

Computabilidad y Complejidad

Cursada 2022

Cursada 2022

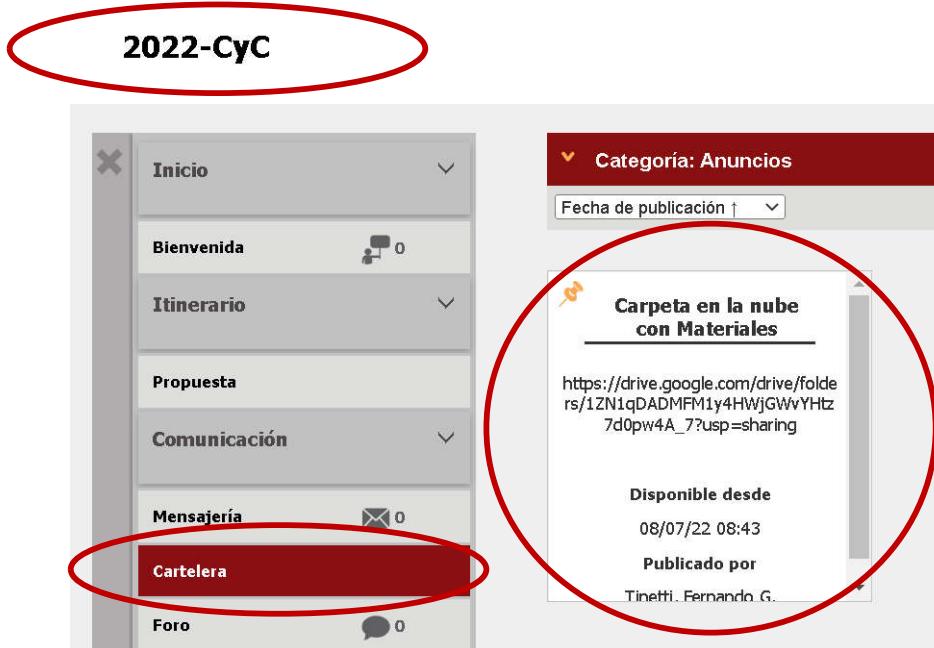
- Contacto directo en clases presenciales
 - Fernando G. Tinetti
 - Leonardo Corbalán
 - Pedro Dal Bianco

Cursada 2022

- Contacto directo en clases presenciales
- Cursada, parcial (2 recuperatorios) y final
- Plataforma Ideas
 - Cartelera - Anuncios - Carpeta en la Nube
 - Disponibilidad de Materiales
 - Fechas
 - Otros

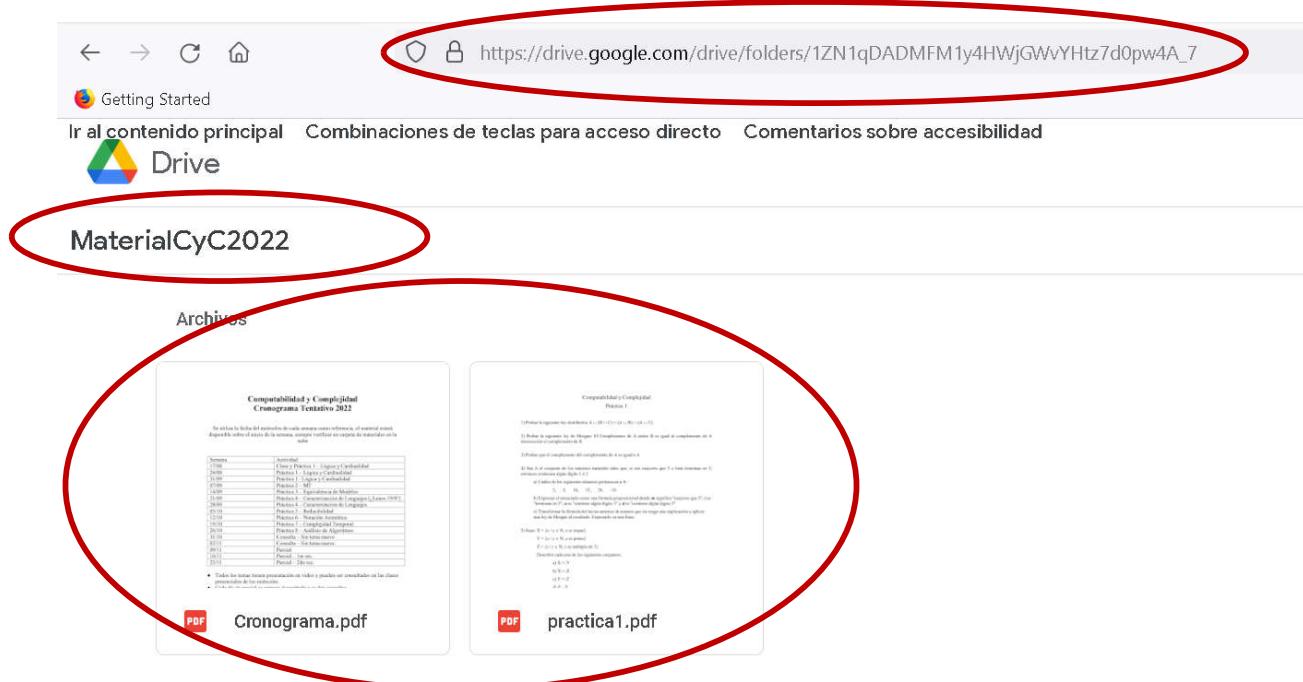
Cursada 2022

- Contacto directo en clases presenciales
- Cursada, parcial (2 recuperatorios) y final
- Plataforma Ideas
 - Cartelera - Anuncios - Carpeta en la Nube



Cursada 2022

- Contacto directo en clases presenciales
- Cursada, parcial (2 recuperatorios) y final
- Plataforma Ideas
 - Cartelera - Anuncios - Carpeta en la Nube



Cursada 2022

- Contenidos
 - Clases de teoría: mayormente videos
 - Generados por la cátedra
 - Comodidad de horario y revisión/repetición
 - “Versus recitado”
 - Quizás “lentos”, a lo sumo ver en 1.5x ó 2x
 - Todo el contenido revisado y orientado a
 - Formalización con detalle
 - Profundidad esperada en cada tema

Cursada 2022

- Contenidos
 - Clases de teoría: mayormente videos
 - Prácticas
 - Enunciados “clásicos”
 - Resolución

Cursada 2022

- Contenidos: Plan LI

TERCER AÑO				
QUINTO SEMESTRE				
SI308	Matemática 3	Semestral	SI102	✉
SI302	Ingeniería de Software 2	Semestral	SI202, SI208	✉
SI306	Conceptos y Paradigmas de Lenguajes de Programación	Semestral	SI203, SI207, SI208	✉
SI307	Orientación a Objetos 2	Semestral	SI206, SI208	✉
SEXTO SEMESTRE				
SI304	Redes y Comunicaciones	Semestral	SI102, SI204, SI208	✉
SI301	Programación Concurrente	Semestral	SI204, SI207, SI208	✉
SI305	Proyecto de Software	Semestral	SI210, SI202, SI203, SI206, SI207, SI208	✉
OI309	Computabilidad y Complejidad	Semestral	SI203, SI308, SI208	✉
CUARTO AÑO				
SÉPTIMO SEMESTRE				
OI401	Teoría de la Computación y Verificación de Programas	Semestral	SI308, SI306	✉

Cursada 2022

- Contenidos: Plan LI

TERCER AÑO				
QUINTO SEMESTRE				
SI308	Matemática 3	Semestral	SI102	✉
SI302	Ingeniería de Software 2	Semestral	SI202, SI208	✉
SI306	Conceptos y Paradigmas de Lenguajes de Programación	Semestral	SI203, SI207, SI208	✉
SI307	Orientación a Objetos 2	Semestral	SI206, SI208	✉
SEXTO SEMESTRE				
SI304	Redes y Comunicaciones	Semestral	SI102, SI204, SI208	✉
SI301	Programación Concurrente	Semestral	SI204, SI207, SI208	✉
SI305	Proyecto de Software	Semestral	SI210, SI202, SI203, SI206, SI207, SI208	✉
OI309	Computabilidad y Complejidad	Semestral	SI203, SI308, SI208	✉
CUARTO AÑO				
SÉPTIMO SEMESTRE				
OI401	Teoría de la Computación y Verificación de Programas	Semestral	SI308, SI306	✉

Cursada 2022

- Contenidos: Plan LI

TERCER AÑO				
QUINTO SEMESTRE				
SI308	Matemática 3	Semestral	SI102	✉
SI302	Ingeniería de Software 2	Semestral	SI202, SI208	✉
SI306	Conceptos y Paradigmas de Lenguajes de Programación	Semestral	SI203, SI207, SI208	✉
SI307	Orientación a Objetos 2	Semestral	SI206, SI208	✉
SEXTO SEMESTRE				
SI304	Redes y Comunicaciones	Semestral	SI102, SI204, SI208	✉
SI301	Programación Concurrente	Semestral	SI204, SI207, SI208	✉
SI305	Proyecto de Software	Semestral	SI210, SI202, SI203, SI206, SI207, SI208	✉
OI309	Computabilidad y Complejidad	Semestral	SI203, SI308, SI208	✉
CUARTO AÑO				
SÉPTIMO SEMESTRE				
OI401	Teoría de la Computación y Verificación de Programas	Semestral	SI308, SI306	✉

Cursada 2022

- Horarios: Plan LI

SEXTO SEMESTRE		
SI304	Redes y Comunicaciones	Semestral
SI301	Programación Concurrente	Semestral
SI305	Proyecto de Software	Semestral
OI309	Computabilidad y Complejidad	Semestral

Cursada 2022

- Horarios: Plan LI

SEXTO SEMESTRE		
SI304	Redes y Comunicaciones	Semestral
SI301	Programación Concurrente	Semestral
SI305	Proyecto de Software	Semestral
OI309	Computabilidad y Complejidad	Semestral

Cursada 2022

- Avance
 - Contenido teórico en video (carpeta nube)
 - Comodidad de horario y revisión/repetición
 - Demostraciones formales detalladas (tiempo)
 - Se pueden ver a 1.25, 1.5 ...

Cursada 2022

- Avance
 - Contenido teórico en video (carpeta nube)
 - Enunciado de prácticas (carpeta nube)
 - Consultas en clases presenciales
 - **Todo** lo que sea necesario
 - Preferentemente 1ro T y luego de P en la misma clase
 - **Consultas, no** desarrollo de P
 - Explicaciones particulares y generales
 - Explicaciones ==> Trabajo siguiente p/ formalizar

Cursada 2022

- Aulas y Horarios
 - Miércoles [y Lunes], 16:00
 - Miércoles, Aula 4
 - [Lunes, Aula 1], cuando en los miércoles se agote el horario y queden consultas pendientes
- Fechas
 - Semestre: 15/8/2022 al 11/2/2023
 - Ver cronograma

Cursada 2022

- Cronograma de la materia
 - Es tentativo
 - Como mínimo una teoría para cada práctica
 - La 1ra práctica tiene varios videos
 - Cortos, por temas autocontenido
 - Mucho es repaso de temas conocidos

Cursada 2022

- Cronograma de la materia
- Cronograma Facultad
- Ideas ≠ Guaraní

Cursada 2022

- Cronograma de la materia
- Cronograma Facultad
- Ideas ≠ Guaraní



Cursada 2022

- Aprovechando y “adaptando” unas charlas
 - Matemática (v1) ==> CyC, demostraciones
 - “Tienen razón todos”
 - Ciudadanía (v2) ==> Profesión
 - Libertad/engaños
- Capitalización + Intuición + Formalización
 - Intuición ==> conjetura... ¿verdad?

Computabilidad y Complejidad

Cursada 2020

“Administrativo”

- Horarios/Aulas
 - Lunes 16:00, Aula 1 (a distancia ...)
 - **Miércoles 16:00**, Aula 4 (a distancia ...)
- Teoría y práctica ambos días excepto que se indique lo contrario (a distancia ...)
- Docentes
 - Fernando G. Tinetti
 - Leonardo Corbalán
 - Pedro Dal Bianco

Aprobación

- Cursada – Parciales
 - No se requiere asistencia
 - Entrega/s de trabajo/s
 - Fechas y formatos de parciales a definir
 - Calendario académico de referencia
 - Corto plazo: avanzar los contenidos
- No hay sistema de promoción

Contenidos en General

- Libros:
 - Introducción a la Teoría de Autómatas, Lenguajes y Computación. Hopcroft y Ullman
 - Teoría de la Computación, Lenguajes Formales, Autómatas y Complejidad. J. Glenn Brookshear
 - Teoría de Autómatas y Lenguajes Formales. Cubero, Moreno y Moriyón Salomón
- Temas:

Algoritmos (específico) ==> Algoritmia, Análisis, Notación Asintótica

Computabilidad
Complejidad } ==> Problemas (general)

Problemas

- Veremos solo **Problemas Computacionales**
 - "Un problema computacional consiste en la especificación de un conjunto de datos de entrada junto con la salida requerida para cada entrada."
 - "Un problema computacional consiste en una caracterización de un conjunto de datos de entrada, junto con una especificación de la salida deseada en base a cada entrada."
 - "We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship."

Instancias

- Instancias de **Problemas Computacionales**
 - "Un problema computacional tiene una o más instancias, valores particulares de los datos de entrada, sobre las cuales se puede ejecutar un algoritmo para resolver el problema"
 - "Un problema computacional tiene una o más instancias (valores particulares que toman los datos de entrada)."
 - "In general, an instance of a problem consists of the input (satisfying whatever constraints are imposed in the problem statement) needed to compute a solution to the problem.“

Ejemplo

- Here is how we formally define the sorting problem:
 - Input: A sequence of n numbers a_1, a_2, \dots, a_n
 - Output: A permutation (reordering), a'_1, a'_2, \dots, a'_n , of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$
- For example, given the input sequence 31, 41, 59, 26, 41, 58, a sorting algorithm returns as output the sequence 26, 31, 41, 41, 58, 59. Such an input sequence is called an instance of the sorting problem.

A Continuación

- Una primera aproximación a Computabilidad
 - Problemas computables
 - Problemas no computables
- Conjuntos
 - Conjunto de Problemas Computacionales PC
 - Conjunto de Problemas Computacionales Computables PCC
 - $PCC \subseteq PC$ sin dudas
 - ¿ $PCC = PC$?
- Conjuntos y propiedades
 - Definidos en términos de lógica

Preliminares

Lógica Proposicional

Secuencia para Práctica 1

- Lógica Proposicional
 - Esta clase/explícacion
- Esquemas Proposicionales
 - “Usa” la lógica proposicional
- Conjuntos
 - “Usa” los esquemas proposicionales
- Cardinalidad de Conjuntos
 - ¿PCC = PC?

Lógica Proposicional

- **Definición.** Una proposición es una oración/afirmación con valor declarativo o informativo, de la cual se puede predicar su verdad o falsedad.
- Ejemplos de proposiciones
 - Hoy es viernes
 - $5 > 25$
 - 7 es primo

Lógica Proposicional

- Ejemplos de expresiones no proposicionales
 - Hola
 - ¿Cómo estás?
 - Quédense quietos
 - ¡Buenísimo!

Lógica Proposicional

- Ejemplos de expresiones no proposicionales
 - Hola
 - ¿Cómo estás?
 - Quédense quietos
 - ¡Buenísimo!

El problema del
lenguaje natural...

Conectivos lógicos

- Operaciones
 - Como si fueran un álgebra (Algebra de Boole)
 - Operaciones: operadores y operandos
 - Negación ==> Unaria
 - Conjunción
 - Disyunción
 - Condicional
 - ...

Conectivos lógicos

- **Negación**
- Dada una proposición p , se denomina la negación de p a otra proposición denotada por $\sim p$ (se lee "no p ") que le asigna el valor de verdad opuesto al de p . Ej:
 - p : Hoy está lloviendo
 - $\sim p$: Hoy no está lloviendo(formal: variable y significado coloquial)

Conectivos lógicos

- **Tabla de verdad de la Negación**

p	$\sim p$
V	F
F	V

Conectivos lógicos

Conjunción

Dadas dos proposiciones p y q , se denomina conjunción de estas proposiciones a la proposición $p \wedge q$ (se lee "p y q"), cuya tabla de verdad es:

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Conectivos lógicos

$\underbrace{5 \text{ es un número impar}}_{p} \text{ y } \underbrace{6 \text{ es un número par}}_{q}$

Por ser p y q ambas verdaderas, la conjunción de ellas es verdadera.

“Hoy es el día 3 de noviembre y mañana es el día de 5 de noviembre”

Esta conjunción es falsa, ya que no pueden ser simultáneamente verdaderas ambas proposiciones.

Conectivos lógicos

Disyunción

Dadas dos proposiciones p y q , la disyunción de las proposiciones p y q es la proposición $p \vee q$ cuya tabla de valor de verdad es:

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Conectivos lógicos

“Tiro las cosas viejas o que no me sirven”

- El sentido de la disyunción es incluyente, pues si tiro algo viejo, y que además no me sirve, la disyunción es verdadera.

Conejativos lógicos

- Condicional o Implicación
- "Si apruebas todas las materias, te dejaré salir el fin de semana".
- p: "Apruebas todas las materias"
- q: "Te dejaré salir el fin de semana"
- Si p es verdad, entonces q también es verdad. Se trata de un enunciado condicional cuya formalización es $p \rightarrow q$, y que se puede leer también como p implica q.
- p es el antecedente (o hipótesis) y q el consecuente (o conclusión).
- Una implicación es siempre verdadera excepto cuando el antecedente es verdadero y el consecuente falso.

Conectivos lógicos

Tabla de verdad del Condicional

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

Conectivos lógicos

Es destacable que el condicional puede ser cierto aunque el consecuente sea falso. Así, si no apruebas todas las materias, pero yo no te permito salir el fin de semana, el condicional "Si apruebas todas las materias, te dejaré salir el fin de semana" es verdadero.

Conectivos lógicos

- En el video con nombre “Condicional” (extraído de <https://www.youtube.com/watch?v=LEzApIgGU9A>) hay una posible explicación detallada de una forma de explicar los valores de verdad. En nuestro caso vamos a usar directamente la tabla como tal. El avance del contenido del video es algo lento, puede ser útil reproducirlo a un factor de velocidad de entre 1.7 y 2 respecto de la velocidad normal.
- “Condicional” e “Implicación” suelen ser utilizados como sinónimos, como en el video mencionado.

Conejitos lógicos

- *Ejercicios:* Determina el valor de las siguientes implicaciones y justifica por qué:
 - a) Si llueve hacia arriba, entonces eres un ser humano.
 - b) Si $2 + 2 = 4$, entonces las ranas tienen pelo.
 - c) Si sabes leer, entonces los círculos son cuadrados.
 - d) Si los burros vuelan, entonces las tortugas saben álgebra

Conejtos lógicos

Equivalencia lógica.

Se dice que dos proposiciones son lógicamente equivalentes, o simplemente **equivalentes**. Si coinciden sus resultados para los mismos valores de verdad. Se indican como $p \Leftrightarrow q$.

Conectivos lógicos

Propiedades del Condicional

Un condicional y su recíproca no son lógicamente equivalentes (distintas tablas de verdad). Se dice que $q \rightarrow p$ es el recíproco de $p \rightarrow q$.

Un condicional y su contrarrecíproca son lógicamente equivalentes (iguales tablas de verdad). El contrarrecíproco del enunciado $p \rightarrow q$ es $\sim q \rightarrow \sim p$

$$p \rightarrow q \Leftrightarrow \sim q \rightarrow \sim p$$

Conectivos lógicos

El bicondicional (o coimplicación)

Ya hemos comprobado que $p \rightarrow q$ no es lo mismo que $q \rightarrow p$. Puede ocurrir, sin embargo, que tanto $p \rightarrow q$ como $q \rightarrow p$ sean verdaderos. Justamente, ello es lo que expresamos con el bicondicional $p \leftrightarrow q$.

En consecuencia, el enunciado $p \leftrightarrow q$ queda definido por el enunciado $(p \rightarrow q) \wedge (q \rightarrow p)$.

Conectivos lógicos

El bicondicional o coimplicador $p \Leftrightarrow q$, que se lee "p si y sólo si q" se define por la siguiente tabla de verdad:

p	q	$p \Leftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

Tautología

- **Tautología.**
- Tautología, es aquella proposición (compuesta) que es cierta para todos los valores de verdad de sus variables.

Tautología

Ejemplo típico de Tautología:

p	q	$\sim p$	$\sim q$	$p \rightarrow q$	$\sim q \rightarrow \sim p$	$(p \rightarrow q) \leftrightarrow (\sim q \rightarrow \sim p)$
F	F	V	V	V	V	V
F	V	V	F	V	V	V
V	F	F	V	F	F	V
V	V	F	F	V	V	V

Tautología

NOTA

Observar que si $p \leftrightarrow q$ es una tautología entonces p y q son lógicamente equivalentes, lo que se puede expresar como $p \Leftrightarrow q$

De la misma manera, cuando $p \rightarrow q$ es una tautología decimos que existe una implicación lógica que la expresamos $p \Rightarrow q$

Tautologías más importantes

1.- Doble negación.

a). $\sim\sim p \Leftrightarrow p$

2.- Leyes conmutativas.

a). $(p \vee q) \Leftrightarrow (q \vee p)$

b). $(p \wedge q) \Leftrightarrow (q \wedge p)$

c). $(p \leftrightarrow q) \Leftrightarrow (q \leftrightarrow p)$

Tautologías más importantes

Tautologías ↪

1.- Doble negación.

a). $\sim\sim p \Leftrightarrow p$

2.- Leyes conmutativas.

a). $(p \vee q) \Leftrightarrow (q \vee p)$

b). $(p \wedge q) \Leftrightarrow (q \wedge p)$

c). $(p \leftrightarrow q) \Leftrightarrow (q \leftrightarrow p)$

Tautologías más importantes

3.- Leyes asociativas.

$$\text{a). } [(p \vee q) \vee r] \Leftrightarrow [p \vee (q \vee r)]$$

$$\text{b). } [(p \wedge q) \wedge r] \Leftrightarrow [p \wedge (q \wedge r)]$$

4.- Leyes distributivas.

$$\text{a). } [p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$$

$$\text{b. } [p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$$

Tautologías más importantes

Tautologías ↔

3.- Leyes asociativas.

$$a). \ [(p \vee q) \vee r] \Leftrightarrow [p \vee (q \vee r)]$$

$$b). \ [(p \wedge q) \wedge r] \Leftrightarrow [p \wedge (q \wedge r)]$$

4.- Leyes distributivas.

$$a). \ [p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$$

$$b). \ [p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$$

Tautologías más importantes

5.- Leyes de idempotencia.

a). $(p \vee p) \Leftrightarrow p$

b). $(p \wedge p) \Leftrightarrow p$

6.- Leyes de Morgan

a). $\sim(p \vee q) \Leftrightarrow(\sim p \wedge \sim q)$

b). $\sim(p \wedge q) \Leftrightarrow(\sim p \vee \sim q)$

Tautologías más importantes

7.- Contrapositiva.

$$\text{a). } (p \rightarrow q) \Leftrightarrow (\neg q \rightarrow \neg p)$$

8.- Implicación.

$$\text{a). } (p \rightarrow q) \Leftrightarrow (\neg p \vee q)$$

$$\text{b). } (p \rightarrow q) \Leftrightarrow \neg(p \wedge \neg q)$$

9.- Equivalencia

$$\text{a). } (p \leftrightarrow q) \Leftrightarrow [(p \rightarrow q) \wedge (q \rightarrow p)]$$

Tautologías más importantes

10.- Adición.

$$\text{a). } p \Rightarrow (p \vee q)$$

11.- Simplificación.

$$\text{a). } (p \wedge q) \Rightarrow p$$

12.- Modus ponens.

$$\text{a). } [p \wedge (p \rightarrow q)] \Rightarrow q$$

Tautologías más importantes

Tautologías →

10.- Adición.

a). $p \Rightarrow (p \vee q)$

11.- Simplificación.

a). $(p \wedge q) \Rightarrow p$

12.- Modus ponens.

a). $[p \wedge (p \rightarrow q)] \Rightarrow q$

Tautologías más importantes

13- Modus tollens.

a). $[(p \rightarrow q) \wedge \neg q] \Rightarrow \neg p$

14.- Transitividad del \leftrightarrow y del \rightarrow

a). ¿Cómo sería?

Las tautologías relacionadas con \leftrightarrow y \Rightarrow permiten “derivar” o “generar” o “tratar” información *nueva* o *distinta*

Tautologías más importantes

13- Modus tollens.

a). $[(p \rightarrow q) \wedge \neg q] \Rightarrow \neg p$

14.- Transitividad del \leftrightarrow y del \rightarrow

a). ¿Cómo sería?

Las tautologías relacionadas con \leftrightarrow y \Rightarrow permiten “derivar” o “generar” o “tratar” información *nueva* o *distinta*

Preliminares

Esquemas Proposicionales

Secuencia para Práctica 1

- Lógica Proposicional
 - 02-Proposicional
- Esquemas Proposicionales
 - **Esta clase/explícacion**
- Conjuntos
 - “Usa” los esquemas proposicionales
- Cardinalidad de Conjuntos
 - ¿PCC = PC?

Esquemas proposicionales

$$x + 2 = 5$$

¿Es una proposición?

Esquemas proposicionales

$$x + 2 = 5$$

¿Es una proposición?

Si $x = 7$ se tiene que $7 + 2 = 5$ es una proposición Falsa

Esquemas proposicionales

- **Definición.** Se llama *esquema proposicional en la indeterminada x* a toda expresión que contiene a x y posee la siguiente propiedad: “Existe por lo menos un nombre tal que la expresión obtenida sustituyendo la indeterminada por dicho nombre, es una proposición”.
- Las indeterminadas suelen llamarse **variables o incógnitas**

Esquemas proposicionales

- Ejemplo: “x es blanca” es un esquema pues existe una constante “esta flor” que en lugar de la variable x produce la siguiente proposición: “Esta flor es blanca”
- Vamos a utilizar símbolos tales como $P(x)$, $Q(x)$, $F(x)$, para designar esquemas de incógnita x.

Operadores Universal y Existencial

Operadores Universal y Existencial

Si se tiene un esquema $P(n)$ puede obtenerse de él una proposición mediante la adjunción de los operadores

UNIVERSAL: $(\forall n):(P(n))$

EXISTENCIAL: $(\exists n):(P(n))$

Esquemas proposicionales

La proposición con el operador universal $(\forall n):(P(n))$ será V si todos los valores posibles de n hacen V la proposición resultante del esquema. Será F en cc.

Esquemas proposicionales

La proposición con el operador universal $(\forall n):(P(n))$ será V si todos los valores posibles de n hacen V la proposición resultante del esquema. Será F en cc.

La proposición con el operador universal $(\exists n):(P(n))$ será V si hay al menos un valor posible de n que hace V la proposición resultante del esquema. Será F en cc.

Operadores Universal y Existencial

Ejercicios

En cada caso decir si se trata de esquemas, en tal caso transformarlo en una proposición y hallar su valor de verdad

$$1) P(n) : n+1 > n$$

$$2) Q(n) : n^2 + 1$$

$$3) R(n) : n^2 - 3n + 2 = 0$$

$$4) S(n) : n \text{ es un número racional}$$

Operadores Universal y Existencial

El alcance del operador llega únicamente al primer esquema, si quisiéramos que alcance a los dos esquemas, tendríamos que poner $(\exists x):(x \text{ es verde} \wedge x \text{ es rojo})$ o sea usaríamos paréntesis.

Se llama **alcance de un operador en x** al esquema más simple que aparece inmediatamente después del operador, salvo que se presenten paréntesis, en cuyo caso deben aplicarse las reglas habituales referentes al uso de paréntesis.

En informática diríamos que los cuantificadores tienen mayor precedencia que lo conectivos lógicos (salvo el \sim)

Negación de los operadores

Negación de los operadores

Sea la siguiente proposición:

$(\forall n)$: n es un número primo,

Vamos ahora a negarla

$\sim(\forall n)$: n es un número primo

$(\exists n)$: n no es un número primo

Negación de los operadores

De lo anterior se puede deducir que son expresiones sinónimas

$$\sim(\forall n): P(n) \text{ y } (\exists n): \sim P(n)$$

De igual manera se obtiene:

$$\sim(\exists n): P(n) \text{ y } (\forall n): \sim P(n)$$

Ejercitación

Ejercicios

Expresar mediante operadores y símbolos las proposiciones

- 1) Todos los hombres son mortales
- 2) Hay algún número que es primo
- 3) Hay honrados y además hay ladrones.
- 4) Hay ladrones o hay honrados
- 5) Hay individuos que son ladrones o comen uvas
- 6) No todos comen uvas

Ejercitación

Expresar en lenguaje corriente las siguientes proposiciones:

- 1) $(\forall x): (x \text{ es metal} \rightarrow x \text{ se funda})$
- 2) $(\forall x): (x \text{ es metal}) \vee \text{el oro se funde}$
- 3) $(\exists x): (x \text{ es cuadrado}) \rightarrow (\exists x): (x \text{ es paralelogramo})$
- 4) $\sim[(\forall x): (x \text{ es hombre} \rightarrow x \text{ es mortal})]$

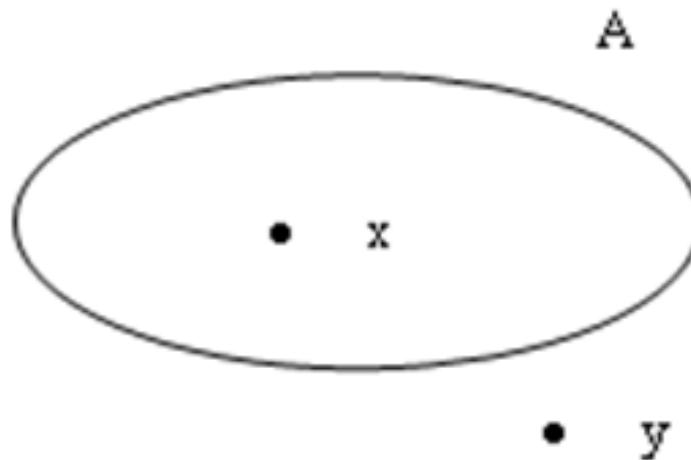
Preliminares – Teoría de Conjuntos Básica

Relación con la Lógica

Secuencia para Práctica 1

- Lógica Proposicional
 - 02-Proposicional
- Esquemas Proposicionales
 - 03-Esquemas
- Conjuntos
 - **Esta clase/explícacion**
- Cardinalidad de Conjuntos
 - ¿PCC = PC?

Teoría de Conjuntos



La idea de conjunto no requiere mucha presentación.

Seguramente estarás familiarizado con gráficos como éste donde se indica que **x** es **un elemento del conjunto A** (lo que de aquí en más escribiremos $x \in A$) y que **y** no es elemento del conjunto A ($y \notin A$).

Conjuntos

- Se podría definir un conjunto como una colección o agrupación de objetos que es tratada como una unidad.

Conjuntos

- Se podría definir un conjunto como una colección o agrupación de objetos que es tratada como una unidad.
- Cada objeto del conjunto se denomina *elemento*, con estas características:
 - Indivisible desde el punto de vista de su pertenencia al conjunto
 - Sin repetición
 - Sin orden, no hay “1ro.”, ni “mayor”, ni ... etc.
- En particular, una colección sin elementos es el conjunto vacío

Conjuntos

Elementos de conjuntos

- \emptyset : Sin elementos
- $A = \{\emptyset\}$ ¿Cuántos elementos tiene?

Conjuntos

Elementos de conjuntos

- \emptyset : Sin elementos
- $A = \{\emptyset\}$ ¿Cuántos elementos tiene?
 - $A = \{\emptyset\} \neq \emptyset$

Conjuntos

Elementos de conjuntos

- $\emptyset = \{ \}$: Conjunto vacío, sin elementos
- $A = \{\emptyset\}$, tiene 1 elemento, que es el \emptyset
- $A = \{1, \{3,-1\}, \emptyset\}$: 3 elementos: 1, $\{3,-1\}$, \emptyset
Ni 3 ni -1 son elementos de A, son
elementos del conjunto $\{3,-1\}$, que es un
elemento del conjunto A.

Conjuntos

Formas de expresión de conjuntos

- Diagrama de Venn (gráfico)
- Enumeración $A = \{1, 2, 3, 4, 5\}$

Para los conjuntos infinitos ...

- $N = \{x / x \text{ es un número natural}\}$ es el conjunto de los números naturales
- $P = \{n / n \text{ es par}\} = \{n / n = 2k \text{ con } k \in N\}$

Conjunto Vacío

Definición. Conjunto Vacío: Es el conjunto que no tiene elementos. Se denota \emptyset o $\{\}$

$$(\forall x):(x \notin \emptyset)$$

Conjunto Vacío

Definición. Conjunto Vacío: Es el conjunto que no tiene elementos. Se denota \emptyset o $\{\}$

$$(\forall x):(x \notin \emptyset)$$

Notar que \emptyset es el único conjunto que hace verdadera esa proposición, que está definida con el cuantificador \forall sobre el esquema proposicional $P(x) : x \notin \emptyset$

Conjunto Vacío

Definición. Conjunto Vacío: Es el conjunto que no tiene elementos. Se denota \emptyset o $\{\}$

$$(\forall x):(x \notin \emptyset)$$

A partir de este punto se comenzará a usar lógica para definir características, propiedades, operaciones, etc.

SubConjuntos

SubConjuntos

Definición. **Subconjunto:** Sean A y B dos conjuntos, se dice que A es un subconjunto de B, o que A es parte de B, si todo elemento de A es también elemento de B.

$$(\forall x):(x \in A \rightarrow x \in B)$$

También se dice que A está incluido o es igual a B y se denota $A \subseteq B$

Si B tiene al menos un elemento que no pertenece a A se dice que A es un subconjunto propio de B o que está incluido de manera propia en B y se denota $A \subset B$

$$(\forall x):(x \in A \rightarrow x \in B) \wedge (\exists x):(x \in B \wedge x \notin A)$$

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a,b\}$$

$$\emptyset \subseteq \emptyset$$

$$\emptyset \subset \emptyset$$

$$A \subset A$$

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a,b\}$$

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a,b\}$$

Hay que evaluar si la definición de inclusión es V o F

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a,b\}$$

Hay que evaluar si la definición de inclusión es V o F

Definición. Subconjunto: Sean A y B dos conjuntos, se dice que A es un subconjunto de B, o que A es parte de B, si todo elemento de A es también elemento de B.

$$(\forall x):(x \in A \rightarrow x \in B)$$

También se dice que A está incluido o es igual a B y se denota $A \subseteq B$

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a, b\}$$

Hay que evaluar si la definición de inclusión es V o F

En particular: $(\forall x):(x \in A \rightarrow x \in B)$

Donde $A = \emptyset$ y $B = \{a, b\}$

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a, b\}$$

Hay que evaluar si la definición de inclusión es V o F

En particular: $(\forall x):(x \in A \rightarrow x \in B)$

Donde $A = \emptyset$ y $B = \{a, b\}$

Con lo cual $x \in A$ en realidad es $x \in \emptyset$, que es siempre F,

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a, b\}$$

Hay que evaluar si la definición de inclusión es V o F

En particular: $(\forall x):(x \in A \rightarrow x \in B)$

Donde $A = \emptyset$ y $B = \{a, b\}$

Con lo cual $x \in A$ en realidad es $x \in \emptyset$, que es siempre F, por lo tanto, el condicional $(x \in A \rightarrow x \in B)$ será siempre V, por la tabla de verdad del condicional,

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a, b\}$$

Hay que evaluar si la definición de inclusión es V o F

En particular: $(\forall x):(x \in A \rightarrow x \in B)$

Donde $A = \emptyset$ y $B = \{a, b\}$

Con lo cual $x \in A$ en realidad es $x \in \emptyset$, que es siempre F, por lo tanto, el condicional $(x \in A \rightarrow x \in B)$ será siempre V, por la tabla de verdad del condicional, y esto a su vez implica que $(\forall x):(x \in A \rightarrow x \in B)$ es V, de hecho independientemente de B.

Ejercicios

Ejercicios: Responder si son verdaderas o falsas las siguientes afirmaciones:

$$\emptyset \subseteq \{a, b\}$$

Hay que evaluar si la definición de inclusión es V o F

En particular: $(\forall x):(x \in A \rightarrow x \in B)$

Donde $A = \emptyset$ y $B = \{a, b\}$

Con lo cual $x \in A$ en realidad es $x \in \emptyset$, que es siempre F, por lo tanto, el condicional $(x \in A \rightarrow x \in B)$ será siempre V, por la tabla de verdad del condicional, y esto a su vez implica que $(\forall x):(x \in A \rightarrow x \in B)$ es V, de hecho independientemente de B.

Por lo tanto, $\emptyset \subseteq \{a, b\}$ es V

Ejercicios

En todos los casos, usar las definiciones. Resumen de la demostración anterior:

$$\emptyset \subseteq \{a, b\}, \text{ Def. de inclusión: } A \subseteq B \Leftrightarrow (\forall x):(x \in A \rightarrow x \in B)$$

$$A = \emptyset, B = \{a, b\}$$

Debería demostrarse que $(\forall x):(x \in A \rightarrow x \in B)$, y más específicamente $(x \in A \rightarrow x \in B)$

$x \in A$ en este caso es $x \in \emptyset$, y por la definición de \emptyset , $(\forall x):(x \notin \emptyset)$, es decir que $x \in \emptyset$ es F para todo x.

Por lo anterior, y por el operador condicional, se tiene que $x \in A \rightarrow x \in B$ es V para cualquier (para todo) valor posible de x, independientemente del conjunto B que sea, y en particular para $B = \{a, b\}$

Por lo que queda demostrado que $\emptyset \subseteq \{a, b\}$

Igualdad

Igualdad

Definición. Igualdad entre conjuntos: Se dice que A es igual a B, se denota $A = B$, si posseen los mismos elementos

$$(\forall x):(x \in A \leftrightarrow x \in B)$$

Observación: $(A = B) \Leftrightarrow ((A \subseteq B) \wedge (B \subseteq A))$

Igualdad

Definición. Igualdad entre conjuntos: Se dice que A es igual a B, se denota $A = B$, si posse los mismos elementos

$$(\forall x):(x \in A \leftrightarrow x \in B)$$

Observación: $(A = B) \Leftrightarrow ((A \subseteq B) \wedge (B \subseteq A))$

Por definición/equivalencia lógica del bicondicional
con la conjunción de los condicionales...

Operaciones básicas

Si bien vamos a utilizar casi siempre las ideas de conjuntos y demostraciones con conjuntos, ahora vamos más directamente orientados hacia los problemas computacionales, para tener una idea en cuanto a si se pueden resolver o no

Operaciones básicas

Si bien vamos a utilizar casi siempre las ideas de conjuntos y demostraciones con conjuntos, ahora vamos más directamente orientados hacia los problemas computacionales, para tener una idea en cuanto a si se pueden resolver o no

- $\text{PCC} \subseteq \text{PC}$ sin dudas
- ¿ $\text{PCC} = \text{PC}$?

Operaciones básicas

Operaciones básicas de Conjuntos

Intersección: $A \cap B = \{x / x \in A \wedge x \in B\}$

Unión: $A \cup B = \{x / x \in A \vee x \in B\}$

Diferencia: $A - B = \{x / x \in A \wedge x \notin B\}$

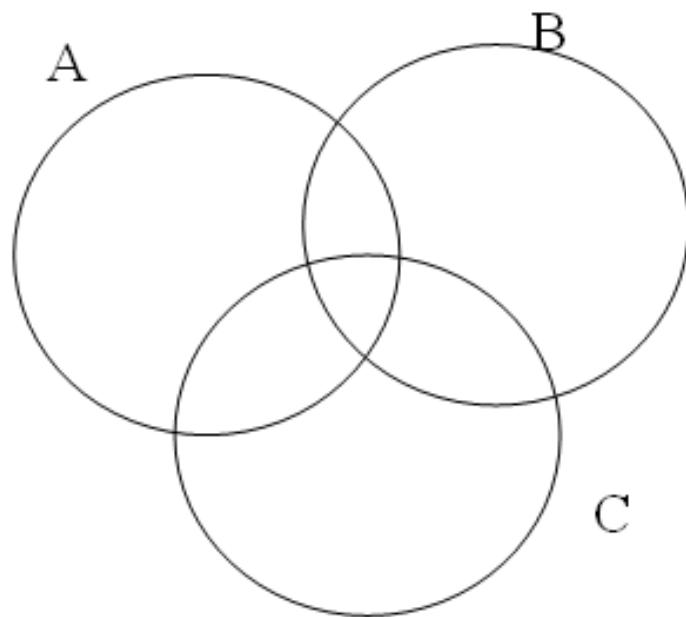
Complemento: Si $A \subseteq B$, $\bar{A}_B = B - A$

Si B es el conjunto universal se denota \bar{A}
y se puede escribir como $\{ x / x \notin A \}$

Ejercicios

1) En el diagrama que sigue indicar

- a) $(A \cup B) - (A \cup C)$ b) $(A \cap B) \cup (C - B)$ c) $(A - B) \cap C$



2) Un subconjunto X de números naturales tiene 12 múltiplos de 4, 7 múltiplos de 6, 5 múltiplos de 12 y 8 números impares. ¿Cuántos elementos tiene X? Grafique el diagrama de Venn

Equivalencias de conjuntos

Leyes conmutativas:

$$A \cup B = B \cup A \text{ y } A \cap B = B \cap A$$

Leyes distributivas:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

Leyes de Morgan:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

¿Por qué se llamarán leyes de Morgan?

Ley de doble complemento:

$$\overline{\overline{A}} = A$$

Ejercitación

Probar que $(A - B) \cap (A - C) = A - (B \cup C)$

Dem.

$$\begin{aligned} & x \in (A - B) \cap (A - C) \\ \Leftrightarrow & x \in (A - B) \wedge x \in (A - C) && (\text{def. de intersec.}) \\ \Leftrightarrow & (x \in A \wedge x \notin B) \wedge (x \in A \wedge x \notin C) && (\text{def. de resta}) \quad \begin{matrix} \text{Asoc.} \\ \text{Conn.} \\ \text{simp.} \end{matrix} \\ \Leftrightarrow & x \in A \wedge \sim(x \in B) \wedge \sim(x \in C) && (\text{Lóg. Prop.}) \\ \Leftrightarrow & x \in A \wedge \sim(x \in B \vee x \in C) && (\text{Morgan}) \\ \Leftrightarrow & x \in A \wedge \sim(x \in (B \cup C)) && (\text{def. de unión}) \\ \Leftrightarrow & x \in A \wedge x \notin (B \cup C) && (\text{def. de } \notin) \\ \Leftrightarrow & x \in A - (B \cup C) && (\text{def. de resta}) \end{aligned}$$

Por lo tanto $(A - B) \cap (A - C) = A - (B \cup C)$

Producto cartesiano

Producto cartesiano

Producto cartesiano: $A \times B = \{ (x, y) / x \in A \wedge y \in B \}$

$A \times A$ se denota A^2 en gral A^n representa el conjunto de n-tuplas de elementos de A

Producto cartesiano

Producto cartesiano: $A \times B = \{ (x, y) / x \in A \wedge y \in B \}$

$A \times A$ se denota A^2 en gral A^n representa el conjunto de n-tuplas de elementos de A

¿Cómo sería un producto cartesiano con \emptyset ?

$$\emptyset \times A, B \times \emptyset, \emptyset \times \emptyset$$

Definición de \emptyset , $(\forall x):(x \notin \emptyset)$

Producto cartesiano

Producto cartesiano: $A \times B = \{ (x, y) / x \in A \wedge y \in B\}$

$A \times A$ se denota A^2 en gral A^n representa el conjunto de n-tuplas de elementos de A

Las funciones establecen una relación entre dos conjuntos, donde $f: A \rightarrow B$ define que un elemento de A (dominio) está relacionado a lo sumo con 1 y solo 1 elemento de B (codominio).

Por lo tanto, una función $f: A \rightarrow B$ se puede especificar como un subconjunto de $A \times B$

¿Por qué una función $f: A \rightarrow B$ necesariamente será un subconjunto propio del producto cartesiano $A \times B$?

Lo más usual en funciones es que sean totales, es decir que estén definidas para todo el dominio. En “general”, son parciales.

Producto cartesiano

$f : N \rightarrow N, f(n) = 2n$

$\{(0, 0), (1, 2), (2, 4), (3, 6), \dots\} = \{(n, 2n), n \in N\}$

$f : R \rightarrow R, f(x) = -(x^{1/2})$

$x^{1/2}$ no es una función

$\{(0, 0), (4, -2), (9, -3), (16, -4), \dots\}$, no está definida para $x < 0$

(ejemplos solamente de N , pero hay pares de reales (x, y) con $x \geq 0$)

$f : R^+ \rightarrow R, f(x) = +(x^{1/2})$

dominio de reales > 0

$\{(4, 2), (9, 3), (16, 4), \dots\}$

$(0, 0)$ no pertenece a este conjunto, ni
ningún par (x, y) con $x \leq 0$

Conjunto de partes

Conjunto de partes

Definición. Conjunto de partes. Si A es un conjunto, se llama conjunto de partes de A o conjunto potencia de A y se denota $\rho(A)$ (en algunos textos también aparece como 2^A) al conjunto formado por todos los subconjuntos de A .

$$\rho(A) = \{B / B \subseteq A\}$$

Ejemplo: $A = \{a, b\}$

$$\rho(A) = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$$

Conjunto de partes

¿ $\rho(\emptyset)$?

$$\rho(\emptyset) = \{\emptyset\} = \{B / B \subseteq \emptyset\}$$

¿ $\rho(\{\emptyset\})$?

¿ $\rho(\{\{\emptyset\}, \emptyset\})$?

En todos los casos, conviene identificar los elementos de los conjuntos, en el caso anterior, se puede hacer como “intermedio”

$A = \{\{\emptyset\}, \emptyset\}$, denotando $a = \{\emptyset\}$ y $b = \emptyset$, y quedaría

$\rho(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ y luego reemplazar a y b

$$\rho(\{a, b\}) = \{\emptyset, \{\{\emptyset\}\}, \{\emptyset\}, \{\{\emptyset\}, \emptyset\}\}$$

Práctica 1

Los conceptos, propiedades y formas de demostraciones vistos hasta este punto son los necesarios y suficientes para resolver la primera parte de la Práctica 1

Preliminares – Teoría de Conjuntos, Cardinalidad

Comparaciones de “Tamaño”

Secuencia para Práctica 1

- Lógica Proposicional
 - 02-Proposicional
- Esquemas Proposicionales
 - 03-Esquemas
- Conjuntos
 - 04-Conjuntos
- Cardinalidad de Conjuntos
 - ¿PCC = PC?

Cardinalidad

Cardinalidad

Definición. Si A es un conjunto finito, se denomina cardinalidad o tamaño de A , y se denota $|A|$ al número de elementos del conjunto A .

Ejercicio para el lector: Demostrar por inducción que si A es un conjunto finito

$$|A| = n \Rightarrow |\rho(A)| = 2^n$$

Nota: Está claro que si A es un conjunto finito, entonces $|A| < |\rho(A)|$

¿Pero qué ocurre si A es un conjunto infinito?

