Integración de Bases de Conocimiento

Clase 2 – Repaso de RBDs, teoría de modelos finitos, y complejidad descriptiva.

Profesores: Maria Vanina Martinez y Ricardo Rodriguez

Ejercicio sobre la clase anterior…

Simandan, D. (2010) Roads to perdition in the knowledge economy. Environment and Planning A, 42(7), pp.1519-1520.

Analizar el artículo y desarrollar las siguientes consignas:

- Describir (a grandes rasgos) como funcionaría un sistema de apoyo de tomas de decisiones para una aplicación que funcione bajo las circunstancias que explica el artículo.
- Enumere decisiones de diseño que ve necesarias para poder paliar las dificultades descriptas en el artículo.

En esta clase…

Cubriremos los siguientes temas:

- Una breve introducción a Bases de Datos (Relacionales)
- Calculo relacional: semántica de modelos
- Conceptos básicos de Complejidad Computacional
- Complejidad Computacional para Bases de Datos:
  - Tipos de complejidad
  - Resultado de imposibilidad de optimización perfecta de consultas: implicancias en bases de datos relacionales
  - Complejidad de lenguajes de consulta

En esta clase…

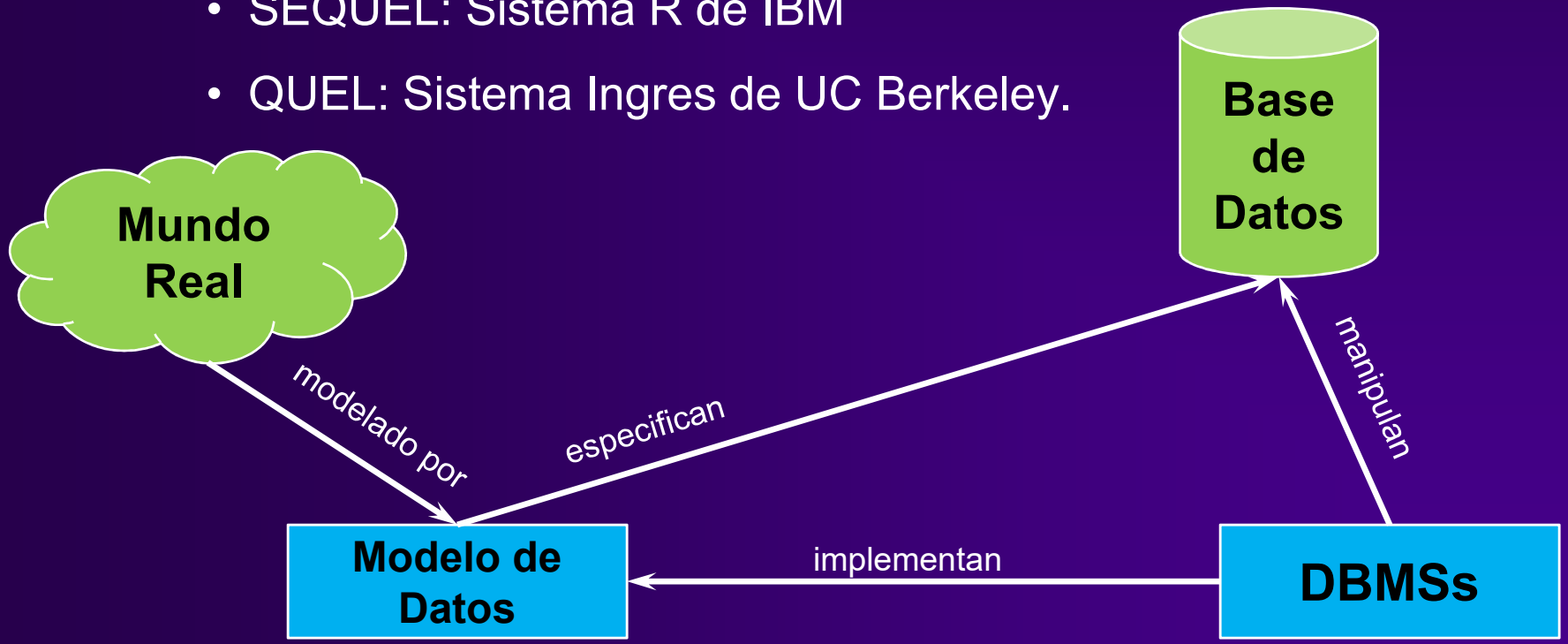
Cubriremos los siguientes temas:

- Una breve introducción a Bases de Datos (Relacionales)
- Calculo relacional: semántica de modelos
- Conceptos básicos de Complejidad Computacional
- Complejidad Computacional para Bases de Datos:
  - Tipos de complejidad
  - Resultado de imposibilidad de optimización perfecta de consultas: implicancias en bases de datos relacionales
  - Complejidad de lenguajes de consulta

Breve historia de las Bases de Datos

1970: Revolución relacional

- Modelo de Bases de Datos Relacionales (E. F. Codd): verdadera *independencia física* de los datos.
- Lenguajes de Consulta Relacionales (SQL)
  - SEQUEL: Sistema R de IBM
  - QUEL: Sistema Ingres de UC Berkeley.



Breve historia de las BDs relacionales

- 1980:
  - Optimización de consultas relacionales
  - Restricciones, teoría de dependencia
  - Datalog (consultas más poderosas que admiten recursión)
- 1990:
  - Nuevos modelos: BDs temporales, BDs OO y ORD
  - Data warehousing (almacén de datos)
  - Minería de datos
- Desde los 90's hasta ahora: revolución de Internet
  - Integración de datos en la Web (manejo de grandes volúmenes de datos)
  - XML, redes de sensores, P2P

Breve historia de las BDs relacionales

- 1980:
  - Optimización de consultas relacionales
  - Restricciones, teoría de dependencia
  - Datalog (consultas más poderosas que admiten recursión)
- 1990:
  - Nuevos modelos: BDs temporales, BDs OO y ORD
  - Data warehousing (almacén de datos)
  - Minería de datos
- Desde los 90's hasta ahora: revolución de Internet
  - Integración de datos en la Web (manejo de grandes volúmenes de datos)
  - XML, redes de sensores, P2P

Breve historia de las Bases de Datos

- Atraviesa muchas áreas en Ciencias de la Computación y Matemática:
  - Complejidad $\rightarrow$ eficiencia en la evaluación de consultas, optimización
  - Lógica, teoría finita de modelos $\rightarrow$ expresividad
  - Programación en lógica, satisfacción de restricciones (IA) $\rightarrow$ Datalog
  - Teoría de grafos $\rightarrow$ descomposición de hipergrafos
  - Teoría de Autómatas $\rightarrow$ modelo de consulta XML, procesamiento de flujo de datos (*data stream processing*)
- Se beneficia de otros campos y contribuye nuevos resultados.

Modelo de datos relacional y Lenguajes de consulta

Modelo de Datos Relacional

El modelo de datos relacional [Codd1970] consiste de:

- Esquemas:
  - *Esquema de bases de datos* es una colección (no vacía y finita) de *esquemas de relación* $\Rightarrow$ signatura y vocabulario
  - Cada *esquema de relación* consiste de un *nombre de relación* y un *conjunto finito de atributos* (columnas)
  - Cada atributo A tiene un dominio $dom(A)$ $\Rightarrow$ universo
- Lenguaje de consulta

Ejemplo: Base de datos de películas

Pelicula(Titulo:String, director:String, Actor:String)

titulo	director	actor

Teatro(t_nombre:String…….)

t_nombre	dirección	telefono

Cartelera(t_nombre:String,….)

t_nombre	Titulo	Hora

Modelo de Datos Relacional

Dado un esquema de base de datos $\mathcal{R}$, una instancia es:

- Una base de datos es una colección de *relaciones* (tablas)

  $\Rightarrow$ estructura.

- Para cada esquema relacional $r \in \mathcal{R}$ tenemos es un conjunto finito de *tuplas* t (filas) (también la llamamos relación r).

- Cada tupla t en r es un *mapeo* de los atributos de r en el dominio de r (unión de los dominios de todos los atributos en r) tal que $t(A) \in dom(A)$ para todos los atributos A de r.

Ejemplo: Base de datos de películas

pelicula

Titulo	Director	Actor
The Trouble with Harry	Hitchcock	Gwenn
The Trouble with Harry	Hitchcock	Forsythe
The Trouble with Harry	Hitchcock	MacLaine
The Trouble with Harry	Hitchcock	Hitchcock
…	…	…
Cries and Whispers	Bergman	Andersson
Cries and Whispers	Bergman	Sylwan
Cries and Whispers	Bergman	Thulin
Cries and Whispers	Bergman	Ullman

teatro

t_nombre	Dirección	Telefono
Gaumont Opera	31 bd. des Italiens	47 42 60 33
Saint Andre des Arts	30 rue Saint André des Arts	43 26 48 18
Le Champo	51 rue des Ecoles	43 54 51 60
…	…	…
Georges V	144 av. des Champs-Elysees	45 62 41 46
Les 7 Montparnassiens	98 bd. du Montparnasse	43 20 32 20

cartelera

t_nombre	Titulo	Hora
Gaumont Opera	Cries and Whispers	20:30
Saint Andre des Arts	The Trouble with Harry	20:15
…	…	…
Georges V	Cries and Whispers	22:15
Les 7 Montparnassiens	Cries and Whispers	20:45

Ejemplo: Base de datos de películas

Esquema:

- *pelicula(titulo,director,actor)*
- *teatro*(*t_ nombre,direccion,telefono*)
- *cartelera*(*t_ nombre*,*titulo*,*hora*)

Instancia:

{(*The Trouble with Harry*, *Hitchcock, Gwenn*), ...,
(*Cries and Whispers*, *Bergman*, *Ullman*)}

{(*Gaumont Opera*, 31 *bd. des Italiens*, 47 42 60 33),
..., (*Les* 7 *Montparnassiens,* 98 *bd. du Montparnasse,* 43 20 32 20)}

{(*Gaumont Opera*, *Cries and Whispers,* 20:30),
..., (*Les* 7 *Montparnassiens, Cries and Whispers*, 20:45)}

Lenguajes de consulta relacionales

Lenguajes formales con sintaxis y semántica:

- *Sintaxis*: formalismo lógico o algebraico, o lenguaje de consulta específico (SQL); usa el vocabulario de esquema de BD.
- *Semántica*: un mapeo $\mathcal{M}[\![Q]\!]$ que transforma una (instancia) de bases de datos D en una (instancia) de base de datos $D_0 = \mathcal{M}[\![Q]\!](D)$.
- No consideraremos consultas que dependan de una representación particular de un dominio.
- Nos enfocaremos en *consultas genéricas*: producen resultados isomorfos en bases de datos isomorfas.

