Teoría del Autómata

Dr. Alfonso Alba Cadena
fac@galia.fc.uaslp.mx

Facultad de Ciencias
UASLP

Introducción a las notas del curso

- Estas notas están diseñadas para ser una guía en un curso básico de teoría de autómatas y lenguajes formales. El curso inicia con un breve repaso de teoría de conjuntos, y posteriormente presenta los conceptos básicos de autómatas, lenguajes, y gramáticas formales, para terminar con la definición de máquinas de Turing, las cuales son una representación abstracta de lo que es una computadora moderna.

- Se sugiere realizar múltiples ejercicios, tanto en clase como en casa, sobre cada uno de los temas revisados, así como implementar simuladores de autómatas y gramáticas en un lenguaje de uso común como C/C++.

Objetivos Generales

- Presentar las máquinas abstractas que forman la base de la teoría de la computación, y estudiar sus características y limitantes.

- Estudiar los lenguajes formales y sus propiedades. Mostrar que cualquier problema de cómputo equivale al reconocimiento de algún lenguaje.

- Introducir los conceptos básicos de computabilidad y complejidad computacional.

Contenido

1. Conceptos básicos de teoría de conjuntos
2. Autómatas finitos y lenguajes racionales
3. Gramáticas y lenguajes
4. Autómatas de pila
5. Máquinas de Turing
6. Computabilidad y complejidad computacional

Bibliografía sugerida

- **Teoría de la Computación: Lenguajes formales, autómatas y complejidad**
  J. Glenn Brookshear
  Pearson, Addison Wesley Longman

- **Automata and Languages**
  John M. Howie
  Clarendon Press, Oxford

Unidad I

Conceptos básicos

Conjuntos

- Definición ingenua: Un **conjunto** es una agrupación o colección de objetos de algún tipo, los cuales pueden o no estar definidos por una o más características.

- **Notación:**
  - Explícita: se enlistan todos los elementos entre llaves.
  - Implícita: se describen las características que definen los elementos del conjunto.
  - Ejemplo: $\{2, 4, 6, \ldots\} = \{x \text{ tal que } x \text{ es entero par positivo}\}$.

Notación

- Los conjuntos típicamente se denotan mediante letras mayúsculas. Ejemplo:

$$A = \{a, e, i, o, u\}, \qquad \Phi = \{x \text{ tal que } x^2 = 1\}.$$

- Los símbolos $\in$ y $\notin$ indican pertenencia o no-pertenencia de un elemento en un conjunto. Ejemplo:

$$u \in A, \qquad z \notin A, \qquad -1 \in \Phi, \qquad 3 \notin \Phi.$$

- Los símbolos : y $\mid$ significan "tal que". Ejemplo:

$$A = \{x \ : \ x \text{ es vocal minúscula}\}, \qquad \Phi = \{x \mid x^2 = 1\}.$$

- Notar que es posible que un conjunto pertenezca a otro:

$$A = \{1, 2\}, B = \{3, 4, 5\} \implies \{A, B\} = \{\{1, 2\}, \{3, 4, 5\}\}.$$

Paradoja de Russell

- Esta paradoja surge a partir de la definición ingenua de conjunto.

- Sea X un conjunto. Llamémosle a X "normal" si $X \notin X$ y "anormal" si $X \in X$.

- Sea N el conjunto de todos los conjuntos normales; es decir,

$$N = \{X \mid X \notin X\}.$$

- Es N normal o anormal?.

Subconjuntos

- Decimos que A es *subconjunto* de B si y sólo si $x \in A \Longrightarrow x \in B$ (es decir, todo elemento de A es también elemento de B).

- A es subconjunto *propio* de B si existe al menos un $x \in B$ tal que $x \notin A$; es decir $A \neq B$.

- **Notación:** $A \subseteq B$ ó $A \subset B$ (algunos autores utilizan el símbolo $\subset$ para indicar un subconjunto propio)

- Notar que: $A = B \iff A \subseteq B$ y $B \subseteq A$.

Operaciones con conjuntos

- **Unión:** $A \cup B = \{x \mid x \in A \text{ ó } x \in B\}$
- **Intersección:** $A \cap B = \{x \mid x \in A \text{ y } x \in B\}$
- **Complemento:** $A' = \{x \mid x \notin A\}$
- **Resta:** $A - B = \{x \mid x \in A \text{ y } x \notin B\}$

Propiedades de las operaciones

- **Conmutativa:** $A \cup B = B \cup A, \;\; A \cap B = B \cap A$
- **Asociativa Unión:** $(A \cup B) \cup C = A \cup (B \cup C)$
- **Asociativa Intersección:** $(A \cap B) \cap C = A \cap (B \cap C)$
- **Distributiva Unión:** $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- **Distributiva Intersección:** $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- **Leyes de DeMorgan:** $(A \cup B)' = A' \cap B'$ y $(A \cap B)' = A' \cup B'$

Producto Cartesiano

- El producto cartesiano $A \times B$ de dos conjuntos A y B se define como el conjunto de todos los pares ordenados en los cuales el primer elemento pertenece a A y el segundo a B:

$$A \times B = \{(a, b) \mid a \in A \text{ y } b \in B\}.$$

- Ejemplo:

$$\begin{aligned} A &= \{2, 3\} \\ B &= \{x, y, z\} \\ A \times B &= \{(2, x), (2, y), (2, z), (3, x), (3, y), (3, z)\} \\ B \times A &= \{(x, 2), (x, 3), (y, 2), (y, 3), (z, 2), (z, 3)\} \end{aligned}$$

- Notar que $A \times B \neq B \times A$.

Conjunto vacío

- El *conjunto vacío* es el conjunto único que no contiene elementos.

- El conjunto vacío se denota por $\emptyset = \{\}$.

- Algunas propiedades:

  1. Para todo x, $x \notin \emptyset$.

  2. $\emptyset \cup A = A$, $\emptyset \cap A = \emptyset$, $A - \emptyset = A$, $\emptyset \times A = \emptyset$.

  3. Para todo $x \in \emptyset$, x cumple cualquier propiedad.

Cardinalidad y conjunto potencia

- Un conjunto es *finito* si tiene un número finito de elementos; de otra forma, el conjunto es *infinito*.

- El número de elementos de un conjunto finito A se denomina *cardinalidad* y se denota por $|A|$.

- Algunas propiedades:

$$\begin{aligned} |A \cup B| &\geq |A| \\ |A \cap B| &\leq |A| \\ |A \cup B| &= |A| + |B| - |A \cap B| \\ |A \times B| &= |A||B| \text{ si } A \text{ y } B \text{ son finitos} \end{aligned}$$

Conjunto potencia

- El *conjunto potencia* $\mathcal{P}(A)$ de un conjunto A es el conjunto cuyos elementos son todos los subconjuntos de A; es decir:

$$\mathcal{P}(A) = \{X \mid X \subset A\}\,.$$

- Ejemplo: si $A = \{1, 2, 3\}$, entonces

$$\mathcal{P}(A) = \{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}\,.$$

- Si A es finito, entonces $|\mathcal{P}(A)| = 2^{|A|}$.

Relaciones

- Una *relación* entre dos conjuntos A y B es un subconjunto P de $A \times B$.

- Si $(a, b) \in P$, para algún $a \in A$ y $b \in B$, entonces decimos que a está relacionado con b mediante P.

Funciones

- Una *función* ϕ de un conjunto A a un conjunto B es una relación de A a B en la que se cumple lo siguiente:

$$\text{Si } (a, b_1) \in \phi \text{ y } (a, b_2) \in \phi, \text{ entonces } b_1 = b_2.$$

- En otras palabras, una función asocia a un elemento de A con un único elemento de B.

- Notación:
  - Una función ϕ de A a B se denota como $\phi : A \to B$.
  - Si $(a, b) \in \phi$ entonces decimos que b es la imagen de a bajo la función ϕ, y esto se escribe como $b = \phi(a)$.

Dominio e imagen de una función

- El *dominio* dom ϕ de una función $\phi : A \to B$ se define como

$$\text{dom } \phi = \{a \in A \mid (a, b) \in \phi \text{ para algún } b \in B\} \subseteq A.$$

  Si dom $\phi = A$, entonces ϕ es una *función total*, de lo contrario, ϕ es una *función estrictamente parcial*. En general, a cualquier función se le puede llamar *función parcial*.

- El *contradominio, rango, o imagen* im ϕ de una función ϕ se define como

$$\text{im } \phi = \{\phi(a) \mid a \in \text{dom } \phi\} \subseteq B.$$

- La *función identidad* $\text{id}_A : A \to A$ de un conjunto A se define como $\text{id}_A(a) = a$ para todo $a \in A$. Notar que dom $\text{id}_A = \text{im } \text{id}_A = A$.

Tipos de funciones

- Una función $\phi : A \to B$ es *uno-a-uno o inyectiva* si para todo $a_1, a_2 \in A$ se cumple que

$$\phi(a_1) = \phi(a_2) \implies a_1 = a_2,$$

  o equivalentemente

$$a_1 \neq a_2 \implies \phi(a_1) \neq \phi(a_2).$$

- Una función $\phi : A \to B$ es *sobre o subyectiva* si im $\phi = B$; es decir, para todo $b \in B$ existe algún $a \in A$ tal que $\phi(a) = b$.

- Una función es *biyectiva* si es uno-a-uno y sobre.

Composición de funciones

- Dadas funciones $\phi : A \to B$ y $\psi : B \to C$, entonces podemos definir la *composición* $\psi \circ \phi : A \to B$ como

$$(\psi \circ \phi)(a) = \psi(\phi(a)),$$

para todo $a \in A$ tal que $\phi(a) \in \text{dom } \psi$.

