Complexité des langages de requête

Cours L3 Bases de Données

Pierre Senellart

1er mars 2017

Évaluation de requêtes

- Requête Q dans un certain langage de requêtes (CQ, FO, FO+μ, FO+μ^+, SO...) – on utilisera le formalisme de la logique
- Base de données D (toujours finie !)
- Évaluation de requêtes : Calcul de $Q(D)$
- Complexité de ce problème ?
- Pour simplifier l'étude de la complexité, on considère que Q est une requête booléenne, c'est-à-dire qu'elle renvoie $\bot$ ou $\top$

Complexité en les données

Pour une certaine requête Q fixée, quelle est la complexité du calcul de $Q(D)$ en fonction de la taille de la base de données D ?

Complexité combinée

Pour un certain langage de requêtes $\mathcal{Q}$, quelle est la complexité du calcul de $Q(D)$ en fonction de la taille de la requête $Q \in \mathcal{Q}$ et de la base de données D ?

Classes de complexité

- On se restreint aux problèmes booléens (renvoyant $\bot$ ou $\top$)
- Ensemble des problèmes résolvables par une méthode de calcul à ressources contraintes :
- Par exemple :

  PTIME : Machine de Turing déterministe en temps polynomial

  NP : Machine de Turing non-déterministe en temps polynomial

  PSPACE : Machine de Turing déterministe en espace polynomial

  AC^0 : Circuit Booléen de taille polynomiale et profondeur constante
- On sait : $AC^0 \subsetneq PTIME \subseteq NP \subseteq PSPACE$
- On ne sait pas si $PSPACE \subseteq PTIME$ (!)

Appartenance et difficulté pour une classe

- Un problème P appartient à une classe de complexité $\mathcal{C}$ (ou dans $\mathcal{C}$) s'il est résolvable par la méthode de calcul à ressources contraintes correspondante
- Un problème P est difficile pour la classe de complexité $\mathcal{C}$ (ou $\mathcal{C}$-difficile) s'il existe une réduction permettant de transformer n'importe quel problème $P' \in \mathcal{C}$ en une instance du problème P
- complet : dans $\mathcal{C}$ + $\mathcal{C}$-difficile
- Différentes manières de définir les réductions
- Ici, on supposera qu'il existe une fonction calculable en temps polynomial qui transforme une instance I' du problème P' en une instance I de P telle que $P(I) = P'(I')$

Complexité descriptive

- Un langage de requêtes $\mathcal{Q}$ capture une classe de complexité $\mathcal{C}$ si :
  - Pour tout $Q \in \mathcal{Q}$, le problème d'évaluation de requêtes pour Q est dans $\mathcal{C}$ (complexité en les données)
  - Pour tout problème P dans $\mathcal{C}$, il existe une requête $Q \in \mathcal{Q}$ telle que l'évaluation de Q résout exactement P (sans réduction) !
- Si $\mathcal{Q}$ capture $\mathcal{C}$ et si $\mathcal{C}$ a des problèmes complets pour $\mathcal{C}$, alors il existe $Q \in \mathcal{Q}$ telle que Q est $\mathcal{C}$-complet, mais la réciproque n'est pas vraie

Plan

Complexité dans les données

Théorème

L'évaluation de CQ est PTIME en complexité dans les données.

Démonstration.

Au tableau, en énumérant toutes les valuations des variables de la requête dans les données. On verra plus loin un résultat beaucoup plus fort. □

Complexité combinée

Théorème

L'évaluation de CQ est NP-complète en complexité combinée.

Démonstration.

Au tableau. Appartenance dans NP est facile. Difficulté pour NP vient du problème de 3-coloriabilité d'un graphe. □

Requête α-acyclique

- Une CQ peut être vue comme un hypergraphe (les sommets sont les variables, les hyperarêtes les atomes de la CQ, étiquetées par le nom de relation)
- Un hypergraphe $\mathcal{H}$ a un arbre de jointure quand on peut trouver un arbre dont les nœuds sont étiquetés par les hyperarêtes de $\mathcal{H}$ et tel que :
  - chaque hyperarête de $\mathcal{H}$ apparaît comme étiquette d'un noeud de l'arbre ;
  - pour chaque sommet x de $\mathcal{H}$, l'ensemble des noeuds de l'arbre étiquetées par une arête incluant x est un sous-arbre connexe de l'arbre
- Une requête est α-acyclique si son hypergraphe admet un arbre de jointure
- Peut-être obtenu en temps linéaire s'il existe [Tarjan and Yannakakis, 1984]