Cardinalidad

$$|A| = n \Rightarrow |\rho(A)| = 2^n$$

Por inducción en n: caso base) n = 0, Hi) n = h, Dem.) n = n+1

Caso Base) n = 0, A = \emptyset

$\rho(A) = \rho(\emptyset) = \{\emptyset\} \Rightarrow |\rho(A)| = |\rho(\emptyset)| = |\{\emptyset\}| = 1 = 2^0$, queda demostrado

$$\text{Hi)} |A| = n \Rightarrow |\rho(A)| = 2^n$$

$$\text{Dem.) } |A'| = n+1 \Rightarrow |\rho(A')| = 2^{n+1}$$

$$A' = \{a_1, \dots, a_n, a_{n+1}\} = \{a_1, \dots, a_n\} \cup \{a_{n+1}\} = A \cup \{a_{n+1}\}$$

$$\rho\{A\} = \{\emptyset, \{a_1\}, \dots, \{a_n\}, \{a_1, a_2\}, \dots, \{a_1, \dots, a_n\}\}, \quad |\rho\{A\}| = 2^n \text{ por Hi}$$

Todos los subconjuntos de A son subconjuntos A', porque todos los elementos de A son elementos de A' (subconjunto de A = elem. de $\rho\{A\}$)
Para determinar la cantidad de elementos de $\rho\{A'\}$ se deben agregar (“faltan”) los subconjuntos de A' que contienen al elemento a_{n+1}

Cardinalidad

Dem.) $|A'| = n+1 \Rightarrow |\rho(A')| = 2^{n+1}$

$$A' = \{a_1, \dots, a_n, a_{n+1}\} = \{a_1, \dots, a_n\} \cup \{a_{n+1}\} = A \cup \{a_{n+1}\}$$

$$\rho\{A\} = \{\emptyset, \{a_1\}, \dots, \{a_n\}, \{a_1, a_2\}, \dots, \{a_1, \dots, a_n\}\}, \quad |\rho\{A\}| = 2^n \text{ por Hi}$$

Todos los subconjuntos de A son subconjuntos A', porque todos los elementos de A son elementos de A' (subconjunto de A = elem. de $\rho\{A\}$)
Para determinar la cantidad de elementos de $\rho\{A'\}$ se deben agregar (“faltan”) los subconjuntos de A' que contienen al elemento a_{n+1} :

$$\begin{array}{ccccccc} \{\emptyset, & \{a_1\}, & \dots, & \{a_n\}, & \{a_1, a_2\}, & \dots, & \{a_1, \dots, a_n\}\} \\ \cup \{a_{n+1}\} & \cup \{a_{n+1}\} & & \cup \{a_{n+1}\} & \cup \{a_{n+1}\} & & \cup \{a_{n+1}\} \end{array}$$

Se puede pensar como que los subconjuntos que “faltan” se pueden formar con los subconjuntos de $A \cup \{a_{n+1}\}$

Por lo tanto, la cantidad total sería el doble de elementos de $\rho\{A\}$, y como $|\rho\{A\}| = 2^n$

Se tiene que $|\rho\{A'\}| = 2 \times 2^n = 2^{n+1}$, que es lo que se debía demostrar

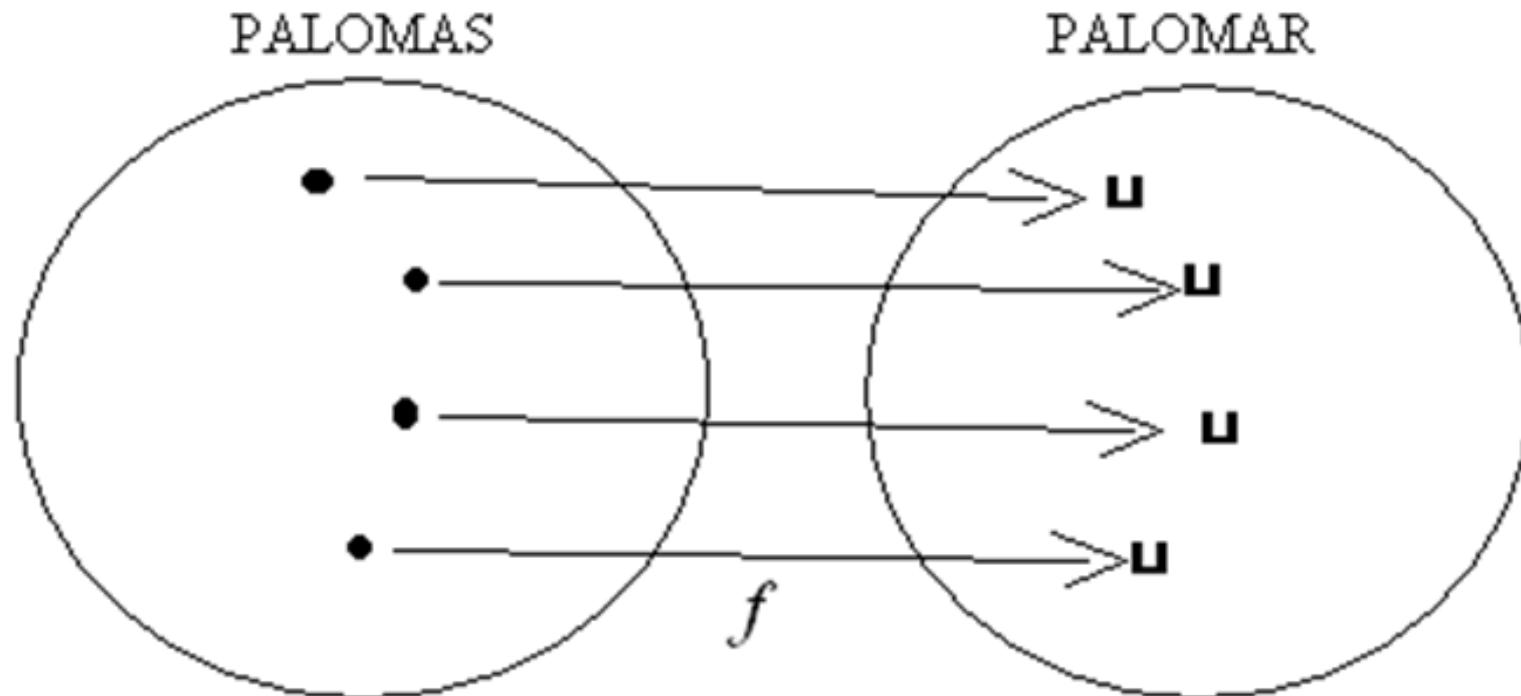
Cardinalidad de conjuntos infinitos

- ¿Cuántos elementos hay en \mathbb{N} ?
- Infinito no es un número de nuestro sistema contable normal
- La cardinalidad de un conjunto finito es el número de elementos del conjunto, pero la cardinalidad de un conjunto infinito no es un número, sino una propiedad del conjunto llamada número cardinal que nos permite hacer comparaciones entre tamaños de conjuntos

Cardinalidad de conjuntos infinitos

- Para comparar las cardinalidades se utiliza el “*Principio del Palomar*”

Cardinalidad de conjuntos infinitos



f es una función inyectiva (uno a uno)

$$f: A \rightarrow B$$

$$x, y \in A, x \neq y \Rightarrow f(x) \neq f(y)$$

$$f(x) = f(y) \Rightarrow x = y$$

Cardinalidad de conjuntos infinitos

Definicion. Se establece que la cardinalidad de un conjunto A es menor o igual que la cardinalidad de un conjunto B, y se denota $|A| \leq |B|$ sí y sólo si se puede establecer una correspondencia “uno a uno” de cada elemento de A con un elemento distinto de B

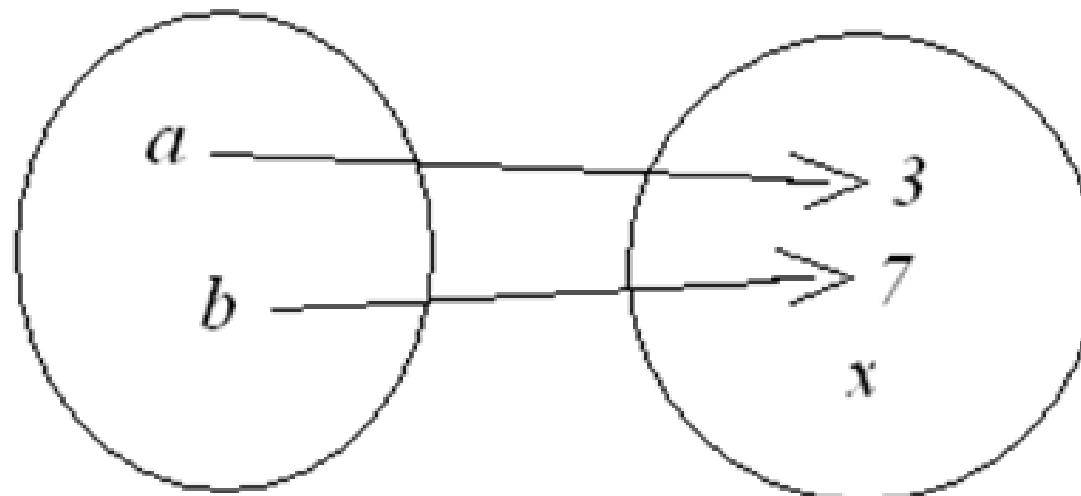
$$|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de } A \text{ en } B)$$

Puede demostrarse que es una relación de orden (reflexiva, antisimétrica y transitiva)

Cardinalidad de conjuntos infinitos

Observación. Esta definición es consistente con la forma de comparar las cardinalidades de conjuntos finitos. Por ejemplo

$$|\{a,b\}| \leq |\{3,7,x\}|$$



Cardinalidad de conjuntos infinitos

Definición. $|A| = |B| \Leftrightarrow |A| \leq |B| \wedge |B| \leq |A|$

Definición. $|A| < |B| \Leftrightarrow |A| \leq |B| \wedge |B| \neq |A|$

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$
- b) Mostrar que $|P| = |N|$ con $P = \{n/n \text{ es un número par}\}$
- c) Mostrar que $|Z| \leq |N|$
- d) Mostrar que $|NxN| = |N|$
- e) Mostrar que $|Q^+| \leq |N|$, siendo Q^+ el conjunto de los racionales positivos

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$

$N^+ = N - \{0\}$, para demostrar $|N| = |N^+|$ hay que demostrar

- 1) $|N| \leq |N^+|$**
- 2) $|N^+| \leq |N|$**

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$

$N^+ = N - \{0\}$, para demostrar $|N| = |N^+|$ hay que demostrar

- 1) $|N| \leq |N^+|$**
- 2) $|N^+| \leq |N|$**

$|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de } A \text{ en } B)$

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$

$N^+ = N - \{0\}$, para demostrar $|N| = |N^+|$ hay que demostrar

- 1) $|N| \leq |N^+|$
- 2) $|N^+| \leq |N|$

En ambos casos para dem. que es V habría que encontrar una función inyectiva, es decir:

- 1) $f_1: N \rightarrow N^+$ iny.
- 2) $f_2: N^+ \rightarrow N$ iny.

No hay restricciones para f_1 y f_2 , que inclusive podrían ser =

Cardinalidad de conjuntos infinitos

a) Mostrar que $|N| = |N^+|$

1) $|N| \leq |N^+| \quad f_1: N \rightarrow N^+ \text{ iny.}$

2) $|N^+| \leq |N| \quad f_2: N^+ \rightarrow N \text{ iny.}$

Cardinalidad de conjuntos infinitos

a) Mostrar que $|N| = |N^+|$

1) $|N| \leq |N^+| \quad f_1: N \rightarrow N^+ \text{ iny.}$

2) $|N^+| \leq |N| \quad f_2: N^+ \rightarrow N \text{ iny.}$

Cuando un conjunto es subconjunto del otro la función inyectiva es “trivial”: la identidad. En este caso:
 $|N^+| \leq |N|$ seguro es V por $\text{Id}: N^+ \rightarrow N$, $\text{Id}(n) = n$ es iny.

Cardinalidad de conjuntos infinitos

a) Mostrar que $|N| = |N^+|$

1) $|N| \leq |N^+| \quad f_1: N \rightarrow N^+ \text{ iny.}$

2) $|N^+| \leq |N| \quad f_2: N^+ \rightarrow N \text{ iny.}$

Cuando un conjunto es subconjunto del otro la función inyectiva es “trivial”: la identidad. En este caso:
 $|N^+| \leq |N|$ seguro es V por $\text{Id}: N^+ \rightarrow N$, $\text{Id}(n) = n$ es iny.

Faltaría demostrar 1), que no es demasiado difícil, ej:
 $\text{Id}: N \rightarrow N^+$, $\text{Succ}(n) = n+1$ es iny.

Con lo cual están demostrados 1) y 2) y por lo tanto
 $|N| = |N^+|$

Cardinalidad de conjuntos infinitos

a) Mostrar que $|N| = |N^+|$

Si bien es “intuitivo” pensar que N “tiene un elemento más” que N^+ , en realidad la idea de “uno más” no puede relacionarse con los infinitos elementos de ambos conjuntos.

De hecho, según la noción de cardinalidad que consideramos, $|N| = |N^+|$, tal como fue demostrado

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$
- b) Mostrar que $|P| = |N|$ con $P = \{n/n \text{ es un número par}\}$

Intuitivamente, el caso b) parece distinto al anterior, dado que en este caso en $|P|$ no solo no “falta” un número perteneciente a N como en el caso de N^+ sino una cantidad infinita de números: todos los impares. Sin embargo, para demostrar $|P| = |N|$ deberíamos probar la definición:

- 1) $|P| \leq |N|$
- 2) $|N| \leq |P|$

En ambos casos: función inyectiva.

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$
- b) Mostrar que $|P| = |N|$ con $P = \{n/n \text{ es un número par}\}$

- 1) $|P| \leq |N|$** “Fácil” xq $\text{Id}: P \rightarrow N$, $\text{Id}(n) = n$ es inyectiva
2) $|N| \leq |P|$

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$
- b) Mostrar que $|P| = |N|$ con $P = \{n/n \text{ es un número par}\}$

- 1) $|P| \leq |N|$** “Fácil” xq $\text{Id}: P \rightarrow N$, $\text{Id}(n) = n$ es inyectiva
2) $|N| \leq |P|$

Para 2) se puede usar $\text{dbl}: N \rightarrow P$, $\text{dbl}(n) = 2n$ es inyectiva

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$
- b) Mostrar que $|P| = |N|$ con $P = \{n/n \text{ es un número par}\}$

1) $|P| \leq |N|$ **Id: $P \rightarrow N$, $Id(n) = n$ es inyectiva**

2) $|N| \leq |P|$ **dbl: $N \rightarrow P$, $dbl(n) = 2n$ es inyectiva**

Cardinalidad de conjuntos infinitos

Ejercicios

- a) Mostrar que $|N| = |N^+|$
- b) Mostrar que $|P| = |N|$ con $P = \{n/n \text{ es un número par}\}$

1) $|P| \leq |N|$ **Id: $P \rightarrow N$, $Id(n) = n$ es inyectiva**

2) $|N| \leq |P|$ **dbl: $N \rightarrow P$, $dbl(n) = 2n$ es inyectiva**

En este caso, aunque “intuitivamente” P tiene “la mitad” de elementos que N en realidad no existe la noción de “mitad de infinito”. Intuitivamente tenemos un problema más relacionado con nuestra forma de entender los conjuntos infinitos que con la noción o definición formal de cardinalidad de conjuntos infinitos

Cardinalidad de conjuntos infinitos

$$|N| = |NxN|$$

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

Cardinalidad de conjuntos infinitos

$$|\mathbb{N}| = |\mathbb{N} \times \mathbb{N}| \Leftrightarrow (|\mathbb{N}| \leq |\mathbb{N} \times \mathbb{N}| \wedge |\mathbb{N} \times \mathbb{N}| \leq |\mathbb{N}|)$$

a) $|\mathbb{N}| \leq |\mathbb{N} \times \mathbb{N}|$, sin problemas ¿función?

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

a) $|N| \leq |NxN|$ $Idd: N \rightarrow NxN, Idd(n) = (n,n)$

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ $Idd: N \rightarrow NxN, Idd(n) = (n,n)$
- b) $|NxN| \leq |N|$ $f : NxN \rightarrow N, f$ inyectiva

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ $Idd: N \rightarrow NxN, Idd(n) = (n,n)$
- b) $|NxN| \leq |N|$ $f : NxN \rightarrow N, f$ inyectiva

¿Cómo dar a cada (i, j) un único n ?

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ Idd: $N \rightarrow NxN$, Idd(n) = (n, n)
- b) $|NxN| \leq |N|$ $f : NxN \rightarrow N$, f inyectiva

¿Cómo dar a cada (i, j) un único n ?

Si pudiéramos ordenar los pares ordenados...

El propio orden es la asociación, o la relación, o en realidad la función inyectiva que a cada par lo relaciona con su posición en el orden que se defina

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ $Idd: N \rightarrow NxN$, $Idd(n) = (n,n)$
- b) $|NxN| \leq |N|$ $f : NxN \rightarrow N$, f inyectiva

¿Cómo dar a cada (i, j) un único n ?

Si pudiéramos ordenar los pares ordenados...

$(0,0) (0,1) (1,0) (0,2) (1,1) (2,0) (0,3) (1,2) (2,1) (3,0)$

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ Idd: $N \rightarrow NxN$, Idd(n) = (n, n)
- b) $|NxN| \leq |N|$ $f : NxN \rightarrow N$, f inyectiva

¿Cómo dar a cada (i, j) un único n ?

Si pudiéramos ordenar los pares ordenados...

Suma 0	Suman 1	Suman 2	Suman 3
$(0,0)$	$(0,1)$ $(1,0)$	$(0,2)$ $(1,1)$ $(2,0)$	$(0,3)$ $(1,2)$ $(2,1)$ $(3,0)$

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ Idd: $N \rightarrow NxN$, Idd(n) = (n, n)
- b) $|NxN| \leq |N|$ $f : NxN \rightarrow N$, f inyectiva

¿Cómo dar a cada (i, j) un único n ?

Si pudiéramos ordenar los pares ordenados...

Suma 0	Suman 1	Suman 2	Suman 3
$(0,0)$	$(\underline{0},1)$ $(1,\underline{0})$	$(0,2)$ $(\underline{1},1)$ $(2,0)$	$(0,3)$ $(1,2)$ $(2,1)$ $(3,0)$

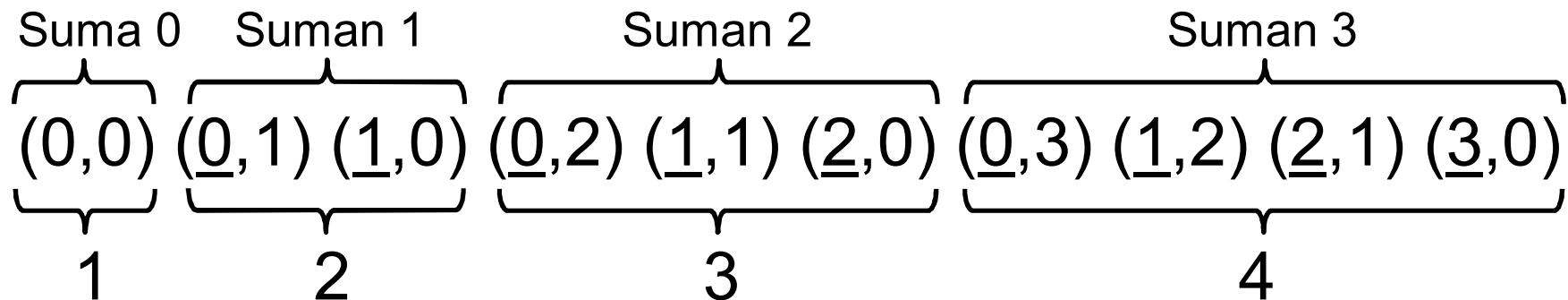
Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

- a) $|N| \leq |NxN|$ Idd: $N \rightarrow NxN$, Idd(n) = (n, n)
b) $|NxN| \leq |N|$ $f : NxN \rightarrow N$, f inyectiva

¿Cómo dar a cada (i, j) un único n ?

Si pudiéramos ordenar los pares ordenados...



Cardinalidad de conjuntos infinitos

Si pudiéramos ordenar los pares ordenados...

Suma 0	Suman 1	Suman 2	Suman 3
(0,0)	(0,1) (1,0)	(0,2) (1,1) (2,0)	(0,3) (1,2) (2,1) (3,0)

1 2 3 4

A excepción de (0,0), para cualquier (i, j), su posición va a ser posterior a todos los que suman

0, 1, ..., i+j-1

Cardinalidad de conjuntos infinitos

Si pudiéramos ordenar los pares ordenados...

Suma 0	Suman 1	Suman 2	Suman 3
(0,0)	(0,1) (1,0)	(0,2) (1,1) (2,0)	(0,3) (1,2) (2,1) (3,0)

1 2 3 4

A excepción de (0,0), para cualquier (i, j) , su posición va a ser posterior a todos los que suman $0, 1, \dots, i+j-1$, que son, respectivamente, $1, 2, \dots, i+j$

Cardinalidad de conjuntos infinitos

Si pudiéramos ordenar los pares ordenados...

Suma 0	Suman 1	Suman 2	Suman 3
(0,0)	(0,1) (1,0)	(0,2) (1,1) (2,0)	(0,3) (1,2) (2,1) (3,0)

1 2 3 4

A excepción de (0,0), para cualquier (i, j), su posición va a ser posterior a todos los que suman

$$0, 1, \dots, i+j-1$$

$$1, 2, \dots, i+j$$

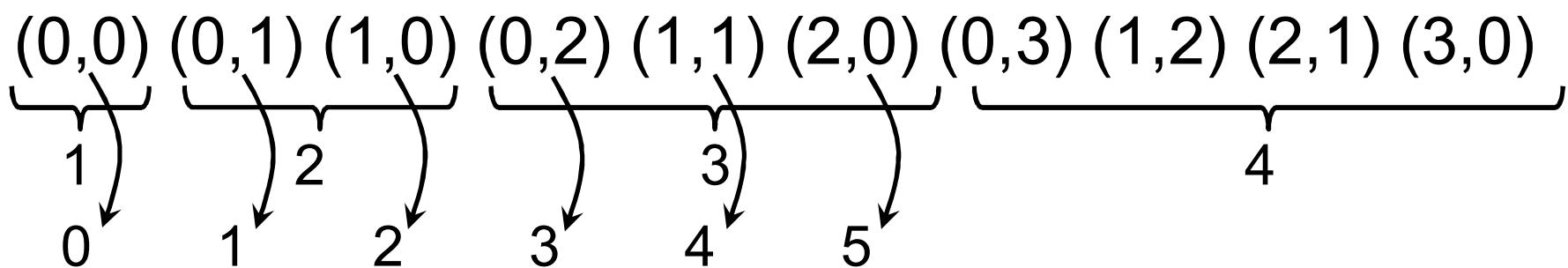
$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

Es decir $\sum_{k=1}^{i+j} k = \frac{(i+j)(i+j+1)}{2}$ son los anteriores

Cardinalidad de conjuntos infinitos

Si pudiéramos ordenar los pares ordenados...

Es decir $\sum_{k=1}^{i+j} k = \frac{(i+j)(i+j+1)}{2}$ son los anteriores



Y a partir de esa cantidad, la posición estará dada por “i”

Es decir que $f(i,j) = \frac{(i+j)(i+j+1)}{2} + i$, es

$f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, f inyectiva

Cardinalidad de conjuntos infinitos

$$|N| = |NxN| \Leftrightarrow (|N| \leq |NxN| \wedge |NxN| \leq |N|)$$

a) $|N| \leq |NxN|$ Idd: $N \rightarrow NxN$, Idd(n) = (n, n)

b) $|NxN| \leq |N|$, $f : NxN \rightarrow N$, f inyectiva

Con $f(i,j) = \frac{(i+j)(i+j+1)}{2} + i$ se tiene esa función

Con a) y b) se tiene demostrado que $|N| = |NxN|$

Conjuntos contables

Definicion. Un conjunto infinito es contable, o también llamado numerable, cuando su cardinalidad es igual a la cardinalidad de los naturales

Si $|A| = |N| \Rightarrow A$ es contable

para saber si un conjunto es contable alcanza con probar que $|A| \leq |N|$ (si A es infinito ya se sabe que $|N| \leq |A|$). Por lo tanto :

Si $|A| \leq |N| \Rightarrow A$ es contable

Un conjunto no contable

Teorema. $|N| < |\rho(N)|$

Dem.

Hay que demostrar dos cosas

- a) $|N| \leq |\rho(N)|$ y b) $|N| \neq |\rho(N)|$

- a) Sea $f: N \rightarrow \rho(N)$, tal que $f(n) = \{n\}$

Claramente f es una función inyectiva de N a $\rho(N)$,
por lo tanto $|N| \leq |\rho(N)|$

Un conjunto no contable

b) Demostraremos por el absurdo que $|N| \neq |\rho(N)|$

Supongamos que $|\rho(N)| \leq |N|$. Por lo tanto los elementos de pueden ponerse en correspondencia uno a uno con los naturales, es decir pueden colocarse en una sucesión como la siguiente:

$$\rho(N) = \{ S_1, S_2, S_3, \dots \}$$

Definimos un conjunto D de la siguiente manera

$$D = \{ n \in N / n \notin S_n \}$$

Un conjunto no contable

Obsérvese que D es un conjunto bien definido, perfectamente válido. Además $D \subseteq N$, por lo tanto $D \in \rho(N)$

Por lo tanto debe existir algún natural k para el cual $D = S_k$

¿Qué pasa con k ? ¿Pertenece o no pertenece a D ?
Bueno, hay dos posibilidades y las dos llevan a una contradicción

Un conjunto no contable

($D = \{n \in \mathbb{N} / n \notin S_n\}$ \wedge $D = S_k$), ¿ $k \in D$ o $k \notin D$?

Un conjunto no contable

($D = \{n \in \mathbb{N} / n \notin S_n\}$ \wedge $D = S_k$), ¿ $k \in D$ o $k \notin D$?

1) $k \in D \Rightarrow k \notin S_k$ (Por definición de D)

$\Rightarrow k \notin D$ (Porque $D = S_k$)

Contradicción

2) $k \notin D \Rightarrow k \in S_k$ (Por definición de D)

$\Rightarrow k \in D$ (Porque $D = S_k$)

Contradicción

Un conjunto no contable

Por lo tanto no puede existir la sucesión como la que se asumió. Es decir no puede existir una función inyectiva que a cada elemento de $\wp(N)$ le asigne uno de N .

Entonces $|\wp(N)| \not\leq |N|$ entonces $|\wp(N)| \neq |N|$

De (a) y (b) quedando demostrado que $|N| < |\wp(N)|$

Un conjunto no contable

Cololario. $\wp(\mathbb{N})$ es no contable

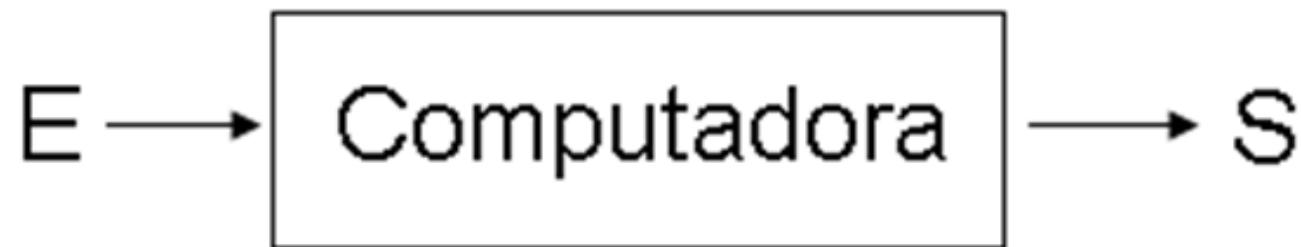
Se podría pensar como que “hay más” subconjuntos de \mathbb{N} que números en \mathbb{N}

Conjuntos no contables

Nota. El resultado del teorema anterior puede generalizarse a cualquier conjunto, es decir:

$$|A| < |\wp(A)|$$

¿Por qué nos interesa todo lo anterior?



E y S pueden codificarse en binario, por lo tanto puede verse como un número natural. Así podemos decir que la computadora implementa una función $f: \mathbb{N} \rightarrow \mathbb{N}$

¿Podríamos construir cualquier $f: \mathbb{N} \rightarrow \mathbb{N}$?

¿Por qué nos interesa todo lo anterior?

Puede verse fácilmente que $|\rho(N)| \leq |\{f / f: N \rightarrow \{0,1\}\}| \leq |F|$

Aclaración: Por cada conjunto A de $\rho(N)$ hay una función $f: N \rightarrow \{0,1\}$ llamada característica del conjunto A definida como:

$$f(n) \quad \begin{cases} f(n) = 0, & \text{si } n \notin A \\ f(n) = 1, & \text{si } n \in A \end{cases}$$

Y como $|N| < |\rho(N)|$, por transitividad se tiene que $|N| < |F|$

Es decir: **F es no contable**

¿Por qué nos interesa todo lo anterior?

Sea PROG el conjunto de todos los programas de computación que pueden escribirse para computar funciones. PROG es contable pues puede verse como un $n \in \mathbb{N}$ escrito en binario

Por lo tanto:

$$|\text{PROG}| \leq |\mathbb{N}| < |\mathcal{P}(\mathbb{N})| \leq |\{f / f: \mathbb{N} \rightarrow \{0,1\}\}| \leq |F|$$

Es decir: $|\text{PROG}| < |F|$

- Esta es la primera “aproximación” al problema de Computabilidad
- Ni siquiera se puede tener un programa para cada $f:N \rightarrow N$
- Con los conceptos vistos hasta este punto ya se puede resolver toda la Práctica 1

Cardinalidades

(Más) Comparaciones de “Tamaño”
Notaciones

Notaciones y \mathbb{R}

- Se suele denotar $|N| = \aleph_0$ (Aleph₀)
 - En nuestro caso, solemos usar solo $|N|$...

Notaciones y \mathbb{R}

- Se suele denotar $|N| = \aleph_0$ (Aleph₀)
- Consideramos $N = \{0, 1, 2, \dots\}$
 - $|\emptyset| = 0$
 - $P = \{n = 2k / k \in N\}$ (0 es, por lo tanto, par)
 - $I = \{n = 2k+1 / k \in N\}$
 - $P \subset N \qquad I \subset N \qquad N = P \cup I$

Notaciones y \mathbb{R}

- Se suele denotar $|N| = \aleph_0$ (Aleph₀)
- Consideramos $N = \{0, 1, 2, \dots\}$
- $|N| < |\mathbb{R}|$
 - Diagonal de Cantor, ver “05b-0...1.mp4”
“Las matemáticas tienen una terrible falla”
<https://youtu.be/RRg38oNQ9vk>

Notaciones y \mathbb{R}

- Se suele denotar $|N| = \aleph_0$ (Aleph₀)
- Consideramos $N = \{0, 1, 2, \dots\}$
- $|N| < |\mathbb{R}|$
 - Diagonal de Cantor, ver “05b-0...1.mp4”
“Las matemáticas tienen una terrible falla”
<https://youtu.be/RRg38oNQ9vk>
 - Se suele denotar $|\mathbb{R}| = c \dots \text{¿}\aleph_1?$
 - ...

Notaciones y \mathbb{R}

- Se suele denotar $|N| = \aleph_0$ (Aleph₀)
- Consideramos $N = \{0, 1, 2, \dots\}$
- $|N| < |\mathbb{R}|$
- Conjunto de Partes
 - Mencionado como “conjunto potencia” en parte de la bibliografía

Notaciones y \mathbb{R}

- Se suele denotar $|N| = \aleph_0$ (Aleph₀)
- Consideramos $N = \{0, 1, 2, \dots\}$
- $|N| < |\mathbb{R}|$
- Conjunto de Partes
- “Curiosidades”
 - \mathbb{N} : Naturales (en nuestro caso, incluye el 0)
 - \mathbb{Z} : Enteros (“Zahlen”, “número” en alemán)
 - \mathbb{Q} : Racionales (Quotient o Quoziente)
 - Racional: “Radio” o “Relación” de proporcionalidad

Sobre las Definiciones

- Definición de Cardinalidad...

$|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de } A \text{ en } B)$

- No usamos “cantidad” o “número”
- Igualdad y $<$ en función de “ \leq ”

Definición. $|A| = |B| \Leftrightarrow |A| \leq |B| \wedge |B| \leq |A|$

Definición. $|A| < |B| \Leftrightarrow |A| \leq |B| \wedge |B| \neq |A|$

Sobre las Definiciones

- Definición de Cardinalidad...
 $|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de } A \text{ en } B)$
- La **igualdad** también def. con biyección
 - O “coordinación” de los elementos (Galileo)

Sobre las Definiciones

- Definición de Cardinalidad...
 $|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de } A \text{ en } B)$
- La **igualdad** también def. con biyección
 - O “coordinación” de los elementos
 - Inyectiva: “ \leq ” 

Sobre las Definiciones

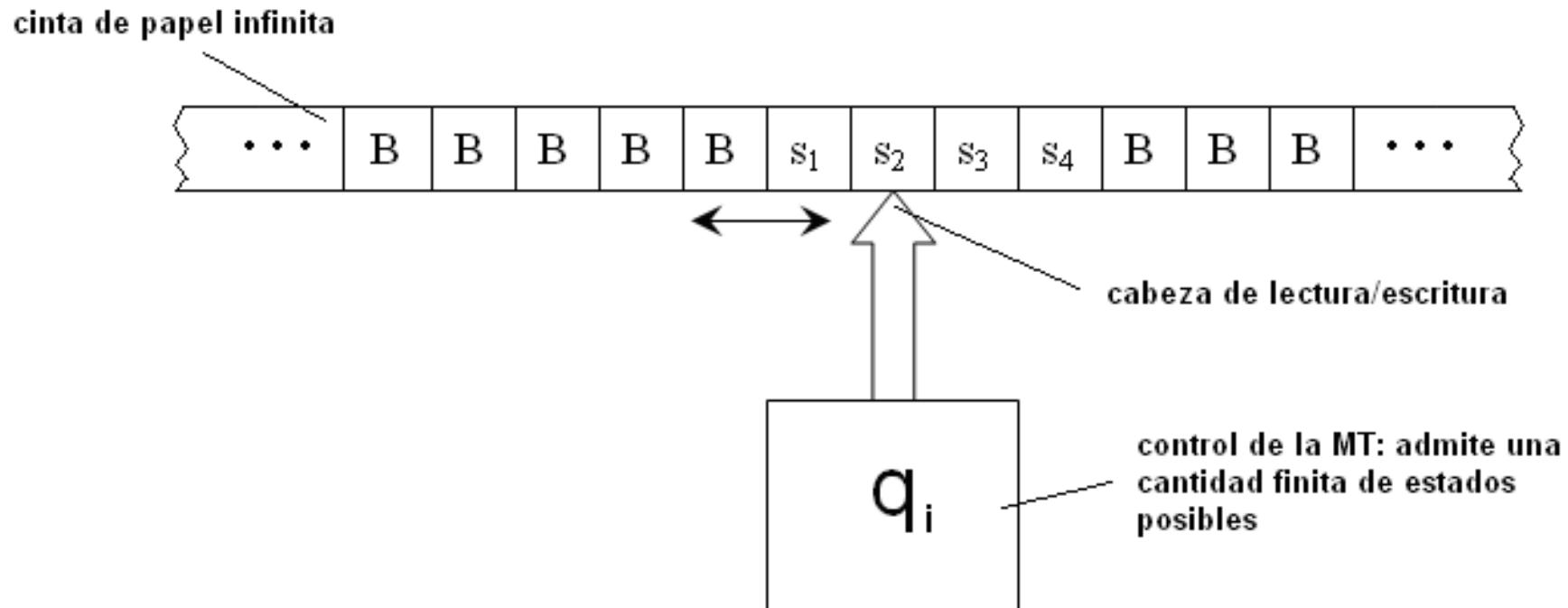
- Definición de Cardinalidad...
 $|A| \leq |B| \Leftrightarrow (\exists f): (f \text{ es una función inyectiva de } A \text{ en } B)$
- La igualdad también def. con biyección
 - O “coordinación” de los elementos
 - Inyectiva: “ \leq ”  $(f: A \rightarrow B)$
 - “Simplifica” en el caso de $A \subseteq B$ (Id)
 - O casos triviales como $|\mathbb{N}| < |\wp(\mathbb{N})|$
 - Está demostrado que
 - Si $\exists f_1: A \rightarrow B$ y $f_2: B \rightarrow A$ inyectivas $\Rightarrow \exists f_3: A \rightarrow B$ biyectiva (Cantor-Schröder-Bernstein)

Máquina de Turing

Características del proceso de cálculo de una persona

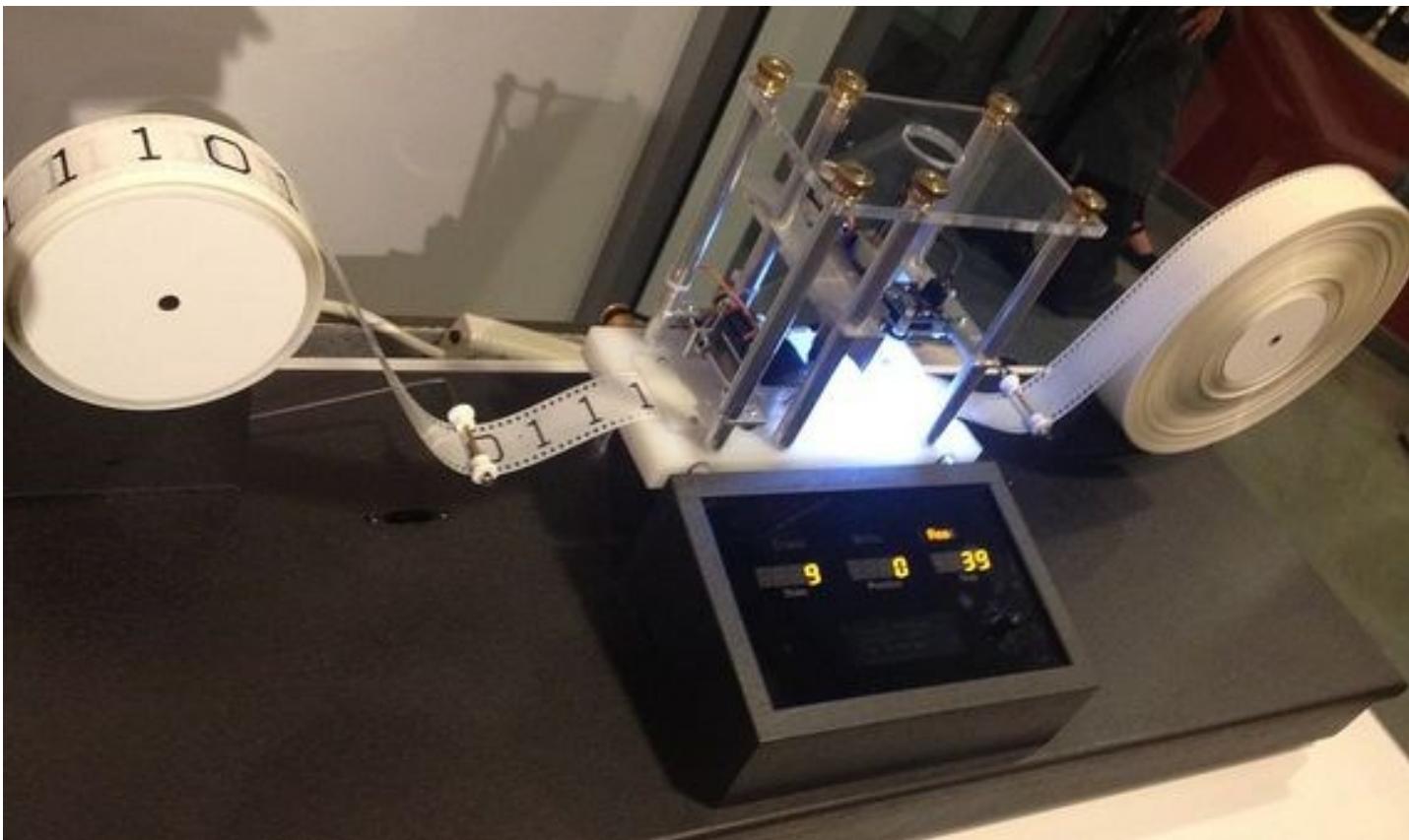
- Se concentra en una porción restringida del papel
- Trabaja con un número finito de símbolos
- Puede cambiar la sección de papel en que se concentra (de acuerdo al símbolo que observa y a sus estado mental)
- Pasa por un número finito de estados mentales distinguibles
- Se asume que siempre contará con el papel suficiente para sus cálculos (se asume infinito)

Máquina de Turing



En cada instante, la máquina se encuentra en algún estado q_i , perteneciente al conjunto finito Q de todos los estados posibles

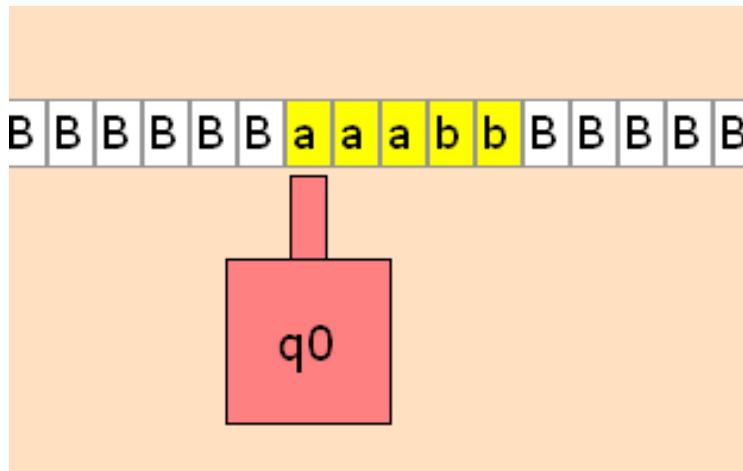
$$Q = \{q_0, q_1, q_2, \dots, q_n\}$$



Exhibición de la Colección de Instrumentos Científicos Históricos de Harvard.

Sin embargo la Máquina de Turing no es una máquina real, sino una máquina abstracta (es un concepto matemático)

Configuración inicial

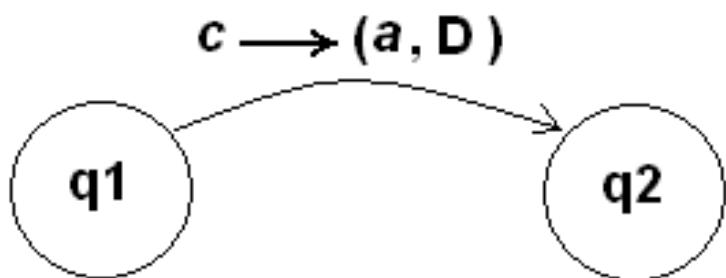
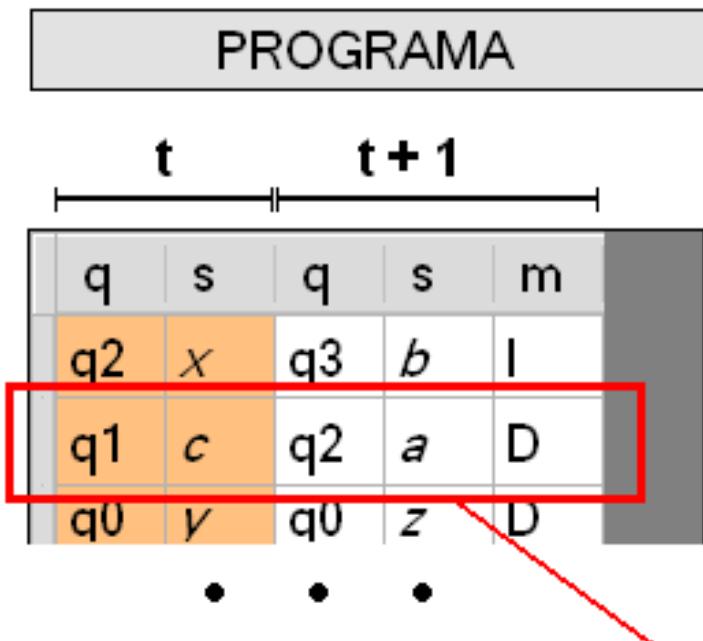


- La máquina siempre comienza en el estado inicial q_0
- Si existe una cadena de símbolos de entrada, la máquina comienza apuntando al primer símbolo de esta cadena.
- Si no existe una cadena de entrada escrita en la cinta, sólo hay símbolos “B” en cada celda de la misma.
- La cadena de entrada estará limitada por infinitos B a izquierda y derecha. Además no hay ningún símbolo B en medio de la cadena

Comportamiento de la máquina de Turing

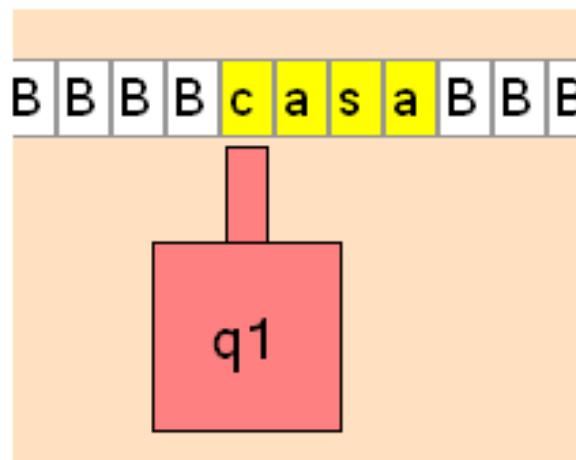
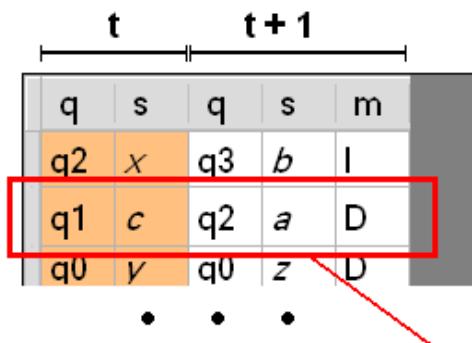
- El comportamiento de la máquina está definido por una función de transición (programa)
- Dependiendo del símbolo en la celda actual y del estado corriente, la máquina efectúa en un único paso de computación las siguientes acciones
 1. Cambia de estado (o vuelve a elegir el actual)
 2. Escribe un símbolo en la celda actual, reemplazando lo que allí había (puede escribir el mismo símbolo que estaba)
 3. Mueve el cabezal a la izquierda o la derecha, exactamente una celda

Ejemplos

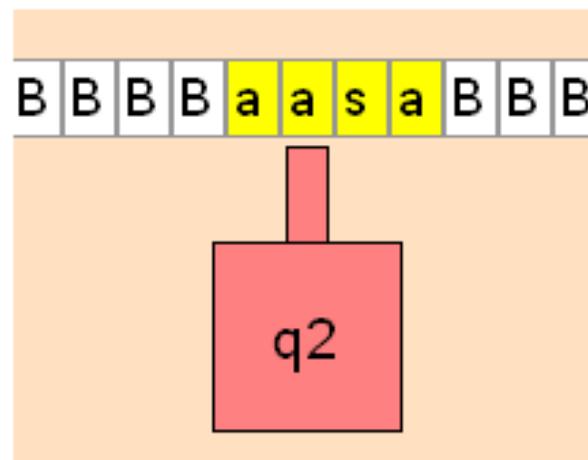


Estando en el estado q_1 , leyendo el símbolo c en la celda corriente, lo reemplaza con el símbolo a y mueve la cabeza a la derecha

PROGRAMA



Instante t



Instante $t+1$

Comportamiento de la máquina de Turing

RECAPITULANDO

- El programa de la MT no es un programa secuencial sino que es una **función matemática** de transición.
- La máquina trabaja haciendo “**pattern matching**”, es decir busca en su programa cuál es la línea (transición) que debe aplicar según su estado actual y símbolo leído.
- Si **no existe ninguna transición** definida para el estado actual y símbolo leído **la máquina se detiene**.

