número de estados) de producciones por lo que es poco aconsejable utilizar esa construcción. Nos conformaremos con saber de su existencia.*
Pero, obsérvese también, hemos probado que se puede suponer que los autómatas con pila indeterministas posee un sólo estado y que, en el contexto de un sólo estado, el paso de autómatas a gramáticas se puede realizar en tiempo polinomial.

[7] Nótese que para las producciones $A \mapsto \lambda$ borramos A.

[8] Bien podríamos haber usado el convenio $Q_0 := \langle q_0 \rangle$, pero lo dejamos por comodidad como si fuera una nueva variable.

5.5 Algunas Propiedades de la clase de lenguajes libres de contexto

Definición 123. *Llamamos lenguajes libres de contexto a los lenguajes generados por una gramática incontextual (o, equivalentemente, los reconocidos por un autómata con pila indeterminista mediante pila y cinta vacías).*

En esta Sección nos ocuparemos de enunciar unas pocas propiedades de la clase de lenguajes libres de contexto.

Teorema 124 (Intersección con Lenguajes Regulares)**.** *La clase de lenguajes libres de contexto está cerrada mediante intersección con lenguajes regulares. Es decir, si $L \subseteq \Sigma^*$ es un lenguaje libre de contexto y si $M \subseteq \Sigma^*$ es un lenguaje regular, entonces, $L \cap M$ es un lenguaje libre de contexto.*

Nota 125. *Veremos que la intersección de dos lenguajes libres de contexto puede no ser un lenguaje libre de contexto. Para ello, consideremos los dos lenguajes siguientes:*

$$L := a^* \cdot \{b^n c^n \ : \ n \in \mathbb{N}\} \subseteq \{a, b, c\}^*.$$

$$M := \{a^n b^n \ : \ n \in \mathbb{N}\} \cdot c^* \subseteq \{a, b, c\}^*.$$

La intersección es el lenguaje:

$$L \cap M := \{a^n b^n c^n \ : \ n \in \mathbb{N}\} \subseteq \{a, b, c\}^*.$$

Veremos más adelante que $L \cap M$ no es un lenguaje libre de contexto.

Definición 126 (Morfismo de monoides)**.** *Dados dos monoides $(M, *)$ y $(N, \perp)$ llamaremos morfismo de monoides a toda aplicación $f : M \longrightarrow N$ que verifica las propiedades siguientes:*

a. *$f(\lambda_M) = \lambda_N$, donde λ_M y λ_N son los respectivos elementos neutros de los monoides M y N.*

b. *$f(x * y) = f(x) \perp f(y)$ para todo $x, y \in M$.*

Teorema 127 (Imágenes inversas por morfismos)**.** *La clase de lenguajes libres de contexto es cerrada por imágenes inversas por morfismos de monoides. Esto es, dados dos alfabetos Σ_1 y Σ_2 y dado un morfismo de monoides $f : \Sigma_1^* \longrightarrow \Sigma_2^*$, para cada lenguaje libre de contexto $L \subseteq \Sigma_2^*$, el siguiente también es un lenguaje libre de contexto:*

$$f^{-1}(L) := \{x \in \Sigma_1^* \ : \ f(x) \in L\}.$$

Teorema 128 (Complementario y Determinismo)**.** *La clase de los lenguajes aceptados por un autómata con pila determinista es cerrada por complementación. Es decir, si $L \subseteq \Sigma^*$ es un lenguaje aceptado por un autómata con pila determinista, su complementario $L^c := \Sigma^* \setminus L$ es también un lenguaje libre de contexto.*

Nota 129. *Como ya habíamos señalado en la Observación 112 los lenguajes aceptados por autómatas determinísticos definen una clase particular* DCGL *dentro de la clase de lenguajes libres de contexto: lenguajes libres de contexto "deterministas". Son los lenguajes generados por una gramática libre de contexto determinística y ejemplos tan simples como el palíndromo no admiten autómtas con pila determinísticos que los decidan. No entremos en esa discusión hasta más adelante.*

El siguiente es un importante resultado de caracterización de lenguajes libres de contexto debido a W.F. Ogden[9].

Definición 130 (Marcador de una palabra)**.** *Llamamos marcador de una palabra* $x \in \Sigma^*$ *a una lista* $\varepsilon := (\varepsilon_1, \ldots, \varepsilon_n) \in \{0,1\}^n$*, donde* $n = |x|$*.*

Nótese que un marcador consiste en señalar ciertos símbolos de una palabra y no otros. Obviamente, el número de marcadores de una palabra x es igual a $2^{|x|}$. Una manera de interpretar un marcador es el de subrayar algunos símbolos de la palabra y no otros, conforme a la regla obvia: subraya el símbolos $i-$ésimo si $\varepsilon_i = 1$ y no lo subrayes en el caso contrario. A modo de ejemplo, tomemos la palabra $x = abbbbabaa$ de longitud 9 y elijamos dos marcados $\varepsilon = (0,1,1,0,0,0,1,0,1)$ y $\varepsilon' = (1,0,0,0,1,0,0,0,1)$. Estos dos marcadores señalan símbolos de la palabra conforma a las siguientes reglas:

$$marcado(x, \varepsilon) := a\underline{b}\underline{b}bba\underline{b}a\underline{a}.$$

$$marcado(x, \varepsilon') := \underline{a}bbb\underline{b}aba\underline{a}.$$

Llamamos número de posiciones distinguidas de un marcador al número de 1's. Así, el número de posiciones disntinguidas de ε es 4 y el de ε' es 3.

Teorema 131 (Lema de Ogden)**.** *Sea* $G := (V, \Sigma, Q_0, P)$ *una gramática libre de contexto. Existe un número natural* $N \geq 1$ *tal que para toda palabra* $z \in L(G)$ *y para todo marcador de* z *con un número de posiciones distinguidas mayor o igual a* N*, existe una factorización:*

$$z = uvwxy,$$

verificando:

a. *La subpalabra* w *contiene, al menos, una posición distinguida.*

b. *Las subpalabras* v *y* x *contienen, al menos, una posición disntinguida entre las dos.*

c. *La subpalabra* vwx *tiene, a lo sumo,* N *posiciones distinguidas.*

d. *Existe un símbolo no terminal* $A \in V$ *tal que se verifica* $Q_0 \vdash_G uAy, A \vdash_G vAx$ $A \vdash_G w$*. En particular, tenemos una propiedad de bombeo, puesto que para todo* $i \geq 0$*,* $uv^iwx^*iy \in L(G)$*.*

[9] W.F. Ogden. "A Helpful Result for Proving Inherent Ambiguity". *Mathematical Systems Theory* **2** (1968) 191–194.

Aunque el enunciado tiene un aspecto complejo, es un instrumento más que útil para mostrar lemguajes que no son libres de contexto. Una de las conclusiones de este Lema de Ogden es el siguiente resultado conocido como Lema de Bar–Hillel[10] aunque es debido a Bar–Hillel, Perles y Shamir.

Corolario 132 (Lema de Bar–Hillel)**.** *Si L es un lenguaje que satisface la siguiente propiedad:*

Para cada número natural N, $N \geq 1$, existe una palabra $z \in L$ en el lenguaje de longitud mayor que N verificando la siguiente propiedad:
Para cualesquiera palabras $\forall u, v, w, x, y \in \Sigma^*$, verificando

$$[z = uvwxy, |vwx| \leq N, |w| \geq 1, |vx| \geq 1] \Rightarrow \exists i \geq 0,\ uv^i wx^i y \notin L.$$

Entonces L no es un lenguaje libre de contexto.

Demostración. La prueba es obvia a partir del Lema de Ogden. Nótese que las condiciones de longitud pueden reescribirse en términos de marcadores. □

Ejemplo 15. *El lenguaje $\{a^n b^n c^n \ : \ n \in \mathbb{N}\}$ no es un lenguaje libre de contexto. Nótese que para cada $N \in \mathbb{N}$ la palabra $a^N b^N c^N$ verifica que para toda factorización*

$$a^N b^N c^N = uvwxy,$$

con las propiedades prescritas: $|vwx| \leq N, |w| \geq 1, |vx| \geq 1$ significa que en vwx no pueden estar más de dos símbolos (o bien $\{a, b\}$ o bien $\{b, c\}$). Por tanto, bombeando $uv^i wx^i y$ en algún momento desequilibramos el número de símbolos. Así, por ejemplo, si vwx sólo contiene símbolos en $\{a, b\}$, bombeando $uv^i wx^i y$ aumentamos el número de a's y de b's, pero no aumentamos el número de c's, con lo que, en algún momento, $uv^i wx^i y \notin L$.

5.6 Problemas

5.6.1 Problemas

Problema 60. *Sea A un Autómata con pila con lenguage por pila y lista vacías $L = L_\emptyset(A)$, y suponed que la palabra vacía no está en el lenguage. Describid cómo modificar el autómata para que accepte también la palabra vacía mediante pila y lista vacías.*

Problema 61. *Hallar Autómatas con Pila asociados a todas y cada una de las gramáticas libres de contexto descritas en los Problemas del Capítulo anterior.*

Problema 62. *Hallar un autómata con pila para el lenguaje*

$$L := \{0^n 1^n \ : \ n \in \mathbb{N}\}.$$

[10] Y. Bar–Hillel, M. Perles, E. Shamir. "Onformal properties of simple phase–structure grammars". *Zeitschrift für Phonetik, Sparchwissenschaft und Kommunikationsforschung* **14**(1961) 143–172.

Problema 63. *Hallar un autómata con pila para el palíndromo.*

Problema 64. *Construir autómatas con pila que acepten los complementarios (en $\{a,b\}^*$) de los siguiente lenguajes:*

a. $\{a^n b^n c^n \ : \ n \in \mathbb{N}\}$.

b. $\{\omega\omega^R \ : \ \omega \in \{a,b\}\}$.

c. $\{a^m b^n a^m b^n \ : \ m,n \geq 1\}$.

Problema 65. [11] *Probar que el conjunto de palabras que pueden aparecer en una pila de un autómata con pila es un lenguaje regular.*

Problema 66. *Describir una gramática sensible al contexto para el lenguaje (que no es libre de contexto) siguiente:*

$$L := \{a^n b^n c^n \ : \ n \in \mathbb{N}\} \subseteq \{a,b,c\}^*.$$

Problema 67. *Probar que si un autómata con pila verifica que existe una constante $k \in \mathbb{N}$ tal que la pila nunca contiene palabras de longitud mayor que k, entonces, el lenguaje aceptado por ese autómata es un lenguaje regular.*

Problema 68. *Diseñar un automata con pila para cada uno de los siguientes lenguajes*

a.

$$\{a^i b^j c^k | i \neq j or j \neq k\}$$

b. El conjunto de todos las cadenas de símbolos que no son de la forma ww, esto es que no son igual a ningún "string" repetido.

Problema 69. *Sea el siguiente PDA definido por*

$$P = (\{q_0,q_1,q_2,q_3,f\},\{a,b\},\{Z_0,A,B\},\delta,q_0,Z_0,\{f\})$$

Donde δ está dada por las siguientes reglas:

$\delta(q_0,a,Z_0) = (q_1,AAZ_0)$	$\delta(q_0,b,Z_0) = (q_2,BZ_0)$	$\delta(q_0,\lambda,Z_0) = (f,\lambda)$
$\delta(q_1,a,A) = (q_1,AAA)$	$\delta(q_1,b,A) = (q_1,\lambda)$	$\delta(q_1,\lambda,Z_0) = (q_0,Z_0)$
$\delta(q_2,a,B) = (q_3,\lambda)$	$\delta(q_2,b,B) = (q_2,BB)$	$\delta(q_2,\lambda,Z_0) = (q_0,Z_0)$
$\delta(q_3,\lambda,B) = (q_3,\lambda)$	$\delta(q_3,\lambda,Z_0) = (q_1,AZ_1)$	

a. Demostrar que la cadena de caracteres bab está en el lenguaje.

b. Demotrar que la cadena abb está en el lenguaje.

c. Dar los contenidos de la pila después de que el autómata ha leído b^7a^4.

d. Describir el lenguaje generado.

[11] Problema Difícil.

Problema 70. *Un autómata con pila se llama restringido si en cada transición la pila solamente aumenta en, a lo sumo, un síimbolo. Demostrar que todo lenguaje dado por un autómata se puede definir mediante autómatas restringidos.*

Problema 71. *Convertir la gramática*

$$Q_0 \mapsto 0Q_01 \mid A, A \mapsto 0A1 \mid Q_0 \mid \lambda,$$

a un autómata con pila que acepte el lenguaje por pila vacía o por estados finales aceptadores.

Problema 72. *Convertir la gramática con producciones*

$$Q_0 \mapsto aAA, A \mapsto aQ_0 \mid bQ_0 \mid a,$$

a un autómata con pila que acepte el lenguaje por pila vacía o por estado finales.

Problema 73. *Diseñar una gramática y un autómata con pila que acepte el siguiente lenguaje:*

$$\{0^n1^m \mid n \leq m \leq 2m.\}$$

Problema 74. *Consideremos la siguiente gramática libre de contexto. :*

$$\begin{aligned}
< q_0 > &\mapsto while \ < EXPRESION > \ \{< PROCEDIMIENTO >\} \\
< q_0 > &\mapsto if \ < EXPRESION > \ then \ \{< PROCEDIMIENTO >\} \\
< PROCEDIMIENTO > &\mapsto < q_0 > \\
< PROCEDIMIENTO > &\mapsto t++; cout << endl << t << endl; \\
< EXPRESION > &\mapsto < NUMERO > \ less \ than \ < VARIABLE > \\
< EXPRESION > &\mapsto < VARIABLE > \ less \ than \ < NUMERO > \\
< EXPRESION > &\mapsto < NUMERO > \ less \ than \ < FORMULA > \\
< FORMULA > &\mapsto < FORMULA > + < VARIABLE > \\
< FORMULA > &\mapsto (< FORMULA >) \\
< NUMERO > &\mapsto 1 < NUMERO > \\
< NUMERO > &\mapsto 0 \\
< VARIABLE > &\mapsto t \\
< VARIABLE > &\mapsto < FORMULA > +t \\
< VARIABLE > &\mapsto < FORMULA > +1
\end{aligned}$$

Denotaremos a las variables utilizando $< x >$, donde la variable de inicio sera denotada con $< q_0 >$. El alfabeto esta formado por los siguientes símbolos:

$$\Sigma := \{while, if, \{, \}, then, t, +, ;, cout, endl, <<, less \ than, 1, 0, (,)\}.$$

Se pide lo siguiente:

a. *Eliminar las producciones unarias de la gramática.*

b. Eliminar los símbolos inútiles de la gramática.

c. Construir la tabla de analísis sintáctico de la gramática.

d. Obtener la siguiente palabra mediante derivaciones a la izquierda:

$while\ t\ less\ than\ 110\ \{if\ t\ less\ than\ 10\ then\ \{t{+}{+}; cout << endl << t << endl;\}\}.$

Problema 75. *Una gramática se dice ambigua si se puede derivar la misma palabra mediante dos derivaciones diferentes a la izquierda.*

Demostrar que la siguiente gramática es ambigua:

$$
\begin{aligned}
< q_0 > &\mapsto if\ < EXPRESION >\ then\ < PROCEDIMIENTO > \\
< Q_0 > &\mapsto if\ < EXPRESION >\ then\ < PROCEDIMIENTO > \\
&\quad else < PROCEDIMIENTO > \\
< PROCEDIMIENTO > &\mapsto if\ < EXPRESION >\ then\ < PROCEDIMIENTO > \\
< PROCEDIMIENTO > &\mapsto if\ < EXPRESION >\ then\ < PROCEDIMIENTO > \\
&\quad else < PROCEDIMIENTO > \\
< PROCEDIMIENTO > &\mapsto System.out.println(1); \\
< PROCEDIMIENTO > &\mapsto System.out.println(0); \\
< EXPRESION > &\mapsto < NUMERO >\ less\ than\ < VARIABLE > \\
< EXPRESION > &\mapsto < VARIABLE >\ less\ than\ < NUMERO > \\
< NUMERO > &\mapsto 1 < NUMERO > \\
< NUMERO > &\mapsto 0 \\
< VARIABLE > &\mapsto t
\end{aligned}
$$

Demostrar que esta gramática genera el mismo lenguaje:

$$
\begin{array}{rcl}
< Q_0 > & \mapsto & if\ < EXPRESION >\ then\ < PROCEDIMIENTO > \\
< Q_0 > & \mapsto & if\ < EXPRESION >\ then\ < PROCED-ELSE > \\
 & & else < PROCEDIMIENTO > \\
< PROCEDIMIENTO > & \mapsto & if\ < EXPRESION >\ then\ < PROCEDIMIENTO > \\
< PROCED-ELSE > & \mapsto & if\ < EXPRESION >\ then\ < PROCED-ELSE > \\
 & & else < PROCED-ELSE > \\
< PROCEDIMIENTO > & \mapsto & System.out.println(1); \\
< PROCEDIMIENTO > & \mapsto & System.out.println(0); \\
< PROCED-ELSE > & \mapsto & System.out.println(1); \\
< PROCED-ELSE > & \mapsto & System.out.println(0); \\
< EXPRESION > & \mapsto & < NUMERO >\ less\ than\ < VARIABLE > \\
< EXPRESION > & \mapsto & < VARIABLE >\ less\ than\ < NUMERO > \\
< NUMERO > & \mapsto & 1 < NUMERO > \\
< NUMERO > & \mapsto & 0 \\
< VARIABLE > & \mapsto & t
\end{array}
$$

y ademas demostrar que no es ambigua.

Problema 76. *Construir una derivacion a la izquierda de la siguiente palabra* $10 + 11 * 1111$ *utilizando la siguiente gramática:*

$$
\begin{array}{rcl}
< Q_0 > & \mapsto & < EXPRESION > * < EXPRESION > \\
< Q_0 > & \mapsto & < EXPRESION > + < EXPRESION > \\
< EXPRESION > & \mapsto & < NUMERO > + < EXPRESION > \\
< EXPRESION > & \mapsto & < NUMERO > - < EXPRESION > \\
< NUMERO > & \mapsto & 1 < NUMERO > \\
< NUMERO > & \mapsto & 0 < NUMERO > \\
< NUMERO > & \mapsto & 0|1.
\end{array}
$$

Decidir si la gramática es ambigua y que pasa si se elimina la segunda producción.

Problema 77. *Supongamos que el siguiente lenguaje de programación dado por la*

siguiente gramática:

$$\begin{aligned}
< Q_0 > &\mapsto < DECLARACION >< Q_0 >< ORDEN > \\
< DECLARACION > &\mapsto < TIPO >< NOMBRE >=< VALOR >; \\
< TIPO > &\mapsto REAL|INTEGER \\
< NOMBRE > &\mapsto a|b \\
< ORDEN > &\mapsto < NOMBRE >=< ORDEN > + < ORDEN >; \\
< ORDEN > &\mapsto < VALOR > \\
< ORDEN > &\mapsto < NOMBRE > \\
< VALOR > &\mapsto 1 < VALOR > \\
< VALOR > &\mapsto 0
\end{aligned}$$

Hallar una derivación a la izquierda para esta palabra (los espacios y retornos de carro se eliminaran):

$$REAL\ a = 10; REAL\ b = 110; a = a + b; b = a + a + a;$$

Demostrar que el lenguaje asi definido admite como correcta la siguiente expresión:

$$REAL\ a = 10; a = a = a + a; +110s;$$

Comprobar que se acepta esta expresion y modificar la gramática para que solo se acepten expresiones correctas de sumas.