- Notar que, en general, $\phi \circ \psi$ no está bien definido y es distinto de $\psi \circ \phi$.

- Si $\phi : A \to B$ es una función, entonces

$$\phi \circ \text{id}_A = \text{id}_B \circ \phi = \phi.$$

Función inversa

- Dada $\phi : A \to B$, decimos que $\theta : B \to A$ es una *función inversa* de ϕ si

$$\theta \circ \phi = \mathrm{id}_A \quad \text{y} \quad \phi \circ \theta = \mathrm{id}_B.$$

- Teorema: La inversa de una función ϕ es única y se escribe como ϕ^{-1}.

- Teorema: Una función $\phi : A \to B$ tiene inversa si y sólo si es biyectiva.

Cardinalidad

- Para un conjunto finito A, su cardinalidad $|A|$ es el número de elementos que contiene.

- Para conjuntos infinitos, el concepto de cardinalidad no es tan claro. Sabemos, por ejemplo, que tanto el conjunto de los números naturales positivos $\mathbb{N}^+$ como el de los números reales $\mathbb{R}$ son infinitos, pero intuitivamente uno espera que $\mathbb{R}$ tenga un mayor número de elementos que $\mathbb{N}^+$ ya que $\mathbb{N}^+ \subset \mathbb{R}$.

- Una forma de mostrar que dos conjuntos tienen la misma cardinalidad es encontrando una función biyectiva entre ellos.

Teoremas sobre cardinalidad

- Teorema: Para cualquier conjunto X, $|X| < |\mathcal{P}(X)|$.

- Corolario: $|X| < |\mathcal{P}(X)| < |\mathcal{P}(\mathcal{P}(X))| < \ldots$.

  Es decir, no existe una cardinalidad máxima.

- Teorema: Para cualquier conjunto infinito X, $|\mathbb{N}^+| \leq |X|$.

  Es decir, no existe ningún conjunto infinito con cardinalidad menor a la de $\mathbb{N}^+$.

Conjuntos contables e incontables

- Un conjunto X es *contable* si es finito, o si $|X| = |N^+|$.
- Si $|X| > |N^+|$, entonces decimos que X es *incontable*.
- El conjunto potencia de cualquier conjunto infinito es incontable.
- Si los elementos de un conjunto X se pueden listar de una manera ordenada, entonces el conjunto es contable (ya que existe una función biyectiva entre X y $\mathbb{N}^+$).

Ejemplos de conjuntos contables e incontables

- $\mathbb{N}^+ \times \mathbb{N}^+$ es contable: se pueden enlistar primero las parejas que suman 2, luego las que suman 3, etc.

$$\mathbb{N}^+ \times \mathbb{N}^+ = \{(1,1),(1,2),(2,1),(1,3),(2,2),(3,1),\ldots\}.$$

- El conjunto $\mathcal{F}$ de funciones de $\mathbb{N}^+$ a $\mathbb{N}^+$ es incontable ya que $|\mathcal{F}| \geq |\mathcal{P}(\mathbb{N}^+)|$.

  Prueba: sea X un subconjunto de $\mathbb{N}^+$, entonces podemos definir la función $f_X : \mathbb{N}^+ \to \mathbb{N}^+$ como

$$f_X(x) = \begin{cases} 1 & \text{si } x \in X \\ 0 & \text{si } x \notin X. \end{cases}$$

  Por lo tanto, hay por lo menos tantas funciones como subconjuntos de $\mathbb{N}^+$.

Otro ejemplo

- Supongamos que tenemos un lenguaje formado por un conjunto finito de palabras $\Sigma = \{w_1, w_2, \ldots w_n\}$, donde $n = |\Sigma|$.

- Un programa equivale entonces a una secuencia de palabras $w_{a_1} w_{a_2} \ldots w_{a_q}$, donde $a_1, a_2, \ldots, a_q \in \{1, 2, \ldots, n\}$.

- La secuencia $a_1 a_2 \ldots a_q$ equivale a un número entero positivo escrito en base n. Esto sugiere un mapeo biyectivo entre el conjunto Σ^* de todas los posibles programas escritos con palabras de Σ, y el conjunto $\mathbb{N}^+$.

- Por lo tanto, Σ^* es contable. Es decir, solamente una cantidad contable de programas se pueden escribir con el lenguaje Σ.

- Sin embargo, hay un número incontable de funciones de $\mathbb{N}^+$ a $\mathbb{N}^+$, lo cual significa que para algunas de estas funciones no es posible escribir un programa que las resuelva.

Unidad II

Autómatas Finitos

Máquinas abstractas

- Para estudiar el alcance y la potencia de cómputo de las máquinas actuales, es necesario desarrollar modelos simples de máquinas que tengan las mismas capacidades, pero que sean mas fáciles de entender y analizar.

- Uno de los modelos más simples (aunque no muy potente) consiste en un sistema en el que nuestra máquina abstracta pueda ubicarse en uno de varios "estados" en un momento determinado.

- Además, la máquina puede recibir datos de entrada, a partir de los cuales decide, según un conjunto de reglas preestablecidas, si se mantiene en el mismo estado o cambia a un estado distinto.

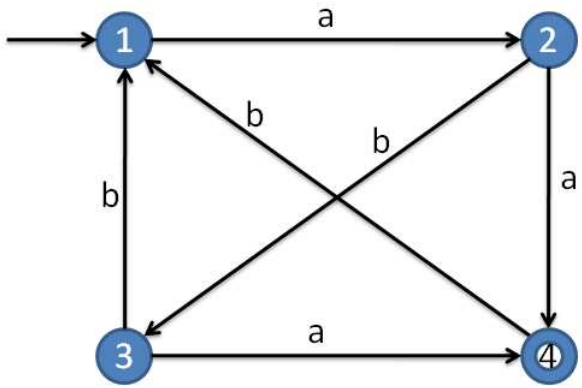
- Estas máquinas abstractas también reciben el nombre de *autómatas*.

Diagramas de transiciones

- Un *diagrama de transiciones* es un grafo en el cual los nodos representan los distintos estados de un autómata, y las aristas representan las posibles transiciones entre estados.

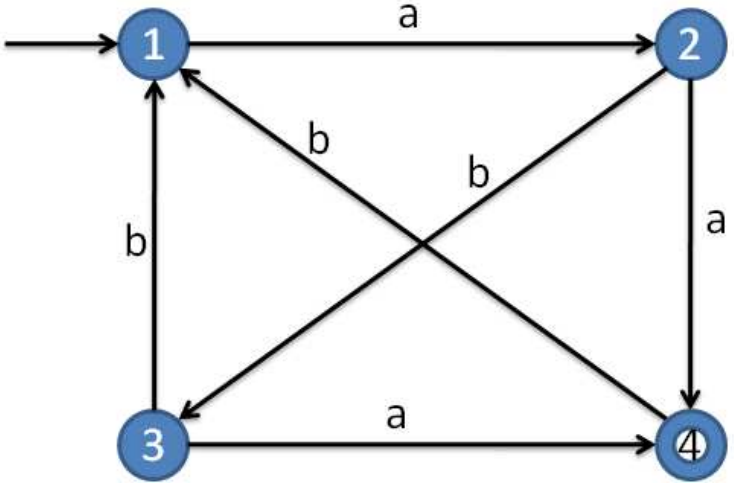
- Cada arista tiene asociado un símbolo. Una transición de un estado a otro solo se realiza si el dato de entrada coincide con el símbolo de la arista correspondiente.

- Típicamente existe un *estado inicial* (indicado con una flecha), y un *estado terminal o de aceptación* indicado con un círculo.



Tablas de transiciones

- También es posible describir un autómata mediante una tabla de transiciones, como se muestra a continuación.



$\Longleftrightarrow$

	1	**2**	**3**	**4**
a	2	4	4	-
b	-	3	1	1

Estado inicial: **1**

Estado final: **4**

Autómatas finitos deterministas

- Formalmente, un *autómata finito determinista* (AFD) se define como una quíntupla $\mathcal{A} = (Q, A, \phi, i, T)$, donde
  - Q es el conjunto finito de estados del autómata.
  - A es el *alfabeto* del autómata; es decir, el conjunto de símbolos que puede recibir como entrada.
  - $\phi : Q \times A \to Q$ es una función de transición que asocia a cada pareja $(q, a), q \in Q, a \in A$ un estado destino $\phi(q, a)$.
  - $i \in Q$ es el estado inicial.
  - $T \subset Q$ es el conjunto de estados terminales.

Palabras

- Dado un alfabeto A (es decir, un conjunto de símbolos), una *palabra* en w en A es una secuencia finita de símbolos $w = a_1 a_2 \ldots a_n$ donde $a_1, a_2, \ldots, a_n \in A$.

- La *longitud* $|w|$ de una palabra $w = a_1 a_2 \ldots a_n$ es precisamente n.

- La *palabra vacía* es aquella que tiene longitud cero, y se denota comúnmente como 1 o ϵ (dado que $1 \notin A$ o $\epsilon \notin A$).

- Las potencias de una palabra w representan la concatenación de w con ella misma. Por ejemplo:

$$a^3 = aaa,\ (ab)^2 = abab,\ b(a^2b)^3a = baabaabaaba.$$

Lenguajes

- Un *lenguaje* L sobre un alfabeto A es cualquier conjunto de palabras en A.

