complexité

cours 5 : complexité et langages de requêtes pour bases de données

objectifs :

- étudier le coût d'évaluation des requêtes
- caractériser l'expressivité des langages de requêtes

plan

- rappels
  - structure du modèle relationnel
  - différents langages de requêtes
    - *SPC* : requêtes conjonctives
    - *CALC* : requêtes relationnelles
    - *fixpoint* : requêtes récursives
- requête et expressivité des langages
  - définition formelle de requête
  - classe de complexité pour langages de requêtes
- les résultats de complexité
  - pour les requêtes conjonctives
  - pour les requêtes relationnelles
  - pour les requêtes récursives
  - impact de l'ordre

rappels

petit retour sur le cours de L2...

- la structure du modèle relationnel
- les paradigmes de présentation
  - règle
  - algèbre
  - calcul
- les différents langages
  - *SPC* : requêtes conjonctives
  - *CALC* : requêtes relationnelles
  - *fixpoint* : requêtes récursives

exemple de requêtes

soit une base de données de schéma $\{G\}$

G est une relation binaire décrivant un graphe

- G est d'arité 2
- **dom**=$\{1,2,3,5,\ldots\}$ ensemble infini de constantes
- une instance de G est l'ensemble fini $\{\langle 1,2\rangle,\langle 3,3\rangle,\langle 2,5\rangle,\ldots\}$

conjonctive	quels sont les sommets de G qui sont leur propre successeur ?
relationnelle	quels sont les sommets de G qui ne sont pas leur propre successeur ?
récursive	existe-t-il un chemin de x à y ?

exemple de requêtes

- quels sont les sommets de G qui sont leur propre successeur ?
  - réponse$(x) \leftarrow G(x,x)$
  - $\{x | G(x,x)\}$
  - $\pi_1(\sigma_{1=2}(G))$
  - SELECT 1 FROM G WHERE 1=2;

exemple de requêtes

- quels sont les sommets de G qui sont leur propre successeur ?
  - réponse$(x) \leftarrow G(x,x)$
  - $\{x | G(x,x)\}$
  - $\pi_1(\sigma_{1=2}(G))$
  - SELECT 1 FROM G WHERE 1=2;
- quels sont les sommets de G qui ne sont pas leur propre successeur ?
  - $\pi_1(G) - \pi_1(\sigma_{1=2}(G))$
  - $\{x | \exists y G(x,y) \wedge \neg G(x,x)\}$

exemple de requêtes

- quels sont les sommets de G qui sont leur propre successeur ?
  - réponse$(x) \leftarrow G(x,x)$
  - $\{x | G(x,x)\}$
  - $\pi_1(\sigma_{1=2}(G))$
  - SELECT 1 FROM G WHERE 1=2;
- quels sont les sommets de G qui ne sont pas leur propre successeur ?
  - $\pi_1(G) - \pi_1(\sigma_{1=2}(G))$
  - $\{x | \exists y G(x,y) \wedge \neg G(x,x)\}$
- existe-t-il un chemin de x à y ?
  - $chemin(x,y) \leftarrow G(x,y)$
  - $chemin(x,y) \leftarrow chemin(x,z), G(z,y)$

intuitivement, une requête q est

une fonction

- entrée : une instance I sur un schéma de BD R
- sortie : une instance de relation sur un schéma de BD S

le schéma S de la requête ne dépend pas des données (seulement de l'expression de la requête)

la fonction est partielle car $q(I)$ peut ne pas être définie

exemple

soit

- la base de données de schéma $R = \{G\}$
- le schéma de G est G[départ,arrivée]
- la requête q =réponse$(x) \leftarrow G(x,x)$

le schéma de q est $S = \{$réponse$\}$

q calcule une instance de schéma S à partir d'une instance de schéma R

codage sur une MT

pour coder l'instance de départ :

soit $d = \{d_1, \ldots, d_n\}$ un sous-ensemble de **dom**

on utilise :

- une fonction $enc()$ donnant l'encodage binaire de i pour coder d_i
- les noms de relation du schéma R
- le nom de relation du schéma S

l'alphabet de codage est donc $=\{0,1,[,],\#\} \cup R \cup S$

exemple de codage

soit l'instance I suivante

P		Q	
a	b	c	c
b	a		

le codage de I est :

$$enc(I) = P[00\#01][01\#00]Q[10\#10]$$

requête calculable

une requête q sur R est calculable si il existe une MT M qui, sur chaque instance I de R

- si $q(I)$ n'est pas définie, M ne termine pas sur $enc(I)$
- si $q(I)$ est définie, M termine sur $enc(I)$ avec $enc(q(I))$ sur le ruban

généricité

la formulation et l'évaluation de requête

- devraient être indépendante de la manière dont les données sont codées (stockées)
- devraient prendre en compte uniquement les relations entre constantes, pas les constantes elles-mêmes

formellement : si ρ est une permutation des constantes du domaines, on dit qu'une requête q sur une instance I est générique si :

$$\rho(q(I)) = \rho(q)(\rho(I))$$

exemple de requête générique/non générique

soit r une relation binaire

exemple de requête générique :

- trouver les tuples de r dont la valeur du premier attribut est 1

exemple de requête non générique :

- trouver le tuple de r contenant comme valeur du premier attribut la première constante de **dom**

définition formelle d'une requête

une requête de schéma S sur une BD de schéma R est

- une fonction partielle
- des instances de schéma R
- vers les instances de schéma S
- qui est générique
- et calculable

comment mesurer la complexité ?