Chapter 6

Una Sucinta Introducción a Parsing

Contents

6.1 Introducción 116
6.1.1 El problema de parsing: Enunciado 119
6.2 Compiladores, Traductores, Intérpretes 119
6.2.1 Traductores, Compiladores, Intérpretes 120
6.2.1.1 Compiladores Interpretados. 121
6.2.2 Las etapas esenciales de la compilación. 121
6.2.2.1 La Compilación y su entorno de la programación. 121
6.2.2.2 Etapas del Proceso de Compilación. 121
6.2.2.3 En lo que concierne a este Capítulo. 122
6.3 Conceptos de Análisis Sintáctico 122
6.3.1 El problema de la Ambigüedad en CFG 122
6.3.2 Estrategias para el Análisis Sintáctico. 124
6.4 Análisis CYK 126
6.4.1 La Tabla CYK y el Problema de Palabra. 126
6.4.2 El Árbol de Derivación con las tablas CYK. 128
6.4.3 El Algoritmo de Análisis Sintáctico CYK 129
6.5 Traductores Push–Down. 130
6.5.0.1 Sistema de Transición asociado a un PDT. 131
6.6 Gramáticas $LL(k)$: Análisis Sintáctico 132
6.6.1 FIRST & FOLLOW 132
6.6.2 Gramáticas $LL(k)$ 136
6.6.3 Tabla de Análisis Sintáctico para Gramáticas $LL(1)$ 138
6.6.4 Parsing Gramáticas $LL(1)$ 139
6.7 Cuestiones y Problemas 142
6.7.1 Cuestiones 142
6.7.2 Problemas 144

En el proceso de compilación, debemos considerar diversos pasos.
En un primer paso (Análisis Léxico) el compilador decide si todas las partes del fichero (trozos en los que está dividido) responden a los patrones de tipos de datos, identificadores o palabras reservadas (en forma de expresiones regulares) que hemos seleccionado. Para ello, aplica los diversos autómatas finitos asociados a las diversas expresiones regulares distinguidas.
Una vez realizado el análisis léxico (omitiendo la gestión de errores en este curso) se procede al proceso de traducción del código fuente a código objetivo. En este proceso utilizaremos un modelo clásico basado en Sistemas de Traducción basados en Sintaxis Directa (SDTS). Con ellos, el proceso de traducción se reduce a un proceso de detección de pertenencia al lenguaje generado por la gramática que define nuestro lenguaje fuente (Problema de Palabra para gramáticas libres de contexto, con gramática fijada "a priori") y a un proceso de descripción de un árbol de derivación de la palabra dada (en el caso de ser aceptada). Los algoritmos que realizan esta tarea son los algoritmos de *parsing* (o Análisis Sintáctico) y la descripción de algunos de ellos es el objetivo de este Capítulo.

6.1 Introducción

El problema al que nos enfrentamos es el siguiente:

Problema Motívico 4 (Problema de Traducción.)**.** *Fijados dos lenguajes de programación $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$, y dada una palabra ω realizar la tarea siguiente:*

- *Decidir si $\omega \in L_1$,*
- *En caso de respuesta afirmativa, traducir ω a una palabra ω_2 en el lenguaje de programación L_2 (i.e. $\omega_2 \in L_2$).*

La traducción en lenguajes naturales sigue siendo un proceso complejo e incompleto (*"traduttore, traditore"*). En lenguajes formales, si ambos son dados por gramáticas libres de contexto, esta tarea es factible y computable. De esto trata el presente Capítulo.
Sin embargo, el proceso de traducción no es evidente y acudiremos a sistemas de traducción basados en la existencia de una relación directa entre las sintaxis de ambos lenguajes (STDS's).

Definición 133 (Syntax–Directed Translation Scheme)**.** *Un esquema de traducción sintáctica directa (SDTS) es un quíntuplo $(V, \Sigma, \Delta, Q_0, P)$ donde:*

- *V es un conjunto finito de símbolos llamados variables o símbolos no terminales.*
- *Σ es un alfabeto finito, llamado alfabeto de input.*
- *Δ es un segundo alfabeto finito llamado alfabeto de output.*
- *$Q_0 \in V$ es un símbolo distinguido llamado símbolo inicial del $SDTS$.*

- $P \subseteq V \times (V \cup \Sigma)^* \times (V \cup \Delta)^*$ *es un conjunto finito conocido como conjunto de pares de producciones.*

Escribiremos las producciones mediante: $A \mapsto (\alpha, \beta)$ para denotar la terna $(A, \alpha, \beta) \in P$.
Un SDTS es un mecanismo de traducción basado en los componentes sintácticos de dos gramáticas.

Definición 134 (Input and Output Grammars)**.** *Dado un SDTS* $T := (V, \Sigma, \Delta, Q_0, P)$ *disponemos de dos gramáticas libres de contexto asociadas:*

- *La gramática de input (o fuente)* $T_{input} := (V, \Sigma, Q_0, P_{input})$ *donde:*

$$P_{input} := \{A \mapsto \alpha \ : \ \exists \beta \in (V \cup \Delta)^*, \ A \mapsto (\alpha, \beta) \in P\}.$$

- *La gramática de output (u objetivo)* $T_{output} := (V, \Delta, Q_0, P_{output})$ *donde:*

$$P_{output} := \{A \mapsto \beta \ : \ \exists \alpha \in (V \cup \Sigma)^*, \ A \mapsto (\alpha, \beta) \in P\}.$$

Nótese que ambas gramáticas son gramáticas libres de contexto. Puede observarse que un sistema de traducción por sintaxis directa $SDTS := (V, \Sigma, \Delta, Q_0, P)$ genera un proceso de traducción entre dos lenguajes. Una traducción es una aplicación:

$$\tau : L_1 \subseteq \Sigma^* \longmapsto L_2 \subseteq \Delta^*,$$

que transforma palabras (programas, ficheros, códigos) en el primer lenguaje en palabras (programas, ficheros, códigos) en el segundo lenguaje. La "traducción" asociada a nuestro sistema $SDTS$ anterior funciona del modo siguiente:

- El lenguaje del cual tomamos las entradas es L_1 y es el lenguaje generado por la gramática T_{input}, i.e. $L_1 := L(T_{input})$.
- El lenguaje al cual deben pertenecer las salidas es L_2 y es el lenguaje generado por la gramática T_{output}, i.e. $L_2 := L(T_{output})$.

La traducción funciona del modo siguiente: dado $\omega \in L_1$ sea

$$Q_0 \to \omega_1 \to \omega_2 \to \cdots \to \omega_n = \omega,$$

una cadena de computaciones o derivaciones en P_{input} (i.e. las producciones de la gramática T_{input}) . Por construcción, tendremos unas producciones en P que se corresponden con los pasos realizados.
Así, el paso de ω_i a ω_{i-1} se habrá realizado porque existen $\gamma, \rho \in \Sigma^*$ tales que $\omega_{i-1} = \gamma A_i \rho$, con $A_i \in V$ y existe una producción $A_i \mapsto \alpha_i$ en P_{input} tal que $\omega_i = \gamma \alpha_i \rho$.
Pero la producción en P_{input} debe proceder de una producción en P que ha de tener la forma:

$$A_i \mapsto (\alpha_i, \beta_i),$$

Ahora comienza a actuar el SDTS del modo siguiente:

- Comenzamos con $Q_0 \to \eta_1 = \beta_1$. Es decir, realizamos la producción emparejada a la producción de P_{input} usada inicialmente.

- Recursivamente, si hemos calculado $Q_0 \to \eta_1 \to \ldots \to \eta_i$, y si la variable A_i está en η_i, recodamos el par de producciones $A_i \mapsto (\alpha_i, \beta_i)$ en P. Si $\eta_i = \gamma_1 A_i \gamma_2$, entonces definimos $\eta_{i+1} = \gamma_1 \beta_i \gamma_2$, en caso contrario devolvemos "error".

Ahora tendremos una cadena de derivaciones basadas en P_{output} de la forma:

$$Q_0 \to \eta_1 \to \eta_2 \to \cdots \to \eta_n = \omega'.$$

Las palabras $\eta_i \in \Delta^*$ se obtienen inductivamente aplicando las producciones $A_i \mapsto \beta_i$ de P_{output}. Si tras el proceso anterior hallamos una forma terminal ω', tendremos que $\omega' \in L_2 = L(T_{output})$ y la podemos denotar mediante $\omega' = \tau(\omega)$ y la llamaremos traducción de ω mediante el SDTS P.
Nótese que ω' no es necesariamente única y ni siquiera tiene por qué existir. En primer lugar, porque no hemos indicado cuál de las posibles apariciones de la variable A_i es la que hay que reemplazar por β_i. También carece del rasgo determinista porque podría haber más de una producción en el SDTS cuya primera producción fuera $A_i \mapsto \alpha_i$. Finalmente podría incluso que no no hubiera ninguna variable A_i en η_{i-1} para reemplazar, con lo que $\eta_{i-1} = \eta_i$. La conclusión es que, en general, τ es correspondencia y no necesariamente aplicación.
Por todo ello, nos conformaremos con disponer de un SDTS de tipo simple.

Definición 135 (SDTS simple)**.** *Un sistema de traducción por sintaxis directa* $T := (V, \Sigma, \Delta, s_0, P)$ *se llama simple, si para cada producción*

$$A \mapsto (\alpha, \beta),$$

las variables que aparecen en α *son las mismas, repetidas con la misma multiplicidad*[1] *que las que aparecen en* β*.*

Por simplicidad de nuestro análisis podemos incluso suponer que el orden en que se presentan las variables en cada una de las producciones (orden de "lectura", de izqda. a derecha, de derecha a iazquierda o con otras ordenaciones) de T es el mismo. En todo caso, una traducción basada en SDTS trata fielmente de seguir el proceso sintáctico de generación de ω en la primera gramática, para reproducirla en la segunda y así producir la traducción.
Nótese que el proceso de traducción natural consistirá en disponer "a priori" de dos gramáticas libres de contexto G_1 y G_2 y generar, a partir de ellas, un SDTS T tal que $L(G_1) = L(T_{input})$ y $L(G_2) = L(T_{output})$. Por simplicidad de la discusión admitiremos que disponemos de un $STDS$ a priori entre dos lenguajes de programación L_1 y L_2.
A partir de él nos planteamos el problema siguiente:

[1] Se podría incluso suponer que la variables aparecen en el mismo orden, éso conduciría a traducciones aún más inmediatas.

6.1.1 El problema de parsing: Enunciado

Fijado un lenguaje libre de contexto $L \subseteq \Sigma^*$, dada una palabra $\omega \in \Sigma^*$, entonces resolver:

- *Decidir si $\omega \in L$.*
- *En caso de respuesta afirmativa, dar un árbol de derivación (o una derivación) que produzca ω.*

El objetivo general de este Capítulo es resolver el problema de parsing siguiendo varias estrategias. Obsérvese que disponiendo de un algoritmo que resuelva el problema de parsing para la gramática P_{input} y disponiendo de un SDTS, disponemos de un compilador (aunque sea indeterminista) que traduzca los "programas" de un lenguaje de programación al otro.
Antes de comenzar nuestro análisis del problema de parsing, dedicaremos un rato a recordar algunos términos informales sobre compiladores.

6.2 Palabras Técnicas de uso Común sobre Compiladores, Traductores, Intérpretes

Presentamos algunas ideas de las palabras de uso común en el mundo de los compiladores. Dejamos el formalismo para el curso próximo.

Definición 136 (Compilador)**.** *Un compilador es un programa (o conjunto de programas) que transforma:*

- *un texto escrito en un lenguaje de programación (lenguaje fuente)*
- *en un texto escrito en otro lenguaje de programación (lenguaje objetivo)*

El texto en lenguaje fuente se llama código fuente. La salida del compilador se llama código objetivo.

Notación 137 (Terminología Básica)**.** *El uso más común de los términos suele ser:*

- **Compilador:** *suele usarse este término cuando el lenguaje objetivo es de nivel más bajo que el lenguaje fuente.*
- **Fuente a Fuente:** *transforma código entre lenguajes de alto nivel. Se suele usar el término reescritura cuando el lenguaje fuente es exactamente el lenguaje objetivo.*
- **Decompilador:** *el lenguaje objetivo es de mayor nivel que el lenguaje fuente. Su uso es harto infrecuente.*

6.2.1 Traductores, Compiladores, Intérpretes

El resultado de un Traductor–Compilador es esencialmente un ejecutable. Sin embargo, presenta un inconveniente, sobre todo en el manejo de códigos fuente de gran tamaño:
El código debe ser primero compilado y sólo tras finalizar la compilación puede ser ejecutado.
La alternativa son los **Intérpretes**. Un Intérprete toma como entrada un lenguaje en código fuente y procede el modo siguiente con cada instrucción (o parte) del código fuente realiza las siguientes tareas:

- carga la instrucción,
- analiza la instrucción,
- ejecuta la parte del código fuente.

6.2.1.0.1 Ventajas del Intérprete. Tiene las siguientes ventajas:

- El programador trabaja en forma interactiva y va viendo los resultados de cada instrucción antes de pasar a la siguiente.
- El programa se utiliza sólo una vez y no importa tanto la velocidad (después se compila y se usa solamente el ejecutable).
- Se espera que cada instrucción se ejecute una sola vez.
- Las instrucciones tienen formas simples y son más fáciles de analizar.

6.2.1.0.2 Inconvenientes de los Intérpretes. Tienen los siguientes inconvenientes:

- La velocidad puede ser del orden de 100 veces más lento que la de un ejecutable.
- No sirve cuando se espera que las instrucciones se ejecuten frecuentemente.
- Tampoco es interesante cuando las instrucciones sean complicadas de analizar.

Ejemplo 16. *Entre los intérpretes más habituales:*

- *shell El intérprete de comandos de Unix. Es una instrucción para el sistema operativo Unix. Se introduce dando el comando de forma textual. Actúa como se ha indicado: comando a comando. El usuario puede ver la acción de cada comando.*
- *LISP Es un lenguaje de programación de procesado de Listas. Common LISP.*
- *Un Intérprete de SQL. Ver curso previo de Bases de datos.*

6.2.1.1 Compiladores Interpretados.

Combinan las cualidades de Compiladores e intérpretes.

- Transforma el código fuente en un lenguaje intermedio.
- Sus instrucciones tienen un formato simple y fácil de analizar.
- La traducción desde el lenguaje fuente al lenguaje intermedio es fácil y rápida.

Ejemplo 17 (Java)**.** *Tiene las propiedades siguientes:*

- *El código JVM (Java Virtual Machine) es un lenguaje intermedio entre Java, de una parte, y Ensamblador y código máquina.*
- *La interpretación de JVM es más rápida que si hubiera un intérprete directo de Java.*
- *JVM es un traductor más un intérprete:*
  - *Traduce código Java a código JVM.*
  - *Interpreta código JVM.*

6.2.2 Las etapas esenciales de la compilación.

6.2.2.1 La Compilación y su entorno de la programación.

Primero ubicaremos el proceso de compilación dentro del proceso completo:

- Recibimos el código fuente y realizamos un *pre–procesamiento* para eliminar extensiones.
- El Compilador traduce el código fuente (limpio de extensiones) en código objetivo (en lenguaje ensamblador).
- El ensamblador asociado a cada máquina concreta transforma (e interpreta) cada ejecutable en ensamblador en código máquina (el código traducido a la máquina concreta en la que será ejecutado).

6.2.2.2 Etapas del Proceso de Compilación.

Las etapas fundamentales de la compilación son las siguientes:

a. **Análisis Léxico:** Utilizando expresiones regulares y/o autómatas finitos, se verifica si el código fuente sigue las reglas básicas de la gramática (regular) en este caso que define el lenguaje. A modo de ejemplo, definimos los tipos admisibles de datos (pongamos real) mediante una expresión regular, después usamos el autómata correspondiente para reconocer una entrada del lenguaje fuente que pertenece al lenguaje definido por esa expresión regular. En caso de encontrar un error, devuelve un mensaje de error.

b. **Análisis Sintáctico.** De nuevo usaremos el Problema de Palabra como referente y como sustrato las gramáticas libres de contexto y los autómatas con pila del Capítulo anterior.

c. **Análisis Semántico.** Revisiones para asegurar que los componentes de un programa se ajustan desde el plano semántico.

d. **Generador de Código Intermedio:** Un código intermedio que sea fácil de producir y fácil de traducir a código objetivo.

e. **Optimización del Código.** Trata de mejorar el código intermedio[2].

f. **Generador de Código Final:** Etapa final de la compilación. Genera un código en ensamblador.

6.2.2.3 En lo que concierne a este Capítulo.

En lo que concierne a este curso nos interesaremos solamente por las etapas de **Análisis Léxico** y **Análisis Sintáctico (o parsing)**.
Como los analizadores léxicos contienen una menor dificultad, nos centraremos en el Análisis Sintáctico o parsing.

6.3 Conceptos de Análisis Sintáctico

El problema central del Análisis Sintáctico es el

Problema 78 (Parsing o Análisis Sintáctico)**.** *Dada una forma terminal $\omega \in \Sigma^*$ sobre una gramática $G := (V, \Sigma, Q_0, P)$, se pide producir:*

- *Un árbol sintáctico con el que continuará la siguiente etapa de la compilación, si la palabra $\omega \in L(G)$.*
- *Un informe con la lista de errores detectados, si la cadena contiene errores sintácticos. Este informe deberá ser lo más claro y exacto posible.*

Omitiremos las distintas estrategias de Manejo de Errores, para concentrarnos en los otros dos aspectos.

6.3.1 El problema de la Ambigüedad en CFG

Definición 138 (Derivaciones Leftmost y Rightmost)**.** *Sea $G = (V, \Sigma, Q_0, P)$ una gramática libre de contexto. Sean $c, c' \in (V \cup \Sigma)^*$ dos formas sentenciales.*

a. *Diremos que c' se obtiene mediante derivación "más a la izquierda" (o leftmost) de c, si existen $\omega \in \Sigma^*, A \in V, \alpha \in (V \cup \Sigma)^*$, y existe una producción $A \mapsto \beta$, con $\beta \in (V \cup \Sigma)^*$ tales que*

$$c = \omega A \alpha, \quad c' = \omega \beta \alpha.$$

Denotaremos mediante $c \to_{lm}^{G} c'$.

[2] La generación de código óptimo es un problema **NP**–completo.

b. *Diremos que c' se obtiene mediante derivación "más a la derecha" (o rightmost) de c, si existen $\omega \in (V \cup \Sigma)^*, A \in V, \alpha \in \Sigma^*$, y existe una producción $A \mapsto \beta$, con $\beta \in (V \cup \Sigma)^*$ tales que*

$$c = \omega A \alpha, \;\; c' = \omega \beta \alpha.$$

Denotaremos mediante $c \to_{rm}^{G} c'$.

Usualmente, y si no hay confusión, omitiremos el super-índice ^{G}.