Lenguajes de consulta relacionales

Poder expresivo de un lenguaje de consulta L:

$$E(L) = \{M[Q] \mid Q \in L\}$$

- De esta manera podemos comparar lenguajes de consulta, por ejemplo:

  si $E(L) \subset E(L_0)$ entonces L_0 es estrictamente mas expresivo que L

- A veces escribimos $L = L_0$ en lugar de $E(L) = E(L_0)$.

Álgebra Relacional

- σ *selección* *
- π *proyección**
- $\times$ *producto cartesiano**
- $\bowtie$ *join*
- ρ *renombre**
- $-$ *diferencia**
- $\cup$ *unión**
- $\cap$ *intersección*

* Operaciones *primitivas*: el resto se puede reescribir como combinaciones de ellas.

Ejemplo: selección y proyección

R:

A	B
1	2
1	2
2	5
5	7
1	7
2	6

$\sigma_{A=1}(R)$

A	B
1	2
1	7
1	6

$\pi_{A}(R)$

A
1
2
5

Set Semantics

Ejemplo: selección y proyección

R:

A	B
1	2
1	2
2	5
5	7
1	7
2	6

$\sigma_{A=1}(R)$

A	B
1	2
1	7
1	6

$\pi_{A}(R)$

A
1
1
2
5
1
2

Bag Semantics

Ejemplo: Join

R:

A	B
1	2
1	2
2	5
5	6
1	7
1	6

S:

B	C
2	3
2	5
6	4
8	9

$R \bowtie S$:

A	B	C
1	2	3
1	2	5
5	6	4
1	6	4

$$R \bowtie S = \pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$$

Ejemplo: Renombre

S:

B	C
2	3
2	5
6	4
8	9

$\rho_{BC \to B'A}(R)$

B′	A
2	3
2	5
6	4
8	9

Ejemplo: Base de datos de películas

Esquema:

- *pelicula(titulo,director,actor)*
- *teatro(t_nombre,direccion,telefono)*
- *cartelera(t_nombre,titulo,hora)*

Instancia:

{(*The Trouble with Harry*, *Hitchcock, Gwenn*), ...,

(*Cries and Whispers*, *Bergman*, *Ullman*)}

{(*Gaumont Opera*, 31 *bd. des Italiens*, 47 42 60 33),

..., (*Les* 7 *Montparnassiens,* 98 *bd. du Montparnasse,* 43 20 32 20)}

{(*Gaumont Opera*, *Cries and Whispers,* 20:30),

..., (*Les* 7 *Montparnassiens, Cries and Whispers*, 20:45)}

Ejemplo: Base de datos de películas

pelicula

Titulo	Director	Actor
The Trouble with Harry	Hitchcock	Gwenn
The Trouble with Harry	Hitchcock	Forsythe
The Trouble with Harry	Hitchcock	MacLaine
The Trouble with Harry	Hitchcock	Hitchcock
…	…	…
Cries and Whispers	Bergman	Andersson
Cries and Whispers	Bergman	Sylwan
Cries and Whispers	Bergman	Thulin
Cries and Whispers	Bergman	Ullman

teatro

Teatro-nombre	Dirección	Telefono
Gaumont Opera	31 bd. des Italiens	47 42 60 33
Saint Andre des Arts	30 rue Saint André des Arts	43 26 48 18
Le Champo	51 rue des Ecoles	43 54 51 60
…	…	…
Georges V	144 av. des Champs-Elysees	45 62 41 46
Les 7 Montparnassiens	98 bd. du Montparnasse	43 20 32 20

cartelera

Teatro-nombre	Título	Hora
Gaumont Opera	Cries and Whispers	20:30
Saint Andre des Arts	The Trouble with Harry	20:15
…	…	…
Georges V	Cries and Whispers	22:15
Les 7 Montparnassiens	Cries and Whispers	20:45

Título de películas en las que Hitchcock actuó o dirigió:

$$\pi_{titulo}(\sigma_{director=Htchcock \vee actor=Hitchcock}(pelicula))$$

Fórmulas de primer orden (Cálculo relacional)

- Un *término* t es una constante o una variable.
- Fórmulas de primer orden se construyen inductivamente:
  - Cada *átomo* $P(t_1,\ldots,t_n)$ es una *fórmula*, donde P es un nombre de relación n-ario, y $t_1,\ldots,t_n$ son términos,
  - Igualdades $t_1 = t_2$ y desigualdades $t_1 \neq t_2$ son fórmulas, donde t_1 y t_2 son términos,
  - Si φ y ϕ son fórmulas, entonces también $(\varphi \vee \phi)$, $(\varphi \wedge \phi)$, y $\neg\varphi$
  - Si ϕ es una fórmula, y X es una variable, entonces $\exists X\, \phi$ y $\forall X\, \phi$ son fórmulas.

Fórmulas de primer orden (Cálculo relacional)

Titulo de películas en las que Hitchcock actuó o dirigió:

$$\pi_{titulo}(\sigma_{director=Htchcock \vee actor=Hitchcock}(pelicula))$$

Como consulta de *primer orden*:

$$\{T \mid \exists D,A\ peliculas(T,D,A) \wedge (A = \text{`}Hitchcock\text{'} \vee D = \text{`}Hitchcock\text{'})\}$$

En notación de *regla*:

$$\mathrm{R}(T) \leftarrow peliculas(T,D,\text{`}Hitchcock\text{'})$$

$$\mathrm{R}(T) \leftarrow peliculas(T,\text{`}Hitchcock\text{'},A)$$

FOL: semántica de modelos

- Una interpretación $I = (\Delta, A)$ consiste de:
  - Un conjunto no vacío Δ que es el dominio de I
  - Una función de interpretación A, que mapea
    - cada constante a a un elemento c^A en Δ
    - cada predicado P de aridad n a una relación n-aria

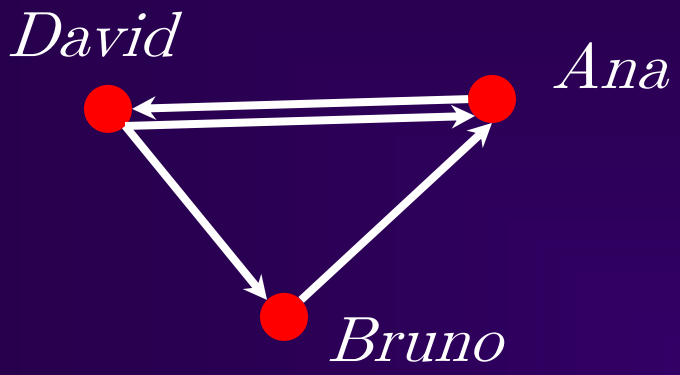
      $P^A: \Delta^n \rightarrow \{T, F\}$

FOL: semántica de modelos

Dada una interpretación $I = (\Delta, A)$ y una sentencia S:

- Si S es una formula atómica $S = P(t_1,\ldots,t_n)$, S es verdadera si y solo si $P^A({t_1}^A,\ldots,{t_n}^A) = \mathrm{T}$
- Si S es una formula atómica $S = t_1 = t_2$, S es verdadera si y solo si ${t_1}^A = {t_2}^A$
- Si S es una formula verdadera entonces $\neg S$ es falsa
- Si S y T son dos formulas, entonces $S \wedge T$ es verdadera sssi S y T son verdaderas; $S \vee T$ es verdadero sssi S es verdadera o T es verdadera
- Si $S = \forall XG$, S es verdadera si G es verdadero para todo $c \in \Delta$
- Si $S = \exists XG$, S es verdadera si G es verdadero para algún $c \in \Delta$

Ejemplo: Evaluación de Fórmulas



sigue	X	Y
	David	Ana
	David	Bruno
	Bruno	Ana
	Ana	David

$D=\{David, Bruno, Ana\}$

$\forall X \exists Y\ sigue(X,Y)$

$\forall X \exists Y \exists Z\ (sigue(Z,X) \wedge sigue(X,Y))$

$\exists X\ \forall Y \exists Z\ (sigue(Z,X) \wedge sigue(X,Y))$

$\forall X \exists Y \exists Z\ (Y \neq Z \wedge sigue(X,Y) \wedge sigue(X,Z))$

$\forall X \exists Y\ (sigue(X,Y) \wedge \exists Z\ (sigue(X,Z) \wedge Y \neq Z))$

FOL: semántica de modelos

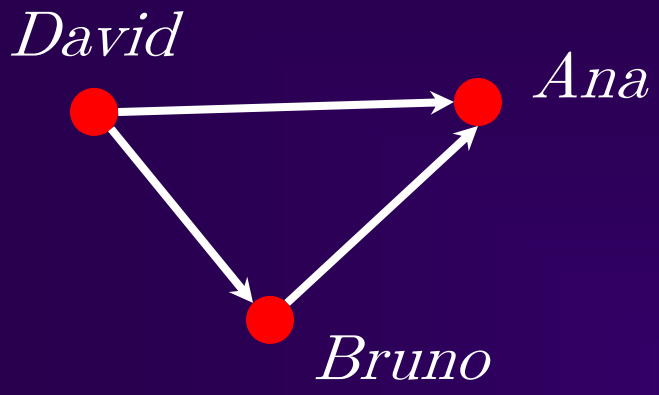
- Modelo: una interpretación I es un modelo para una sentencia S sssi S evalúa a verdadero bajo I, se denota $I \models S$.
- Validez: una formula S es valida sssi S evalúa a verdadero bajo todas las posibles interpretaciones, se denota $\models S$.
- Inconsistencia: una formula S es inconsistente si evalúa a falso bajo todas las posibles interpretaciones.

FOL Consultas Conjuntivas

Dado $\mathcal{R}$ un esquema relacional:

- Una consulta conjuntiva (CQ) sobre $\mathcal{R}$ tiene la forma $\mathcal{Q}(\mathrm{X}) = \exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$, Φ es una conjunción de átomos (no necesariamente *ground o básicas*).
- Una consulta conjuntiva Booleana (BCQ) sobre $\mathcal{R}$ tiene la forma $\mathcal{Q}() = \exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$, Φ es una conjunción de átomos.
- Note que X o bien son constantes o variables libres.