- complexité d'expression : complexité d'évaluer sur une instance fixe les différentes requêtes exprimables dans un langage donné
- complexité de données : complexité d'évaluer une requête fixe sur différentes instances d'une BD

mais la taille d'une requête $<<$ la taille d'une BD, donc on préfère la complexité de données

complexité de données

le problème se ramène à un problème de reconnaissance :

étant données

- q une requête,
- une instance I
- et un tuple u,

déterminer si $u \in q(I)$,

c'est à dire reconnaitre le langage suivant :

$$\{enc(I)\#enc(u) | u \in q(I)\}$$

c'est la taille de I qui sert de mesure de l'entrée

exemple

soit

- la requête $q = \{x | \forall y G(x,y)\}$
- sur l'instance $G = \{\langle a,b\rangle, \langle b,a\rangle, \langle b,b\rangle\}$
- et le tuple $\langle a\rangle$

mesurer la complexité de q revient à construire la MT reconnaissant :

$$\{G[0\#1][1\#0][1\#1]\#[0] | \langle a\rangle \in q(I)\}$$

mesurer la complexité de construction du résultat

complexité de construction du résultat :

- en général, se ramène à la complexité en données
- permet de distinguer les requêtes dont la réponse est volumineuse de celles dont la réponse est petite
- ne permet pas de distinguer la difficulté des requêtes dont les réponses ont le même ordre de grandeur

exemple

soit une relation binaire G décrivant un graphe

soit les requêtes :

- $cross(G) = \pi_1(G) \times \pi_2(G)$
- $path(G) = \{\langle x,y \rangle |$ il existe un chemin de x à y dans $G\}$
- $self(G) = G$

complexité en temps

pour la construction du résultat :

- de $cross = O(n^2)$
- et $path = O(n^2)$

pourtant *path* semble plus couteuse que *cross*

pour la reconnaissance :

- de $cross = O(n)$
- de $path = O(n^2)$

complexité en temps

pour la construction du résultat :

- de *cross* $= O(n^2)$
- de *self* $= O(n)$

pourtant, pour la reconnaissance :

- de *cross* $= O(n)$
- de *self* $= O(n)$

classe de complexité de requêtes

si C est une classe de complexité usuelle et L un langage de requêtes

- L est dans QC si toute requête de L a une complexité en données de C
- L exprime QC si chaque requête dans QC est exprimable par une requête de L
- L est complet pour C si
  - chaque requête de L est dans QC et
  - il existe une requête de L exprimant un problème C-complet

classe de complexité de requêtes

exemples :

- *QLOGSPACE* : classe des requêtes ayant une complexité de données dans *LOGSPACE*
- *QPTIME* :classe des requêtes ayant une complexité de données dans *PTIME*

exemple

soit une relation unaire G

considérons la requête $even(G) =$ vrai si $|G|$ est paire, faux sinon

- complexité de données :
  - linéaire en fonction de l'entrée
  - donc *even* est dans *QLOGSPACE*
- ne peut être exprimée ni avec *CALC* ni avec *fixpoint*

exemple

soit une relation unaire G

considérons la requête $even(G) =$ vrai si $|G|$ est paire, faux sinon

- complexité de données :
  - linéaire en fonction de l'entrée
  - donc *even* est dans *QLOGSPACE*
- ne peut être exprimée ni avec *CALC* ni avec *fixpoint*
  - la généricité impose que l'on ne puisse pas se servir du codage
  - on ne peut donc pas exprimer :

exemple

soit une relation unaire G

considérons la requête $even(G) =$ vrai si $|G|$ est paire, faux sinon

- complexité de données :
  - linéaire en fonction de l'entrée
  - donc *even* est dans *QLOGSPACE*
- ne peut être exprimée ni avec *CALC* ni avec *fixpoint*
  - la généricité impose que l'on ne puisse pas se servir du codage
  - on ne peut donc pas exprimer :
    - retirer un élément de G
    - compter le nombre d'éléments retirés

résultats de complexité

pour

- les requêtes conjonctives
- les requêtes relationnelles
- les requêtes point fixe

dans ce qui suit...