Nótese que hay dos elementos indeterministas en las traducciones así planteadas. De una parte tenemos la elección de sobre qué variable actuamos, de otro lado qué pareja de producciones elegimos. La idea de introducir derivaciones "más a la izquierda" o "más a la derecha" consiste en tratar de reducir el ingrediente indeterminístico a la hora de seleccionar qué variable es la variable sobre la que vamos a actuar con nuestras producciones. En el proceso leftmost, seleccionamos la variable que se encuentra más a la izquierda y actuamos sobre ella mediante alguna de las producciones aplicables. En el rightmost hacemos lo propio con la variable que se encuentra más a la derecha. Nótese que esto no significa que el proceso de derivación sea determinístico: aunque seleccionemos sobre cuál de las variables actuamos primero, es claro que puede haber más de una producción que actúa sobre esa variable y mantiene sus rasgos indeterministas.

Definición 139 (Cadenas de Derivaciones Leftmost y Rightmost)**.** *Con las mismas notaciones de la Definición anterior,*

a. *Diremos que c' es deducible de c mediante derivaciones más a la izquierda (y lo denotaremos mediante $c \vdash_{lm}^{G} c'$) si existe una cadena finita de derivaciones más a la izquierda que va de c a c'. Esto es, si existen:*

$$c = c_0 \to_{lm}^{G} c_1 \to_{lm}^{G} \cdots c_{k-1} \to_{lm}^{G} c_k = c'.$$

b. *Diremos que c' es deducible de c mediante derivaciones más a la derecha (y lo denotaremos mediante $c \vdash_{rm}^{G} c'$) si existe una cadena finita de derivaciones más a la derecha que va de c a c'. Esto es, si existen:*

$$c = c_0 \to_{rm}^{G} c_1 \to_{rm}^{G} \cdots c_{k-1} \to_{rm}^{G} c_k = c'.$$

De nuevo, omitiremos el super-índice ^{G} siempre que su omisión no induzca confusión alguna.

La idea es una cadena de pasos más a la izquierda, esto es, elegimos la variable más a la izquierda (leyendo de izquierda a derecha) de c. Aplicamos una producción de la gramática sobre esa variable y c' es simplemente el resultado de aplicar esa producción.

Ejemplo 18. *Tomemos la gramática cuyas producciones son:*

$$P := \{Q_0 \mapsto AB \mid CA \mid AQ_0 \mid 0, A \mapsto BA \mid 0A0 \mid 1, B \mapsto Q_0A, C \mapsto 1\}.$$

Una cadena de derivaciones leftmost (más la izquierda) sería la siguiente:

$$Q_0 \to AB \to CAB \to 1AB \to 11B \to 11Q_0A \to 110A \to 1101.$$

Una cadena de derivaciones rightmost (más a la derecha) sería la siguiente:

$$Q_0 \to AB \to AQ_0A \to AQ_01 \to A01 \to 0A001 \to 01001.$$

Definición 140 (Gramáticas Ambiguas)**.** *Una gramática se dice ambigua si existe una forma sentencial $\omega \in (V \cup \Sigma)^*$ alcanzable desde el símbolo inicial (i.e $Q_0 \vdash_G \omega$) tal que existen al menos dos computaciones (derivaciones) más a la izquierda (o más a la derecha) distintas que permiten generar ω.*

Ejemplo 19. *Tomemos la gramática $P := \{E \mapsto E + E \mid E * E \mid a\}$. Ahora disponemos de dos cadenas de derivación para $a + a * a$ distintas:*

$$E \to_{lm} E + E \to_{lm} a + E \to_{lm} a + E * E \to_{lm} a + a * E \to_{lm} a + a * a.$$

Y también

$$E \to_{lm} E * E \to_{lm} E + E * E \to_{lm} a + E * E \to_{lm} a + a * E \to_{lm} a + a * a.$$

Por lo que la anterior gramática es ambigua.

El problema con la ambigüedad de las gramáticas se enfrenta a la siguiente dificultad:

Teorema 141. *Decidir si una gramática libre de contexto es ambigua es indecidible (i.e. no existe algoritmo que permita decidir la cualidad de ser ambigua).*

La razón es la Indecidibilidad del Post Correspondence Problem. En algunos casos se pueden dar estrategias que eliminen la condición de ambigüedad de algunas gramáticas, pero el anterior Teorema nos garantiza que ésto no es posible en general. En todo caso, los lenguajes de programación se diseñan generados mediante gramáticas no ambiguas.

6.3.2 Estrategias para el Análisis Sintáctico.

En este Capítulo nos dedicaremos a mostrar algunos ejemplos de análisis sintáctico, basados en distintos ejemplos de estrategias.

a. Estrategias de carácter general aplicables a cualquier gramática libre de contexto, incluyendo el caso indeterminista (en el Autómata con Pila) pero realizando un análisis determinístico. Hemos seleccionado la estrategia de `Parsing CYK` de Cocke[3], Younger[4] y Kasami[5]. Otros procesos generales, como la estrategia de Earley[6] , pueden seguirse en [Aho-Ullman, 72a]. Este tipo de análisis

[3] Hays se lo atribuye a Cocke en D.G. Hays. *Introduction to Computational Linguistics.* Elsevier, New York, 1967. Véase también [Cocke-Schwartz, 70].

[4] D.H. Cocke. " Recognition and parsing of context–free languages in time n^3. *Information and Control* **10** (1967) 189–208.

[5] Véase el informe técnico de 1965, AFCRL–65–758, escrito por Kasami y el paper T. Kasami, K. Torii."A syntax analysis procedure for unambiguous context–free grammars". *J. of the ACM* **16** (1969) 423-431.

[6] J. Earley. *An Efficient context–free parsing algorithm*, Ph.D. Thesis, carnegie–Mellon, 1968.Y el trabajo J. Earley. "An Efficient context–free parsing algorithm". *Commun. ACM* **13** (1970) 94–102.

generalista, que admite lenguajes libres de contexto indeterminísticos, tendrá un coste en complejidad superior a los otros. De hecho, la complejidad del algoritmo que presentamos es $O(n^3)$. Entre los algoritmos de Análisis Sintáctico generalista, el más eficiente es la variante de CYK introducida por Leslie G. Valiant[7] quien, en 1975, adapta estrategias de multiplicación rápida de matrices a la estrategia CYK para obtener un algoritmo de parsing generalista en tiempo $O(n^{2.38})$. Dado que estas estrategias algorítmicas se escapan a los contenidos usuales del curso las omitiremos,pero recomendamos al lector acudir a las fuentes de autores cono V. Strassen, Sh. Winograd, V. Pan o, más recientemente, A. Storjohann o G. Villard y sus referencias.

b. **Análisis Descendente (Top–Down):LL(k).** Es un modelo de análisis sintáctico adaptado a *lenguajes libres de contexto deterministicos* (es decir, que son aceptados por un autómata con pila determinístico). Por tanto, no son analizadores generaliastas como los anteriores y no sirven para tratar todos los lenguajes libres de contexto. El modelo de análisis recorre un árbol de derivación desde la raíz (con el símbolo inicial Q_0, hasta las hojas. Basa su construcción en el uso de derivaciones "más a la izquierda". El proceso resulta determinista en el caso de que la gramática involucrada sea de un tipo específico: gramáticas $LL(k)$ (de " from left to right with left canonical derivation and a look-ahead of k symbols deterministic recognizable"). Haremos la exposición para gramáticas $LL(1)$. Estas estretagias más intuitivas (para un Europeo) de parsing fueron introducidas por N. Wirth en [Wirth, 96] y adaptadas a lenguajes como `Pascal`. Un ejemplo de generador de parsers cuya ideologí se apoya fuertemente en lo que podemos llamar *LL–ismo* es ANTLR[8]. Hasta mediados de los 90, los analizadores para gramáticas en $LL(k)$ fueron descartados y considerados como impracticables porque tenían una complejidad exponencial en k (de hecho, la tabla de parsing $LL(k)$ crece de manera exponencial en k). La aparición de ANTLR (y su antecesor PCCTS) en 1992, revitalizó este tipo de analizadores, mostrando que la complejidad del caso peor es raramente alcanzada en la práctica.

c. **Análisis Ascendente (Bottom–Up):LR(k).** De nuevo es un modelo de análisis sintáctico válido solamente para *lenguajes libres de contexto deterministas*. Fueron introducidos por D.E. Knuth en [Knuth, 65]. Knuth demostraba que todos los lenguajes libres de contexto deterministas admiten algún analizador sintáctico $LR(k)$ y que todo lenguaje libre de contexto determinista admite una gramática $LR(1)$. También demostraba que la complejidad del análisis $LR(k)$ era lineal en el tamao de la entrada. El proceso recorre un árbol de derivación desde las hojas hasta la raíz (Bottom–Up). Basa su análisis en derivaciones más a la derecha (right-most). Por ello el output de este tipo de parsers es el reverso de un árbol de derivación de una palabra aceptada. El término LR viene de " Left-to-right scan, Rightmost derivation. Durante años

[7] L.G. Valiant. " General Context–Free Recognition in less than cubic time". *J. of Comput. and Syst. Science* **10** (1975) 308–314.

[8] Consultar en `http://www.antlr.org/`, por ejemplo

ha sido el analizador sintáctico preferido por los programadores. Mencionaremos como ejemplos:

- `Yacc`: *Yet Another Compiler Compiler* desarrollado por los laboratorios AT & T. Genera código en lenguaje C y es mantenido actualmente por la compañía SUN
- `Bison`: Es el compilador de compiladores del proyecto GNU. Genera código en C++.
- `SableCC`: Creado por Étienne Gagnon, este programa contiene los últimos avances en materia de compiladores.

En la actualidad los analizadores dominantes son los basados en estretagias `LALR` que combinan la estrategia look ahead del diseño descedente, con la eficiencia de los analizadores LR.

A modo de comentario adicional, el análisis descendente es más intuitivo y permite gestionar mejor los errores en el código fuente. Cuando oímos trozos de una frase, estamos analizando smiultáneamente (esta es la estretgia *look ahead*. No necesitamos oír la frase entera para ir deduciendo que quién será el sujeto o quién el predicado. Es más, notamos, sin tener que acabar la frase podemos deducir que no es una frase correcta en español.

6.4 Análisis CYK

Es el modelo de análisis debido a Cocke, Younger y Kasami. Es un modelo aplicable a cualquier gramática en forma normal de Chomsky (CNF) y $\lambda-$libre.

6.4.1 La Tabla CYK y el Problema de Palabra.

El input del algoritmo está formado por una gramática libre de contexto $G = (V, \Sigma, q_0, P)$, y por una forma terminal, es decir, $\omega := a_1 a_2 \cdots a_{n-1} a_n \in \Sigma^*$.
El output del algoritmo es una tabla triangular. Para interpretarlo, podemos entender la tabla de output como una aplicación:

$$t : \{(i,j) \ : 1 \leq i \leq n, 1 \leq j \leq n - i + 1\} \longrightarrow \mathcal{P}(V),$$

donde V son los símbolos no terminales de la gramática y $\mathcal{P}(V)$ son los subconjuntos de V. A la imagen $t(i,j) \in \mathcal{P}(V)$ la denotaremos con sub-índices. Es decir, escribiremos $t_{i,j} \in \mathcal{P}(V)$.
En el conjunto $t_{i,j}$ escribiremos todas las variables $A \in V$ tales que

$$A \vdash^G a_i a_{i+1} \cdots a_{i+j-1}.$$

Nótese que, en realidad, t depende de G y de ω.

INPUT: Dos elementos:

- Una gramática $G := (V, \Sigma, Q_0, P)$ libre de contexto, en forma normal de Chomsky y $\lambda-$libre.
- Una palabra $\omega = a_1 a_2 \cdots a_n \in \Sigma^*$ de longitud n.

OUTPUT: La tabla

$$t : \{(i,j) \ : 1 \leq i \leq n, 1 \leq j \leq n-i+1\} \longrightarrow \mathcal{P}(V),$$

según las propiedades anteriores.
Inicializar: Hallar para cada i, $1 \leq i \leq n$ los conjuntos siguientes:

$$t_{i,1} := \{A \ : \ A \mapsto a_i \in P\}.$$

$j := 1$
while $j \leq n$ **do**
  for $i = 1$ **to** $n - j + 1$ **do**
    $t_{i,j} := \{A \ : \ \exists k, \ 1 \leq k < j, \ B \in t_{i,k}, \ C \in t_{i+k,j-k}, \ \text{ and } A \mapsto BC \in P\}$.
    next i
  od
  next j
od
end

Teorema 142. *El algoritmo calcula la Tabla t del análisis CYK. El tiempo de ejecución del algoritmo es $O(n^3)$ y la memoria usada es $O(n^2)$, donde n* [9].

Demostración. Es una mera comprobación. □

Ejemplo 20. *Hallar la tabla para la gramática G cuyas producciones son:*

$$Q_0 \mapsto AA \mid AQ_0 \mid b, \ A \mapsto Q_0 A \mid AQ_0 \mid a.$$

y la palabra $\omega = abaab$. La solución se puede encontrar en la tabla 6.1

Teorema 143. *Con las notaciones anteriores, $\omega \in L(G)$ si y solamente si $Q_0 \in t_{1,n}$. En particular, el cálculo de la tabla por el método CYK resuelve el problema de palabra para gramáticas en forma normal de Chomsky y $\lambda-$libres en tiempo $O(n^3)$ y espacio $O(n^2)$.*

Demostración. Por definición de la propia tabla, $Q_0 \in t_{1,n}$ si y solamente si $Q_0 \vdash \omega$. □

[9] Aunque se debe tener en cuenta que, en el tamaño de la tabla también juega su papel $\sharp(V)$, conb lo que la complejidad depende también del tamaño de la gramática. Sin embargo, la gramática permanece "fija" en el proceso de análisis sintáctico correspondiente, con lo que ese tamaño de la gramática se "oculta" dentro de la constante de $O()$.

$t_{i,j}$	1	2	3	4	5
1	A	Q_0	A	A	Q_0
2	Q_0	A	Q_0	A	
3	A	Q_0	Q_0		
4	A	Q_0,A			
5	Q_0				

Table 6.1: Los diferentes $t_{i,j}$ donde i es el número de fila y j es la columna

6.4.2 El Árbol de Derivación con las tablas CYK.

A partir de la construcción de la tabla CYK, podemos desarrollar un algoritmo que no sólo resuelva el problema de palabra para gramáticas libres de contexto, sino que, además, genera un árbol de derivación para las palabras aceptadas.
Lo que haremos será definir una serie de aplicaciones: $gen(i, j, -)$ definidas en los subconjuntos $t_{i,j}$ de la tabla construida a partir del algoritmo CYK antes definido.
Comenzaremos introduciendo una enumeración en al conjunto de las producciones. Elegimos números enteros positivos $\{1, \ldots, N\}$ que usaremos para asignar un número a cada producción de P. La manera de enumerar es libre pudiendo elegir la que más nos convenga o guste. Esta es una cualidad significativa del análisis CYK: no importa la preferencia de nuestra enumeración, podríamos recuperar todas las posibles opciones de árbol de derivación. Así, buscando localmente, podemos encontrar una de las soluciones "globales" (con lo que admite una ideología "greedy" (voraz)). De hecho, esto es lo que permite enfrentar clases de lenguajes libres de contextos cualesquiera (sean o no determinísticos y forma parte intr´inseca de su naturaleza generalista.
La idea de base es la siguiente:

- Si $A \notin t_{i,j}$, la imagen de $gen(i, j, A)$ no está definida. En ese caso devolveremos **Error**.

- Si $A \in t_{i,1}$, es porque la producción (m) es de la forma $A \mapsto a_i$. Definamos $gen(i, 1A) := [m]$

- Si $A \in t_{i,j}$, entonces existe un k, $1 \leq k < j$ y existe una producción con número (m) de la forma $A \mapsto BC$ con $B \in t_{i,k}$, $C \in t_{i+k,j-k}$. Entonces,

$$gen(i, j, A) = [m, gen(i, k, B), gen(i + k, j - k, C)].$$

Esta definición adolecede de una dificultad fundamental:
No es aplicación puesto que podría haber más de una producción con las propiedades prescritas. Por ejemplo, podríamos tener:

- Una producción con número (m) de la forma $A \mapsto BC$, $B \in t_{i,k}$, $C \in t_{i+k,j-k}$ y

- otra producción con número (r) de la forma $A \mapsto XY$, $X \in t_{i,k}$, $Y \in t_{i+k,j-k}$.

Para corregir este "indeterminismo" y aprovechando su naturaleza de matroide, modificamos la definición de *gen* y lo transformamos en una determinística mediante, por ejemplo, eligiendo la producción de menor número entre las posibles. Es decir, el último item de la definición anterior queda:

- Si $A \in t_{i,j}$, consideremos todas las producciones de P tales que existe un k, $1 \leq k < j$ y la producción $A \mapsto BC$ con $B \in t_{i,k}$, $C \in t_{i+k,j-k}$. *Sea m el mínimo de las enumeraciones de tales producciones.* Entonces, definiremos

$$gen(i, j, A) = [m, gen(i, k, B), gen(i + k, j - k, C)]$$

6.4.3 El Algoritmo de Análisis Sintáctico CYK

Supondremos fijada una gramática libre de contexto $G := (V, \Sigma, Q_0, P)$ en CNF, $\lambda-$libre, en la que las producciones de P están enumeradas de 1 a N.

INPUT: Una forma terminal $x = x_1x_2 \cdots x_n \in \Sigma^*$.
OUTPUT: Si $x \in L(G)$, devuelve un árbol de derivación de x en G, en caso contrario devuelve **error**.

Calcular la tabla $\{t_{i,j}\}$ del algoritmo CYK anterior (obsérvese que depende de x y, obviamente, de G).
if $Q_0 \notin t_{1,n}$ OUTPUT **error**
    else do
        eval $gen(1, n, Q_0)$
fi
OUTPUT $gen(1, n, Q_0)$.
end

Teorema 144. *El anterior algoritmo da como output un árbol de derivación de ω si $\omega \in L(G)$ y devuelve* **error** *en caso contrario. El tiempo de ejecución es del orden $O(n^3)$ y el espacio consumido es de orden $O(n^2)$.*

Ahora ya estamos listos para hallar una derivación de la palabra $x = abaab$ Empezaremos etiquetando las producciones:

(1)	$Q_0 \mapsto AA$	(4)	$A \mapsto Q_0A$
(2)	$Q_0 \mapsto AQ_0$	(5)	$A \mapsto AQ_0$
(3)	$Q_0 \mapsto b$	(6)	$A \mapsto a$

y ahora volvamos a la tabla 6.1. Por la definición sabemos que

$$\begin{aligned} gen(1, 5, Q_0) &= [2, gen(1, 1, A), gen(2, 4, Q_0)] \\ &= [2, [6], [1, gen(2, 2, A), gen(4, 2, Q_0)]] \\ &= [2, [6], [1, [4, gen(2, 1, Q_0), gen(3, 1, A)], [5, gen(4, 1, A), gen(5, 1, Q_0)]]] \\ &= [2, [6], [1, [4, [3], [6]], [5, [6], [3]]]] \end{aligned}$$

Podemos asociar esto a la representación de un árbol utilizando pre orden.
[.2 6 [.1 [.4 3 6] [.5 6 3]]] [.Q_0 [.A a] [.Q_0[.A[.Q_0b][.Aa]][.A[.Aa][.Q_0b]]]]

Allado, hemosdibujadoelárboldederivaciondelapalabra.

