Certified Graph Query Processing

Stefania Dumbrava[1]

IRISA - ENS Rennes - Inria Rennes

EUTypes WG @LambdaDays

February 23rd, 2019

[1] Joint with A. Bonifati (Lyon 1/LIRIS) & E.J.G. Arias (PSL/Mines ParisTech)

Graph Databases

Graphs Topologies are *pervasive* in numerous domains:

- Knowledge Representation and the Semantic Web
- Linked Open Data
- Scientific Repositories (medicine, biology, chemistry)

Graph Databases

Graph Datasets are *readily available* and *continously growing*

- **DBPedia**: multi-domain ontology derived from Wikipedia
- **WikiData**: Wikipedia's openly curated knowledge graph
- **BioRDF**: linked data for the life sciences

Graph Databases

Graph Databases are tailored to store graph-shaped data

- explicit graph model structure
- support *massive*, *connected* data
- better performance w.r.t RDBMSs & NoSQL aggregate stores

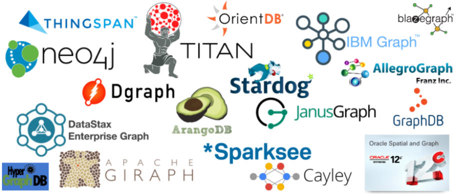


Figure: (Part of the) Graph Database Ecosystem

Graph Database Models

Basic Model – edge-labeled graph

- nodes: abstract entities
- edges: relationships between them

Graph Database Models

Basic Model – edge-labeled graph

- nodes: abstract entities
- edges: relationships between them

Enhanced Models:

- *directionality*: ordered edges – directed graph
- *heterogeneity*: multiple edges & labels – multi-graph
- *expressivity*: multiple node & edge properties – property graph

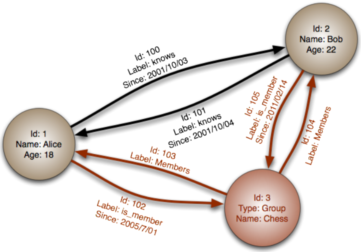


Figure: Graph Model Example

Graph Database Models

Basic Model – edge-labeled graph

- nodes: abstract entities
- edges: relationships between them

Enhanced Models:

- *directionality*: ordered edges – directed graph
- *heterogeneity*: multiple edges & labels – multi-graph
- *expressivity*: multiple node & edge properties – property graph

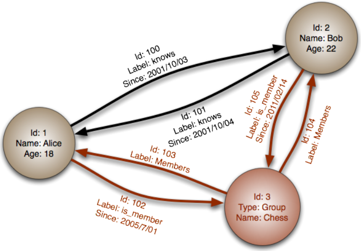


Figure: Graph Model Example

Graph Query Languages

- graph queries: *navigation & label-constrained reachability*
- multiple implementations, various levels of expressivity

GraphQL Gremlin openCypher SPARQL

- *no standard* → raises development & interoperability issues

G-CORE Manifesto: [Angles et. al, 2017]

Find suitable counterpart to SQL in the graph database setting.

Graph Query Languages

- graph queries: *navigation* & *label-constrained reachability*
- multiple implementations, various levels of expressivity

GraphQL Gremlin openCypher SPARQL

- *no standard* $\rightarrow$ raises development & interoperability issues

G-CORE Manifesto: [Angles et. al, 2017]

Find suitable counterpart to SQL in the graph database setting.

Challenge: *expressivity* vs. *tractability* trade-off

- recursion: needed to model graph properties
  - ...bottleneck for graph query engines [Bagan et al., 2017]
- query containment decidability: desirable for optimization
  - ...generally undecidable

Graph Query Languages

- graph queries: *navigation & label-constrained reachability*
- multiple implementations, various levels of expressivity

GraphQL Gremlin openCypher SPARQL

- *no standard* → raises development & interoperability issues

G-CORE Manifesto: [Angles et. al, 2017]

Find suitable counterpart to SQL in the graph database setting.