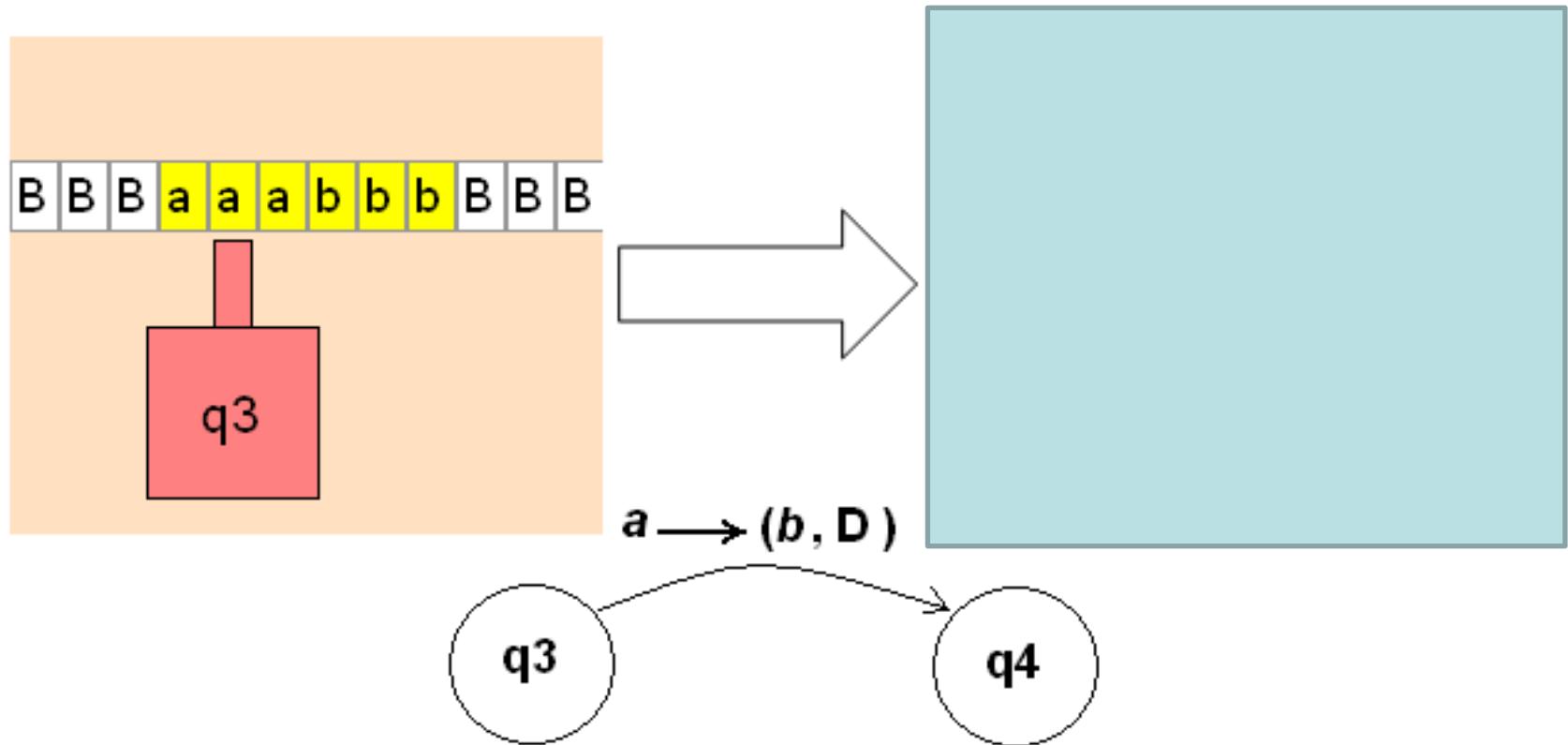
Comportamiento de la máquina de Turing

Para pensar

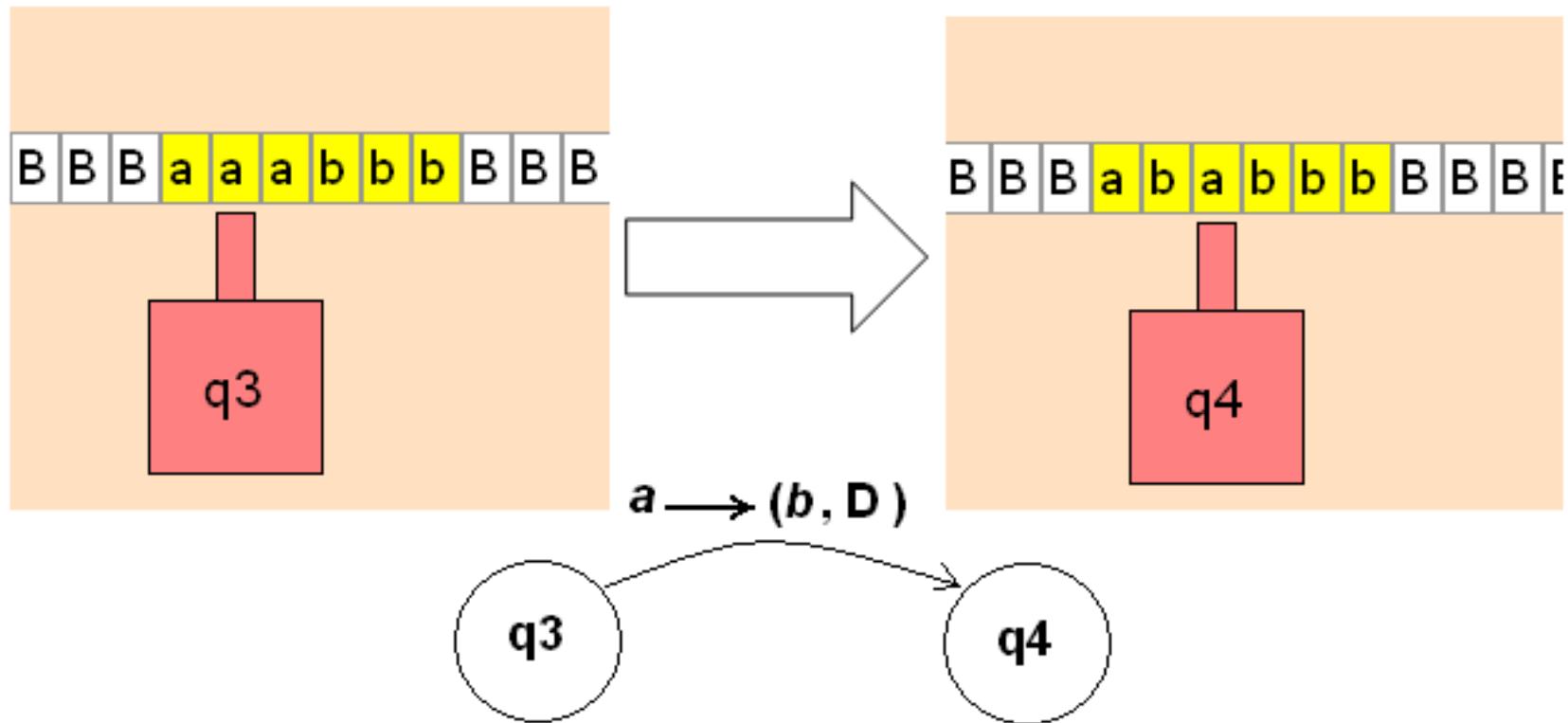
¿Qué ocurriría si más de una línea hiciese “pattern matching” en el mismo momento?

- ¿Cómo se imagina que actuaría la MT?
- ¿El programa de la MT seguiría siendo una función matemática?
- El modelo de MT no determinísticas (MTND) que veremos más adelante busca precisamente el efecto anterior. Además se define de tal forma que el programa sigue siendo una función matemática

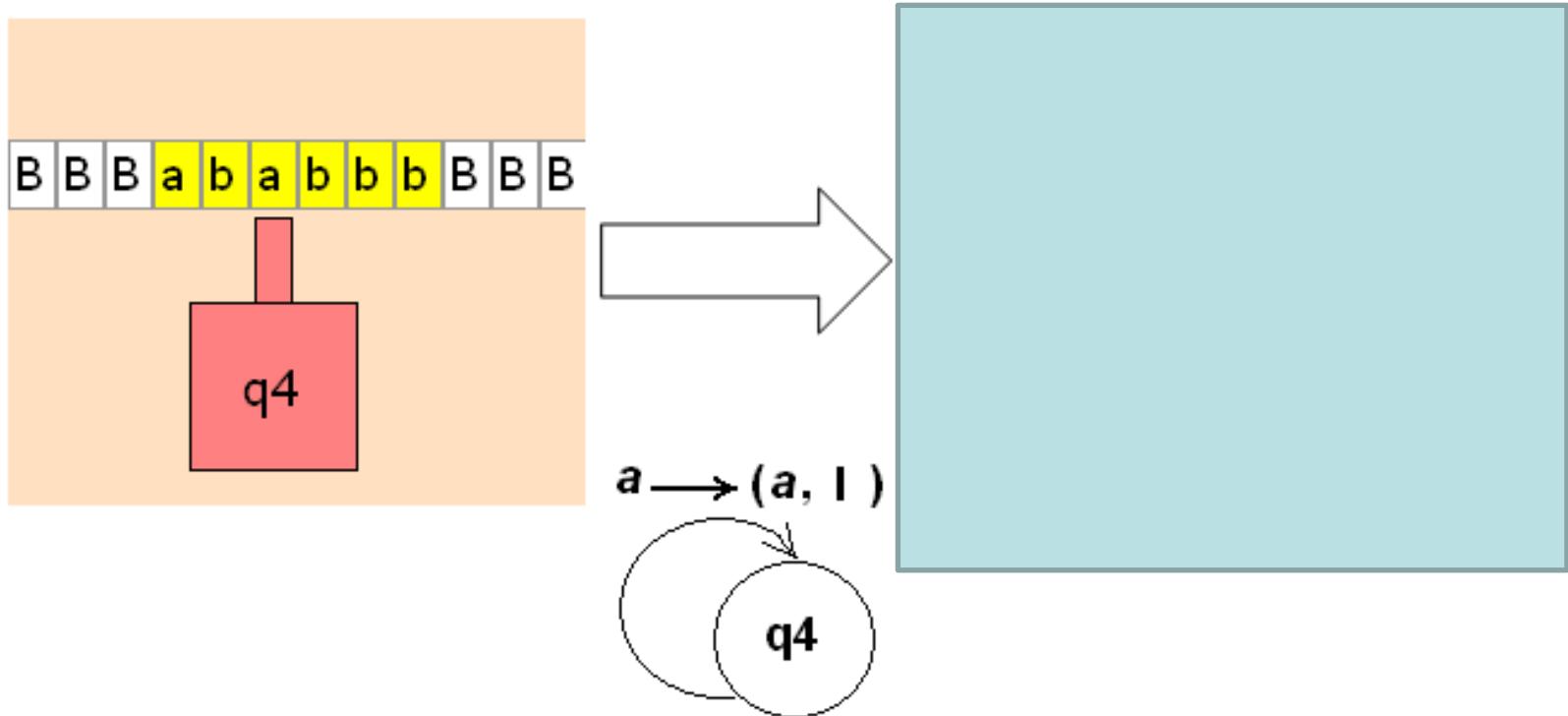
Más ejemplos de transiciones



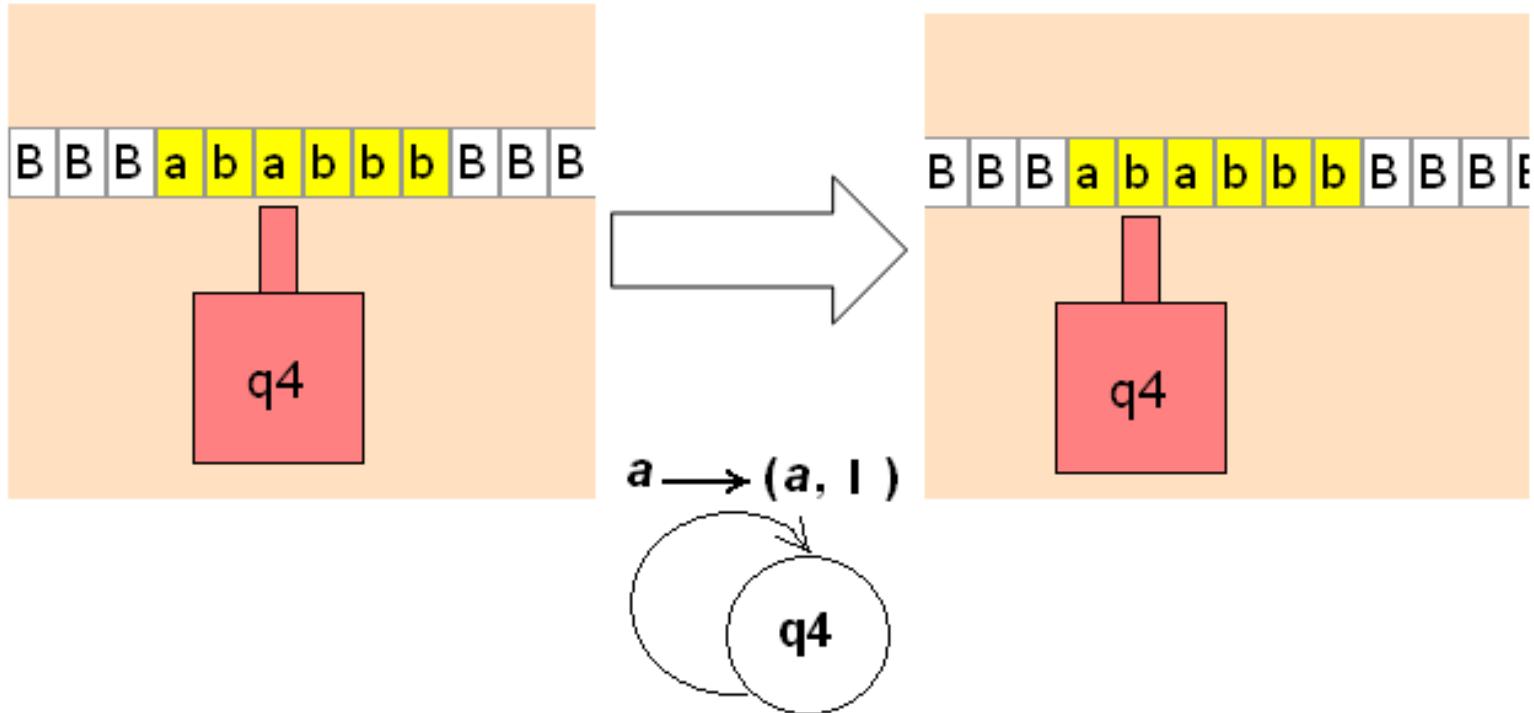
Más ejemplos de transiciones



Más ejemplos de transiciones



Más ejemplos de transiciones



Actividades-Resolver con MT

Supongamos cadenas formadas sólo por símbolos *a* y *b*.

1. Una MT que borra el primer símbolo de la cadena sólo si es un símbolo *a*
2. Una MT que borra el primer símbolo de la cadena
3. Una MT que borra todos los símbolos de la cadena
4. Una MT que borra los símbolos de la cadena en las posiciones pares
5. Una MT que hace zig-zag sobre la cadena de entrada recorriéndola hacia la derecha y luego hacia la izquierda indefinidamente.

Actividades-Resolver con MT

6. Escribir símbolos “1” a la derecha indefinidamente
7. Escribir símbolos “0” a la izquierda indefinidamente
8. Escribir la palabra “casa”
9. Escribir indefinidamente “casa-casa-casa-casa” hacia la izquierda
10. Escribir “1” hacia la derecha y “0” hacia la izquierda en zigzag indefinidamente, es decir: va a derecha para escribir un 1 al final, y cambia el sentido hacia la izquierda para escribir un 0, y cambia el sentido hacia la derecha, así indefinidamente

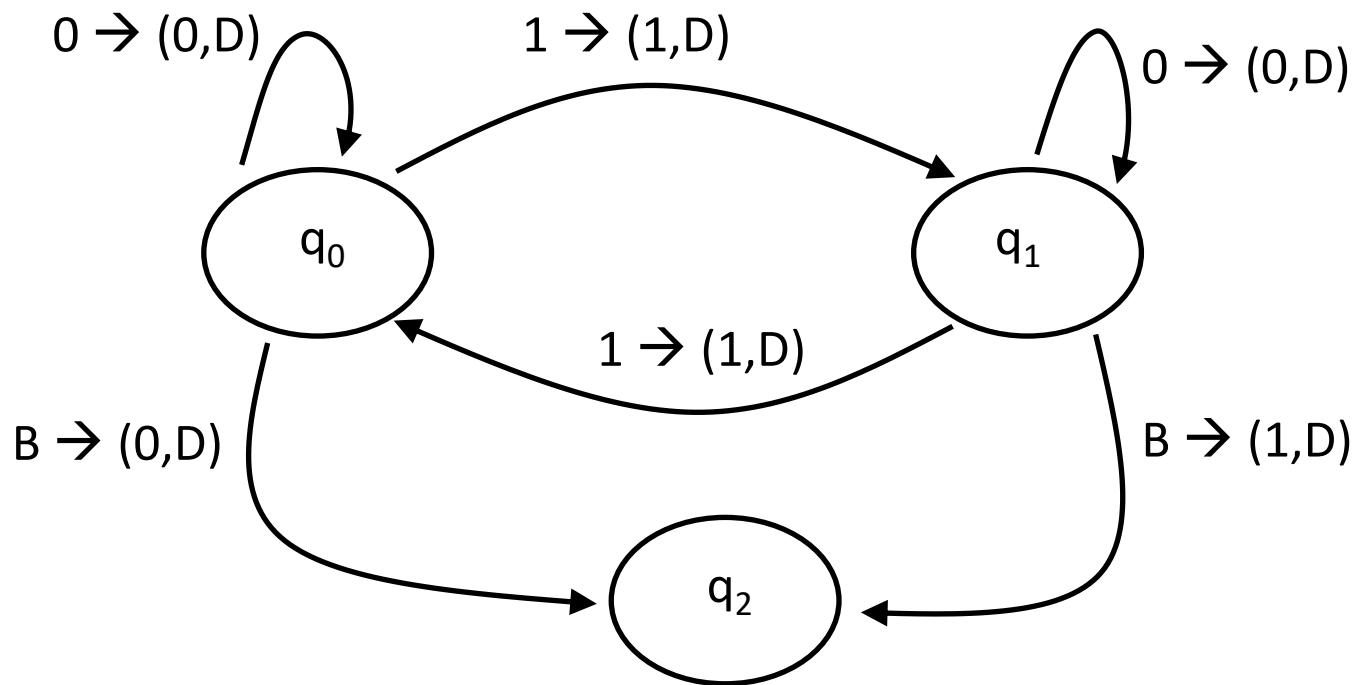
Ejercicios

Ej. 1. Construir una máquina de Turing que agregue un bit de paridad a una secuencia binaria de entrada, para que la cantidad de “1” sea par. ($\Sigma=\{0,1\}$ y $\Gamma =\{0,1,B\}$)

Alfabetos (Conjuntos finitos de símbolos)

- Σ alfabeto de la entrada: símbolos con los que se forma la cadena de entrada
- Γ alfabeto de la cinta: símbolos que la máquina de Turing puede escribir en la cinta y por lo tanto también leer. Necesariamente Σ está incluido en Γ y B (celda en blanco) siempre pertenece a Γ

Solución Ej. 1



Ejercicios

Ej. 2. ¿Qué hacen las siguientes máquinas de Turing?

$$\begin{array}{l} 1 \rightarrow (B,D) \\ 0 \rightarrow (B,D) \end{array}$$

$$B \rightarrow (B,D)$$

$$\Gamma = \{0,1,B\}$$

a)



b)

$$\begin{array}{l} 1 \rightarrow (B,D) \\ 0 \rightarrow (B,D) \end{array}$$

$$\Gamma = \{0,1,B\}$$

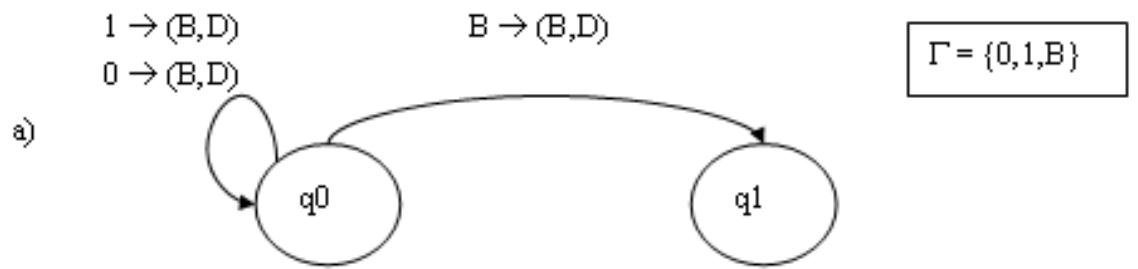
c)

$$\begin{array}{l} B \rightarrow (B,D) \\ 0 \rightarrow (B,D) \\ 1 \rightarrow (B,D) \end{array}$$

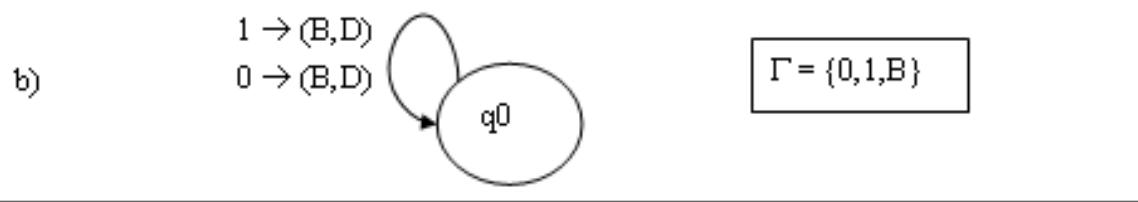
$$\Gamma = \{0,1,B\}$$

Respuesta

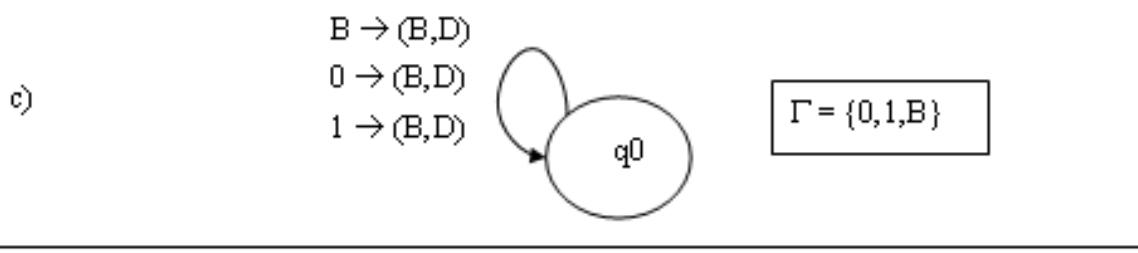
Las tres MT hacen lo mismo, borrar el contenido de la cinta, pero lo hacen de manera distinta



Borra la cinta y se detiene en el estado q_1



Borra la cinta y se detiene en el estado q_0



Borra la cinta pero no se detiene nunca, avanza a la derecha infinitamente

Ejercicios

- **Ej. 3.** Sumar 1 al número unario escrito en la cinta. En unario, el número n se representa como una cadena de n símbolos 1, el cero es una cadena vacía. ($\Sigma=\{1\}$ y $\Gamma =\{1,B\}$)
- **Ej. 4.** Construir una máquina de Turing que haga un corrimiento a derecha de la cadena binaria en la cinta, marcando con un símbolo especial “#” la celda que correspondía al primer símbolo desplazado. ($\Sigma=\{0,1\}$ y $\Gamma =\{0,1,B,\#\}$)

Máquina de Turing de/para Cómputo

Definición. Una Máquina de Turing de Cómputo se puede definir con una quíntupla

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0 \rangle$$

tal que:

Q es un conjunto finito de estados de M

Σ es el alfabeto de la entrada

Γ es el alfabeto de la cinta. $\Sigma \subset \Gamma$ y $B \in (\Gamma - \Sigma)$

q_0 es el estado inicial de M ($q_0 \in Q$)

δ es la función de transición de M .

Se define $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{D, I\}$,

D e I representan el movimiento del cabezal a derecha e izquierda respectivamente.

EL "resultado" del cómputo es lo que queda en la cinta

Máquina de Turing de/para Cómputo

Definición. Una Máquina de Turing de Cómputo se puede definir con una quíntupla

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0 \rangle$$

El cálculo terminará cuando la máquina alcance una situación de indefinición, es decir que δ no esté definida para el símbolo y estado corrientes.

Máquina de Turing de/para Cómputo

Modelo alternativo: MT con estado de detención q_d

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle$$

Q es un conjunto finito de estados de M

Σ es el alfabeto de la entrada

Γ es el alfabeto de la cinta. $\Sigma \subset \Gamma$ y $B \in (\Gamma - \Sigma)$

q_0 es el estado inicial de M ($q_0 \in Q$)

q_d es el estado de detención de M ($q_d \notin Q$)

δ es la función de transición de M .

Se define $\delta: Q \times \Gamma \rightarrow Q \cup \{q_d\} \times \Gamma \times \{D, I\}$

Necesariamente la máquina dejará de computar cuando

llegue a q_d porque no puede aplicar δ (q_d no es parte del dominio)

Máquina de Turing de/para Cómputo

Modelo alternativo: MT con estado de detención q_d

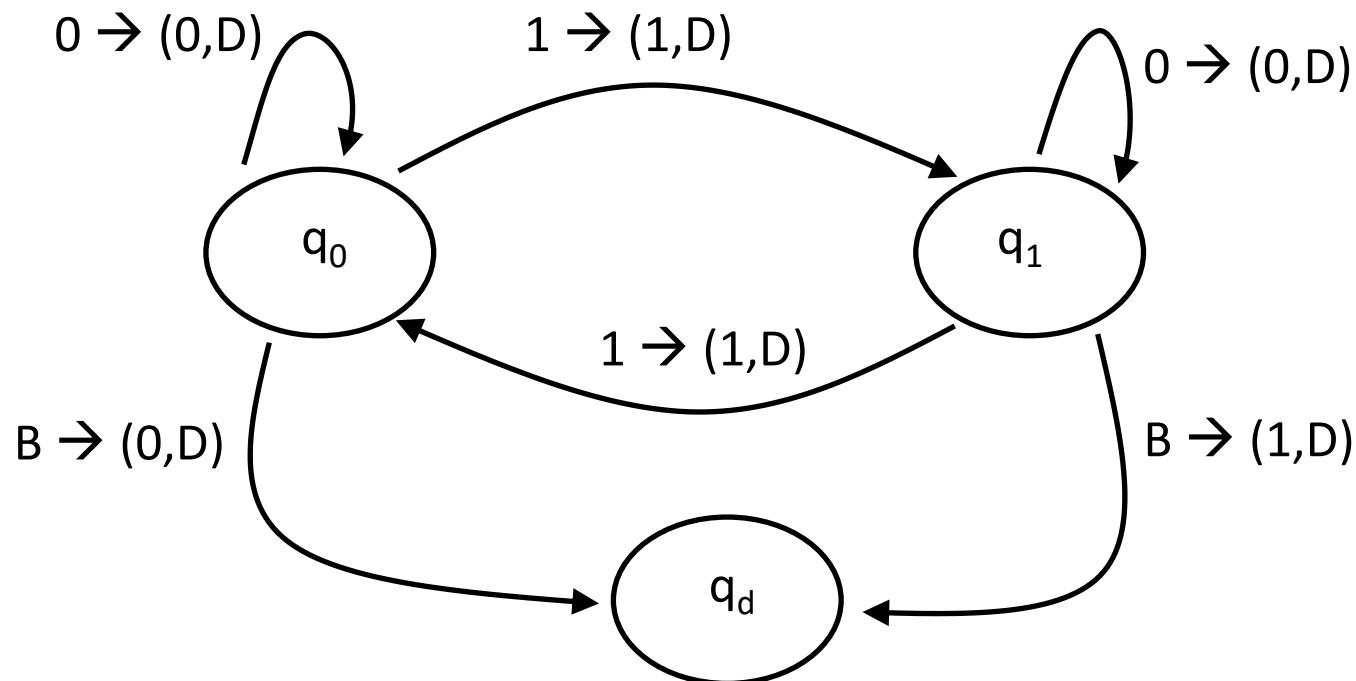
$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle$$

Para este caso (máquina de Turing de cómputo con estado de detención) se exige que la función δ esté completamente definida en todo su dominio $Q \times \Gamma$

Ejemplo

$$M = \langle \{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, q_d \rangle$$

Grafo de δ



Ejemplo

$$M = \langle \{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, q_d \rangle$$

Notación funcional de δ

$$\delta: Q \times \Gamma \rightarrow Q \cup \{q_d\} \times \Gamma \times \{D, I\}$$

$$\delta(q_0, 0) = (q_0, 0, D)$$

$$\delta(q_0, 1) = (q_1, 1, D)$$

$$\delta(q_0, B) = (q_d, 0, D)$$

$$\delta(q_1, 0) = (q_1, 0, D)$$

$$\delta(q_1, 1) = (q_0, 1, D)$$

$$\delta(q_1, B) = (q_d, 1, D)$$

Descripción Instantánea de una MT

Denotaremos $s_1 s_2 \dots q s_i \dots s_n$ a la **configuración o descripción instantánea** de la MT M que indica:

- El contenido de la cinta es

$\dots B B B s_1 s_2 \dots s_i \dots s_n B B B \dots$

- El estado actual de M es q
- El cabezal se encuentra barriendo el símbolo s_i

Importante: Γ y Q son disjuntos ($\Gamma \cap Q = \emptyset$)

Descripción Instantánea de una MT

Denotaremos $s_1 s_2 \dots q s_i \dots s_n$ a la **configuración o descripción instantánea** de la MT M que indica:

- El contenido de la cinta es
 $\dots B B B s_1 s_2 \dots s_i \dots s_n B B B \dots$
- El estado actual de M es q
- El cabezal se encuentra barriendo el símbolo s_i

Importante: Γ y Q son disjuntos ($\Gamma \cap Q = \emptyset$)

Descripción Instantánea de una MT

Denotaremos $s_1 s_2 \dots q s_i \dots s_n$ a la **configuración o descripción instantánea** de la MT M que indica:

- El contenido de la cinta es

$\dots B B B s_1 s_2 \dots s_i \dots s_n B B B \dots$

- El estado actual de M es q
- El cabezal se encuentra barriendo el símbolo s_i

Importante: Γ y Q son disjuntos ($\Gamma \cap Q = \emptyset$)

Descripción Instantánea de una MT

Denotaremos $s_1 s_2 \dots q s_i \dots s_n$ a la **configuración o descripción instantánea** de la MT M que indica:

- El contenido de la cinta es

$\dots B B B s_1 s_2 \dots s_i \dots s_n B B B \dots$

- El estado actual de M es q

- El cabezal se encuentra barriendo el símbolo s_i

Importante: Γ y Q son disjuntos ($\Gamma \cap Q = \emptyset$)

Descripción Instantánea de una MT

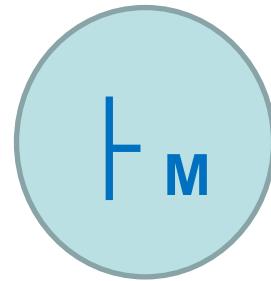
Denotaremos $s_1 s_2 \dots q s_i \dots s_n$ a la **configuración o descripción instantánea** de la MT M que indica:

- El contenido de la cinta es

$\dots B B B s_1 s_2 \dots s_i \dots s_n B B B \dots$

- El estado actual de M es q
- El cabezal se encuentra barriendo el símbolo s_i

Importante: Γ y Q son disjuntos ($\Gamma \cap Q = \emptyset$)

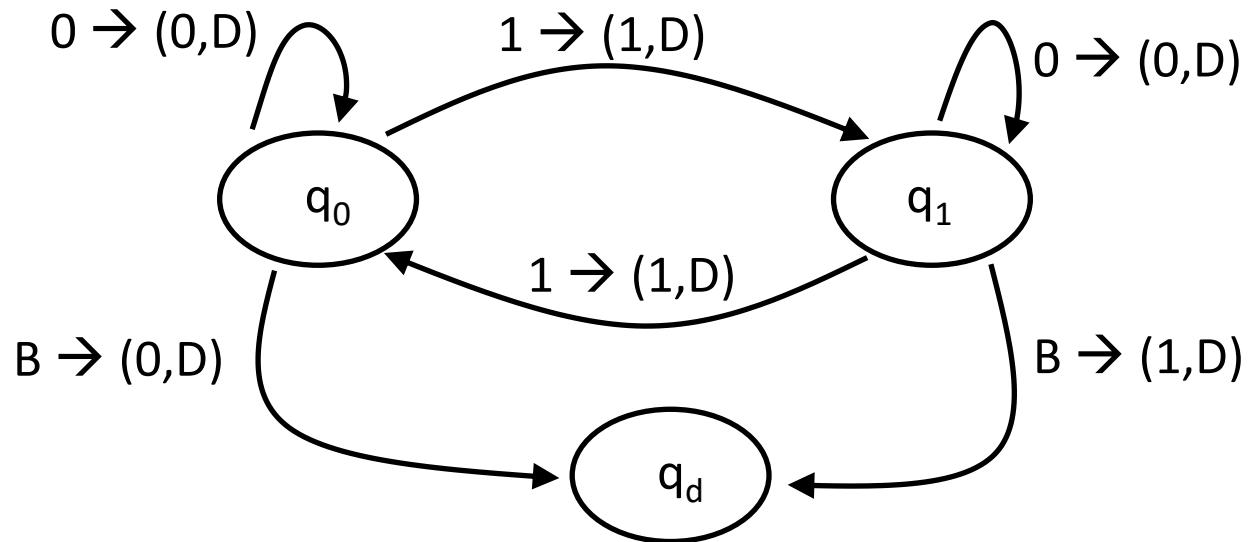


Movimiento o Paso de una MT

Movimiento de Cómputo/Computación:

- La expresión $C_1 \vdash_M C_2$ indica que la máquina de Turing M pasa en un solo movimiento o paso, de la configuración C_1 a la configuración C_2 .
- La expresión $C_1 \vdash_M^* C_2$ indica que la máquina de Turing M pasa en cero o más pasos de C_1 a C_2 .
- Traza: Secuencia de movimientos

Traza, Ejemplo



Para la MT M del grafo, con la entrada 100101

$q_0 100101 \vdash_M 1 q_1 00101 \vdash_M 10 q_1 0101 \vdash_M 100 q_1 101 \vdash_M$
 $1001 q_0 01 \vdash_M 10010 q_0 1 \vdash_M 100101 q_1 B \vdash_M 1001011 q_d B$

Lenguajes formales

Contexto: Teoría de Autómatas y Lenguajes Formales

- Veremos a los lenguajes desde el punto de vista de su aplicación a problemas de computación
- Lenguajes con estructuras sintácticas más o menos complicadas se asocian a problemas más o menos complicados de computar.

Lenguajes formales

Los Lenguajes son conjuntos de **sentencias** (*strings* o *cadenas*) construidas a partir de un conjunto finito de símbolos (el alfabeto).

Cada una de las sentencias de un lenguaje es una **secuencia finita** de estos símbolos.

Lenguajes formales

Sintaxis vs. Semántica

• **Sintaxis:** Principios y procesos que permiten combinar los símbolos para formar las sentencias de un lenguaje particular. Corresponde a la pregunta ¿Es gramaticalmente correcto?

• **Semántica:** Mecanismo subyacente a través del cual se le asigna un significado a las sentencias de un lenguaje particular. Corresponde a las preguntas ¿Qué significa esta sentencia? ¿Qué sentencia tiene sentido?

Es claro que para que una sentencia tenga sentido es conveniente que sea sintácticamente correcta.

Para trabajar con lenguajes formales sólo hace falta observar la **sintaxis** (las formas).

Lenguajes formales

Desde el punto de vista sintáctico (es decir en el contexto de los lenguajes formales) existen dos cuestiones importantes:

La generación: Gramáticas para generar sentencias sintácticamente correctas.

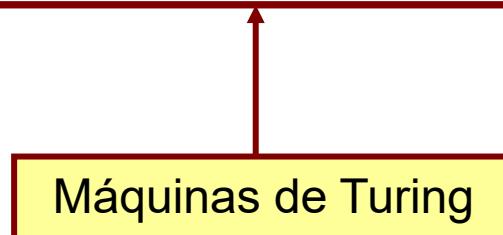
El reconocimiento o aceptación: Autómatas capaces de reconocer si una sentencia es sintácticamente correcta para un determinado lenguaje. Las MT son un ejemplo de un tipo particular de autómata.

Lenguajes formales

Clases de lenguajes (Chomsky)

Según el tipo de autómata que lo acepte o gramática que lo genere:

- Lenguajes **Regulares**
- Lenguajes **Libres de contexto**
- Lenguajes **Sensibles al contexto**
- Lenguajes **Recursivos y Recursivos Enumerables**



Lenguajes formales

Clases de lenguajes (Chomsky)



Definiciones

Definición. Alfabeto: Diremos que un conjunto finito Σ es un alfabeto si $\Sigma \neq \emptyset$ y $(\forall x)(x \in \Sigma \rightarrow x \text{ es un símbolo indivisible})$

Ejemplos $\Sigma = \{a, b\}$, $\Sigma = \{0, 1\}$, $\Sigma = \{a, b, \dots, z\}$ son alfabetos
 $\Sigma = \{0, 1, 00, 01\}$ $\Sigma = \{sa, ca, casa\}$ no lo son

Definiciones

Definicion. Palabra: Se dice que w es una palabra (cadena, sentencia o string) sobre Σ si w es una **secuencia finita** de símbolos de Σ

Ejemplos: si $\Sigma = \{0,1\}$, entonces:

0011, 101, 1 son palabras sobre Σ

Definiciones

Definicion. Longitud de una palabra: Se denota $|w|$, es el número de símbolos que contiene w .

Por ejemplo: $|\text{perro}|=5$ $|010|=3$

Nota: notaremos con Σ^* al conjunto de todas las palabras formadas por símbolos de Σ incluida la cadena nula (o vacía) que tiene longitud cero y denotaremos con λ ($|\lambda| = 0$)

Ejemplo: $\Sigma = \{a,b\}$

$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa\dots\}$

Definiciones

Concatenación: La notación utilizada para denotar la concatenación de dos palabras w y v es $w.v$ (o simplemente wv).

La concatenación es asociativa pero no conmutativa:

$$(v.w).x = v.(wx) \quad v.w \neq w.v$$

Se cumple que:

$$|v.w| = |v| + |w|$$

La cadena vacía es el elemento neutro para la concatenación $\lambda.w = w.\lambda = w$

Definiciones

Definición. Sea una palabra $w \in \Sigma^*$ y un número natural i , se define la **potencia i-ésima** de w como:

$$\begin{aligned} w^0 &= \lambda \\ w^{(i+1)} &= w \cdot w^i \quad (\forall i) (i \geq 0) \end{aligned}$$

Ejemplo: si $w = ab$, $w^3 = ababab$

Definiciones

Definición. Se denomina lenguaje definido sobre Σ a cualquier subconjunto de Σ^*

Ejemplo: si $\Sigma = \{0, 1\}$

\emptyset , Σ^* , $\{\lambda\}$, $\{w \in \Sigma^* / w \text{ comienza con } 1\}$ son lenguajes sobre Σ

Si L es un lenguaje sobre Σ , su complemento también lo es (Complemento de L respecto de Σ^* es $\bar{L} = \Sigma^* - L$)

Definiciones

Nota: Llamaremos \mathcal{L} al conjunto de todos los lenguajes definidos sobre el alfabeto Σ , es decir:

$$\mathcal{L} = \rho(\Sigma^*)$$

Máquina de Turing como reconocedoras de cadenas de símbolos

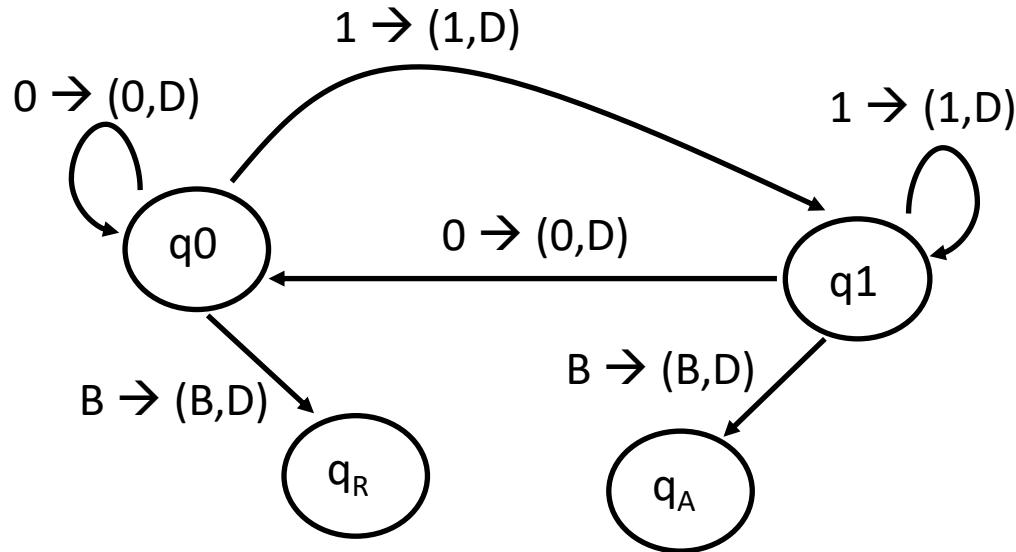
Para este tipo de máquina de Turing vamos a considerar dos estados de parada:

- q_A : el estado de aceptación.
- q_R : el estado de rechazo.
- Se define δ de manera completa, considerando todos los estados de Q . Sin embargo, ni q_A ni q_R pertenecen a Q .

Máquina de Turing como reconocedoras de cadenas de símbolos

Ejemplo 1

$$\Sigma = \{0,1\} \quad \Gamma = \{0, 1, B\}$$



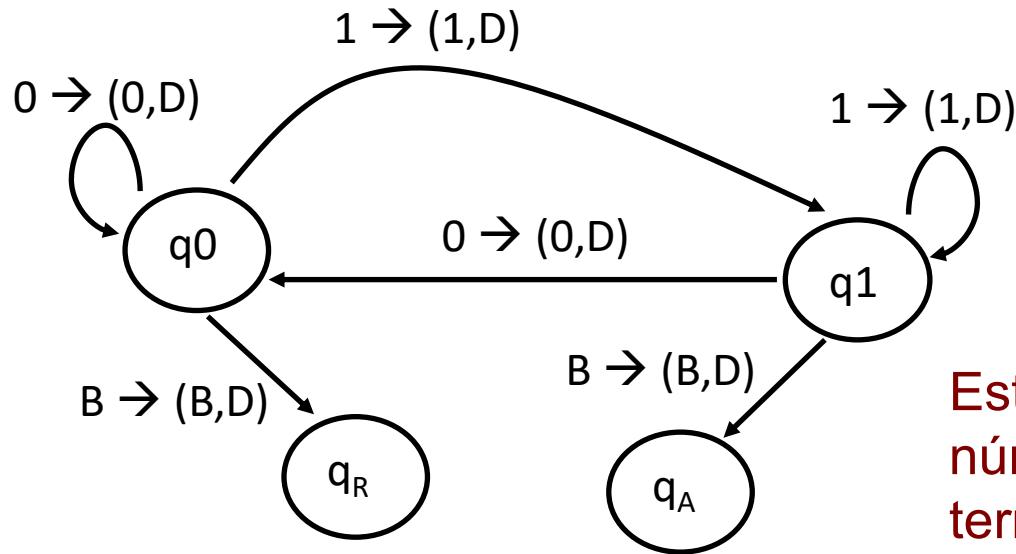
¿Qué strings reconoce esta máquina?

Es decir ¿para qué cadenas de símbolos la máquina se detiene en q_A ?