6.5 Traductores Push–Down.

Los procesos de traducción con gramáticas libres de contexto son gestionados por un modelo de máquina basado en los autómatas con pila descritos en el Capítulo precedente: *Los traductores push–down* o traductores con pila.
Informalmente, un traductor con pila es un objeto compuesto de los siguientes elementos:

- **Un autómata con pila.** Esto es, disponemos de una cinta de entrada (IT), una unidad de control con una cantidad finita de memoria, y una pila.
- **Una cinta de output.** En la que el autómata simplemente puede escribir, no puede leer sus contenidos, y puede avanzar un paso a la derecha siempre que la celda anterior no esté vacía.

Las operaciones de un traductor con pila son sucesiones de operaciones del tipo siguiente:

a. **Read.** Lee una celda en la cinta de entrada y el top de la pila. Eventualmente puede hacer operaciones de lectura λ en la cinta de entrada. por supuesto, lee también el estado actual en la unidad de control.

b. **Transition.** De acuerdo con una función de transición (o de una tabla como la que usaremos en las secciones siguientes) el autómata indica tres operaciones básicas a realizar en cada uno de los cuatro elementos.

c. **Write and Move.** Escribirá en cinta, pila y/o cinta de outpt en función de las reglas naturales:

   - En la cinta de Input: Si la lectura es una **Lambda**-lectura, no hace nada en la cinta de input. En caso de tratarse de una lectura propiamente(**Read**) borrará un símbolo de de la cinta y avanzará un paso hacia la derecha.
   - En la unidad de control, modifica el estado conforme se indica en la Transición.
   - En la pila realiza la operación `push`(`pop`(Pila), z) donde z es el símbolo indicado por la transición. Será una operación `push` si $z \neq \lambda$ y una operación `pop` si $z = \lambda$, como ya se indicó en los Autómatas con Pila.
   - En la cinta de Output escribe lo que se le indique. Puede ser que no escriba nada, en cuyo caso no se mueve, o que escriba un símbolo en cuyo caso se mueve un paso a la derecha hasta la siguiente celda vacía.

La computación se termina con *pila y cinta vacías.* Es decir, el autómata funciona con un gran ciclo **while** cuya condición de parada es que la cinta y la pila están vacías.
En ese caso, se dice que el input es aceptado (lo que significará que el input está en el lenguaje generado por la gramática de input). El output, es el contenido de la cinta de output y es el árbol sintáctico de derivación de la palabra escrita en la cinta

de input (en un cierto sentido, la "traducción" de la palabra escrita en la cinta de input).
Esta definición informal va acompañada de una definición formal que escribimos a continuación:

Definición 145 (PDT)**.** *Un traductor con pila (push–down transducer o PDT), es una lista* $T := (Q, \Sigma, \Gamma, \Delta, Q_0, Z_0, F, \delta)$ *donde:*

a. Q *es un conjunto finito (espacio de estados)*

b. Σ *es un alfabeto finito, llamado alfabeto del input.*

c. Γ *es un alfabeto finito, llamado alfabeto de la pila.*

d. Δ *es un alfabeto finito, llamado alfabeto del output.*

e. $q_0 \in Q$ *es el estado inicial.*

f. $F \subseteq Q$ *son los estados finales aceptadores.*

g. Z_0 *es un símbolo especial, llamado fondo de la pila.*

h. δ *es una correspondencia llamada función de transición:*

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{Z_0\}) \to Q \times \Gamma^* \times \Delta^*.$$

6.5.0.1 Sistema de Transición asociado a un PDT.

Denominaremos configuraciones de un traductor push-down a los elementos del conjunto

$$\mathcal{S} := Q \times \Sigma^* \times Z_0 \cdot \Gamma^* \times \Delta^*.$$

Una configuración $c = (q, x, Z_0 z, y)$ está describiendo q como el estado de la unidad de control, x es el contenido de la cinta de input, $Z_0 z$ es el contenido de la pila, y es el contenido de la cinta de output.
De la manera obvia se describen las transiciones $c \to c'$ entre dos transiciones del sistema. Lo dejamos como ejercicio para los alumnos.
La configuración inicial en una palabra $\omega \in \Sigma^*$ será dada por:

$$I(\omega) := (q_0, \omega, Z_0, \lambda),$$

es decir, ω está en la cinta de input, q_0 en la unidad de control, la pila está vacía y la cinta de output también.
Una configuración final aceptadora es una configuración con pila y cinta vacías, esto es, una configuración de la forma (q, λ, Z_0, y) con $y \in \Delta^*$.
Una palabra $\omega \in \Sigma^*$ es aceptada si alcanza una configuración final aceptadora dentro del sistema de transición. Esto es, si ocurre que:

$$I(\omega) = (q_0, \omega, Z_0, \lambda) \vdash (q, \lambda, Z_0, y).$$

la palabra y es el resultado de la traducción de ω en el caso de que omega sea aceptada por el PDT (i.e. $y = \tau(\omega)$).

Nota 146. *En lo que sigue, la traducción se hará a través de parsing y SDTS. Por tanto, usaremos PDT's del modo siguiente: Si L_1 es el lenguaje del input y T_{input} es la gramática del input asociada a nuestro sistema SDTS, procederemos enumerado los producciones de T_{input}. Definiremos el lenguaje de output Δ como $\Delta := \{1, \ldots, N\}$, donde N es el número de las producciones de T_{input}. Así, el output de un analizador sintáctico (descrito también mediante un PDT) es una lista $i_1 i_2 \cdots i_r \in \Delta^*$, que indican (en modelo directo o en reverso) las producciones que se aplican.*

Nota 147. *En los casos que siguen, el PDT y sus estados se describirán mediante diversos tipos de tablas. En cada caso iremos mostrando cómo se construyen esas tablas.*

6.6 Gramáticas $LL(k)$: Análisis Sintáctico

Se trata de un modelo de análisis sintáctico descendente (top–down) basado en left–parsing (o sea, buscando árboles de derivación más a la izquierda) y es determinista para ciertas clases de gramáticas: las gramáticas $LL(k)$.
El ejemplo clásico de lenguaje de programación que admiten parsing LL(1) es `Pascal`. Véase la obra de Niklaus Wirth y la tradición "europea" de análisis sintático (cf. [Wirth, 96]).

6.6.1 FIRST & FOLLOW

Definición 148 (Frist)**.** *Sea $G := (V, \Sigma, Q_0, P)$ una gramática libre de contexto. Para cada forma sentencial $\alpha \in (V \cup \Sigma)^*$ y para cada $k \in \mathbb{N}$ definiremos la función*

$$FIRST_k^G(\alpha) := \{x \in \Sigma^* \ : \ \begin{cases} |x| = k & \exists \beta \in \Sigma^*,\ \alpha \vdash_{lm}^G x\beta \\ |x| < k & \alpha \vdash_{lm}^G x \end{cases} \}.$$

Omitiremos el superíndice ^{G} siempre que su presencia sea innecesaria por el contexto.

En otras palabras, el operador $FIRST_k$ asocia a cada forma sentencial los primeros k símbolos de cualquier forma terminal alcanzable desde α mediante derivaciones "más a la izquierda". Si α alcanza una forma terminal con menos de k símbolos $x \in \Sigma^*$, con derivaciones "más a la izquierda", entonces también x está en $FIRST_k(\alpha)$.
Si $\alpha \in \Sigma^*$, $FIRST_k(\alpha)$ son los primeros k símbolos de α. Más específicamente, si $\alpha := x_1 \cdots x_k x_{k+1} \cdots x_n \in \Sigma^*$, entonces

$$FIRST_k(\alpha) = \{x_1 \cdots x_k\},$$

y si $|\alpha| \leq k$, $FIRST_k(\alpha) = \{\alpha\}$.
Nos ocuparemos, sobre todo, del operador $FIRST(\alpha) := FIRST_1(\alpha)$. Para dar un algoritmo que los calcule comenzaremos con algunas propiedades básicas del Operador FIRST. Más aún, comenzaremos con una construcción de un operador entre lenguajes:

Definición 149. *Sean $L_1, \ldots, L_n \subseteq (V \cup \Sigma)^*$ lenguajes no vacíos. Definiremos el lenguaje $L_1 \oplus_1 \cdots \oplus_1 L_n \subseteq (V \cup \Sigma)^*$ mediante la siguiente igualdad: Sea $j \in \{1, \ldots, n\}$ tal que $\lambda \in L_i$ para $1 \leq i \leq j-1$ y $\lambda \notin L_j$. Entonces,*

$$L_1 \oplus_1 \cdots \oplus_1 L_n := \bigcup_{i=1}^{j} L_i.$$

Supongamos dada una aplicación

$$F : (V \cup \Sigma) \longrightarrow \mathcal{P}((V \cup \Sigma))^*,$$

escribiremos $\oplus_1^F \alpha$ para cada forma sentencial α queriendo denotar

$$\oplus_1^F \alpha := F(X_1) \oplus_1 \cdots \oplus_1 F(X_n),$$

cuando $\alpha = X_1 \cdots X_n$.

Ejemplo 21. *Dados $L_1 = \{\lambda, abb\}$ y $L_2 = \{b, bab\}$, entonces*

$$L_1 \oplus L_2 = \{abb, b, bab, \lambda\} = L_1 \cup L_2, \quad L_2 \oplus L_1 = \{b, bab\} = L_2. \tag{6.1}$$

Lema 150. *Con las anteriores notaciones, se tienen las siguientes propiedades.*

a. *Si $X = \lambda$, $FIRST(\lambda) = \{\lambda\}$.*

b. *Si $X = a \in \Sigma$, $FIRST(X) = \{a\}$.*

c. *Si $\alpha := X_1 \cdots X_n$ donde $X_i \in (V \cup \Sigma)^*$, entonces*

$$FIRST(\alpha) = \oplus_1^{FIRST} \alpha = FIRST(X_1) \oplus_1 \cdots \oplus_1 FIRST(X_n).$$

d. *Si V_λ son los símbolos no terminales que alcanzan la palabra vacía, entonces $\lambda \in FIRST(X)$ si y solamente si $X \in V_\lambda$.*

A partir de Lema anterior, el cálculo de $FIRST$ de una forma sentencial cualquiera puede reducirse al cálculo de los $FIRST$'s de sus símbolos. Definimos el siguiente algoritmo incremental:

INPUT: una gramática libre de contexto $G := (V, \Sigma, Q_0, P)$.

Hallar $V_\lambda := \{A \in V \ : \ A \vdash \lambda\}$.
if $A \in \Sigma$, **then** $F(A) := \{A\}$[10]
else do
$G(A) := \emptyset$

$$F(A) := \begin{cases} \{A\} & \text{si } A \notin V_\lambda \\ \{A, \lambda\} & \text{si } A \in V_\lambda \end{cases}$$

[10] Aceptaremos que $A \mapsto_{lm} A$.

while $F(A) \neq G(A)$ para algún $A \in V$ **do** [11]
$G(A) := F(A)$
$F(A) := \{\oplus_1^F \alpha \ : \ X \mapsto \alpha, \ X \in F(A)\} \cup \{F(A)\}$
od

OUTPUT: $F(A) \cap (\Sigma \cup \{\lambda\})$, para cada $A \in V \cup \Sigma$.

Proposición 151 (Evaluación de FIRST)**.** *El anterior algoritmo evalúa la función* $FIRST(X)$ *para cada* $X \in V \cup \Sigma$.

Definición 152 (FOLLOW)**.** *Con las mismas notaciones anteriores, para cada forma sentencial* $\alpha \in (V \cup \Sigma)^*$ *definiremos la función* $FOLLOW_k^G(\alpha)$ *del modo siguiente.*

- *Si existe una forma sentencial* $\omega\alpha$ *(i.e. si* $Q_0 \vdash_G \omega\alpha$*), con* $\omega \in (V \cup \Sigma)^*$*, entonces* $\lambda \in FOLLOW_k^G(\alpha)$.
- *Adicionalmente, definamos*

  $$FOLLOW_k^G(\alpha) := \{x \in \Sigma^* \ : \ Q_0 \vdash \omega\alpha\gamma, \ \omega, \gamma \in (V \cup \Sigma)^*, \ x \in FIRST_k^G(\gamma)\}.$$

  De nuevo, omitiremos el super–índice ^{G} *cuando no genere confusión.*

De nuevo nos ocuparemos solamente de $FOLLOW := FOLLOW_1$. Obsérvese que $FOLLOW_k(\alpha) \subseteq \Sigma^*$ y que para cada $x \in FOLLOW_k(\alpha)$, $|x| \leq k$.
Obsérvese que para cada variable $A \in V$, $FOLLOW(A)$ son todos los símbolos terminales que pueden aparecer a la derecha de A en alguna forma sentencial de la gramática. Si A estuviera al final de alguna forma sentencial, la palabra vacía también se añade.

[11] Nótese que la frase debe escribirse como **while** $\exists A \in V, F(A) \neq G(A)$**do**

INPUT: Una gramática libre de contexto $G := (V, \Sigma, Q_0, P)$ que supondremos libre de símbolos inútiles[12].

Hallar $FIRST(X)$, para cada $X \in (V \cup \Sigma)$.
$G(X) := \emptyset$, para cada $X \in V$
$F(Q_0) := \{\lambda\}$
$F(A) := \emptyset$, para cada $A \neq Q_0$.
while $F(A) \neq G(A)$ para algún $A \in V$, **do**
$G(A) = F(A)$ para cada $A \in V$

$$F(A) := \left[\bigcup_{B \mapsto \omega A \omega'} (FIRST(\omega') \setminus \{\lambda\})\right] \bigcup$$

$$\bigcup \left[\bigcup_{B \mapsto \omega A \omega',\ \lambda \in FIRST(\omega')} F(B)\right] \bigcup F(A)$$

od
OUTPUT: $F(A) \cap (\Sigma \cup \{\lambda\})$ para cada $A \in V$.

Ejemplo 22. *Consideremos la gramática con* $V := \{Q_0, E', X, T, T'\}$, $\Sigma := \{id, (,), +, *\}$. *Las producciones son:*

$$P := \{Q_0 \mapsto TE', E' \mapsto +TE' \mid \lambda, T \mapsto XT', T' \mapsto *XT' \mid \lambda, X \mapsto (Q_0) \mid id\}.$$

Tendremos

$$FIRST(Q_0) = \{(, id\}, FIRST(E') := \{+, \lambda\},$$
$$FIRST(X) = \{(, id\}, FIRST(T) = \{(, id\}, FIRST(T') = \{*, \lambda\}.$$

Calculemos todos los $FOLLOW$*'s de las variables:*

- *Inicializar:*

$$G(Q_0) = \emptyset, G(E') = \emptyset, G(X) = \emptyset, G(T) = \emptyset, G(T') = \emptyset,$$
$$F(Q_0) = \{\lambda\}, F(E') = \emptyset, F(X) = \emptyset, F(T) = \emptyset, F(T') = \emptyset.$$

- *Primer* **while***:*
  - *Variable* Q_0*:*
    - *Producción* $F \mapsto (Q_0)$*: Añadir* $FIRST())$ *a* $F(Q_0)$*:*

$$F(Q_0) := F(Q_0) \cup FIRST()) = \{\lambda,)\}.$$

[12] De hecho, nos interesa que no haya símbolos inaccesibles.

- *Variable T:*
  - *Producción $Q_0 \mapsto TE'$: Añadir $FIRST(E') \setminus \{\lambda\}$ a $F(T)$:*
  - *Producción $Q_0 \mapsto TE'$, nótese que $\lambda \in FIRST(E')$: Añadir $F(Q_0)$ a $F(T)$.*
  - *Producción $E' \mapsto +TE'$: Añadir $FIRST(E') \setminus \{\lambda\}$ a $F(T)$.*
  - *Producción $E' \mapsto +TE'$, nótese que $\lambda \in FIRST(E')$: Añadir $F(E')$ a $F(T)$.*

$$F(T) = (FIRST(E') \setminus \{\lambda\}) \cup F(Q_0) \cup (FIRST(E') \setminus \{\lambda\}) \cup F(E'),$$

$$F(T) = \{+, \lambda\} \cup \{)\} \cup \{+\} \cup \emptyset = \{+, \lambda,)\}.$$

- *Variable E':*
  - *Producción $Q_0 \mapsto TE'$, $\gamma = \lambda$, $\lambda \in FIRST(\gamma)$: Añadir $F(Q_0)$ a $F(E')$.*
  - *Producción $E' \mapsto +TE'$:No añade nada nuevo.*

$$F(E') = F(Q_0) = \{\lambda,)\}.$$

- *Variable X:*
  - *Producción $T \mapsto XT'$: Añadir $FIRST(T') \setminus \{\lambda\}$ a $F(X)$.*
  - *Producción $T' \mapsto *XT'$: Idem.*
  - *Producción $T \mapsto XT'$, como $\lambda \in FIRST(T')$: Añadir $F(T)$ a $F(X)$.*

$$F(X) = (FIRST(T') \setminus \{\lambda\}) \cup F(T) = \{*, \lambda, +,)\}.$$

- *Variable T':*
  - *Producción $T \mapsto XT'$: Añadir $F(T)$ a $F(T')$*
  - *Producción $T' \mapsto *XT'$: idem.*

$$F(T') = \{+, \lambda,)\}.$$

- *Segundo* **while***: Todos coinciden.*
- OUTPUT:

$$F(Q_0) = \{\lambda,)\}, F(E') = \{\lambda,)\}, F(X) = \{*, \lambda,), +\}, F(T) = \{+, \lambda,)\}, F(T') = \{+, \lambda,)\}.$$

6.6.2 Gramáticas $LL(k)$

Definición 153 (Gramáticas LL(k))**.** *Una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$ se dice de clase $LL(k)$ si verifica la siguiente propiedad: Dadas dos derivaciones, donde $\omega \in \Sigma^*, A \in V, \alpha, \beta, \gamma \in (V \cup \Sigma)^*$, del tipo siguiente:*

- $Q_0 \vdash_{lm} \omega A\gamma \vdash_{lm} \omega\alpha\gamma \vdash \omega x \in \Sigma^*$,
- $Q_0 \vdash_{lm} \omega A\gamma \vdash_{lm} \omega\beta\gamma \vdash \omega y \in \Sigma^*$,

Si $FIRST_k(x) = FIRST_k(y)$*, entonces* $\alpha = \beta$*.*

La idea es que si hacemos dos derivaciones a izquierda desde una variable de nuestra gramática, y si llegamos a dos formas terminales en las que los primeros k símbolos a partir de A de una forma terminal coinciden, entonces es que hemos tenido que hacer la misma derivación desde A.
La expresión formal es delicadamente retorcida, pero su sentido no se verá hasta que no procedamos a la construcción de la tabla de predicción y análisis sintáctico. Por ahora veamos unos poco ejemplos.