- El lenguaje que contiene a todas las palabras (finitas) formadas con símbolos de un alfabeto A se denota como A^*.

$$A^* = \left\{ a_1 a_2 \dots a_n \ : \ a_j \in A \text{ para } j = 1, \dots, n, \text{ y } n \in \mathbb{N} \right\}.$$

- El lenguaje que contiene a todas las palabras en un alfabeto A de longitud positiva se denota por A^+. Este lenguaje no contiene a la palabra vacía.

$$A^* = A^+ \cup \{1\}.$$

Lenguaje aceptado por un AFD

- **Notación:** Podemos representar una transición $\phi(q_1, a) = q_2$ como $q_1 a = q_2$, de manera que es posible escribir algo como:

$$qa_1a_2 = (qa_1)a_2 = q'a_2, \text{ si } \phi(q, a_1) = q'.$$

- Si $w = a_1a_2 \ldots a_n \in A^*$, entonces

$$qw = qa_1a_2 \ldots a_n = (((qa_1)a_2) \ldots) a_n.$$

- Dado un AFD $\mathcal{A} = (Q, A, \phi, i, T)$ y una palabra $w \in A$, podemos entonces determinar el estado qw al que se llega cuando se ingresa la palabra w al autómata estando en el estado q.

- El *lenguaje* $L(\mathcal{A})$ *aceptado* por el autómata $\mathcal{A}$ se define entonces como

$$L(\mathcal{A}) = \{w \in A^* \ : \ iw \in T\}.$$

- Un lenguaje $L \subset A^*$ es *reconocible* por un AFD $\mathcal{A}$ si y sólo si $L = L(\mathcal{A})$.

Ejercicios:

1. Diseñar un diagrama de transiciones para reconocer expresiones aritméticas simples de números enteros positivos separados por los símbolos $+$, $-$, $\times$, $\div$. Por ejemplo:

$$54 + 23,\ 12 - 8,\ 12345 \times 54321,\ 8 \div 3.$$

2. Diseñar un AFD para una máquina vendedora de refrescos que llegue a un estado de aceptación cuando el usuario ha introducido la cantidad suficiente para comprar un refresco. Los refrescos cuestan \$6 y la máquina acepta monedas de \$1, \$2, y \$5. Cómo manejaría el caso en el que el usuario introduce más de \$6?

3. Dibuje el diagrama de transiciones y describa el lenguaje reconocido por el AFD $\mathcal{A} = (Q = \{q_1, q_2, q_1\},\ A = \{a, b\},\ \phi,\ i = q_1,\ T = \{q_3\})$, donde ϕ está dada por la siguiente tabla:

	q_1	q_2	q_3
a	q_2	q_2	q_3
b	q_1	q_3	q_3

Límites computacionales de los AFD's

- El lenguaje $L = \{a^n b^n \ : \ n \geq 1\}$ no es reconocible por ningún AFD.

- **Prueba:** Suponer que existe un AFD $\mathcal{A} = (Q, A, \phi, i, T)$ tal que $L(\mathcal{A}) = L$. Sea $q_n = ia^n$ para $n = 1, 2, \ldots$. Entonces $q_n b^n \in T$ para todo n. Ya que Q es finito, entonces deben existir m, n distintos tales que $q_m = q_n$. Entonces, $ia^m b^n = q_m b^n = q_n b^n \in T$; sin embargo $a^m b^n \notin L$, lo cual contradice nuestra suposición inicial.

- Así como L, existen muchos lenguajes que no pueden reconocerse por ningún autómata finito.

Lema del bombeo

- Sea L un lenguaje infinito reconocible en A^*. Existe entonces un entero positivo N tal que toda $z \in L$ con $|z| > N$ puede factorizarse como $z = uvw$ donde

  1. $u, w \in A^*, \quad v \in A^+$,
  2. $|uv| \leq N$,
  3. $uv^m w \in L$ para todo $m \geq 0$.

- El lema del bombeo se utiliza comúnmente para mostrar (por contradicción) que un lenguaje L no es reconocido por ningún AFD: uno supone que L es reconocible y elige una palabra $z \in L$ y una factorización $z = uvw$ adecuadas, de manera que para algún m, $uv^m w \notin L$.

Ejemplo de uso del Lema del Bombeo

- Sea $A = \{a, b\}$ y $L = \{w \in A^* \ : \ w^R = w\}$, donde w^R denota el reverso de una palabra. En otras palabras, L es el lenguaje formado por todos los palíndromos en A^*.

- Suponer que L es reconocible por algún AFD. Entonces, existe un $N > 0$ para el cual se cumple el Lema del Bombeo.

- Sea $n > N$. Considerar la palabra $z = a^n b a^n \in L$. Por el Lema del Bombeo, z puede factorizarse como $z = uvw$ con $|v| \geq 1$ y $|uv| \leq N < n$. Por lo tanto, $u = a^p$, $v = a^q$, y $w = a^r b a^n$ donde $q \geq 1$ y $p + q + r = n$.

- Sin embargo, $uv^m w \notin L$ para $m \neq 1$. Por ejemplo, para $m = 0$,

$$uv^0 w = uw = a^{n-q} b a^n \notin L.$$

- Por lo tanto, L no es reconocible.

Observación

- Teorema: Para cualquier alfabeto A (no vacío), existe una infinidad de lenguajes en A^* que no son reconocibles.

- Prueba: A^* es infinito contable, por lo tanto, el conjunto de todos los lenguajes en A^*, es decir, $\mathcal{P}(A^*)$ es incontable. Por otro lado, el conjunto de todos los AFD cuyo alfabeto es A es contable (se pueden enlistar primero todos los AFD con un estado, luego los que tienen dos estados, etc). Por lo tanto, no existen suficientes AFD para reconocer todos los lenguajes en A^*.

Ejercicios

1. Mostrar, usando el Lema del Bombeo, que el lenguaje

$$L = \{a^p \ : \ p \text{ es primo}\}$$

no es reconocible por ningún AFD.

2. Sea $\mathcal{A} = (Q, A, i, T, \phi)$ un AFD. Diseñe otro AFD que reconozca el complemento de $L(\mathcal{A})$ en A^*; es decir, $A^* - L(\mathcal{A})$.

3. Diseñe un AFD que reconozca números, con o sin decimales, positivos o negativos. Por ejemplo, las cadenas

$$3.1416, \quad 6, \quad -5.8$$

son reconocidas, mientras que

$$3.44.4, \quad 5 - 44. - 4, \quad .. - -. - . - ..$$

no lo son. Utilice como alfabeto $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, .\}$.

Autómatas Finitos No-Deterministas

- Un *autómata finito no-determinista* (AFN) es una máquina de estado finito en la cual de un mismo estado y con el mismo símbolo de entrada puede haber varias posibles transiciones a distintos estados.

- La diferencia con un AFD es que, durante una transición, el estado destino es único y está bien determinado por la función de transición.

- En el caso de los AFN, la función de transición no devuelve un solo estado, sino un conjunto de estados destino.

Definición formal de un AFN

- Un AFN es una quíntupla $\mathcal{N} = (Q, A, \phi, i, T)$ donde
  1. Q es un conjunto finito no-vacío de estados.
  2. A es un alfabeto (conjunto de símbolos finito no-vacío).
  3. $\phi : Q \times A \to \mathcal{P}(Q)$ es la función de transición.
  4. $i \in Q$ es el estado inicial.
  5. $T \subseteq Q$ es el conjunto de estados terminales.

- De modo que si $q \in Q$ y $a_1, a_2 \in A$, entonces qa_1 es un subconjunto de Q y

$$qa_1a_2 = (qa_1)a_2 = \bigcup_{p \in qa_1} pa_2$$

  es también subconjunto de Q.

- De igual manera se puede obtener el subconjunto $qw \subseteq Q$, donde $q \in Q$ y $w \in A^*$.

ϵ-transiciones

- Es posible extender la definición de un AFN para que admita transiciones de un estado a otro sin consumir ningún símbolo de entrada.

- Este tipo de transiciones se conocen como *ϵ-transiciones* (donde ϵ representa la palabra vacía), de manera que la función de transición se define como

$$\phi : Q \times (A \cup \{\epsilon\}) \rightarrow \mathcal{P}(A).$$

- Es posible mostrar que para todo autómata $\mathcal{A}$ que utilice ϵ-transiciones puede encontrarse un autómata $\mathcal{A}'$ sin ϵ-transiciones tal que $L(\mathcal{A}') = L(\mathcal{A})$.

Lenguaje aceptado por un AFN

- El lenguaje aceptado por un AFN se define como el conjunto de todas las palabras para las cuales existe una ruta que lleva al autómata del estado inicial a un estado terminal al dar como entrada la palabra.

- Formalmente, si $\mathcal{N} = (Q, A, \phi, i, T)$ es un AFN, entonces

$$L(\mathcal{N}) = \{w \in A^* \ : \ iw \cap T \neq \emptyset\}.$$

Equivalencia entre AFN's y AFD's

- Teorema: Sea $\mathcal{A} = (Q, A, \phi, i, T)$ un autómata, no necesariamente determinista, y sea $L(\mathcal{A})$ el lenguaje reconocido por $\mathcal{A}$. Entonces existe un autómata determinista completo $\mathcal{A}'$ tal que $L(\mathcal{A}') = L(\mathcal{A})$.

- Método de conversión: Se define el AFD

  $\mathcal{A}' = \left(\mathcal{P}(Q),\ A,\ \psi : \mathcal{P}(Q) \times A \to \mathcal{P}(Q),\ \{i\},\ T' = \{P \in \mathcal{P}(Q)\ :\ P \cap T \neq \emptyset\}\right),$

  donde la función de transición ψ está dada por

$$\psi(P, a) = \bigcup_{p \in P} \phi(p, a), \text{ para todo } P \in \mathcal{P}(Q),\ a \in A.$$

Accesibilidad

- En un autómata determinista $\mathcal{A} = (Q, A, \phi, i, T)$, un estado $q \in Q$ es *accesible* si existe $w \in A^*$ tal que $iw = q$; es decir, si es posible llegar a q a partir del estado inicial mediante alguna secuencia de símbolos.

