Sur l'analyse statique des requêtes SPARQL avec la logique modale

Nicola Guido

▶ **To cite this version:**

Nicola Guido. Sur l'analyse statique des requêtes SPARQL avec la logique modale. Web. Université Grenoble Alpes, 2015. Français. NNT : 2015GREAM059 . tel-01250984v1

HAL Id: tel-01250984
https://inria.hal.science/tel-01250984v1

Submitted on 5 Jan 2016 (v1), last revised 28 Apr 2016 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Static Analysis for SPARQL Queries using Modal Logic

Nicola Guido
Ph.D Defense

3 December, 2015

Supervisor: Cécile Roisin
Co-supervisor: Pierre Genevès

Université Joseph Fourier GRENOBLE

upmf Grenoble Université Pierre Mendès-France Sciences sociales & humaines

Motivation

Context

- Semantic Web - web of data that can be processed by machines
- Technologies: URI, Unicode, XML and Namespace
- W3C Standards: RDF, RDFS, OWL, ...
- Huge graphs that can be queried with SPARQL

Problem

- Querying huge graphs is time consuming
- Query optimization benefits from Static Analysis

Static analysis

Many optimization tasks rely on three simple questions:

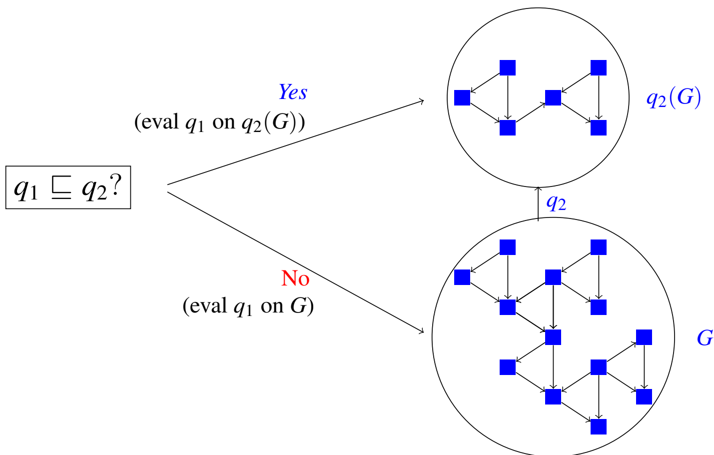
- Does query q ever produce a non-empty result?
- Does query q_1 always produce the same result as query q_2?
- Does query q_1 always produce a subset of the results of query q_2?

Definition (Query containment)

Given two queries q_1 and q_2, q_1 is contained in q_2, denoted $q_1 \sqsubseteq q_2$, if for every dataset G, $q_1(G) \subseteq q_2(G)$.

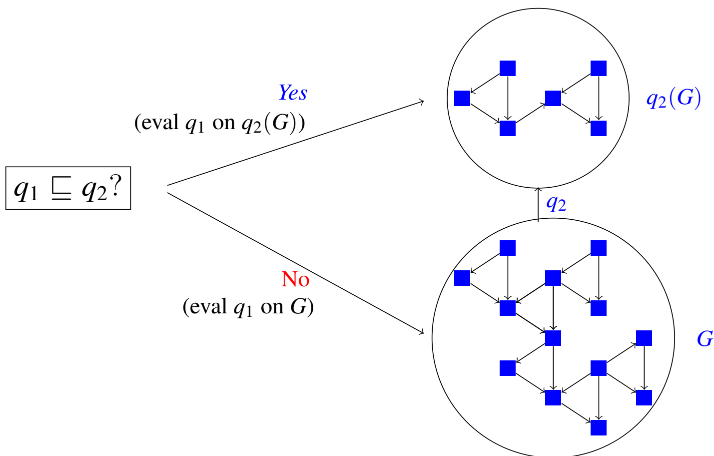
Query containment problem

SPARQL Queries: q_1, q_2 RDF Graph: G $|q_2(G)| << |G|$



Query containment problem

SPARQL Queries: q_1, q_2 RDF Graph: G $|q_2(G)| << |G|$



Main difficulty: containment must hold for ALL graphs

Query containment is well-studied

1. conjunctive queries [Chandra-and-Merlin:1977], NP-Complete
2. union of conjunctive queries: NP-Complete [Sagiv-et-al:1980]
3. conjunctive query containment under DL schema axioms: 2EXPTIME [Calvanese-et-al:2008]
4. static analysis of XPath queries: $2^{O(|n|)}$ [Geneves-et-al:2007]

Query containment is well-studied

1. conjunctive queries [Chandra-and-Merlin:1977], NP-Complete
2. union of conjunctive queries: NP-Complete [Sagiv-et-al:1980]
3. conjunctive query containment under DL schema axioms: 2EXPTIME [Calvanese-et-al:2008]
4. static analysis of XPath queries: $2^{O(|n|)}$ [Geneves-et-al:2007]

but, much less for SPARQL

- union of conjunctive queries under DL schema axioms: 2EXPTIME [Chekol-et-al:2012],
- containment in presence of OPTIONAL operators: Π_2^P-complete [Letelier-et-al:2012]

Outline

1 Motivation

2 Query Containment Problem for SPARQL
- Preliminaries
- Logical Approach
- Experiments

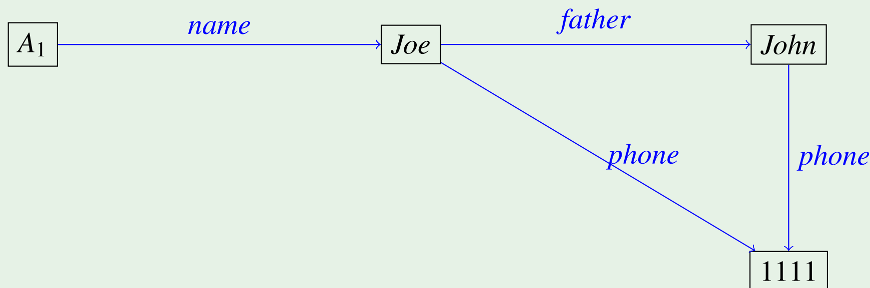
3 Conclusion and Perspectives

Resource Description Framework

- is a graph data format for the representation of information in the Web
- subject-predicate-object structures
- an RDF graph is a set of triples

Example (An RDF graph describing employers information)

$G = \{(A_1, \textit{name}, \textit{Joe}), (\textit{Joe}, \textit{father}, \textit{John}), (\textit{Joe}, \textit{phone}, 1111), (\textit{John}, \textit{phone}, 1111)\}$