Ejemplo 23. *Un ejemplo de gramática* $LL(1)$ *es la dada mediante:* $Q_0 \mapsto aAQ_0 \mid b, A \mapsto a \mid bQ_0A$

Ejemplo 24. *La gramática* $\{Q_0 \mapsto \lambda \mid abA, A \mapsto Q_0aa \mid b\}$ *es una gramática* $LL(2)$

Ejemplo 25. *La gramática* $G_3 = (\{Q_0, A, B\}, \{0, 1, a, b\}, P_3, Q_0)$*, donde*

$$P_3 := \{Q_0 \mapsto A \mid B, A \mapsto aAb \mid 0, B \mapsto aBbb \mid 1\},$$

no es una gramática $LL(k)$ *para cualquier* k*. El lenguaje generado* $L(G_3)$ *es el lenguaje dado por*

$$L(G_3) := \{a^n 0 b^n \ : \ n \geq 0\} \cup \{a^n 1 b^{2n} \ : \ n \geq 0\}.$$

Proposición 154. *Una gramática* $G = (V, \Sigma, Q_0, P)$ *es* $LL(k)$ *si y solamente si se verifica la siguiente propiedad:*
Dadas dos producciones $A \mapsto \beta$ *y* $A \mapsto \gamma$ *tales que* A *es accesible y se tiene* $Q_0 \vdash_{lm} \omega A\alpha$*, con* $\omega \in \Sigma^*$ *y* $\alpha \in (V \cup \Sigma)^*$*, entonces*

$$FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha) = \emptyset.$$

Demostración. Siguiendo la propia definición. □

Ejemplo 26. *La gramática* $\{Q_0 \mapsto aQ_0 \mid a\}$ *no puede ser* $LL(1)$ *porque* $FIRST_1(aQ_0) = FIRST_1(a) = a$*.*

Como nos dicta la intuición, en las gramáticas $LL(k)$ tendremos que calcular $FIRST_k(\alpha)$, donde α es una forma no terminal. Estudiemos primero algunas propiedades que generalizan el caso $k = 1$.

Definición 155. *Sea* $L_1, L_2 \in \Sigma^*$*, dos lenguajes definimos:*

$$L_1 \oplus_k L_2 = \left\{ \omega : \exists x \in L_1, \ \exists y \in L_2 \begin{cases} |xy| \leq k \ y \ \omega = xy, \ o \\ w = FIRST_k(xy). \end{cases} \right\}$$

Nótese que esta definición puede ser extendida a de una forma recursiva a $L_1 \oplus_k \ldots L_n$ simplemente calculando primero $M_1 = L_1 \oplus_k L_2$ y repetir el argumento con $M_2 = M_1 \oplus_k L_3$ sucesivamente.

Lema 156. *Dada una gramática libre de contexto* G *y una forma sentencial* $\alpha\beta$ *se tiene que*

$$FIRST_k(\alpha\beta) = FIRST_k(\alpha) \oplus_k FIRST_k(\beta).$$

Demostración. Las palabras que se pueden derivar a partir de α forman un lenguaje, que podemos llamar L_1, lo mismo las palabras que se pueden derivar de β. Tenemos que si

$$\alpha\beta \vdash_{lm} xy \text{ entonces } \alpha \vdash_{lm} x, \qquad \beta \vdash_{lm} y.$$

Por lo tanto, si $\omega \in FIRST_k(L_1L_2) \iff \omega \in L_1 \oplus_k L_2$. □

Para calcular el $FIRST_k(\alpha)$ utilizaremos una generalización de que aparece en [Aho-Ullman, 72a].

INPUT: una gramática libre de contexto $G := (V, \Sigma, Q_0, P)$.

Definir $F_i(a) = a$ para todo símbolo del alfabeto y para todo $0 \leq i \leq k$.
Definir $F_0(A) = \{x \in \Sigma^k : A \mapsto x\alpha\}$ para todo símbolo del alfabeto y para todo $0 \leq i \leq k$.
Para $1 \leq i \leq k$ **y mientras** $F_{i-1}(A) \neq F_i(A)$ **para alguna variable** A **hacer**
    Para cada variable A hacer
        $F_i(A) = \{x \in \Sigma^k : A \mapsto Y_1 \dots Y_n \text{ y } x \in F_{i-1}(Y_1) \oplus_k \cdots \oplus_k F_{i-1}(Y_n)\}$.
        fin hacer
    fin hacer
OUTPUT: $F(A) \cap \Sigma \cup \{\lambda\}$, para cada $A \in V \cup \Sigma$.

6.6.3 Tabla de Análisis Sintáctico para Gramáticas $LL(1)$

Antes de comenzar, enumeraremos nuestras producciones, asignándole un número natural a cada una de ellas. Además, introduciremos un nuevo símbolo § que hará las funciones de fondo de la pila.
Construiremos una tabla

$$M : (V \cup \Sigma \cup \{\S\}) \times (\Sigma \cup \{\lambda\}) \longrightarrow \mathcal{P}(P) \cup \{\mathbf{pop}, \mathbf{accept}, \mathbf{error}\},$$

donde $\mathcal{P}(P)$ es el conjunto de todos los subconjuntos del conjunto de las producciones.

INPUT: Una gramática libre de contexto $G = (V, \Sigma, Q_0, P)$.
Establecemos una tabla M cuyas filas están indicadas por los elementos de $V \cup \Sigma \cup \{\S\}$ y cuyas columnas están indicadas por los elementos de $\Sigma \cup \{\lambda\}$.
Definiremos M del modo siguiente:

- Dada una producción (i) $A \mapsto \alpha$
  - Para cada $a \in FIRST(\alpha)$, $a \neq \lambda$, añadir i a la casilla $M(A, a)$.
  - Si $\lambda \in FIRST(\alpha)$ añadir i en todas las casillas $M(A, b)$ para cada $b \in FOLLOW(A)$.
- $M(a, a) =$**pop** para cada $a \in \Sigma$.

- $M(\S, \lambda) =$**accept**.
- En todos los demás casos escribir $M(X, i) =$**error**.

Nota 157. *Si bien es verdad que para simplificar la escritura de una tabla, conviene enumerar las producciones, se hubiera podido hacer de la misma manera incluyendo la producción en cada casilla. Como se ha comentado antes, se usará esta enumeración tanto para definir la tabla de análisis sintáctico (parsing) como los procesos que sigan en la Subsección siguiente.*
Como ejemplo, consideremos la gramática $G = (V, \Sigma, Q_0, P)$, donde las producciones son:

$$P := \{Q_0 \mapsto aAQ_0 \mid b, A \mapsto a \mid bQ_0A\}.$$

Enumeramos estas producciones del modo siguiente:

$$\begin{array}{ll} (1) & Q_0 \mapsto aAQ_0 \\ (2) & Q_0 \mapsto b \\ (3) & A \mapsto a \\ (4) & A \mapsto bQ_0A \end{array}$$

Ejemplo 27. *Veamos un ejemplo basado en la gramática descrita en el ejemplo anterior. La tabla de análisis sintáctico correspondiente será la siguiente:*

	a	b	λ
Q_0	1	2	**error**
A	3	4	**error**
a	**pop**	**error**	**error**
b	**error**	**pop**	**error**
$\S$	**error**	**error**	**accept**

Obsérvese que la gramática es $LL(1)$ porque la tabla tiene siempre una entrada (y no más).

Proposición 158. *Dada una gramática libre de contexto G, y dada $T(G)$ la tabla construida por el algoritmo anterior, entonces G es $LL(1)$ si y solamente si todos las casillas de la tabla $T(G)$ contienen exactamente una producción o una de las palabras seleccionadas (***pop, accept, error***).*

6.6.4 Parsing Gramáticas $LL(1)$

Vamos a construir un traductor con pila (PDT) asociado a cada gramática $LL(1)$. Además, ese traductor será determinista cuando la gramática sea $LL(1)$. El PDT se define con las reglas siguientes:

- El espacio de estados estará formado por un estado más relevante M que hace referencia a la tabla de análisis sintáctico tal y como se ha descrito en la Subsección anterior, un estado inicial q_0 que sólo aparece al inicializar el

proceso, un segundo estado **error** que indica que se ha producido un error en el análisis sintáctico y un último estado **accept** que indica que ha terminado la computación y hemos aceptado el input. Por tanto, $F = \{\mathbf{accept}\}$, $Q := \{q_0, M, \mathbf{error}, \mathbf{accept}\}$.

- El alfabeto de la cinta de input es el alfabeto Σ de la gramática dada.
- El alfabeto Δ de la cinta de output son los números naturales $\{1, \ldots, N\}$ de una enumeración de las producciones de la gramática original.
- El alfabeto de la pila Γ es la unión de los alfabetos V (conjunto de variables de la gramática), Σ (el alfabeto de la cinta de input) y el símbolo § que jugará el papel de fondo de la pila.

$$\Gamma = V \cup \Sigma \cup \{\S\}.$$

- La función de transición δ vendrá dada por

$$\delta : \{q_0, M\} \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\S\}) \longrightarrow \{M\} \times \Gamma^* \times \Delta.$$

Las transiciones quedarán definidas por las reglas siguientes:

a. Inicializar $\delta(q_0, \lambda, \S) = (M, Q_0, \lambda)$, donde Q_0 es la variable inicial de la gramática a analizar. Significa que comenzamos con una $\lambda-$transición para "cargar" la variable inicial en la pila, sin borrar ningún dígito de la cinta de input y sin añadir ninguna producción en la cinta de output.

b. Dados $X \in \Gamma$, $u \in \Sigma \cup \{\lambda\}$, supongamos que $M(X, u) \neq$ **error**, **pop**, **accept**. [13] Entonces, existe una producción (i) tal que $i \in M(X, u)$. Supongamos que esa producción es de la forma $X \mapsto \beta$. La transición es, definida mediante:

$$\delta(M, u, X) = (M, \beta^R, M(X, u)) = (M, \beta^R, i)$$

, donde i es el número de la producción correspondiente y $\beta^R \in \Gamma^*$ es el reverso de la lista de símbolos que aparecen a la derecha en esa producción.

Significa que hacemos `push`(`pop`($'lista'$), β^R) en la pila, y que añadimos i a la cinta de output, pasando a la siguiente celda vacía. No nos movemos en la cinta de entrada, entendiñdola como una Lambda-transición.

c. Dados $X \in \Gamma$, $u \in \Sigma$, supongamos que $M(X, u) =$**pop**, definimos $\delta(M, u, X) = (M, \lambda, \lambda)$ (indicando que hacemos **pop** en la pila, borramos una celda en la cinta de input y no escribimos nada en la cinta de output).

d. Dados $X \in \Gamma$, $u \in \Sigma \cup \{\lambda\}$, supongamos que $M(X, u) =$**error**, entonces, el proceso de análisis sintáctico cambia a estado **error** y se detienen las computaciones, aunque no se aceptan.

[13] Necesariamente X ha de ser una variable por la definición de la tabla M

e. Por último se define $\delta(M/\mathbf{accept}, \lambda, \S) = (M/\mathbf{accept}, \S, \lambda)$ (indicando que ha acabado la computación y aceptamos).

Teorema 159. *Existe un algoritmo que, en tiempo lineal $O(n)$ en el tamaño de la entrada, realiza el análisis sintáctico de los lenguajes dados por gramáticas $LL(1)$.*

Demostración. Es el algoritmo dado por el anterior traductor con pila. □

Ejemplo 28. *Retomemos el ejemplo de Subsecciones anteriores. Es la gramática $G = (V, \Sigma, Q_0, P)$, donde $V = \{Q_0, A\}$, $\Sigma = \{a, b\}$, y las producciones son:*

$$P := \{Q_0 \mapsto aAQ_0 \mid b, A \mapsto a \mid bQ_0A\}.$$

Enumeramos estas producciones del modo siguiente:

$$\begin{array}{ll} (1) & Q_0 \mapsto aAQ_0 \\ (2) & Q_0 \mapsto b \\ (3) & A \mapsto a \\ (4) & A \mapsto bQ_0A \end{array}$$

Tomamos la tabla M de análisis sintáctico:

M	a	b	λ
Q_0	1	2	**error**
A	3	4	**error**
a	**pop**	**error**	**error**
b	**error**	**pop**	**error**
§	**error**	**error**	**accept**

Ahora tenemos el autómata correspondiente que, esencialmente, está descrito en esa tabla. Así, podemos evaluar:

$\delta(M, -, -)$	a	b	λ
Q_0	$(Q_0Aa, 1)$	$(b, 2)$	**error**
A	$(a, 3)$	$(AQ_0b, 4)$	**error**
a	**pop**	**error**	**error**
b	**error**	**pop**	**error**
§	**error**	**error**	**accept**

Ejemplo 29 (Sistema de transición asociado al ejemplo)**.** *A modo de ejemplo, podemos considerar el sistema de transición asociado a este ejemplo y al PDT definido en el ejemplo anterior.*
Tomemos una palabra de input $\omega = abbab$ y construimos la configuración inicial en la palabra ω

$$I(\omega) := (M, abbab, \S, \lambda).$$

Las computaciones del autómata irán como sigue:

- *Inicializamos $I(\omega) \to (M, abbab, Q_0\S, \lambda)$.*

- *Leemos* $M(a, Q_0) = 1$*, luego* $\delta(M, a, Q_0) = (M, aAQ_0, 1)$*, con lo que tenemos la transición:*

$$(M, abbab, Q_0\S, \lambda) \to (M, abbab, aAQ_0\S, 1).$$

- *Leemos* (M, a, a) *y* $M(a, a) =$**pop***, así que borramos*

$$(M, abbab, aAQ_0\S, 1) \to (M, bbab, AQ_0\S, 1).$$

- *Leemos* (M, b, A) *y* $M(b, A) = 4$*, luego aplicamos la transición* $\delta(M, b, A) = (M, bQ_0A, 4)$ *y tenemos la transición:*

$$(M, bbab, AQ_0\S, 1) \to (M, bbab, bQ_0AQ_0\S, 14).$$

- *Acudiendo a la tabla, iremos obteniendo las transiciones siguientes:*

$$(M, bbab, bQ_0AQ_0\S, 14) \to (M, bab, Q_0AQ_0\S, 14) \to (M, bab, bAQ_0\S, 142) \to$$
$$\to (M, ab, AQ_0\S, 142) \to (M, ab, aQ_0\S, 1423) \to (M, b, Q_0\S, 1423) \to$$
$$\to (M, b, b\S, 14232) \to (M, \lambda, \S, 14232) \to (\textbf{accept}, \lambda, \S, 14232).$$

  Dejamos que el alumno verifique que las transiciones se comportan como se ha indicado.

Nótese cómo en la cinta de output se han ido escribiendo los números de las producciones que, aplicados con estrategia leftmost, componen en árbol de derivación de la palabra aceptada.

6.7 Cuestiones y Problemas

6.7.1 Cuestiones

Cuestión 27. *Compara la siguiente versión iterativa del algoritmo propuesto para el cálculo de FIRST:*

Definiremos los conjuntos $F_i(X)$ *para valores crecientes de* i *del modo siguiente:*

- **if** $X \in \Sigma$ **then** $F_i(X) = \{X\}$*, para todo* i*.*
- $F_0(X) := \{x \in \Sigma \cup \{\lambda\} \ : \ \exists X \mapsto x\alpha \in P\}$*.*
- $F_i(X) := \{x \ : \ \exists X \mapsto Y_1 \cdots Y_n \in P \text{ y } x \in F_{i-1}(Y_1) \oplus F_{i-1}(Y_2) \oplus \cdots \oplus F_{i-1}(Y_n)\} \cup F_{i-1}(X)$*.*

if $F_i(X) = F_{i+1}(X)$*, para todo* X **then** OUTPUT

$$\{F_i(X) \ : \ X \in V \cup \Sigma\}.$$

Cuestión 28. *Consideremos el siguiente SDTS. Denotaremos a las variables utilizando $< x >$:*

$$
\begin{array}{rcl}
< exp > & \mapsto & sums < exp >^1 \ \ with \ \ < var > \mapsto < exp >^2 \ to < exp >^2 \ do < statement >, \\
 & & begin \ \ < var > \mapsto < exp > \\
 & & if < var > \leq < exp >^1 \ then; \\
 & & begin < statement > \\
 & & < var > \mapsto < var > +1; \\
 & & endend \\
< var > & \mapsto & < id >, < id > \\
< exp > & \mapsto & < id >, < id > \\
< id > & \mapsto & a < id >, a < id > \\
< id > & \mapsto & b < id >, b < id > \\
< id > & \mapsto & a, a \\
< id > & \mapsto & b, b
\end{array}
$$

Razonar porque no es un sistema de traducción directa. ¿ Cual deberia ser la traducción para la siguiente palabra:

$$for \ a \mapsto b \ to \ aa \ do \ baa \mapsto bba.$$

Cuestión 29. *Deducir como se escriben las transiciones en un traductor con pila.*

Cuestión 30. *En el algoritmo CYK, para construir los posibles árboles de derivación, hay que calcular varios valores $t_{i,j}$. Discutir que representan estos valores y por qué hay que calcular los valores $t_{1,n}$, donde n es la longitud de la palabra.*

Cuestión 31. *Dar un ejemplo de una gramática libre de contexto ambigua, construir un traductor con pila y explicar porque no es aconsejable su utilización en lenguajes de programación.*

Cuestión 32. *Suponed que en una gramática libre de contexto se tiene la siguiente producción $A \mapsto AA$. Discutir si la gramática es ambigua.*

Cuestión 33. *Dada la siguiente gramática:*

$$Q_0 \mapsto 0A0|1B1B, \ A \mapsto 0BQ_0|1|\lambda, \ B \mapsto 0|A|\lambda.$$

Calcular $FIRST(AB)$, $FIRST(AA)$, $FOLLOW(1B)$.

Cuestión 34. *Demostrar que la siguiente gramática es LL(1):*

$$Q_0 \mapsto aAQ_0|b, \ A \mapsto a|bQ_0A.$$

6.7.2 Problemas

Problema 79. *Consideremos el siguiente SDTS. Denotaremos a las variables utilizando $< x >$:*

$$\begin{array}{rcl}
< exp > & \mapsto & sums < exp >^1 \ \ with \ \ < var > \mapsto < exp >^2 \ to < exp >^3, \\
 & & begin\ local\ t; \\
 & & t = 0; \\
 & & for < var > \mapsto < exp >^2 \ to < exp >^3 \ do: \\
 & & t \mapsto t + < exp >^1;\ result\ t; \\
 & & end \\
< var > & \mapsto & < id >, < id > \\
< exp > & \mapsto & < id >, < id > \\
< id > & \mapsto & a < id >, a < id > \\
< id > & \mapsto & b < id >, b < id > \\
< id > & \mapsto & a, a \\
< id > & \mapsto & b, b
\end{array}$$

Dar la traducción para las siguientes palabras:

a. *sum aa witha $\mapsto$ b to bb.*

b. *sum sum a withaa $\mapsto$ aaa toaaaawithb $\mapsto$ bb tobbb.*

Problema 80. *Sea L_1, L_2 lenguajes libres de contexto con intersección vacía. Construir un SDTS tal que traduzca*

$$\{(x, y) | si\ x \in\ L_1\ entonces\ y = 0\ y\ si\ x \in L_2\ entonces\ y = 1\}$$

Problema 81. *Sea el siguiente traductor con pila,*

$$(\{q, q_1\}, \{a, +, *\}, \{*, +, E\}, \{a, *, +\}, \delta, q, E)$$

donde δ está definido por las siguientes relaciones:

$$\begin{array}{rcl}
\delta(q, a, E) & = & \{(q, e, a)\} \\
\delta(q, e, E) & = & \{(q_1, e, e)\} \\
\delta(q, +, E) & = & \{(q, EE+, e)\} \\
\delta(q, *, E) & = & \{(q, EE*, e)\} \\
\delta(q, e, *) & = & \{(q, e, *), (q_1, e, *)\} \\
\delta(q, e, +) & = & \{(q_1, e, +), (q, e, +)\} \\
\delta(q_1, e, *) & = & \{(q_1, e, e)\} \\
\delta(q_1, e, E) & = & \{(q_1, e, e)\} \\
\delta(q_1, e, +) & = & \{q_1, e, e)\}
\end{array}$$

Traducir la siguiente palabra $w = + * aaa$.
Definir un autómata para el lenguaje de partida.