- Al autómata $\mathcal{A}$ se le dice *accesible* si todo $q \in Q$ es accesible.

- Dado un autómata no accesible $\mathcal{A}$, se puede obtener un autómata accesible $\mathcal{A}'$ tal que $L(\mathcal{A}') = L(\mathcal{A})$ simplemente eliminando los estados no accesibles.

Coaccesibilidad

- Un estado q de un autómata $\mathcal{A} = (Q, A, \phi, i, T)$ es *coaccesible* si existe $w \in A^*$ tal que $qw \in T$; es decir, si existe alguna ruta que lleve al autómata de q a algún estado terminal.

- Un autómata es coaccesible si todos sus estados son coaccesibles.

- Un autómata *podado* (trim) es aquél que es tanto accesible como coaccesible.

- El lenguaje reconocido por un autómata no se modifica si uno elimina todos los estados que no son coaccesibles.

Lenguajes racionales

- Dados dos lenguajes $L_1, L_2 \subseteq A^*$, definimos la concatenación o producto L_1L_2 como

$$L_1L_2 = \{uv : : u \in L_1, v \in L_2\}.$$

- Dado un subconjunto $L \in A^*$ definimos

$$L^* = \{u_1u_2 \dots u_n \ : \ n \geq 0, \ u_1, u_2, \dots, u_n \in L\}.$$

  Esta operación se conoce como *operación estrella de Kleene*.

- Un subconjunto de A^* se llama *racional* si puede obtenerse a partir de subconjuntos finitos de A^* mediante un número finito de operaciones de unión, concatenación, y estrella de Kleene.

- El conjunto de todos los subconjuntos racionales de A^* se denota por Rat A^*.

Capacidad de reconocimiento de los autómatas finitos

- Sea A un alfabeto finito. Cualquier conjunto $\{w\}$, $w \in A^*$ (es decir, lenguajes que consistan de una sola palabra) es reconocible.

- El conjunto vacío es reconocible.

- Sea A un alfabeto finito y sean L_1, L_2 subconjuntos reconocibles de A^*. Entonces, la unión $L_1 \cup L_2$ es reconocible.

- Sea A un alfabeto finito. Entonces, cualquier subconjunto finito de A^* es reconocible.

Teorema de Kleene

- Sea A un alfabeto finito y L un subconjunto de A^*. Entonces, L es reconocible si y solo si L es racional.

- La prueba del teorema consiste en mostrar que los lenguajes reconocibles son cerrados bajo las operaciones de unión, concatenación y estrella de Kleene.

Unidad III

Gramáticas y Lenguajes

Gramáticas

- Una *gramática formal* o *gramática de estructura de frases* es una cuádrupla $\Gamma = (V, A, \pi, \sigma)$ donde

  1. V es un conjunto finito de símbolos llamado *vocabulario* de Γ.
  2. $A \subseteq V$ es el *alfabeto terminal*.
  3. $\pi \subseteq (V - A)^+ \times V^*$ es el conjunto (finito) de *producciones*.
  4. $\sigma \in V - A$ es el *símbolo inicial*.

- Los elementos de π se llaman *producciones* y se denotan por $u \to v$, con $u \in (V - A)^+$ y $v \in V^*$.

- Si $u \to v$ y $u \to w$, podemos escribir simplemente $u \to v|w$.

- La acción de una gramática consiste en generar palabras mediante la sustitición de símbolos no terminales por cadenas de V^*. Esta acción termina una vez que la palabra generada contiene solamente símbolos terminales.

Lenguaje generado por una gramática

- Dada una gramática $\Gamma = (V, A, \pi, \sigma)$, definimos una *derivación* $w \Rightarrow w'$ para $w, w' \in V^*$ si existen $x, y \in V^*$ y una producción $u \to v$ en π tales que

$$w = xuy,\ w' = xvy.$$

- En general, si existe una cadena de derivaciones $w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_n$, podemos escribir simplemente $w_1 * \Rightarrow w_n$.

- De esta manera, se define el *lenguaje* $L(\Gamma)$ *generado por* Γ como el conjunto de palabras en el alfabeto terminal que pueden derivarse del símbolo inicial. Es decir,

$$L(\Gamma) = \{w \in A^* \ :\ \sigma * \Rightarrow w\}.$$

Ejemplos de gramáticas

- **Ejemplo 1:** Sea $\Gamma = (V, A, \pi, \sigma)$ con

$$\begin{aligned} A &= \{\text{el, un, perro, gato, muerde, come}\}, \\ V &= A \cup \{\text{oración, artículo, sujeto, verbo}\}, \end{aligned}$$

  y π consta de las siguientes producciones:

$$\begin{aligned} \text{oración} &\rightarrow \text{artículo sujeto verbo} \\ \text{artículo} &\rightarrow \text{el} \mid \text{un} \\ \text{sujeto} &\rightarrow \text{perro} \mid \text{gato} \\ \text{verbo} &\rightarrow \text{muerde} \mid \text{come} \end{aligned}$$

- **Ejemplo 2:** Considerar la gramática cuyas producciones son:

$$\sigma \rightarrow a\sigma,\ \sigma \rightarrow b\sigma, \sigma \rightarrow a\lambda,\ \lambda \rightarrow b.$$

Tipos de gramáticas

- Las gramáticas más simples son las *gramáticas regulares*, en las cuales todas las producciones en π son de la forma

$$\alpha \to x\beta, \quad x \in A^+, \ \alpha, \beta \in V - A,$$

  o bien

$$\alpha \to y, \quad \alpha \in V - A, \ y \in A^*.$$

- Una gramática *libre de contexto* es aquella cuyas producciones son todas de la forma

$$\alpha \to z, \quad \alpha \in V - A, \ z \in V^*.$$

- Un lenguaje es *regular* (resp. *libre de contexto*) si puede ser generado por una gramática regular (resp. libre de contexto).

- Notar que toda gramática/lenguaje regular es libre de contexto.

Lenguajes regulares y racionales

- **Teorema:** Un lenguaje es regular si y solo si es racional.

- **Teorema:** Dado un alfabeto finito A y un subconjunto $L \subseteq A^*$, entonces los siguientes enunciados son equivalentes (es decir, o bien todos son ciertos, o son todos falsos):

  - L es racional.

  - Existe un autómata finito $\mathcal{A}$ tal que $L = L(\mathcal{A})$.

  - Existe una gramática regular Γ tal que $L = L(\Gamma)$.

Ejercicios

1. Considere la gramática $\Gamma = (V, A, \pi, \sigma)$ cuyas producciones son:

$$\begin{aligned} \sigma &\rightarrow \epsilon | x\lambda | y\gamma \\ \gamma &\rightarrow y\gamma | \epsilon \\ \lambda &\rightarrow x\lambda | \epsilon \end{aligned}$$

   a) Describa el lenguaje generado por Γ.

   b) Dibuje el diagrama de transiciones de un autómata finito que reconozca el mismo lenguaje.

2. Escriba una gramática regular que reconozca el lenguaje $L = b^2(ab)^*a^2$.

3. Sean L_1 y L_2 lenguajes regulares, generados respectivamente por las gramáticas $\Gamma_1 = (V_1, A_1, \phi_1, \sigma_1)$ y $\Gamma_2 = (V_2, A_2, \phi_2, \sigma_2)$. Construya una gramática regular que genere la concatenación L_1L_2.

4. Escriba una gramática que genere todos los nombres de variables permitidos en el lenguaje C.

Gramáticas libres de contexto

- Como se vió anteriormente, una gramática libre de contexto (LDC) es aquella en la cual todas las producciones son de la forma

$$\alpha \to z, \quad \alpha \in V - A, \quad z \in V^*.$$

- Las gramáticas LDC son más generales que las regulares, por lo que todo lenguaje racional es libre de contexto.

- Sin embargo, no todo lenguaje libre de contexto es regular.

Lenguajes libres de contexto

- **Ejemplo 1:** el lenguaje $L = \{a^n b^n \ : \ n \geq 1\}$ no es regular, pero es LDC ya que es generado por la siguiente gramática:

$$\begin{aligned} \sigma &\rightarrow a\lambda b \\ \lambda &\rightarrow a\lambda b \\ \lambda &\rightarrow \epsilon \end{aligned}$$

- **Ejemplo 2:** Mostrar que el lenguaje $L = \left\{w \in \{a,b\}^* \ : \ w = w^R\right\}$ es libre de contexto, pero no regular.

Lenguajes libres de contexto

- **Ejemplo 3:** Dado un alfabeto A, si $a \in A$ y $w \in A^*$, entonces $|w|_a$ denota el número de ocurrencias de a en w. Verificar que el lenguaje

$$L = \{w \in \{a,b\}^* \ : \ |w|_a = |w|_b\}$$

es generado por la siguiente gramática:

$$\begin{array}{lllllllll} \sigma & \to & a\mu & \lambda & \to & a & \mu & \to & b \\ \sigma & \to & b\lambda & \lambda & \to & a\sigma & \mu & \to & b\sigma \\ & & & \lambda & \to & b\lambda^2 & \mu & \to & a\mu^2 \end{array}$$

Forma Normal de Chomsky

- Una gramática libre de contexto que genere un lenguaje en A^+ se encuentra en la *Forma Normal de Chomsky* si cada una de sus producciones es de alguno de los siguientes tipos:

$$\begin{aligned} \lambda &\rightarrow \mu\nu, \quad \lambda, \mu, \nu \in V - A, \\ \lambda &\rightarrow a, \quad \lambda \in V - A, \ a \in A. \end{aligned}$$

- **Teorema:** Todo lenguaje libre de contexto en A^+ puede generarse mediante una gramática en la Forma Normal de Chomsky.