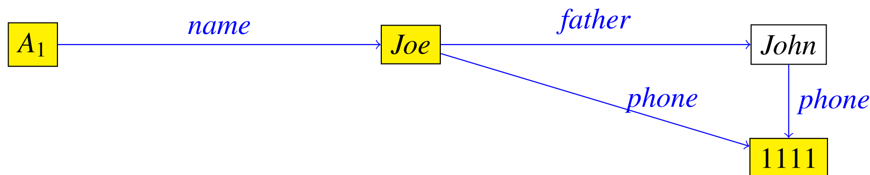


SPARQL Protocol And RDF Query Language

- W3C standard query language for RDF graphs
- based-on subgraph matching

$$q = \quad \mathit{SELECT} \quad ?a\ ?n\ ?p \quad \mathit{WHERE} \quad \left\{ \left((?a,\ name,\ ?n)\ \mathit{AND}\ (?n,\ phone,\ ?p) \right) \right\}$$

Evaluation:



$[\![q]\!]_G = \{\varrho = \{?a \to A_1,\ ?n \to Joe,\ ?p \to 1111\}\},$ where $\varrho : V \to U$

SPARQL queries containment: example

q_1	q_2
$SELECT \quad ?a \; ?n$ $HERE \quad \{(?a,\ name,\ ?n)\}$	$SELECT \quad ?a \; ?n \; ?p$ $WHERE \quad \{(?a,\ name,\ ?n)$ $OPT\ (?a,\ phone,\ ?p)\}$

$$G = \{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111)\}$$

$$[\![q_1]\!]_G = \{\varrho_1 = \{?a \to A_1,\ ?n \to Joe\}\}$$

$$[\![q_2]\!]_G = \{\varrho_2 = \{?a \to A_1,\ ?n \to Joe,\ ?p \to 1111\}\}$$

$[\![q_1]\!]_G \not\subseteq [\![q_2]\!]_G$ we have that $\varrho_1 \notin [\![q_2]\!]_G$

We consider subsumption:
$[\![q_1]\!]_G \sqsubseteq [\![q_2]\!]_G$ because q_2 contains more information than that of q_1

Expressivity, decidability and complexity

Abstact syntax of graph patterns:

$$P ::= tp \mid P \ AND \ P \mid P \ OPT \ P \mid P \ UNION \ P$$

Complexity of query containment

SPARQL Fragment	Complexity	
UNION-OPT-AND	Undecidable	[Trahktenbrot:1950]
OPT-AND	Undecidable	[Letelier-et-al:2012]
UNION-AND	NP-Complete	[Sagiv-et-al:1980]
AND	NP-Complete	[Chandra-et-al:1977]
Well-Designed OPT-AND	Π_2^P-Complete	[Letelier-et-al:2012]

Well-designed SPARQL queries [Perez-et-al:2009]

Restriction on variable usage

$$P = (?a, name, Joe)\ OPT\ \Big(\underbrace{\boxed{(?y, email, ?e)}\ OPT\ (?a, phone, ?p)}_{P_1}\Big)$$ ✗

$$P = (?a, name, Joe)\ OPT\ \Big(\underbrace{(?a, email, ?e)\ OPT\ (?a, phone, ?p)}_{P_1}\Big)$$ ✓

Well-designed SPARQL Queries allows a Normal Form

(1) $P_1\ AND\ (P_2\ OPT\ P_3)\ \rightarrow\ (P_1\ AND\ P_2)\ OPT\ P_3$

(2) $(P_1\ OPT\ P_2)\ AND\ P_3)\ \rightarrow\ (P_1\ AND\ P_2)\ OPT\ P_3$

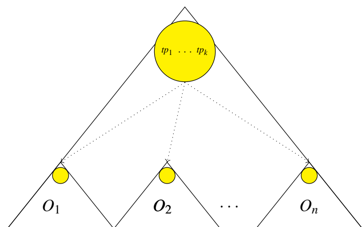
Well-designed SPARQL queries [Perez-et-al:2009]

$$\underbrace{((\ldots(tp_1 \ AND \ tp_2 \ AND \ \ldots \ AND \ tp_k) \ OPT \ O_1) \ OPT \ O_2) \ldots OPT \ O_n)}_{O_0} \qquad (\star)$$

$O_1, O_2, \ldots, O_n$ have the same form of $(\star)$

OPT only outside the scope of *AND* components.

OPT operators define a tree structure of *AND* components.



- Well-designed SPARQL queries support the Berlin Benchmark [Bizer-et-al:2009].
- 48% of DBpedia queries using OPT are well-designed [Picalausa-et-al:2011]

This fragment preserves decidability of containment in presence of OPT-AND

State-of-the-art Implementations for SPARQL Containment

1. SPARQL-Algebra for well-designed OPT-AND queries, Π_2^P-complete[Letelier-et-al, 2012] (No OWL constraints)
2. Logical approach for UNION-AND queries (+ OWL constraints), 2*Exptime* upper bound [Chekol-et-al, 2012] (No OPT)

Initial questions:

- Is it possible to extend the SPARQL Algebra with OWL constraints?
- Is it possible to find an intermediate approach with a better complexity?
- Is it possible to extend the logical approach to support OPT constructors?

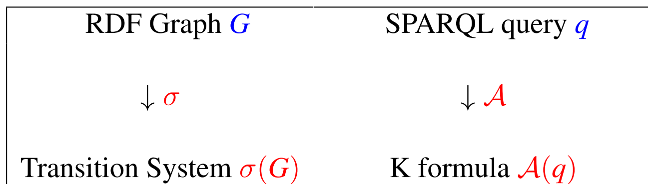
Why a logical approach?

1. close connection between query containment and validity of a formula in logic
2. approach successfully applied for UNION-AND fragments
3. it is efficiently implementable and extensible
4. existing solvers optimized for years
5. transferring complexity results.
6. possibility of query evaluation through model checking

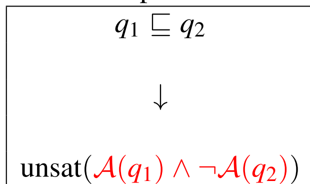
I am going to show that very efficient SPARQL containment checkers can be implemented using K Logic.