Máquina de Turing como reconocedoras de cadenas de símbolos

Ejemplo 1

$$\Sigma = \{0,1\} \quad \Gamma = \{0, 1, B\}$$



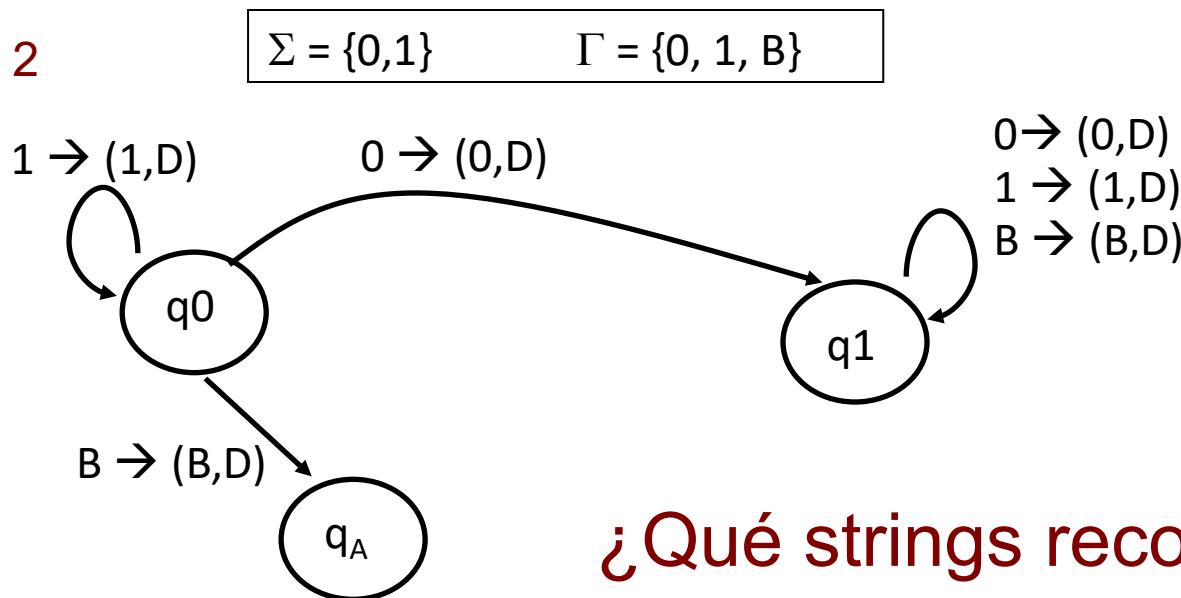
Esta máquina reconoce números binarios terminados en 1
 $L = \{w1, w \in \Sigma^*\}$

Observar que:

- No hay links que salen de los estados q_A y q_R .
- δ está definida para todos los demás casos

Máquina de Turing como reconocedoras de cadenas de símbolos

Ejemplo 2

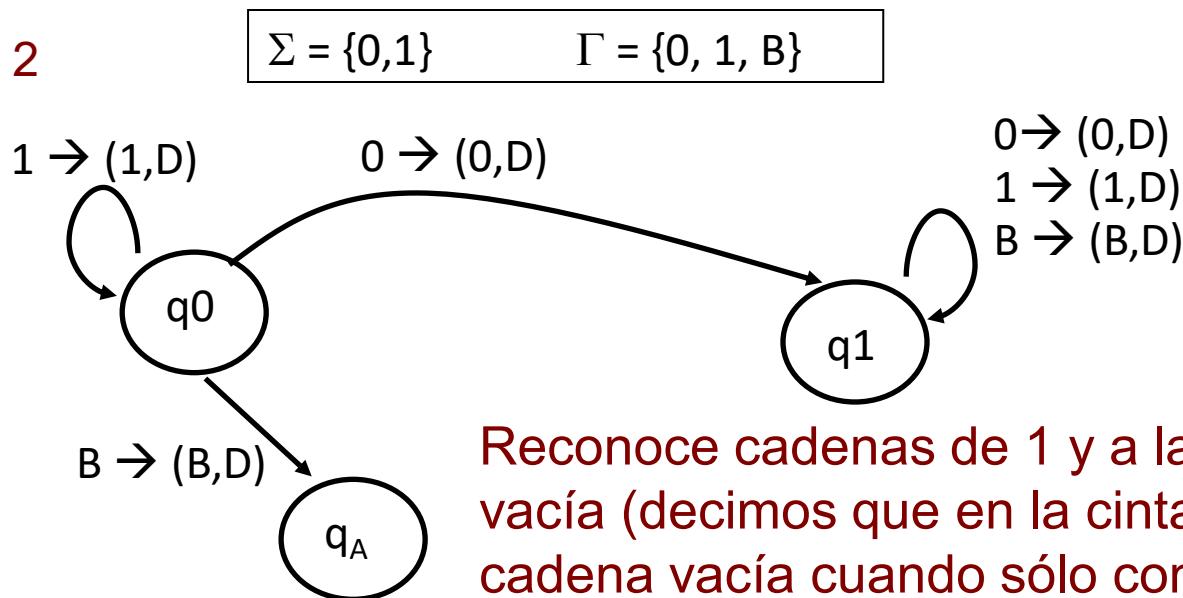


¿Qué strings reconoce esta máquina?

Es decir ¿para qué cadenas de símbolos la máquina se detiene en q_A ?

Máquina de Turing como reconocedoras de cadenas de símbolos

Ejemplo 2



Reconoce cadenas de 1 y a la cadena vacía (decimos que en la cinta está la cadena vacía cuando sólo contiene símbolos B)

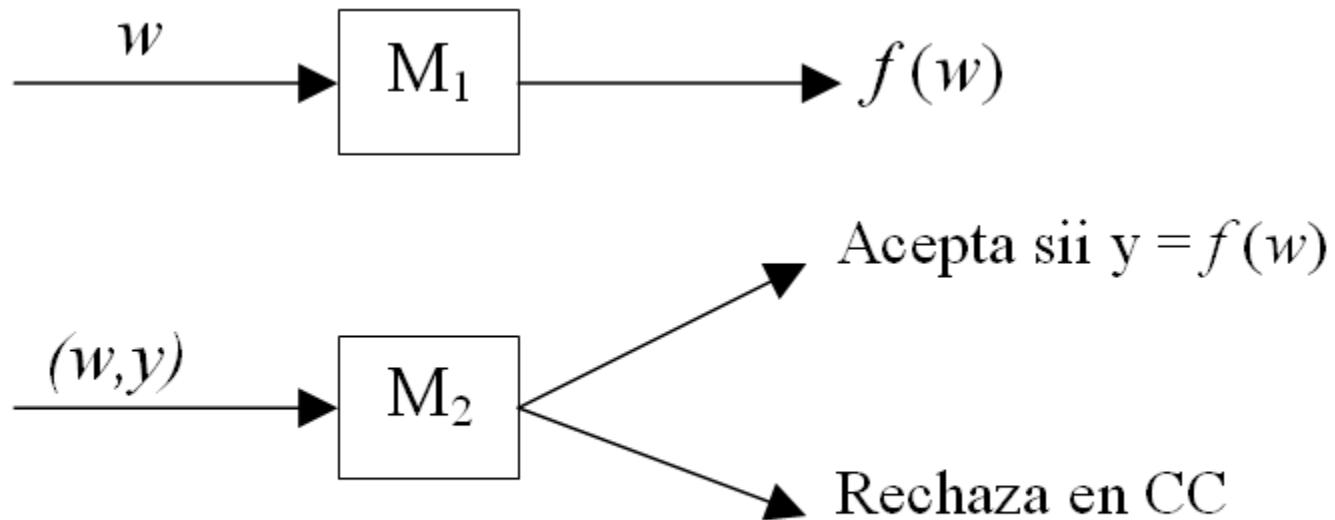
$$L = \{1^n, n \geq 0\} \quad (\lambda \in L, n = 0)$$

Observar que:

- No hay transiciones al estado q_R (es perfectamente válido)
- Se rechazan las cadenas "loopando" (lazo o loop infinito), es decir, la máquina no se detiene nunca.

Máquina de Turing como reconocedoras de cadenas de símbolos

Para el estudio de la computabilidad podemos quedarnos únicamente con las máquinas de Turing reconocedoras sin perder generalidad. Intuitivamente:



M_2 es una máquina de Turing reconocedora

Si se puede computar $f(w)$ se puede reconocer el lenguaje de los pares $(w, f(w))$ y viceversa

Formalización del modelo de Máquina de Turing q_A , q_R

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle \text{ con } q_A, q_R \notin Q$$

Donde:

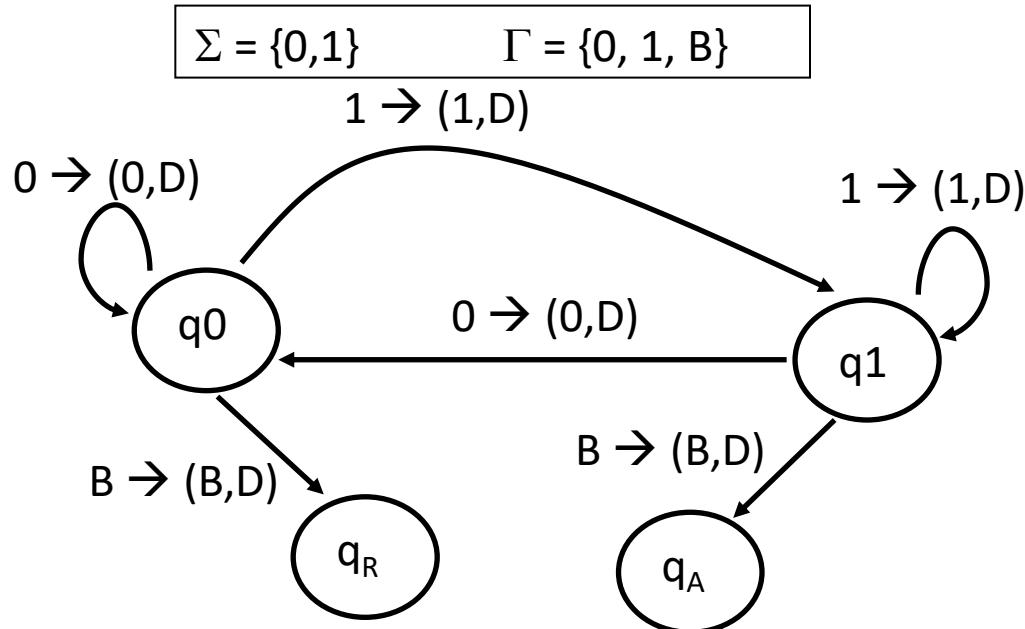
Q, Σ, Γ y q_0 se definen como en el caso de las MT de cómputo

$\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$, completa

q_A es el estado de aceptación, q_R es el estado de rechazo

M se detiene $\Leftrightarrow M$ pasa al estado q_A o q_R

Ejemplo (revisitado)



$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$$

$$Q = \{q_0, q_1\} \quad \Sigma = \{0, 1\} \quad \Gamma = \{B, 0, 1\}$$

$$\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$$

$$\delta(q_0, 0) = (q_0, 0, D) \quad \delta(q_0, 1) = (q_1, 1, D) \quad \delta(q_0, B) = (q_R, B, D)$$

$$\delta(q_1, 0) = (q_0, 0, D) \quad \delta(q_1, 1) = (q_1, 1, D) \quad \delta(q_1, B) = (q_A, B, D)$$

Ejemplo (revisitado)

Otra forma de especificar la función de transición δ es con una tabla

δ	0	1	B
q_0	$(q_0, 0, D)$	$(q_1, 1, D)$	(q_R, B, D)
q_1	$(q_0, 0, D)$	$(q_1, 1, D)$	(q_A, B, D)

Ejemplo - Traza

δ	0	1	B
q0	(q0,0,D)	(q1,1,D)	(q _R ,B,D)
q1	(q0,0,D)	(q1,1,D)	(q _A ,B,D)

Traza de ejecución (todos los movimientos) de la MT que reconoce secuencias binarias terminadas en 1 para la entrada $w=0101$

$$q_0 0101 \vdash_M 0 q_0 101 \vdash_M 01 q_1 01 \vdash_M 010 q_0 1 \vdash_M 0101 q_1 B \vdash_M 0101 B q_A B$$

Se puede escribir entonces:

$$q_0 0101 \vdash_M^* 0101 B q_A B$$

Lenguaje Aceptado por una MT

Definición: El lenguaje aceptado por una MT
 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ es:

$$L(M) = \{ w \in \Sigma^* / q_0 w \vdash_M^* \alpha q_A \beta, \alpha \beta \in \Gamma^* \}$$

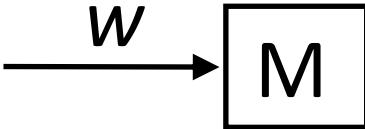
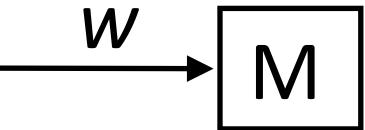
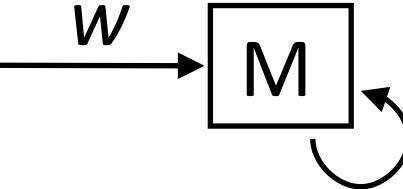
Dicho de otra forma:

$w \in L(M) \Leftrightarrow$ con entrada w , M para en q_A

Nota: Para las cadenas que no pertenecen a $L(M)$
la MT M para en q_R o loopea

Lenguaje Aceptado por una MT

Por cada entrada, hay 3 casos posibles:

- 1)  M se detiene en $q_A \Rightarrow w \in L(M)$
- 2)  M se detiene en $q_R \Rightarrow w \notin L(M)$
- 3)  M no se detiene $\Rightarrow w \notin L(M)$

Observar que: una MT M reconocedora realiza una partición de Σ^* en dos conjuntos: $L(M)$ y $\overline{L(M)}$, donde $\overline{L(M)} = \Sigma^* - L(M)$

Lenguaje Aceptado por una MT

Ejercicios:

- a) Construir una máquina de Turing M tal que $L(M) = \{0^n1^n / n \geq 1\}$ y describir los movimientos de la máquina (traza de computación) para las entradas $w_1=0011$ y $w_2=011$
- b) Construir una máquina de Turing que busque en la cinta el patrón “abab” y se detenga si y sólo si encuentra ese patrón. $\Sigma = \{a,b,c\}$

Modelos de MT

- Existen numerosos modelos de MT que se han probado equivalentes
- **Definición.** Dos MT M_1 y M_2 son equivalentes si $L(M_1)=L(M_2)$
- **Definición.** Dos modelos de MT son equivalentes si para cada MT de un modelo existe una MT equivalente en el otro modelo.

Modelos de MT

- **Teorema:** El **modelo q_A - q_R** de MT visto **es equivalente** al modelo de MT que comienza su computación apuntando al **último símbolo de la cadena** de entrada (al que llamaremos también "*ultsim*" para abbreviar).
- **Demostración.** Hay que probar 2 cosas:
 - 1) Para toda MT **M** del **modelo q_A - q_R** existe una MT **M'** equivalente que comienza con el cabezal apuntando al último símbolo de la cadena de entrada
 - 2) para toda MT **M'** del modelo ***ultsim*** existe una MT **M** del **modelo q_A - q_R** equivalente.

Modelos de MT

1) Vamos a probar que para toda MT M del modelo $q_A - q_R$ existe una máquina de MT M' del modelo *ultsim* equivalente.

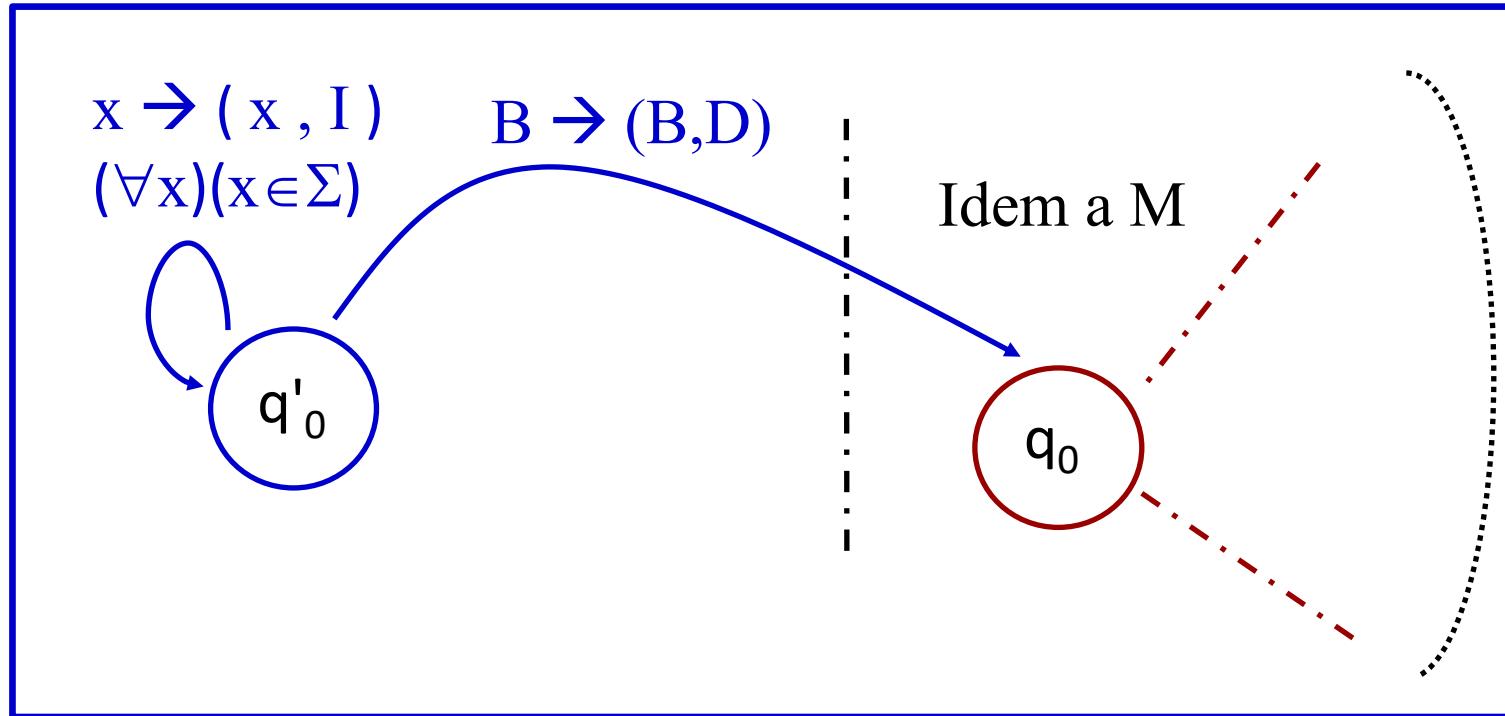
Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ una MT arbitraria del modelo $q_A - q_R$, construiremos M' del modelo *ultsim* tal que $L(M')$ sea igual a $L(M)$

$$M' = \langle Q', \Sigma, \Gamma, \delta', q'_0, q_A, q_R \rangle$$

con $Q' = Q \cup \{q'_0\}$ y $q'_0 \notin Q$

Modelos de MT

M'



Notar que la entrada tiene solo símbolos de Σ
¿Qué pasa si la entrada es λ ?

Modelos de MT

Definimos $\delta': Q' \times \Gamma \rightarrow Q' \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$

a) Si $\delta(q_i, x) = (q_j, y, Z)$ con $x, y \in \Gamma, Z \in \{D, I\}$

definimos $\delta'(q_i, x) = (q_j, y, Z)$

(con esto se tiene en M' lo mismo que en M)

b) y agregamos las siguientes transiciones:

$\delta'(q'_0, x) = (q'_0, x, I), \quad (\forall x)(x \in \Sigma)$

$\delta'(q'_0, B) = (q_0, B, D)$

(con esto M' queda apuntando al inicio y en estado q_0)

Hay que probar $L(M) = L(M')$

i) $L(M) \subseteq L(M')$

ii) $L(M') \subseteq L(M)$

Sea $w = s_1s_2\dots s_n \in L(M)$ (si $n = 0$ entonces $w = \lambda$)

$\Rightarrow q_0s_1s_2\dots s_n \vdash_M^* \alpha q_A \beta$ (por def. $L(M)$)

Para M' se cumple que:

$s_1s_2\dots q'_0s_n \vdash_{M'}^* q_0s_1s_2\dots s_n \vdash_{M'}^* \alpha q_A \beta$

\downarrow
(por def. $\delta' b$)

\downarrow
(por def. $\delta' a$)

$\Rightarrow s_1s_2\dots q'_0s_n \vdash_{M'}^* \alpha q_A \beta$, (por def. \vdash^*)

$\Rightarrow w \in L(M')$ por def. $L(M')$ y también se cumple para $w = \lambda$

Por lo tanto $L(M) \subseteq L(M')$ ✓

Hay que probar $L(M) = L(M')$

i) $L(M) \subseteq L(M')$ ✓

ii) $L(M') \subseteq L(M)$ Usaremos contrarecíproca ($w \notin L(M) \Rightarrow w \notin L(M')$)

Sea $w = s_1s_2\dots s_n$ tal que $w \notin L(M)$, por def. de $L(M)$ se tienen dos casos:

- A) M se detiene en q_R con entrada w
- B) M no se detiene con entrada w

Traza de M : $q_0s_1.s_2\dots s_n \vdash_M^* \alpha q_R \beta$,

Traza de M' :

$$s_1s_2\dots q'_0s_n \vdash_{M'}^* q_0s_1s_2\dots s_n \vdash_M^* \alpha q_R \beta$$

\downarrow
(por def. $\delta' b$)

\downarrow
(por def. $\delta' a$)

$$\Rightarrow s_1s_2\dots q'_0s_n \vdash_{M'}^* \alpha q_R \beta \quad (\text{por def. } \vdash^*)$$

$$\Rightarrow w \notin L(M') \quad (\text{por def. } L(M')) \quad \checkmark$$

Observe que también se cumple para $w=\lambda$

Hay que probar $L(M) = L(M')$

i) $L(M) \subseteq L(M')$ ✓

ii) $L(M') \subseteq L(M)$ Usaremos contrarecíproca ($w \notin L(M) \Rightarrow w \notin L(M')$)

Sea $w = s_1s_2\dots s_n$ tal que $w \notin L(M)$, por def. de $L(M)$ se tienen dos casos:

A) M se detiene en q_R con entrada $w \Rightarrow w \notin L(M')$ ✓

B) M no se detiene con entrada w

A partir de $q_0s_1s_2\dots s_n$ M nunca se detiene

Para M' se cumple que:

$s_1s_2\dots q'_0s_n \xrightarrow{M'} q_0s_1s_2\dots s_n$ y a partir de aquí M' loopea

(por def. $\delta' b$)

(por def. $\delta' a$)

\Rightarrow A partir de $s_1s_2\dots q'_0s_n$ M' nunca se detiene

$\Rightarrow w \notin L(M')$ (por def. $L(M')$) ✓

Observe que también se cumple para $w=\lambda$

Hay que probar $L(M) = L(M')$

i) $L(M) \subseteq L(M')$ ✓

ii) $L(M') \subseteq L(M)$ Usaremos contrarecíproca ($w \notin L(M) \Rightarrow w \notin L(M')$)

Sea $w = s_1s_2\dots s_n$ tal que $w \notin L(M)$, por def. de $L(M)$ se tienen dos casos:

A) M se detiene en q_R con entrada $w \Rightarrow w \notin L(M')$ ✓

B) M no se detiene con entrada $w \Rightarrow w \notin L(M')$ ✓

Por lo tanto si $w \notin L(M) \Rightarrow w \notin L(M')$ (por casos A y B)

Por contrarecíproca si $w \in L(M') \Rightarrow w \in L(M)$.

Por lo tanto $L(M') \subseteq L(M)$. ✓

Hay que probar $L(M) = L(M')$

- i) $L(M) \subseteq L(M')$ ✓ ii) $L(M') \subseteq L(M)$ ✓ $\Rightarrow L(M) = L(M')$

Se ha demostrado que para toda MT M del **modelo $q_A - q_R$** existe una MT M' equivalente del modelo *ultsim*

Para demostrar que ambos modelos son equivalentes faltaría demostrar que para toda MT M' del modelo *ultsim* existe una MT M del **modelo $q_A - q_R$** equivalente (la demostración es análoga),

Modelos de MT

Modelo D-I-S (Derecha-Izquierda-Sin movimiento)

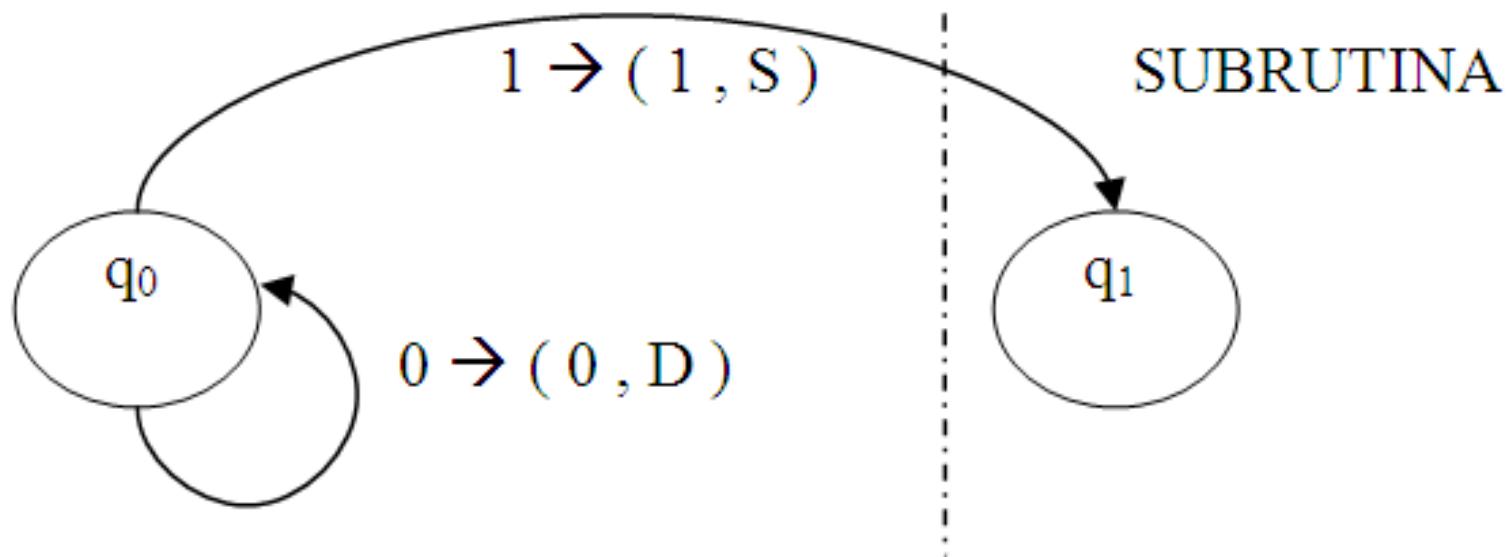
Máquina de Turing que admite transiciones sin movimiento del cabezal de la cinta.

$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ con $Q, \Sigma, \Gamma, q_0, q_A, q_R$, definidos como en el modelo de referencia (que llamaremos modelo D-I en este caso)

y $\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I, S\}$
(S significa sin movimiento)

Modelos de MT

Ejemplo: Construir una máquina de Turing que se posicione en el primer símbolo ‘1’ del input de la cinta para luego saltar a una subrutina ($\Sigma = \{0,1\}$)



Modelos de MT

Teorema: Los modelos de máquinas de Turing D-I-S y D-I son equivalentes

Preguntas:

- ¿Qué se necesita demostrar?
- ¿Alguna demostración es trivial? ¿Por qué?

Modelos de MT

Se demuestra trivialmente que para toda MT del modelo D-I existe una MT equivalente del modelo D-I-S, pues las máquinas del modelo D-I son un caso particular de las del modelo D-I-S

Modelos de MT

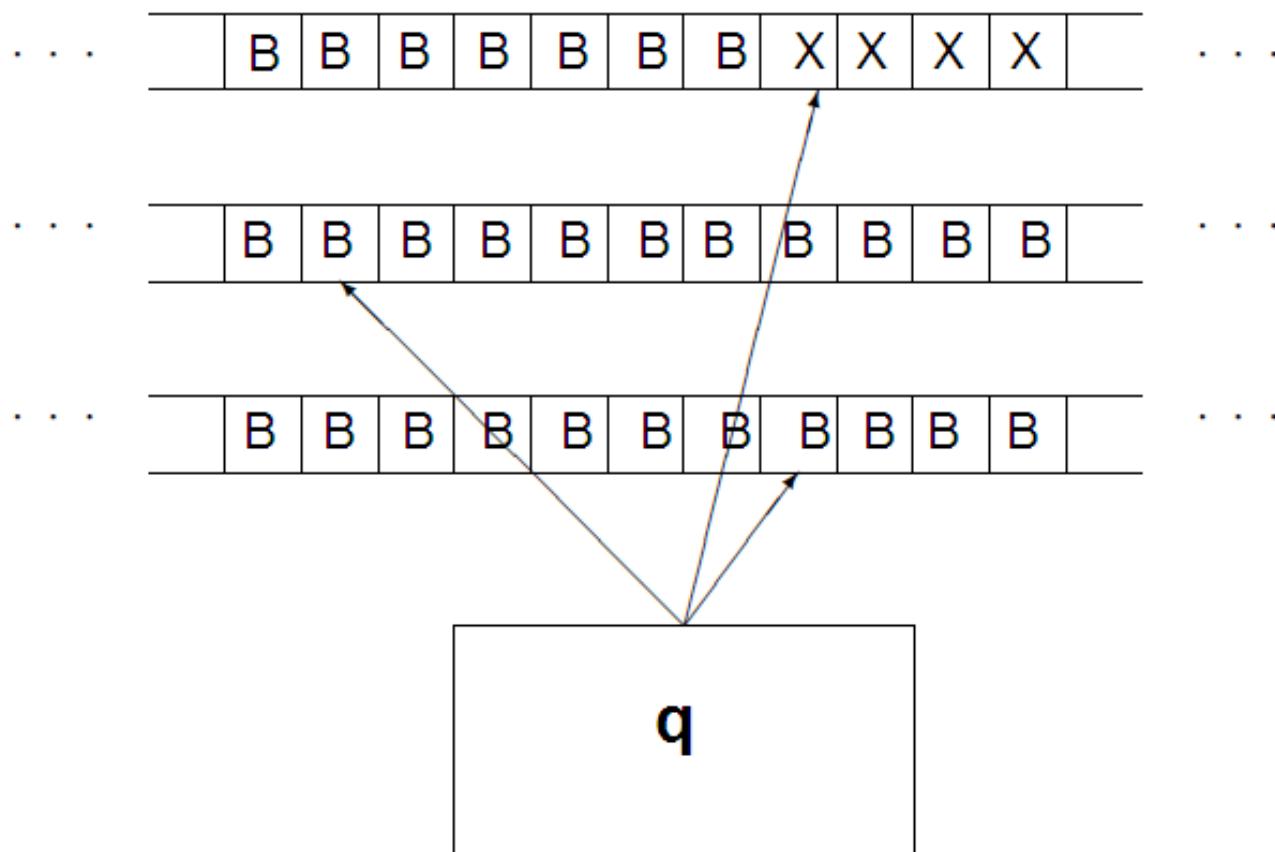
- **Ejercicio 1:** Demostrar que para toda MT del modelo D-I-S existe una MT del modelo D-I equivalente
- **Ejercicio 2:** Demostrar que para toda MT del modelo D-I-S existe una MT M' equivalente con la restricción de no poder cambiar el símbolo de la cinta y mover el cabezal simultáneamente

Modelo de MT de k Cintas

- Consiste en un control con k cintas y k cabezales que pueden moverse en forma independiente.
- La entrada se encuentra en la primera cinta y todas las demás están en blanco.

Modelo de MT de k Cintas

Máquina de Turing de 3 cintas



Modelo de MT de k Cintas

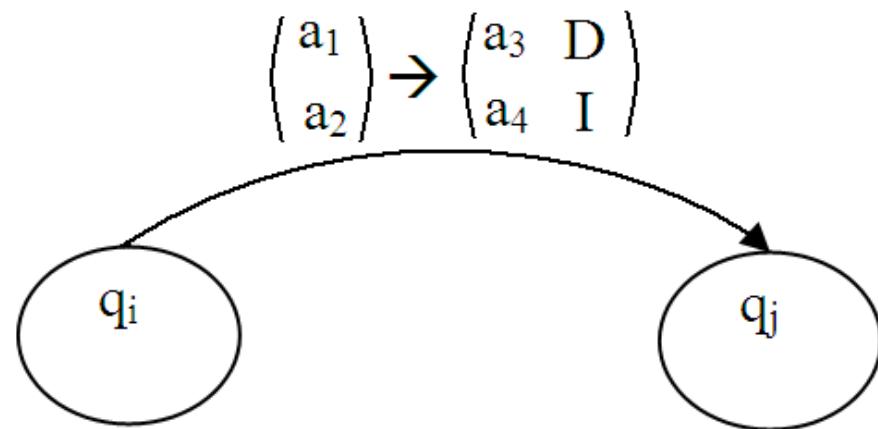
$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$$

con $Q, \Sigma, \Gamma, q_0, q_A, q_R$ definidos como en el modelo estándar D-I-S de una cinta

$$\text{y } \delta: Q \times \Gamma^k \rightarrow Q \cup \{q_A, q_R\} \times (\Gamma \times \{D, I, S\})^k$$

Modelo de MT de k Cintas

Ejemplo: $\delta(q_i, (a_1, a_2)) = (q_j, (a_3, D), (a_4, I))$



Estando en el estado q_i , al leer a_1 en la primera cinta y a_2 en la segunda, escribe a_3 en la primera y a_4 en la segunda, mueve a la derecha el cabezal de la primera cinta y a la izquierda en de la segunda cinta.

Modelo de MT de k Cintas

NOTA: Puede probarse que este modelo multicinta es equivalente a cualquiera de los que ya hemos visto.

Modelo de MT de k Cintas

Ejercicio: Definir la δ de transición de una MT con 2 cintas que reconozca el lenguaje:

$$L = \{ 0^n 1^n / n \geq 1 \}$$

$$\delta(q_0, (0, B)) = (q_0, (0, D), (0, D))$$

$$\delta(q_0, (1, B)) = (q_1, (1, S), (B, I))$$

$$\delta(q_1, (1, 0)) = (q_1, (1, D), (0, I))$$

$$\delta(q_1, (B, B)) = (q_A, (B, S), (B, S))$$

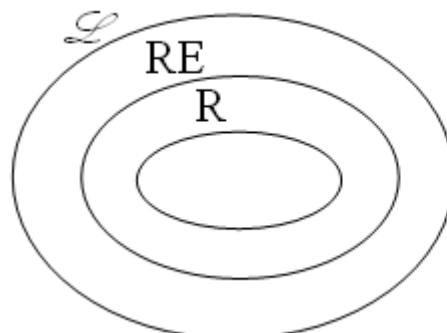
Las transiciones que faltan van todas a q_R

Caracterización de Lenguajes

Def.: un lenguaje L es recursivamente enumerable (RE) sii existe una MT M que lo acepte, es decir $L = L(M)$.

Def.: un lenguaje L es recursivo (R) o decidable sii existe una MT M tal que $L = L(M)$ y M siempre se detiene para todo input de Σ^*

Recordar: Dado un alfabeto Σ , denotamos con Σ^* al conjunto de todas las cadenas formadas por símbolos de Σ . \mathcal{L} es el conjunto de todos los lenguajes definidos sobre el alfabeto Σ , es decir $\mathcal{L} = \rho(\Sigma^*)$, es decir el conjunto de todos los subconjuntos posibles de Σ^* . Se tiene la siguiente situación:



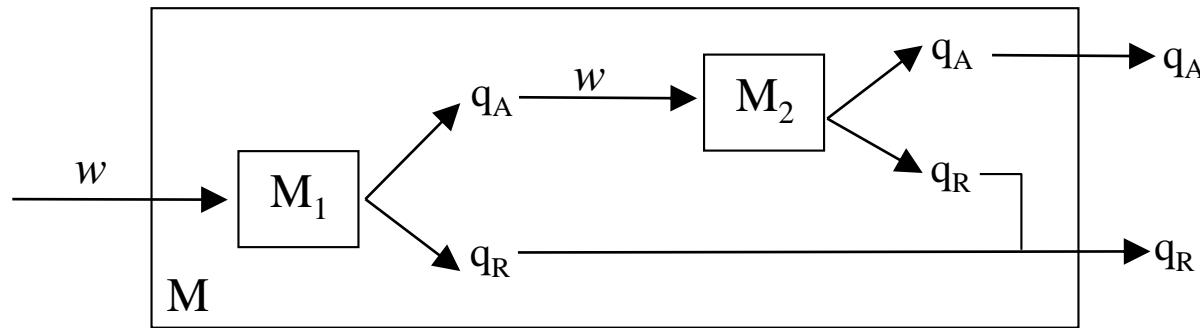
$$\underbrace{R \subseteq RE \subseteq \mathcal{L}}_{\text{por las definiciones}}$$

Caracterización de Lenguajes

Interrogantes: ¿Las inclusiones son propias?

Es decir $\left\{ \begin{array}{l} \text{¿L-RE} \neq \emptyset? \\ \text{¿RE - R} \neq \emptyset? \end{array} \right.$

Ejercicio: Sean $L_1 \in R$ y $L_2 \in R$ ¿ $L_1 \cap L_2 \in R$?



Rta.: sí $L_1 \cap L_2 \in R$

Dem.: Sean M_1 y M_2 MT de **una sola cinta** tq $L_1 = L(M_1)$ y $L_2 = L(M_2)$

Además se eligen ambas MT tal que se detienen para toda entrada, seguro existen porque ambos lenguajes pertenecen a R

$$M_1 = \langle Q_1, \Sigma, \Gamma_1, \delta^1, q_0^1, q_A^1, q_R^1 \rangle$$

$$\text{con } Q_1 \cap Q_2 = \emptyset$$

$$M_2 = \langle Q_2, \Sigma, \Gamma_2, \delta^2, q_0^2, q_A^2, q_R^2 \rangle$$

Se construye una MT de **dos cintas** que funciona de la siguiente manera:

- 1) Copia la entrada en la segunda cinta y posiciona el cabezal en el principio de la entrada en la 2da. cinta.
- 2) Simula M_1 sobre la cinta 2. Si M_1 para en q_R^1 , M para en q_R , si M_1 para en q_A^1 ir al punto 3)
- 3) Borra la cinta 2
- 4) Copia w de la cinta 1 a la cinta 2.
- 5) Simula M_2 sobre w en la cinta 2.
Si M_2 para en q_R^2 , M para en q_R
Si M_2 para en q_A^2 , M para en q_A

¿ Cómo sería concretamente la codificación de esta MT ?

Para formalizar: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$

$\delta: Q \times \Gamma^2 \rightarrow Q \cup \{q_A, q_R\} \times (\Gamma \times \{D, I, S\})^2$

con $Q_1 \cup \{q_A^1, q_R^1\} \cup Q_2 \cup \{q_A^2, q_R^2\} \subseteq Q$; $\Gamma_1 \cup \Gamma_2 \subseteq \Gamma$

$\delta(q_i, a, b) = (q_j, (c, m_1), (d, m_2))$ con $q_i \in Q; q_j \in Q \cup \{q_A, q_R\}; a, b, c, d \in \Gamma; m_1, m_2 \in \{D, I, S\}$

Símbolos en cinta 1 y cinta 2 → en la cinta 1 → en la cinta 2

- 1) Copia la entrada en la segunda cinta y posiciona el cabezal en el principio de la entrada en la 2da. cinta.

$\delta(q_0, (x, B)) = (q_0, (x, D), (x, D)) \quad (\forall x)(x \in \Sigma)$ (Copia la entrada en cinta 2)

$\delta(q_0, (B, B)) = (q_1, (B, S), (B, I))$

(Fin de copia, empieza a buscar el inicio del string en la cinta 2)

$\delta(q_1, (B, x)) = (q_1, (B, S), (x, I)) \quad (\forall x)(x \in \Sigma)$ (se dirige al inicio del string en la cinta 2)

$\delta(q_1, (B, B)) = (q_0^1, (B, S), (B, D))$

(Queda apuntando al inicio del string en la cinta 2, para comenzar la simulación de M_1 sobre la cinta 2)

Para formalizar: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$

$\delta: Q \times \Gamma^2 \rightarrow Q \cup \{q_A, q_R\} \times (\Gamma \times \{D, I, S\})^2$

con $Q_1 \cup \{q_A^1, q_R^1\} \cup Q_2 \cup \{q_A^2, q_R^2\} \subseteq Q$; $\Gamma_1 \cup \Gamma_2 \subseteq \Gamma$

$\delta(q_i, a, b) = (q_j, (c, m_1), (d, m_2))$ con $q_i \in Q; q_j \in Q \cup \{q_A, q_R\}; a, b, c, d \in \Gamma; m_1, m_2 \in \{D, I, S\}$

Símbolos en cinta 1 y cinta 2 → en la cinta 1 → en la cinta 2

2) Simula M_1 sobre la cinta 2. Si M_1 para en q_R^1 , M para en q_R , si M_1 para en q_A^1 ir al punto 3)

Para cada $\delta^1(q_i^1, x) = (q_j^1, y, m)$ se define

$\delta(q_i^1, (B, x)) = (q_j^1, (B, S), (y, m))$ (Simulación en cinta 2)

$\delta(q_R^1, (B, x)) = (q_R, (B, S), (x, S)) \quad \forall x \in \Gamma_1$ (si M_1 Rechaza, M también)

$\delta(q_A^1, (B, x)) = (q_3, (B, S), (x, S)) \quad \forall x \in \Gamma_1$ (M_1 Acepta ir al punto 3)

q_3 es el estado en el que comienza la ejecución del punto 3)

Para formalizar: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$

$\delta: Q \times \Gamma^2 \rightarrow Q \cup \{q_A, q_R\} \times (\Gamma \times \{D, I, S\})^2$

con $Q_1 \cup \{q_A^1, q_R^1\} \cup Q_2 \cup \{q_A^2, q_R^2\} \subseteq Q$; $\Gamma_1 \cup \Gamma_2 \subseteq \Gamma$

$\delta(q_i, a, b) = (q_j, (c, m_1), (d, m_2))$ con $q_i \in Q; q_j \in Q \cup \{q_A, q_R\}; a, b, c, d \in \Gamma; m_1, m_2 \in \{D, I, S\}$

Símbolos en cinta 1 y cinta 2 → en la cinta 1 → en la cinta 2

3) Borra la cinta 2

Pregunta: ¿Cómo saber cuánto borrar de la cinta 2? Tenga en cuenta que luego de simular M_1 la cinta posee cualquier string de Γ_1^*

Ejercicio1: De acuerdo a la estrategia elegida como respuesta a la pregunta anterior, completar la función delta de transición para los puntos 3), 4) y 5)

Ejercicio2: Demostrar que $L(M) = L(M_1) \cap L(M_2)$ y que M se detiene siempre. Con eso quedaría demostrado que $L_1 \cap L_2 \in R$.

Más definiciones

Def.: un lenguaje $L \in \text{Co-R}$ si $\overline{L} \in R$ (\overline{L} es el complemento de L respecto de Σ^* , es decir $\overline{L} = \Sigma^* - L$)

Def.: un lenguaje $L \in \text{Co-RE}$ si $\overline{L} \in \text{RE}$

Más interrogantes:

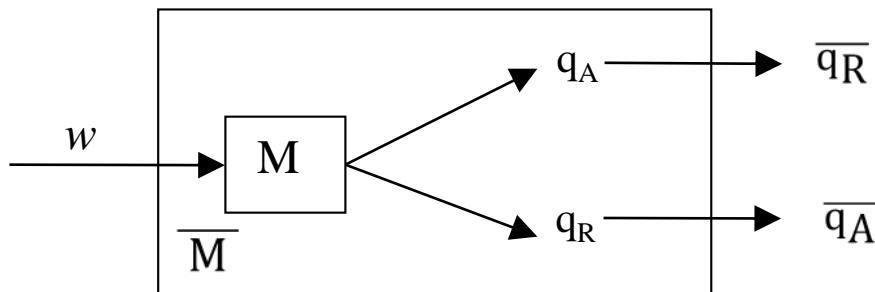
¿Qué relación habrá entre R , Co-R RE y Co-RE ?

Teorema 1: $R \subseteq \text{Co-R}$

Demostración. Hay que demostrar que si $L \in R \Rightarrow L \in \text{Co-R}$, es decir que si $L \in R \Rightarrow \overline{L} \in R$.

Sea $L \in R \Rightarrow$ Existe una MT $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ tq $L = L(M)$ y M se detiene en algún momento para toda entrada.

Se construye $\overline{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, \overline{q_A}, \overline{q_R} \rangle$ con $\overline{q_A} = q_R$ y $\overline{q_R} = q_A$



Hay que probar que $L(\overline{M}) = \overline{L}$ y además que \overline{M} se detiene en algún momento para toda entrada.

$$\begin{array}{ccccccc}
 1) \text{ Sea } w \in L(\overline{M}) & \Leftrightarrow q_0 w \vdash_{\overline{M}}^+ \alpha_1 \overline{q_A} \alpha_2 & \Leftrightarrow q_0 w \vdash_M^+ \alpha_1 q_R \alpha_2 & \Leftrightarrow w \notin L(M) & \Leftrightarrow w \notin L & \Leftrightarrow w \in \overline{L} \\
 \text{def.} L(\overline{M}) & & \text{Constr.} & & \text{def.} L(M) & \text{Hip.} & \text{def.} \overline{L}
 \end{array}$$

Por lo tanto, $L(\overline{M}) = \overline{L}$

2) ¿ \overline{M} se detiene para toda entrada?: Sí, por construcción \overline{M} se detiene cuando M se detiene y por hipótesis M se detiene para toda entrada.

De 1) y 2) $L \in R \Rightarrow L \in \text{Co-}R$

Por lo tanto, $R \subseteq \text{Co-}R$

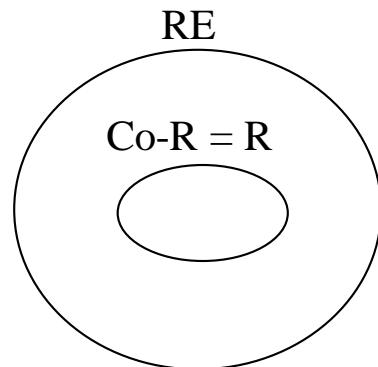
Teorema 2: $\text{Co-R} \subseteq R$

Sea $L \in \text{Co-R} \Rightarrow \overline{L} \in R \Rightarrow \overline{L} \in \text{Co-R} \Rightarrow \overline{\overline{L}} \in R \Rightarrow L \in R$

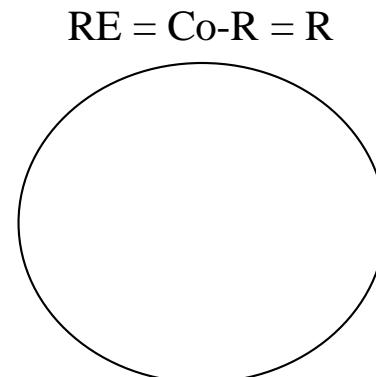
def.Co-R teor. ant. def. Co-R prop. de compl.

Por lo tanto, $\text{Co-R} \subseteq R$

Corolario: de los dos teoremas anteriores surge que $R = \text{Co-R}$



O bien

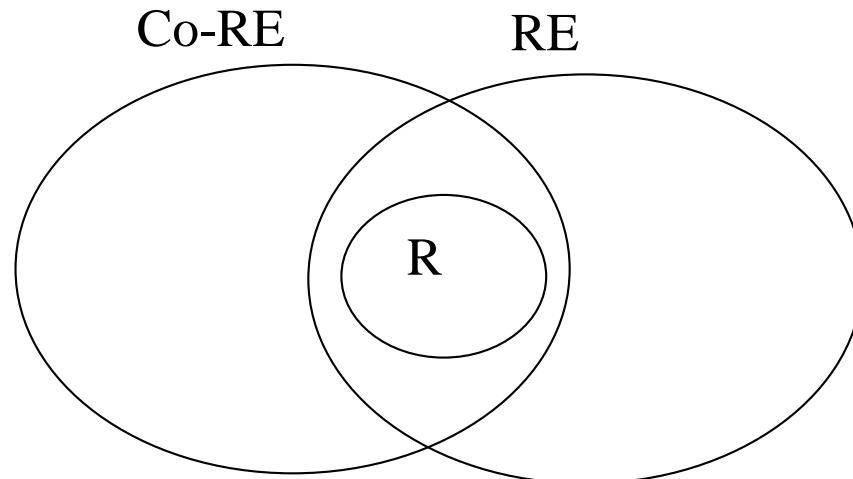


Teorema 3: $R \subseteq \text{Co-RE}$

Dem.: Sea $L \in R \Rightarrow \overline{L} \in R \Rightarrow \overline{L} \in \text{RE} \Rightarrow L \in \text{Co-RE}$
teor. 1 (def.R y RE) def. Co-RE

Por lo tanto, $R \subseteq \text{Co-RE}$.

