Querying Key-Value Stores Under Simple Semantic Constraints : Rewriting and Parallelization

Olivier Rodriguez
Université de Montpellier
olivier.rodriguez@etu.umontpellier.fr

Corentin Colomier
Université de Montpellier
corentin.colomier@etu.umontpellier.fr

Cecilie Rivière
Université de Montpellier
cecilie.riviere@etu.umontpellier.fr

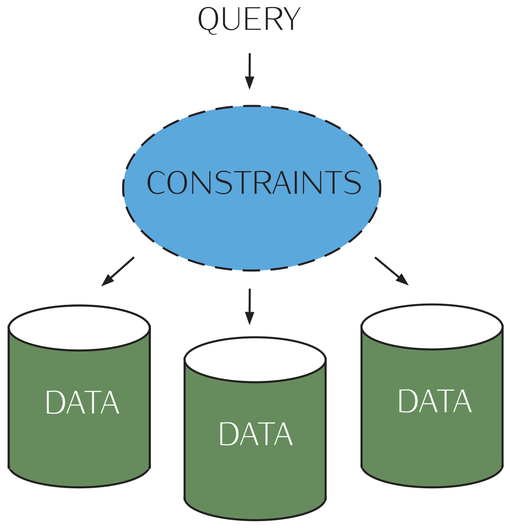
Reza Akbarinia
INRIA Sophia Antipolis
reza.akbarinia@inria.fr

Federico Ulliana
Université de Montpellier
federico.ulliana@lirmm.fr

Accessing data under semantic constraints

Popular paradigm, considered for example in ontology-based data integration.

Usual setting is : DL or existential rules + relational or RDF data + conjunctive queries.



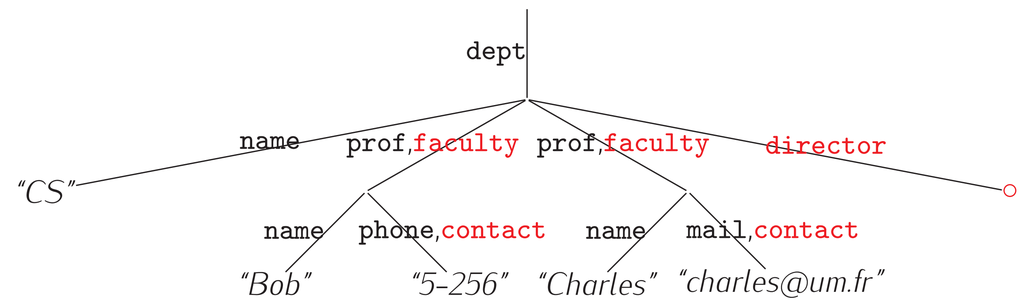
What if the data to access is *not* relational, like JSON records ?

Goal : exploit fast data access provided by Key-value Stores systems

Constraints : rules on JSON data (Mugnier et al. AAAI 2016, Bienvenu et al. IJCAI 2017)

The tree view of JSON records

```
{ dept :
   { name : "CS",
     prof : [
       {name : "Bob",   phone : "5-256" },
       {name : "Charles",   mail : "charles@um.fr" } ] }
}
```



Q_1 : `find( { dept : { prof : { contact : { $exists : true } } } } )`

Q_2 : `find( { dept : { $elemMatch : { name : "CS", director : { $exists : true } } } } )`

Demo Contribution

A system for accessing Key-Value stores that

1. accounts for semantic constraints over keys
2. supports MongoDB queries akin to tree-pattern queries (without joins)
3. exploits query reformulation techniques for data access
4. parallelizes the computing of large rewriting sets over multiple threads

Key Constraints

General form of a *key constraint* :

(∀-rule)	$k_1 \longrightarrow k_2$	(inclusion between keys)
(∃-rule)	$k_1 \longrightarrow \exists\, k_2$	(mandatory key)