Problema 82. *Construir un SDTS tal que*

$$\{(x, y)|x \in \{a, b\}^* \ y \ y = c^i\}$$

donde i es el valor absoluto del numero de a menos el numero de b de x.

Problema 83. *Diseñar un traductor con cola que realice la siguiente función. Dado el lenguaje sobe el alfabeto $\Sigma = \{a, +, -\}$*
definido por la siguiente expresión regular

$$(+^* -^* a)^*$$

elimine todos los operadores innecesarios. Como ejemplo, la siguiente palabra $w = ++a++ + -a--a$ debería ser traducida a $w' = a - a + a$.

Problema 84. *Sea R un lenguaje regular, construir un traductor con pila M, tal que para un lenguaje libre de contexto L, la traducción*
de L sea L-R.

Problema 85. *Dada la siguiente gramática,*

$$Q_0 \mapsto 0Q_0Q_00|A, \ A \mapsto \lambda|1Q_01|,$$

Aplicar el algoritmo CYK para la palabra $w = 0110$.

Problema 86. *Deducir el número de operaciones del algoritmo CYK y la capacidad de memoria necesaria utilizada.*

Problema 87. *Construir la tabla de análisis sintáctico y el traductor con pila para la siguiente gramática LL(1):*

$$Q_0 \mapsto AB|BB, \ A \mapsto 0A|1, \ B \mapsto 2B12|3.$$

Problema 88. *Construir la tabla de análisis sintáctico y el traductor con pila para la siguiente gramática LL(1):*

$$Q_0 \mapsto BAB|CBC, \ A \mapsto 0B|1C, \ B \ mapsto 1|0BB, \ C \mapsto 0C|1Q_0.$$

Problema 89. *Demostrar que la siguiente gramática es LL(1),*

$$Q_0 \mapsto TE', \ E' \mapsto +TE'|\lambda, \ T \mapsto FT', \ T' \mapsto *FT'|T'\lambda, \ F \mapsto (Q_0)|a$$

*Calcular el traductor con pila y construir el árbol de derivación de la palabra w=a+a+a*a.*

Problema 90. *Construir la tabla CYK y la tabla LL(1) para la gramática siguiente (escrita en BNF y con las numeraciones indicadas para las producciones):*

$$\begin{array}{rcrr}
\langle exp\rangle & := & \langle term\rangle\langle termTail\rangle & (1) \\
\langle termTail\rangle & := & \langle addop\rangle\langle term\rangle\langle termTail\rangle \mid \lambda & (2 \mid 3) \\
\langle term\rangle & := & \langle factor\rangle\langle factorTail\rangle & (4) \\
\langle facorTail\rangle & := & \langle multop\rangle\langle factor\rangle\langle factorTail\rangle \mid \lambda & (5 \mid 6) \\
\langle factor\rangle & := & (\langle exp\rangle) \mid NUM \mid ID & (7 \mid 8 \mid 9) \\
\langle addop\rangle & := & + \mid - & (10 \mid 11) \\
\langle multop\rangle & := & * \mid / & (12 \mid 13)
\end{array}$$

Problema 91. *Consideremos la siguiente gramática libre de contexto. Denotaremos a las variables utilizando* $< x >$ *y* $\Sigma = \{sums, with, \mapsto, to, do, a, b\}$

$$\begin{aligned}
< exp > &:= sums < exp > \ with \ < var > \mapsto < exp > to < exp > \\
< exp > &:= sums \ b \ with \ < var > \mapsto < exp > to < exp > \\
< exp > &:= < id > \\
< var > &:= < id > \\
< id > &:= a < id > \\
< id > &:= b < id > \\
< id > &:= a \\
< id > &:= b
\end{aligned}$$

Hallar una derivación mas a la izquierda y otra más a la derecha de la siguiente palabra utilizando el algoritmo CYK:

$$sums \ b \ with \ a \mapsto b \ to \ aa$$

Discutir si la gramática es ambigua. Eliminar la segunda producción y construir la tabla de análisis sintáctico. Discutir si la gramática es LL(1).

Problema 92. *Dada la siguiente gramática:*

$$\begin{aligned}
< Orden > &:= < IntroducirElemento >< Orden > \\
< Orden > &:= < EliminarElemento >< Orden > \\
< Orden > &:= \lambda \\
< IntroducirElemento > &:= push < id >; \\
< EliminarElemento > &:= remove < id >; \\
< id > &:= b < id > \\
< id > &:= a \\
< id > &:= b \\
< id > &:= \lambda
\end{aligned}$$

Discutir si es una gramática propia. Aplicar el algoritmo CYK a la palabra

$$push \ baa; push \ baaa; remove \ baaa; push \ b; remove \ baa;$$

Dar una derivación a la izquierda.

Problema 93. *Hallar* $FIRST(< id >< id >< Orden >)$ *y* $FOLLOW(< id >< $

$id >< EliminarElemento >$):

$$
\begin{aligned}
< Orden > &:= < IntroducirElemento >< Orden > \\
< Orden > &:= < EliminarElemento >< Orden > \\
< Orden > &:= \lambda \\
< IntroducirElemento > &:= push < id >; \\
< EliminarElemento > &:= remove < id >; \\
< id > &:= b < id > \\
< id > &:= a \\
< id > &:= b \\
< id > &:= \lambda
\end{aligned}
$$

Appendix A

Sucinta Introducción al Lenguaje de la Teoría Intuitiva de Conjuntos

Contents

A.1 Introducción . . . 150
A.2 Conjuntos. Pertenencia. . . . 150
A.2.1 Algunas observaciones preliminares. . . . 151
A.3 Inclusión de conjuntos. Subconjuntos, operaciones elementales. . . . 151
A.3.1 El retículo $\mathcal{P}(X)$. . . . 153
A.3.1.1 Propiedades de la Unión. . . . 153
A.3.1.2 Propiedades de la Intersección. . . . 153
A.3.1.3 Propiedades Distributivas. . . . 153
A.3.2 Leyes de Morgan. . . . 153
A.3.3 Generalizaciones de Unión e Intersección. . . . 154
A.3.3.1 Un número finito de uniones e intersecciones. . . . 154
A.3.3.2 Unión e Intersección de familias cualesquiera de subconjuntos. . . . 154
A.3.4 Conjuntos y Subconjuntos: Grafos No orientados. . . . 154
A.4 Producto Cartesiano (`list`). Correspondencias y Relaciones. . . . 155
A.4.1 Correspondencias. . . . 156
A.4.2 Relaciones. . . . 157
A.4.2.1 Relaciones de Orden. . . . 158
A.4.2.2 Relaciones de Equivalencia. . . . 159
A.4.3 Clasificando y Etiquetando elementos: Conjunto Cociente. . . . 160
A.5 Aplicaciones. Cardinales. . . . 161
A.5.1 Determinismo/Indeterminismo. . . . 162
A.5.2 Aplicaciones Biyectivas. Cardinales. . . . 164

A.1 Introducción

A finales del siglo XIX, G. Cantor introduce la Teoría de Conjuntos. Su propósito inicial es el modesto propósito de fundamentar matemáticamente el proceso de "contar". Eso sí, no se trataba solamente de contar conjuntos finitos sino también infinitos, observando, por ejemplo, que hay diversos infinitos posibles ($\aleph_0, 2^{\aleph_0}$ o $\aleph_1$, por ejemplo). Más allá del propósito inicial de Cantor, la Teoría de Conjuntos se transformó en un instrumento útil para las Matemáticas, como un lenguaje formal sobre el que escribir adecuadamente afirmaciones, razonamientos, definiciones, etc...
Lo que aquí se pretende no es una introducción formal de la Teoría de Conjuntos. Para ello sería necesario hacer una presentación de los formalismos de Zermelo–Frænkel o de Gödel–Bernays, lo cual nos llevaría un tiempo excesivo y sería de todo punto infructuoso e ineficaz. Al contrario, pretendemos solamente unos rudimentos de lenguaje que nos serán de utilidad durante el curso, un "apaño", para poder utilizar confortablemente el lenguaje si tener que produndizar en honduras innecesarias para la Ingeniería Informática. El recurso, también usado corrientemente en Matemáticas, es acudir a la Teoría Intuitiva de Conjuntos tal y como la concibe Haussdorff. Éste es el propósito de estas pocas páginas.

A.2 Conjuntos. Pertenencia.

Comencemos considerando los conjuntos como conglomerados de objetos. Estos objetos pasarán a denominarse *elementos* y diremos que *pertenecen a un conjunto.* Para los objetos que no están en un conjunto dado, diremos que *no pertenecen al conjunto* (o no son elementos suyos).
Como regla general (aunque con excepciones, que se indicarán en cada caso), los conjuntos se denotan con letras mayúsculas:

$$A, B, C, \ldots, X, Y, Z, A_1, A_2, ...,$$

mientras que los elementos se suelen denotar con letras minúsculas:

$$a, b, c, d, \ldots, x, y, z, a_1, a_2, ...$$

El símbolo que denota pertenencia es $\in$ y escribiremos

$$x \in A, \ \ ; x \notin B,$$

para indicar "el elemento x pertenece al conjunto A" y "el elemento x no pertenece al conjunto B", respectivamente.
Hay muchas formas para definir un conjunto. Los símbolos que denotan conjunto son las dos llaves { y } y su descripción es lo que se escriba en medio de las llaves.

- *Por extensión:* La descripción de todos los elementos, uno tras otro, como, por ejemplo:
$$X := \{0, 2, 4, 6, 8\}.$$

- *Por una Propiedad que se satisface:* Suele tomar la forma

$$X := \{x \ : \ P(x)\},$$

donde P es una propiedad (una fórmula) que actúa sobre la variable x. Por ejemplo, el conjunto anterior puede describirse mediante:

$$X := \{x \ : \ [x \in \mathbb{N}] \wedge [0 \leq x \leq 9] \wedge [2 \mid x]\},$$

donde hemos usado una serie de propiedades como $[x \in \mathbb{N}]$ (es un número natural), $[0 \leq x \leq 9]$ (entre 0 y 9), $[2 \mid s]$ (y es par). Todas ellas aparecen ligadas mediante la conectiva $\wedge$ (conjunción). Sobre la forma y requisitos de las propiedades no introduciremos grandes discusiones.

A.2.1 Algunas observaciones preliminares.

- Existe un único conjunto que no tiene ningún elemento. Es el llamado **conjunto vacío** y lo denotaremos por $\emptyset$. La propiedad que verifica se expresa (usando cuantificadores) mediante:

$$\neg\left(\exists x, x \in \emptyset\right),$$

o también mediante la fórmula

$$\forall x, x \notin \emptyset.$$

- La **Estructura de Datos** relacionada con la noción de conjunto es el tipo `set`, ya visto es el curso correspondiente y que no hace sino reflejar la noción global.

A.3 Inclusión de conjuntos. Subconjuntos, operaciones elementales.

Se dice que un conjunto X está incluido (o contenido) en otro conjunto Y si *todos los elementos de X son también elementos de Y*. También se dice que X es subconjunto de Y en ese caso. Se usa el símbolo $\subseteq$ para indicar inclusión y la propiedad se "define" mediante:

$$X \subseteq Y := \left[\forall x, x \in X \Longrightarrow x \in Y\right].$$

- Nótese la identificación entre la inclusión $\subseteq$ y la implicación $\Longrightarrow$ (o $\longrightarrow$, en la forma más convencional de la Lógica).

- Obviamente, a través de esa identificación, el conjunto vacío está contenido en cualquier conjunto. Es decir,

$$\emptyset \subseteq X,$$

para cualquier conjunto X.

- Dos conjuntos se consideran iguales si poseen los mismos elementos. En términos formales:

$$(A = B) \Leftrightarrow ((A \subseteq B) \wedge (B \subseteq A)) .$$

  Lo que también puede escribirse con elementos mediante:

$$(A = B) \Leftrightarrow \forall x, ((x \in A) \Longleftrightarrow (x \in B)) .$$

- La familia de todos los subconjuntos de un conjunto X dado se denomina la *clase de partes de* X y se suele denotar mediante $\mathcal{P}(X)$.

Ejemplo 30. *Es fácil, por ejemplo, mostrar la siguiente igualdad que describe las partes del conjunto* $X := \{0, 1, 2\}$*:*

$$\mathcal{P}(\{0, 1, 2\}) = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 1\}\} .$$

No resulta tan fácil probar que la clase $\mathcal{P}(\mathbb{N})$ *es justamente el intervalo* $[0, 1]$ *de la recta real* $\mathbb{R}$*. Lo dejamos para más tarde (en forma puramente esquemática).*

Las conectivas lógicas del cálculo proposicional, permiten definir operaciones entre subconjuntos de un conjunto dado. Supongamos que tenemos un conjunto X dado y sean $A, B \in \mathcal{P}(X)$ dos de sus subconjuntos. Definimos:

- **Unión:**
$$A \cup B := \{x \in X \ : \ (x \in A) \vee (x \in B)\}.$$

- **Intersecćon:**
$$A \cap B := \{x \in X \ : \ (x \in A) \wedge (x \in B)\}.$$

- **Complementario:**
$$A^c := \{x \in X \ : \ x \notin A\}.$$

Obsérvese que $\emptyset^c = X$ y que $(A^c)^c = A$ para cualquier $A \in \mathcal{P}(X)$.
Adicionalmente, podemos reencontrar la diferencia entre conjuntos y la traslación del EXCLUSIVE OR (denotado por XOR en Electrónica Digital) o por $\oplus$ (en Teoría de Números, hablando de restos enteros módulo 2, i.e. $\mathbb{Z}/2\mathbb{Z}$; aunque, en este caso se suele denotar simplemente mediante $+$).

- **Diferencia:**
$$A \setminus B := \{x \in X \ : \ (x \in A) \wedge (x \notin B)\}.$$

- **Diferencia Simétrica:**

$$A \Delta B := \{x \in X \ : \ (x \in A) \oplus (x \in B)\}.$$

Las relaciones evidentes con estas definiciones se resumen en las siguientes fórmulas:

$$A \setminus B := A \cap B^c, \ \ A \Delta B = (A \cup B) \setminus (A \cap B) = (A \setminus B) \cup (B \setminus A).$$

A.3.1 El retículo $\mathcal{P}(X)$.

Sería excesivo e innecesario expresar aquí con propiedad las nociones involucradas, pero dejemos constancia de la propiedades básicas de estas operaciones:

A.3.1.1 Propiedades de la Unión.

Sean A, B, C subconjuntos de un conjunto X.

- **Idempotencia:** $A \cup A = A$, $\forall A \in \mathcal{P}(X)$.
- **Asociativa:** $A \cup (B \cup C) = (A \cup B) \cup C$, $\forall A, B, C \in \mathcal{P}(X)$.
- **Conmutativa:** $A \cup B = B \cup A$, $\forall A, B \in \mathcal{P}(X)$.
- **Existe Elemento Neutro:** El conjunto vacío $\emptyset$ es el elemento neutro para la unión:
$$A \cup \emptyset = \emptyset \cup A = A, \quad \forall A \in \mathcal{P}(X).$$

A.3.1.2 Propiedades de la Intersección.

Sean A, B, C subconjuntos de un conjunto X.

- **Idempotencia:** $A \cap A = A$, $\forall A \in \mathcal{P}(X)$.
- **Asociativa:** $A \cap (B \cap C) = (A \cup B) \cup C$, $\forall A, B, C \in \mathcal{P}(X)$.
- **Conmutativa:** $A \cap B = B \cap A$, $\forall A, B \in \mathcal{P}(X)$.
- **Existe Elemento Neutro:** El conjunto total X es el elemento neutro para la intersección:
$$A \cap X = X \cap A = A, \quad \forall A \in \mathcal{P}(X).$$

A.3.1.3 Propiedades Distributivas.

Sean A, B, C subconjuntos de un conjunto X.

- $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, $\forall A, B, C \in \mathcal{P}(X)$.
- $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$, $\forall A, B, C \in \mathcal{P}(X)$.
- $A \cap (B \Delta C) = (A \cap B) \Delta (A \cap C)$, $\forall A, B, C \in \mathcal{P}(X)$.

A.3.2 Leyes de Morgan.

Por ser completos con los clásicos, dejemos constancia de las Leyes de Morgan. Sean A, B subconjuntos de un conjunto X.

$$(A \cap B)^c = (A^c \cup B^c), \quad (A \cup B)^c = (A^c \cap B^c).$$

A.3.3 Generalizaciones de Unión e Intersección.

Tras todas estas propiedades, dejemos las definiciones y generalizaciones de la unión e intersección en el caso de varios (o muchos) subconjuntos de un conjunto dado. Nótese la identificación entre $\cup$, $\vee$ y el cuantificador existencial $\exists$ (y, del mismo modo, la identificación entre $\cap$, $\wedge$ y el cuantificador universal $\forall$.

A.3.3.1 Un número finito de uniones e intersecciones.

Dados $A_1, \ldots, A_n$ unos subconjuntos de un conjunto X. Definimos:

$$\bigcup_{i=1}^{n} A_i := A_1 \cup A_2 \cup \cdots \cup A_n = \{x \in X \ : \ \exists i,\ 1 \leq i \leq n,\ x \in A_i\}.$$

$$\bigcap_{i=1}^{n} A_i := A_1 \cap A_2 \cap \cdots \cap A_n = \{x \in X \ : \ \forall i,\ 1 \leq i \leq n,\ x \in A_i\}.$$

A.3.3.2 Unión e Intersección de familias cualesquiera de subconjuntos.

Supongamos $\{A_i \ : \ i \in I\}$ es una familia de subconjuntos de un conjunto X. Definimos:

$$\bigcup_{i \in I} A_i := \{x \in X \ : \ \exists i \in I,\ x \in A_i\}.$$

$$\bigcap_{i \in I} A_i := \{x \in X \ : \ \forall i \in I,\ x \in A_i\}.$$

En ocasiones nos veremos obligados a acudir a uniones e intersecciones de un número finito o de un número infinito de conjuntos.

A.3.4 Conjuntos y Subconjuntos: Grafos No orientados.