Conversión a la Forma Normal de Chomsky

1. Eliminar las producciones del tipo $\lambda \rightarrow \epsilon$.

2. Eliminar producciones triviales del tipo $\lambda \rightarrow \mu$, $\mu \in V - A$.

3. Por cada $a \in A$, agregar un no-terminal β_a y reemplazar las ocurrencias de a en todas las producciones por β_a. Luego, agregar producciones $\beta_a \rightarrow a$ para cada $a \in A$.

4. Cambiar cada produccion de la forma $\lambda \rightarrow \alpha_1\alpha_2 \ldots \alpha_n$ por una cadena de producciones $\lambda \rightarrow \alpha_1\lambda_1$, $\lambda_1 \rightarrow \alpha_2\lambda_2$, $\ldots$, $\lambda_{n-2} \rightarrow \alpha_{n-1}\alpha_n$.

Ejercicios

1. Para cada uno de los lenguajes siguientes, escribir una gramática libre de contexto que lo genere:

   (a) $L = \{a^r b^s c^t \ : \ s = r + t\}$.

   (b) $L = \{a^m b^n a^m \ : \ m, n \geq 1\}$.

   (c) $L = \{a^n b^m \ : \ 0 \leq n < m\}$.

2. Para cada una de las siguientes gramáticas, describir el lenguaje que genera y convertirla a la Forma Normal de Chomsky:

   (a)
   $$\begin{array}{lcl} \sigma & \rightarrow & a\sigma b \mid c\lambda d, \\ \lambda & \rightarrow & \sigma \mid \epsilon. \end{array}$$

   (b)
   $$\begin{array}{lcl} \sigma & \rightarrow & \lambda c\mu, \\ \lambda & \rightarrow & a\lambda \mid \epsilon, \\ \mu & \rightarrow & b\mu \mid \epsilon. \end{array}$$

Unidad IV

Autómatas de pila

Limitantes de los autómatas de estado finito

- Los autómatas de estado finito realizan una transición hacia un nuevo estado determinado únicamente por el estado actual y el símbolo de entrada, sin importar los estados recorridos anteriormente, o las entradas previas.

- Esta incapacidad de recordar estados o entradas previas limita la capacidad de reconocimiento de los autómatas finitos.

- Para superar esta limitante, podemos agregar algún tipo de memoria al autómata. Una de las estructuras más básicas de memoria es una pila, en la cual se pueden insertar y extraer símbolos únicamente por un solo extremo de la pila, llamado *tope*.

Autómatas de pila

- Un *autómata de pila* (ADP) es un autómata de estado finito no-determinista al que se le agrega una pila de símbolos y la siguiente funcionalidad:

  1. En cada transición, se extrae un símbolo de la pila y se utiliza para decidir a qué estado ir.

  2. En cada transición se puede insertar cualquier cantidad de símbolos en la pila.

- De esta forma, el estado destino está determinado por: (1) el estado actual, (2) el símbolo de entrada, y (3) el símbolo extraído del tope de la pila.

Definición formal de autómata de pila

- Formalmente, un ADP es una séptupla $\mathcal{M} = (Q, A, S, \phi, i, \zeta, T)$, donde
  - Q es un conjunto finito de estados
  - A es el alfabeto del autómata
  - S es un conjunto finito de símbolos llamado el *alfabeto de la pila*
  - $\phi : Q \times (A \cup \{\epsilon\}) \times (S \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q \times S^*)$ es la función de transición.
  - i es el estado inicial.
  - $\zeta \in S$ es el símbolo inicial de la pila.
  - $T \subseteq Q$ es el conjunto de estados terminales.

- La pila del autómata es simplemente una cadena $p \in S^*$, la cual se modifica en cada transición. Inicialmente, la pila contiene únicamente al símbolo ζ.

Transiciones

- La función de transición se define como

$$\phi \ : \ Q \times (A \cup \{\epsilon\}) \times (S \cup \{\epsilon\}) \ \rightarrow \ \mathcal{P}(Q \times S^*),$$

por lo tanto, podemos definir una transición como una quíntupla de la forma

$$(q, a, \alpha) \rightarrow (q', z), \quad q, q' \in Q, \ a \in A \cup \{\epsilon\}, \ \alpha \in S \cup \{\epsilon\}, \ z \in S^*,$$

si $(q', z) \in \phi(q, a, \alpha)$.

Esto significa que, estando en el estado q, con el símbolo de entrada a, y la cadena $p\alpha$, $p \in S^*$ en la pila, el autómata puede ir al estado q', quedando la cadena pz en la pila (se extrae α y se inserta z).

Ejemplo

- Considerar el ADP

$$\mathcal{M} = (\{q_0, q_1, q_2\}, \{a, b\}, \{\zeta, \alpha\}, \phi, q_0, \zeta, \{q_2\}),$$

donde ϕ contiene las siguientes transiciones:

$$\begin{aligned}
\phi(q_0, a, \zeta) &= \{(q_0, \alpha\zeta)\}, \\
\phi(q_0, a, \alpha) &= \left\{(q_0, \alpha^2)\right\}, \\
\phi(q_0, b, \alpha) &= \{(q_1, \epsilon)\}, \\
\phi(q_1, b, \alpha) &= \{(q_1, \epsilon)\}, \\
\phi(q_1, \epsilon, \zeta) &= \{(q_2, \zeta)\}.
\end{aligned}$$

- Notar que solamente se llega al estado terminal si la cadena de entrada tiene la forma $a^n b^n$, $n \geq 1$.

Cómputos

- Una *descripción instantánea* (DI) es una tercia (q, τ, w) donde
  - $q \in Q$ es el estado en el que se encuentra el autómata
  - $\tau \in S^*$ es el contenido de la pila
  - $w \in A^*$ es la cadena de símbolos de entrada que aún no ha sido leída.

- Un *cómputo* en un ADP es una secuencia de DI's que se obtiene a partir de la aplicación de una o más transiciones, y puede representarse de la siguiente manera:

$$(q_1, \tau_1, a_1 a_2 \ldots a_n) \to (q_2, \tau_2, a_2 \ldots a_n) \to \ldots \to (q', \tau', w').$$

Lenguaje reconocido por un ADP

- Sea $\mathcal{M} = (Q, A, S, \phi, i, \zeta, T)$ un ADP.

- Una palabra $w \in A$ es reconocida $\mathcal{M}$ si existe un cómputo de la forma

$$(i, \zeta, w) \to \ldots \to (q_t, \tau, \epsilon), \quad q_t \in T, \ \tau \in S^*,$$

es decir, que lleve al autómata desde el estado inicial a un estado terminal en el cual la secuencia de símbolos de entrada es precisamente w.

- El lenguaje $L(\mathcal{M})$ reconocido por $\mathcal{M}$ es el conjunto de todas las palabras $w \in A$ que son reconocidas por $\mathcal{M}$:

$L(\mathcal{M}) = \{w \in A^* \ : \text{ existe un cómputo } (i, \zeta, w) \to \ldots \to (q_t, \tau, \epsilon), \ q_t \in T, \ \tau \in S^*\}.$

Reconocimiento por pila vacía

- Como se mencionó anteriormente, una palabra w es reconocida por un autómata de pila $\mathcal{M} = (Q, A, S, \phi, i, \zeta, T)$ si existe un cómputo

$$(i, \zeta, w) \to \ldots \to (t, \tau, \epsilon), \quad t \in T, \ \tau \in S^*.$$

- Otra posibilidad consiste en considerar una palabra w como reconocida si y solo si la pila queda vacía una vez que se ha consumido la palabra de entrada; es decir, si y solo si existe un cómputo

$$(i, \zeta, w) \to \ldots \to (q, \epsilon, \epsilon), \quad q \in Q.$$

  Notar que en este caso, ya no es necesario definir un conjunto $T \subseteq Q$ de estados terminales.

Equivalencia entre criterios de reconocimiento

- Llamemos **ADP-ET** a los autómatas de pila que reconocen palabras cuando se llega a un estado terminal, y **ADP-PV** a los que reconocen palabras al vaciarse la pila.

- Se puede demostrar fácilmente que los ADP-ET y los ADP-PV pueden reconocer la misma clase de lenguajes; es decir, para cada ADP-ET existe un ADP-PV que reconoce el mismo lenguaje, y viceversa.

Autómatas de pila y lenguajes libres de contexto

- Teorema: Sea A un alfabeto y $L \subseteq A^*$. Entonces, L es libre de contexto si y solo si es reconocido por un ADP.

- Demostración (parcial): Suponer que L es LDC, y es generado por una gramática $\Gamma = (V, A, \pi, \sigma)$ en la Forma Normal de Chomsky. Construír el ADP

$$\mathcal{M} = (\{i, q, t\}, A, (V - A) \cup \{\zeta\}, \phi, i, \zeta, \{t\}),$$

donde ϕ contiene las transiciones

  - $\pi(i, \zeta, \epsilon) = \{(q, \zeta\sigma)\}$,
  - $\pi(q, \lambda, \epsilon) \Rightarrow (q, \beta\alpha)$, por cada producción de la forma $\lambda \rightarrow \alpha\beta$,
  - $\pi(q, \zeta, \epsilon) = \{(t, \zeta)\}$,
  - $\pi(q, \lambda, a) \Rightarrow (q, \epsilon)$, por cada producción de la forma $\lambda \rightarrow a$.

No es difícil ver que $L(\mathcal{M}) = L(\Gamma)$.