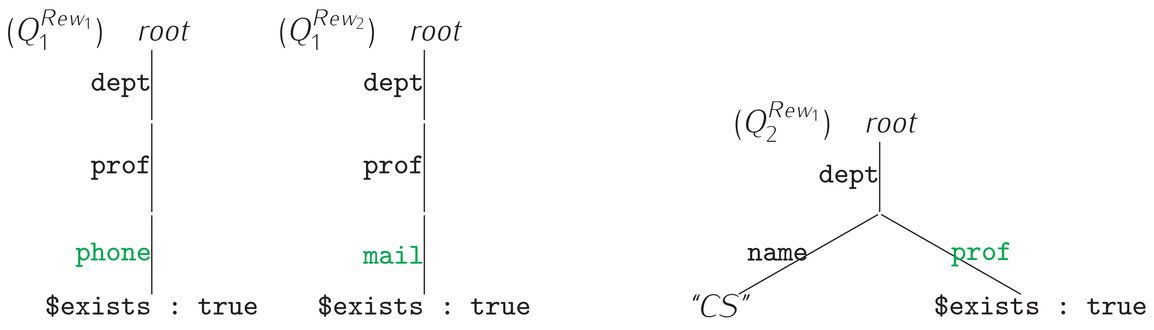
(σ_1)	`phone` $\longrightarrow$ `contact`	Every phone is a contact
(σ_2)	`mail` $\longrightarrow$ `contact`	Every mail is a contact
(σ_3)	`prof` $\longrightarrow \exists$ `director`	If there is a professor then there is a director
(σ_4)	`prof` $\longrightarrow$ `faculty`	A professor is a faculty member

Query Rewriting

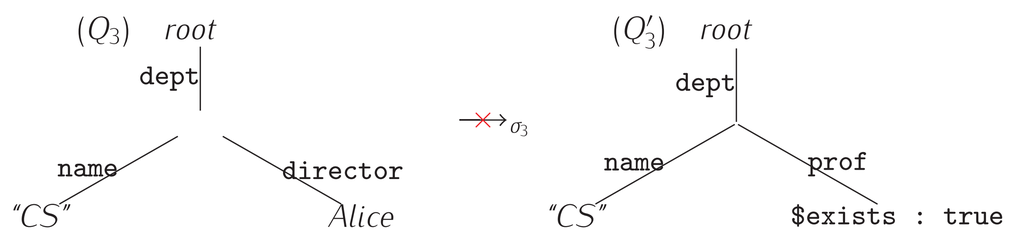
Rewrite the edges of the tree-like query :

(∀-rule) $k_1 \longrightarrow k_2$ replace k_2 with k_1

(∃-rule) $k_1 \longrightarrow \exists\, k_2$ replace k_2 with k_1, only on "existential leaves".



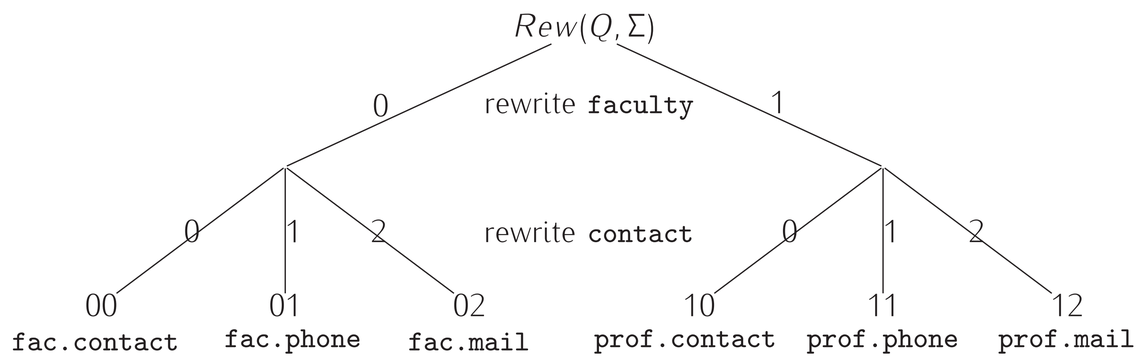
∃-rules must be applied only on existential leaves to ensure soundness.



Parallelization

Goal : parallelize the generation of the whole rewriting set.

Rewrite Q : {`faculty` : {`contact` : { `$exists : true` }}} with the rules $\sigma_1, \sigma_2, \sigma_4$.



1) Enumerate the edges of the query and build an unranked decision tree where

- the level j corresponds to the rewriting of the edge j of the query ($1 \leq j \leq |Q|$)
- the degree of a node d_j corresponds to the possible rewritings of the edge j

2) A leaf-code $(c_1 \dots c_\rho) \in \{0, 1, \dots, |\Sigma|\}^{\rho \leq |Q|}$ given $(d_1, \dots, d_\rho)$ is the integer $p_{(c_1 \dots c_\rho)} = c_\rho * B_1 + c_{\rho-1} * B_2 + \dots + c_1 * B_\rho$ with $B_1 = 1$ and $B_{i \geq 2} = d_{i-1} * B_{i-1}$.

3) Assign to each of n rewriters an interval of queries to be generated $[N, \dots, N+\lambda]$ with $\lambda \approx |leaves(Rew(Q, \Sigma))|/n$.