Approach

Encoding graphs and queries into a logic

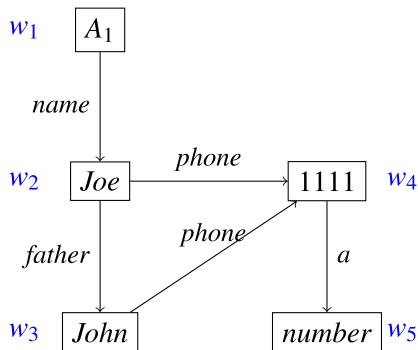


Reduce query containment to the problem of unsatisfiability in K-logic



Transition system

- consists of a set of states and transitions between states, formally, $\mathcal{M} = (W, R, V)$



- $W = \{w_1, w_2, w_3, w_4, w_5\}$
- $R(name) = \{(w_1, w_2)\}$
  $R(phone) = \{(w_2, w_4), (w_3, w_4)\}$
  $\ldots$
- $V(w_1) = \{A_1\}; \ldots$

K logic

Syntax

$$\varphi \ ::= \ \top \mid q \mid \underline{q} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle m \rangle \varphi$$

where:

$q \in \mathcal{AP}$ (Atomic Propositions)

$\underline{q} \in \mathcal{AP}_\Sigma$ (Exclusive Atomic Propositions)

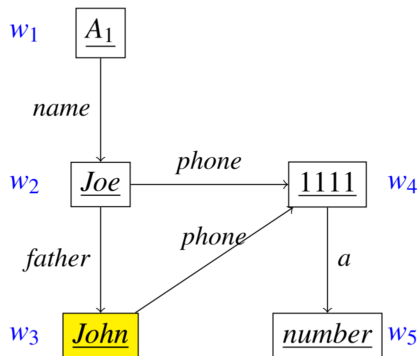
$$\forall(\underline{u}, \underline{u}') \in (\mathcal{AP}_\Sigma)^2, \ (\underline{u} \neq \underline{u}') \Rightarrow \neg\underline{u} \vee \neg\underline{u}'$$

$m \in MOD = \{\alpha, \beta, s, o, ..\}$ (Modalities)

Syntactic Sugar:

Abbreviation	*Formula*
$\bot$	$\neg\top$
$\varphi \vee \psi$	$\neg(\neg\varphi \wedge \neg\psi)$
$[m]\varphi$	$\neg\langle m \rangle \neg\varphi$

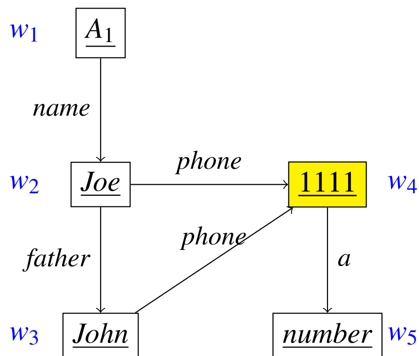
K formulæ: Examples



Formulæ:

- *John*

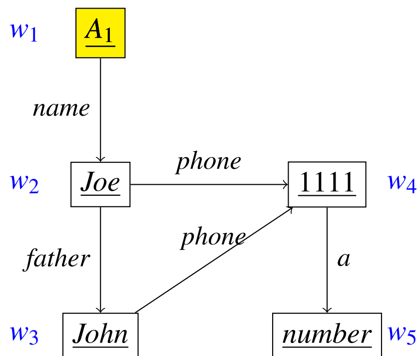
K formulæ: Examples



Formulæ:

- *John*
- $\langle a \rangle$*number*

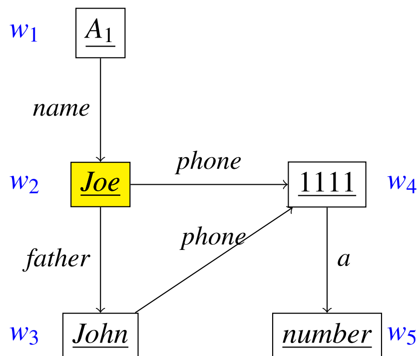
K formulæ: Examples



Formulæ:

- $\underline{John}$
- $\langle a \rangle \underline{number}$
- $\langle name \rangle \langle father \rangle \underline{John}$

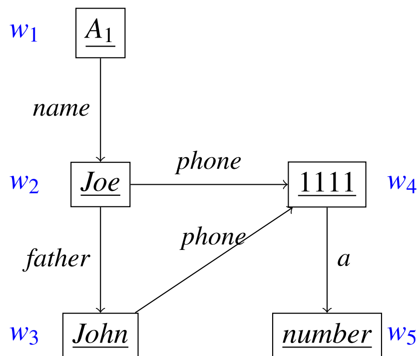
K formulæ: Examples



Formulæ:

- *John*
- $\langle a \rangle$*number*
- $\langle name \rangle \langle father \rangle$*John*
- *Joe* $\wedge \langle father \rangle$*John*

K formulæ: Examples

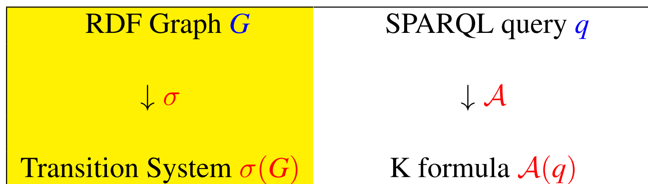


Formulæ:

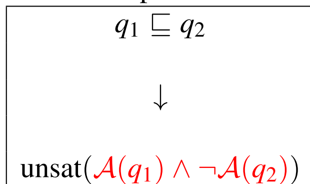
- $\underline{John}$
- $\langle a \rangle \underline{number}$
- $\langle name \rangle \langle father \rangle \underline{John}$
- $\underline{Joe} \wedge \langle father \rangle \underline{John}$
- $\underline{Joe} \wedge \underline{John}$

Approach

Encoding graphs and queries into a logic

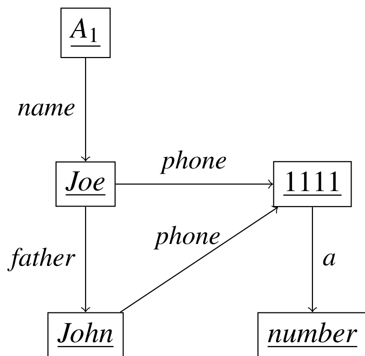


Reduce query containment to the problem of unsatisfiability in K-logic



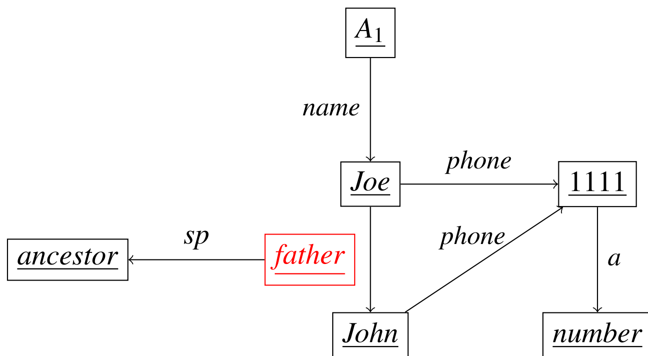
Encoding RDF graphs as transition systems

Use RDF graph as a transition system.