Recordemos que un grafo no orientado (o simplemente un grafo) es una lista $\mathcal{G} := (V, E)$ formada por dos objetos:

- **Vértices:** son los elementos del conjunto V (que usualmente se toma finito[1]) aunque podremos encontrar "grafos" con un conjunto "infinito" de vértices.
- **Aristas:** Es un conjunto de subconjuntos de V, es decir, $E \subseteq \mathcal{P}(V)$ con la salvedad siguiente: los elementos $A \in E$ (que, recordemos, son subconjuntos de V) son no vacíos y tienen a lo sumo dos elementos distintos.

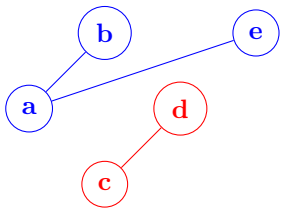
Ejemplo 31. *Un sencillo ejemplo sería:*

- **Vértices:** $V := \{a, b, c, d, e\}$
- **Aristas:**

$$E := \{\{a, b\}, \{a, e\}, \{c, d\}\} \subseteq \mathcal{P}(V).$$

[1] Aceptemos esta disgresión sin haber definido finitud

Al ser no orientado la matriz de adyacencia es simétrica y las componentes conexas son, también, subconjuntos de V, aunque de mayor cardinal. Gráficamente:



Nótese que podríamos haber aceptado aristas que van desde un nodo a sí mismo (tipo $\{a\}$ o $\{e\}$, por ejemplo) pero que el orden en que son descritos los elementos de una arista no es relevante: por eso hablamos de grafos no orientados.

A.4 Producto Cartesiano (`list`). Correspondencias y Relaciones.

Si en los grafos no orientados considerábamos aristas descritas de forma $\{a, b\}$ y el orden no interviene ($\{a, b\} = \{b, a\}$) ahora nos interesa destacar el papel jugado por el orden, hablamos de pares ordenados (a, b) y se corresponde al tipo de datos `list`. Así, por ejemplo, $(a, b) = (b, a)$ si y solamente si $a = b$. Una manera de representar las listas mediante conjuntos podría ser escribiendo (a, b) como abreviatura de $\{\{a\}, \{a, b\}\}$. Pero nos quedaremos con la intuición del tipo de datos `list`.
Dados dos conjuntos A y B definimos el producto cartesiano de A y B como el conjunto de las listas de longitud 2 en las que el primer elemento está en el conjunto A y el segundo en B. Formalmente,

$$A \times B := \{(a, b) \ : \ a \in A, b \in B\}.$$

Pero podemos considerar listas de mayor longitud: dados $A_1, \ldots, A_n$ definimos el producto cartesiano $\prod_{i=1}^{n} A_i$ como las listas de longitud n, en las que la coordenada $i-$ésima está en el conjunto A_i.

$$\prod_{i=1}^{n} A_i := \{(x_1, \ldots, x_n) \ : \ x_i \in A_i, \ 1 \leq i \leq n\}.$$

En ocasiones, se hacen productos cartesianos de familias no necesariamente finitas $\{A_i \ : \ i \in I\}$ (como las sucesiones, con $I = \mathbb{N}$) y tenemos el conjunto:

$$\prod_{i \in I} A_i := \{(x_i \ : \ i \in I) \ : \ x_i \in A_i, \ \forall i \in I\}.$$

En otras ocasiones se hace el producto cartesiano de un conjunto consigo mismo, mediante las siguientes reglas obvias:

$$A^1 = A, \quad A^2 := A \times A, \quad A^n := A^{n-1} \times A = \prod_{i=1}^{n} A.$$

Algunos casos extremos de las potencias puedebn ser los siguientes:

- **Caso $n = 0$:** Para cualquier conjunto A se define A^0 como el conjunto formado por un único elemento, que es el mismo independientemente de A, y se conoce con la palabra vacía y se denota por λ. No se debe confundir $A^0 := \{\lambda\}$ con el conjunto vacío $\emptyset$.

- **Caso $I = \mathbb{N}$:** Se trata de las sucesiones (infinitas numerables) cuyas coordenadas viven en A. Se denota por $A^{\mathbb{N}}$. Los alumnos han visto, en el caso $A = \mathbb{R}$ el conjunto de todas las sucesiones de números reales (que se denota mediante $A^{\mathbb{N}}$).

Nota 160 (Palabras sobre un Alfabeto)**.** *El conjunto de las palabras con alfabeto un conjunto A jugará un papel en este curso, se denota por A^* y se define mediante*

$$A^* := \bigcup_{n\in\mathbb{N}} A^n.$$

Volveremos con la noción más adelante

A.4.1 Correspondencias.

Una correspondencia entre un conjunto A y otro conjunto B es un subconjunto R del producto cartesiano $A \times B$. En esencia es un grafo bipartito que hace interactuar los elementos de A con elementos de B. Los elementos que interactúan entre sí son aquellos indicados por los pares que están en R. En ocasiones se escribirán una notación funcional de la forma $R : A \longrightarrow B$, aunque poniendo gran cuidado porque no siempre son funciones.

Ejemplo 32. *Tomando $A = B = \mathbb{R}$, podemos definir la relación $R_1 \subseteq \mathbb{R}^2$ mediante:*

$$R_1 := \{(x, y) \in \mathbb{R}^2 \ : \ x = y^2\}.$$

Estaremos relacionando los número reales con sus raíces cuadradas. Obsérvese que los elementos x tales que $x < 0$ no están relacionados con ningún número y (no tienen raíz cuadrada real). El 0 se relaciona con un único número (su única raíz cuadrada) y los número reales positivos se relacionan con sus dos raices cuadradas.

Ejemplo 33. *Tomando los mismos conjuntos $A = B = \mathbb{R}$, podemos definir la relación $R_2 \subseteq \mathbb{R}^2$ distinta de la anterior:*

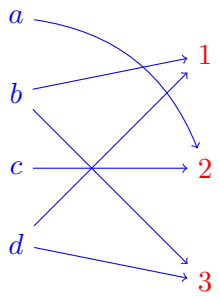
$$R_1 := \{(x, y) \in \mathbb{R}^2 \ : \ x^2 = y\}.$$

En este caso tenemos una función que relaciona cualquier x en $\mathbb{R}$ con su cuadrado.

Ejemplo 34. *Un grafo bipartito podría ser, por ejemplo, $A := \{a, b, c, d\}$, $B := \{1, 2, 3\}$ y una relación como $R \subseteq A \times B$:*

$$R := \{(a, 2), (b, 1), (b, 3), (c, 2), (d, 1), (d, 3)\},$$

cuyo grafo sería:



Nota 161. *En ocasiones abusaremos de la notación, escribiendo $R(x) = y$ o xRy, para indicar que los elementos $x \in A$ e $y \in B$ están en correspondencia a través de $R \subseteq A \times B$.*

A.4.2 Relaciones.

Las relaciones son correspondencia $R \subseteq A \times A$, es decir, aquellas correspondencias donde el conjunto de primeras coordenadas es el mismo que el conjunto de las segundas coordenadas.

Nota 162 (Una Relación no es sino un grafo orientado.)**.** *Aunque, por hábito, se suele pensar en que los grafos orientados son relaciones sobre conjuntos finitos, pero admitiremos grafos con un conjunto infinito de vértices.*

Pongamos algunos ejemplos sencillos:

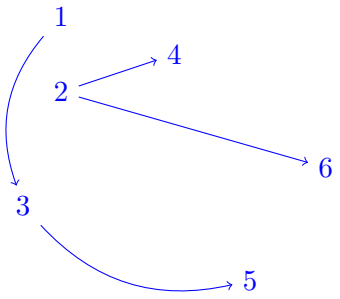
Ejemplo 35 (Un ejemplo al uso)**.** *Consideremos el grafo $\mathcal{G} := (V, E)$ donde V es el conjunto de vértices dado por:*

$$V := \{1, 2, 3, 4, 5, 6\},$$

y $E \subseteq V \times V$ es el conjunto de aristas orientadas siguiente:

$$E := \{(1, 3), (3, 5), (2, 4), (2, 6)\}.$$

Gráficamente tendremos



Ejemplo 36 (La circunferencia unidad)**.** *Es un grafo infinito cuyos vértices son los números reales $V = \mathbb{R}$ y cuyas aristas son dadas mediante:*

$$E := \{(x, y) \in \mathbb{R}^2 \ : \ x - y \in \mathbb{Z}\}.$$

No lo dibujaremos (tenemos demasiados vértices y demasiadas aristas) pero las componentes conexas están identicadas con los puntos de la circunferencia unidad

$$S^1 := \{(x, y) \in \mathbb{R}^2 \ : \ x^2 + y^2 - 1 = 0\}.$$

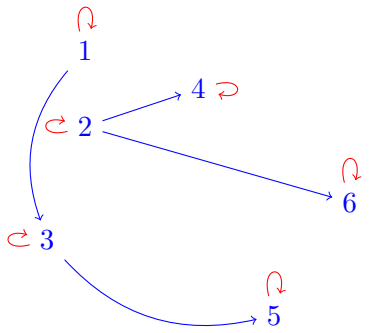
Algunos tipos de relaciones son más relevantes que otras por sus consecuencias. Destaquemos dos clases:

A.4.2.1 Relaciones de Orden.

Son aquellas relaciones $R \subseteq V \times V$, que verifican las propiedades siguientes:

- *Reflexiva:* $\forall x \in V, \ \ (x, x) \in R$. La relación descrita en el Ejemplo 35 anterior no verifica esta propiedad. Para verificarla, se necesitaría que también fueran aristas las siguientes:

$$\{(1,1), (2,2), (3,3), (4,4), (5,5), (6,6)\} \subseteq E.$$



  En cambio el ejemplo de la circunferencia verifica la propiedad reflexiva.

- *Antisimétrica:* Que se expresa mediante:

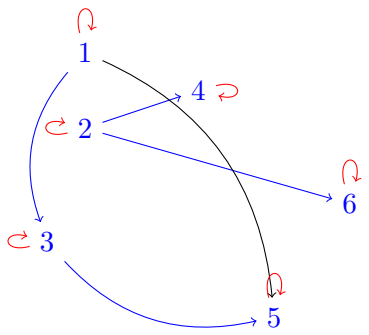
$$\forall x, y \in V, \ \ ((x, y) \in R) \wedge ((y, x) \in R) \Rightarrow (x = y)\,.$$

  La relación descrita en el Ejemplo 35 sí verifica la propiedad antisimétrica porque no se da ningún caso que verifique simultáneamente las dos hipótesis. Incluso si añadimos todas las refelxivas todo funciona bien. En el ejemplo de la circunferencia, sin embargo, no se da la antisimétrica: por ejemplo 1 y 0 verifican que $(1, 0) \in R$, $(0, 1) \in R$ y, sin embargo, $1 \neq 0$.

- *Transitiva:* Que se expresa mediante:

$$\forall x, y, z \in V, \ \ ((x, y) \in R) \wedge ((y, z) \in R) \Rightarrow ((x, z) \in R)\,.$$

  La relación descrita en el Ejemplo 35 no verifica la transitiva. Tenemos que $(1, 3) \in E$ y $(3, 5) \in E$, pero $(1, 5) \notin E$. Tendríamos que añadirla para tener un grafo como:



Este último grafo ya nos dará una relación de orden. En el ejemplo de la circunferencia, sin embargo, se da la Transitiva, aunque no es relación de orden por no satisfacerse la anti–siméttrica.

A.4.2.2 Relaciones de Equivalencia.

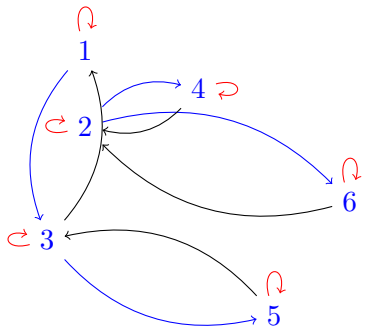
Son aquellas relaciones $R \subseteq V \times V$, que verifican las propiedades siguientes:

- *Reflexiva:* (como en el caso anterior) $\forall x \in V, \quad (x, x) \in R$. En el Ejemplo 35 debemos completar nuestra lista de aristas, mientras que el ejemplo de la circunferencia ya la verifica.

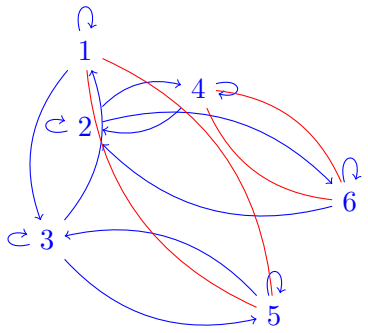
- *Simétrica:* $\forall x \in V, \quad (x, y) \in R \Leftrightarrow (y, x) \in R$. En el caso de la circunferencia ya se satisface. Mientras que en el caso del Ejemplo 35 debems completar nuestra lista, añadiendo las aristas:

$$\{(3, 1), (5, 3), (4, 2), (6, 2)\},$$

para que se satisfaga. Esto nos da un grafo como:



- *Transitiva:* Que se expresa como ya se ha indicado. Es claro que el caso de la circunferencia tenemos una relación de equivalencia y en el caso del Ejemplo 35 habrá que completar con todos los casos. Esto nos dará una figura como la siguiente:



Este último grafo ya nos dará una relación de equivalencia.

A.4.3 Clasificando y Etiquetando elementos: Conjunto Cociente.

Mientras las relaciones de orden pretenden establecer una preferencia de unos elementos sobre otros, dentro de un cierto conjunto, las relaciones de equivalencia pretenden clasificar los elementos del mismo conjunto para, posteriormente etiquetarlos. Se llamará conjunto cociente al conjunto de etiquetas finales.
El término etiqueta no es tan espúreo dado que las etiquetas son lo que definen, de manera bastante desafortunada en ocasiones, cosas tan dispares como la sociedad de consumo o la claisificación e Linneo de los seres vivos, por ejemplo.
Así, tomemos un conjunto X y una relación de equivalencia $\sim \subseteq X \times X$ definida sovre él. Para un elementos $x \in X$, consideraremos su clase de equivalencia como el conjunto formado por todos los elementos de X equivalentes a x, es decir,

$$[x] := \{y \in X \ : \ x \sim y\}.$$

Las clases son sunconjuntos de X y se verifican las siguientes tres propiedades que indican que se trata de una partición de X:

- $X = \bigcup_{x \in X} [x]$,
- $[[x] \cap [y] = \emptyset] \vee [[x] = [y]]$.
- $[x] \neq \emptyset$.

Así, retomando los ejemplos, podemos clasificar un colectivo X de personas (los habitantes de una ciudad, por ejemplo) mediante la relación de equivalencia *"$x \sim y$ si y solamente si [x tiene el mismo modelo de coche que y]"*. Se trata claramente de una relación de equivalencia sobre X. La relación no es fina como clasificador puesto que hay individuos que poseen más de un coche y, por tanto, más de un modelo, con lo que podríamos tener que un "Dacia" y un BMW están relacionados. Admitamos que la relación se refina mediante *"$x \sim y$ si y solamente si [x e y poseen un mismo modelo de coche y ambos le prefieren entre los de su propiedad]"*.
Ciertamente cada clase de euivalencia recorre todos los individuos de la una ciudad que poseen el mismo modelo de coche. Así, podríamos tener la clase [Luis], formada por todas las personas que no tienen coche o [Juan] formada por todas las personas que tienen un Dacia Logan del 96. De hecho, la etiqueta del coche define la clase.

Podríamos usar el símbolo $\emptyset$ para describir la clase de quienes poseen ningún coche y el símbolo TT para quienes posean un Audi TT. Recíprocamente, en la sociedad de consumo, la publicidad no nos vende el coche que sale en un anuncio sino todos los coches equivalentes a él, es decir, todos los que tienen las mismas caracteísticas de los que fabrica esa empresa...Es lo que se llama "Marca" o "etiqueta" y es lo que los ciudadanos de las sociedades llamadas avanzadas compran.
En la clasificación de Linneo también tenemos una relación de equivalencia, esta vez entre todos los seres vivos. Dos seres vivos serían equivalentes si pertenecen al mismo Reino, Orden, Familia, Género, Especie....Luego se imponen las etiquetas. Así, la *rana muscosa* esla etiqueta que define la clase de equivalencia de todas las ranas de montaña de patas amarillas y no distingue entre ellas como individuos: una de tales ranas pertenece a la clase (etiqueta) pero ella no es toda la clase. En tiempos más recientes, el afán clasificatorio de Linneo se reconvierte en el afán clasificatorio de los genetistas: dos seres vivos son equivalentes si poseen el mismo sistema cromosósico (mapa genético), quedando el código genético como etiqueta individual.
Con ejemplos matemáticos, es obvio que en un grafo no orientado, las clases de equivalencia son las clausuras transitivas (o componentes conexas) de cada elemento. En el caso de los número racionales, por ejemplo, la clase de equivalencia de $2/3$ está formada por todos los pares de números enteros a/b, con $b \neq 0$, tales que $2b = 3a$.
Una vez queda claro que disponemos de clases de equivalencia, podemos considerarlas como elementos. Nace así el conjunto cociente que es el conjunto formado por las clases de equivalencia, es decir,

$$X/\sim := \{[x] \ : \ x \in X\}.$$

En los ejemplos anteriores, el conjunto cociente es el conjunto de las etiquetas de coches, el conjunto de los nombres propuestos por Linneo para todas las especies de animales, etc. Nótese que el conjunto cociente es algo que, en muchas ocasiones, se puede escribir (por eso el término etiqueta) aunque hay casos en los que los conjuntos cocientes no son "etiquetables" en el sentido de formar un lenguaje. El ejemplo más inmediato es el caso de los números reals $\mathbb{R}$ que son las etiquetas de las clases de equivalencia de sucesiones de Cauchy, pero que no son expresabels sobre un alfabeto finito.

A.5 Aplicaciones. Cardinales.

Las aplicaciones son un tipo particular de correspondencias.

Definición 163 (Aplicaciones)**.** *Una aplicación entre dos conjuntos A y B es una correspondencia $R \subseteq A \times B$ que verifica las propiedades siguientes:*

- *Todo elemento de A está realcionado con algún elemento de B:*

$$\forall x \in A, \ \exists y \in B, \ (x, y) \in R.$$

- *No hay más de un elemento de B que pueda estar relacionado con algún elemento de A:*

$$\forall x \in A, \forall y, y' \in B, \ ((x, y) \in R) \wedge \big((x, y') \in R\big) \Longleftarrow y = y'.$$

En ocasiones se resume con la expresión:

$$\forall x \in A, \exists \mid y \in B, \ \ (x,y) \in R,$$

donde el cuantificador existencial modificado $\exists \mid$ *significa "existe uno y sólo uno".*

Notación 164. *Notacionalmente se expresa* $R : A \longrightarrow B$*, en lugar de* $R \subseteq A \times B$ *y, de hecho, se suelen usar letras minúsculas del tipo* $f : A \longrightarrow B$ *para indicar la aplicación* f *de* A *en* B*. Al único elemento de* B *relacionado con un* $x \in A$ *se le representa* $f(x)$ *(es decir, escribimos* $x \longmapsto f(x)$ *en lugar de* $(x, f(x)) \in f$*).*
Por simplicidad, mantendremos la notación (inadecuada, pero unificadora) $f : A \longrightarrow B$ *también para las correspondencias, indicando en cada caso si hacemos referencia a una aplicación o a una correspondencia, y reservaremos la notación* $R \subseteq A \times A$ *para las relaciones.*

Ejemplo 37 (Aplicación (o función) característica de un subsonjunto)**.** *Sea* X *un conjunto* $L \subseteq X$ *un subconjunto. De modo natural tenemos defindа una aplicación que toma como entradas los elementos de* X *y depende fuertemente de* L*: el la función característica*

$$\chi_L : X \longrightarrow \{0,1\}$$

y que viene definida para cada $x \in X$ *mediante:*

$$\chi_L(x) := \begin{cases} 1, & \text{si } \ x \in L \\ 0, & \text{en otro caso-} \end{cases}$$

Se usa la expresión función cuando se trata de aplicaciones $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, expresión que viene de la tradición del Análisis Matemático.