Lema del Bombeo para lenguajes LDC

- Sea L un lenguaje libre de contexto. Entonces, existe un entero $N > 0$ que depende únicamente de L, con la propiedad de que si $z \in L$ y $|z| > N$, entonces z puede factorizarse como $z = uvwxy$ de manera que

  1. $|vx| \geq 1$,

  2. $|vwx| < N$,

  3. $uv^m wx^m y \in L$ para todo $m \geq 0$.

- Ejemplo: mostrar que el lenguaje $L = \{a^n b^n a^n \ : \ n \geq 1\}$ no es libre de contexto.

Ejemplos de lenguajes no LDC

- $\left\{w^2 \ : \ w \in \{a,b\}^+\right\}$
- $\{a^{n^2} \ : \ n \geq 1\}$
- $\{a^p \ : \ p \text{ es primo}\}$
- $\left\{w \in \{a,b,c\}^+ \ : \ |w|_a = |w|_b = |w|_c\right\}$
- $\left\{ww^Rw \ : \ w \in \{a,b\}^*\right\}$

Propiedades de cerradura de los lenguajes LDC

1. Si L_1 y L_2 son LDC, entonces $L_1 \cup L_2$ y $L_1 L_2$ también lo son.

2. Si L es LDC, entonces L^* también lo es.

3. Si L_1 y L_2 son LDC, entonces $L_1 \cap L_2$ no es necesariamente LDC. Ejemplo:
   - $L_1 = \{a^m b^m \ : \ m \geq 1\}$ es LDC
   - $L_2 = \{b^n a^n \ : \ n \geq 1\}$ es LDC
   - $L_3 = L_1 \cdot a^+ = \{a^m b^m a^n \ : \ m, n \geq 1\}$ es LDC
   - $L_4 = a^+ \cdot L_2 = \{a^m b^n a^n \ : \ m, n \geq 1\}$ es LDC
   - Sin embargo, $L_3 \cap L_4 = \{a^n b^n a^n\}$ no es LDC.

4. Si $L \subseteq A^*$ es LDC, entonces $A^* - L$ no es necesariamente LDC (usar leyes de DeMorgan).

5. Si L_1 es regular y L_2 es LDC, entonces $L_1 \cap L_2$ es LDC.

Ejercicios

1. Para cada uno de los siguientes lenguajes, encontrar un ADP que lo reconozca:

   (a) $\{a^n b^{2n} \ : \ n \geq 1\}$

   (b) $\{a^p b^{p+q} a^q \ : \ p, q \geq 1\}$

   (c) $\{a^m b^n \ : \ m > n \geq 1\}$

   (d) $\left\{wa^{|w|} \ : \ w \in \{a, b\}^+\right\}$

2. Describir el lenguaje reconocido por el siguiente ADP:

$$\begin{array}{rcl@{\qquad\qquad}rcl} (i, a, \zeta) & = & \{(i, \zeta\alpha^2)\}, & (i, a, \alpha) & = & \{(i, \alpha^3)\}, \\ (i, b, \alpha) & = & \{(q, \alpha^2)\}, & (q, b, \alpha) & = & \{(q, \alpha^2)\}, \\ (q, a, \alpha) & = & \{(q', 1)\}, & (q', a, \alpha) & = & \{(q', 1)\}, \\ (q', 1, \zeta) & = & \{(t, \zeta)\}. & & & \end{array}$$

Unidad V

Máquinas de Turing

Introducción

- Podemos pensar en un autómata (de estado finito o de pila) como una máquina que escanea una palabra $w = a_1 a_2 \ldots a_n$ impresa en una cinta:

$$\begin{array}{c} \begin{array}{cccc} i & & & \\ \downarrow & & & \end{array} \\ \hline \begin{array}{cccc} a_1 & a_2 & \ldots & a_n \end{array} \\ \hline \end{array}$$

  donde la máquina está en el estado inicial i, escaneando el símbolo más a la izquierda.

- Tanto los autómatas de estado finito como los autómatas de pila, escanean un símbolo en la cinta, y se mueven a la derecha, posiblemente cambiando de estado (y en el caso de los ADP's, haciendo posibles ajustes a la pila).

Limitantes de los autómatas

- Los autómatas de estado finito se ven limitados en los siguientes aspectos:

  1. Solamente pueden escanear la cinta hacia la derecha, siendo imposible regresar a escanear símbolos anteriores.

  2. Solamente pueden leer el contenido de la cinta, pero no modificarlo.

- Los autómatas de pila pueden modificar el contenido de la pila (la cual puede verse también como una cinta), pero solamente en un extremo. Tampoco pueden recorrer la cinta más que hacia la derecha.

Máquinas de Turing

- **Definición formal:** Una *máquina de Turing* (MT) es una séptupla $\mathcal{T} = (Q, A, M, \Delta, \theta, i, T)$, donde
  - Q es un conjunto finito de estados.
  - A es un alfabeto finito.
  - $M \supseteq A$ es el conjunto finito de *símbolos de la cinta* de $\mathcal{T}$.
  - $\Delta \in M - A$ es el símbolo blanco.
  - $i \in Q$ es el estado inicial.
  - $T \subseteq Q$ es un conjunto no-vacío de estados terminales.
  - θ es una función parcial de $Q \times M$ a $Q \times M \times \{L, R\}$, donde L y R indican *izquierda* y *derecha*, respectivamente.

Descripciones instantáneas

- Una *descripción instantánea* (DI) es una palabra $w_1 q w_2$, donde $w_1 \in M^*$, $w_2 \in M^+$, y $q \in Q$, e indica que el contenido de la cinta es $w_1 w_2$, y que la MT se encuentra en el estado q, escaneando el primer símbolo de w_2.

- Si $w_1 = a_1 a_2 \ldots a_m$, y $w_2 = b_1 b_2 \ldots b_n$, entonces la DI $w_1 q w_2$ puede representarse con el siguiente diagrama:

$$\begin{array}{cccccccc} & & & & q & & & \\ & & & & \downarrow & & & \\ \hline a_1 & a_2 & \ldots & a_m & b_1 & b_2 & \ldots & b_n \\ \hline \end{array}$$

Cómputos

- Consideremos la DI $w_1 q w_2$, donde $w_1 = w_1'\alpha$ y $w_2 = \beta w_2'$ (con $\alpha, \beta \in M$). Entonces, el comportamiento de la máquina de Turing puede ser uno de los siguientes:

  1. Si $\theta(q, \beta) = (q', \beta', R)$, entonces
  $$w_1'\alpha q \beta w_2' \longrightarrow w_1'\alpha\beta' q' w_2'.$$
  2. Si $\theta(q, \beta) = (q', \beta', L)$, entonces
  $$w_1'\alpha q \beta w_2' \longrightarrow w_1' q'\alpha\beta' w_2'.$$
  3. Si q se encuentra cerca del final de la DI y la máquina se mueve hacia la derecha de la cinta, entonces se insertan símbolos blancos conforme sea necesario. Por ejemplo, si $\theta(q, \beta) = (q', \beta', R)$, entonces
  $$w_1 q \beta \longrightarrow w_1 \beta' q' \Delta.$$

- Si dos DI's pueden conectarse mediante una secuencia de transiciones como las mostradas arriba, entonces a esa secuencia se le llama un *cómputo* en $\mathcal{T}$.

Lenguaje reconocido por una MT

- Si $w \in A^*$, entonces decimos que w es reconocida por $\mathcal{T}$ si existe un cómputo

$$iw \to \ldots \to z_1 t z_2,$$

donde i es el estado inicial, t es un estado terminal, $z_1 \in M^*$, y $z_2 \in M^+$.

- El lenguaje $L(\mathcal{T})$ reconocido por $\mathcal{T}$ es el conjunto de todas las palabras $w \in A^*$ reconocidas por $\mathcal{T}$.

Ejemplo

- Considerar la MT $\mathcal{T} = (Q, A, M, \Delta, \theta, i, T)$ con $Q = \{i, q_1, q_2, q_3, q_4, t\}$, $M = \{a, b, \Delta, \lambda, \mu\}$, $A = \{a, b\}$, $T = \{t\}$, y θ dada por

$$\begin{array}{rclrcl}
\theta(i,a) & = & (q_1,\lambda,R), & \theta(q_3,\mu) & = & (q_3,\mu,L),\\
\theta(q_1,a) & = & (q_1,a,R), & \theta(q_3,a) & = & (q_3,a,L),\\
\theta(q_1,\mu) & = & (q_2,\mu,R), & \theta(q_3,\lambda) & = & (i,\lambda,R),\\
\theta(q_2,b) & = & (q_3,\mu,L), & \theta(i,\mu) & = & (q_4,\mu,R),\\
\theta(q_2,\mu) & = & (q_3,\mu,L), & \theta(q_4,\mu) & = & (q_4,\mu,R),\\
\theta(q_1,b) & = & (q_3,\mu,L), & \theta(q_4,\Delta) & = & (t,\Delta,R).
\end{array}$$

- Verificar que $L(\mathcal{T}) = \{a^n b^n \ : \ n \geq 1\}$.

Ejercicios

1. Construír máquinas de Turing que reconozcan los siguientes lenguajes:

   (a) $L = \{ww^R \ : \ w \in \{a,b\}^*\}$.

   (b) $L = \{w \in \{a,b\}^+ \ : \ |w|_a = |w|_b\}$.

   (c) $L = \{a^n b^n a^n \ : \ n \geq 1\}$.

   (d) $L = \{w \in \{a,b,c\}^+ \ : \ |w|_a = |w|_b = |w|_c\}$.

   Notar que los lenguajes (c) y (d) no son libres de contexto.