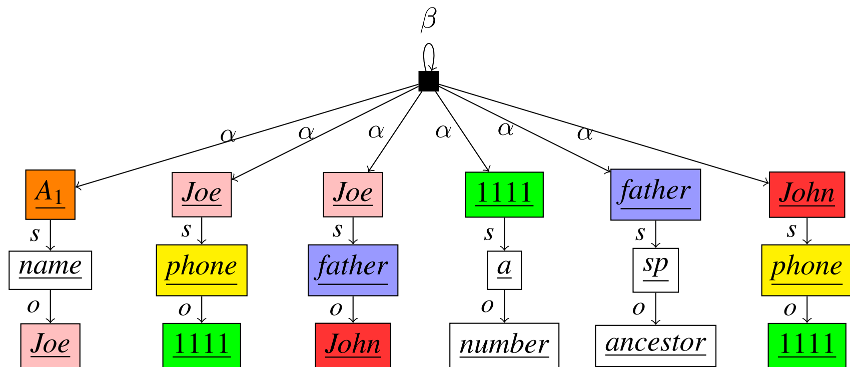
Encoding RDF graphs as transition systems

Predicates can be nodes in an RDF graph



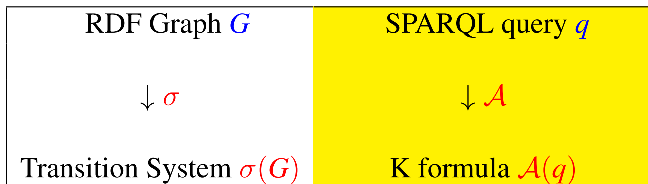
Our encoding

- we add a node called *graph node* ■
- connect it to all subject nodes
- finite set of modalities $\{\alpha, \beta, s, o\}$.
- α for *AND* operators and β for *OPT* operators

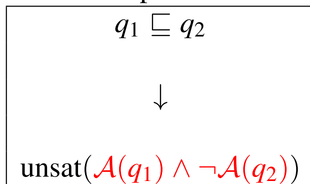


Approach

Encoding graphs and queries into a logic

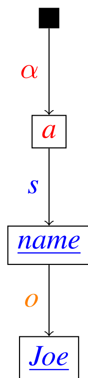


Reduce query containment to the problem of unsatisfiability in K-logic



Encoding SPARQL queries as K formulæ

$q = \{(?a,\ name,\ Joe)\}$



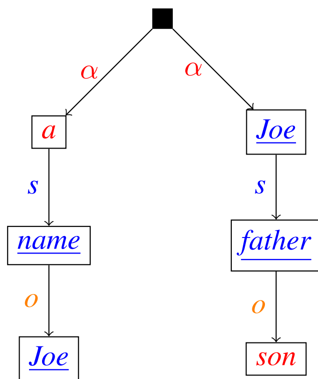
- each triple is connected to a graph node,
- the graph node is connected to a subject, a subject to a predicate and a predicate to an object

- every URI is encoded with an exclusive atomic proposition ($\mathcal{AP}_\Sigma$) and $\forall(\underline{u}, \underline{u}') \in (\mathcal{AP}_\Sigma)^2\ (\underline{u} \neq \underline{u}') \Rightarrow \neg\underline{u} \vee \neg\underline{u}'$
- variables are encoded with atomic propositions ($\mathcal{AP}$)

$$\langle \alpha \rangle \Big(a \wedge \langle s \rangle (\underline{name} \wedge \langle o \rangle \underline{Joe}) \Big)$$

Encoding SPARQL queries as K formulæ

$q = \{(?a,\ name,\ Joe)\ AND\ (Joe,\ father,\ ?son)\}$



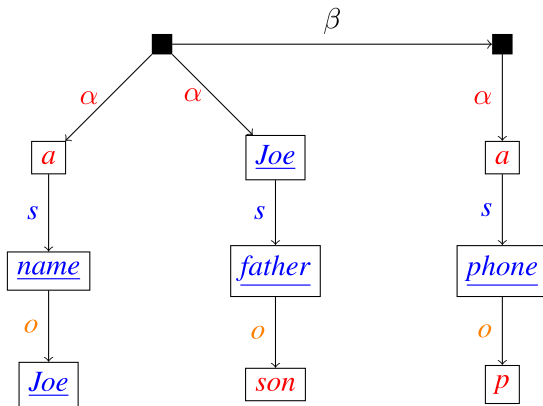
- each triple is connected to a graph node,
- the graph node is connected to a subject, a subject to a predicate and a predicate to an object

- every URI is encoded with an exclusive atomic proposition ($\mathcal{AP}_\Sigma$) and $\forall(\underline{u}, \underline{u}') \in (\mathcal{AP}_\Sigma)^2\ (\underline{u} \neq \underline{u}') \Rightarrow \neg\underline{u} \vee \neg\underline{u}'$
- variables are encoded with atomic propositions ($\mathcal{AP}$)

$$\langle\alpha\rangle\Big(a \wedge \langle s\rangle(\underline{name} \wedge \langle o\rangle\underline{Joe})\Big) \wedge \langle\alpha\rangle\Big(\underline{Joe} \wedge \langle s\rangle(\underline{father} \wedge \langle o\rangle son)\Big)$$

Encoding SPARQL queries as K formulæ

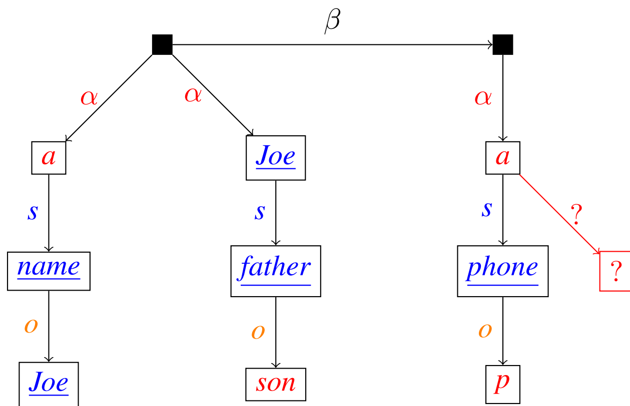
$q = \{\big((?a,\ name,\ Joe)\ AND\ (Joe,\ father,\ ?son)\big)\ OPT\ (?a,\ phone,\ ?p)\}$



$$\langle\alpha\rangle\Big(a\wedge\langle s\rangle(\underline{name}\wedge\langle o\rangle\underline{Joe})\Big)\wedge\langle\alpha\rangle\Big(\underline{Joe}\wedge\langle s\rangle(\underline{father}\wedge\langle o\rangle son)\Big)\wedge[\beta]\langle\alpha\rangle\Big(a\wedge\langle s\rangle(\underline{phone}\wedge\langle o\rangle p)\Big)$$