Ejemplo: Consultas Conjuntivas Booleanas



sigue	X	Y
	David	Ana
	David	Bruno
	Bruno	Ana

Consultas Conjuntiva:

$Q(\mathrm{X}) = \exists Y\, sigue(X, Y)$

$Q(\mathrm{X},\mathrm{Y}) = \exists Z\, (sigue(X, Y) \wedge sigue(Y, Z))$

Consultas Conjuntiva Booleana:

$Q() = \exists Y\, sigue(X, Y)$

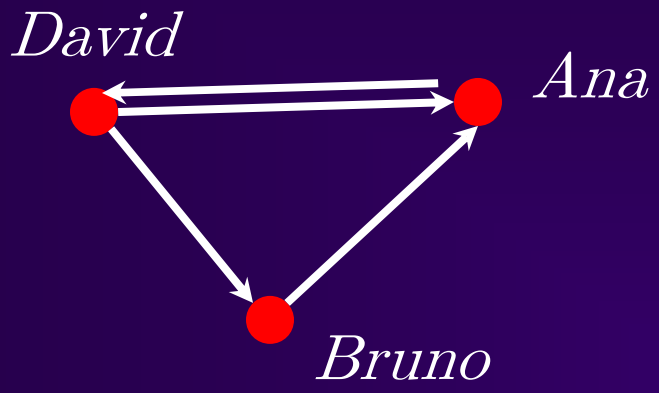
$Q() = \exists Z\, (sigue(david, Y) \wedge sigue(Y, ana))$

FOL Consultas Conjuntivas (semántica)

Dado un esquema de BD $\mathcal{R}$, una instancia D sobre $\mathcal{R}$ y una BCQ $Q() = \exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$:

- Si $\exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$ es una formula cerrada entonces Q *es verdadera sssi* $\exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$ *es verdadera en* D.
- Equivalente a: existe alguna manera de asignar valores del dominio a las variables en Y talque hagan verdadera la formula.

Ejemplo: Consultas Conjuntivas Booleanas



sigue	X	Y
	David	Ana
	David	Bruno
	Bruno	Ana

Consultas Conjuntiva Booleana Cerrada:

$$Q() = \exists Y\,(sigue(david, Y) \wedge sigue(Y, ana))$$

Es verdadera si y solo si existe un valor para Y tal que la formula (reemplazando ese valor) pertenezca a la relación.

FOL Consultas Conjuntivas (semántica)

Dado $\mathcal{R}$, una instancia D sobre $\mathcal{R}$ y una CQ $Q(\mathrm{X}) = \exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$ y no es una formula cerrada entonces:

- El conjunto de *respuestas* para Q en D es el conjunto de tuplas t sobre D tal que:
  - $\exists\ \mu$: $\mathrm{X} \cup \mathrm{Y} \rightarrow D$ t.q. $\mu(\Phi(\mathrm{X},\mathrm{Y})) \subseteq D$, y $\mu(\mathrm{X}) = t$

Ejemplo: $Q(\mathrm{X}) = \exists Y\ (sigue(X, Y) \wedge sigue(Y, X))$

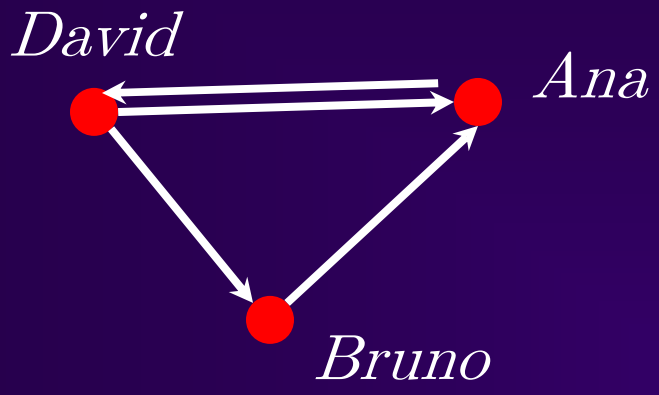
El resultado es una relación con todos los X que siguen a alguien que los sigue.

FOL Consultas Conjuntivas (semántica)

Dado $\mathcal{R}$, una instancia D sobre $\mathcal{R}$ y una CQ $Q(\mathrm{X}) = \exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$ y no es una formula cerrada entonces:

- El conjunto de *respuestas* para Q en D es el conjunto de tuplas t (es una relación!!) sobre D tal que:
  - $\exists\ \mu$: $\mathrm{X} \cup \mathrm{Y} \rightarrow D$ t.q. $\mu(\Phi(\mathrm{X},\mathrm{Y})) \subseteq D$, y $\mu(\mathrm{X}) = t$
  - Es el conjunto de valores para X que hacen verdadera la formula en D.
- Una BCQ $Q() = \exists \mathrm{Y}\ \Phi(\mathrm{X},\mathrm{Y})$ es verdadera sssi el conjunto de valores para X que hacen verdadera la formula en D es no vacío.

Ejemplo: Consultas Conjuntivas Booleanas



sigue	X	Y
	David	Ana
	David	Bruno
	Bruno	Ana

Consultas Conjuntiva:

$Q(\mathrm{X}) = \exists Y\, sigue(X, Y)$

$Q(\mathrm{X},\mathrm{Y}) = \exists Z\, (sigue(X, Y) \wedge sigue(Y, Z))$

Consultas Conjuntiva Booleana:

$Q() = \exists Y\, sigue(X, Y)$

$Q() = \exists Y\, (sigue(Y, ana) \wedge sigue(Y, david))$

Del Álgebra Relacional a Fórmulas de primer orden

Dado un esquema de relación $R(A,B)$

- La selección $\sigma_{A=3}(R)$ se traduce en:

$$R' = \{(X,\ Y) \mid R(X,Y) \wedge X = 3\}$$

(en notación de *regla*: $R'(\mathrm{X},\mathrm{Y}) \leftarrow \ R(\mathrm{X},\mathrm{Y}) \wedge \mathrm{X} = 3$)

- La proyección $\pi_A(R)$ se traduce a:

$$R' = \{X \mid \exists\, Y\ R(X,Y)\}$$

(en notación de *regla*: $R'(X) \leftarrow \ R(X,Y)$)

Del Álgebra Relacional a Fórmulas de primer orden

Dados dos esquemas de relación $R(A,B)$ y $S(C,D)$

- La selección $R \bowtie_{B=C} S$ se traduce en:

$$R' = \{(X,\ Y,\ Z) \mid R(X,Y) \wedge S(Y,Z)\}$$

en notación de *regla*: $R'(\mathrm{X},\mathrm{Y},\mathrm{Z}) \leftarrow R(\mathrm{X},\mathrm{Y}) \wedge S(Y,Z)$

o también $R'(\mathrm{X},\mathrm{Y},\mathrm{Z}) \leftarrow R(\mathrm{X},\mathrm{Y}) \wedge S(T,Z) \wedge Y{=}Z$

- Ejercicios:
  - ¿Cómo se traduce la unión y la diferencia?
  - Traducir a una consulta en álgebra relacional:

$$S(X,Y) \leftarrow \exists Z\,(R(X,Y) \wedge (P(Y,Z) \vee Q(Z,Y)))$$

Consultas *Unsafe*

- Con la interpretación estándar de lógica de primer orden existen consultas que no son *safe* (seguras), dependen del dominio, y no pueden ser representadas en AR.
- La consulta $R' = \{X \mid \forall Y\, R(X, Y)\}$ retorna:
  - con $R = \{\langle 1, 1\rangle\}$ y $\mathrm{dom} = \{1, 2\} \Rightarrow R' = \varnothing$, y
  - con $R = \{\langle 1, 1\rangle\}$ y $\mathrm{dom} = \{1\} \Rightarrow R' = \{1\}$
- El resultado de la consulta depende del *dominio*, ¡*no de los valores en la base de datos*!

Consultas *Unsafe*

- Otro ejemplo de consulta *unsafe* es la siguiente:
- Para la consulta $R' = \{X \mid \neg R(X)\}$ se obtiene:
  - con $R{=}\{\langle 1 \rangle\}$ y $\mathrm{dom} = \{1, 2\} \Rightarrow R' = \{2\}$, y
  - con $R{=}\{\langle 1 \rangle\}$ y $\mathrm{dom} = \{1\} \Rightarrow R' = \varnothing$
- Nuevamente, el resultado de la consulta depende del dominio.

Ejercicio: ¿Existen consultas Booleanas *unsafe*? Justifique.

Dominio activo

- La interpretación de *dominio activo* restringe las variables cuantificadas en la consulta al rango determinado por el dominio activo (los valores que efectivamente aparecen) en la consulta y en la instancia de base de datos.
- Bajo la semántica de dominio activo, la consulta $R' = \{X \mid \forall Y\, R(X, Y)\}$ retorna:

  Con $R = \{\langle 1, 1\rangle\}$ y $\mathrm{dom} = \{1, 2\}$ $(\mathrm{adom} = \{1\}) \Rightarrow R' = \{1\}$

Dominio activo

- Bajo la interpretación del *dominio activo*, cada fórmula en FOL es expresable en AR, y por lo tanto ambos lenguajes tienen el mismo poder expresivo:

$$AR = FOL$$ (*Codd 1972, AHV95 sección 5*).

- El resultado aplica también a consultas *safe*, o consultas independientes del dominio.
- Verificar que una consulta sea *safe* es un problema *no decidible* (lo veremos más adelante).

Restricciones de Integridad

- Las restricciones de integridad (ICs, integrity constraints en Inglés) establecen el significado pretendido de los datos de acuerdo al dominio de aplicación:
  - Restringen los modelos (lógicos) a sólo aquellos que "reflejan" la semántica de los datos.
- Son propiedades que se suponen deben ser respetadas por todas las instancias del esquema de base de datos.

Nota a futuro: La ICs tradicionales tienen una semántica estática es decir, la BD satisfice o no las restricciones, no proveen una manera de "arreglar" las violaciones.

ICs en RBDs

Dependencias funcionales (FDs):

- Permiten expresar relaciones funcionales entre atributos: “Si existen dos tuplas con el mismo número de legajo, entonces el campo “Nombre” debe coincidir.

- Por ejemplo, consideremos una tabla conteniendo información sobre empleados de una empresa:

$$\mathrm{Emp}(\mathrm{NroLegajo}, \mathrm{Nombre}, \mathrm{Area}, \mathrm{Leg\_Sup}).$$

Esa dependencia funcional expresada formalmente en un lenguaje de bases de datos, se le asocia a la relación Emp:

$$\mathrm{Emp}[\mathrm{NroLegajo}] \rightarrow \mathrm{Emp}[\mathrm{Nombre}]$$

- ¿Qué dependencias funcionales conocen?