2. Construír una máquina de Turing que reconozca el lenguaje

   $$L = \{a^m b^n \ : \ m \text{ es divisor de } n\}.$$

   Sugerencia: por cada a encontrada, cambiarla por λ y luego buscar una b y cambiarla por μ. Una vez que no haya mas a's, verificar si ya no quedan b's. En ese caso, aceptar la palabra. De otra manera, cambiar todas las λ's por a's y comenzar nuevamente.

Máquinas de Turing como procesadores de texto

- Si consideramos la cinta de una MT como una cadena de caracteres, entonces uno puede considerar la MT como un procesador de texto que realiza alguna tarea específica.

- Por ejemplo, es relativamente sencillo construír máquinas de Turing que realicen los siguientes procesos:

  1. Reemplazar un caracter por otro
  2. Reemplazar una sección del texto por otra de la misma longitud
  3. Insertar un símbolo, desplazando el texto hacia la derecha
  4. Eliminar un símbolo, desplazando el texto siguiente a la izquierda
  5. Duplicar una sección del texto
  6. Verificar si dos palabras son iguales

Clase de lenguajes reconocidos por las MT

- Es fácil demostrar que cualquier lenguaje regular puede ser reconocido por una MT.

- También es posible demostrar que una máquina de Turing no-determinista tiene la misma capacidad de cómputo que una MT determinista.

- Utilizando lo anterior, es posible demostrar que cualquier lenguaje libre de contexto puede ser reconocido por una MT (determinista o no determinista).

- De hecho, la clase de lenguajes que son reconocidos por las máquinas de Turing, son precisamente los lenguajes de estructura de frases; es decir, aquellos que pueden ser generados por una gramática de estructura de frases, donde todas las producciones son de la forma:

$$u \rightarrow v, \quad u \in (V - A)^{+}, \; v \in V^{*}.$$

- Los lenguajes reconocidos por las máquinas de Turing también se llaman *recursivamente ennumerables*.

Conjuntos reconocidos por las MT

- Utilizando las subrutinas de procesamiento de textos, es posible realizar algunas operaciones relativamente complejas con una MT. Algunos ejemplos son:

Operación	**Entrada**	**Salida**
Suma	$a^m\Delta a^n\Delta$	$a^{m+n}\Delta$
Resta	$a^m\Delta a^n,\ n \leq m$	$a^{m-n}\Delta$
Producto	$a^m\Delta a^n\Delta$	$a^{mn}\Delta$
División	$a^m\Delta a^n\Delta$	$a^q\Delta a^r\Delta,\ m = nq + r,\ r < n$

Ejercicios

- Diseñe una máquina de Turing que invierta una cadena de texto.

- Describa, en términos de subrutinas de procesamiento de textos, cómo construír máquinas de Turing que reconozcan los siguientes lenguajes:

  1. $L = \{a^{n^2} \ : \ n \geq 1\}$

  2. $L = \{a^p \ : \ p$ es primo$\}$.

Jerarquía de Chomsky

- Sea A un alfabeto finito no vacío, y
  - $\mathcal{F}(A^*)$ el conjunto de todos los lenguajes finitos en A^*,
  - Rat A^* el conjunto de todos los lenguajes regulares en A^*,
  - LDC A^* el conjunto de todos los lenguajes libres de contexto en A^*,
  - RE A^* el conjunto de todos los lenguajes recursivamente ennumerables en A^*,

- Entonces, se cumple la siguiente relación:

$$\mathcal{F}(A^*) \subset \text{Rat } A^* \subset \text{LDC } A^* \subset \text{RE } A^* \subseteq \mathcal{P}(A^*).$$

Conjuntos que no son reconocidos por las MT

- Dado un alfabeto finito no-vacío A, existe una cantidad innumerable de lenguajes en A^*. Esto es debido a que A^* es infinito contable, y por lo tanto, $\mathcal{P}(A^*)$ es infinito incontable.

- Por otra parte, el número de máquinas de Turing que pueden diseñarse es contable (pueden enlistarse primero todas las MT con un estado, luego todas las MT con dos estados, etc).

- Esto significa que existen lenguajes en A^* que no pueden ser reconocidos por una máquina de Turing.

- Estos lenguajes, sin embargo, no son fáciles de encontrar, ya que no pueden ser descritos mediante ningún proceso algorítmico.

Codificación de una MT

- Supongamos que se desea describir completamente una máquina de Turing $\mathcal{T} = (Q, A, M, \Delta, \theta, i, T)$ mediante una cadena en $\{a, b\}^*$; es decir, mediante una codificación binaria.

- Para hacer esto, podemos ordenar los elementos de Q, T, A, y $M - A$ de una manera conveniente:

  1. $Q = \{q_1, q_2, \ldots, q_k, t_1, \ldots, t_l\}$, con $i = q_1$ y $T = \{t_1, \ldots, t_l\}$,

  2. $A = \{a_1, \ldots, a_m\}$, con $a_1 = a$ y $a_2 = b$,

  3. $M - A = \{\alpha_1, \ldots, \alpha_n\}$, con $\alpha_1 = \Delta$.

Codificación de una MT

- A continuación, podemos definir las siguientes funciones:
  - $c_1(q_i) = ba^{2i+1}b$, para $i = 1, \ldots, k$,
  - $c_1(t_j) = ba^{2j}b$, para $j = 1, \ldots, l$,
  - $c_2(a_r) = ba^{2r+1}b$, para $r = 1, \ldots, m$,
  - $c_2(\alpha_s) = ba^{2s}b$, para $s = 1, \ldots, n$,
  - $c_3(L) = bab$,
  - $c_3(R) = ba^2b$.

- Definimos también la función $c : Q \times M \times Q \times M \times \{L, R\} \rightarrow \{a, b\}^*$ como:

$$c(q, \alpha, q', \alpha', X) = c_1(q)bc_2(\alpha)bc_1(q')bc_2(\alpha')bc_3(X).$$

- Notar que la función c codifica una instrucción de la MT en una cadena de a's y b's:

$$\theta(q, \alpha) = (q', \alpha', X) \longrightarrow c(q, \alpha, q', \alpha', X).$$

Codificación de una MT

- Una MT puede definirse completamente mediante una secuencia finita de instrucciones, por lo que podemos pensar en cualquier MT como un elemento del conjunto

$$(Q \times M \times Q \times M \times \{L, R\})^p, \ p \leq 1.$$

- Sea $K : (Q \times M \times Q \times M \times \{L, R\})^+ \rightarrow \{a, b\}^*$ una función que mapea una MT a una palabra en $\{a, b\}^*$ definida como:

$$K(z_1, z_2, \ldots, z_p) = c(z_1)b^2c(z_2)b^2 \ldots b^2c(z_p),$$

  donde $z_i = Q \times M \times Q \times M \times \{L, R\}$.

- K es una función uno-a-uno, de manera que a cada máquina de Turing le corresponde una codificación única bajo K. Esa codificación depende del orden en que se enlisten los estados, símbolos, e instrucciones de la MT.

- Si bien una codificación de una MT es una palabra en $\{a, b\}^*$, ss importante notar que no cualquier palabra en $\{a, b\}^*$ corresponde a la codificación de alguna MT.

Un lenguaje no reconocido por ninguna MT

- Sea $A = \{a, b\}$, y $W \subset A^*$ el conjunto de palabras que son codificación de una máquina de Turing; es decir,

  $$W = \{K(z_1, z_2, \ldots, z_p) \ : \ z_i \in Q \times M \times Q \times M \times \{L, R\}, i = 1, \ldots, p, \ p \geq 1\}.$$

- Para $w \in W$, sea $\mathcal{T}(w)$ la máquina de Turing tal que $K(\mathcal{T}(w)) = w$.

- Uno podría preguntarse si, dada una palabra $w \in W$, la MT $\mathcal{T}(w)$ reconoce w. Decimos que w es una *buena codificación* si

  $$w \in L\left(\mathcal{T}(w)\right).$$

- Sea $G \subset W$ el conjunto de palabras que son buenas codificaciones.

- Entonces, el conjunto $D = A^* - G$; es decir, el complemento de G en A^*, no es recursivamente enumerable.

- La prueba se realiza por contradicción, suponiendo que $D = L(\mathcal{T})$ para alguna MT $\mathcal{T}$ y verificando si una codificación w de $\mathcal{T}$ es buena o no lo es.

Máquinas de Turing universales

- Una máquina de Turing universal $\mathcal{U}$ es una máquina de Turing que toma como entrada una palabra

$$w = w_1 \Delta w_2 \in \{a, b\},$$

donde $w_1 = K(\mathcal{M})$ es la codificación de una máquina de Turing $\mathcal{M}$ con alfabeto A, y w_2 es la codificación de una palabra en A^*; es decir,

$$w_2 = c_2(a_1)c_2(a_2)\ldots c_2(a_n),\ a_1a_2\ldots a_n \in A^*.$$

- La acción de $\mathcal{U}$ consiste en simular la acción de la máquina $\mathcal{M} = \mathcal{T}(w_1)$ cuando la entrada de $\mathcal{M}$ es $a_1a_2\ldots a_n$ (se puede demostrar que existe una máquina de Turing $\mathcal{U}$ es capaz de hacer esto).

- Por lo tanto, una máquina de Turing universal es una máquina de Turing que es "programable" cuyo programa se escribe (de manera codificada) en la cinta.

Máquinas de Turing universales

- Consideremos el lenguaje

$$G = \{w \in \{a, b\} \ : \ w \in L\left(\mathcal{T}(w)\right)\};$$

es decir, el lenguaje que consiste en las codificaciones de todas aquellas máquinas de Turing que aceptan su propia codificación.