Encoding SPARQL queries as K formulæ

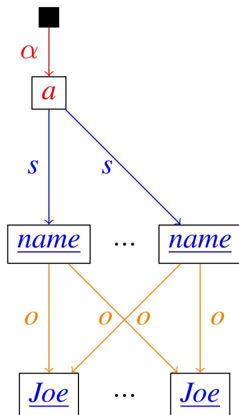
$q = \{\big((?a,\ name,\ Joe)\ AND\ (Joe,\ father,\ ?son)\big)\ OPT\ (?a,\ phone,\ ?p)\}$



$$\langle\alpha\rangle\Big(a \wedge \langle s\rangle(\underline{name} \wedge \langle o\rangle \underline{Joe})\Big) \wedge \langle\alpha\rangle\Big(\underline{Joe} \wedge \langle s\rangle(\underline{father} \wedge \langle o\rangle son)\Big) \wedge [\beta]\langle\alpha\rangle\Big(a \wedge \langle s\rangle(\underline{phone} \wedge \langle o\rangle p)\Big)$$

Query encoding: requirements on outgoing edges

$$q = \{(?a,\ name,\ Joe)\}$$
$$\downarrow \mathcal{A}$$
$$\mathcal{A}(?a,\ name,\ Joe) = \langle\alpha\rangle\big(a \wedge \langle s\rangle T \wedge [s](\underline{name} \wedge \langle o\rangle T \wedge [o]\underline{Joe})\big)$$



- All s-transitions from a subject node go to predicate nodes labelled *name*.
- All o-transitions from a predicate node go to object nodes labelled *Joe*

$$\langle a\rangle\varphi = \langle a\rangle\top \wedge [a]\varphi \qquad a = \{s, o\}$$

Query encoding: requirements on outgoing edges

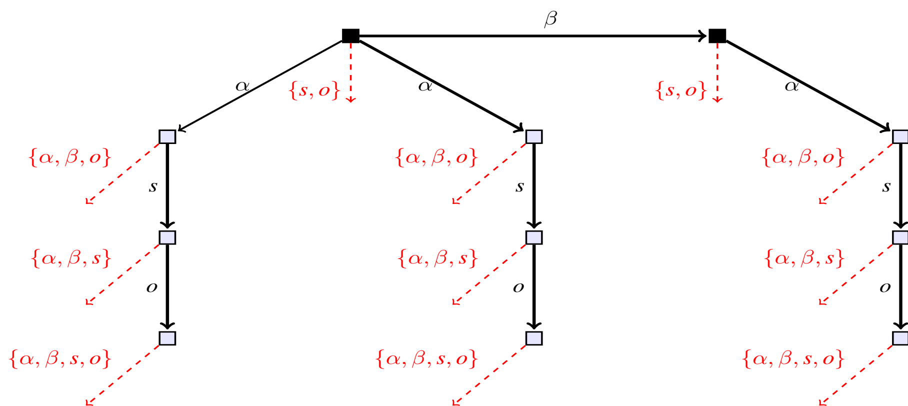
$$\xi = \texttt{isG} \wedge [\alpha](\texttt{isS} \wedge [s](\texttt{isP} \wedge [o]\texttt{isO}))$$

$\texttt{isG} = \neg\langle s\rangle\top \wedge \neg\langle o\rangle\top$

$\texttt{isO} = \neg\langle\beta\rangle\top \wedge \neg\langle\alpha\rangle\top \wedge \neg\langle s\rangle\top \wedge \neg\langle o\rangle\top$

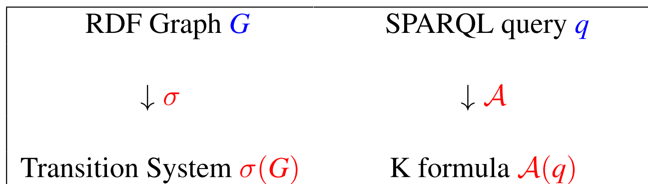
$\texttt{isS} = \neg\langle\beta\rangle\top \wedge \neg\langle\alpha\rangle\top \wedge \neg\langle o\rangle\top$

$\texttt{isP} = \neg\langle\beta\rangle\top \wedge \neg\langle\alpha\rangle\top \wedge \neg\langle s\rangle\top$

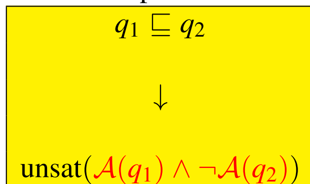


Approach

Encoding graphs and queries into a logic



Reduce query containment to the problem of unsatisfiability in K-logic



Reduce query containment to unsatisfiability in K

$$q_1 \sqsubseteq q_2 \;\Leftrightarrow\; \mathrm{val}\Big(\mathcal{A}(q_1) \rightarrow \mathcal{A}(q2)\Big) \;\Leftrightarrow\; \mathrm{val}\Big(\neg\mathcal{A}(q_1) \vee \mathcal{A}(q_2)\Big)$$

$q_1 : \{(?a,\ name,\ ?n)\}$ $\quad$ $q_2 : \{(?a,\ name,\ ?n)\ OPT\ (?a,\ phone,\ ?p)\}$

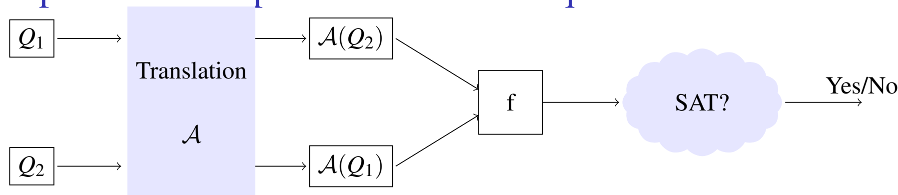
$$\mathcal{A}(q_1) = \underbrace{\xi \wedge \langle\alpha\rangle\mathcal{A}(?a,\ email,\ ?e)}_{\varphi} \wedge \neg\langle\beta\rangle\top$$

$$\mathcal{A}(q_2) = \underbrace{\xi \wedge \langle\alpha\rangle\mathcal{A}(?a,\ email,\ ?e)}_{\varphi} \wedge [\beta]\Big(\underbrace{\xi \wedge \langle\alpha\rangle\mathcal{A}(?a,\ phone,\ ?p) \wedge [\beta]\bot}_{\psi}\Big)$$

$$\mathrm{val}\Big(\neg\mathcal{A}(q_1) \vee \mathcal{A}(q_2)\Big) \;\Leftrightarrow\; \mathrm{unsat}\Big(\mathcal{A}(q_1) \wedge \neg\mathcal{A}(q_2)\Big)$$