Además, como por definición $R \subseteq \text{RE}$ se tiene que $R \subseteq (\text{RE} \cap \text{Co-RE})$



Teorema 4: $(\text{RE} \cap \text{Co-RE}) \subseteq \text{R}$

Sea $L \in (\text{RE} \cap \text{Co-RE})$

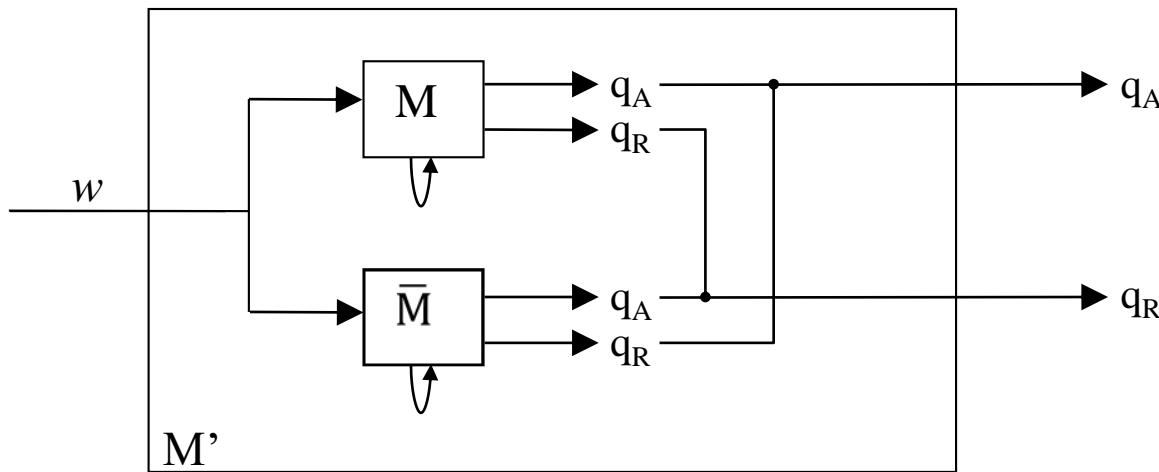
$$\Rightarrow L \in \text{RE} \wedge \overline{L} \in \text{RE} \quad (\text{por def. de } \cap \text{ y de Co-RE})$$

$$\Rightarrow \text{existen } M \text{ y } \overline{M}, \text{ dos MT tq } L = L(M) \text{ y } \overline{L} = L(\overline{M})$$

Hay que construir una MT M' que reconozca L y que se detenga siempre.

$\forall w \in \Sigma^*$ o bien \overline{M} para en q_A o bien M para en q_A (no puede darse nunca el caso que ambas "loopeen").

Por lo tanto hay que construir M' simulando en paralelo M y \overline{M} , si M para en q_A o \overline{M} para en $q_R \Rightarrow M'$ para en q_A y si M para en q_R o \overline{M} en $q_A \Rightarrow M'$ para en q_R .



¿Cómo simular dos máquinas en paralelo? No importa la eficiencia, en una máquina de 4 cintas:

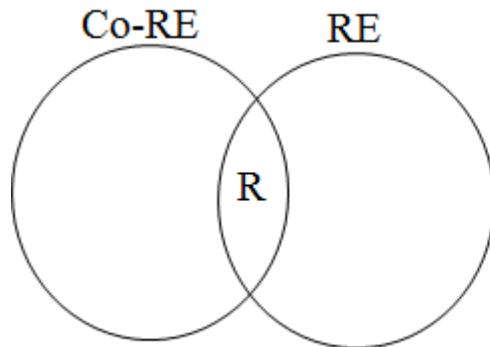
- 1) Escribir el número 1 en la cinta 4 (sea i ese valor).
- 2) Copiar w a las cintas 2 y 3.
- 3) Simular a lo sumo i pasos de M en la cinta 2 y a lo sumo i pasos de \overline{M} en la cinta 3. si M para en q_A o \overline{M} para en $q_R \Rightarrow M'$ para en q_A y si M para en q_R o \overline{M} para en $q_A \Rightarrow M'$ para en q_R .
- 4) Borrar las cintas 2 y 3. Incrementar i en la cinta 4 y volver al punto 2.

Demostrar como ejercicio que $L = L(M')$ y que M' se detiene siempre.

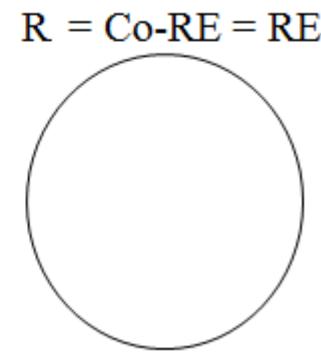
Pregunta: ¿Cómo se pueden simular i pasos?

Corolario: $(\text{RE} \cap \text{Co-RE}) = \text{R}$ (por los teoremas 3 y 4)

Por lo tanto hasta ahora nuestra situación es:



O bien



Def.: orden canónico para Σ^* : se listan todas las palabras en orden según su tamaño con las palabras del mismo tamaño en orden lexicográfico.

Ej: $\Sigma = \{0, 1\}$, el orden canónico es:

$\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001 \dots$

Obsérvese que si w es un string de $\{0,1\}^*$, la posición i que ocupa en el orden canónico se escribe en binario como $1w$. Decimos entonces que w es el i -ésimo string y por ello lo denotamos w_i

Por ejemplo el string λ ocupa la posición 1 (1λ) el string 01 ocupa la posición 5 (101), el string 0000 la posición 16 (10000)

Pregunta 1: ¿Puede una MT generar las palabras de $\{0,1\}^*$ en orden canónico?

Rta.: Sí. Idea: ir sumando 1 en binario, cuando el resultado necesita un bit más, se ponen todos los bits en cero y se vuelve al proceso de sumar uno en binario.

Ejercicios para el lector

Ej. 1) construir una MT que escriba en la primera cinta las palabras de $\{0, 1\}^*$ en orden canónico separadas por “,”

Ej. 2): construir una MT que genere las palabras de $\{a, b, c\}^*$ en orden canónico separadas por “,” es decir: $\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab \dots$

Pregunta 2: Si L es un lenguaje recursivo ($L \in R$) ¿Puede una MT generar todas las palabras de L en orden canónico?

Rta.: Sí, porque existirá alguna MT M que reconoce L y siempre se detiene. Se construye una MT M' que va generando en una cinta los strings de Σ^* en orden canónico, simula M sobre cada string generado y si M lo acepta M' lo escribe en la cinta 1.

Pregunta 3: si L es un lenguaje recursivamente enumerable ($L \in \text{RE}$) ¿Puede una MT generar todas las palabras de L ?

Rta.: Sí. Se generan todos los pares (i, j) en orden de su suma, $i+j$, y entre los de igual suma en orden creciente de i (ver ejercicio de la práctica 1)

(1, 1); (1, 2); (2, 1); (1, 3); (2, 2); (3, 1), ...

Por cada par (i, j) generado se simulan j pasos de la MT M que reconoce el lenguaje L ($L = L(M)$), sobre w_i (i-ésimo string de Σ^* en orden canónico). Si M acepta w_i en esos j pasos \Rightarrow se escribe w_i en la cinta 1.

Pregunta 4: ¿Puede codificarse una MT como un string de un alfabeto de 2 símbolos?

Rta.: Sí. Se puede hacer de muchas formas, acá se muestra una.

Ej: se quiere codificar $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$

$Q = \{q_0, q_1, \dots, q_K\}$ $\Sigma = \{a_1, a_2, \dots, a_L\}$ $\Gamma = \{B, a_1, a_2, \dots, a_L, a_{L+1}, \dots, a_n\}$

Se puede codificar M con un alfabeto binario $\{0, 1\}$ de la sgte. forma:

Estados: $q_A = 1$ $q_R = 11$ $q_0 = 111$ $q_1 = 1111$... $q_i = 1^{(i+3)}$

Símbolos: $B = 1$, $a_1 = 11$ $a_2 = 111$... $a_i = 1^{(i+1)}$

Movimiento del Cabezal: $D = 1$ $I = 11$ $S = 111$

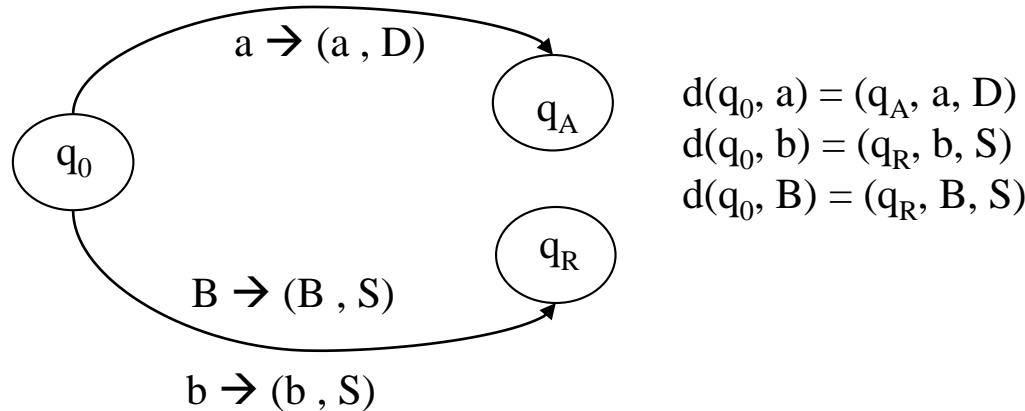
Función de Transición: cada transición se codifica como una quíntupla de elementos separados por un símbolo 0. Ej.:

$\delta(q_0, a_2) = (q_1, a_4, D)$ se codifica como

$(q_0, a_2, q_1, a_4, D) = (11101110111101111101)$
 q_0 , a_2 , q_1 , a_4 , D

M se codifica como una sucesión de quíntuplas separadas por 00.

Ej.: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ $Q = \{q_0\}$ $\Sigma = \{a, b\}$ $\Gamma = \{B, a, b\}$



Ej.: $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle$ $Q = \{q_0\}$ $\Sigma = \{a, b\}$ $\Gamma = \{B, a, b\}$

$q_A = 1$	$q_R = 11$	$q_0 = 111$
$B = 1$	$a = 11$	$b = 111$
$D = 1$	$I = 11$	$S = 111$

$\langle M \rangle = 1110110101101001110111011011101110011101011010111$
 $\delta(q_0, a) = (q_A, a, D)$ $\delta(q_0, b) = (q_R, b, S)$ $\delta(q_0, B) = (q_R, b, S)$

Note que existen otros posibles códigos para M . Las tres transiciones pueden listarse en distinto orden ($3!$ formas distintas) por lo tanto hay 6 códigos distintos para la misma M

Para pensar

¿Cuál sería la codificación de MT que tiene menor valor numérico?

Es decir: toda codificación de MT puede verse como un número binario, ¿cuál sería la MT codificada de "menor" valor numérico?

Todas tienen:

- Los estados q_0 , q_A y q_R
- Al menos 2 símbolos en Γ : B y 1 símbolo de Σ
- δ debe ser completa $\delta: Q \times \Gamma \rightarrow Q \cup \{q_A, q_R\} \times \Gamma \times \{D, I\}$, completa

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R \rangle \quad Q = \{q_0\} \quad \Sigma = \{a\} \quad \Gamma = \{B, a\}$$

$$\begin{array}{lll} q_A = 1 & q_R = 11 & q_0 = 111 \\ B = 1 & a = 11 & \\ D = 1 & I = 11 & S = 111 \end{array}$$

$$\langle M \rangle = \textcolor{blue}{111011010110100} \textcolor{red}{1110101010111}$$

$$\delta(q_0, a) = (q_A, a, D) \quad \delta(q_0, B) = (q_A, B, S)$$

¿Esta es la codificación de menor valor numérico?

Pregunta 5: ¿ $L_{\langle M \rangle} \in R$?

Con $L_{\langle M \rangle} = \{w \in \{0, 1\}^* / w \text{ es el código bien formado de una MT}\}$

Rta.: Sí, para demostrarlo hay que construir una MT que realice un chequeo sintáctico y siempre se detenga.

El código binario de una MT M se denotará con $\langle M \rangle$

Def. Se denomina *i*-ésima MT y se denota M_i a la MT cuyo código es w_i (*i*-ésimo string binario en orden canónico), es decir $\langle M_i \rangle = w_i$

Convención: Si w_i no es un código válido de MT se considera que codifica a M_i , siendo M_i una MT que se detiene inmediatamente rechazando cualquier entrada, $L(M_i) = \{\}$. Así, todo string w_i de $\{0,1\}^*$ se corresponde con, o codifica, una MT M_i

Def.: se define el lenguaje Diagonal L_D (primer ejemplo de lenguaje no recursivamente enumerable que se verá) de la siguiente manera:

$$L_D = \{w_i \in \Sigma^* / w_i \notin L(M_i)\}$$

siendo $\Sigma = \{0,1\}$; w_i el *i*-ésimo string en orden canónico de Σ^* y M_i la *i*-ésima MT.

L_D : codificaciones de MT que rechazan su propia codificación

$$L_D = \{ w_i \in \Sigma^* / w_i \notin L(M_i) \}$$

Teorema: $L_D \notin \text{RE}$

Dem.: Por reducción al absurdo, se asume que $L_D \in \text{RE} \Rightarrow$ existirá una MT M que lo acepta y cuya codificación está en alguna posición del orden canónico, es decir que existe algún natural k para el cual $M = M_k$ y $L_D = L(M_k)$.

Considerando w_k el k -ésimo string de Σ^* , hay dos posibilidades, o bien $w_k \in L_D$ o bien $w_k \notin L_D$

$$\begin{aligned} a) \quad & w_k \in L_D \Rightarrow w_k \notin L(M_k) \Rightarrow \\ & \Rightarrow w_k \notin L_D \\ & \qquad \text{(contradicción)} \end{aligned}$$

(por def. L_D)
(porque $L(M_k) = L_D$)

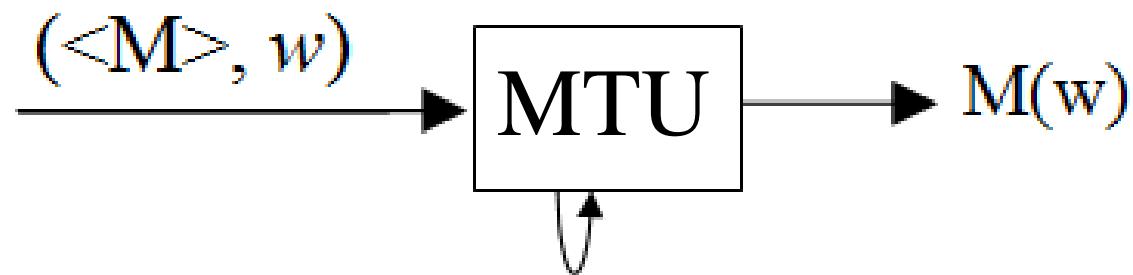
$$\begin{aligned} b) \quad & w_k \notin L_D \Rightarrow w_k \in L(M_k) \Rightarrow \\ & \Rightarrow w_k \in L_D \\ & \qquad \text{(contradicción)} \end{aligned}$$

(por def. L_D)
(porque $L(M_k) = L_D$)

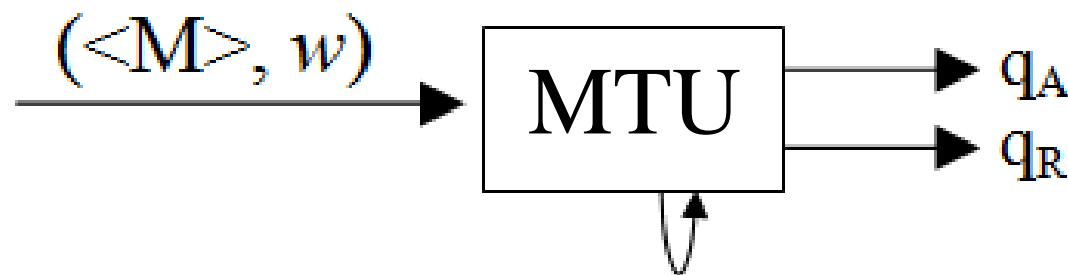
En ambos casos se llega a una contradicción y por ende no puede existir una M_k que reconozca al lenguaje L_D , por lo tanto L_D no puede ser recursivamente enumerable.

Por lo tanto $L_D \notin \text{RE}$.

Máquina de Turing Universal (MTU): es una MT que recibe como entrada la codificación de otra MT M y una entrada w . La MTU ejecuta la MT M sobre la entrada w .



Nota: también se tiene una versión reconocedora de lenguajes



Esta máquina responde a cuestiones relativas a otras MT

Claramente la MTU puede construirse puesto que el proceso de mirar el estado y el símbolo corriente, buscar la quíntupla de δ que se va a aplicar y realizar lo que indica o detenerse es un procedimiento efectivo.

Se puede asumir que la entrada w se separa del código de la MT con **000**, y M se codifica de la manera que ya se ha visto.

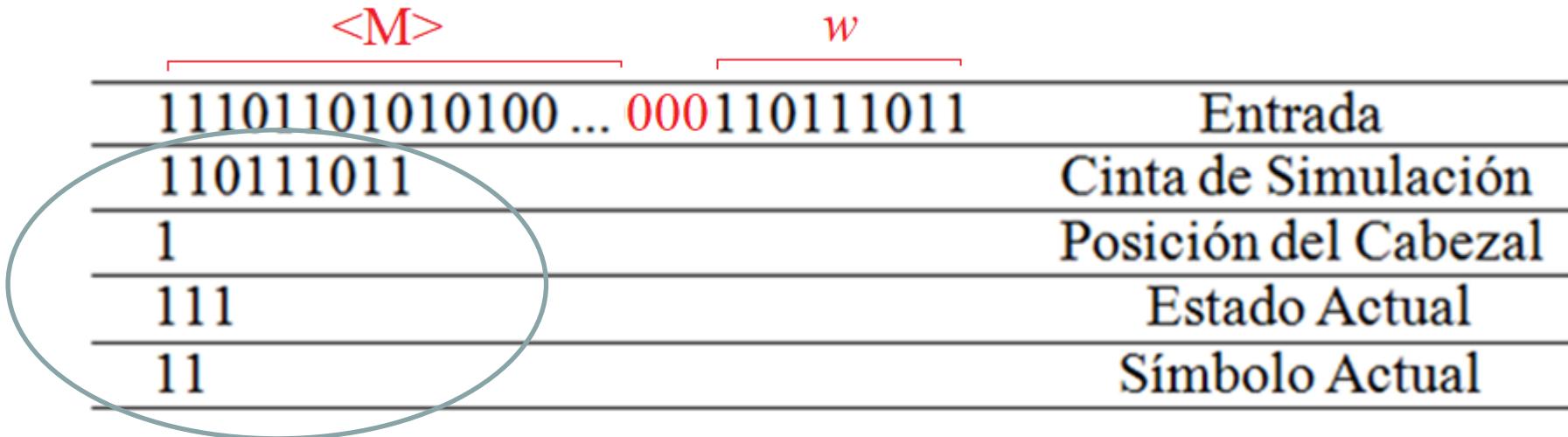
Una idea de cómo construir una MTU

Se copia w en la cinta 2 sobre la que se realiza la simulación

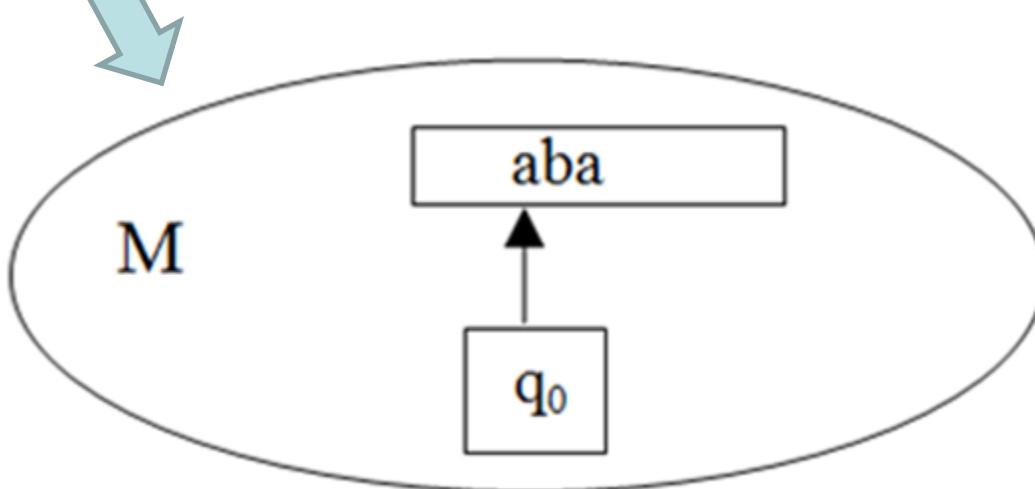
Es necesario identificar:

- Posición del cabezal (se puede usar una tercera cinta)
- Estado actual (se puede usar una cuarta cinta)
- Símbolo actualmente leído (se puede usar una quinta cinta)

MUT de 5 cintas



Representa esta situación

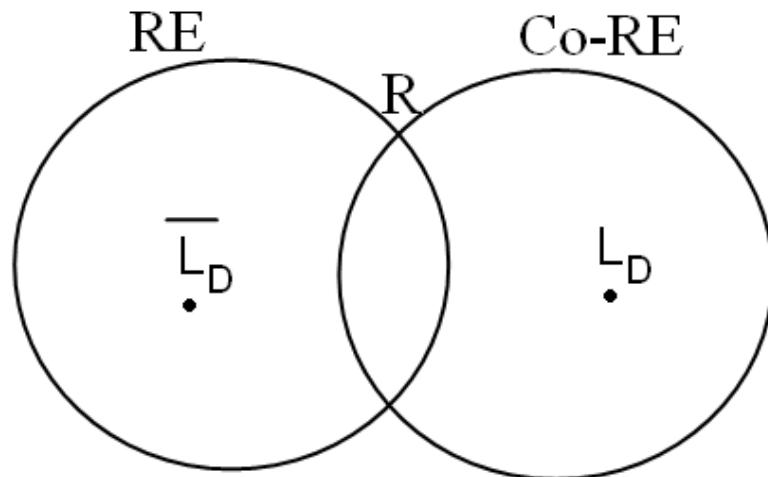


Nota1: puede probarse que L_D pertenece a Co-RE, pues fácilmente se verifica que:

$$\overline{L_D} \in \text{RE} \quad (\quad \overline{L_D} = \{ w_i \in \Sigma^* / w_i \in L(M_i) \} \quad)$$

Para ello se construye una MT que utilizando una MTU acepta $\overline{L_D}$ ejecutando el código de M_i sobre w_i aceptando si M_i acepta w_i . Recuérdese que el código $\langle M_i \rangle$ de la MT M_i se obtiene fácilmente pues $\langle M_i \rangle = w_i$

Nota 2: Además $\overline{L_D}$ no puede estar en R, ya que esto implicaría que L_D también esté en R (y ya sabemos que L_D no pertenece a RE). Por lo tanto $\overline{L_D} \in \text{RE} - \text{R}$.



Def.: se define L_u , el lenguaje universal, como:

$$L_u = \{(\langle M \rangle, w) / M \text{ acepta } w\}$$

Pregunta: ¿ $L_u \in \text{RE}$?

Rta.: claramente sí. Se puede construir M_u de la siguiente manera:

- 1) Si $(\langle M \rangle, w)$ no es un par válido parar en q_R
- 2) En caso contrario separar $\langle M \rangle$ de w
- 3) Si $\langle M \rangle$ es un código inválido parar en q_R
- 4) Simular M sobre w . Si M para en $q_A \Rightarrow M_u$ para en q_A . Si M para en $q_R \Rightarrow M_u$ para en q_R . Si M loopea $\Rightarrow M_u$ también loopea.

Claramente $L_u = L(M_u)$, por lo tanto $L_u \in \text{RE}$

¿ $L_u \in \text{R}$?

Se verá que no es cierto probando que $\overline{L_u} \notin \text{RE}$

$L_u = \{(\langle M \rangle, w) / M \text{ acepta } w\}$ entonces :

$(\langle M \rangle, w) \in \overline{L_u}$ si y sólo si $(\langle M \rangle, w)$ no es un par válido o M rechaza w

Teorema: $\overline{L_u} \notin \text{RE}$

Dem.: se demuestra que si $\overline{L_u} \in \text{RE} \Rightarrow L_D \in \text{RE}$, lo cual es absurdo pues ya se vio que $L_D \notin \text{RE}$.

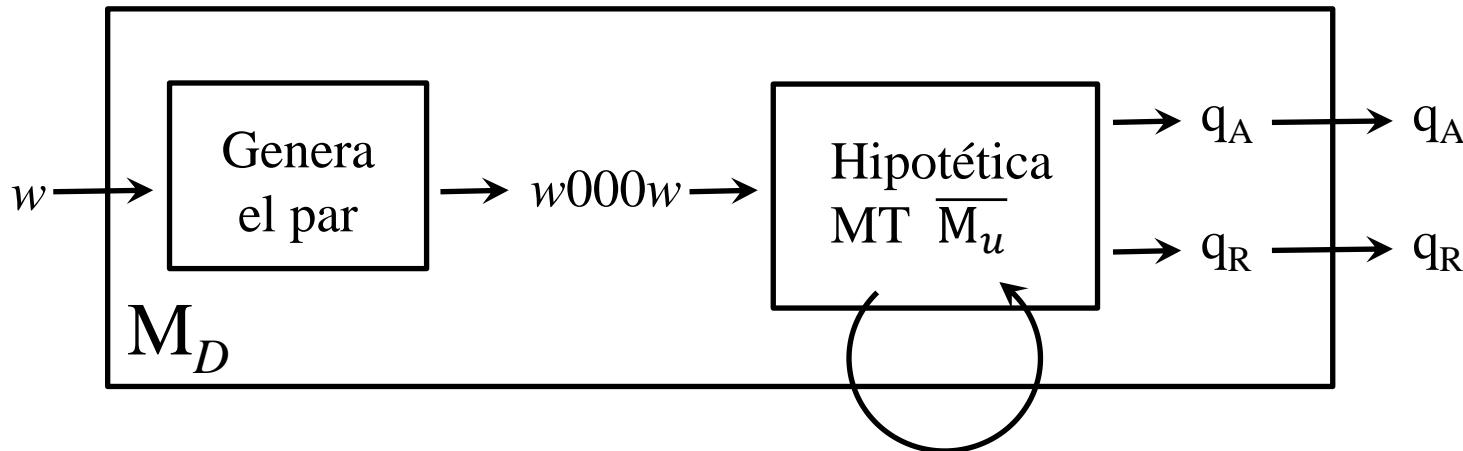
Si $\overline{L_u} \in \text{RE} \Rightarrow \exists \text{ MT } \overline{M_u} \text{ que acepta } \overline{L_u}$, es decir que dada una entrada $(\langle M \rangle, w)$ puede identificar que M rechaza w

Considerando que existe tal $\overline{M_u}$ se construye la MT M_D que acepta L_D de la siguiente manera:

- 1) M_D a partir de la entrada w genera el par $w000w$ (recordar que $\langle M_i \rangle = w_i$)
- 2) M_D simula $\overline{M_u}$ sobre $w000w$. M_D acepta w si $\overline{M_u}$ acepta $w000w$

$L_u = \{(\langle M \rangle, w) / M \text{ acepta } w\}$ entonces :

$(\langle M \rangle, w) \in \overline{L_u}$ si y sólo si $(\langle M \rangle, w)$ no es un par válido o M rechaza w



Si w es w_i en nuestra numeración se tiene que:

$$M_D \text{ acepta } w_i \Leftrightarrow \overline{M_u} \text{ acepta } \langle M_i \rangle 000w_i \Leftrightarrow M_i \text{ rechaza } w_i \Leftrightarrow w_i \in L_D$$

De esta forma $L(M_D) = L_D$ lo que significa que $L_D \in \text{RE}$

Por lo tanto demostramos que si $\overline{L_u} \in \text{RE} \Rightarrow L_D \in \text{RE}$,

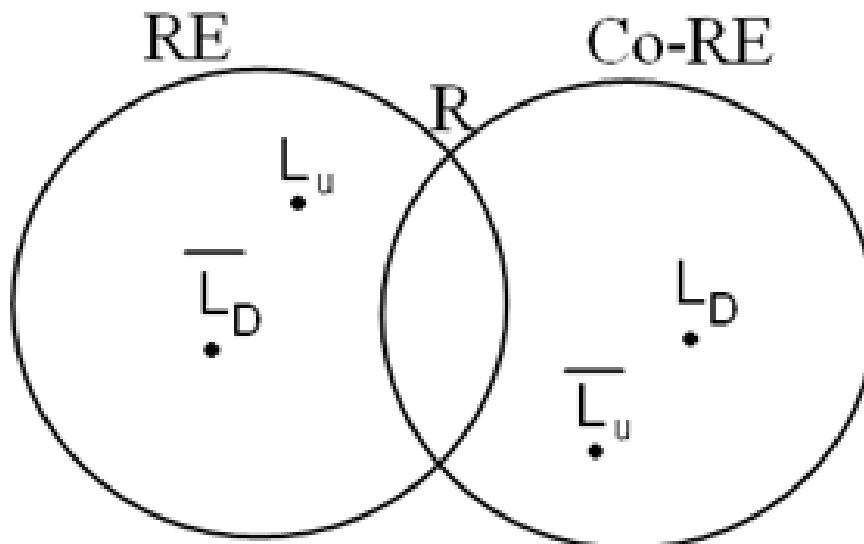
Por contrarrecíproca se tiene que $L_D \notin \text{RE} \Rightarrow \overline{L_u} \notin \text{RE}$

y como ya se conoce que $L_D \notin \text{RE}$ se tiene que $\overline{L_u} \notin \text{RE}$

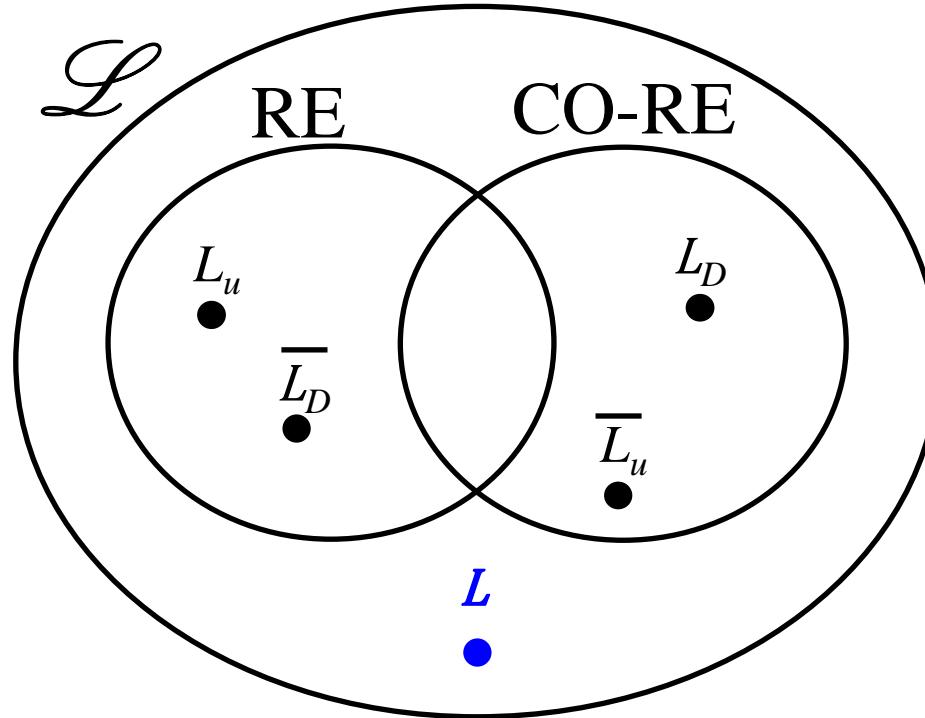
Corolario: $L_u \in (\text{RE} - \text{R})$.

Inmediato pues ya sabemos que L_u está en RE y que L_u no puede estar en R pues ello implicaría que $\overline{L_u}$ también estuviese en R lo que sería un absurdo pues acabamos de demostrar que $\overline{L_u} \notin \text{RE}$

Hasta acá se tiene entonces la siguiente situación:



Lenguaje del conjunto $\mathcal{L} - (\text{RE} \cup \text{CO-RE})$



$$L = \{1w / w \in L_D\} \cup \{0w / w \notin L_D\}$$

$$L \in \mathcal{L} - (\text{RE} \cup \text{CO-RE})$$

Demostración

Claramente $L \in \mathcal{L}$. Falta mostrar que $L \notin (\text{RE} \cup \text{CO-RE})$, es decir que $L \notin \text{RE}$ y que $L \notin \text{Co-RE}$

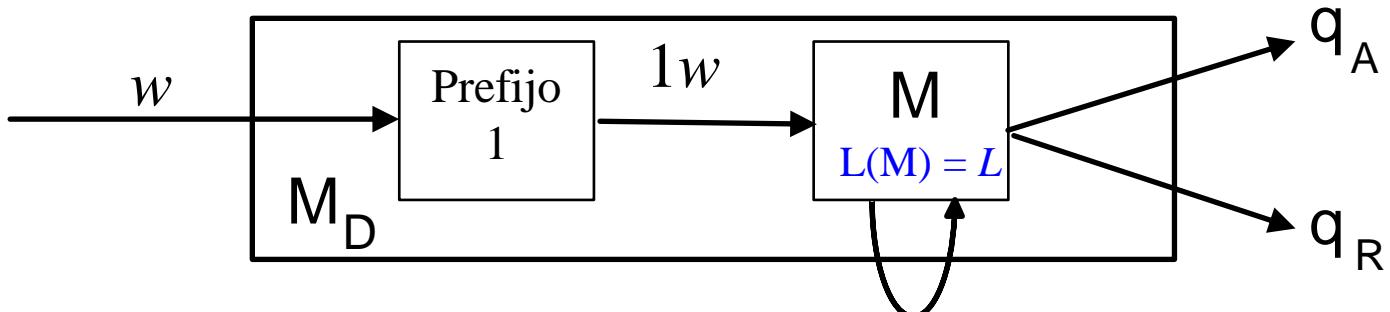
a) Demostrar que $L \notin \text{RE}$

Vamos a demostrar que si $L \in \text{RE}$ entonces $L_D \in \text{RE}$ (absurdo, ya demostramos que $L_D \notin \text{RE}$) por lo tanto la hipótesis $L \in \text{RE}$ no puede ser cierta.

Si $L \in \text{RE}$ entonces existe una MT M que acepta L . Vamos a construir una MT M_D que acepte L_D trabajando de esta manera:

- 1) A partir de una entrada w genera la cadena $1w$ en la cinta
- 2) Simula M sobre $1w$, respondiendo como M con entrada $1w$

$$L = \{1w / w \in L_D\} \cup \{0w / w \notin L_D\}$$



Veamos que $L_D = L(M_D)$

(1) $w \in L_D \Rightarrow 1w \in L$

(por definición de L)

$\Rightarrow M$ acepta $1w$

(porque M reconoce L)

$\Rightarrow M_D$ acepta w

(por construcción de M_D)

$\Rightarrow w \in L(M_D)$

(2) $w \notin L_D \Rightarrow 1w \notin L$

(por definición de L)

$\Rightarrow M$ rechaza $1w$

(porque M reconoce L)

$\Rightarrow M_D$ rechaza w

(por construcción de M_D)

$\Rightarrow w \notin L(M_D)$

por el contra-recíproca $w \in L(M_D) \Rightarrow w \in L_D$

De (1) y (2) $w \in L_D \Leftrightarrow w \in L(M_D)$ Por lo tanto $L_D = L(M_D)$

Así se demostró que si $L \in \text{RE} \Rightarrow L_D \in \text{RE}$

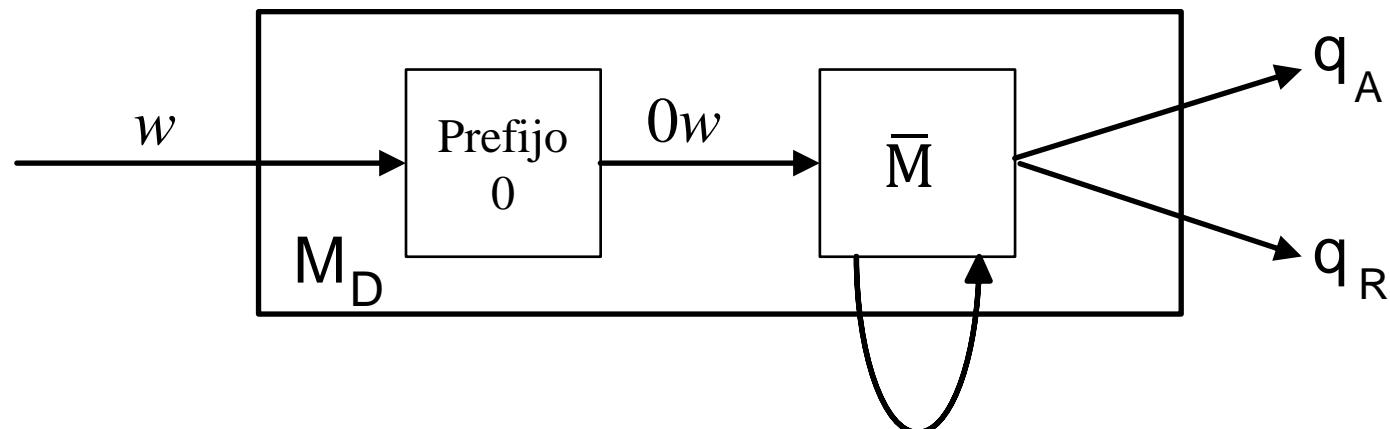
Pero ya sabemos que $L_D \notin \text{RE}$ por lo tanto $L \notin \text{RE}$

b) Demostrar que $L \notin \text{CO-RE}$

Vamos a demostrar que si $L \in \text{CO-RE}$ entonces $L_D \in \text{RE}$ (absurdo, ya demostramos que $L_D \notin \text{RE}$) por lo tanto la hipótesis $L \in \text{CO-RE}$ no puede ser cierta.

Si $L \in \text{CO-RE}$ entonces $\bar{L} \in \text{RE}$, entonces existe una MT \bar{M} que acepta \bar{L} . Vamos a construir una MT M_D que acepte L_D trabajando de esta manera:

- 1) A partir de una entrada w genera la cadena $0w$ en la cinta
- 2) Simula \bar{M} sobre $0w$, respondiendo como \bar{M} con entrada $0w$



$$L = \{1w / w \in L_D\} \cup \{0w / w \notin L_D\}$$

Veamos que $L_D = L(M_D)$

$$(1) w \in L_D \Rightarrow 0w \notin L$$

$\Rightarrow \bar{M}$ acepta $0w$

$\Rightarrow M_D$ acepta w

$\Rightarrow w \in L(M_D)$

(por definición de L)

(porque \bar{M} reconoce \bar{L} y $0w \in \bar{L}$)

(por construcción de M_D)

$$(2) w \notin L_D \Rightarrow 0w \in L$$

$\Rightarrow \bar{M}$ rechaza $0w$

$\Rightarrow M_D$ rechaza w

$\Rightarrow w \notin L(M_D)$

(por definición de L)

(porque \bar{M} reconoce \bar{L} y $0w \notin \bar{L}$)

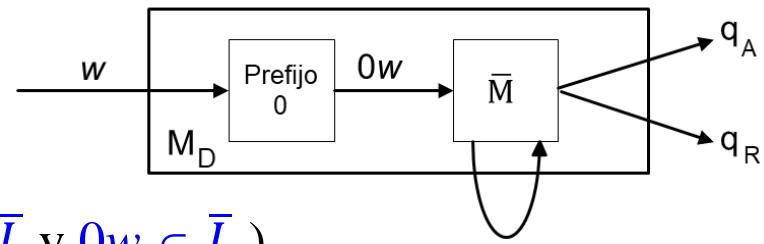
(por construcción de M_D)

por el contra-recíproca $w \in L(M_D) \Rightarrow w \in L_D$

De (1) y (2) $w \in L_D \Leftrightarrow w \in L(M_D)$ Por lo tanto $L_D = L(M_D)$

Así se demostró que si $L \in \text{CO-RE} \Rightarrow L_D \in \text{RE}$

Pero ya sabemos que $L_D \notin \text{RE}$ por lo tanto $L \notin \text{CO-RE}$



Por lo demostrado en a) y b) se tiene que

$L \notin (\text{RE} \cup \text{CO-RE})$

Reducibilidad

- Def.: Sean $L_1, L_2 \subseteq \Sigma^*$ se dirá que L_1 se reduce a L_2 ($L_1 \leq L_2$) si existe una **función total computable** (o **recursiva**) $f: \Sigma^* \rightarrow \Sigma^*$ tal que

$$\forall w \in \Sigma^*, \underbrace{w \in L_1 \Leftrightarrow f(w) \in L_2}_{\text{↓}}$$

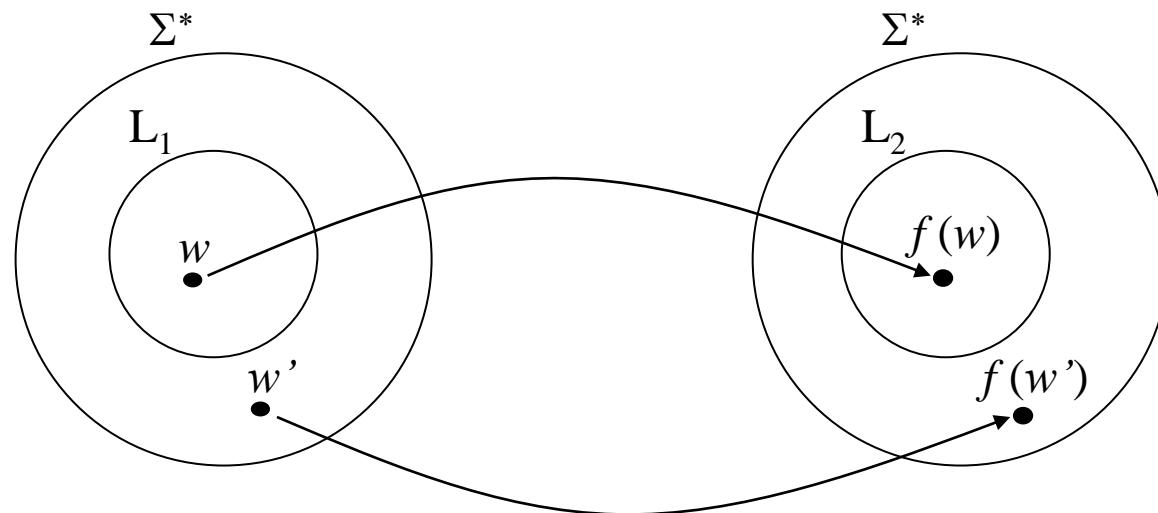
$$w \in L_1 \Rightarrow f(w) \in L_2 \quad \text{AND} \quad \underbrace{f(w) \in L_2 \Rightarrow w \in L_1}_{\text{↓}}$$

$$w \notin L_1 \Rightarrow f(w) \notin L_2$$

Reducibilidad

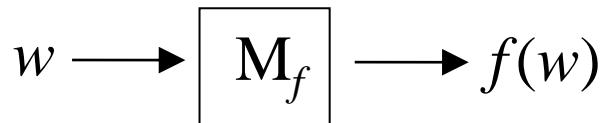
- Def.: Sean $L_1, L_2 \subseteq \Sigma^*$ se dirá que L_1 se reduce a L_2 ($L_1 \leq L_2$) si existe una **función total computable** (o **recursiva**) $f: \Sigma^* \rightarrow \Sigma^*$ tal que

$$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$$



Reducibilidad

- Se dice que f es computable si existe una MT que la computa y que siempre se detiene. Notar la importancia de que f sea completa.



M_f nunca loopea. A veces, usaremos la expresión $M_f(w)$ para referirnos a la función computada por la MT M_f

Nota: la reducibilidad es una transformación de instancias de un problema en instancias de otro problema.

Reducibilidad

Ej.: $L_1 = \{w \in \{a, b\}^* / w \text{ comienza con } a\}$

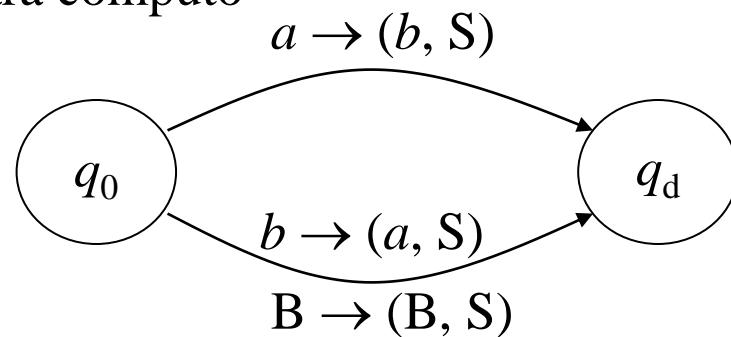
$L_2 = \{w \in \{a, b\}^* / w \text{ comienza con } b\}$

Vamos a demostrar que $(L_1 \alpha L_2)$

$M_f = \langle Q, \Sigma, \Gamma, \delta, q_0, q_d \rangle \quad Q = \{q_0\} \quad \Sigma = \{a, b\} \quad \Gamma = \{B, a, b\}$

M_f es una MT de/para cómputo

δ :



Puede demostrarse fácilmente que:

1. M_f siempre se detiene
2. $\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$

Reducibilidad

Ejercicio 1:

$$L_1 = \{w \in \{0, 1\}^* / \text{cant}_1(w) \text{ es par}\}$$

$$L_2 = \{w \in \{0, 1\}^* / \text{cant}_1(w) \text{ es impar}\}$$

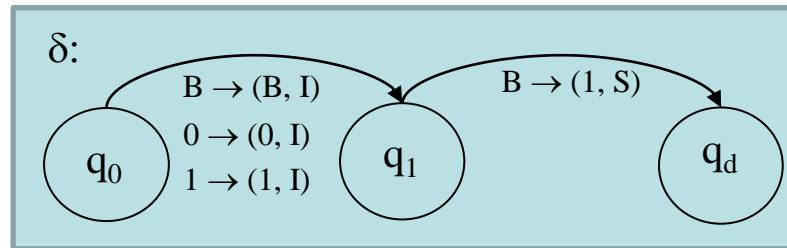
Donde $\text{cant}_1(w)$ es la cantidad de 1 que hay en w

Demostrar que $L_1 \alpha L_2$.

Reducibilidad

Se construye M_f , una MT que computa la función de reducibilidad.

$M_f = \langle \{q_0, q_1\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, q_d \rangle$, M_f : MT de/para cómputo



Hay que demostrar que M_f siempre se detiene y que $w \in L_1 \Leftrightarrow M_f(w) \in L_2$

- 1) M_f siempre se detiene: claramente sí, pues para cualquier input solamente ejecuta dos pasos de computación y se detiene.

$L_1 = \{w \in \{0, 1\}^* / \text{cant}_1(w) \text{ es par}\}$

$L_2 = \{w \in \{0, 1\}^* / \text{cant}_1(w) \text{ es impar}\}$

Reducibilidad

2) ¿ $w \in L_1 \Leftrightarrow M_f(w) \in L_2$?