ICs en RBDs

Dependencias funcionales (FDs):

- Permiten expresar relaciones funcionales entre atributos: “Si existen dos tuplas con el mismo número de legajo, entonces el campo “Nombre” debe coincidir.

- Por ejemplo, consideremos una tabla conteniendo información sobre empleados de una empresa:

$$\mathrm{Emp}(\mathrm{NroLegajo}, \mathrm{Nombre}, \mathrm{Area}, \mathrm{Leg\_Sup}).$$

Esa dependencia funcional expresada formalmente en un lenguaje de bases de datos, se le asocia a la relación Emp:

$$\mathrm{Emp}[\mathrm{NroLegajo}] \rightarrow \mathrm{Emp}[\mathrm{Nombre}]$$

- Las claves primarias son un caso especial de FDs.

ICs en RBDs

Dependencias de inclusión:

- Permiten expresar la necesidad de la existencia de ciertas tuplas particulares: “Todo supervisor es empleado”.

Esa dependencia de inclusión expresada formalmente en un lenguaje de bases de datos:

$$\mathrm{Emp}[\mathrm{legSup}] \subseteq \mathrm{Emp}[\mathrm{nroLeg}]$$

Semántica: Si existe $\mathrm{Emp}(\mathrm{N1}, \mathrm{Nom1}, \mathrm{A1}, \mathrm{Sup1})$ entonces debe existir $\mathrm{Emp}(\mathrm{Sup1}, \mathrm{Nom2}, \mathrm{A2}, \mathrm{Sup2})$.

- ¿Qué dependencias de inclusión conocen?

ICs en RBDs

Dependencias de inclusión:

- Permiten expresar la necesidad de la existencia de ciertas tuplas particulares: “Todo supervisor es empleado”.

Esa dependencia de inclusión expresada formalmente en un lenguaje de bases de datos:

$$\mathrm{Emp}[\mathrm{legSup}] \subseteq \mathrm{Emp}[\mathrm{nroLeg}]$$

Semántica: Si existe $\mathrm{Emp}(\mathrm{N1}, \mathrm{Nom1}, \mathrm{A1}, \mathrm{Sup1})$ entonces debe existir $\mathrm{Emp}(\mathrm{Sup1}, \mathrm{Nom2}, \mathrm{A2}, \mathrm{Sup2})$.

- Las dependencias de inclusión permiten definir claves foráneas.

ICs en RBDs

Restricciones de negación (Denial Constraints):

- Permiten expresar que la existencia de ciertos registros (consideradas juntas) no es posible: “Ningún empleado puede ser supervisor de si mismo”

Esa dependencia expresada formalmente en un lenguaje lógico:

$$\mathrm{Emp}(\mathrm{N1}, \mathrm{Nom1}, \mathrm{A1}, \mathrm{Sup1}) \wedge \mathrm{N1} = \mathrm{Sup1} \rightarrow \bot$$

- Las dependencias funcionales pueden expresarse como restricciones de negación dependiendo de la expresividad del lenguaje.

ICs en RBDs

- Las ICs se agregaron para enriquecer semánticamente el modelo relacional.
- En un principio se estudiaron ICs expresadas en como sentencias arbitrarias de FOL.
- Por consideraciones de factibilidad se empezaron a estudiar clases mas restrictivas (y menos expresivas).
- Sin embargo, con el desarrollo de modelos mas abstractos y lenguajes de mas alto nivel se vuelven a considerar ICs más expresivas (OBDA).

Universal Constraints

- Un caso más general que vamos a usar mas adelante en la materia.
- Las restricciones universales son formulas de la forma:

$$\forall \mathbf{X}\ \neg[\mathrm{R}_1(\mathbf{X}_1) \wedge \ldots\ \wedge \mathrm{R}_n(\mathbf{X}_n) \wedge \neg \mathrm{P}_1\ (\mathbf{Y}_1) \wedge \ldots \wedge \neg \mathrm{P}_m(\mathbf{Y}_m) \wedge \Phi(\mathbf{X})]$$

donde $\Phi(\mathbf{X})$ no tiene cuantificadores y corresponde a predicados built-in y $\mathbf{Y}_1 \cup \ldots \cup \mathbf{Y}_m \subseteq \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_m = \mathbf{X}$

- Se lo suele escribir como:

$$\mathrm{R}_1(\mathbf{X}_1) \wedge \ldots\ \wedge \mathrm{R}_n(\mathbf{X}_n) \wedge \Phi(\mathbf{X}) \rightarrow \mathrm{P}_1\ (\mathbf{Y}_1) \wedge \ldots \wedge \mathrm{P}_m(\mathbf{Y}_m)$$

Nota a futuro…

- La ICs en RDBs tienen una semántica estática: la BD satisfice o no las ICs, no nos dicen como “arreglar” las violaciones.
- ¿Qué potenciales problemas puede tener esta decisión?
- Si el mundo fuera estático y nosotros diseñadores de BDs perfectos…no necesitamos más!!!
- Realidad: no somos perfectos y el mundo es dinámico y cada vez más necesitamos integrar conocimiento…
- Los datos están destinados a no satisfacer las restricciones… ¿Qué hacemos?

Complejidad computacional y

análisis de algoritmos:

Conceptos básicos

¿Qué es la complejidad computacional?

- Estudia la complejidad *intrínseca* de las tareas computacionales.
- Preguntas *absolutas*:
  - ¿Cuánto tiempo se necesita para llevar a cabo una tarea?
  - ¿Cuántos recursos se necesitan?
- Preguntas *relativas*:
  - ¿Es esta tarea más difícil que esta otra?
  - ¿Existen tareas "*más difíciles que ninguna otra*"?
- Existen muchas respuestas relativas, y pocas absolutas.

¿Qué es la complejidad computacional?

- Teoría de la *computabilidad*:
  - Se enfoca en diferentes modelos de "cómputo".
  - Explora su poder de expresión (expresibilidad).
  - Pregunta central: ¿*Qué se puede computar*?
- Teoría de la *complejidad*:
  - Intenta encontrar métricas significativas de *costo* computacional.
  - *Clasifica* problemas y los *relaciona* entre sí.
  - Busca *cotas* inferiores y superiores de costos de cómputo.
  - Pregunta central: ¿*Qué se puede computar eficientemente*?

Problemas como lenguajes formales

- Se utiliza un modelo de cómputo a *muy alto nivel*.
- Tenemos una "máquina" con entrada y salida:
  - $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ conjunto finito de símbolos
  - $w = \{w_1, \ldots, w_l\}$ es una cadena sobre Σ; $w_i \in \Sigma$
  - l es la longitud de w, también denotado por $|w|$
  - ε es la cadena vacía; $|\varepsilon| = 0$
  - Σ^k es el conjunto de cadenas de longitud k.
  - $\Sigma^* = \bigcup_{k \geq 0} \Sigma^k$ son todas las cadenas sobre Σ.
- Un *lenguaje* es un conjunto $L \subseteq \Sigma^*$; operaciones:
  - Unión, intersección, diferencia, complemento
  - Concatenación

Problemas como lenguajes formales

Ejemplo: NAT (los números naturales)

- Tenemos $\Sigma = \{0, ..., 9\}$
- Σ^* es el conjunto de todas las cadenas formadas por dígitos (incluyendo cosas como "003").
- Sea $[n]_{10}$ la representación decimal de $n \in \mathbb{N}$.
- $\mathrm{NAT} = \{[n]_{10} \mid n \in \mathbb{N}\}$
- $3 \in \mathrm{NAT}$, pero $003 \notin \mathrm{NAT}$
- Máquina para calcular el cuadrado de un número natural:
  - Entrada: $w = [n]_{10} \in \Sigma^*$
  - Salida: $v \in \Sigma^*$ tal que $v = [n^2]_{10}$

  La máquina debe manejar adecuadamente las $w \notin \mathrm{NAT}$

Problemas como lenguajes formales

Otro ejemplo: PRIMOS (los números primos)

- Tenemos $\Sigma = \{0, ..., 9\}$
- $\mathrm{PRIMOS} = \{[n]_{10} \mid n \text{ es un número primo}\}$
- Máquina para verificar primalidad:
  - Entrada: $w = [n]_{10} \in \Sigma^*$
  - Salida: 1 si n es primo, 0 si no lo es
- Esto es un ejemplo de un problema de decisión:
  - PRIMOS es el conjunto de instancias *positivas*.
  - $\Sigma^* - \mathrm{PRIMOS}$ (todas las demás cadenas) son las instancias *negativas*.
  - La máquina debe distinguir *ambos casos* ("*decide*" PRIMOS).

Modelo de cómputo

- Ya tenemos definido el formato de entrada y salida.
- El *modelo* de cómputo debe:
  - Definir lo que queremos decir con “cómputo”, “acciones”, etc.
  - Ser simple y fácil de utilizar, pero poderoso.
  - Hay muchas propuestas: Funciones recursivas parciales, Sistemas de reescritura (gramáticas), Máquinas de Turing.
- La *tesis de Turing-Church* nos dice que todos los problemas solubles pueden ser resueltos por cualquiera de estos formalismos.
- En general se utilizan las *Máquinas de Turing*.

Máquinas de Turing

- Las MT son modelos muy simplificados de computadoras:
  - Una *cinta* en la cual pueden leer y escribir:
    - Contiene *celdas*, con lugar para un símbolo cada una
    - Se utiliza para la entrada, salida, y almacenamiento temporario
  - Una *cabeza* lecto-escritora:
    - Puede cambiar el símbolo en la cinta en la posición actual
    - Se mueve de a una celda en cualquier dirección
  - Una máquina de *estados finitos*, con estados iniciales, finales y una función que dicta las transiciones.
- Parece simple, pero es un modelo muy poderoso.

Máquinas de Turing

Definición formal: Una MT es una 5-tupla $M = (Q, \Gamma, \delta, q_0, F)$:

- Q es un conjunto finito de *estados*;
- Γ es el *alfabeto* de cinta, incluyendo el símbolo nulo $\square$.
- q_0 es el estado *inicial*
- F es el conjunto de estados *finalizadores*
- δ es la función de *transición*:

$$\delta\colon (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{I, D, N\}$$

Inicio: En q_0, entrada w en cinta y la cabeza sobre el primer símbolo.