Is there a model for $(\varphi \wedge \neg\langle\beta\rangle\top) \wedge \neg(\varphi \wedge [\beta]\psi)$? NO

Experiments: implementation techniques



Custom implementation of K-SOLVER [Chapter 5]

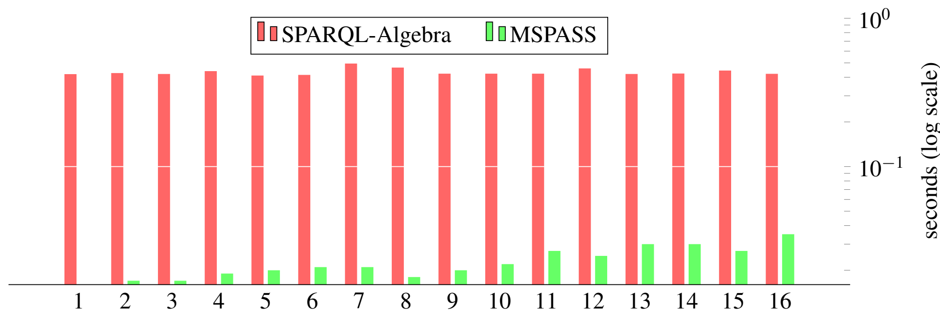
- better understanding of the logical approach
- java implementation with BDDs based on [pan-et-al.:2005]
- Solves the problem

MSPASS [Hustadt-et-al:2000]

- Optimized solver supporting K logic
- Solves the problem much faster [finding of Chapter 6]

Experiments

Queries taken either from third-party benchmarks (BERLIN Bench) or that we have automatically generated.



Result: the use of our translation and the MSPASS solver offers 10x performance gain compared to SPARQL Algebra (the closest competitor).

Conclusion

Main contributions:

1. A translation of well-designed (OPT-AND) queries in terms of formulas expressed in the modal logic and a formulation of the containment problem as unsatisfiability in logic.
2. implementation of the approach and experiments with benchmarks;
3. a finding: the use of the MSPASS solver in this context provides a 10x performance gain compared to SPARQL Algebra.

Perspectives:

- Study extensions to support constraints

Perspectives: containment under schema

Motivation: Reduction of the input graphs

Let $\mathcal{C} = \{c_1, \ldots, c_n\}$ be a finite set of constraints.

- We define a function η that encode our set of constraints as modal formulæ

$$\eta(\mathcal{C}) = \eta(c_1) \wedge \ \ldots \wedge \ \eta(c_n)$$

- Let q_1 and q_2 be two well-designed queries and C be a set of axioms, we should have

$$q_1 \sqsubseteq_{\mathcal{C}} q_2 \ \Leftrightarrow \ \eta(\mathcal{C}) \ \wedge \mathcal{A}(q_1) \wedge \neg \mathcal{A}(q_2)$$

Open Questions:

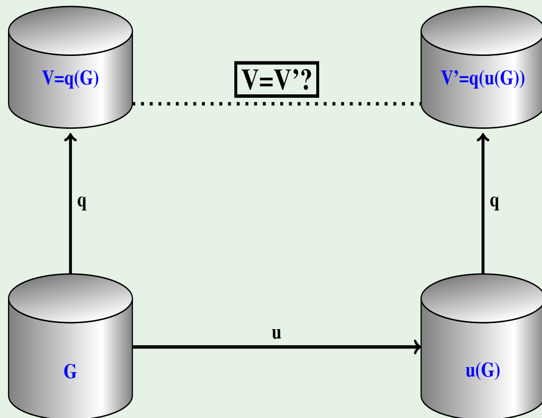
1. Complexity?
2. What kind of schemas can be captured by K?
3. To which extent my performance gains transfer to this setting?
4. How to generalize this approach to the query-update independence problem [Guido-et-al:2015]?

Any Questions?

Query-Update Independence

The release of an update language for SPARQL introduces a setting where one might now study the problem of determining independence between SPARQL queries and SPARQL Updates

Example



Condition of Independence

Definition (Condition for independence)

Let $\mathcal{Q}$ be the set of all triple patterns that appear in the query pattern of the query q and $\mathcal{U}$ be the set of all quad patterns that appear in the DeleteClause and InsertClause of an update u:

$$\forall G.\ [\![q]\!]_G = [\![q]\!]_{u(G)} \Leftrightarrow\ \mathcal{Q} \cap \mathcal{U} = \emptyset$$

where $\mathcal{Q} \cap \mathcal{U} = \emptyset$ is defined as follows:

$$\forall t_q \in \mathcal{Q},\ \forall t_u \in \mathcal{U}, t_q \neq t_u$$

and $\quad t_q \neq t_u \Leftrightarrow \forall \varrho_q \in [\![t_q]\!]_G, \forall \varrho_u \in [\![t_u]\!]_G, \varrho_q(t_q) \neq \varrho_u(t_u).$

Example

Example

$G = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$

$$q = \begin{array}{ll} SELECT & ?a\ ?n\ ?p \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ phone,\ ?p)\Big\} \end{array}$$

$[\![q]\!]_G = \{\varrho_1 = \{?a \rightarrow A_1, ?n \rightarrow Joe, ?p \rightarrow 1111\}, \varrho_2 = \{?a \rightarrow A_4, ?n \rightarrow John, ?p \rightarrow 4444\}\}$