Claramente $\text{cant}_1(M_f(w)) = \text{cant}_1(w) + 1$, por lo tanto:

- si $\text{cant}_1(w)$ es par entonces $\text{cant}_1(M_f(w))$ es impar y
- si $\text{cant}_1(w)$ es impar entonces $\text{cant}_1(M_f(w))$ es par

Entonces:

a) Si $w \in L_1 \Rightarrow \text{cant}_1(w)$ es par $\Rightarrow \text{cant}_1(M_f(w))$ es impar $\Rightarrow M_f(w) \in L_2$

b) Si $w \notin L_1 \Rightarrow \text{cant}_1(w)$ es impar $\Rightarrow \text{cant}_1(M_f(w))$ es par $\Rightarrow M_f(w) \notin L_2$

De a) y b) se tiene que $w \in L_1 \Leftrightarrow M_f(w) \in L_2$.

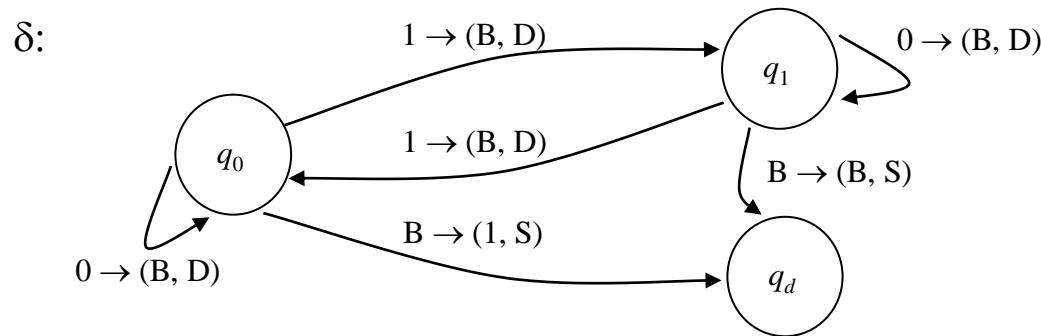
Reducibilidad

Ejercicio 2: Sean $L_1 = \{w \in \{0, 1\}^* / \text{cant}_1(w) \text{ es par}\}$

$$L_2 = \{w \in \{0, 1\}^* / \text{cant}_1(w) = 1\}$$

Demostrar que $L_1 \alpha L_2$.

Se construye M_f tal que $w \in L_1 \Leftrightarrow M_f(w) \in L_2$



Tarea para el lector: Demostrar que:

- 1) M_f se detiene para todo w
- 2) $w \in L_1 \Leftrightarrow M_f(w) \in L_2$.

Reducibilidad

Ejercicio 3: Sea L un lenguaje recursivo ($L \in R$)

$L_1 = \{\langle M \rangle / \langle M \rangle \text{ es un cód. válido de MT, } L(M) = L \text{ y } M \text{ siempre se detiene}\}$

$L_2 = \{\langle M \rangle / \langle M \rangle \text{ es un cód. válido de MT y } L(M) = \bar{L}\}$

Demostrar que $L_1 \alpha L_2$

Hay que encontrar una MT M_f que siempre se detenga para la cual sea cierto que:

$$w \in L_1 \Leftrightarrow M_f(w) \in L_2$$

Se construye M_f que trabaja de la siguiente manera: Si w no es un código válido de MT M_f se detiene sin hacer nada. De lo contrario (w es un código $\langle M \rangle$ de MT válido) recorre todas las quintuplas de w , si en la 3ra posición encuentra q_A lo reemplaza por q_R , si encuentra q_R lo reemplaza por q_A .

Nuevamente hay que demostrar que:

- 1) M_f se detiene
- 2) $w \in L_1 \Leftrightarrow M_f(w) \in L_2$.

$L_1 = \{ \langle M \rangle / \langle M \rangle \text{ es un cód. válido de MT, } L(M) = L \text{ y } M \text{ siempre se detiene} \}$

$L_2 = \{ \langle M \rangle / \langle M \rangle \text{ es un cód. válido de MT y } L(M) = \bar{L} \}$

1) M_f siempre se detiene porque la entrada es finita.

2) $w \in L_1 \Leftrightarrow M_f(w) \in L_2$?

2.a) $w \in L_1 \Rightarrow M_f(w) \in L_2$?

Si $w \in L_1 \Rightarrow w$ es un código válido de una MT M que reconoce L y siempre se detiene \Rightarrow

$\begin{cases} \text{si } x \in L \Rightarrow M \text{ para en } q_A \Rightarrow M_f(w) \text{ codifica una MT que para en } q_R \text{ con input } x \\ \text{si } x \notin L \Rightarrow M \text{ para en } q_R \Rightarrow M_f(w) \text{ codifica una MT que para en } q_A \text{ con input } x \end{cases}$

observar que M nunca rechaza loopando (ver def. de L_1)

por lo tanto $M_f(w)$ acepta $\bar{L} \Rightarrow M_f(w) \in L_2$

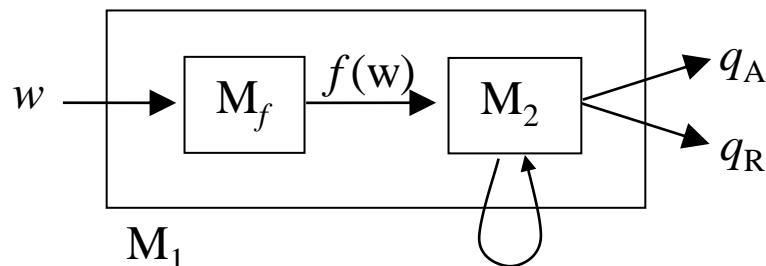
2.b) Se demuestra similarmente que $w \notin L_1 \Rightarrow M_f(w) \notin L_2$. Además, considérese que si w es un código inválido no pertenece a L_1 y $M_f(w)$ tampoco pertenece a L_2 .

Por lo tanto $w \in L_1 \Leftrightarrow M_f(w) \in L_2$.

Reducibilidad

Nota: si $L_1 \alpha L_2$ entonces se puede construir una MT que acepte L_1 a partir de la MT que acepta L_2 (si existe).

Debe quedar claro que “en cierto sentido” L_1 no puede ser más difícil computacionalmente que L_2 porque se puede utilizar L_2 para resolver L_1 .



Intuitiva y coloquialmente, la reducción establece una relación de “ \leq ” grado de dificultad computacional entre lenguajes/problemas...

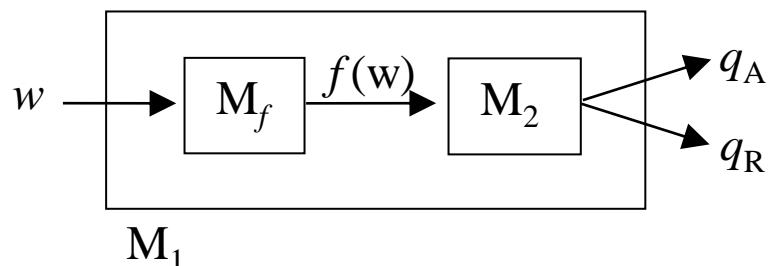
Reducibilidad

Teorema 1: $L_1, L_2 \subseteq \Sigma^*$ y existe la reducción $L_1 \alpha L_2$, entonces:

$$\text{si } L_2 \in R \Rightarrow L_1 \in R$$

Dem.: $L_2 \in R \Rightarrow$ existe una máquina de Turing M_2 tal que $L_2 = L(M_2)$ y M_2 siempre se detiene. Se construye M_1 que hace:

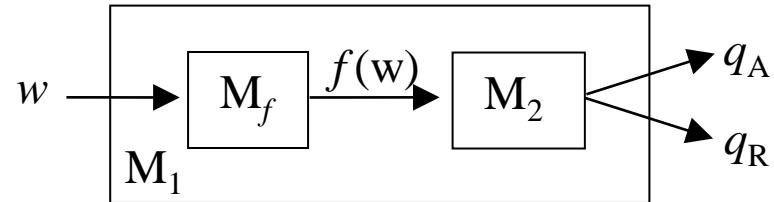
- 1) Simula M_f sobre w y obtiene $f(w)$
- 2) Simula M_2 sobre $f(w)$ y acepta si M_2 acepta



Reducibilidad

Se debe demostrar que:

- a) $L_1 = L(M_1)$
- b) M_1 siempre de detiene.



a.1) $w \in L_1 \Rightarrow f(w) \in L_2$ (porque $L_1 \alpha L_2$)

$\Rightarrow M_2$ para en q_A con input $f(w)$

$\Rightarrow M_1$ para en q_A con input w

$\Rightarrow w \in L(M_1)$

a.2) $w \notin L_1 \Rightarrow f(w) \notin L_2$ (porque $L_1 \alpha L_2$)

$\Rightarrow M_2$ para en q_R con input $f(w)$

$\Rightarrow M_1$ para en q_R con input w

$\Rightarrow w \notin L(M_1)$

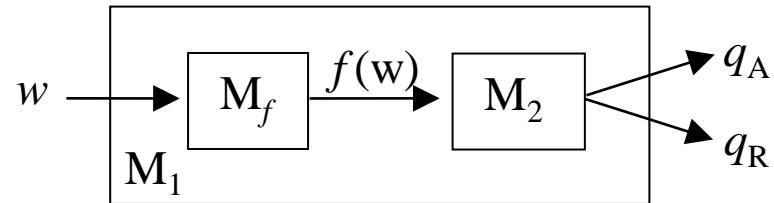
Por a.1) y a.2) se tiene que $L_1 = L(M_1)$



Reducibilidad

Se debe demostrar que:

- a) $L_1 = L(M_1)$ ✓
- b) M_1 siempre detiene.

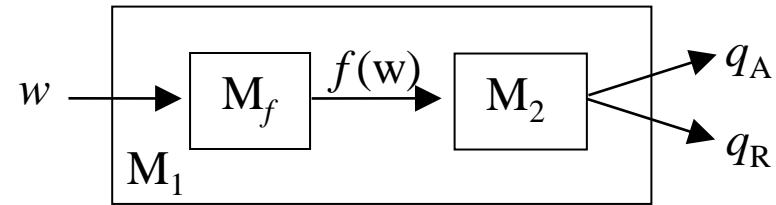


Claramente M_1 siempre detiene pues M_f y M_2 se detienen siempre por hipótesis. ✓

Reducibilidad

Se debe demostrar que:

- a) $L_1 = L(M_1)$ ✓
- b) M_1 siempre detiene. ✓



Por lo tanto, de a) y b) se tiene que $L_1 \in R$.

Reducibilidad

Teorema 2: $L_1, L_2 \subseteq \Sigma^*$ y existe la reducción $L_1 \alpha L_2$, entonces:

$$L_2 \in \text{RE} \Rightarrow L_1 \in \text{RE}$$

Dem.: Se demuestra de manera similar a la demostración del teorema anterior

Corolario: Sean $L_1, L_2 \subseteq \Sigma^*$ y la reducción $L_1 \alpha L_2$, por las respectivas contrarecíprocas del teorema 1 y del teorema 2, se cumple que:

$$L_1 \notin \text{R} \Rightarrow L_2 \notin \text{R}$$

$$L_1 \notin \text{RE} \Rightarrow L_2 \notin \text{RE}$$

Halting Problem

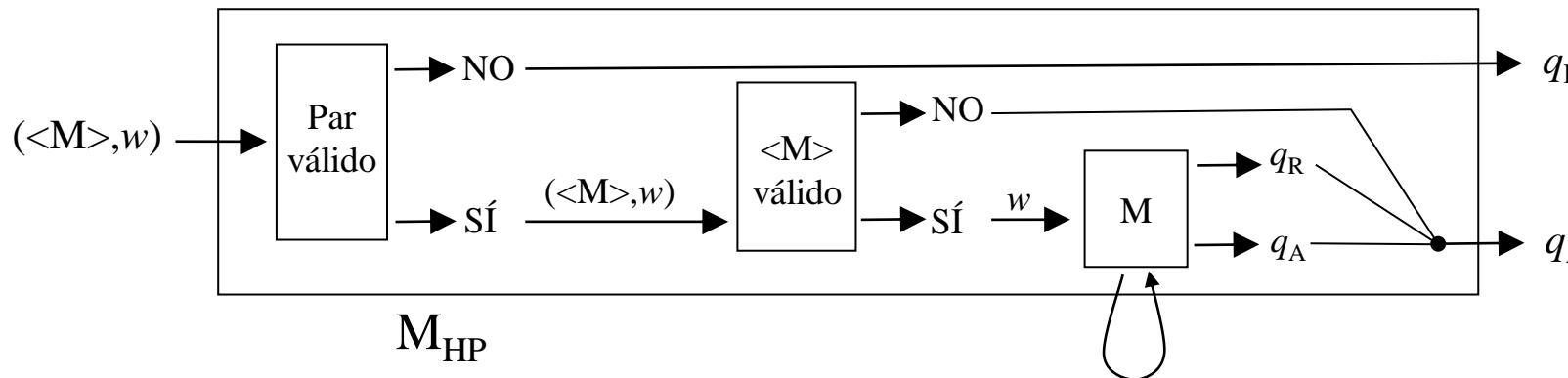
El Lenguaje *Halting Problem* se define como:

$$\text{HP} = \{(\langle M \rangle, w) \text{ tal que } M \text{ se detiene con input } w\}$$

Ejercicio: demostrar que $\text{HP} \in (\text{RE-R})$.

I) $\text{HP} \in \text{RE}$.

Se construye una MT M_{HP} tal que $L(M_{\text{HP}}) = \text{HP}$. M_{HP} chequea sintácticamente la entrada $(\langle M \rangle, w)$, si es un par inválido para en q_R , si es un par válido pero $\langle M \rangle$ es un código de MT inválido para en q_A , si ambos son válidos simula M sobre w , si M para (en q_A o en q_R) M_{HP} para en q_A , si M no para, M_{HP} no para.



Halting Problem

II) $\text{HP} \notin R$.

Se puede probar que existe la reducción $L_u \alpha \text{HP}$, y como $L_u \notin R$ será cierto $\text{HP} \notin R$.

Dem. Sea la siguiente MT M_f que computa la función f de reducibilidad

$$M_f((\langle M \rangle, w)) = (\langle M' \rangle, w)$$

M_f trabaja de la siguiente manera:

Si $(\langle M \rangle, w)$ no es un par válido o $\langle M \rangle$ no es un código válido de MT borra la cinta (deja λ como salida), de lo contrario busca en las quíntuplas de $\langle M \rangle$ el estado q_R y lo reemplaza por un nuevo estado q . Luego agrega las quíntuplas (q, x, q, x, S) por cada símbolo x del alfabeto de la cinta de M . Así la máquina M' construida por M_f entra en loop cuando M para en q_R .

Halting Problem

Hay que demostrar que M_f es una máquina que computa la función de reducibilidad, es decir se debe probar que:

- 1) f es computable ?
- 2) $(\langle M \rangle, w) \in L_u \Leftrightarrow (\langle M' \rangle, w) \in HP$?

Claramente f es computable pues M_f siempre se detiene pues la entrada es finita y luego de recorrerla agrega un número finito de quintuplas y se detiene ✓

Halting Problem

Hay que demostrar que M_f es una máquina que computa la función de reducibilidad, es decir se debe probar que:

1) f es computable ✓

2) $(\langle M \rangle, w) \in L_u \Leftrightarrow (\langle M' \rangle, w) \in HP$?

2.a) $(\langle M \rangle, w) \in L_u \Rightarrow (\langle M' \rangle, w) \in HP$?

2.b) $(\langle M \rangle, w) \notin L_u \Rightarrow (\langle M' \rangle, w) \notin HP$?

$(\langle M \rangle, w) \in L_u \Rightarrow M$ acepta w
 $\Rightarrow M$ para en q_A
 $\Rightarrow M'$ para en q_A (por construcción)
 $\Rightarrow M'$ se detiene con input w
 $\Rightarrow (\langle M' \rangle, w) \in HP$ ✓

Halting Problem

Hay que demostrar que M_f es una máquina que computa la función de reducibilidad, es decir se debe probar que:

1) f es computable ✓

2) $(\langle M \rangle, w) \in L_u \Leftrightarrow (\langle M' \rangle, w) \in HP$?

2.a) $(\langle M \rangle, w) \in L_u \Rightarrow (\langle M' \rangle, w) \in HP$? ✓

2.b) $(\langle M \rangle, w) \notin L_u \Rightarrow (\langle M' \rangle, w) \notin HP$?

i) Si $(\langle M \rangle, w)$ no es un par válido o $\langle M \rangle$ no es un código válido de máquina de Turing $\Rightarrow (\langle M' \rangle, w) = \lambda \Rightarrow (\langle M' \rangle, w) \notin HP$ ✓

ii) M rechaza $w \Rightarrow M$ loopea o para en q_R con input $w \Rightarrow M'$ loopea con input $w \Rightarrow (\langle M' \rangle, w) \notin HP$ ✓

Halting Problem

Hay que demostrar que M_f es una máquina que computa la función de reducibilidad, es decir se debe probar que:

1) f es computable ✓

2) $(\langle M \rangle, w) \in L_u \Leftrightarrow (\langle M' \rangle, w) \in HP ?$

2.a) $(\langle M \rangle, w) \in L_u \Rightarrow (\langle M' \rangle, w) \in HP ?$ ✓

2.b) $(\langle M \rangle, w) \notin L_u \Rightarrow (\langle M' \rangle, w) \notin HP ?$ ✓

Halting Problem

Hay que demostrar que M_f es una máquina que computa la función de reducibilidad, es decir se debe probar que:

- 1) f es computable 
- 2) $(\langle M \rangle, w) \in L_u \Leftrightarrow (\langle M' \rangle, w) \in HP$ 

De 1) y 2) se tiene que M_f computa la función de reducibilidad buscada y por lo tanto queda demostrado que $L_u \alpha HP$.

Como $L_u \notin R \Rightarrow HP \notin R$

De I) y II) $HP \in (RE - R)$

Lenguaje L_{Σ^*}

Ejercicio: Probar que $L_{\Sigma^*} = \{\langle M \rangle / L(M) = \Sigma^*\}$ no es recursivo.

Mostraremos que existe una reducción $L_u \alpha L_{\Sigma^*}$

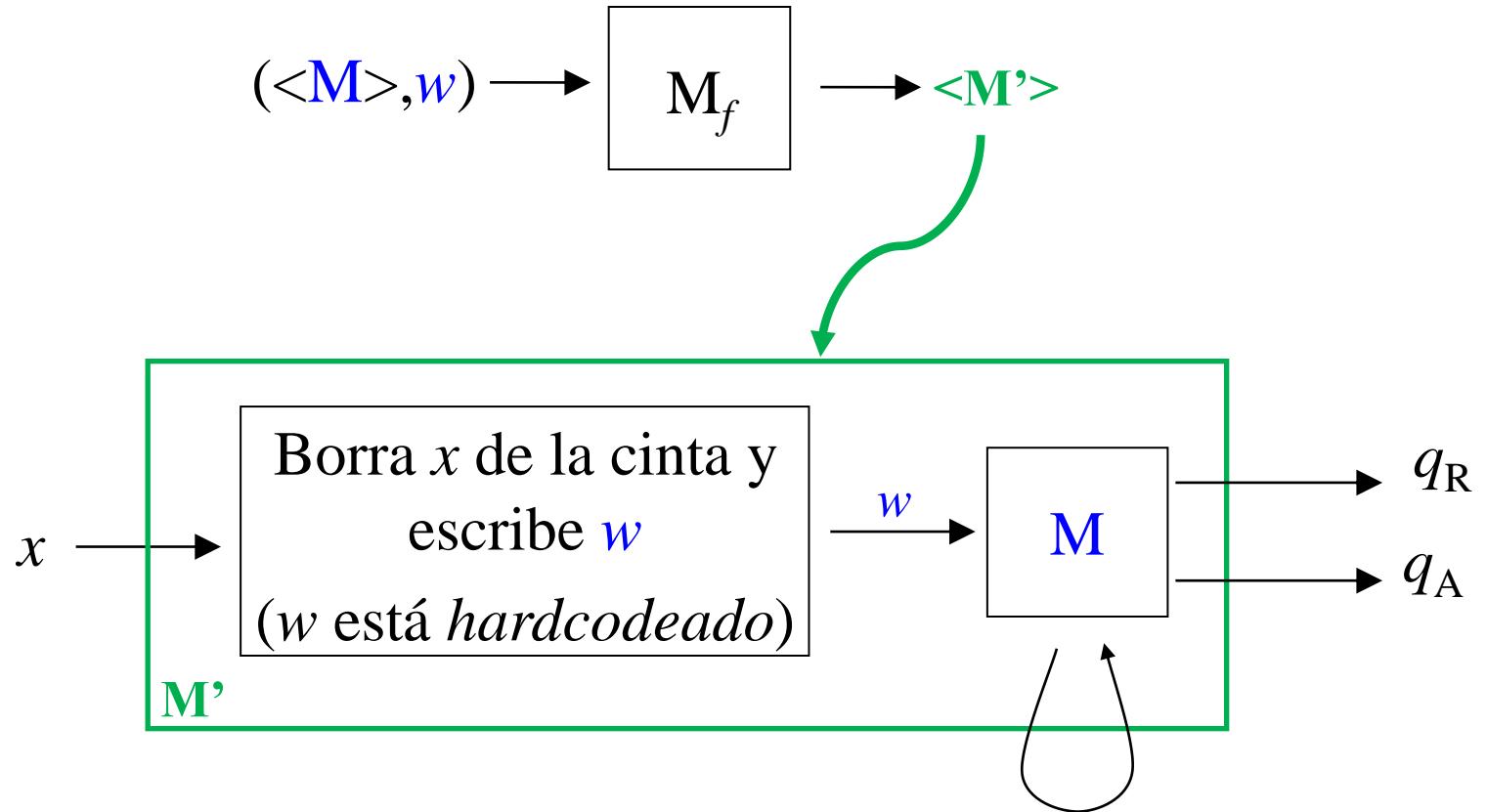
Hay que encontrar una función total computable tal que

$$(\langle M \rangle, w) \in L_u \Leftrightarrow f(\langle M \rangle, w) \in L_{\Sigma^*}$$

Sea M_f la máquina de Turing que computa la función $f(\langle M \rangle, w) = \langle M' \rangle$ y trabaja de la siguiente manera:

Si $(\langle M \rangle, w)$ no es un par válido o $\langle M \rangle$ no es un código de MT válido, M_f borra la cinta (deja λ como salida), de lo contrario M_f construye $\langle M' \rangle$ escribiendo las quíntuplas necesarias para que M' borre la entrada y escriba w en la cinta, posicione el cabezal y simule M sobre w . Así M' para en q_A para cualquier input $\Leftrightarrow M$ acepta w

Lenguaje Σ^*



Para cualquier x de Σ^* la máquina M' ejecuta M sobre w

Lenguaje L_{Σ^*}

1) $f((\langle M \rangle, w))$ es computable? Claramente sí, pues M_f se detiene luego de realizar una cantidad finita de acciones.

2) $\langle (\langle M \rangle, w) \in L_u \Leftrightarrow \langle M' \rangle \in L_{\Sigma^*} \rangle$

a) Sea $\langle (\langle M \rangle, w) \in L_u \Rightarrow M$ para en q_A con entrada $w \Rightarrow M'$ para en q_A con cualquier entrada $\Rightarrow \langle M' \rangle \in L_{\Sigma^*}$

b) Sea $\langle (\langle M \rangle, w) \notin L_u \Rightarrow$

i. Si $\langle (\langle M \rangle, w)$ no es un par válido o $\langle M \rangle$ no es un código de MT válido $\Rightarrow \langle M' \rangle = \lambda \Rightarrow \langle M' \rangle \notin L_{\Sigma^*}$

ii. M rechaza $w \Rightarrow M'$ rechaza toda entrada $\Rightarrow \langle M' \rangle \notin L_{\Sigma^*}$

De a) y b) se tiene que $\langle (\langle M \rangle, w) \in L_u \Leftrightarrow \langle M' \rangle \in L_{\Sigma^*} \rangle$

De 1) y 2) se tiene que $L_u \alpha L_{\Sigma^*}$

Por lo tanto $L_{\Sigma^*} \notin a R$ (porque $L_u \notin a R$)

Lenguaje L_{Σ^*}

Nota: Puede demostrarse también que existe una reducción $\bar{L}_u \alpha L_{\Sigma^*}$ y como $\bar{L}_u \notin \text{RE}$ se tiene que $L_{\Sigma^*} \notin \text{a RE}$

Para practicar. Demostrar que existe una reducción de $\bar{L}_u \alpha L_\emptyset$ con $L_\emptyset = \{\langle M \rangle / L(M) = \emptyset\}$

Lenguaje L_{EQ}

Teorema. $L_{EQ} = \{(\langle M_1 \rangle, \langle M_2 \rangle) / L(M_1) = L(M_2)\} \notin RE$

Prueba: $L_{\Sigma^*} \alpha L_{EQ}$

La función f de reducibilidad que computa M_f es

$f(\langle M \rangle) = (\langle M \rangle, \langle M_{\Sigma^*} \rangle)$ Siendo $\langle M_{\Sigma^*} \rangle$ el código de una máquina de Turing que acepta L_{Σ^*}

Por ejemplo si $\Sigma = \{a, b\}$, la δ de transición de M_{Σ^*} puede ser la siguiente:

$$\delta(q_0, a) = (q_A, a, S); \delta(q_0, b) = (q_A, b, S); \delta(q_0, B) = (q_A, B, S)$$

Claramente M_f se detiene, por lo tanto f es computable

Además:

$$\langle M \rangle \in L_{\Sigma^*} \Leftrightarrow L(M) = \Sigma^* \Leftrightarrow L(M) = L(M_{\Sigma^*}) \Leftrightarrow (\langle M \rangle, \langle M_{\Sigma^*} \rangle) \in L_{EQ}$$

Lenguaje L_{EQ}

Teorema. $L_{EQ} = \{(\langle M_1 \rangle, \langle M_2 \rangle) / L(M_1) = L(M_2)\} \notin RE$

Prueba: $L_{\Sigma^*} \alpha L_{EQ}$

La función f de reducibilidad que computa M_f es

$f(\langle M \rangle) = (\langle M \rangle, \langle M_{\Sigma^*} \rangle)$ Siendo $\langle M_{\Sigma^*} \rangle$ el código de una máquina de Turing que acepta Σ^*

Por ejemplo si $\Sigma = \{a, b\}$, la δ de transición de M_{Σ^*} puede ser la siguiente:

$$\delta(q_0, a) = (q_A, a, S); \delta(q_0, b) = (q_A, b, S); \delta(q_0, B) = (q_A, B, S)$$

Claramente M_f se detiene, por lo tanto f es computable

Además:

$$\langle M \rangle \in L_{\Sigma^*} \Leftrightarrow L(M) = \Sigma^* \Leftrightarrow L(M) = L(M_{\Sigma^*}) \Leftrightarrow (\langle M \rangle, \langle M_{\Sigma^*} \rangle) \in L_{EQ}$$

Por lo tanto, f es una función de reducibilidad y queda probado que $L_{\Sigma^*} \alpha L_{EQ}$

Entonces $L_{EQ} \notin RE$

Notación Asintótica

Contexto

- Matemáticamente: “crecimiento” de func.
 - Con su correspondiente def. formal
- Cantidades de operaciones
 - Para los problemas computables
- Función de tamaño de la entrada
 - “Monótonamente” crecientes
- Asintótica
 - Variable (entrada) “crece” arbitrariamente

Definición

Una función $t(n)$ está **en el orden de $f(n)$** si existe una constante real positiva c y un umbral u_0 tal que $t(n) \leq c.f(n) \forall n > u_0$

Notación: Conjuntos

\mathbb{N} (incluye el 0)

$\mathbb{R}^{\geq 0}$: $x \in \mathbb{R}, x \geq 0$

\mathbb{R}^+ : $x \in \mathbb{R}, x > 0$

Definición

Notación: Cuantificadores

\exists

\exists^∞

\forall^∞

\forall

Definición

Notación: Cuantificadores

\exists

\exists^∞

\forall^∞

\forall

$$O(f(n)) = \{t: N \rightarrow R^{\geq 0} / (\exists c \in R^+) (\forall^\infty n \in N) [t(n) \leq c f(n)]\}$$

Definición

Notación: Cuantificadores

\exists

\exists^∞

\forall^∞

\forall

$$O(f(n)) = \{t: N \rightarrow R^{\geq 0} / (\exists c \in R^+) (\forall^\infty n \in N) [t(n) \leq c f(n)]\}$$

¿Podría ser $t:N \rightarrow R^+$?

Definición

$$O(f(n)) = \{t: N \rightarrow R^{\geq 0} / (\exists c \in R^+) (\forall^\infty n \in N) [t(n) \leq c f(n)]\}$$

Otra definición:

$$O(f(n)) = \{t: N \rightarrow R^+ / \exists c \in R^+, n_0 \in N \text{ tq } t(n) \leq c f(n), n \geq n_0\}$$

Definición

$$O(f(n)) = \{t: N \rightarrow R^{\geq 0} / (\exists c \in R^+) (\forall^\infty n \in N) [t(n) \leq c f(n)]\}$$

Otra definición:

$$O(f(n)) = \{t: N \rightarrow R^+ / \exists c \in R^+, n_0 \in N \text{ tq } t(n) \leq c f(n), n \geq n_0\}$$

Es más "precisa":

- R^+ (cantidades de operaciones)
- “Reemplazando” \forall^∞ por el “umbral”, es decir con las excepciones (a la cota del \leq) para los valores de n anteriores a n_0

Terminología

- Conjuntos ==> \in
 - "está en el orden de" (conjuntos)
 - "es"
 - $n^2 = O(n^3)$ (one-way equality, podría relacionarse con el "es")
 - $f(n) = 2n^2 + O(n)$
 - $2n^2 + O(n) = O(n^2)$
- (cont.)...

Terminología

(cont.)...

- Se acepta que $t(n) \in O(f(n))$ si $\exists c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$ tq $0 \leq t(n) \leq c f(n)$; $n \geq n_0$

Sin poner restricciones para $n < n_0$, donde $t(n)$ y $f(n)$ podrían dar valores negativos o no estar definidas, por ejemplo:

$O(n / \log n)$, $n = 0$ y $n = 1$ no están definidos

$t(n) = n^3 - 3n^2 - n - 8 \in O(n^3)$, aunque $n \leq 3 \implies t(n) < 0$

Definiciones

$$O(f(n)) = \{t: N \rightarrow R^+ / \exists c \in R^+, n_0 \in N \text{ tq } t(n) \leq c f(n), n \geq n_0\}$$

$$\Omega(f(n)) = \{t: N \rightarrow R^+ / \exists c \in R^+, n_0 \in N \text{ tq } t(n) \geq c f(n), n \geq n_0\}$$

Se suele mencionar que tanto $O(f(n))$ como $\Omega(f(n))$ son “ambiguas” o “excesivas” en cuanto a que se puede usar cualquier función como cota. Más precisión: $\Theta(f(n))$

Definiciones

$$O(f(n)) = \{t: N \rightarrow R^+ / \exists c \in R^+, n_0 \in N \text{ tq } t(n) \leq c f(n), n \geq n_0\}$$

$$\Omega(f(n)) = \{t: N \rightarrow R^+ / \exists c \in R^+, n_0 \in N \text{ tq } t(n) \geq c f(n), n \geq n_0\}$$

Se suele mencionar que tanto $O(f(n))$ como $\Omega(f(n))$ son “ambiguas” o “excesivas” en cuanto a que se puede usar cualquier función como cota. Más precisión: $\Theta(f(n))$

$$\Theta(f(n)) = \{t: N \rightarrow R^+ / \exists c_1, c_2 \in R^+, n_0 \in N \text{ tq } c_1 f(n) \leq t(n) \leq c_2 f(n), n \geq n_0\}$$

Propiedades

- $g(n) \in \Omega(f(n))$ sii $f(n) \in O(g(n))$
(Regla de Dualidad)
- $g(n) \in \Theta(f(n))$ sii
 $g(n) \in O(f(n))$ y $g(n) \in \Omega(f(n))$
- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

Propiedades

- $g(n) \in \Omega(f(n))$ sii $f(n) \in O(g(n))$
(Regla de Dualidad)
- $g(n) \in \Theta(f(n))$ sii
 $g(n) \in O(f(n))$ y $g(n) \in \Omega(f(n))$
- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

¿Cómo se demuestran? (definiciones de referencia)

Propiedades

- *Reflexividad y Transitividad* de la pertenencia a $O()$, $\Omega()$ y $\Theta()$
- $f(n) \in O(f(n))$
- Si $f(n) \in O(g(n))$ y $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$

¿Por qué “serían” ciertas?

¿Cómo demostrarlas?

Regla del Umbral

- El umbral n_0 de las definiciones de $O()$, $\Omega()$ y $\Theta()$ puede resultar útil pero nunca es necesario cuando se consideran funciones estrictamente positivas, es decir $t, f: \mathbb{N} \rightarrow \mathbb{R}^+$
- Regla del Umbral: $f, t: \mathbb{N} \rightarrow \mathbb{R}^+$,
 $t(n) \in O(f(n)) \iff$ existe $c \in \mathbb{R}^+$ tal que
 $t(n) \leq c f(n)$ para todo natural n

Regla del Umbral

- El umbral n_0 de las definiciones de $O()$, $\Omega()$ y $\Theta()$ puede resultar útil pero nunca es necesario cuando se consideran funciones estrictamente positivas, es decir $t, f: \mathbb{N} \rightarrow \mathbb{R}^+$
- Regla del Umbral: $f, t: \mathbb{N} \rightarrow \mathbb{R}^+$,
 $t(n) \in O(f(n)) \iff \exists c \in \mathbb{R}^+ \text{ tal que } t(n) \leq c f(n) \text{ para todo natural } n$

$$O(f(n)) = \{t: \mathbb{N} \rightarrow \mathbb{R}^+ / \exists c_1 \in \mathbb{R}^+, n_0 \in \mathbb{N} \text{ tq } t(n) \leq c_1 f(n), n \geq n_0\}$$

Regla del Umbral

1) $f, t: N \rightarrow R^+$, $t(n) \in O(f(n)) \implies$ existe $c \in R^+$ tal que $t(n) \leq c f(n) \forall n \in N$

$t(n) \in O(f(n)) \implies \exists c_1 \in R^+, n_0 \in N$ tq $t(n) \leq c_1 f(n), n \geq n_0$ **(a)** xq

$$O(f(n)) = \{t: N \rightarrow R^+ / \exists c_1 \in R^+, n_0 \in N \text{ tq } t(n) \leq c_1 f(n), n \geq n_0\}$$

Deberíamos encontrar c tq $t(n) \leq c f(n) \forall n \in N$

Se puede definir $b = \max\{t(n) / f(n)\} \text{ con } 0 \leq n < n_0$

$$\implies t(n)/f(n) \leq b \implies t(n) \leq b f(n) \quad 0 \leq n < n_0 \quad \text{**(b)**}$$

Teniendo en cuenta **(a)** y **(b)** se define $c = \max(b, c_1)$ y valdría

$$c \in R^+ \text{ tal que } t(n) \leq c f(n) \forall n \in N$$

Faltaría probar la recíproca, \Leftarrow

Regla del Umbral

$t(n) \in O(f(n)) \iff \text{existe } c \in R^+ \text{ tal que}$
 $t(n) \leq c f(n)$ para todo natural n

2) Existe $c \in R^+$ tal que $t(n) \leq c f(n) \quad \forall n \in N$
 $\implies t(n) \in O(f(n))$

Esta demostración puede considerarse trivial por definición de $O(f(n))$

$$O(f(n)) = \{t:N \rightarrow R^+ / \exists c_1 \in R^+, n_0 \in N \text{ tq } t(n) \leq c_1 f(n), n \geq n_0\}$$

De 1) y 2) se tiene demostrada la Regla del Umbral

Regla del Máximo

$$f, g: N \rightarrow R^+, O(f(n) + g(n)) = O(\max(f(n), g(n)))$$

Idea de la demostración: se usará que

$$f(n) + g(n) = \min(f(n), g(n)) + \max(f(n), g(n))$$

$$0 \leq \min(f(n), g(n)) \leq \max(f(n), g(n))$$

sumando $\max(f(n), g(n))$ a todos los términos

$$\max(f(n), g(n)) \leq \min(f(n), g(n)) + \max(f(n), g(n)) \leq 2 \max(f(n), g(n))$$

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$$

Cont.

Regla del Máximo

$$f, g: N \rightarrow R^+, O(f(n) + g(n)) = O(\max(f(n), g(n)))$$

Cont.

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$$

- $t(n) \in O(f(n) + g(n)) \implies t(n) \in O(\max(f(n), g(n)))$
- $t(n) \in O(\max(f(n), g(n))) \implies t(n) \in O(f(n) + g(n))$

Regla del Máximo

$f, g: N \rightarrow R^+, O(f(n) + g(n)) = O(\max(f(n), g(n)))$

Cont.

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$$

- $t(n) \in O(f(n) + g(n)) \implies t(n) \in O(\max(f(n), g(n)))$
- $t(n) \in O(\max(f(n), g(n))) \implies t(n) \in O(f(n) + g(n))$

Regla del Máximo

$f, g: N \rightarrow R^+, O(f(n) + g(n)) = O(\max(f(n), g(n)))$

Cont.

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$$

- $t(n) \in O(f(n) + g(n)) \implies t(n) \in O(\max(f(n), g(n)))$
- $t(n) \in O(\max(f(n), g(n))) \implies t(n) \in O(f(n) + g(n))$

Regla del Máximo

- Vale para Θ
- Vale para suma de cualquier cantidad de func.
- Tener en cuenta que vale solo para funciones $f: N \rightarrow R^+$, porque sino

$$\begin{aligned}\Theta(n) &= \Theta(n + n^2 - n^2) = \Theta(\max(n, n^2, -n^2)) = \\ &= \Theta(n^2)\end{aligned}$$

Erróneo...

Regla del Máximo

- Vale para Θ
- Vale para suma de cualquier cantidad de func.
- Tener en cuenta que vale solo para funciones $f: N \rightarrow R^+$, porque sino

$$\begin{aligned}\Theta(n) &= \Theta(n + n^2 - n^2) = \Theta(\max(n, n^2, -n^2)) = \\ &= \Theta(n^2)\end{aligned}$$

Error

Regla del Máximo

- Vale para Θ
- Vale para suma de cualquier cantidad de func.
- Tener en cuenta que vale solo para funciones $f: N \rightarrow R^+$, porque sino

$$\begin{aligned}\Theta(n) &= \Theta(n + n^2 - n^2) = \Theta(\max(n, n^2, -n^2)) = \\ &= \Theta(n^2)\end{aligned}$$

Error

Error, $-n^2$ no es $N \rightarrow R^+$

Regla del Límite

- La idea de la notación **asintótica** tiene relación con la idea de crecimiento arbitrario de la E/ y del *comportamiento* de las funciones *en el límite*, de allí que se puede relacionar la notación asintótica con los límites
- 1) $\lim_{n \rightarrow \infty} f(n) / g(n) \in \mathbb{R}^+ \implies f(n) \in O(g(n)) \text{ y } g(n) \in O(f(n))$
 - 2) $\lim_{n \rightarrow \infty} f(n) / g(n) = 0 \implies f(n) \in O(g(n)) \text{ y } g(n) \notin O(f(n))$
 - 3) $\lim_{n \rightarrow \infty} f(n) / g(n) \rightarrow \infty \implies f(n) \notin O(g(n)) \text{ y } g(n) \in O(f(n))$

Regla del Límite

- Considerando los tres conjuntos definidos para la notación asintótica
 - 1) $\lim_{n \rightarrow \infty} f(n) / g(n) \in \mathbb{R}^+ \implies f(n) \in \Theta(g(n))$
 - 2) $\lim_{n \rightarrow \infty} f(n) / g(n) = 0 \implies f(n) \in O(g(n)) \text{ y } f(n) \notin \Theta(g(n))$
 - 3) $\lim_{n \rightarrow \infty} f(n) / g(n) \rightarrow \infty \implies f(n) \notin \Omega(g(n)) \text{ y } f(n) \notin \Theta(g(n))$

Complejidad Temporal

Máquinas de Turing No Determinísticas (MTN)

Una **MTN** es una Máquina de Turing que para un estado y un símbolo barrido por el cabezal, tiene un **conjunto finito de alternativas** para el siguiente movimiento. La MTN acepta su entrada si cualquier sucesión de alternativas de movimiento se detiene en q_A

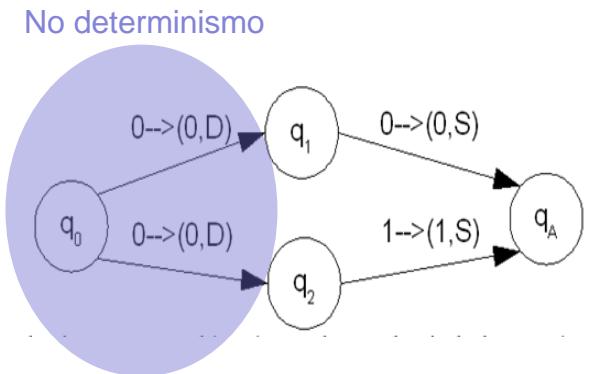
$$M = \langle Q, \Sigma, \Gamma, \Delta, q_0, q_A, q_R \rangle, \quad \Delta: Q \times \Gamma \rightarrow \rho((Q \cup \{q_A, q_R\}) \times \Gamma \times \{D, I, S\})$$

M acepta su entrada w , si existe al menos una computación a partir de w alcanza el estado q_A

Nota: Para referirnos a las máquinas de Turing determinísticas utilizaremos MTD. Obsérvese que una MTD es un caso particular de una MTN donde el conjunto de alternativas es siempre un conjunto unitario

Máquinas de Turing No Determinísticas (MTN)

Ejemplo: Construir una MTN M tal que
 $L(M) = \{w \in \{0,1,2,3\}^* / w \text{ comienza con } 00 \text{ ó } 01\}$



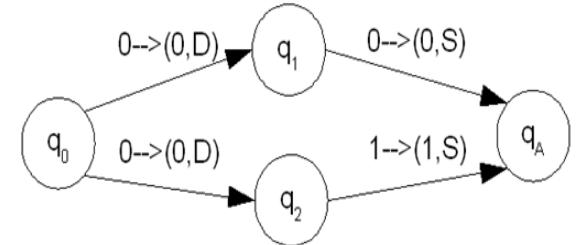
En el gráfico faltan todas las otras combinaciones de estado símbolo que tienen que llevar al estado q_R .

¿Donde se observa el no determinismo en el grafo de la función de transición?

