KGRAM: une machine abstraite de graphes de connaissance

Olivier Corby*, Catherine Faron-Zucker**

*INRIA
2004 route des lucioles - BP 93 - FR-06902 Sophia Antipolis cedex
olivier.corby@sophia.inria.fr
**I3S, UNS, CNRS
930 route des Colles - BP 145 - FR-06903 Sophia Antipolis cedex
catherine.faron-zucker@unice.fr

Résumé. Cet article présente la machine abstraite de graphes de connaissance KGRAM qui unifie les notions d'homomorphisme de graphe et de calcul de requêtes telles que celles du langage SPARQL sur des données RDF. KGRAM implémente un ensemble extensible d'expressions qui définissent une famille de langages abstraits d'interrogation de graphes, GRAAL. Nous décrivons la sémantique dynamique de GRAAL en Sémantique Naturelle et nous présentons la machine abstraite KGRAM conçue comme l'interprète de GRAAL, qui implémente les règles de sémantique naturelle du langage.

1 Introduction

Dans cet article nous présentons la machine abstraite de graphes de connaissance KGRAM (acronyme pour Knowledge Graph Abstract Machine) qui unifie les notions d'homomorphisme de graphe et de calcul de requêtes telles que celles du langage SPARQL sur des données RDF/S. KGRAM implémente un ensemble extensible d'expressions qui définissent une famille de langages abstraits d'interrogation de graphes de connaissance quelconques, que nous appelons GRAAL (acronyme pour GRAph Abstract query Languages). Ce travail d'abstraction que nous avons mené pour définir GRAAL a été alimenté par les résultats du projet GRIWES (Baget et al., 2008) auquel nous avons participé. Nous définissons ici GRAAL par sa sémantique dynamique en Sémantique Naturelle et nous définissons KGRAM comme l'implémentation des règles de sémantique naturelle de GRAAL.

Quant à la machine KGRAM proprement dite, nous montrons le haut niveau d'abstraction de son implémentation qui ne manipule que des interfaces, aussi bien pour ses structures de données que pour ses opérations de graphes. Ainsi, différents gestionnaires de graphes et évaluateurs de contraintes peuvent être connectés — en implémentant les APIs de KGRAM. Nous montrons l'interopérabilité de KGRAM en la connectant indifféremment aux gestionnaires de graphes des moteurs sémantiques Corese [1] (Corby et al., 2004) et Jena [2] et à l'évaluateur de contraintes de Corese.

La partie 2 suivante présente le langage GRAAL et la partie 3 la machine KGRAM.

1. http ://www-sop.inria.fr/edelweiss/software/corese/
2. http ://jena.sourceforge.net/

2 La famille GRAAL de langages de requête

Syntaxe abstraite. La syntaxe abstraite du cœur de GRAAL est donnée par la grammaire suivante :

```
Exp ::= NODE | EDGE | FILTER
      | and(Exp, Exp) | union(Exp, Exp) | option(Exp)
      | graph(NODE, Exp) | query(Exp) | not(Exp) | exist(Exp)
```

Les primitives que partagent tous les langages de la famille GRAAL sont les expressions NODE et EDGE permettant de rechercher un nœud ou une relation n-aire (hyperarc) dans un graphe. Une expressions FILTER permet d'exprimer des contraintes sur les nœuds recherchés dans le graphe interrogé. Remarquons que les expressions NODE, EDGE et FILTER sont primitives et nous verrons dans la partir 3 qu'elles correspondent à des interfaces dans la machine abstraite KGRAM qui interprète GRAAL.
Les expressions AND et UNION permettent d'exprimer une conjonction ou une disjonction entre deux expressions.
Une expression OPTION permet de considérer certaines expressions comme optionnelles.
Une expression GRAPH permet de spécifier le graphe de connaissances sur lequel évaluer une expression (en l'absence d'une telle expression, c'est un graphe par défaut qui est considéré).
Une expression QUERY permet d'exprimer des sous-requêtes dont le résultat détermine des liaisons de variables de la requête appelante.
Une expression NOT exprime la négation par l'échec. Une expression EXIST permet de ne rechercher qu'*une* solution (la première trouvée).
D'autres expressions encore sont disponibles dans GRAAL qui ne sont pas présentées ici faute de place. En outre, GRAAL est conçu pour être extensible.