Definición 165 (Composición)**.** *Dados tres conjuntos* A*,* B *y* C *y dos aplicaciones* $f : A \longrightarrow B$*,* $g : B \longrightarrow C$*, podemos definir una aplicacin (llamada composición de* f *y* g*) que denotaremos* $g \circ f : A \longrightarrow C$ *y viene definida por la regla:*

$$x \longmapsto g(f(x)), \ \forall x \in A,$$

es decir, "primero aplicamos f *sobre* x *y luego aplicamos* g *a* $f(x)$*".*

Para una aplicación (o correspondencia) $f : A \longrightarrow A$ podemos definir la potencia mediante:

$$\begin{array}{ll} f^0 := & Id_A, \ \text{(la identidad)}, \\ f^1 := & f, \\ f^n := & f^{n-1} \circ f, \ \ \forall n \geq 2. \end{array}$$

A.5.1 Determinismo/Indeterminismo.

A partir de una aplicación (o correspondencia) $f : A \longrightarrow A$, podemos definir una estructura de grafo orientado "natural", definiendo los vértices como los elementos de A y las aristas mediante

$$V := \{(x, f(x)) \ : \ x \in A\}.$$

En algunos casos, los alumnos habrán llamdo a este conjunto de vértices *"el grafo de la función f"*.
Dentro de ese grafo orientado, podemos considerar la parte de la "componente conexa" de x que son descendientes de x. Este conjunto vendrá dado por las iteraciones de f, es decir:

$$\{, x, f(x), f^2(x), f^3(x), \ldots, f^n(x), \ldots\}.$$

La diferencia entre el hecho de ser f aplicación o correspondencia se traduce en términos de "determinismo" o "indeterminismo":

- En el caso de ser aplicación, el conjunto de sucesores es un conjunto, posiblemente, infinito, en forma de camino (un árbol sin ramificaciones):

  $$x \longrightarrow f(x) \longrightarrow f^2(x) \longrightarrow f^3(x) \longrightarrow \cdots .$$

  Se dice que (A, f) tiene una dinámica *determinista*.

- En el caso de ser correspondencia, el conjunto de los sucesores de x toma la forma de árbol (con posibles ramificaciones): algunos valores no tendrán descendientes y otros tendrán más de un descendiente directo. Se dice que (A, f) tiene una dinámica *indeterminista*.

Ejemplo 38. *Tomemos $A := \mathbb{Z}/5\mathbb{Z} := \{0, 1, 2, 3, 4\}$, las clases de restos módulo 5 y consideremos $f := A \longrightarrow A$, dada mediante:*

$$x \longmapsto f(x) = x^2, \forall x \in A.$$

Es una aplicación, por lo que los descendientes de cada valor $x \in A$ forman un camino (un árbol sin ramificaciones). Por ejemplo, $\{0\}$ es el conjunto de todos los descendientes de 0, mientras que, si empezamos con 3 tendremos:

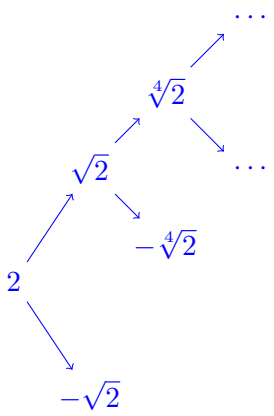
$$3 \longmapsto f(3) = 4 \longmapsto f^2(3) = 1 \longmapsto f^3(3) = 1 \longmapsto 1 \longmapsto \cdots ,$$

COmo f es aplicación tendremos, para cada $x \in A$ una dinámica determinista.

Ejemplo 39. *Un ejemplo de indeterminismo sería $A = \mathbb{R}$ y la correspondencia:*

$$R := \{(x, y) \ : \ x = y^2\}.$$

En este caso, si $x < 0$ no hay descendientes, si $x = 0$ hay solamente un deecendiente, y si $x > 0$ tenemos una infinidad de descendientes en forma de árbol no equilibrado. Por ejemplo,



Los vértices $-\sqrt{2}, -\sqrt[4]{2}, \ldots$ *no tendrán descendientes, mientras los positivos tienen un par de descendientes directos.*
Debe señalarse que este ejemplo muestra un indeterminismo fuertemente regular (sabemos la regla) pero, en general, el indeterminismo podría presentar una dinámica muy impredecible.

A.5.2 Aplicaciones Biyectivas. Cardinales.

Sólo un pequeño resumen del proceso de contar el número de elementos de un conjunto, noción que preocupaba originalmente a G. Cantor.

Definición 166 (Aplicaciones inyectivas, suprayectivas, biyectivas)**.** *Sea* $f : A \longrightarrow B$ *una aplicación.*

- *Decimos que* f *es inyectiva si verifica:*

$$\forall x, x' \in A, \quad \big(f(x) = f(x')\big) \Longrightarrow x = x'.$$

- *Decimos que* f *es suprayectiva si verifica:*

$$\forall y \in B, \exists x \in A, \quad f(x) = y.$$

- *Decimos que* f *es biyectiva si es, a la vez, inyectiva y suprayectiva.*

Obsérvese que una aplicación $f : A \longrightarrow B$ es biyectiva si y solamente si disponemos de una aplicación (llamada inversa de f) que se suele denotar por $f^{-1} : B \longrightarrow A$ y que satisface:

$$f \circ f^{-1} = Id_B, \quad f^{-1} \circ f = Id_A,$$

donde $\circ$ es la composición y Id_A e Id_B son las respectivas aplicación identidad en A y en B. En general las aplicaciones no tienen inversa, es decir, no podemos suponer siempre que sean biyectivas.
El proceso de contar no es sino la fundamentación del proceso "infantil" de contar mediante identificación de los dedos de las manos con los objetos a contar. Este proceso es una biyección.

Definición 167 (Cardinal)**.** *Se dice que dos conjuntos A y B tienen el mismo cardinal (o número de elementos) si existe una biyección $f : A \longrightarrow B$. También se dice que son biyectables.*

- *Un conjunto A se dice finito si existe un número natural $n \in \mathbb{N}$ y una biyección*

$$f : A \longrightarrow \{1, 2, 3, \ldots, n\}.$$

  Por abuso de lenguaje se identifican todos los conjuntos del mismo cardinal y escribiremos, en el caso finito, $\sharp(A) = n$, cuando A sea biyectable a $\{1, 2, \ldots, n\}$.

- *Un conjunto A se dice (infinito) numerable si hay una biyección $f : A \longrightarrow \mathbb{N}$*

- *Un conjunto se dice contable si es finito o numerable.*

Proposición 168. *Si dos conjuntos A e B son biyectables, también son biyectables $\mathcal{P}(A)$ y $\mathcal{P}(B)$ (i.e. , las familias de sus subconjuntos).*

Demostración. Baste con disponer de una biyección $f : A \longrightarrow B$ para poder definir:

$$\widetilde{f} : \mathcal{P}(A) \longrightarrow \mathcal{P}(B),$$

dada mediante:

$$X \longmapsto \widetilde{f}(X) := \{f(x) \in B \ : \ x \in X\} \subseteq B.$$

La inversa de esta transformación será:

$$\widetilde{f}^{-1} : \mathcal{P}(B) \longrightarrow \mathcal{P}(A),$$

dada mediante

$$Y \longmapsto \widetilde{f}^{-1}(Y) := \{x \in A \ : \ f(x) \in Y\}.$$

□

Usualmente se utiliza la notación $f(X)$ y $f^{-1}(Y)$ en lugar de $\widetilde{f}(X)$ y $\widetilde{f}^{-1}(Y)$, usadas en la prueba anterior.
Algunos cardinales y propiedades básicas:

a. Los conjuntos $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ son conjuntos numerables, mientras que $\mathbb{R}$ o $\mathbb{C}$ son conjuntos infinitos (no son finitos) y son no numerables (no son biyectables con $\mathbb{N}$).

b. Los subconjuntos de un conjunto finito son también finitos. Entre los subconjuntos A, B de un conjunto finito se tiene la propiedad

$$\sharp(A \cup B) + \sharp(A \cap B) = \sharp(A) + \sharp(B).$$

c. Los subconjuntos de un conjunto contable son también contables.

d. Si A y B son finitos tendremos:

$$\sharp\,(A \times B) = \sharp(A)\sharp(B).$$

e. Si A es un conjunto finito, el cardinal de $\mathcal{P}(A)$ (el número de todos sus subconjuntos) es dado por
$$\sharp\left(\mathcal{P}(A)\right) = 2^{\sharp(A)}.$$

f. Si A es un conjunto finito, el número $\sharp(\mathcal{A})$ de aplicaciones $f : A \longrightarrow \{0,1\}$ verifica:
$$\sharp(\mathcal{A}) = \sharp\left(\mathcal{P}(A)\right) = 2^{\sharp(A)}.$$

g. Si A es un conjunto finito,
$$\sharp(A^n) = \left(\sharp(A)\right)^n.$$
Por ejemplo, si K es un cuerpo finito de la forma $K := \mathbb{Z}/p\mathbb{Z}$, donde $p \in \mathbb{N}$ es un número primo, el cardinal
$$\sharp(K^n) = \sharp(K)^n,$$
por lo que se tiene que para cada espacio vectorial V de dimensión finita sobre un cuerpo K finito se tiene:
$$\dim V = \log_{\sharp(K)} \sharp(V).$$

h. Si A es un conjunto finito $\sharp(A) = n$, el número de permutaciones (es decir, biyecciones de A en sí mismo) es $n!$. Además, el número de subconjuntos de cardinal k de A es dado por el número combinatorio:
$$\binom{n}{k} := \frac{n!}{k!(n-k)!}.$$
De ahí que se tenga:
$$2^n := \sum_{k=0}^{n} \binom{n}{k}.$$

Algunas propiedades elementales de los cardinables contables se resumen en:

Proposición 169. *Productos finitos de conjuntos contables es un conjunto contable. Es decir, dados $\{A_1, \ldots, A_n\}$ una familia finita de conjuntos contables, entonces el producto cartesiano $\prod_{i=1}^{n} A_i$ es también contable.*
La unión numerable de conjuntos contables es contable, es decir, dados $\{A_n : n \in \mathbb{N}\}$ una familia numerable de conjuntos, de tal modo queo cada A_n es contable, entonces, tambiés es contable el conjunto:
$$A := \bigcup_{n\in\mathbb{N}} A_n.$$

Si A es un conjunto contable (finito o numerable), el conjunto de palabras A^* también es contable.

Ejemplo 40 (Los subconjuntos de $\mathbb{N}$)**.** *Por lo anterior, los subconjuntos de* $\mathbb{N}$ *son siempre conjuntos contables (finitos o numerables) pero la cantidad de subconjuntos de* $\mathbb{N}$ *es infinita no numerable (es decir, el cardinal de* $\mathcal{P}(\mathbb{N})$ *es infinito no numerable). Para comprobarlo, vamos a mostrar una biyección entre* $\mathcal{P}(\mathbb{N})$ *y el intervalo* $[0,1] \subseteq \mathbb{R}$ *de números reales. Nótese que el cardinal del intervalo* $[0,1]$ *es igual al cardinal de los números reales. Usaremos la función característica asociada a cada subconjunto* $L \subseteq \mathbb{N}$*. Así, dado* $L \in \mathcal{P}(\mathbb{N})$*, definiremos el número real:*

$$L \longmapsto x_L := \sum_{i=1}^{\infty} \frac{\chi_L(i)}{2^i} \in [0,1].$$

Nótese que el número real asociado al conjunto vacío $\emptyset$ *es el número real* $x_\emptyset = 0$*, mientras que el número real* $x_{\mathbb{N}} \in [0,1]$ *es precisamente* $x_{\mathbb{N}} = 1 \in [0,1]$*.*
Recíprocamente, dado cualquier número real $x \in [0,1]$*, éste posee una única expansión "decimal" en base dos (para ser más correcto, digamos, una única expansión binaria):*

$$x := \sum_{i=1}^{\infty} \frac{x_i}{2^i}.$$

Definamos el sunconjunto $L_x \subseteq \mathbb{N}$ *mediante:*

$$L_x := \{i \in \mathbb{N} \ : \ x_i = 1\}.$$

Ambas aplicaciones ($x \longmapsto L_x$ *y* $L \longmapsto x_L$*) son una inversa de la otra y definen biyecciones entre* $[0,1]$ *y* $\mathcal{P}(\mathbb{N})$ *y recíprocamente).*
Dejamos al lector el esfuerzo de verificar que hay tantos número reales (en todo $\mathbb{R}$*) como número reales en el intervalo* $[0,1]$*.*

Bibliography

[Aho–Hopcroft–Ullman, 75] A.V. Aho, J.E. Hopcroft, J.D. Ullman. *The Design and Analysis of Computer Algotrithms.* Addison–Wesley (1975).

[Aho-Ullman, 72a] A.V. Aho, J.D. Ullman. *The Theory of Parsing, Translation and Compiling. Vol I: Parsing.* Prentice Hall, 1972.

[Aho-Ullman, 72b] A.V. Aho, J.D. Ullman. *The Theory of Parsing, Translation and Compiling. Vol II: Compilers.* Prentice Hall, 1972.

[Aho–Ullman, 95] A.V. Aho, J.D. Ullman. *Foundations of Computer Science.* W. H. Freeman (1995).

[Alfonseca, 07] . Alfonseca. *Teoría De Autómatas y Lenguajes Formales.* McGraw–Hill, 2007.

[Balcázar–Díaz–Gabarró, 88] J.L. Balcazar, J.L. Díaz and J. Gabarró. *"Structural Complexity I"*, EATCS Mon. on Theor. Comp. Sci. **11**, Springer (1988).

[Balcázar–Díaz–Gabarró, 90] J.L. Balcázar, J.L. Díaz and J. Gabarró. *"Structural Complexity II"*, EATCS Mon. on Theor. Comp. Sci. Springer (1990).

[Chomsky, 57] N. Chomsky. *Syntactic Structures.* Mouton and Co., The Hague, 1957.

[Chomsky–Miller, 57] N. Chomsky, G. A. Miller."Finite state languages". *Information and Control* **1**:2 (1957) 91–112.

[Chomsky, 59a] N. Chomsky. "On certain formal properties of grammars". *Information and Control* **2**:2 (1959) 137–167.

[Chomsky, 59b] N. Chomsky. "A note on phrase structure grammars". *Information and Control* **2**: 4 (1959) 393–395.

[Chomsky, 62] N. Chomsky. "Context–free grammarsand pushdown storage". *Quarterly Progress Report* No. **65**. Research Laboratory of Electronics, M. I. T., Cambridge, Mass., 1962.

[Chomsky, 65] N. Chomsky. "Three models for the description of language". *IEEE Trans. on Information Theory* **2** (1965) 113-124.

[Cocke-Schwartz, 70] . Cocke, J.T. Shwartz. *Programming Languages and their Compilers*. Courant Institute of Mathematical Sciences, NYU, 1970.

[RE Davis, 89] R.E. Davis. *" Truth, Deduction and Computation" (Logic and Semantics for Computer Science)"*. W.H. Freeman (1989).

[M Davis, 82] M. Davis. *"Computability and Unsolvability"* Dover (1982).

[M Davis, 97] M. Davis. "Unsolvable Problems". *Handbook of Mathematical Logic* North–Holland (1997) 567–594.

[Davis-Weyuker, 94] M.D. Davis, E.J.Weyuker. *"Computability, Complexity, and Languages (Fundamentals of Theoretical Computer Science), 2nd Ed."*. Academic Press (1994).

[Eilenberg, 74] S. Eilenberg. *"Automata,Languages and Machines", vol. A*. Academica Press, Pure and App. Math. **59**–A (1974).

[Garey–Johnson, 79] M.R. Garey, D.S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP–Completness"*. W.H. Freeman (1979).

[Hopcroft-Motwani–Ullman, 07] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3/Ed.* Addison-Wesley, 2007.

[Kleene, 52] S.S. Kleene. *Introduction to Metamathematics*. Van Nostrand Reinhold, New York, 1952.

[Kleene, 56] S.S. Kleene. "Representation of events in nerve nets". In *Automata Studies*. Shannon and McCarthy eds. Princeton University Press, Princeton, N.J. (1956) 3–40.

[Knuth, 65] D.E. Knuth. "On the Translation of Languages from Left to Right". *Information Control* **8** (1965) 707–639.

[Knuth, 97–98] D.E. Knuth. *"The art of computer programming (2nd Ed.), vol. 2 Seminumerical Algorithms"*. Addison–Wesley (1997–98).

[Kozen, 92] D.C. Kozen. *"The Design and Analysis of Algorithms"*. Texts and Monographs in Computer Science, Springer Verlag (1992).

[HandBook, 92] Van Leeuwen, J. (ed.).ıt Handbook of Theoretical Computer Science Elsevier, 1992.

[Lewis-Papa, 81] H.L. Lewis, C.H. Papadimitriou. *"Elements of the Theory of Computation"*. Prentice–Hall (1981).

[Martin, 03] J. Martin. *"Introduction to Languages and the Theory of Computation, 3rd Edition"*. McGraw Hill, 2003.

[Marcus, 67] S. Marcus. *"Algebraic Linguistics; Analytic Models"*. Mathematics in Science and Engineering, vol. **29**, Academic Press (1967).

[Papadimitriou, 94] Christos H. Papadimitrou. *Computational Complexity.* Addison–Wesley (1994)

[Pratt, 75] V.R. Pratt. "Every Prime has a succinct certificate". *SIAM J. on Comput.* **4** (1975) 214–220.

[Savitch, 70] W.J. Savitch. "Relationships between nondeterministic and deterministic tape complexities". *J. Comput. System. Sci.* **4** (1970) 177–192.

[Schönhage–Vetter, 94] A. Schönhage,E. Vetter. *"Fast Algorithms. A Multitape Turing machine Implementation".* Wissenschaftverlag (1994).

[Sipser, 97] M. Sipser (1997). *"Introduction to the Theory of Computation".* PWS Publishing (1997).

[Wagner–Wechsung, 86] Klaus Wagner and Gerd Wechsung. " *Computational complexity*" . D. Reidel (1986).

[Weihrauch, 97] K. Weihrauch. *"Computability".* EATCS monographs on Theor. Comp. Sci. **9**, Springer Verlag (1987).

[Wirth, 96] N. Wirth, *Compiler Construction.* Addison–Wesley International Computer SCience Service, 1996.