Pasos: Leer el símbolo a en la posición actual de la cabeza

Buscar $\delta(q, a) = (p, b, Dir)$

Cambiar al estado p, escribir b, mover la cabeza según Dir.

Se *detiene* en cuanto $q \in F$. La salida es el contenido de la cinta.

Máquinas de Turing

- Una *configuración* $(w,\ q,\ v)$ denota el estado actual del cómputo luego de cada paso:
  - La cinta contiene wv (con cantidad infinita de $\square$ a la izquierda y derecha)
  - La cabeza está sobre el primer símbolo de v.
  - La máquina está en estado q.
- Configuración inicial $(\varepsilon,\ q_0,\ w)$
- Configuración final: $(v,\ q,\ z)$, con $q \in F$
  - Salida z, denotada también $M(w)$
  - Si la máquina *no se detiene*, tenemos $M(w) = \nearrow$

Máquinas de Turing

- La relación de *transición entre configuraciones* formaliza la semántica de las MT.

$$(wa,\ q,\ bv) \vdash \quad (wac,\ p,\ v) \quad \text{si } \delta(q,\ b) = (p,\ c,\ R)$$

$$(wa,\ p,\ cv) \quad \text{si } \delta(q,\ b) = (p,\ c,\ N)$$

$$(w,\ p,\ acv) \quad \text{si } \delta(q,\ b) = (p,\ c,\ L)$$

- Notación:

  Desde α se alcanza β en un paso: $\alpha \vdash \beta$

  Desde α alcanza β en k pasos: $\alpha \vdash^k \beta$

  Desde α se alcanza β en alguna cantidad de pasos: $\alpha \vdash^* \beta$

MT aceptadoras vs. transductoras

La definición dada hasta ahora: la MT recibe la entrada y computa la salida:

- Esto recibe el nombre de máquina *transductora*.
- La MT es una función f: $\Sigma^* \rightarrow \Sigma^*$
- Éstas son las funciones *computables*:
  - Funciones *totales*: definidas para todas las posibles entradas.
  - Funciones *parciales*: hay entradas para las cuales no están definidas (para éstas, la MT no se detiene, o "*cicla*").

MT aceptadoras vs. transductoras

Muchas veces sólo estamos interesados en resolver un problema de *decisión* (la respuesta es sí/no).

- Las MT que resuelven estos problemas se llaman *aceptadoras*.
- La MT se detiene y acepta o rechaza; el contenido de la cinta no importa.
- La máquina M acepta el lenguaje $L(M)$ definido como:

$$L(M) := \{ w \in \Sigma^* \mid \exists y,z \in \Gamma^* : (\varepsilon,\ q_0,\ w) \vdash^* (y,\ q_{sí},\ z) \}$$

No determinismo

- Hasta ahora sólo hablamos del caso llamado *determinístico*: en cada momento hay un único próximo paso.
- MT *no determinística*:

$$\delta\colon (Q - F) \times \Gamma \to \mathcal{P}\big(Q \times \Gamma \times \{I, D, N\}\big)$$

- No hay algoritmo de elección del camino a seguir:
  - Se puede usar para modelar la incertidumbre
  - Surge un *árbol* de cómputo
  - Cada camino es un cómputo posible.
  - La MT acepta sólo si *existe un camino* aceptador.
- Es sólo un modelo teórico, ¡*no se puede implementar*!

Decidibilidad

- Un problema es *decidible* si existe una MT que siempre termina y acepta/rechaza adecuadamente las entradas.
  - El lenguaje PRIMOS es decidible (algoritmo sencillo).
  - El problema de la detención (dada una MT y una entrada, decidir si ésta se detiene) no es decidible.
- *Semi-decidibilidad*: si existe una MT que acepta sólo las cadenas del lenguaje, pero puede ciclar en las demás.
  - El problema de decidir la relación de consecuencia en lógica de primer orden es semi-decidible.

Reducciones

- Las *reducciones* son la herramienta para formalizar la relación que existe entre problemas:
  - Tenemos dos problemas, A y B
  - Supongamos que sabemos resolver B
  - Construimos un algoritmo que resuelve A usando el que tenemos (por hipótesis) para resolver B.
  - Se dice entonces que "*redujimos A a B*", y que A es "más fácil" que B.
- Existen variedades diferentes de reducciones, según las *restricciones* impuestas.

Reducciones

Reducción "*muchos a uno*":

$A \subseteq \Sigma^*$ es reducible "*muchos a uno*" (*many-one*) a $B \subseteq \Sigma^*$ (denotado $A \leq_m B$) si existe una función $f\colon \Sigma^* \to \Sigma^*$ (computable y total) tal que:

$$\forall w \in \Sigma^* : w \in A \Leftrightarrow f(w) \in B$$

- La función f *mapea instancias* positivas a positivas, y negativas a negativas.
- Se puede utilizar el concepto para demostrar decidibilidad / indecidibilidad.

De computabilidad a complejidad

- ¿Qué pueden hacer los algoritmos con *recursos limitados*?
  - Recursos: tiempo de ejecución y memoria.
  - Aún sabiendo que la MT se va a detener, *¿cuántos pasos necesita, y cuántas celdas de la cinta usa*?
- Más que una cuantificación exacta de estas métricas, nos interesa saber su comportamiento *asintótico*.
- Para esto usamos los símbolos de Landau; en general, el más utilizado es $O(.)$ ("*big*-O", en inglés):

  Dada $g\colon \mathbb{N} \to \mathbb{N}$, $O(g)$ denota el *conjunto* de todas las funciones $f\colon \mathbb{N} \to \mathbb{N}$ tales que existen n_0 y c tales que

  $$\forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

Símbolos de Landau

- Ejemplos:
  - $n \cdot log(n) \in O(n^2)$
  - $3 \cdot n^3 + 4 \in O(n^3)$
  - $n^c \in O(2^n)$ para cualquier constante c.
  - $O(1)$ son las funciones acotadas por una constante.
- Existen otros símbolos:
  - Cotas inferiores: $\Omega(.)$
  - Cotas superiores/inferiores estrictas: $\omega(.)$, $o(.)$
  - Crecimiento “equivalente”: $\Theta(.)$

Métricas sobre recursos

- *Tiempo de ejecución* de una MT M con entrada w:

$$\text{time}_M(w) := \max\{t \geq 0 \mid \exists y, z \in \Gamma^*, q \in F : (w, q_0, \varepsilon) \vdash^t (y, q, z)\}$$

- Análogamente, para *espacio* tenemos:

$$\text{space}_M(w) := \max\{n \geq 0 \mid M \textit{ utiliza } n \textit{ celdas en la cinta}\}$$

- Ahora podemos definir algunos conjuntos interesantes:
  - Dada $t: \mathbb{N} \to \mathbb{N}$, si para todas las entradas vale que $\text{time}_M(w) \leq t(|w|)$, entonces M es $t(n)$-acotada en tiempo. Además:

$$\text{DTIME}(t(n)) := \{L(M) \mid M \text{ es } t(n)\textit{-acotada en tiempo}\}$$

  - Dada $s: \mathbb{N} \to \mathbb{N}$, si para todas las entradas vale que $\text{space}_M(w) \leq s(|w|)$, entonces M es $s(n)$-acotada en espacio. Además:

$$\text{DSPACE}(s(n)) := \{L(M) \mid M \text{ es } s(n)\textit{-acotada en espacio}\}$$

Métricas sobre recursos

- Para MT *no determinísticas*, tenemos $\mathrm{NTIME}(t(n))$ y $\mathrm{NSPACE}(s(n))$:
- Las cotas de tiempo y espacio deben valer para *cualquier camino* en el árbol.
- Aclaración: $\mathrm{space}_M(w)$ sólo se computa sobre "cintas de trabajo":
  - Agregar cintas a las MT no afecta su poder de cómputo.
  - Las cintas de entrada y de salida *no cuentan*; esto permite cosas como espacio sub-lineal (por ejemplo, $\log(|w|)$).

Clases de complejidad

Tiempo determinístico:

- Tiempo lineal:

$$\mathrm{LINTIME} := \bigcup_{c \geq 1} \mathrm{DTIME}(cn + c) = \mathrm{DTIME}(O(n))$$

- Tiempo polinomial:

$$\mathrm{P} := \bigcup_{c \geq 1} \mathrm{DTIME}(n^c + c) = \mathrm{DTIME}(n^{O(1)})$$

- Funciones de tiempo polinomial:

$$\mathrm{FP} := \bigcup_{c \geq 1} \mathrm{FTIME}(n^c + c) = \mathrm{FTIME}(n^{O(1)})$$

- Tiempo exponencial:

$$\mathrm{EXP} := \bigcup_{c \geq 1} \mathrm{DTIME}\left(2^{n^c + c}\right) = \mathrm{DTIME}\left(2^{n^{O(1)}}\right)$$

Clases de complejidad

Espacio determinístico:

- Espacio logarítmico:

$$\mathrm{L} := \mathrm{DSPACE}\big(O(log(n))\big)$$

- Espacio polinomial:

$$\mathrm{PSPACE} := \mathrm{DSPACE}(n^{O(1)})$$

- Espacio exponencial:

$$\mathrm{EXPSPACE} := \mathrm{DSPACE}\big(2^{n^{O(1)}}\big)$$

Las clases no determinísticas se definen análogamente:

$\mathrm{NLINTIME}$, NP, NEXP, NL, $\mathrm{NPSPACE}$, $\mathrm{NEXPSPACE}$

Crecimiento asintótico

Supongamos que tenemos una computadora capaz de ejecutar 1.000 instrucciones cada microsegundo (10^9 *instrucciones/segundo*):

/*Entrada*/ (n)	$\log(n)$	n	$n \log(n)$	n^2	2^n	$n!$
10	0,003 µs	0,001 µs	0,033 µs	0,1 µs	1 ms	3.63 ms
20	0,004µs	0,02 µs	0,086 µs	0,4 µs	1 s	77.1 años
30	0,005 µs	0,03 µs	0,147 µs	0,9 µs	16.3 min	$8{,}4\times10^{15}$ años
40	0,005 µs	0,04 µs	0,213 µs	1,6 µs	13 días	
50	0,006 µs	0,05 µs	0,282 µs	2,5 µs	4×10^{13} años	
100	0,007 µs	0,1 µs	0,644 µs	10 µs		
1.000	0,010 µs	1 µs	9,966 µs	1 ms		
10.000	0,013 µs	10 µs	130 µs	100 ms		
100.000	0,017 µs	0,1 ms	1,67 ms	10 s		
1.000.000	0,020 µs	1 ms	19.93 ms	16,7 min		
10.000.000	0,023 µs	0.01 s	0,23 s	1,16 días		
100.000.000	0,027 µs	0.1 s	2,66 s	115,7 días		
1.000.000.000	0,030 µs	1 s	29,9 s	31.7 años		

¿Y si me compro una supercomputadora?