Sémantique naturelle. La Sémantique Naturelle a été initialement conçue par Kahn (1987) pour fournir une sémantique opérationnelle aux langages de programmation, les règles de sémantique naturelle constituant les spécifications des interprètes de ces langages. De manière analogue, en définissant une sémantique opérationnelle de GRAAL, nous spécifions la machine abstraite KGRAM qui peut être vue comme un interprète de GRAAL, les expressions de celle-ci permettant d'interroger des bases de graphes de connaissance.

En Sémantique Naturelle, la sémantique opérationnelle d'un langage est donnée sous la forme d'un ensemble de règles d'inférence, où les expressions du langage sont évaluées dans un environnement et l'application d'une règle d'inférence produit de nouveaux environnements. Ainsi, les règles d'inférence de la sémantique de GRAAL décrivent l'évolution de l'environnement lors de l'évaluation des expressions du langage qui composent une requête. Plus précisément, une expression dans une requête est évaluée dans un environnement consistant en une liste de variables apparaissant dans la requête et liées à des nœuds du graphe interrogé, ces liaisons résultant de l'évaluation précédente d'autres expressions de la requête. L'évaluation d'une expression peut produire plusieurs environnements (dans le cas de plusieurs solutions) et les autres expressions de la requête évaluées ensuite le sont dans chacun d'eux. Lorsque toutes les expressions d'une requête ont été évaluées, chaque environnement qui en résulte correspond à une solution trouvée.

Nous décrivons ici l'opérationalisation des expressions NODE, EDGE et FILTER de GRAAL au travers de règles de sémantique naturelle. Faute de place, les règles opérationalisant les

autres primitives de GRAAL ne sont pas décrites dans cet article. les règles 1 et 2 suivantes montrent le calcul des listes d'environnements lors de l'évaluation d'une expression demandant la recherche d'un nœud ou d'une relation dans un graphe. L'évaluation d'une telle expression dans un environnement ENV requiert de rechercher la liste d'environnements LENV contenant les appariements possibles du nœud ou de la relation dans le graphe interrogé et de fusionner ENV et LENV. Ces deux opérations sont synthétisées dans les règles 1 et 2 par les bases de règles *graph* et *merge* qui décrivent la sémantique respectivement du gestionnaire de graphe et du gestionnaire d'environnement.

$$\frac{graph(ENV \vdash NODE \rightarrow LENV) \ \wedge \ merge(ENV, LENV \rightarrow LENV')}{ENV \vdash NODE \rightarrow LENV'} \quad (1)$$

$$\frac{graph(ENV \vdash EDGE \rightarrow LENV) \ \wedge \ merge(ENV, LENV \rightarrow LENV')}{ENV \vdash EDGE \rightarrow LENV'} \quad (2)$$

Les règles 3 et 4 suivantes montrent qu'un filtre est évalué en utilisant les liaisons des variables de la requête dans l'environnement courant. Si le filtre est évalué à faux (règle 3), l'environnement devient vide (il n'y a pas de solution) ; sinon (règle 4), l'opérateur *list* transforme l'environnement en une liste d'un unique environnement.

$$\frac{eval(ENV \vdash FILTER : false)}{ENV \vdash FILTER \rightarrow \phi} \quad (3) \qquad \frac{eval(ENV \vdash FILTER : true)}{ENV \vdash FILTER \rightarrow list\ ENV} \quad (4)$$

Des langages GRAAL remarquables. Selon le sous-ensemble des primitives de GRAAL que l'on considère, on adopte un langage de requêtes particulier ou un autre. Une limitation aux expressions NODE et NODE définit un langage correspondant à celui des Graphes Conceptuels simples Chein et Mugnier (2008). L'opérationalisation des règles de sémantique naturelle associées à ces expressions correspond à la recherche d'homomorphismes de graphes étiquetés dont les relations peuvent être n-aires. Nous verrons dans la partie 3 que ce calcul des homomorphismes de graphes est la "colonne vertébrale" de l'algorithme de KGRAM.

