A Top-down Query Evaluation Engine for Datalog

Hana Sebia
Tarik Boumaza

Juin 2021

1 Objectif

Le sujet soumis à l'étude est l'implémentation d'un moteur d'évaluation de requêtes Datalog en Java, en utilisant la méthode de recherche optimale top down.

2 Méthode

Structure du code

L'implémentation complète du programme d'évaluation de requête respecte le squelette fourni (permettant le parsing des fichiers "exemples.txt") et le raisonnement développé par le sujet.

On vous propose la description des 3 grandes étapes permettant l'implémentation du moteur d'évaluation.

Initialisation et pré-traitement

Cette première étape consiste en l'ornement et est réalisée dans la classe `AdornedRules`. On génère le programme orné en commençant par la requête. On détermine l'ornement de chaque IDB composant le corps de la requête de gauche à droite et on propage de façon récursive l'ornement aux règles qui défissent ces IDB.

QSQR State est une classe permettant de stocker pour chaque adorned IDB une input et output relation initialement vides. Une input relation a les variables bound de l'adorned atom comme attributs (pour permettre l'évaluation des règles) ; l'output a comme attributs toutes les variables de l'adorned atom (pour stocker les tuples résultats). On instancie la classe QSQR State à l'entrée de l'évaluation du programme et on la met à jour à chaque évaluation de requête en ajoutant les nouveaux tuples générés.

QSQRSubroutine

Étant donnée une règle ornée et l'input relation de la tête, on génère un QSQRTemplate de cette règle qui représente le schéma des relations supplémentaires. Les attributs de la première relation sup_0 sont les variables bound de la tête.
La dernière relation sup est composée avec tous les attributs de la tête de la règle.
Les relations intermédiaires sont construites à partir des attributs qui sont désignés comme "bound" dans les atomes précédents, ou ceux référencés après.

Après avoir créé le QSQR Template, on instancie les relations se basant sur ces schémas. Ensuite, on les remplit comme suit:

- On copie le input relation de la tête dans le $\sup_0$
- Ensuite, pour chaque atome composant le corps, on distingue deux cas:
  - Si c'est un EDB : on construit une relation contenant tous les tuples de l'EDB qu'on joint avec la relation supplémentaire précédente et on projette le résultat sur les attributs de la relation supplémentaire actuelle pour la remplir.
  - Si c'est un IDB :on ajoute les nouveaux tuples de la relation supplémentaire précédente dans le input relation de cette IDB. Si l'input relation change, on évalue récursivement les règles définissant cette IDB. On remplit alors la relation supplémentaire actuelle en appliquant une jointure entre l'output relation de cette IDB et la relation supplémentaire précédente.
- Au final, les tuples de la dernière relation supplémentaire sont copiés dans l'output relation de la tête.

Je précise que nous avons nous même implémenté les opérations d'algèbre relationnelle mentionnés : la jointure et la projection dans la classe Relation.

QSQR Engine

Pour effectuer ces étapes dans l'ordre, on déroule l'algorithme suivant :
Etant donné un programme Datalog P et une requête query

- On crée le programme orné
- On initialise toutes les relations auxiliaires
- On identifie la règle définissant la query et on l'évalue à l'aide de la méthode QSQR Subroutine.
- On répète cette instruction tant que les input et output relation des adorned atom sont modifiés. Pour cela, on se sert de deux compteurs présents dans la classe QSQR State qu'on incrémente à chaque ajout d'un nouveau tuple.
- On retourne le résultat output_{query}

Résultats

Validité

Une série de 14 tests a été exécutée pour vérifier la validité de notre programme. En particulier, les exemples fournis dans le sujet et la relation reachable. Je vous propose de le vérifier.

Pour vérifier la validité des 14 tests, on compare les résultats obtenus par notre programme avec ceux retournés par DES. Vous trouverez les exemples traités dans l'archive déposée.

Efficacité

Les mesures ont été réalisé en considérant l'étape de calcul de réponse uniquement (à savoir computation time pour datalog). On n'a pas pris en compte le temps nécessaire au parsing.

En effet, une partie importante de la différence constatée peut être expliquée par la contrainte de structure des EDB/IDB imposée par le parsing.