- Para poner los números de la tabla anterior en contexto, la computadora más rápida de la actualidad es la *Tianhe-2*:
  - Costo: USD 390.000.000
  - 3.120.000 núcleos en 16.000 nodos (!)
  - Cada nodo tiene 88GiB de RAM (1.34 PiB en total)
  - Consume 24MW (alcanza para más de 4.000 hogares)
  - 33.86 PFLOPs (1 PFLOP = 10^{15} FLOPs)
- Aun en el caso de que 1 instrucción = 1 FLOP (*no es así*), la diferencia con la máquina de la tabla es un factor de 10^6.
- ¡OJO! Los 33.86 PFLOPs los logra con un *paralelismo masivo*...

Relaciones entre clases

- De las definiciones se desprende que:

  $$\mathrm{LINTIME} \subseteq \mathrm{P} \subseteq \mathrm{EXP}$$

  $$\mathrm{NLINTIME} \subseteq \mathrm{NP} \subseteq \mathrm{NEXP}$$

- Aquí sólo veremos que están *incluidas*; un resultado de *separación* sería: ¿existe un $L \in \mathrm{P} - \mathrm{LINTIME}$?
- Si bien en general se cree que $\mathrm{P} \neq \mathrm{NP}$, no se ha probado.
- Sí sabemos que $\mathrm{PSPACE} = \mathrm{NPSPACE}$.
- El complemento de una clase C se define:

  $$\text{co-C} := \{\bar{L} \mid L \in \mathrm{C}\}$$

- En clases *determinísticas*, tenemos $\mathrm{C} = \text{co-C}$ (*¿por qué?*)

Otra caracterización de NP

- Una relación $R \subseteq \Sigma^* \times \Sigma^*$ se dice *polinomialmente acotada* si existe un polinomio $p(n)$ tal que:

$$\forall (x, y) \in R : |y| \leq p(|x|)$$

- NP es la clase de lenguajes L tales que existe una relación polinomialmente acotada $R_L \subseteq \Sigma^* \times \Sigma^*$ tal que:
  - $R_L \in \mathrm{P}$
  - $x \in L \Leftrightarrow \exists w : (x, w) \in R_L$

  Decimos que w es el "*testigo*" (*witness*) o "prueba" para $x \in L$, y R_L es la relación testigo.

Demostrar vs. verificar

- Para $L \in \mathrm{P}$:
  - La MT debe decidir membresía de x en tiempo polinomial.
  - Es decir, debe encontrar una prueba de que $x \in L$.
- Para $L \in \mathrm{NP}$:
  - Tenemos un testigo w asociado con x..
  - Queremos verificar la prueba para $x \in L$.
- En ambos casos hay una noción de *eficiencia*:
  - Para P, el tiempo de ejecución está acotado (*prueba* eficiente).
  - Para NP, el tamaño del testigo está acotado (*verificación* eficiente).

Demostrar vs. verificar

- co-NP es la clase de lenguajes L tales que existe una relación polinomialmente acotada $R_L \subseteq \Sigma^* \times \Sigma^*$ tal que:
  - $R_L \in \mathrm{P}$
  - $x \in L \Leftrightarrow \forall w : (x, w) \notin R_L$
- Ahora, la relación R_L es de "*prueba de no membresía*" :
  - Para cada instancia *negativa* x, hay un testigo w asociado.
  - La verificación de la prueba para $x \notin L$ es eficientemente verificable.

Reducciones acotadas

- Supongamos que tenemos dos problemas A y B:
  - ¿Cuál es la relación entre A y B en cuanto a "*dificultad*"?
  - Dado un tercer problema C, ¿es "más fácil" que A o B?
  - Tal vez lo parece, pero todavía no encontramos buenos algoritmos para A o B...
- Volvemos a las reducciones, con una *restricción* adicional a la función f.
- *Reducción polinomial* (o de Cook, $\leq_m^p$): igual que la definición de reducción muchos a uno, pero además $f \in \mathrm{FP}$.

Problemas C-hard y C-completos

- Se dice que un problema A es *C-hard* si:

$$\forall L \in \mathrm{C} : L \leq_m^p A$$

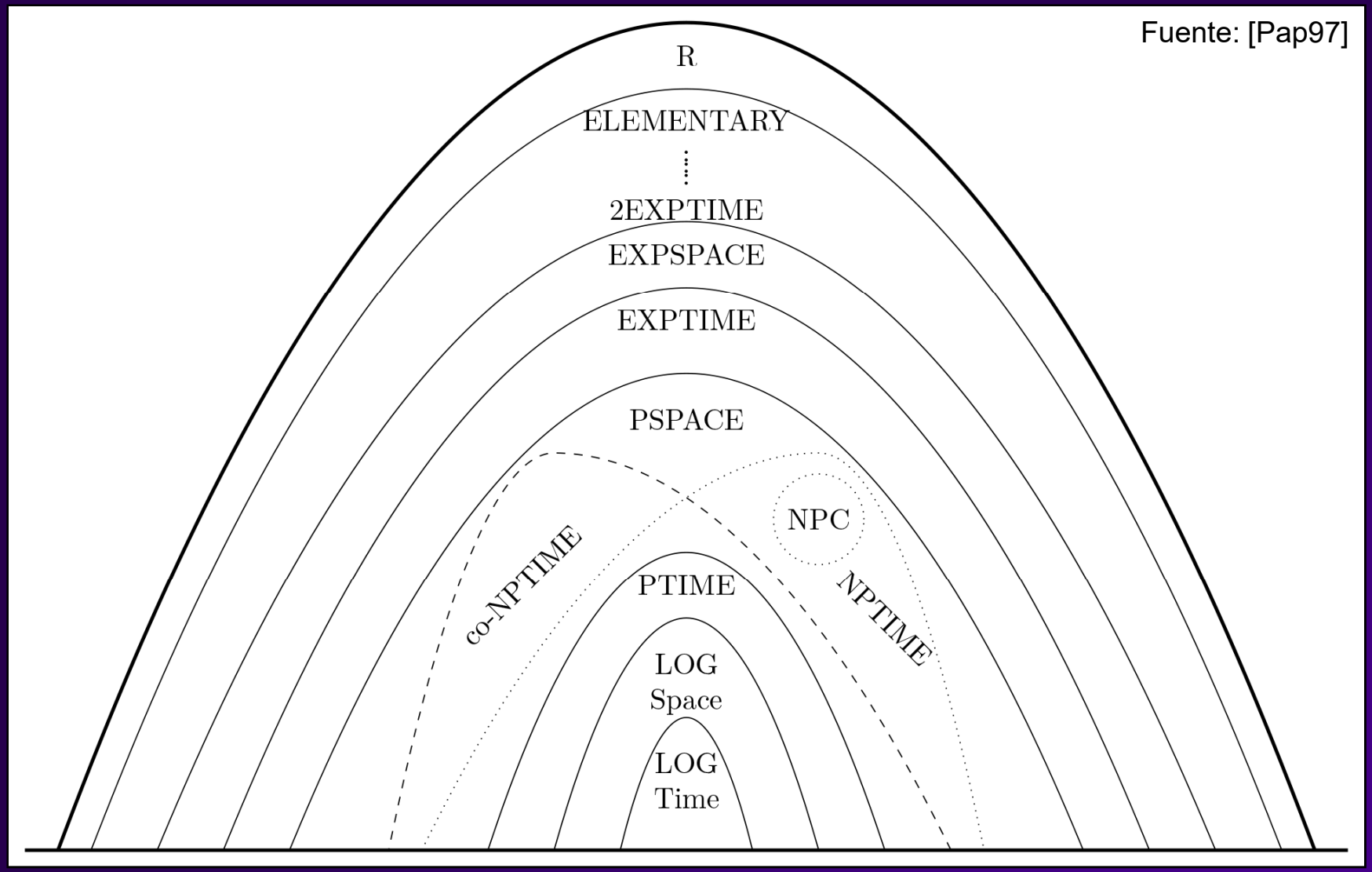
- Si además $A \in \mathrm{C}$, entonces A es *C-completo*.
- Consecuencia: dado un algoritmo eficiente para un problema C-completo, se tiene uno para todos los problemas en C.
- Teorema de Cook-Levin: El problema de decidir la satisfabilidad de fórmulas booleanas es NP-completo.
- Corolario: El problema de decidir *insatisfabilidad* de fórmulas booleanas es $\mathrm{co\text{-}NP}$-completo.

Mucho por investigar...