En ajoutant au langage des expressions FILTER, on considère le modèle des Graphes Conceptuels *avec contraintes* tel que présenté par Baget et Mugnier (2002).

En ajoutant au langage les expressions AND, UNION, OPTION et GRAPH, nous définissons le fragment cœur du langage SPARQL.

3 La machine abstraite KGRAM

Interfaces abstraites. La machine abstraite KGRAM accède au graphe interrogé au travers d'une interface abstraite (API) qui en masque la structure et l'implémentation. Autrement dit, KGRAM opère sur une abstraction de graphe, au travers de structures et de fonctions abstraites. Le graphe interrogé est modélisé sous forme de sommets et d'arcs d'arité n quelconque au travers des interfaces *Node* et *Edge*. Le langage de requêtes de KGRAM permet ainsi d'interroger tout type de graphe de connaissances, par exemple aussi bien des graphes conceptuels (dont les relations sont n-aires) que des graphes RDF (dont les relations sont binaires).

Non seulement les structures de données manipulées par KGRAM sont abstraites, mais aussi ses opérateurs :

- Le gestionnaire de graphes de KGRAM qui permet d'accéder au graphe interrogé est un objet implémentant l'interface *Producer* qui énumère des arcs (resp. sommets) cibles correspondant à un arc (resp. sommet) requête.
- Le test de correspondance entre les sommets (resp. arcs) est effectué par un objet implémentant l'interface *Matcher*. Cet objet a la charge de la comparaison des étiquettes (labels et types) de sommets. Selon l'implémentation de l'interface *Matcher*, il prendra en compte dans cette comparaison les relations de subsomption entre types, il pourra autoriser des appariements approchés basés sur des mesures de similarité, etc.
- Les contraintes (ou filtres) sont des objets abstraits qui implémentent l'interface *Filter* et qui sont évalués par un objet qui implémente l'interface *Evaluator*. KGRAM ignore la structure interne des filtres qu'il manipule, il se contente d'appeler la fonction *eval* de l'interface *Evaluator* sur des objets *Filter*, en passant en argument un *Environment*.

Ces interfaces montrent bien le haut niveau d'abstraction auquel nous nous sommes tenus dans la conception de KGRAM. L'algorithme de KGRAM qui manipule ces interfaces est ainsi totalement abstrait, indépendant de toute implémentation et de toute structure de données.

Algorithme. La fonction `eval` de KGRAM évalue une expression du langage GRAAL stockée sous la forme d'une pile `stack` d'expressions élémentaires. Elle fait appel à un gestionnaire de graphes abstrait `producer` qui considère une expression `exp` de la pile `stack` et l'environnement courant `memory` des liaisons de variables du graphe requête avec des nœuds du graphe interrogé. La fonction `candidate` du gestionnaire de graphes utilise l'environnement pour trouver les éventuels nœuds dans l'expression `exp` déjà liés, de sorte qu'elle retourne les seules relations candidates compatibles avec ces liaisons présentes dans l'environnement. Les relations candidates qui respectent les liaisons dans l'environnement sont ajoutées dans une pile des relations, ainsi que les nœuds de la relation dans la pile des nœuds de l'environnement. Cette pile de nœuds peut être utilisée comme environnement d'évaluation des filtres, comme nous le verrons plus loin. La recherche d'homomorphisme aboutit et l'homomorphisme partiel en construction devient complet lorsque le sommet de la pile d'expressions de la requête est atteint : l'environnement courant représente alors une solution.