- q est une requête de schéma S sur une instance de schéma R
- I est une instance de schéma R
- u est un tuple de schéma S
- on décrit une MT
  - avec $enc(I)\#enc(u)$ sur le ruban d'entrée
  - qui termine dans un état d'acceptation si $u \in q(I)$

requêtes conjonctives sans quantifieur

exemple :

- requête : $q = \{x,y,z | r(x,y) \wedge s(y,z)\}$
- $I = \{r(a,b), r(a,c), \ldots\}$
- $u = \langle a,b,c \rangle$

requêtes conjonctives sans quantifieur

construisons la MT :

- u est connu, donc on utilise la valuation v telle que $u = v(t)$
- il faut parcourir $enc(I)$ pour trouver si chaque atome de $v(q) \in I$

requêtes conjonctives sans quantifieur

construisons la MT :

- u est connu, donc on utilise la valuation v telle que $u = v(t)$
- il faut parcourir $enc(I)$ pour trouver si chaque atome de $v(q) \in I$
- pour chaque tuple de I,
  - le comparer avec la partie de u qui s'y rapporte
  - on note sur le ruban :
    - la position de début du tuple et
    - la position courante dans le tuple

l'espace utilisé est inférieur au logarithme de l'espace utilisé pour coder l'entrée

donc ce langage est dans *QLOGSPACE*

requêtes conjonctives

ajoutons à ce langage le quantifieur $\exists$ (la projection)

- on obtient le langage *SPC*

forme générale : $q = \{x_1, \ldots, x_n | \exists y_1, \ldots, \exists y_m q'\}$ où

- q' est une requête sans quantifieur
- ayant pour variables libres $x_1, \ldots, x_n$

ce langage est dans *QLOGSPACE*

démonstration : par induction sur le nombre de quantifieurs $\exists$

requêtes conjonctives

supposons que :

- $q = \exists x q'$
- $q' \in QLOGSPACE$
- q' est une requête avec n quantifieurs $\exists$

toutes les valeurs possibles pour x doivent être testées

- si une valeur existe telle que q' est vraie alors la MT doit accepter,
- sinon la MT doit rejeter

requêtes conjonctives

les valeurs à utiliser pour x

- sont celles apparaissant sur le ruban d'entrée,
- sont prises dans l'ordre d'apparition sur le ruban d'entrée

il faut donc garder trace de la valeur courrante

- soit n_c le nombre de constantes dans I
- $n_c < |enc(I)\#enc(u)|$
- écrire la valeur demande donc de l'ordre de $log(n_c)$

requêtes conjonctives

évaluer la valeur de q' pour chaque valeur de x

utilise de l'ordre de $k \times log(|enc(I)\#enc(u)|)$ espace pour une constante k par hypothèse d'induction

donc le coût total en espace est de l'ordre de
$k + 1 \times log(|enc(I)\#enc(u)|)$

quod erat demonstrandum

requêtes relationnelles

ajoutons à *SPC* la négation et l'union

- on obtient le langage *CALC*

forme générale : $q = \{x_1, \ldots, x_n | \forall^{\overleftarrow{y}} \exists^{\overleftarrow{z}} q'\}$ où

- q' est une requête sans quantifieur
- ayant pour variables libres $x_1, \ldots, x_n$

CALC est dans *QLOGSPACE*

démonstration

- par induction sur le nombre de quantifieurs
- schéma de preuve similaire à celui utilisé pour SPC

requêtes point fixe

prenons *CALC* :

- retirons la négation
- ajoutons la possibilité d'exprimer la récursivité

nous obtenons *fixpoint* (Datalog)

forme générale : un ensemble de règles de la forme

$$R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$$

où

- les R_i sont des noms de relations
- les u_i sont des tuples libres
- chaque variable de u_1 apparaît au moins une fois dans $u_2, \ldots, u_n$

requêtes point fixe

fixpoint est dans *QPTIME*

principe : soit q une requête *fixpoint*, I une instance, u un tuple

1. écriture sur le ruban de la réponse $q(I)$
2. vérification que $u \in q(I)$

- l'étape 1 est polynomiale en fonction de $|enc(I)|$
- l'étape 2 est linéaire en fonction de $|enc(I)|$

requêtes point fixe

pour l'étape 1 :
au pire, il faut générer toutes les combinaisons $R_i(v(u))$ telles que R_i apparait dans la tête d'une règle et $v(u)$ est une valuation se servant uniquement des constantes de $adom(I)$

- $adom(I)$ est de l'ordre de $|enc(I)| = n$
- il y a p règles, avec p une constante
- pour chaque tête R_i l'arité est fixée, soit a la plus grande
- le nombre de combinaisons possibles est : n^a
- il y a de l'ordre de pn^a de combinaisons à générer
- générer une combinaison est en $O(n)$

requêtes point fixe

le fait de rajouter la négation ne change pas la complexité

mais ne permet toujours pas d'exprimer *even*

even est dans *QLOGSPACE*, donc dans *QPTIME* mais ni *CALC* ni *fixpoint* ne peut l'exprimer

impact de l'ordre

on pourrait exprimer *even* si les constantes étaient ordonnées :

- prendre les deux dernières constantes et les supprimer

ajoutons à une instance I une relation binaire *succ*

- si $adom(I) = \{a,b,c,d\}$ avec $a < b < c < d$
- l'instance de *succ* est $\{\langle a,b\rangle,\langle b,c\rangle,\langle c,d\rangle\}$

fixpoint sur une instance ordonnée exprime *QPTIME*

conclusion

encore un problème ouvert :

- existe-t-il un langage de requêtes ne tenant pas compte de l'ordre et exprimant *QPTIME* ?

conjecture : non

- car cela impliquerait que P=NP
- car il existe un tel langage pour *QNPTIME* !