Claramente hay dos posibilidades de movimiento para el mismo caso ($q_0, 0$)

Máquinas de Turing No Determinísticas (MTN)

Ejemplo: Construir una MTN M tal que
 $L(M) = \{w \in \{0,1,2,3\}^* / w \text{ comienza con } 00 \text{ ó } 01\}$



En el gráfico faltan todas las otras combinaciones de estado símbolo que tienen que llevar al estado q_R .

$$\Delta(q_0, 0) = \{(q_1, 0, D), (q_2, 0, D)\} \quad \text{No determinismo}$$

$$\Delta(q_0, x) = \{(q_R, x, S)\} \quad \text{para todo } x \in \Gamma - \{0\}$$

$$\Delta(q_1, 0) = \{(q_A, 0, S)\}$$

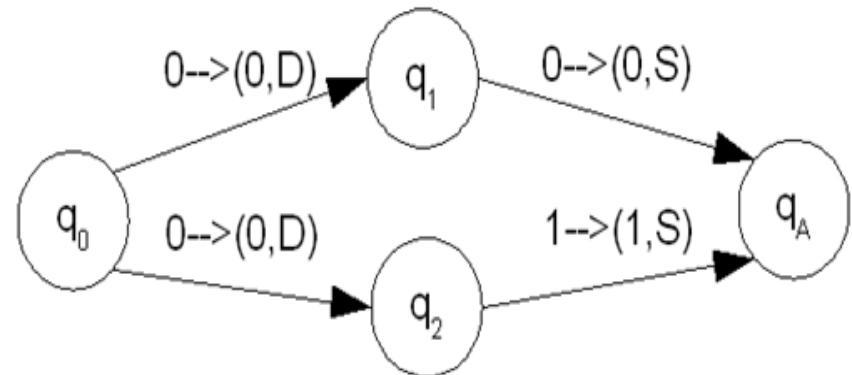
$$\Delta(q_1, x) = \{(q_R, x, S)\} \quad \text{para todo } x \in \Gamma - \{0\}$$

$$\Delta(q_2, 1) = \{(q_A, 1, S)\}$$

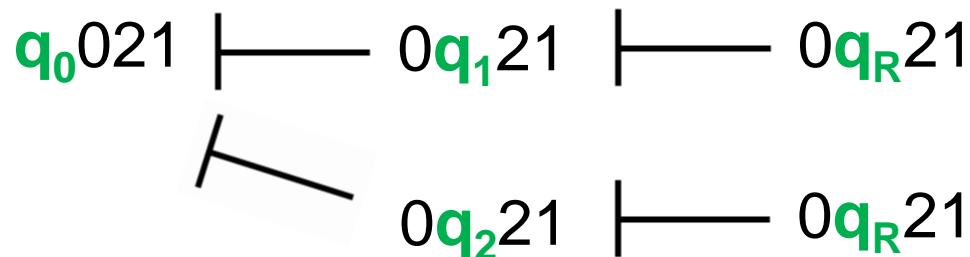
$$\Delta(q_2, x) = \{(q_R, x, S)\} \quad \text{para todo } x \in \Gamma - \{1\}$$

Máquinas de Turing No Determinísticas (MTN)

Traza de computación. Para una MTN la traza tendrá forma de árbol dónde cada rama representa una computación (una sucesión de alternativas de movimiento)



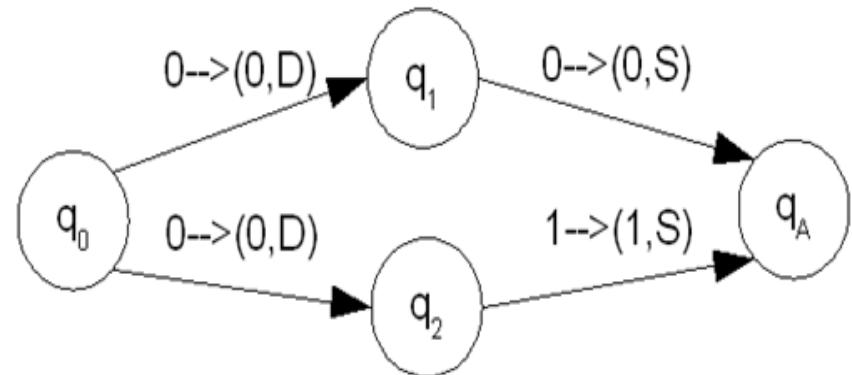
traza para $w = 021$



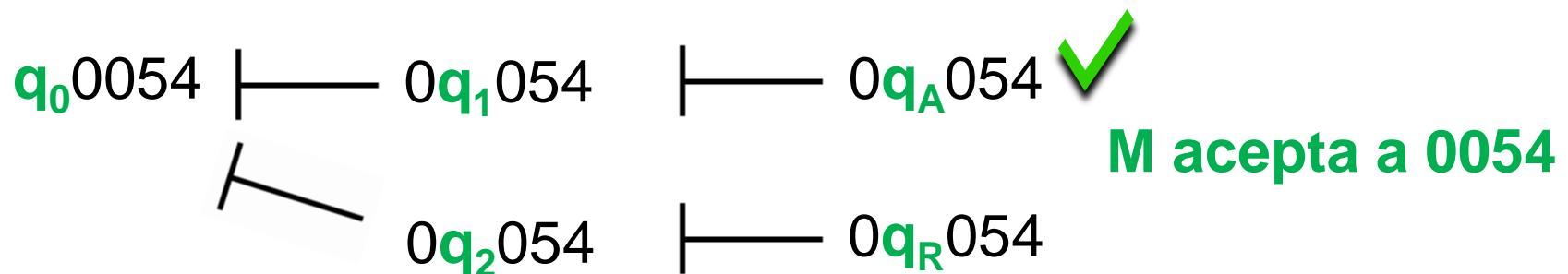
M rechaza a 021

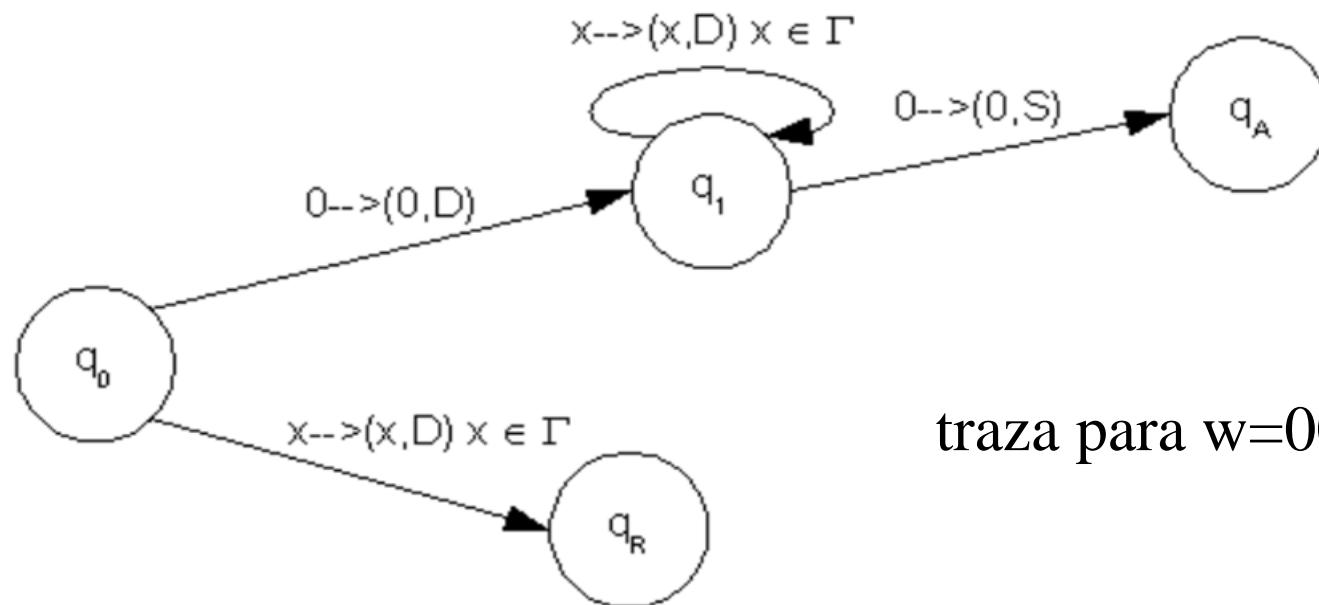
Máquinas de Turing No Determinísticas (MTN)

Traza de computación. Para una MTN la traza tendrá forma de árbol dónde cada rama representa una computación (una sucesión de alternativas de movimiento)

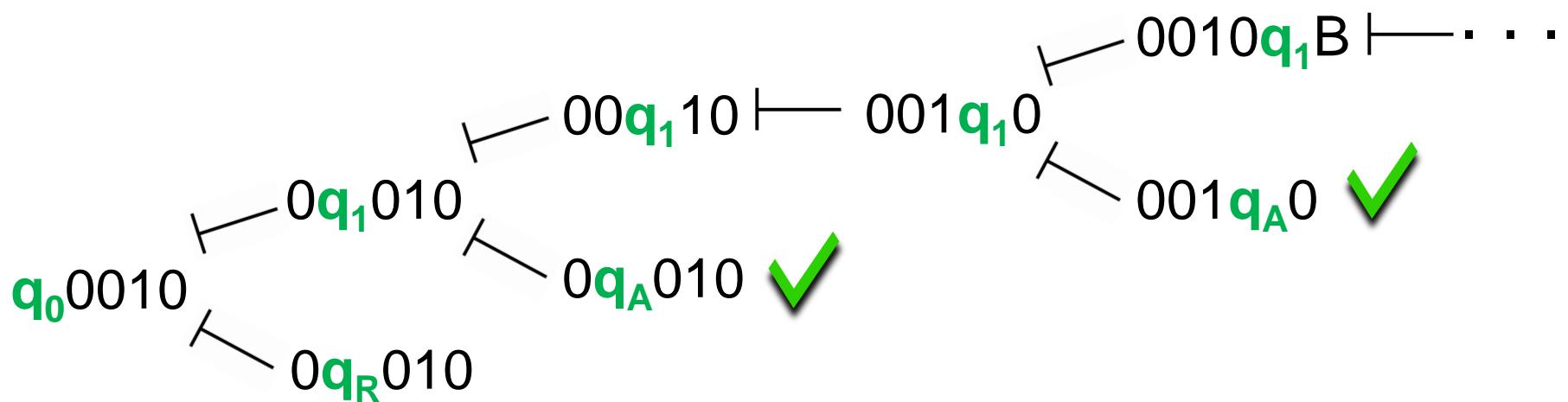


traza para $w=0054$





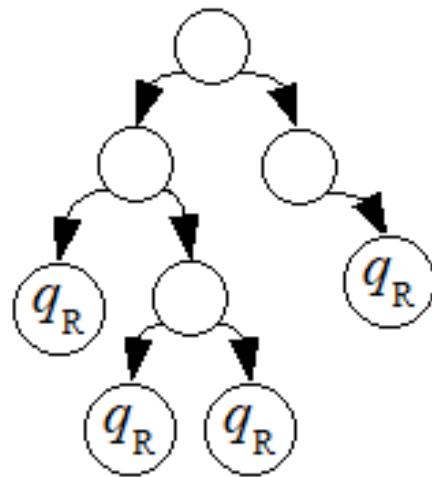
traza para $w=0010$



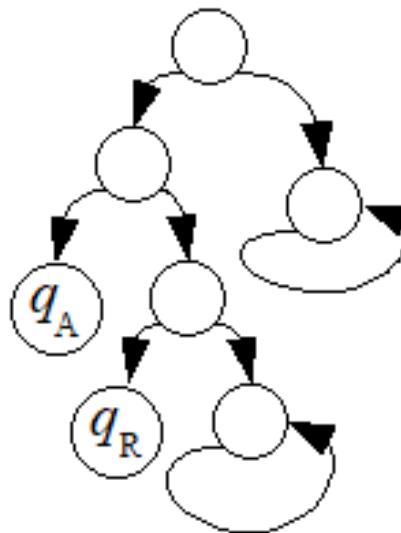
M acepta a 0010

Máquinas de Turing No Determinísticas (MTN)

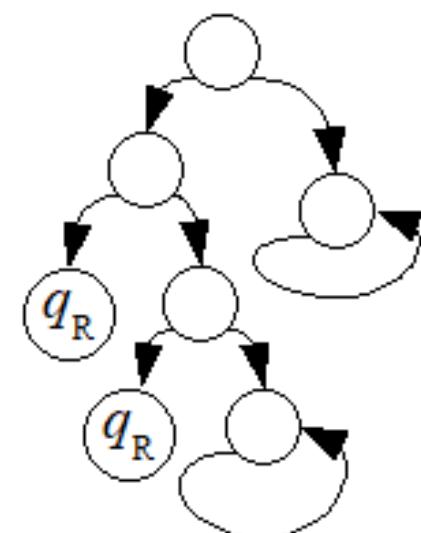
Gráfico de Aceptación / Rechazo



Rechaza



Acepta



Rechaza

Máquinas de Turing No Determinísticas (MTN)

Ejercicio: Construir una MTN tal que:

$$L(M) = \{w \in \Sigma^* / \text{cant}_1(w) \text{ sea divisible por } 2 \text{ o por } 3\}$$

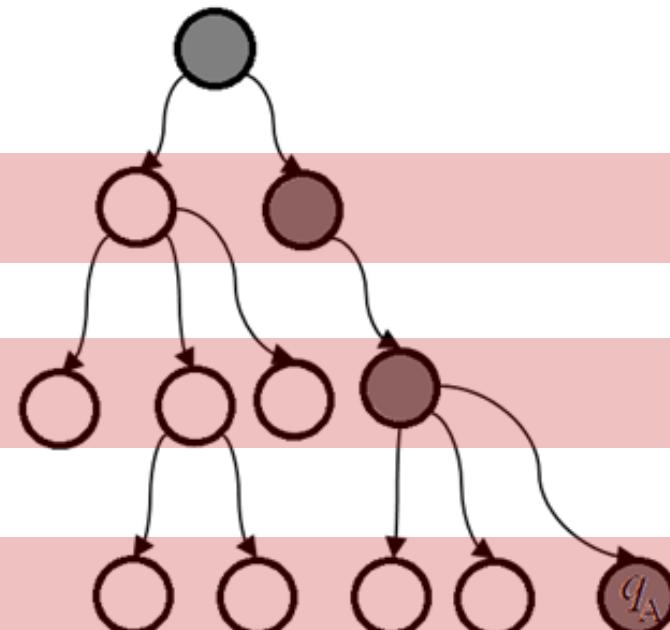
$$\Sigma = \{0, 1\}$$

Máquinas de Turing No Determinísticas (MTN)

Teorema: Si L es un lenguaje aceptado por una MTN M_1 entonces L es aceptado por una MTD M_2 .

Idea de la demostración: Realizar un recorrido BFS del árbol de computación

1)

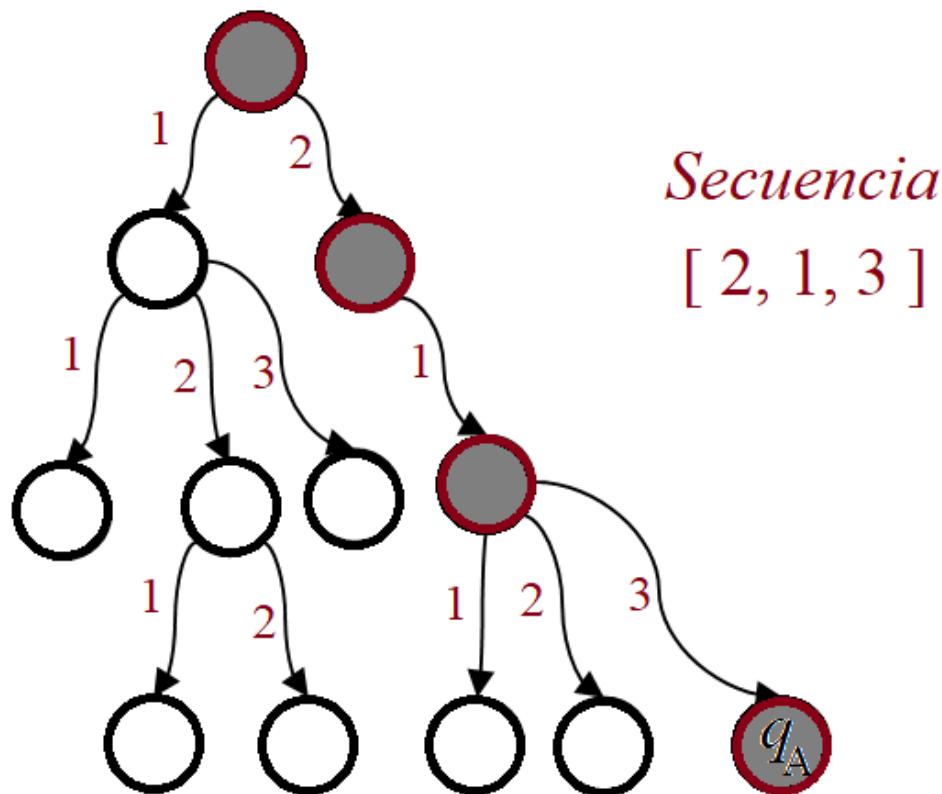


2)

3)

Máquinas de Turing No Determinísticas (MTN)

Sea r el número máximo de alternativas para cualquier par (estado, símbolo), cualquier computación (rama del árbol) puede representarse por una secuencia de dígitos del 1 a r .



Máquinas de Turing No Determinísticas (MTN)

Construimos M_2 con 3 cintas. En la cinta 1 está la entrada, en la cinta 2 M_2 va generando a medida que lo necesite secuencias de dígitos de 1 a r en orden canónico (primero las más cortas, para igual longitud según orden numérico). Supongamos $r = 3$, se irá generando:

[1] [2] [3] [1-1] [1-2] [1-3] [2-1] [2-2] [2-3] [3-1] [3-2] [3-3] [1-1-1] [1-1-2] ...

Para cada secuencia que va generándose en la cinta 2 M_2 copia la entrada en la cinta 3 y simula M_1 sobre la cinta 3 utilizando la secuencia de la cinta 2 para elegir los movimientos de M_1 . Algunas secuencias se descartan pues no existen r alternativas para todos los casos. Si M_1 alcanza q_A , M_2 acepta parando en q_A .

Máquinas de Turing No Determinísticas (MTN)

¿ $L(M_1) = L(M_2)$?

Claramente si existe una sucesión de alternativas que lleva a la aceptación, ésta en algún momento será generada en la cinta 2 y al simular M_1 , M_2 aceptará.

Si ninguna sucesión de alternativas de movimientos lleva a la aceptación de M_1 , M_2 tampoco aceptará.

Complejidad Temporal

Def. Sea M una MTD con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n , M hace a lo sumo $t(n)$ movimientos.

Def. Sea M una MTN con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n , ninguna secuencia de alternativas de movimiento ocasiona que M haga más de $t(n)$ movimientos.

Clases de Complejidad

Def. Un lenguaje $L \in \text{DTIME}(t(n))$ si existe una MTD M , con $L=L(M)$ tal que la complejidad temporal de M pertenece a $O(t(n))$

Def. Un lenguaje $L \in \text{NTIME}(t(n))$ si existe una MTN M , con $L=L(M)$ tal que la complejidad temporal de M pertenece a $O(t(n))$

Corolario: $\text{DTIME}(t(n))$ está incluido en $\text{NTIME}(t(n))$

Clases de Complejidad

Def. La clase de lenguajes **P** comprende todos los lenguajes para los que existe una **MTD** que lo acepta **en tiempo polinomial**

$$P = \bigcup_{i \geq 0} \text{DTIME}(n^i)$$

Def. La clase de lenguajes **NP** comprende todos los lenguajes para los que existe una **MTN** que lo acepta **en tiempo polinomial**

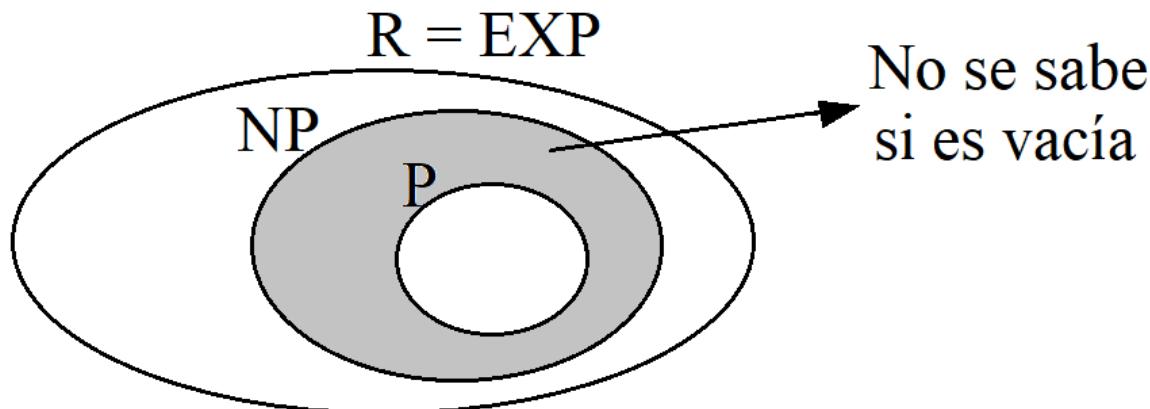
$$NP = \bigcup_{i \geq 0} \text{NTIME}(n^i)$$

Corolario: P es un subconjunto de NP (inmediato, pues las MTD son un caso particular de las MTN)

Clases de Complejidad

Teorema: Si L es un lenguaje recursivo aceptado por una MTN M_1 en tiempo $t(n)$, existe una MTD M_2 que lo acepta en tiempo menor o igual que $d^{t(n)}$, con d una constante mayor que 1 que depende de M_1

NOTA: Se sabe que existen lenguajes recursivos que no pertenecen a NP ($R-NP \neq \emptyset$) pero no se sabe si $NP=P$



Tratabilidad

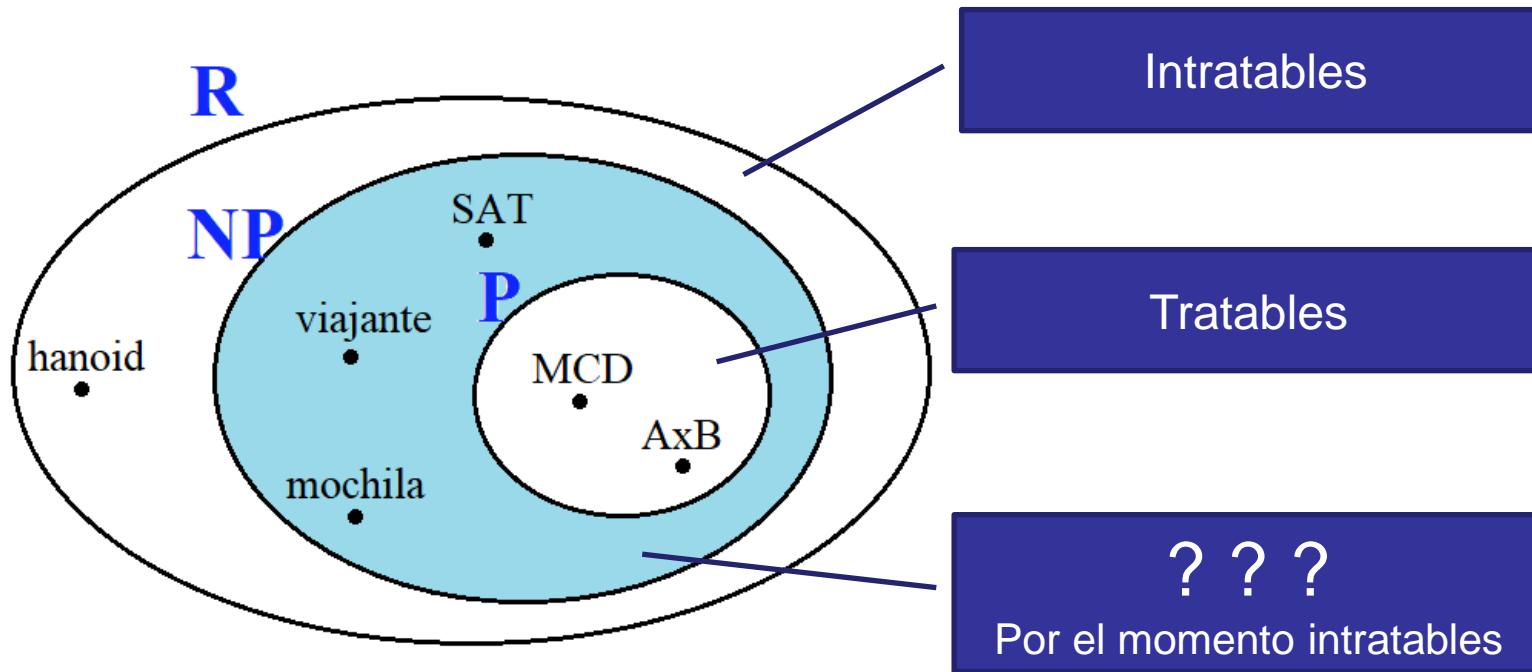
En general se está de acuerdo que la clase P representa a los problemas tratables. La mayoría de los problemas de esta clase tienen algoritmos con implementación razonable (Máximo Común Divisor, multiplicación de matrices, 2-SAT, etc).

Sin embargo existen ciertos problemas en P que realmente no son tratables debido a:

- el grado del polinomio es muy grande
- las constantes ocultas de la notación O son muy grandes
- la demostración de pertenencia a P no es constructiva y no se conoce algoritmo eficiente

Pero estos son casos extremos, y se considera a P una razonable aproximación al concepto de tratabilidad

Tratabilidad

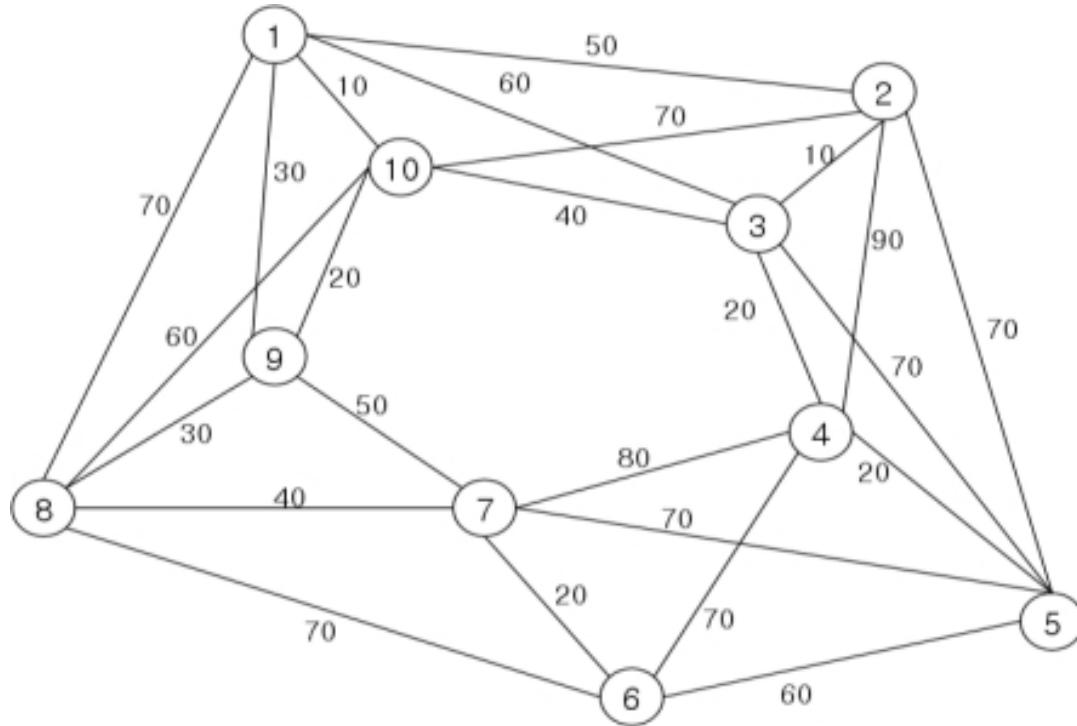


Esta brecha es inaceptablemente grande pues existen muchos problemas importantes que pertenecen a NP a los que no se le conoce una solución tratable, pero tampoco se ha demostrado que esta solución no exista.

Ejemplo clásico de problema NP

El **Problema del Viajante de Comercio** (TSP por sus siglas en inglés - *Travelling Salesman Problem*-)

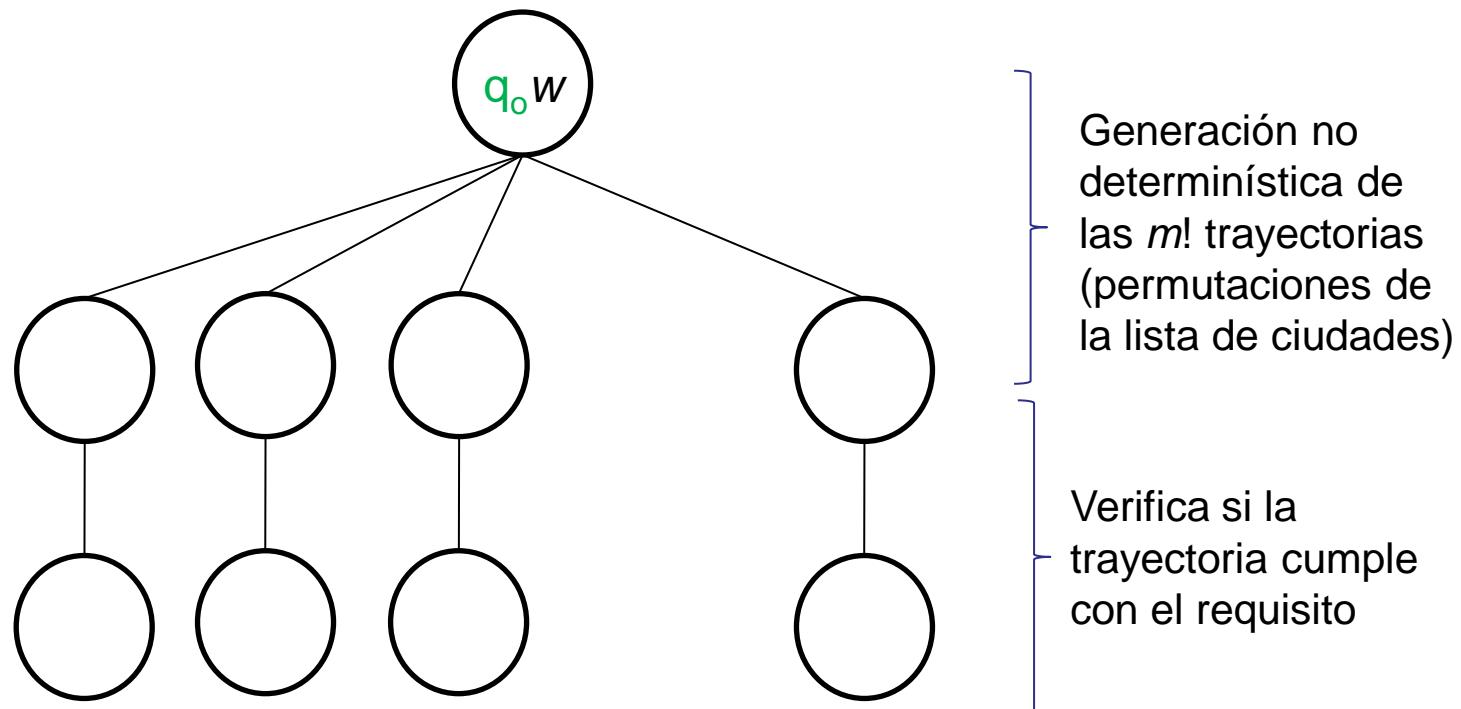
Dado un **conjunto de ciudades** y **caminos** que las unen, y una **longitud máxima d** , hay que decidir si existe alguna trayectoria que pase una sola vez por cada ciudad recorriendo una **distancia no mayor que d**



Ejemplo clásico de problema NP

TSP puede resolverse con una **MTN** de la siguiente manera:

- 1) El input w contiene la lista de m ciudades (identificadas con números de 1 a m), las distancias de los caminos entre las ciudades y la longitud máxima d permitida



Ejemplo clásico de problema NP

TSP puede resolverse con una **MTN** de la siguiente manera:

- 1) El input w contiene la lista de m ciudades (identificadas con números de 1 a m), las distancias de los caminos entre las ciudades y la longitud máxima d permitida
- 2) De **manera no determinística** se escribe una permutación cualquiera de los números entre 1 y m . Puede hacerse así:

Se escriben 2 listas, la primera con los números ordenados de 1 a m y la segunda vacía. **Se elige de manera no determinista** uno de la primera lista que se tacha y pasa a la segunda. Cada ciclo tiene un costo de $O(m)$, que se repite m veces hasta completar la permutación ($O(m^2)$), como la lista de ciudades es una parte de la entrada podemos acotarlo con $O(n^2)$

- 3) Se calcula la distancia del recorrido elegido. Suponiendo un costo $O(n)$ para localizar en la entrada la distancia entre 2 ciudades cualquiera, el costo de calcular la distancia del recorrido es $O(n^2)$
- 4) Si la distancia del recorrido es $\leq d$ para en q_A sino para en q_R .

Claramente M trabaja en tiempo $O(n^2)$

CO-NP

Definición: CO-NP = { $L / \bar{L} \in NP$ }

Teorema 1: Si $L \in P \Rightarrow L \in (NP \cap CO-NP)$

- a) Si $L \in P \Rightarrow L \in NP$ (por def. de P y NP)
- b) Si $L \in P \Rightarrow \bar{L} \in P$ (simplemente intercambiando q_A y q_R en la MTD)
 $\Rightarrow \bar{L} \in NP$ (por def. de P y NP)
 $\Rightarrow L \in CO-NP$ (por def. CO-NP)

De a) y b) se tiene que $L \in (NP \cap Co-NP)$

Nota: No se sabe cómo están relacionados NP y CO-NP. No se puede asumir que son iguales. Reconocer el complemento de un lenguaje, intentando intercambiar estados q_A por q_R de su MTN no funciona, una MTN puede aceptar teniendo computaciones que terminan en q_R pues alcanza que una sola termina en q_A .

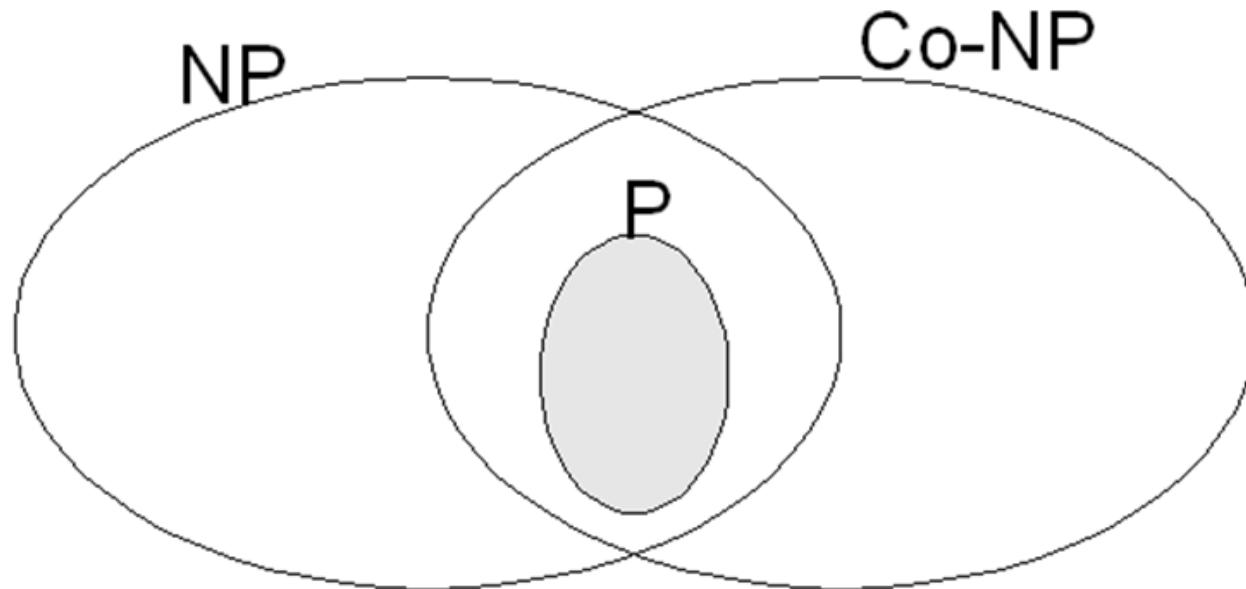
CO-NP

Teorema 2: $\text{NP} \neq \text{Co-NP} \Rightarrow \text{P} \neq \text{NP}$

Dem: Si $\text{P} = \text{NP} \Rightarrow \text{NP} = \text{CO-NP}$ (porque $\text{P} = \text{CO-P}$)

entonces $\text{NP} \neq \text{CO-NP} \Rightarrow \text{P} \neq \text{NP}$ (por contra-recíproca)

Esta es la situación conocida



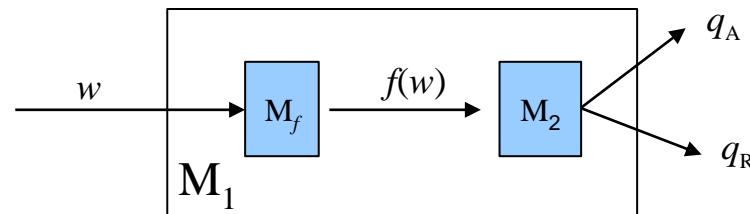
Reducción polinomial

Sean L_1 y L_2 dos lenguajes incluidos en Σ^* , decimos que L_1 se reduce polinomialmente a L_2 (se denota $L_1 \alpha_p L_2$) si $L_1 \subseteq L_2$ y además la función de reducción f es computada por una MTD que trabaja en tiempo polinomial ($f \in P$)

Reducción polinomial

Teorema 3: $L_1 \alpha_p L_2$ y $L_2 \in P \Rightarrow L_1 \in P$

Dem: Sea M_f la MTD que computa f en tiempo polinomial. Construimos M_1 una MTD tal que $L_1 = L(M_1)$ de la siguiente manera:



$L_1 = L(M_1)$?

Dado que $w \in L_1$ si $f(w) \in L_2$ se tiene que $L_1 = L(M_1)$

M_1 trabaja en tiempo polinomial ?

Sea $n = |w|$

M_f computa $f(w)$ en a lo sumo cn^k pasos,

por lo tanto $|f(w)| \leq cn^k$

por lo tanto M_1 hará a lo sumo $cn^k + c_2(cn^k)^{k^2}$

por lo tanto M_1 trabaja en tiempo $O(n^{k^3})$

Por lo tanto $L_1 \in P$

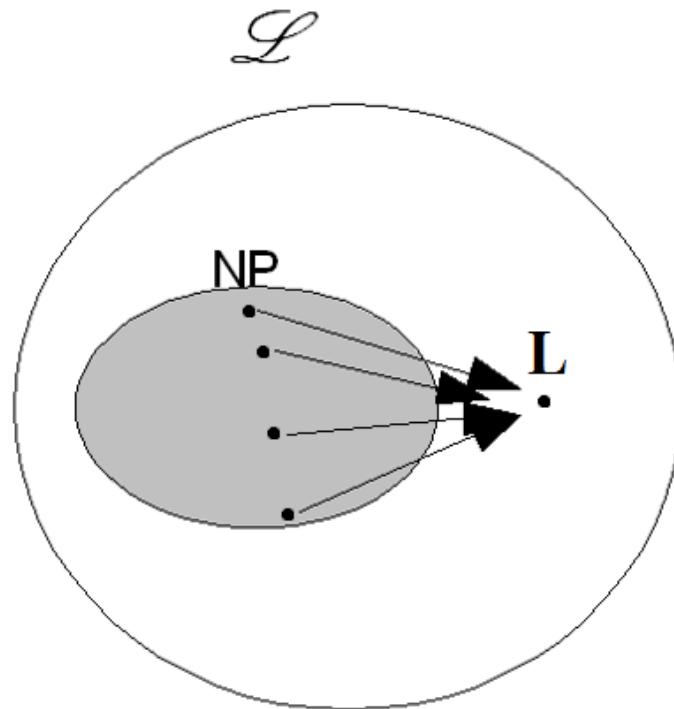
Reducción polinomial

Teorema 4: Sea L un lenguaje tal que $\emptyset \subset L \subset \Sigma^*$ entonces para cualquier lenguaje L' perteneciente a P , vale que $L' \alpha_p L$

Dem: Como $\emptyset \subset L \subset \Sigma^*$ entonces existe algún $x \in L$ y algún $y \notin L$. La reducción consiste para cada w , resolver L' con su MTD polinomial, y asignar x o y según se *acepte* o *rechace* w respectivamente.

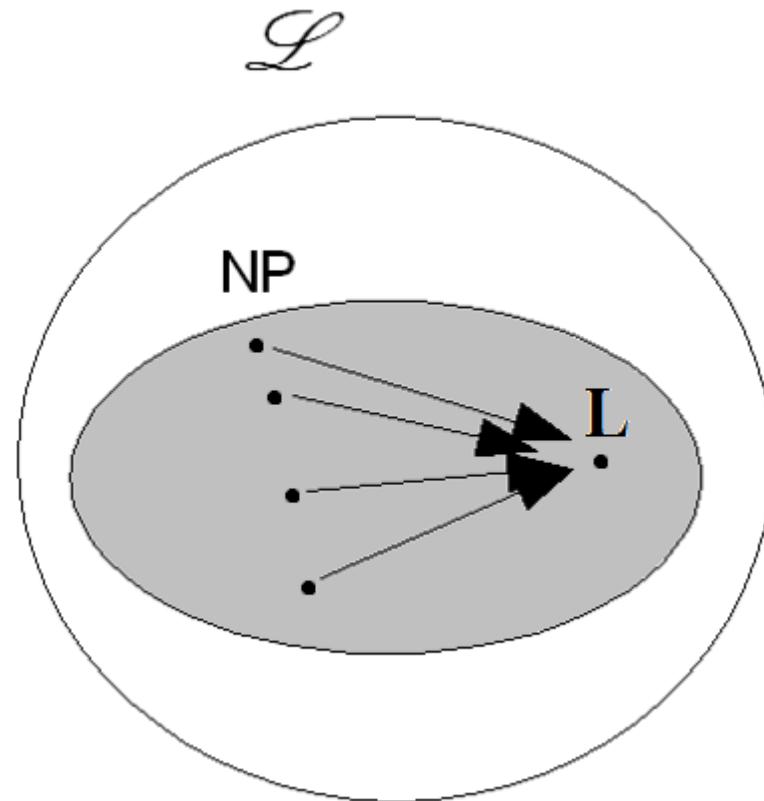
NP-Hard

Definición: $L \in \text{NPH}$ (NP-Hard) si para todo $L' \in \text{NP}$ se cumple $L' \leq_p L$



NP-Completo

Definición: $L \in \text{NPC}$ (NP-Completo) si $L \in \text{NPH}$ y $L \in \text{NP}$



NP-Completo

Hasta ahora no se ha encontrado ningún lenguaje que cumpla con esto

Teorema 5: Si $(L \in \text{NPC}) \text{ AND } (L \in P) \Rightarrow P = NP$

Dem: Sea L' un lenguaje arbitrario de NP

$$\begin{aligned} L' \in NP &\Rightarrow L' \leq_p L \text{ (porque por hipótesis } L \in \text{NPC)} \\ &\Rightarrow L' \in P \text{ (por hipótesis } L \in P \text{ y por teorema 3)} \end{aligned}$$

Por lo tanto $NP \subseteq P$

Por lo tanto $P = NP$

Nota: Según este teorema para demostrar que $P = NP$ alcanzaría con encontrar una solución polinomial para cualquiera de los problemas NPC conocidos.

NP-Completo

Teorema 6: Sean $L_1, L_2 \in \text{NP}$. Si $L_1 \in \text{NPC}$ y $L_1 \alpha_p L_2$ entonces $L_2 \in \text{NPC}$

Dem: Para todo $L \in \text{NP}$ existe $L \alpha_p L_1$ (por $L_1 \in \text{NPC}$)

Dado que por hipótesis $L_1 \alpha_p L_2$ se tiene que:

Para todo $L \in \text{NP}$ existen $L \alpha_p L_1$ y $L_1 \alpha_p L_2$

Por lo tanto, para todo $L \in \text{NP}$ existe $L \alpha_p L_2$ (porque α_p es transitiva)

Por lo tanto $L_2 \in \text{NPH}$ (por definición de NPH)

Por lo tanto $L_2 \in \text{NPC}$ (por hipótesis $L_2 \in \text{NP}$)

Ejercicio para el lector: demostrar que α_p es transitiva

Ejemplo de lenguaje NPC - Lenguaje SAT

Literal: es una variable proposicional o la negación de una variable proposicional.

FNC: fórmula normal conjuntiva, es una conjunción de cláusulas, donde una cláusula es una disyunción de literales. Ejemplo:

$$\varphi = (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_4)$$

Enunciado del problema SAT: Dada una fórmula proposicional en FNC, determinar si es satisfactible, es decir si existe alguna asignación de verdad para las variables que haga verdadera la fórmula.

$$\text{SAT} = \{\varphi \mid \varphi \text{ es una fórmula booleana FNC satisfactible}\}$$

Ejemplo de lenguaje NPC - Lenguaje SAT

Se ha demostrado que SAT pertenece a NPC (**Teorema Cook/Levin**).

Se prueba que SAT pertenece a NP y que para cualquier lenguaje arbitrario L perteneciente a NP existe $L \alpha_p \text{SAT}$.

La idea de la reducción es la siguiente:

Como L pertenece a NP sabemos que existe una MTN M (con $L=L(M)$) que acepta o rechaza un string w en tiempo polinomial.

Se define una función de reducción que puede ser computada en tiempo polinomial por una MTD M_f que a partir de una entrada w y una MTN M define una fórmula bien formada $\varphi = f(M, w)$ con el objetivo que φ sea satisfactible si y solo si M acepta w

Ejemplo de lenguaje NPC - Lenguaje SAT

Más de 1000 problemas de dominios diferentes, y con variadas aplicaciones se han probado que pertenecen a NPC haciendo reducciones desde un lenguaje que ya se conozca su pertenencia a NPC. Para ninguno de ellos se ha podido encontrar una solución polinomial.

Ejemplos:

SAT α_p 3-SAT

3-SAT α_p VC={(G,k)/ G es un grafo que tiene un cubrimiento de vértices de tamaño k }

Sea $G=(V,E)$ un grafo y sea $C \subset V$ un subconjunto de vértices. Decimos que C es un cubrimiento de vértices si cualquier $e \in E$ tiene un extremo que pertenece a C .

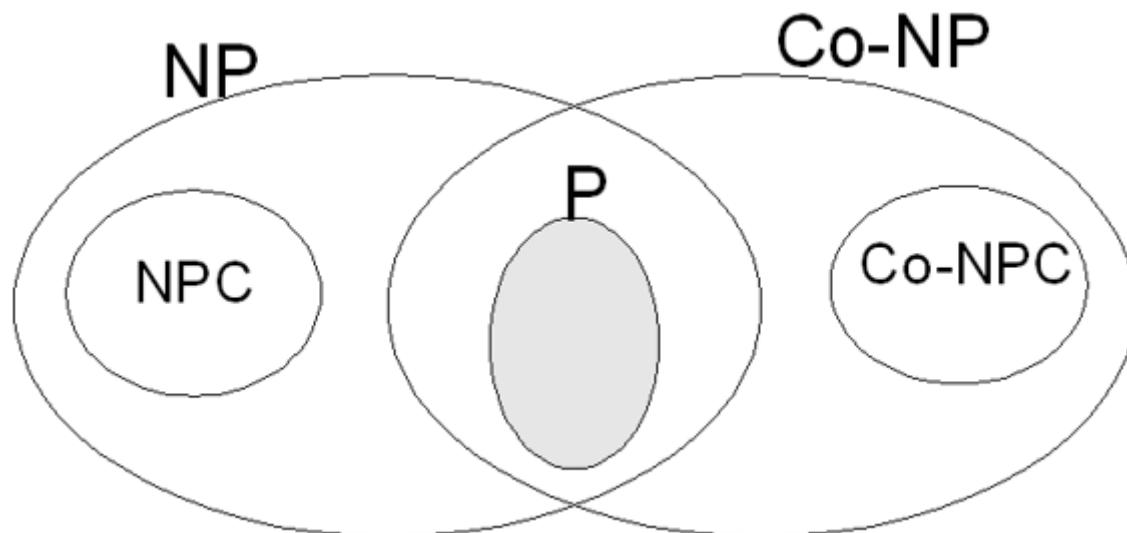
VC α_p k -clique={(G,k)/ G tiene un clique de tamaño k },

Clique es un subgrafo totalmente conectado

¿? $P = NP$??

Por varias décadas de estudio los investigadores no han podido responder a esta pregunta.