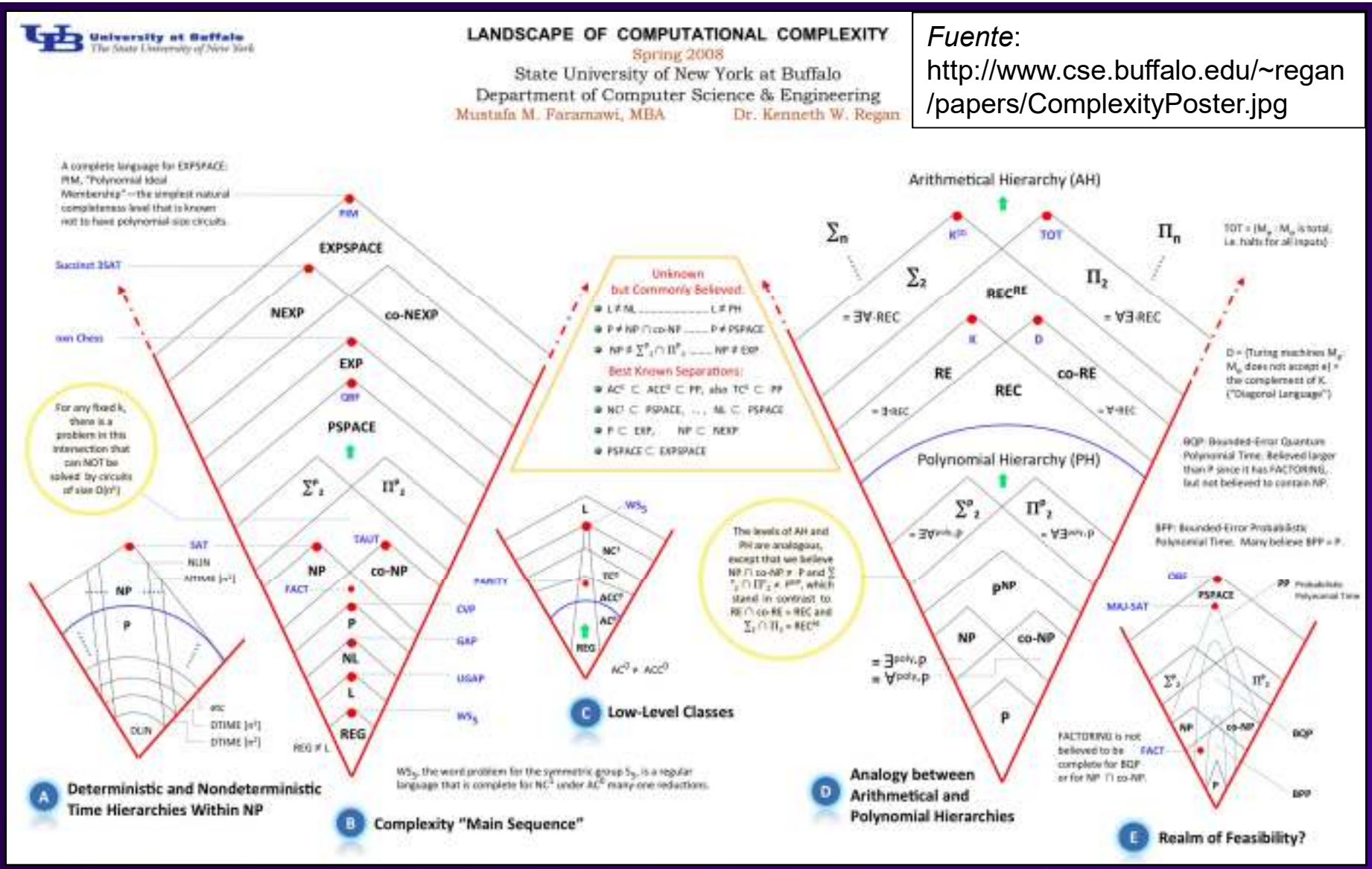
Algunos ejemplos más de cosas que *aun no se saben*:

- la relación entre NP y $\mathrm{co\text{-}NP}$
- la relación entre P y $\mathrm{NP} \cap \mathrm{co\text{-}NP}$
- la relación entre NP y PSPACE
- ... ni siquiera entre P y PSPACE
- La complejidad exacta del problema de *isomorfismo de grafos* (candidato a demostrar la relación entre P y NP...)

Mapa de clases



Mapa de clases (bis)



Fuente: http://www.cse.buffalo.edu/~regan/papers/ComplexityPoster.jpg

Complejidad de

Lenguajes de consulta

¿Qué medir?

- Como vimos anteriormente, las clases de complejidad en general se definen para problemas de *decisión* (si/no).
- Dado que las consultas pueden tener una *salida* grande, no sería justo contar su tamaño como "complejidad".
- Por ello, consideraremos los siguientes problemas de decisión, que son *computacionalmente equivalentes* para los propósitos de este curso (equivalencia LOGSPACE):
  - Existencia de BD que satisface: $D \models Q$?
  - Membresía ("*Query of Tuple*"): $t \in Q(D)$?
  - Consulta vacía: $Q(D) \neq \varnothing$?

Diferentes tipos de complejidad

Dependiendo de qué partes del problema se consideran *fijas*, tenemos diferentes tipos de complejidad:

- Combinada: *nada* se considera fijo.
- ba-combinada: la *aridad* de los símbolos relacionales se considera fija.
- Data: el *esquema* y la *consulta* se consideran fijos.
- Query: el *esquema* y la *base de datos* se consideran fijos.

Complejidad de consultas FO

- Teorema: La evaluación de consultas Booleanas en *FO* o *RA* tiene las siguientes complejidades:
  - PSPACE-completo en la complejidad combinada;
  - PSPACE-completo en la complejidad query;
  - en LOGSPACE en la complejidad data.
- Además, los mismos resultados valen para los problemas de *membresía* y *consulta vacía*.

QBF

Antes de ver las demostraciones de estos resultados, debemos introducir el problema de decidir la validez de una *Quantified Boolean Formula* (QBF):

$$Q_1 x_1 \; Q_2 \, x_2 \ldots \; Q_n \, x_n \; \phi(x_1, \, x_2, \, \ldots, \, x_n)$$

donde:

- los $Q_1 \in \{\exists, \forall\}$ son cuantificadores,
- ϕ es una fórmula en forma normal conjuntiva (CNF),
- los cuantificadores se alternan entre $\exists$ y $\forall$.

Este problema se llama también *QSAT*.

Ejemplos de QBFs

La QBF:

$$\exists x_1 \, \forall x_2 \, \exists x_3 \, (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge$$

$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$$

es *falsa*, mientras que la siguiente:

$$\exists x_1 \, \forall x_2 \, \exists x_3 \, (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3) \wedge$$

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

es *verdadera*.

QBF

QBF puede verse como un *juego* entre dos jugadores $\exists$ y $\forall$:

- Primero, el jugador $\exists$ elige un valor para x_1, luego el jugador $\forall$ elige uno para x_2, y así sucesivamente.
- Luego de que todos los valores fueron elegidos, el jugador $\exists$ *gana* si los valores constituyen una asignación que *satisface* a la fórmula ϕ.

Una QBF es *verdadera* si el jugador $\exists$ tiene una estrategia ganadora; es decir, si para cualquier elección del jugador $\forall$, el jugador $\exists$ puede jugar de tal manera de *asegurarse la victoria*.

Algoritmo para QBFs

Algoritmo $Verdad(\Phi)$

Si Φ no tiene cuantificadores, retornar SAT(Φ).

Sea $\Phi = Q_1 x_1\ Q_2\ x_2 \ldots\ Q_n\ x_n\ \phi(x_1, x_2, \ldots, x_n)$;

$b_0 := Verdad(Q_2\ x_2 \ldots\ Q_n\ x_n\ \phi(0, x_2, \ldots, x_n))$;

$b_1 := Verdad(Q_2\ x_2 \ldots\ Q_n\ x_n\ \phi(1, x_2, \ldots, x_n))$;

Si $Q_1 = \exists$, retornar $b_0 \vee b_1$

Si $Q_1 = \forall$, retornar $b_0 \wedge b_1$

QBF en PSPACE

Analizando el algoritmo anterior, podemos concluir rápidamente que el problema QBF está *en PSPACE*:

- La *profundidad* de la recursión es n.
- En cada paso, el tamaño de la *pila* de recursión es polinomial en n.
- Por lo tanto, alcanza con una cantidad de *espacio polinomial* en el tamaño de la entrada.

Veamos ahora que también es *PSPACE-completo*...

Algoritmo para consultas FO

Algoritmo $Eval(I, \phi)$

case

ϕ es $p(t_1, ..., t_k)$: retornar $p^I(t_1, ..., t_k)$;

ϕ es $\neg\psi$: retornar $\neg Eval(I, \psi)$;

ϕ es $\theta \wedge \psi$: retornar $Eval(I, \theta) \wedge Eval(I, \psi)$;

ϕ es $\exists x\, \psi$:

$B :=$ falso;

for $a \in dom$ do

$B := B \vee Eval(I, \psi_{x \to a})$;

retornar B.

Análisis del algoritmo *Eval*

Analizando el algoritmo $Eval$, y suponiendo que $n = |\mathcal{I}|$ y $m = |\phi|$, tenemos:

- *Profundidad* de la recursión: m.
- En cada paso de la recursión, debemos *almacenar*:
  - la *posición* de la variable siendo procesada: $\log\ m$, y
  - para cada variable con valor asignado en dom, la *posición* en dom: $m \log\ n$.
- Complejidad de *espacio*: $m\ (\log\ m + m \log\ n)$.

Complejidad: Cotas superiores

Complejidad de espacio: $m\ (\log\ m + m \log\ n)$:

- *Combinada* (donde tanto m como n son parte de la entrada):

  $m\ (\log\ m + m \log\ n) \Rightarrow$ en PSPACE

- *Data* (n es parte de la entrada, m es constante):

  $\log\ n \Rightarrow$ en LOGSPACE

- *Query* (m es parte de la entrada, n es constante):

  $m^2 \Rightarrow$ en PSPACE

Complejidad: Cotas inferiores

- Veamos cómo obtener los dos resultados *PSPACE-hard*.
- Reducción desde *QBF*, que es PSPACE-completo:
  - $\mathrm{dom} = \{0, 1\}$;
  - base de datos: $verdadero(1)$, $falso(0)$;
  - la entrada entonces se mapea directamente:

    $\exists x_1 \forall x_2 \exists x_3 \ (x_1 \vee \neg x_2 \vee x_3) \wedge \ldots$

    $\exists x_1 \forall x_2 \exists x_3 \ (verdadero(x_1) \vee falso(x_2) \vee verdadero(x_3)) \wedge \ldots$
- La *cota inferior* para la complejidad data corresponde a una clase llamada logtime-uniform AC^0, que veremos más adelante.

Algunos resultados centrales de la Teoría de Bases de Datos

Optimización de consultas

- Una pregunta que podemos plantearnos es:

  Dada una consulta Q en Álgebra Relacional, ¿existe alguna bases de datos D tal que $Q(D) \neq \varnothing$?

- Si la respuesta es *negativa*, entonces la consulta Q no tiene sentido, y podemos directamente reemplazarla por el conjunto vacío de tuplas.
- Esto podría ahorrar mucho tiempo de cómputo; claramente, también puede aplicarse a subconsultas.
- Lamentablemente, este problema es *indecidible*...

El Teorema de Trakhtenbrot (1950)

Teorema:

Para cada vocabulario relacional σ con al menos un símbolo relacional binario, el problema de determinar si una sentencia de *primer orden* Φ sobre σ es finitamente satisfacible es indecidible.

Traducido a la terminología de *bases de datos*, tenemos:

Teorema:

Dado un esquema de BD σ con al menos una relación binaria, el problema de determinar si una consulta Booleana Q de primer orden o en RA sobre σ es satisfecha por al menos una base de datos es indecidible.