KGRAM est conçu comme l'interprète du langage GRAAL. Son algorithme repose sur l'opérationalisation des règles 1 et 2 de sémantique naturelle associées aux expressions NODE et EDGE de GRAAL qui permettent de rechercher des homomorphismes de graphe :

```
eval(stack, n){
  if (stack.size() == n){store(); return;}
  exp = stack(n);
  switch(exp){
    case EDGE:
      for (Edge r : producer.candidate(exp, memory)){
        if (match(exp, r)){ // test bindings, types, etc.
           memory.push(exp, r); eval(stack, n+1); memory.pop(exp, r);}}
    case NODE: ... // similar to case EDGE
    case FILTER:
      if (evaluator.eval(exp, memory)){eval(stack, n+1);}
    ... }}
```

Dans l'instruction de contrôle `switch`, les blocs avec les types NODE et EDGE comme valeur de la variable `exp` implémentent les paquets de règles *graph* et *merge* des règles 1 et 2 et

donc augmentent l'environnement courant par des liaisons des nœuds de la requête avec des nœuds du graphe interrogé. KGRAM permet ainsi une recherche d'homomorphismes guidée par l'appariement de nœuds et/ou par l'appariement de relations.

Le bloc avec le type FILTER comme valeur de la variable de contrôle `exp` opérationalise les règles 3 et 4 de sémantique naturelle relatives aux expressions FILTER de GRAAL, et ce faisant KGRAM implémente la recherche d'homomorphismes de graphe sous contraintes. L'algorithme de KGRAM utilise un évaluateur de filtres abstrait `evaluator` et reste indépendant de la nature des filtres traités — qui dépendent du langage de filtres implémenté par l'évaluateur appelé. La fonction d'évaluation de l'évaluateur de filtres prend en argument le filtre à évaluer et la pile de nœuds qui constitue l'environnement courant d'évaluation. Dans le cas où le filtre est évalué à vrai, la recherche d'homomorphismes continue avec comme environnement celui issu de l'évaluation du filtre. Dans le cas contraire, l'homomorphisme partiel que constituait l'environnement courant n'aboutit pas à une solution.

Les règles de sémantique naturelle des expressions AND, UNION et OPTION de GRAAL sont opérationalisées dans KGRAM en intégrant à la "colonne vertébrale" de l'algorithme présentée ci-dessus trois nouveaux blocs dans l'instruction de contrôle `switch`. KGRAM implémente ainsi le cœur du langage SPARQL et nous avons développé un compilateur qui traduit des expressions SPARQL en expressions GRAAL. KGRAM opérationalise également les règles de sémantique naturelle des autres expressions GRAAL présentées dans la partie 2 : GRAPH, QUERY, NOT et EXIST. Cela n'est pas détaillé dans cet article, faute de place.

Interopérabilité. Nous avons testé la portabilité de KGRAM en réalisant une implémentation des interfaces *Producer*, *Matcher* et *Evaluator* de KGRAM par Corese et par Jena. Les interfaces de KGRAM ont été conçues pour minimiser le "code glue" à réaliser de sorte que ces maquettes de portage de KGRAM sur un moteur ou sur l'autre ont demandé relativement peu de développement. La connexion de Corese à KGRAM a été presque immédiate, ce qui s'explique par le fait que nous avons conçu KGRAM en abstrayant les principes de Corese. Connecté à Corese, KGRAM interprète l'ensemble des expressions de GRAAL. Avec Jena, le portage a nécessité 416 lignes de code et quatre classes : `EdgeImpl`, `ProducerImpl`, `EvaluatorImpl` et `NodeImpl`. Avec Jena, KGRAM interprète actuellement les expressions EDGE, FILTER, UNION, AND et OPTION. Le travail de maquettage se poursuit.