Example

Example

$$G = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

$$q = \begin{array}{ll} SELECT & ?a\ ?n\ ?p \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ phone,\ ?p)\Big\} \end{array}$$

$$[\![q]\!]_G = \{\varrho_1 = \{?a \to A_1, ?n \to Joe, ?p \to 1111\}, \varrho_2 = \{?a \to A_4, ?n \to John, ?p \to 4444\}\}$$

$$u = \begin{array}{ll} DELETE & (?a,\ name,\ ?n) \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ fax,\ ?f)\Big\} \end{array}$$

$$u(G) = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

Example

Example

$$G = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

$$q = \begin{array}{ll} SELECT & ?a\ ?n\ ?p \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ phone,\ ?p)\Big\} \end{array}$$

$$[\![q]\!]_G = \{\varrho_1 = \{?a \to A_1, ?n \to Joe, ?p \to 1111\}, \varrho_2 = \{?a \to A_4, ?n \to John, ?p \to 4444\}\}$$

$$u = \begin{array}{ll} DELETE & (?a,\ name,\ ?n) \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ fax,\ ?f)\Big\} \end{array}$$

$$u(G) = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

$$[\![q]\!]_{u(G)} = \{\varrho_1 = \{?a \to A_1, ?n \to Joe, ?p \to 1111\}\}$$

$[\![q]\!]_{u(G)} \neq [\![q]\!]_G \Leftrightarrow$ q and u are dependent

Example

Example

$$G = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

$$q = \begin{array}{ll} SELECT & ?a\ ?n\ ?p \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ phone,\ ?p)\Big\} \end{array}$$

$$[\![q]\!]_G = \{\varrho_1 = \{?a \to A_1, ?n \to Joe, ?p \to 1111\}, \varrho_2 = \{?a \to A_4, ?n \to John, ?p \to 4444\}\}$$

$$\widehat{u} = \begin{array}{ll} DELETE & (?a,\ fax,\ ?n) \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ fax,\ ?f)\Big\} \end{array}$$

$$\widehat{u}(G) = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444)\Big\}$$

Example

Example

$$G = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

$$q = \begin{array}{ll} SELECT & ?a\ ?n\ ?p \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ phone,\ ?p)\Big\} \end{array}$$

$$[\![q]\!]_G = \{\varrho_1 = \{?a \rightarrow A_1, ?n \rightarrow Joe, ?p \rightarrow 1111\}, \varrho_2 = \{?a \rightarrow A_4, ?n \rightarrow John, ?p \rightarrow 4444\}\}$$

$$\widehat{u} = \begin{array}{ll} DELETE & (?a,\ fax,\ ?n) \\ WHERE & \Big\{(?a,\ name,\ ?n)\ AND\ (?a,\ fax,\ ?f)\Big\} \end{array}$$

$$\widehat{u}(G) = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444)\Big\}$$

$[\![q]\!]_{\widehat{u}(G)} = [\![q]\!]_G \Leftrightarrow$ q and u are independent

Approaches

Syntactic checking - We define the function $\underline{Equiv} : T \times T \rightarrow \{true, false\}$ that returns *true* if the pair of terms in input matches, *false* otherwise

$$\underline{Equiv}(t_1, t_2) = \begin{cases} true & \text{if } t_1 \in BV \quad \text{or} \quad t_2 \in BV \quad \text{or} \quad t_1 = t_2 \\ false & \text{otherwise} \end{cases}$$

$$\underline{Equiv}\Big(t_1, t_2\Big) = \underline{Equiv}(s_1, s_2) \;\&\&\; \underline{Equiv}(p_1, p_2) \;\&\&\; \underline{Equiv}(o_1, o_2)$$

What happens in the presence of the following triple pattern in the DeleteClause or in the InsertClause?

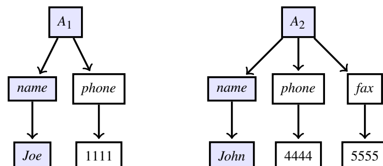
$$\{?s \;\; ?p \;\; ?o\}$$

Approaches

We study the structure of a query and an update in terms of their graph patterns.

$t_1 \equiv A_1$ *name Joe*
$t_2 \equiv A_1$ *phone* 1111
$t_3 \equiv A_2$ *fax* 5555

$t_1 \equiv A_2$ *name John*
$t_2 \equiv A_2$ *phone* 4444



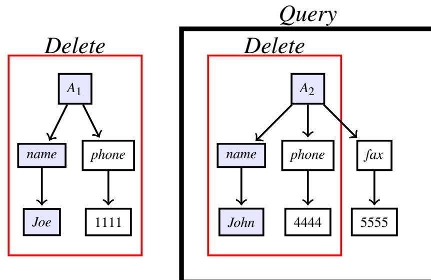
When the containment study concludes that the query is more constrained than the delete operation or the insert operation is more constrained than the query, it is possible to avoid a full re-computation of a query over an updated graph.

Approaches

We study the structure of a query and an update in terms of their graph patterns.

$t_1 \equiv A_1$ *name Joe*
$t_2 \equiv A_1$ *phone* 1111
$t_3 \equiv A_2$ *fax* 5555

$t_1 \equiv A_2$ *name John*
$t_2 \equiv A_2$ *phone* 4444



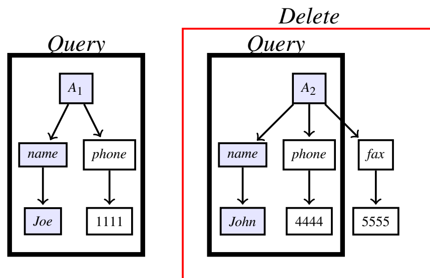
When the containment study concludes that the query is more constrained than the delete operation or the insert operation is more constrained than the query, it is possible to avoid a full re-computation of a query over an updated graph.

$$\begin{aligned} \rightarrow \textit{Delete} = \{t_1, t_2\} &\subseteq \textit{Query} = \{t_1, t_2, t_3\} \\ \textit{Query} = \{t_1, t_2\} &\subseteq \textit{Delete} = \{t_1, t_2, t_3\} \\ \textit{Insert} = \{t_1, t_2\} &\subseteq \textit{Query} = \{t_1, t_2, t_3\} \\ \textit{Query} = \{t_1, t_2\} &\subseteq \textit{Insert} = \{t_1, t_2, t_3\} \end{aligned}$$

Approaches

We study the structure of a query and an update in terms of their graph patterns.

$t_1 \equiv A_1$ *name Joe* $\quad t_1 \equiv A_2$ *name John*
$t_2 \equiv A_1$ *phone* 1111 $\quad t_2 \equiv A_2$ *phone* 4444
$t_3 \equiv A_2$ *fax* 5555



When the containment study concludes that the query is more constrained than the delete operation or the insert operation is more constrained than the query, it is possible to avoid a full re-computation of a query over an updated graph.