Algorithme de Yannakakis [Yannakakis, 1981]

- Algorithme d'évaluation des requêtes acycliques (non nécessairement booléennes) :
  1. Construction de l'arbre de jointure
  2. Élimination de tous les tuples des relations inutiles au calcul avec l'opérateur de semi-jointure $\ltimes$: $R \ltimes S = \Pi_{R.*}(R \bowtie S)$ par un parcours dans l'arbre de jointure, de bas en haut, puis de haut en bas
  3. Évaluation de la requête de bas en haut, en calculant les jointures suivant l'arbre et en projetant au fur et à mesure sur les seules variables encore utiles
- Complexité polynomiale en la taille de la requête, de l'entrée, de la sortie (complexité combinée)

Plan

Complexité dans les données

Théorème

L'évaluation de FO est PTIME en complexité dans les données.

Démonstration.

Au tableau, par mise en forme prénexe et évaluation naïve. □

FO ne capture pas tout PTIME

Théorème

On ne peut pas calculer en FO qu'une relation contenant un ordre total a un nombre pair d'éléments, ou qu'un graphe est connexe.

Non prouvé, la preuve repose sur les jeux d'Ehrenfeucht–Fraïssé.

Complexité dans les données, plus précis

Théorème

L'évaluation de FO est AC^0 en complexité dans les données.

Démonstration.

Au tableau, intuition de preuve basée sur l'algèbre relationnelle. □

Complexité combinée

Théorème

L'évaluation de FO est PSPACE-complète en complexité combinée.

Démonstration.

Au tableau. Appartenance dans PSPACE facile. Difficulté pour PSPACE vient du problème QSAT. □

Plan

Complexité en les données

Théorème

L'évaluation de $FO{+}\mu^{+}$ est PTIME en complexité en les données.
L'évaluation de $FO{+}\mu$ est PSPACE en complexité en les données.

Démonstration.

Au tableau. Facile. □

Connexité, parité

Théorème

Le problème de connexité peut s'exprimer en $FO+\mu^+$.

Le problème de parité du nombre d'éléments (ou de tuples) dans une relation arbitraire ne peut pas s'exprimer en $FO+\mu$.

Démonstration.

Preuve de la première partie au tableau, facile, cf. le cours précédent.

Deuxième partie admise. □

Parité et FO+μ^+, avec ordre

Théorème

FO+μ peut calculer si une relation contenant un ordre total a un nombre pair d'éléments.

Démonstration.

Au tableau, par construction explicite. □

Résultats plus généraux

Théorème

$FO{+}\mu^{+}$ exprime PTIME sur les données avec un ordre total sur les éléments du domaine.

$FO{+}\mu$ exprime PSPACE sur les données avec un ordre total sur les éléments du domaine.

Démonstration.

Intuition de preuve au tableau. □

Complexité en les données (bis)

Théorème

L'évaluation de $FO{+}\mu^{+}$ est PTIME-complète en complexité en les données.
L'évaluation de $FO{+}\mu$ est PSPACE-complète en complexité en les données.

Démonstration.

Au tableau. Difficulté vient de la complexité descriptive sur les structures ordonnées. □

Au delà : Théorème de Fagin

Théorème

$\exists SO$ (fragment de SO sans quantification universelle de deuxième ordre) exprime NP.

Résultat fondamental, preuve difficile.

Complexité descriptive et P≠NP

- On a une caractérisation de NP : ∃SO
- Si on avait une caractérisation de P...(indépendamment d'ordre sur les éléments du domaine !) par une classe $\mathcal{Q}$
- ... on pourrait montrer P≠NP en exhibant une requête de ∃SO non exprimable dans $\mathcal{Q}$!
- Mais on en est loin...

Plan

Références

- Chapitre 17 de [Abiteboul et al., 1995] sur la complexité de FO, des langages à point fixe
- [Libkin, 2004] sur la théorie des modèles finis et la complexité descriptive (jeux d'Ehrenfeucht–Fraïssé, parité, connexité, théorème de Fagin, etc.)

Bibliographie I

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi : 10.1007/978-3-662-07003-1. URL `http://dx.doi.org/10.1007/978-3-662-07003-1`.

Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3) :566–579, 1984. doi : 10.1137/0213035. URL `http://dx.doi.org/10.1137/0213035`.

Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.