- Este lenguaje puede reconocerse mediante una máquina de Turing universal que realice los pasos siguientes:

  1. Duplicar la palabra en la cinta: $w\Delta \longrightarrow w\Delta w\Delta$.
  2. Codificar la segunda palabra en la cinta:

$$w\Delta w\Delta \longrightarrow w\Delta c_2(a_1)c_2(a_2)\ldots c_2(a_n)\Delta,$$

     donde $w = a_1a_2\ldots a_n$.
  3. Simular la máquina descrita por w con entrada w.

- Ya que $D = A^* - G$ no es reconocible por ninguna máquina de Turing, entonces es claro que el complemento de un lenguaje recursivamente enumerable no necesariamente es recursivamente enumerable.

Lenguajes decidibles

- Un lenguaje $L \subset A^*$ es *decidible* si existe una máquina de Turing capaz de decidir, para cada $w \in A^*$, si w pertenece o no a L.

- En otras palabras, un lenguaje $L \subset A^*$ si tanto L como $A^* - L$ son reconocibles por máquinas de Turing.

- Todos los lenguajes libres de contexto son decidibles.

- El lenguaje $G = \{w \in \{a, b\} \ : \ w \in L\left(\mathcal{T}(w)\right)\}$ no es decidible.

- Uno puede construír una máquina de Turing que "decida" un lenguaje decidible L haciendo que ésta escriba una a al final de la cinta si la palabra de entrada es aceptada, o una b si la palabra no es aceptada.

Problema de la parada

- Una máquina de Turing $\mathcal{T}$ con alfabeto $\{a, b\}$ es *autoterminante* si eventualmente se detiene cuando se le da como entrada la palabra $K(\mathcal{T})$ (independientemente de si la palabra es o no aceptada por la máquina).

- Definamos el lenguaje $H \subset \{a, b\}^*$ como

$$H = \{K(\mathcal{T}) \ : \ \mathcal{T} \text{ es autoterminante}\}.$$

- Consideremos el problema de decidir H. De cierta forma, esto implica decidir si una máquina de Turing se detiene cuando se le da como entrada una cadena específica. Debido a esto, al problema de decidir H se le conoce como el *problema de la parada*.

- Puede demostrarse que H no es decidible (suponer que lo es, considerar la MT $\mathcal{T}_H$ que decide H y modificarla para que entre en un ciclo infinito cuando una palabra es aceptada, preguntarse si esta nueva máquina es autoterminante).

Unidad VI

Computabilidad y complejidad computacional

Computabilidad

- Nos interesa ahora el estudio de aquellos procesos que son realizables por una máquina de Turing.

- En general, podemos pensar que cualquier tipo de proceso computacional toma una *entrada* y devuelve una *salida* correspondiente a dicha entrada.

- Por lo tanto, cualquier proceso computacional puede representarse como una función.

- Aunque los dominios y contradominios de las funciones computables pueden ser muy diversos, en general es posible, mediante codificaciones adecuadas, representar cualquier función computable como una **función parcial** de la forma

$$f \ : \ \mathbb{N}^m \ \rightarrow \ \mathbb{N}^n.$$

Funciones iniciales

- Es fácil ver que las siguientes funciones son computables:
  - **Función cero:** $\zeta() = 0$.
  - **Función sucesor:** $\sigma(x) = x + 1$.
  - **Proyecciones:** $\pi_k^m(x_1, x_2, \ldots, x_m) = x_k$.

Combinación y composición

- La **combinación** de dos funciones $f : \mathbb{N}^m \to \mathbb{N}^p$ y $g : \mathbb{N}^m \to \mathbb{N}^q$ es la función $f \times g : \mathbb{N}^m \to \mathbb{N}^{p+q}$, definida como

$$(f \times g)(\bar{x}) = (f(\bar{x}), g(\bar{x})),$$

  es computable si f y g lo son.

- La **composición** de dos funciones $f : \mathbb{N}^m \to \mathbb{N}^p$ y $g : \mathbb{N}^p \to \mathbb{N}^n$ es la función $g \circ f : \mathbb{N}^m \to \mathbb{N}^n$, definida como

$$(g \circ f)(\bar{x}) = g(f(\bar{x})),$$

  es computable si f y g lo son.

Recursividad primitiva

- Una función recursiva primitiva $h : \mathbb{N}^{m+1} \to \mathbb{N}^n$ se define a partir de otras dos funciones $f : \mathbb{N}^m \to \mathbb{N}^n$ y $g : \mathbb{N}^{m+n+1} \to \mathbb{N}^n$ mediante la siguiente fórmula recursiva:

$$\begin{aligned} h(x, 0) &= f(x), \\ h(x, y+1) &= g(x, y, h(x, y)) \end{aligned}$$

- Si f y g son computables, entonces h es también computable.

- **Ejemplo:** La función suma $sum : \mathbb{N}^2 \to \mathbb{N}$ puede definirse como

$$\begin{aligned} sum(x, 0) &= \pi_1^1(x), \\ sum(x, y) &= (\sigma \circ \pi_3^3)(x, y, sum(x, y)) \end{aligned}$$

Funciones recursivas primitivas

- Las **funciones recursivas primitivas** son todas aquellas que pueden construírse a partir de funciones iniciales mediante la aplicación de un número finito de combinaciones, composiciones, y recursiones primitivas.

- Toda función recursiva primitiva es computable y total. En otras palabras, si $f : \mathbb{N}^m \to \mathbb{N}^n$ es recursiva primitiva, entonces el dominio de f es $\mathbb{N}^m$.

- La mayoría de las funciones que se requieren en procesos de cómputo tradicionales son recursivas primitivas.

Ejemplos de funciones recursivas primitivas

Función constante:
$K_c^m(\bar{x}) = c$

$$\begin{aligned} K_0^m(\bar{x}) &= \zeta(), \\ K_c^m(\bar{x}) &= \sigma(K_{c-1}^m(y)) \end{aligned}$$

Producto:
$prod(x, y) = x \cdot y$

$$\begin{aligned} prod(x, 0) &= K_0^1(x), \\ prod(x, y+1) &= sum(x, prod(x, y)). \end{aligned}$$

Potencia:

$$\begin{aligned} pot(x, 0) &= K_1^1(x), \\ pot(x, y+1) &= prod(x, pot(x, y)). \end{aligned}$$

Predecesor:

$$\begin{aligned} pred(0) &= \zeta(), \\ pred(y+1) &= \pi_1^2(y, pred(y)). \end{aligned}$$

Resta no-negativa:
$monus(x, y) = x \dot{-} y$

$$\begin{aligned} monus(x, 0) &= \pi_1^1(x), \\ monus(x, y+1) &= pred(monus(x, y)). \end{aligned}$$

Comparación:
$x == y$

$$eq(x, y) = monus(1, sum(monus(x, y), monus(y, x))).$$

Ejemplos de funciones recursivas primitivas

Funciones características: $\kappa_i(x) = eq(x, K_i^0)$.

Negación lógica: $neg(x) = monus(K_1^0, x)$.

Funciones tabulares:

$$\begin{aligned} f(x) \quad = \quad & K_{y_1}^0() \cdot \kappa_{x_1}(x) + \\ & K_{y_2}^0() \cdot \kappa_{x_2}(x) + \\ & \quad \vdots \\ & K_{y_0}^0 \cdot neg(\kappa_{x_1}(x)) \cdot neg(\kappa_{x_2}(x)). \end{aligned}$$

Cociente:

$$\begin{aligned} coc(0, y) &= \zeta(), \\ coc(x+1, y) &= coc(x, y) + eq(x+1, (coc(x, y) \cdot y) + y). \end{aligned}$$

Función totales no recursivas primitivas

- No toda función total es recursiva primitiva. Algunos ejemplos son:

  1. Es claro que el conjunto F de todas las funciones recursivas primitivas es contable, por lo tanto, pueden enlistarse como $PR = \{f_1, f_2, \ldots\}$. Considerar la función $f : \mathbb{N} \to \mathbb{N}$ tal que $f(n) = f_n(n) + 1$. Claramente f es total ya que está bien definida para toda $n \in \mathbb{N}$. Sin embargo, se puede probar también que $f \notin PR$, y por lo tanto, f no es recursiva primitiva.

  2. La función de Ackermann $A : \mathbb{N}^2 \to \mathbb{N}$ se define recursivamente mediante las siguientes ecuaciones:

$$\begin{aligned} A(0, y) &= y + 1, \\ A(x + 1, 0) &= A(x, 1), \\ A(x + 1, y + 1) &= A(x, A(x + 1, y)). \end{aligned}$$

     Se puede demostrar que A es computable y total, pero no recursiva primitiva (se demuestra que la función $f(n) = A(n, n)$ crece más rápido con respecto a n que cualquier función recursiva primitiva).

Funciones recursivas parciales

- Para obtener una función parcial $f : \mathbb{N}^m \rightarrow \mathbb{N}^n$ (es decir, que no esté necesariamente definida para todo $\bar{x} \in \mathbb{N}^m$), se requiere del *operador de minimalización* μ, cuya acción puede definirse como sigue:

  $$f(\bar{x}) = \mu_y \left[g(\bar{x}, y) == 0 \right],$$

  lo cual significa que $f(\bar{x})$ es el menor valor de y tal que $g(\bar{x}, y) == 0$ y $g(\bar{x}, z)$ está definida para todo $z < y$.

- Ejemplo: La división entera es una función recursiva parcial que puede definirse como sigue:

  $$div(x, y) = \mu_t \left[((x + 1) \dot{-} ((t \cdot y) + y)) == 0 \right].$$

- Si $g(\bar{x}, y)$ es computable, entonces $f(\bar{x}) = \mu_y [g(\bar{x}, y) == 0]$ también lo es.

- Se puede demostrar que la clase de funciones recursivas parciales es la misma que la clase de funciones computables por una máquina de Turing.

Complejidad computacional

-