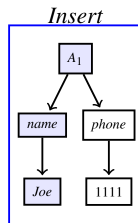
$$\begin{aligned} Delete = \{t_1, t_2\} &\subseteq Query = \{t_1, t_2, t_3\} \\ \rightarrow Query = \{t_1, t_2\} &\subseteq Delete = \{t_1, t_2, t_3\} \\ Insert = \{t_1, t_2\} &\subseteq Query = \{t_1, t_2, t_3\} \\ Query = \{t_1, t_2\} &\subseteq Insert = \{t_1, t_2, t_3\} \end{aligned}$$

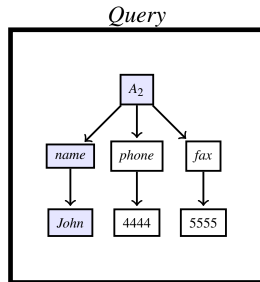
Approaches

We study the structure of a query and an update in terms of their graph patterns.

$t_1 \equiv A_1$ *name Joe*
$t_2 \equiv A_1$ *phone* 1111

$t_1 \equiv A_2$ *name John*
$t_2 \equiv A_2$ *phone* 4444
$t_3 \equiv A_2$ *fax* 5555





When the containment study concludes that the query is more constrained than the delete operation or the insert operation is more constrained than the query, it is possible to avoid a full re-computation of a query over an updated graph.

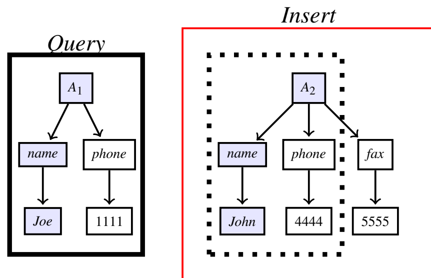
$$\begin{array}{rcl} Delete = \{t_1, t_2\} & \subseteq & Query = \{t_1, t_2, t_3\} \\ Query = \{t_1, t_2\} & \subseteq & Delete = \{t_1, t_2, t_3\} \\ \rightarrow Insert = \{t_1, t_2\} & \subseteq & Query = \{t_1, t_2, t_3\} \\ Query = \{t_1, t_2\} & \subseteq & Insert = \{t_1, t_2, t_3\} \end{array}$$

Approaches

We study the structure of a query and an update in terms of their graph patterns.

$t_1 \equiv A_1$ *name Joe*
$t_2 \equiv A_1$ *phone* 1111
$t_3 \equiv A_2$ *fax* 5555

$t_1 \equiv A_2$ *name John*
$t_2 \equiv A_2$ *phone* 4444



When the containment study concludes that the query is more constrained than the delete operation or the insert operation is more constrained than the query, it is possible to avoid a full re-computation of a query over an updated graph.

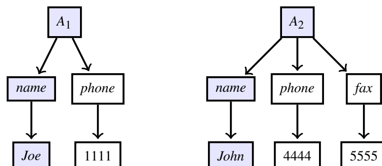
$$
\begin{aligned}
Delete = \{t_1, t_2\} &\subseteq Query = \{t_1, t_2, t_3\} \\
Query = \{t_1, t_2\} &\subseteq Delete = \{t_1, t_2, t_3\} \\
Insert = \{t_1, t_2\} &\subseteq Query = \{t_1, t_2, t_3\} \\
\rightarrow Query = \{t_1, t_2\} &\subseteq Insert = \{t_1, t_2, t_3\}
\end{aligned}
$$

Approaches

We study the structure of a query and an update in terms of their graph patterns.

$t_1 \equiv A_1$ *name Joe*
$t_2 \equiv A_1$ *phone* 1111
$t_3 \equiv A_2$ *fax* 5555

$t_1 \equiv A_2$ *name John*
$t_2 \equiv A_2$ *phone* 4444



When the containment study concludes that the query is more constrained than the delete operation or the insert operation is more constrained than the query, it is possible to avoid a full re-computation of a query over an updated graph.

$Delete = \{t_1, t_2\}$	$\subseteq$	$Query = \{t_1, t_2, t_3\}$
$Query = \{t_1, t_2\}$	$\subseteq$	$Delete = \{t_1, t_2, t_3\}$
$Insert = \{t_1, t_2\}$	$\subseteq$	$Query = \{t_1, t_2, t_3\}$
$Query = \{t_1, t_2\}$	$\subseteq$	$Insert = \{t_1, t_2, t_3\}$

Perspective - Query-Update Independence under Schema

Constraints expressed in a description logic supporting negation are game-changers for the independence problem.

$$G = \Big\{(A_1,\ name,\ Joe), (A_1,\ phone,\ 1111), (A_4,\ name,\ John), (A_4,\ phone,\ 4444), (A_4,\ fax,\ 5555)\Big\}$$

$$\begin{array}{ll} SELECT & ?s \\ WHERE & \Big\{(?s,\ phone,\ 1111)\Big\} \end{array} \qquad \begin{array}{ll} DELETE & (?s,\ ?p,\ ?o) \\ WHERE & \Big\{(A_4,\ phone,\ ?s)\Big\} \end{array}$$

In absence of a schema the query and the update are dependent. The situation changes in presence of the following schema

$$Domain(phone, Person) \qquad Range(phone, Number) \qquad DisjointClasses(Person, Number)$$

The variable $?x$ has type *Person* in the query and type *Number* in the update, hence a contradiction has been detected that should imply independence.

Experiments: The Benchmark

Benchmark

- Queries based on the specific requirements of a real world use case.
- Increasing expected difficulty by using more constructors.
  - OPT-AND constructors;
  - the number of triple patterns;
  - the number of nested OPT constructors;

The Logical Approach

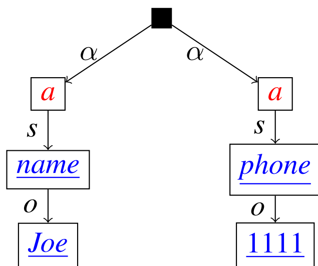
- MSPASS

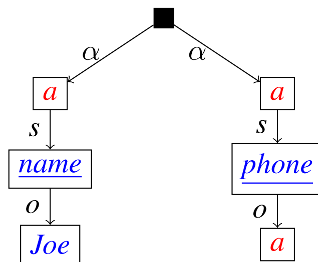
Ad Hoc Approach [Letelier-et-al.:2012]

- SPARQL-Algebra

Encoding of variables

$SELECT\ \ ?a$ $WHERE\ \ \{(?a,\ name,\ Joe)\ AND$ $(?a,\ phone,\ 1111)\}$	$SELECT\ \ ?a$ $WHERE\ \ \{(?a,\ name,\ Joe)\ AND$ $(?a,\ phone,\ ?a)\}$





- Subjects ={}
- Predicates ={*name*, *phone*}
- Objects ={*Joe*, 1111}

$?a \rightarrow a$ is in subject and object position

- $?a \rightarrow a \vee Joe$
- $?a \rightarrow a \vee 1111$