Por ello se cree que $P \neq NP$ es mas plausible que $P = NP$



Análisis de Algoritmos

Introducción

Contexto

- Ya podemos hacer la asociación
 - Problema computacional \iff MTD
 - Entrada \iff Instancia del problema comput.
 - Problemas computables \iff R

Contexto

- Ya podemos hacer la asociación
 - Problema computacional \iff MTD
 - Entrada \iff Instancia del problema comput.
 - Problemas computables \iff R
 - Complejidad \iff “Análisis” de R
 - Computable \neq “Tratable”
 - MTD \iff Algoritmo

Contexto

- Ya podemos hacer la asociación
 - Problema computacional \iff MTD
 - Entrada \iff Instancia del problema comput.
 - Problemas computables \iff R
 - Complejidad \iff “Análisis” de R
 - Computable \neq “Tratable”
 - MTD \iff Algoritmo
 - Problema computable \iff Infinitas MTD (alg.)
 - MTD \iff Cantidad de pasos (δ de transición)

Contexto

- De la clase inicial

Temas:

Algoritmos (específico) ==> Algoritmia, Análisis,
Notación Asintótica

Computabilidad
Complejidad

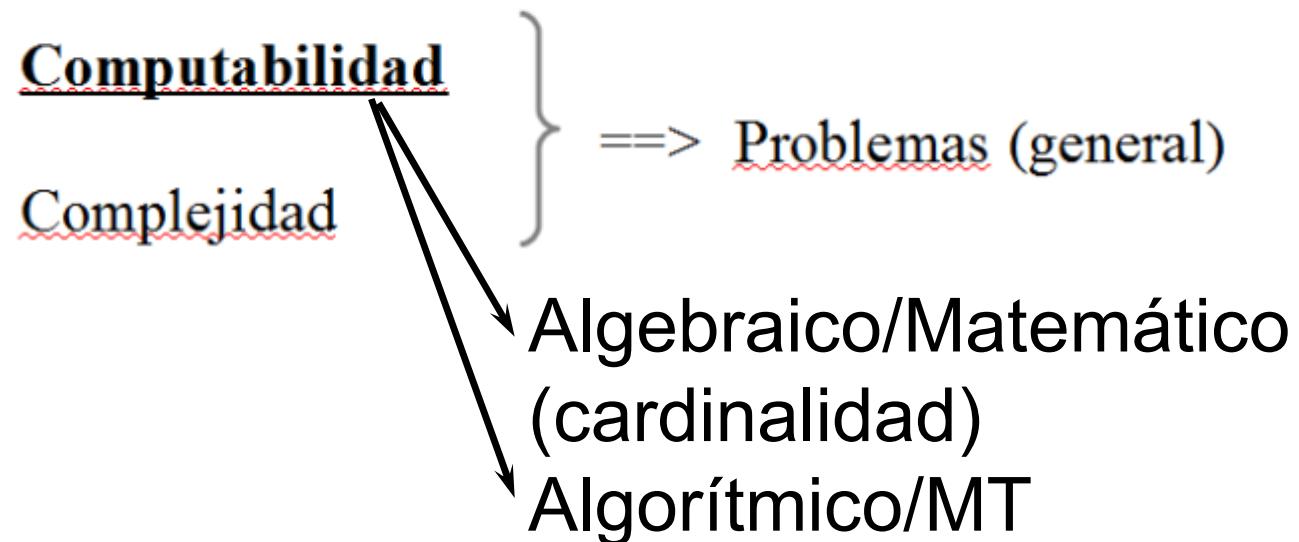
} ==> Problemas (general)

Contexto

- De la clase inicial

Temas:

Algoritmos (específico) ==> Algoritmia. Análisis, Notación Asintótica

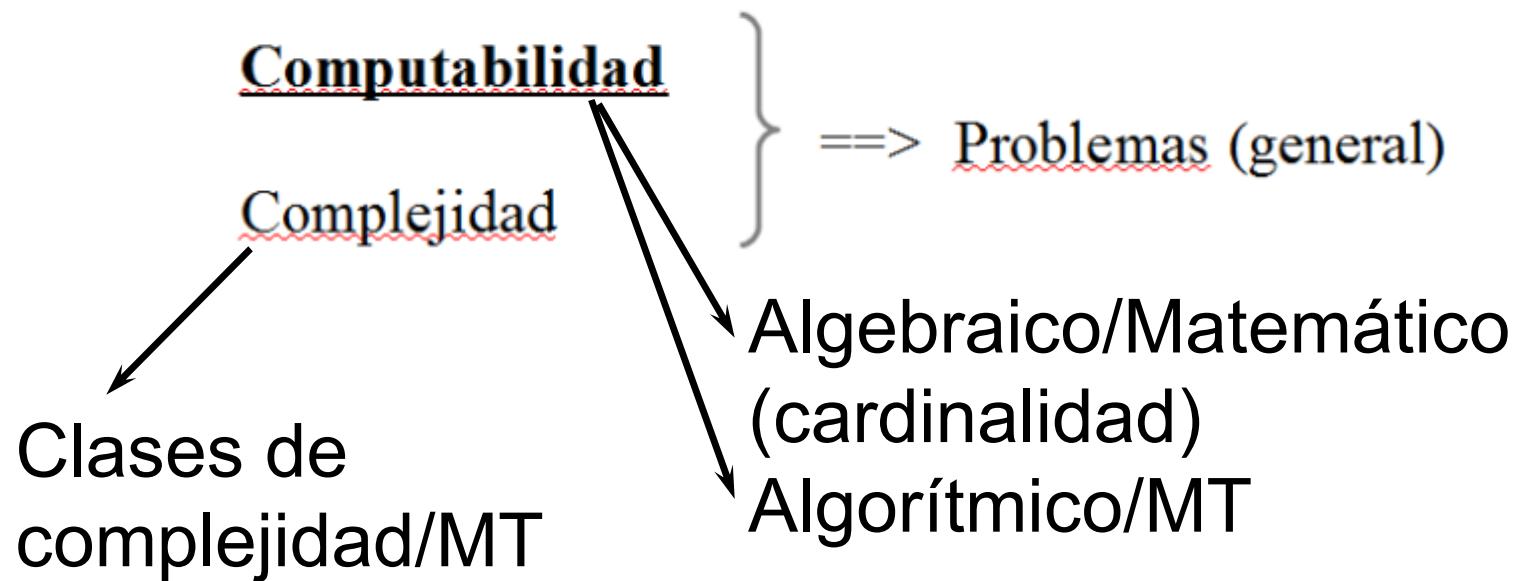


Contexto

- De la clase inicial

Temas:

Algoritmos (específico) ==> Algoritmia, Análisis,
Notación Asintótica



Algoritmia

Temas:

Algoritmos (específico) ==> Algoritmia. Análisis,
Notación Asintótica

Algoritmia

Temas:

Algoritmos (específico) ==> Algoritmia. Análisis, Notación Asintótica

Algoritmia

Diseño (creación-propuesta)
Análisis (comportamiento)

} ==> De algoritmos

Algoritmia

- Diseño: lo han visto hasta este punto en múltiples asignaturas
- Análisis de Algoritmos (Leiserson): "The theoretical study of computer-program performance and resource usage"
- Análisis de Eficiencia: Cantidad de recursos que se utilizan (tiempo, espacio)

Algoritmia

- ¿Por qué hacer análisis de algoritmos?
 - Comparación teórica
 - Escalabilidad (cuánto más grande se puede, hasta dónde llego)
 - Matemática algorítmica provee un lenguaje de comportamiento de programas
 - Análisis de t y e se pueden relacionar en algunos casos

Algoritmia

- El análisis teórico está apoyado por el principio de invarianza:
 - "Según el principio de invarianza, dos implementaciones diferentes del mismo algoritmo no difieren en tiempo de ejecución más que en una constante multiplicativa"
- Relación directa con
 - Sea M una MTD con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n , M hace a lo sumo $t(n)$ mov.

Tiempo y Espacio

- Análisis de t y e
 - Dado un algoritmo A, el tiempo de ejecución $t_A(n)$ de A es la cantidad de pasos, operaciones o acciones elementales que debe realizar el algoritmo al ser ejecutado en una instancia de tamaño n.
 - El espacio $e_A(n)$ de A es la cantidad de datos elementales que el algoritmo necesita al ser ejecutado en una instancia de tamaño n, sin contar la representación de la entrada

Análisis de Algoritmos

- Definiciones de $t_A(n)$ y $e_A(n)$ son ambiguas en dos sentidos:
 - No especifica claramente cuáles son las operaciones o datos elementales.
 - Dado que existe más de una instancia de tamaño n no está claro cuál de ellas es la que se tiene en cuenta para el análisis (o cuáles, si son relevantes).

Análisis de Algoritmos

- Operaciones Elementales:
 - Tiempo de ejecución está acotado por una constante que depende sólo de la implementación usada
 - No depende del tamaño de la entrada
 - Sólo interesa el número de operaciones elementales
 - En general, las sumas, multiplicaciones y asignaciones son operaciones elementales

Análisis de Algoritmos

- Tipos de instancias (¿Por qué) - Tipos de análisis (si hay dependencia de datos): peor-mejor-promedio-probabilístico
- Relación con
 - Sea M una MTD con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n , M hace a lo sumo $t(n)$ mov.

Análisis de Algoritmos

- Tipos de instancias (¿Por qué) - Tipos de análisis (si hay dependencia de datos): peor-mejor-promedio-probabilístico
- Relación con
 - Sea M una MTD con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n , M hace **a lo sumo $t(n)$ mov.**

Análisis de Algoritmos

- Notación Asintótica
 - Utilizaremos la noción de tiempo ignorando las constantes multiplicativas que lo afectan
 - Caracteriza en forma simple la eficiencia (o uso de recursos) de los algoritmos
 - Se independiza de las características específicas de la implementación
 - Análisis del comportamiento de las funciones en el límite, considerando sólo su tasa de crecimiento (notación asintótica)

Análisis de Algoritmos

- (1) Estructuras de control
- (2) Barómetro
- (3) Análisis del caso promedio
- (4) Análisis amortizado
- (5) Recurrencias } ¿? Un tipo específico de algoritmos
- (6) No veremos análisis asociados a algoritmos específicos *de* estructuras de datos
- (7) Diseño + análisis {
 - Greedy
 - Divide-and-Conquer
 - Dynamic programming
 - Algoritmos probabilísticos

Análisis de Algoritmos

1. Estructuras de Control



Análisis de Algoritmos

- (1) Estructuras de control
- (2) Barómetro
- (3) Análisis del caso promedio
- (4) Análisis amortizado
- (5) Recurrencias } ¿? Un tipo específico de algoritmos
- (6) No veremos análisis asociados a algoritmos específicos *de* estructuras de datos
- (7) Diseño + análisis
 - Greedy
 - Divide-and-Conquer
 - Dynamic programming
 - Algoritmos probabilísticos



1. Estructuras de Control

- Secuencias:

$P_1; P_2 \quad t_1(n) \text{ y } t_2(n)$

$$t_{1;2}(n) = t_1(n) + t_2(n)$$

Regla del máximo

$$t_{1;2}(n) = t_1(n) + t_2(n) \in O(\max(t_1(n), t_2(n)))$$

$$t_{1;2}(n) = t_1(n) + t_2(n) \in \Theta(\max(t_1(n), t_2(n)))$$



1. Estructuras de Control

- Condicional (dep. de n asumido):

If (cond) t1

Then (cuerpo then) t2

Else (cuerpo else) t3

Se considera directamente el peor caso:

$t_1 + \max(t_2, t_3)$ o directamente

$\max(t_1, t_2, t_3)$



1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For $i \leftarrow 1$ to m

Do $P(i)$

Puede ser $P(i, n)$, $t(i)$

– Si $t(i)$ no depende de $i \Rightarrow t(i) = t$

- $t_{\text{for}} = m t$
- En cualquier caso, identificar #iter
- $t_{\text{for}} = \#iter t$



1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For $i \leftarrow 1$ to m

Do $P(i)$

Puede ser $P(i, n)$, $t(i)$

- Si $t(i)$ depende de i

- $t_{\text{for}} = \sum_{i=1}^m t(i)$

- Sumas útiles:

- $\sum_{i=1}^m i = \frac{m(m+1)}{2}$

- $\sum_{i=1}^m i^2 = \frac{m(m+1)(2m+1)}{6}$



1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For $i \leftarrow 1$ to m

Do $P(i)$

Puede ser $P(i, n)$, $t(i)$

- No confundir “peor caso” con el límite

- For $j \leftarrow i$ to m o For $j \leftarrow 1$ to i
- Cantidad de iteraciones: $m-i+1$ o i
- No hay “peor caso” ($i=1$, $i=m$)



1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For $i \leftarrow 1$ to m

Do $P(i)$

Puede ser $P(i, n)$, $t(i)$

- Todo lo anterior vale solamente si $1 \leq m$
- En general: $\text{inicio} \leq \text{fin}$



1. Estructuras de Control

- Iteraciones no uniformes
 - while y repeat
 - Cantidad de iteraciones desconocida a priori
 - Dos formas de análisis
 - Funciones de variables que decrece
 - Recurrencias
 - Veremos ambas formas con un ejemplo



1. Estructuras de Control

- Iteraciones no uniformes

function bin_search(T[1...n], x) // x está en T

i \leftarrow 1; j \leftarrow n

While i < j Do ==> Cantidad determinada por la dif.

k \leftarrow (i + j) % 2

Case x < T[k]: j \leftarrow k-1

x = T[k]: i, j \leftarrow k // Return k

x > T[k]: i \leftarrow k+1

Return i

(cont.)



1. Estructuras de Control

- Iteraciones no uniformes: función
 - Variables de la iteración
 - El valor de la función decrece a medida que se llevan a cabo más iteraciones
 - El valor de la función debe ser un entero positivo ==> el algoritmo termina
 - ¿Cuándo? Entender la forma en que decrece la función. Ej: bin_search

(cont.)



1. Estructuras de Control

- Iteraciones no uniformes: función
 - `bin_search`
 - Mejor caso: se encuentra en la 1ra evaluación
 - Peor caso: se encuentra cuando $i=j$
 - Función $d = j - i + 1$ (cantidad de elems.)
 - $d \geq 2$ (análisis de “mitad”)
 - $d = 1$ (índice del elem.)

(cont.)



1. Estructuras de Control

- Iteraciones no uniformes: función

- `bin_search`: $d = j - i + 1$

- ¿Decrece? Veamos los valores de i , j y d al principio y al final de una iteración cualquiera, i' , j' , y d' :

- 1) $x < T[k]$:

$$j' = (i + j) \div 2 - 1; \quad i' = i;$$

$$d' = (i + j) \div 2 - 1 - i + 1 \leq (i + j) / 2 - i$$

$$= -i/2 + j/2 < -i/2 + j/2 + 1/2 = (j - i + 1) / 2 = d/2 < d$$

(cont.)



1. Estructuras de Control

- Iteraciones no uniformes: función

- bin_search: $d = j - i + 1$

¿Decrece? Veamos los valores de i , j y d al principio y al final de una iteración cualquiera, i' , j' , y d' :

2) $x > T[k]$:

$$j' = j; \quad i' = (i + j) \div 2 + 1;$$

$$d' = j - (i + j) \div 2 - 1 + 1 \leq j - (i + j) / 2 =$$

$$= j/2 - i/2 < j/2 - i/2 + 1/2 = (j - i + 1) / 2 = d/2 < d$$

(cont.)



1. Estructuras de Control

- Iteraciones no uniformes: función

- `bin_search`: $d = j - i + 1$

- ¿Decrece? Veamos los valores de i , j y d al principio y al final de una iteración cualquiera, i' , j' , y d' :

- 3) $x = T[k]$:

$$d' = 1 \leq d/2$$

(cont.)

$\Rightarrow d = j-i+1$ decrece en cada iteración



1. Estructuras de Control

- Iteraciones no uniformes: función

- bin_search: $d = j - i + 1$

- ¿Cuántas iteraciones? d_k : valor de $j_k - i_k + 1$ al final de la iteración k

$$d_0 = n$$

$$d_1 = n/2$$

...

$$d_i = n/2^i$$

Peor caso, “todas” las iteraciones hasta que $d_k = 1$

$$d_k = n/2^k = 1; \text{ ¿k? } n = 2^k \implies \log_2 n = \log_2 2^k \implies$$

$$k = \log_2 2^n \implies \lceil k = \log_2 n \rceil$$



1. Estructuras de Control

- Iteraciones no uniformes
 - Dos formas de análisis
 - Funciones de variables que decrece
 - Explicación y ejemplo bin_search
 - Recurrencias



1. Estructuras de Control

- Iteraciones no uniformes: recurrencias
 - $t(n)$: t para resolver el problema con n elem.
 - $$t(n) = \begin{cases} 1 & n = 1 \\ t(n/2) + c & n > 1 \end{cases}$$
 - Veremos las recurrencias más adelante
- En general, las iteraciones no uniformes son complicadas de analizar (función y/o recurrencia)



1. Estructuras de Control

- Resumen:
 - Paso a paso, muy detallado
 - Iteraciones no uniformes complicadas
 - $t(n)$ relativamente detallado y “combinado”
 - Puede implicar mucho tiempo por el detalle de cada estructura de control



Análisis de Algoritmos

- 2. Barómetro
- 3. Promedio
- 4. Amortizado



Análisis de Algoritmos

- (1) Estructuras de control
- (2) Barómetro
- (3) Análisis del caso promedio
- (4) Análisis amortizado
- (5) Recurrencias } ¿? Un tipo específico de algoritmos
- (6) No veremos análisis asociados a algoritmos específicos *de* estructuras de datos
- (7) Diseño + análisis
 - Greedy
 - Divide-and-Conquer
 - Dynamic programming
 - Algoritmos probabilísticos



2. Instrucción Barómetro

- Una instrucción barómetro es la que se ejecuta al menos tantas veces como cualquier otra excepto quizás una cantidad constante (acotada) de veces.
- Si $t(n)$ es el tiempo del algoritmo a analizar, $t(n)$ es $\Theta(f(n))$ donde $f(n)$ es la cantidad de veces que se ejecuta la instrucción barómetro



2. Instrucción Barómetro

- A tener en cuenta/conocer:
 - Instrucción barómetro
 - Encontrar $f(n)$
 - Se usan los principios de las estructuras de control para determinar la cantidad de veces que se ejecuta la instrucción barómetro



2. Instrucción Barómetro

- Ej:

Function select_sort(T[1...n]) // n pasadas

For i ← 1 to n-1 Do

 minj ← i; minx ← T[i]

 For j ← i+1 to n Do

 If T[j] < minx Then

 minj ← j

 minx ← T[j]

 T[minj] ← T[i]

 T[i] ← minx



2. Instrucción Barómetro

- Ej:

Function select_sort(T[1...n]) // n pasadas

For i \leftarrow 1 to n-1 Do

 minj \leftarrow i; minx \leftarrow T[i]

 For j \leftarrow i+1 to n Do

 If T[j] < minx Then

 minj \leftarrow j

 minx \leftarrow T[j]

 T[minj] \leftarrow T[i]

 T[i] \leftarrow minx



2. Instrucción Barómetro

- Resumen
 - Identificación de barómetro
 - Cantidad de veces \Leftrightarrow Estructuras de contr.
 - $t(n) \in \Theta(f(n))$
 - No se tiene $t(n)$...



3. Caso Promedio

- Uso implícito/explícito de distribución de probabilidades de las instancias de tamaño n
- En principio, para ordenar vectores, el tiempo de las $n!$ posibles dividido $n!$
- Asumir que son todas igualmente probables y usar esto en el análisis



3. Caso Promedio

- Ej. de insertion sort, Brassard/Bratley (p. 62, 111)
- En este caso específico, se tiene en cuenta la distribución de probabilidad de los números del arreglo en partes cada vez menores del mismo.
- Es complementaria al análisis de estructuras de control o barómetro (casos)



4. Análisis Amortizado

- Para los casos en los que
 - Es muy poco probable que en todas las llamadas se tenga siempre el peor caso
 - La cantidad de operaciones está relacionada con una secuencia de uso de un algoritmo
 - Ej: estructuras de datos, inserción en un grafo
 - Ej: IncBin a continuación



4. Análisis Amortizado

- IncBin

Procedure IncBin(Cntr[1...m]) // Cntr[i] $\in \{0, 1\}$

$j \leftarrow m+1$

 Repeat

$j \leftarrow j - 1$

$C[j] \leftarrow 1 - C[j]$

 Until ($C[j] = 1$) Or ($j = 1$)

- Se tiene el peor caso solamente 1 vez cada ...



4. Análisis Amortizado

- IncBin

Errores en la explicación de la diapositiva anterior:

- Los bits no se incrementan, se invierten (es lo necesario para incrementar el contador binario representado con un array de bits)
- Los bits se invierten de derecha a izquierda, no de izquierda a derecha, es correcta la expresión “menos a más significativos”, es incorrecto “de izquierda a derecha”



4. Análisis Amortizado

- Promedio de $t(n)$ en llamadas sucesivas, no independientes
- Tres métodos
 - Agregado
 - Contable
 - Potencial
- Sería
 - Solo para algunos casos/algoritmos
 - Alternativo a casos mejor/peor/probabilístico



Análisis de Algoritmos

5. Recurrencias

Análisis de Algoritmos

- (1) Estructuras de control
- (2) Barómetro
- (3) Análisis del caso promedio
- (4) Análisis amortizado
- (5) Recurrencias } ¿? Un tipo específico de algoritmos
- (6) No veremos análisis asociados a algoritmos específicos *de* estructuras de datos
- (7) Diseño + análisis {
 - Greedy
 - Divide-and-Conquer
 - Dynamic programming
 - Algoritmos probabilísticos

5. Recurrencias

- Funciones de t dadas en función de sí mismas: "An equation that defines a function in terms of its own value on smaller inputs"
- Una recurrencia describe una secuencia de números, donde el/los término/s inicial/es (cantidad finita) son dados explícitamente y los siguientes términos se definen como una función de uno o más anteriores

5. Recurrencias

- Se los suele asociar en la bibliografía a
 - Estrategia D&C (top-down de diseño)
 - D-S/C-C (Divide-Solve/Conquer-Combine)
 - Cierto si es “igual problema-instancias menor”
- En todos los casos, se asocia a algoritmos básicamente recursivos
 - “Hacer lo mismo pero con menos cantidad”
- Ej: Factorial(n)

5. Recurrencias

- Ej: factorial

Factorial(n)

 if (n=0)
 return 1

 else

 return n * Factorial(n-1)

$$- t_{\text{fac}}(n) \begin{cases} 1 & n = 0 \\ t_{\text{fac}}(n-1) + 2 & n > 0 \end{cases}$$

5. Recurrencias

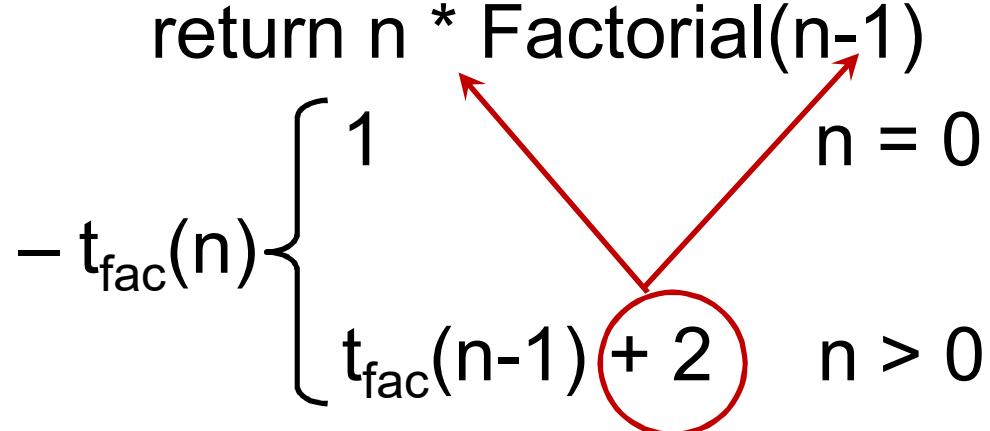
- Ej: factorial

Factorial(n)

 if (n=0)
 return 1

 else

 return n * Factorial(n-1)



5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
 - Evolución de llamadas...
 - $t(n) = t(n-1) + 2$ (1)
 - $t(n) = t(n-2) + 2 + 2$ (2)
 -
 - $t(n) = t(0) + 2 + \dots + 2$ (n veces "+ 2") (n)

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
 - Evolución de llamadas...
 - $t(n) = t(n-1) + 2$ (1)
 - $t(n) = t(n-2) + 2 + 2$ (2)
 -
 - $t(n) = t(0) + 2 + \dots + 2$ (n veces "+ 2") (n)
 - $t(n) = t(0) + 2 n$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?

– Evolución de llamadas...

$$\bullet \quad t(n) = t(n-1) + 2 \quad (1)$$

$$\bullet \quad t(n) = t(n-2) + 2 + 2 \quad (2)$$

• ...

...

$$\bullet \quad t(n) = t(0) + 2 + \dots + 2 \text{ (n veces "+ 2") } \quad (n)$$

$$\bullet \quad t(n) = t(0) + 2n$$

$$\bullet \quad t(n) = 2n + 1$$

$$t_{\text{fac}}(n) = \begin{cases} 1 & n = 0 \\ t_{\text{fac}}(n-1) + 2 & n > 0 \end{cases}$$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
 - Evolución de llamadas...
 - $t(n) = t(n-1) + 2$ (1)
 - $t(n) = t(n-2) + 2 + 2$ (2)
 -
 - $t(n) = t(0) + 2 + \dots + 2$ (n veces "+ 2") (n)
 - $t(n) = t(0) + 2n$
 - $t(n) = 2n + 1$
 - Demostrar (usualmente por inducción)

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
 - Demostrar por inducción, $t_{\text{fac}}(n) = 2n+1$
 - ¿n = 0? $t(0) = 2 \times 0 + 1 = 1$ que es la rec. $t(0) = 1$
 - Hi) $t(h) = 2 h + 1$
 - Dem) ¿ $t(h+1) = 2(h+1) + 1$?
 $t(h+1) = t(h) + 2$ Def. Recurrencia

$$t_{\text{fac}}(n) \begin{cases} 1 & n = 0 \\ t_{\text{fac}}(n-1) + 2 & n > 0 \end{cases}$$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
 - Demostrar por inducción, $t_{\text{fac}}(n) = 2n+1$
 - ¿n = 0? $t(0) = 2 \times 0 + 1 = 1$ que es la rec. $t(0) = 1$
 - Hi) $t(h) = 2 h + 1$
 - Dem) ¿ $t(h+1) = 2(h+1) + 1$?
$$\begin{aligned} t(h+1) &= t(h) + 2 && \text{Def. Recurrencia} \\ &= 2 h + 1 + 2 && \text{Hi} \end{aligned}$$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
 - Demostrar por inducción, $t_{\text{fac}}(n) = 2n+1$
 - ¿n = 0? $t(0) = 2 \times 0 + 1 = 1$ que es la rec. $t(0) = 1$
 - Hi) $t(h) = 2 h + 1$
 - Dem) ¿ $t(h+1) = 2(h+1) + 1$?
$$\begin{aligned} t(h+1) &= t(h) + 2 && \text{Def. Recurrencia} \\ &= 2 h + 1 + 2 && \text{Hi} \\ &= 2 (h+1) + 1 && \text{Lo que se quiere dem.} \end{aligned}$$

5. Recurrencias

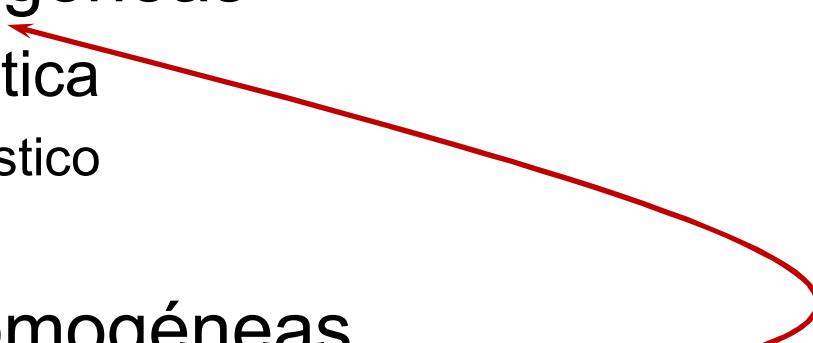
- Ej: factorial, ¿cómo analizarlo?
 - Demostrar por inducción, $t_{\text{fac}}(n) = 2n+1$
 - ¿n = 0? $t(0) = 2 \times 0 + 1 = 1$ que es la rec. $t(0) = 1$
 - Hi) $t(h) = 2 h + 1$
 - Dem) ¿ $t(h+1) = 2(h+1) + 1$?
$$\begin{aligned} t(h+1) &= t(h) + 2 && \text{Def. Recurrencia} \\ &= 2 h + 1 + 2 && \text{Hi} \\ &= 2 (h+1) + 1 && \text{Lo que se quiere dem.} \end{aligned}$$
 - Brassard-Bratley: “Intelligent guesswork”

5. Recurrencias

- Brassard-Bratley:
 - Intelligent guesswork
 - Cambio de variables
 - Recurrencias homogéneas
 - Ecuación característica
 - Polinomio característico
 - » ... → Solución

5. Recurrencias

- Brassard-Bratley:
 - Intelligent guesswork
 - Cambio de variables
 - Recurrencias homogéneas
 - Ecuación característica
 - Polinomio característico
 - » ... → Solución
 - Recurrencias no homogéneas
 - Recurrencias homogéneas



5. Recurrencias

- Brassard-Bratley:
 - Intelligent guesswork
 - Cambio de variables
 - Recurrencias homogéneas
 - Ecuación característica
 - Polinomio característico
 - » ... → Solución
 - Recurrencias no homogéneas
 - Recurrencias homogéneas
 - Solución → *Backtrack* (relación con las no homogéneas)

5. Recurrencias

- Dos “recetas”
 - $t(n) = a t(n-b) + f(n)$ (reduce restando)
 - $t(n) = a t(n/b) + f(n)$ (reduce dividiendo)
 - a: cantidad de llamadas recursivas
 - b: constante
 - f(n): operaciones “extra” llamadas recursivas

5. Recurrencias

- Recursión, $n \Rightarrow n-b$

$$- t(n) = \begin{cases} c & n \leq b \\ a t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

$$- t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

- Vale con todos $\Theta()$

5. Recurrencias

- Recursión, $n \Rightarrow n-b$:

Procedure Hanoi (n, i, j) // Traslada los n anillos más pequeños de i a j (1, 2, 3)

If $n > 0$ Then

Hanoi ($n-1$, i, $6-i-j$)

write “ $i \rightarrow j$ ”

Hanoi($n-1$, $6-i-j$, j)

5. Recurrencias

- Recursión, $n \Rightarrow n-b$:

Procedure Hanoi (n, i, j)

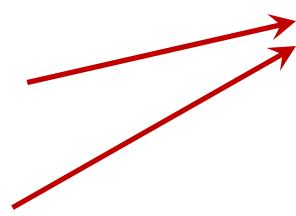
If $n > 0$ Then

Hanoi (n-1, i, 6-i-j)

write “ $i \rightarrow j$ ”

Hanoi(n-1, 6-i-j, j)

2 llamadas recursivas



5. Recurrencias

- Recursión, $n \Rightarrow n-b$:

Procedure Hanoi(n, i, j)

If $n > 0$ Then

Hanoi($n-1, i, 6-i-j$)

write " $i \rightarrow j$ "

Hanoi($n-1, 6-i-j, j$)

2 llamadas recursivas

Reduce en 1 el tamaño

5. Recurrencias

- Recursión, $n \Rightarrow n-b$:

Procedure Hanoi (n, i, j)

If $n > 0$ Then

Hanoi ($n-1$, i, $6-i-j$)

write “ $i \rightarrow j$ ”

Hanoi($n-1$, $6-i-j$, j)

2 llamadas recursivas
Reduce en 1 el tamaño
Trabajo extra constante

5. Recurrencias

- Recursión, $n \Rightarrow n-b$:

Procedure Hanoi (n, i, j)

```
If n > 0 Then  
    Hanoi (n-1, i, 6-i-j)  
    write "i → j"  
    Hanoi(n-1, 6-i-j, j)
```

2 llamadas recursivas
Reduce en 1 el tamaño
Trabajo extra constante

$$t(n) = \begin{cases} c & n \leq b \\ a t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

a = 2
b = 1
k = 0

$$t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

5. Recurrencias

- Recursión, $n \Rightarrow n-b$:

Procedure Hanoi (n, i, j)

```
If n > 0 Then  
    Hanoi (n-1, i, 6-i-j)  
    write "i → j"  
    Hanoi(n-1, 6-i-j, j)
```

$$t(n) = \begin{cases} c & n \leq b \\ a t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

$$t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

2 llamadas recursivas
Reduce en 1 el tamaño
Trabajo extra constante

$$\begin{aligned} a &= 2 \\ b &= 1 \\ k &= 0 \end{aligned}$$

$$\Rightarrow t(n) \in O(2^n)$$

5. Recurrencias

- Dos “recetas”
 - $t(n) = a t(n-b) + f(n)$ (reduce restando)
 - $t(n) = a t(n/b) + f(n)$ (reduce dividiendo)
 - a: cantidad de llamadas recursivas
 - b: constante
 - f(n): operaciones “extra” llamadas recursivas

5. Recurrencias

- Recursión, $n \Rightarrow n/b$

$$- t(n) = \begin{cases} c & n \leq b \\ a t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

$$- t(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$$

- Vale con todos $\Theta(\)$. Teorema Maestro o Método Maestro

5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k

5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

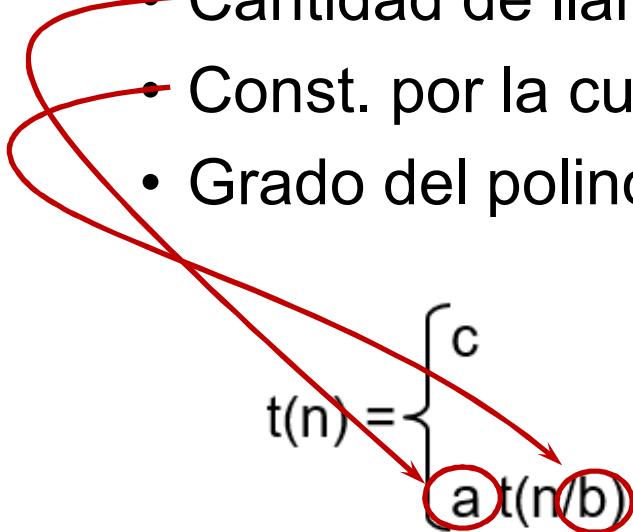
5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$


5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

The diagram shows the recurrence relation $t(n)$ defined by two cases. For $n \leq b$, the value is a constant c . For $n > b$, the value is $a t(n/b) + f(n)$, where $f(n)$ is part of a term enclosed in a red oval, indicating it is bounded by $O(n^k)$.

5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k
 - Aplicar la receta

5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Nuevamente, es necesario identificar
 - Cantidad de llamadas recursivas, a
 - Const. por la cual se divide el tamaño de la E/, b
 - Grado del polinomio del trabajo extra, k
 - Aplicar la receta teniendo en cuenta a, b y k

$$t(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$$

5. Recurrencias

- Recursión, $n \Rightarrow n/b$
 - Ej: bin_search
 - $a = 1$
 - $b = 2$
 - $k = 0$
 - Ej: merge_sort
 - $a = 2$
 - $b = 2$
 - $k = 1$

5. Recurrencias

- Dos “recetas”
 - $t(n) = a t(n-b) + f(n)$ (reduce restando)
 - $t(n) = a t(n/b) + f(n)$ (reduce dividiendo)
 - No todas las recurrencias “cubiertas”
 - Fibonacci recursiva
- FibRec(n)
- ```
if (n < 2)
 then return n
else return FibRec(n-1) + FibRec(n-2)
```

# 5. Recurrencias

- Sin “receta”: Fibonacci recursiva

FibRec(n)

if ( $n < 2$ )

    then return  $n$

else return FibRec( $n-1$ ) + FibRec( $n-2$ )

$$t_{\text{fib}} = \begin{cases} 1 & n < 2 \\ t_{\text{fib}}(n-1) + t_{\text{fib}}(n-2) + \text{cte} & n \geq 2 \end{cases}$$

Ninguna de las dos recetas es aplicable

# 5. Recurrencias

- Sin “receta”: Fibonacci recursiva
- Fibonacci iterativa

FibIter(n)

i  $\leftarrow$  i; j  $\leftarrow$  0

for k  $\leftarrow$  1 to n do

j  $\leftarrow$  i + j

i  $\leftarrow$  j - i

return j

# Análisis de Algoritmos

6. Estructuras de Datos

7. Diseño + Análisis

# Análisis de Algoritmos

- (1) Estructuras de control
- (2) Barómetro
- (3) Análisis del caso promedio
- (4) Análisis amortizado
- (5) Recurrencias } ¿? Un tipo específico de algoritmos
- (6) No veremos análisis asociados a algoritmos específicos *de* estructuras de datos
- (7) Diseño + análisis      {
  - Greedy
  - Divide-and-Conquer
  - Dynamic programming
  - Algoritmos probabilísticos

# 6. Estructuras de Datos

- Al menos una asignatura específica
- La mayoría/varios son recursivos
- En algunos casos: amortizado
  - Al construir un índice en una base de datos

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Divide and Conquer
  - Algoritmos Greedy
  - Programación dinámica
  - Algoritmos probabilísticos
- Muy específico de cada algoritmo más que del propio diseño

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Divide and Conquer
  - Algoritmos Greedy
  - Programación dinámica
  - Algoritmos probabilísticos
- Muy específico de cada algoritmo más que del propio diseño

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Divide and Conquer
    - En realidad: recursivos ==> recurrencias

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - Problemas de optimización, construyendo la solución paso a paso de manera iterativa
  - Decisiones en cada paso, dependiendo del estado de avance/solución
  - Se elige entre un conjunto de alternativas que se evalúan ==> lo “mejor”
  - Ejemplo del viajante de comercio

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



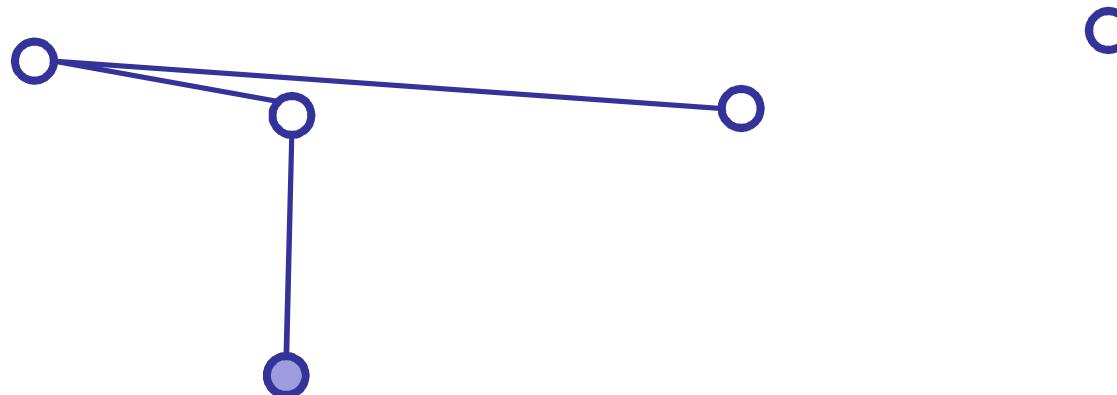
# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



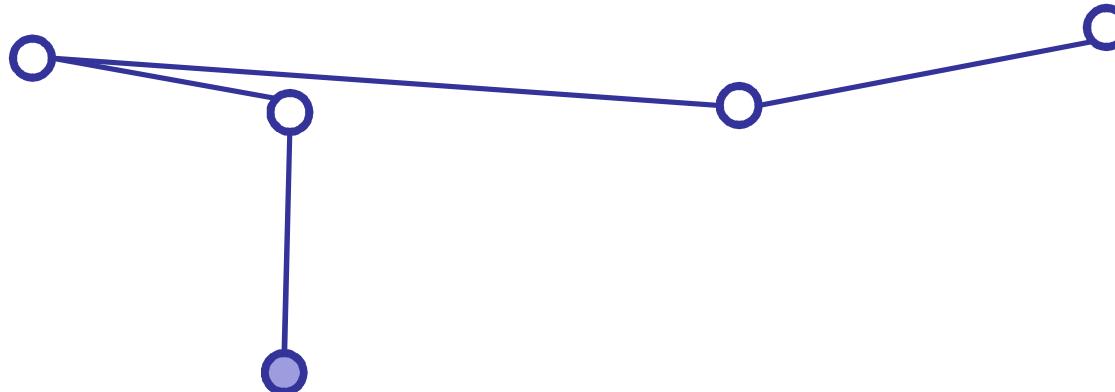
# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



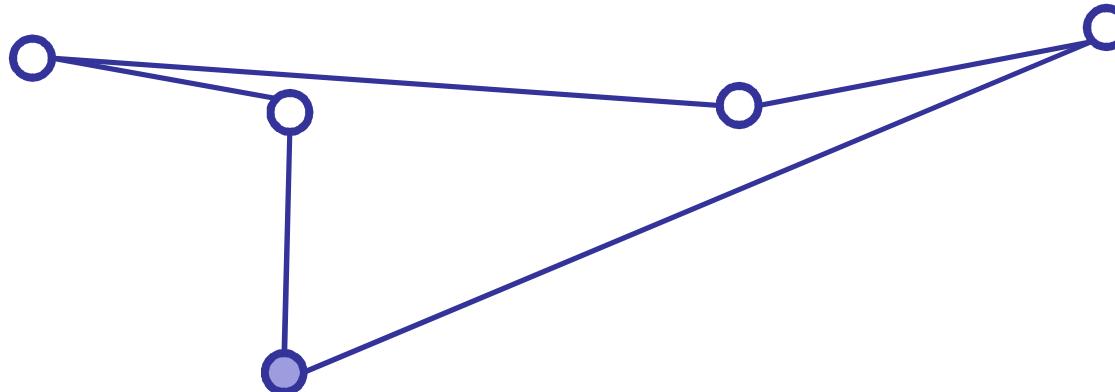
# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



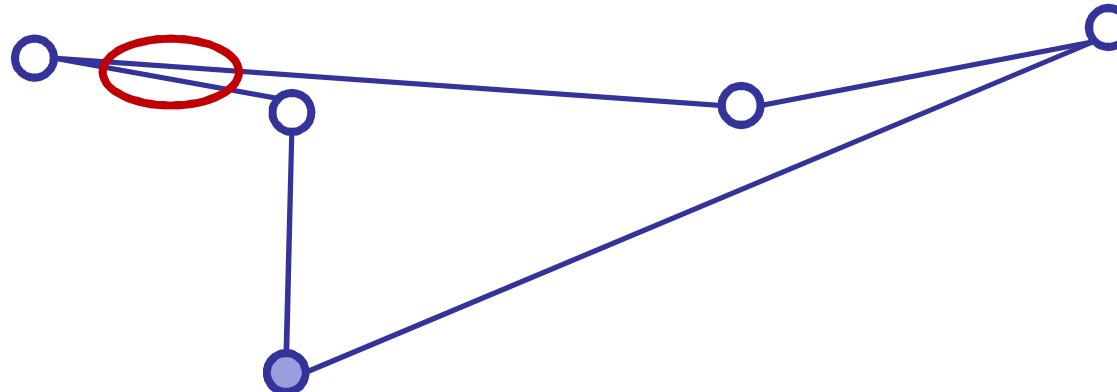
# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



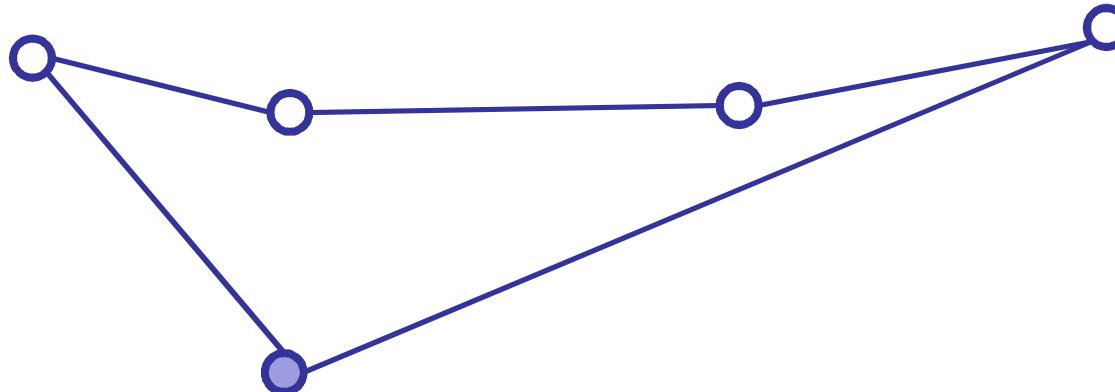
# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - ...
  - Ejemplo del viajante de comercio



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Programación dinámica
    - Estrategia bottom up (asociado a top-down, rec.)
    - Se comienza resolviendo las partes o problemas más sencillos posibles
    - Se reutilizan resultados intermedios (tablas)
    - Ej. fibonacci:  $f(n) = f(n-1) + f(n-2)$

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
    - Cuando se debe tomar una decisión, se toma al azar, no se computan costos p/ evaluar.
  - a) Numéricos: intervalo de confianza sobre la respuesta, ej: 90% de acierto para  $x \pm y$
  - b) Monte Carlo: respuesta exacta con alta probabilidad, pero puede ser errónea a veces
  - c) Las Vegas: respuesta exacta o sin resp.

# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular



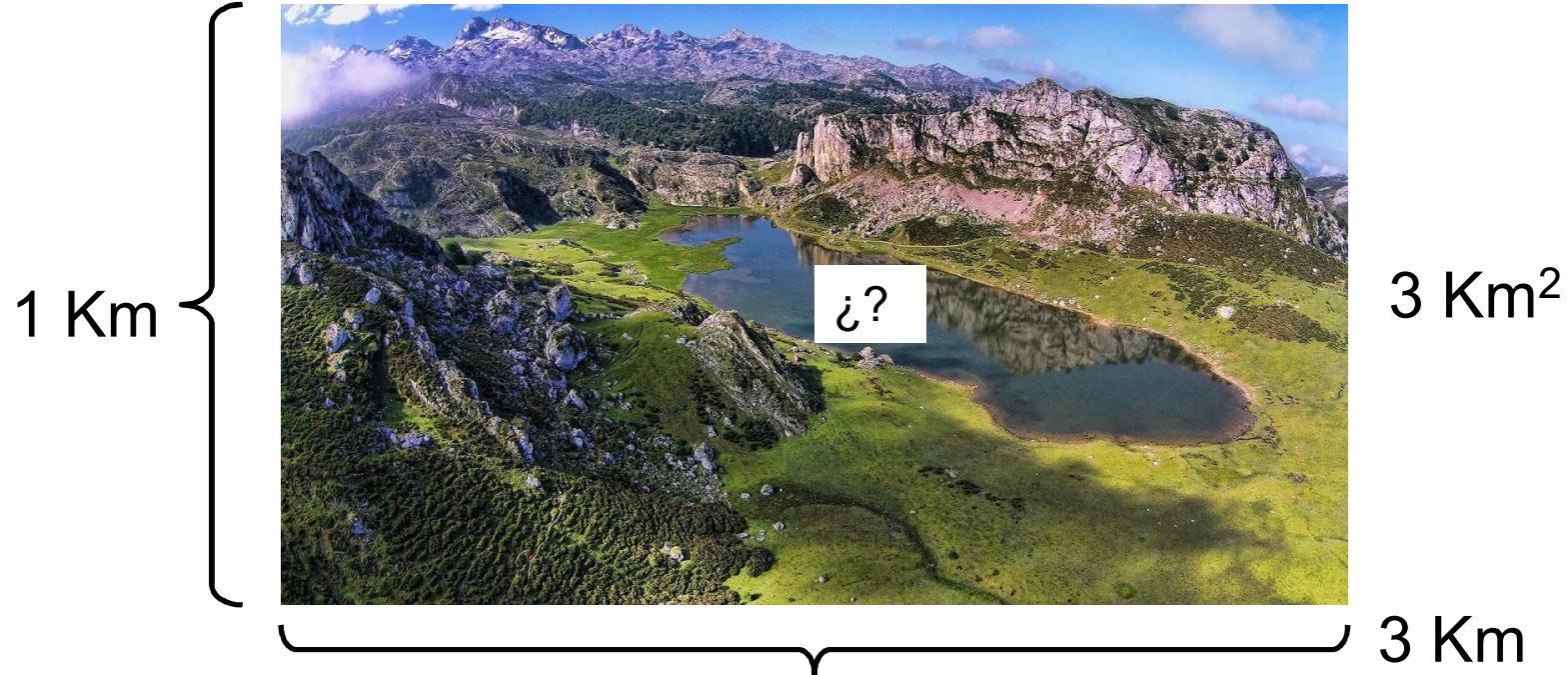
# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular

Posiciones aleatorias en el rectángulo

- Dentro del lago

- Fuera del lago

- #in vs. #out

-  $\#in / \#out \Rightarrow$  proporción del total ocupada por el lago

-  $\text{AreaLago} = (\#in / \#out) \times 3 \text{ km}^2$



# 7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular



Posiciones aleatorias en el rectángulo

- Dentro del lago
- Fuera del lago
- #in vs. #out
- $\#in / \#out \Rightarrow$  proporción del total ocupada por el lago
- $\text{AreaLago} = (\#in / \#out) \times 3 \text{ km}^2$
- ¿Error?  $\Rightarrow$  A mayor generación de aleatorios, menor error