Ces deux implémentations témoignent de la généricité de la conception de KGRAM et laissent supposer une connexion facile de KGRAM à d'autres implémentations de gestionnaires de graphes de connaissance.

4 Conclusion

Nous avons présenté dans cet article la machine abstraite de graphes de connaissances KGRAM et son langage de requête à base de graphes GRAAL. Nous avons établi des règles de sémantique naturelle pour chacune des expressions de GRAAL, ces règles d'inférence constituant les spécifications de KGRAM qui les opérationalise et qui peut être vu comme un interprète du langage GRAAL. Nous avons mis en lumière le niveau d'abstraction de KGRAM, la simplicité de son algorithme reposant sur la manipulation d'*interfaces* aussi bien pour les opérateurs que pour les structures de données.

Dans la continuité des résultats présentés dans cet article, nous travaillons actuellement à l'intégration d'optimisations telles que celles proposées par (Corby et Faron-Zucker, 2007), comme le tri des relations dans la pile de l'environnement selon des heuristiques permettant ensuite des opérations de backjump dans la pile, ou encore la possibilité de demander plusieurs relations adjacentes au gestionnaire de graphes plutôt que de les demander une à une.

Dans le prolongement de notre expérience réussie d'implémentation des interfaces de KGRAM par les moteurs Corese et Jena, nos perspectives de travail sont d'aborder le problème de la distribution du traitement du web de données en interconnectant différents gestionnaires de graphes responsables chacun d'une base d'annotations et implémentant chacun l'APIs de KGRAM. Nous envisageons KGRAM d'une part comme un élément de réponse au problème du passage à l'échelle dans le traitement du web de données et d'autre part comme la clé de voute d'applications de mashup combinant les résultats de différents gestionnaires de graphe.

Références

Baget, J., O. Corby, R. Dieng-Kuntz, C. Faron-Zucker, F.Gandon, A. Giboin, A. Gutierrez, M. Leclère, M. Mugnier, et R. Thomopoulos (2008). GRIWES : Generic Model and Preliminary Specifications for a Graph-Based Knowledge Representation Toolkit. In *Proc. of the 16th International Conference on Conceptual Structures, ICCS 2008, Toulouse, France*, Volume 5113 of *Lecture Notes in Computer Science*, pp. 297–310. Springer.

Baget, J. et M. Mugnier (2002). Extensions of Simple Conceptual Graphs : the Complexity of Rules and Constraints. *J. Artif. Intell. Res. (JAIR) 16*, 425–465.

Chein, M. et M. Mugnier (2008). *Graph-based Knowledge Representation : Computational Foundations of Conceptual Graphs*. Springer London Ltd.

Corby, O., R. Dieng-Kuntz, et C. Faron-Zucker (2004). Querying the Semantic Web with Corese Search Engine. In *Proc. of the 16th Eureopean Conference on Artificial Intelligence, ECAI 2004*, pp. 705–709. IOS Press.

Corby, O. et C. Faron-Zucker (2007). Implementation of SPARQL Query Language Based on Graph Homomorphism. In *Proc. of the 15th International Conference on Conceptual Structures, ICCS 2007, Sheffield, UK*, Volume 4604 of *Lecture Notes in Computer Science*, pp. 472–475. Springer.

Kahn, G. (1987). Natural Semantics. In *Proc. of 4th Annual Symposium on Theoretical Aspects of Computer Science, Passau, Germany, STACS 87*, Volume 247 of *Lecture Notes in Computer Science*, pp. 22–39. Springer.

Summary

In this paper we present the KGRAM Knowledge Graph Abstract Machine that unifies graph homomorphism and SPARQL-like query processing on RDF datasets. KGRAM implements an extensible set of expressions which define the GRAAL family of GRAph Abstract query Languages. We describe the dynamic semantics of GRAAL in Natural Semantics and e present KGRAM which is designed as the interpret of GRAAL and implements the rules of natural semantics of GRAAL.