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…

Analizar el artículo y desarrollar las siguientes consignas:

– Describir (a grandes rasgos) como funcionaría un sistema de apoyo de tomas de decisiones en ese entorno.

– 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 las Bases de Datos (Relacionales)
- 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

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 $n = |I|$ 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:
    - $0 \rightarrow 10;$
    - $1 \rightarrow 01;$
    - $\# \rightarrow 11;$
    - $\_ \rightarrow 00.$

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:

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

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

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

- 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:

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

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:

$$\exists t \ \big(S(s_a, t) \lor S(s_b, t)\big).$$

- 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.