Demostración (esquema)

La idea básica para demostrar este teorema es la siguiente:

- Definir una signatura relacional σ que permita *codificar* los cómputos finitos de una MT.
- Para una MT M y entrada I dadas, construir una *fórmula* FO $\Phi_{M,I}$ tal que:

M se detiene con la entrada I si y sólo si existe una estructura finita (es decir, una BD) D sobre σ tal que $D \models \Phi_{M,I}$.

Demostración (1)

- Construyamos la MT $M = (Q, \Sigma, \Gamma, \delta, q_{co}, q_{ac}, q_{re})$.
- Suposiciones *simplificadoras*:
  - σ puede tener relaciones unarias y binarias (siempre se puede codificar todo en una binaria).
  - Alfabeto de cinta $\Gamma = \Sigma = \{0, 1\}$.
  - Con este alfabeto se puede codificar todo, por ejemplo:
    - $0 \to 10$;
    - $1 \to 01$;
    - $\# \to 11$;
    - $\_ \to 00$.

Demostración (2)

- Más suposiciones:
  - La cabeza nunca se mueve *más allá* de la izquierda de la primera celda.
  - La máquina se detiene si entra en el estado q_{ac} o q_{re}, y *sólo* en estos estados.
- Estas condiciones se pueden asegurar mediante modificaciones simples que preservan la "*equivalencia de detención*".
- Incluso, se puede suponer que la entrada es *vacía*.

Demostración (3)

MT $M = (Q, \Sigma, \Gamma, \delta, q_{co}, q_{ac}, q_{re})$

Construyamos el *esquema* relacional:

$\Sigma = \{<, Min(.), T_0(.,.), T_1(.,.), H(.,.), S(.,.)\}$

Con los siguientes significados:

- Orden lineal $<$; escribimos $x < y$ en vez de $<(x,y)$. Los elementos se usan para simular *instantes* de tiempo y *celdas* en la cinta.
- $Min(x)$ es verdadero si y sólo si x es el elemento *mínimo* de $<$; nótese que podríamos haber usado una constante min.

Demostración (4)

MT $M = (Q, \Sigma, \Gamma, \delta, q_{co}, q_{ac}, q_{re})$

Construyamos el *esquema* relacional:

$\Sigma = \{<, Min(.), T_0(.,.), T_1(.,.), H(.,.), S(.,.)\}$

Con los siguientes significados:

- T_0 y T_1 son predicados de *cinta*: $T_0(p,t)$ y $T_1(p,t)$ indican que la celda número p contiene 0 y 1 al momento t, respectivamente.
- $H(p,t)$ indica que la *cabeza* está en la posición p al momento t.
- $S(s,t)$ indica que la MT está en el *estado* s al momento t.

Demostración (5)

Construimos ahora la *sentencia* $\Phi_{M,I}$, conjunción de:

- Una sentencia que afirma que $<$ es un *orden lineal* y que Min contiene su elemento *mínimo*; ésta se crea mediante la conjunción de:
  - Totalidad: $\forall x,y \left(x \neq y \rightarrow (x < y \vee y < x)\right)$;
  - Antisimetría y reflexividad: $\forall x,y \neg\left(x < y \wedge y < x\right)$;
  - Transitividad: $\forall x,y,z \left((x < y \wedge y < z) \rightarrow x < z\right)$;
  - Elemento mínimo: $\forall x,y \left(Min(x) \rightarrow (x = y \vee x < y)\right)$.

Demostración (6)

Construimos ahora la *sentencia* $\Phi_{M,I}$, conjunción de:

- Una sentencia de la forma:

  $$\exists s_0, s_1, \ldots, s_k \, (\Phi_{est} \wedge \Phi_{res}),$$

  donde las s_i son variables que representan el *estado* i de la MT M (es decir, suponemos que $|Q| = k+1$), y

  $$\Phi_{est} = \bigwedge_{i \neq j} s_i \neq s_j.$$

Por último, la sentencia Φ_{res} describe el *comportamiento* de la MT de la siguiente manera:

Demostración (7)

Φ_{res} es la *conjunción* de las siguientes sentencias:

- Una fórmula que define la *configuración inicial* de M con I en su cinta de entrada, la cual se forma mediante la conjunción de:
  - Suponiendo $|I| = n$, denotamos el i-ésimo *bit* con b_i. Entonces, para cada posición $0 \leq i < n$ tenemos:

    $$\forall p, t \left((Min(t) \wedge [p = i]) \rightarrow T_{b_i}(p, t) \right),$$

    donde $[p = i]$ es una abreviatura para la fórmula FO que afirma que p es el i-ésimo elemento de $<$.

    Esta fórmula entonces afirma que al instante 0 la cinta contiene la cadena de *entrada* I.

Demostración (8)

Φ_{res} es la *conjunción* de las siguientes sentencias:

- Una fórmula que define la *configuración inicial* de M con I en su cinta de entrada, la cual se forma mediante la conjunción de:
  - $\forall p,t\left(([p \geq n] \wedge Min(t)) \rightarrow T_0(p,t)\right)$

    El resto de las celdas contiene 0 al momento 0.
  - $\forall t\left(Min(t) \rightarrow H(t,t)\right)$

    La cabeza comienza en la posición 0.
  - $\forall t\left(Min(t) \rightarrow S(\mathrm{s}_0,t)\right)$

    La MT comienza en el estado s_0 (el inicial).

Demostración (9)

Φ_{res} es la *conjunción* de las siguientes sentencias:

- Una fórmula que afirma que en cada configuración, cada *celda* contiene exactamente un símbolo:

$$\forall p,t \left((T_0(p,t) \vee T_1(p,t)) \wedge \neg (T_0(p,t) \equiv \mathrm{T}_1(p,t)) \right).$$

- Una fórmula que afirma que la MT está en un *único estado* en cada momento dado:

$$\forall t \left((\bigvee_{1 \leq i \leq k} S(s_i, t)) \wedge \bigwedge_{i \neq j} \neg (S(s_i,t) \wedge S(s_j,t)) \right).$$

- En base a éstas, dejamos como ejercicio la sentencia que afirma que la cabeza está en una única posición.

Demostración (10)

- Faltan las fórmulas que describan las *transiciones* de estado. Tenemos una fórmula por cada tupla en δ.
- Por *ejemplo*, si una transición especifica que cuando la MT está en el estado s_4 y lee 0 entonces escribe 1, se mueve a la derecha y cambia al estado s_6, tenemos:

$$\forall p,t \Big(\big(H(p,t) \wedge T_0(p,t) \wedge S(s_4,t)\big) \rightarrow \exists p',t' \big(p' = p+1 \wedge t' = t+1 \wedge H(p',t') \wedge S(s_6,t') \wedge T_1(p,t') \wedge \forall r \neq p \, (T_0(r,r') \equiv T_0(r,t))\big)\Big)$$

Demostración (11)

- También debemos afirmar que M se *detiene* en algún momento cuando la entrada es I; suponiendo que tenemos $s_a = q_{ac}$ y $s_b = q_{re}$, tenemos:

$$\exists t \left(S(s_a, t) \vee S(s_b, t) \right).$$

- Con esto completamos la descripción de $\Phi_{M,I}$; dado que esta fórmula describe el precisamente el comportamiento de M ante la entrada I, entonces podemos concluir que:

> M se *detiene* con entrada I si y sólo si existe una base de datos D tal que $D \models \Phi_{M,I}$.

y, por lo tanto, el problema es *indecidible*.

Más resultados de indecidibilidad

Utilizando el Teorema de Trakhtenbrot, también podemos probar que los siguientes problemas son *indecidibles*:

- "*Safety*" de consultas FO
  (es decir, independencia del dominio).
- Equivalencia entre dos consultas en FO o RA.
- Verificar si una consulta está contenida en otra: $Q_1 \subseteq Q_2$
  (es decir, $\forall D\ Q_1(D) \subseteq Q_2(D)$).

Hacia SQL

- El lenguaje por excelencia para consultar (como así definir y modificar) bases de datos *relacionales* es SQL.
- Tiene aspectos extra-lógicos, tales como orden y posibilidad de tuplas duplicadas.
- Es fácil de ver que SQL es un superconjunto de RA:
  - Selección $\sigma_{A=B}(R)$:

```
SELECT * FROM R WHERE R.A = R.B
```

  - Proyección $\pi_{A}(R)$:

```
SELECT DISTINCT R.A FROM R
```

Hacia SQL

- Es fácil de ver que SQL es un superconjunto de RA (cont.):
  - Producto Cartesiano $R \times S$:

```
SELECT * FROM R, S
```

  - Renombre $\delta_{\text{AID, PID} \rightarrow \text{AID1, PID1}}(R)$:

```
SELECT AID AS AID1, PID AS PID1 FROM R
```

  - Diferencia $R - S$:

```
SELECT * FROM R EXCEPT SELECT * FROM S
```

  - Unión $R \cup S$:

```
SELECT * FROM R UNION SELECT * FROM S
```

Corolario

Como *corolario* de esta observación y el Teorema de Trakhtenbrot, tenemos:

> Corolario:
>
> Para un esquema de base de datos σ con al menos una relación binaria, el problema de decidir si una consulta *SQL* Q sobre σ tiene un resultado no vacío para al menos una base de datos es indecidible.

Por lo tanto, *no puede existir un algoritmo perfecto para optimizar consultas SQL*.

Referencias

[Pap94] C.H. Papadimitriou: "*Computational Complexity*". Addison-Wesley, 1994.

[John90] D.S. Johnson: "*A Catalog of Complexity Classes*". En J. van leeuwen, ed., Handbook of Theoretical Computes Science, A:2, pp. 67–161. MIT Press, 1990.

"Theory of Data and Knowledge Bases", dictado originalmente en TU Wien por Georg Gottlob y luego en University of Oxford por Georg Gottlob y Thomas Lukasiewicz.

M. Stigge: "*Introduction to Computational Complexity (Lecture Notes for a 5-day Graduate Course)*". Uppsala University, Suecia, julio de 2009.

[AHV95] S. Abiteboul, R. Hull, V. Vianu: "*Foundations of Databases*". Addison-Wesley, 1995.

[Lib04] L. Libkin: "*Elements of Finite Model Theory*". Springer, 2004.

Parte del contenido de este curso está basado en trabajo de investigación realizado en colaboración con Thomas Lukasiewicz, Georg Gottlob, V.S. Subrahmanian, Avigdor Gal, Andreas Pieris, Giorgio Orsi, Livia Predoiu y Oana Tifrea-Marciuska.