Foundational Commonality

- all are subsumed by the Datalog language
- zoom-in on a desirable fragment (Regular Datalog)

Datalog Language

Datalog Language

Function-free, *range-restricted* (decidable) Horn logic fragment

Datalog Language

Datalog Language

Function-free, *range-restricted* (decidable) Horn logic fragment

Main Features

- terminating (safety $\rightarrow$ guaranteed for finite set queries)
- declarative (efficient evaluation)
- uniform (relations, views, queries, data dependencies)

Datalog Language

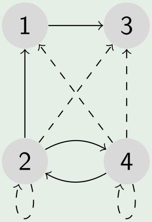
Datalog Language

Function-free, *range-restricted* (decidable) Horn logic fragment

Example: Transitive Closure Computation

$e(1, 3).$
$e(2, 1).$
$e(4, 2).$
$e(2, 4).$

$\mathrm{tc}(X, Y) \leftarrow e(X, Y).$
$tc(X, Y) \leftarrow tc(X, Z), tc(Z, Y).$



Datalog Language

Resurge of Interest in 2010

Datalog 2.0 Manifesto: `http://www.datalog20.org/`

- powerful *abstraction* for querying recursive structures

  → renewed academic interest in emerging domains:

  - *data integration and exchange, security, program analysis, etc.*

- modular, scalable and extensible *programming language*

  → successful industrial applications:

  - *DLV, Exeura, Neotide, Lixto, Dedalus, Clingo, etc.*
  - ...even **full enterprise software stack** powered by Datalog:

LogicBlox

Ensuring Reliability of Datalog Engines

Desideratum

- Formal specification of Datalog languages.
  Blueprint for ongoing standardisation efforts.
- Strong safety guarantees for real-world Datalog-based engines.
  Blueprint for principled (graph) database development.

Mechanical Certification

- **specification**: rigorous definition of *expected behavior*
- **verification**: *observed behavior* preserves invariants
  - e.g., termination, soundness, completeness

$\Rightarrow$ *correct-by-construction* **implementation**

Towards Certifying Commercial Datalog Engines

Long-Term Goal: A Refinement Based Methodology

- high-level formalization suitable for proof development
- mechanization of an efficient implementation
- refinement proofs of their extensional equivalence

Towards Certifying Commercial Datalog Engines

Long-Term Goal: A Refinement Based Methodology

- high-level formalization suitable for proof development
  - key ingredient: **finite model theory**
- mechanization of an efficient implementation
- refinement proofs of their extensional equivalence

Towards Certifying Commercial Datalog Engines

Long-Term Goal: A Refinement Based Methodology

- high-level formalization suitable for proof development
  - key ingredient: **finite model theory**
    - central to Datalog semantics
    - support: **Mathematical Components Library** (MathComp)
- mechanization of an efficient implementation
- refinement proofs of their extensional equivalence

Towards Certifying Commercial Datalog Engines

Mathematical Components Library

- multi-purpose **mathematical theories**
  - relevant libraries for reasoning over *finite types*
  - *finite group theory* (Feit-Thompson classification theorem)
  - *finite set theory* and big operators

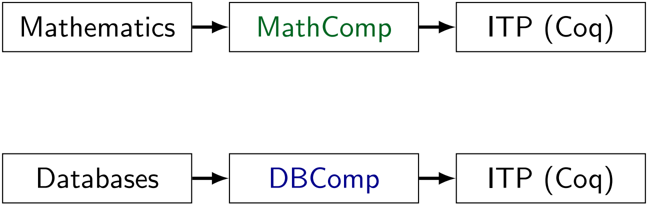
Towards Certifying Commercial Datalog Engines

Mathematical Components Library

- multi-purpose **mathematical theories**
  - relevant libraries for reasoning over *finite types*
  - *finite group theory* (Feit-Thompson classification theorem)
  - *finite set theory* and big operators
- **SSReflect** tactic language
  - generic *reflection* mechanism
  - *succinct* proof scripts
  - *compositional* proof development

Certified Database Components

- Similar to Mathematical Components (MathComp)
- Database Components (DBComp):
  bridge DB Foundations & Interactive Theorem Proving (ITP)



Outline

From Graph Databases to Regular Datalog

How to leverage Datalog to query *graph-shaped* data ?

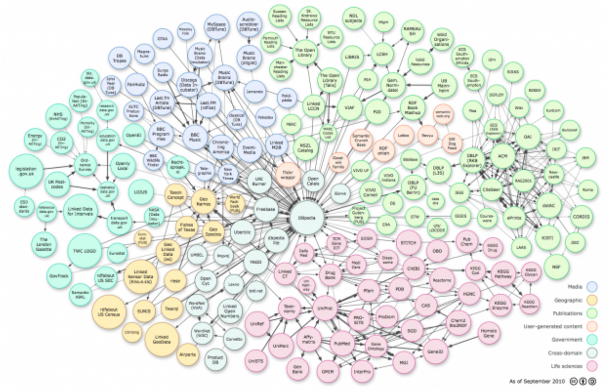


Figure: DBpedia Snapshot

From Graph Databases to Regular Datalog

Graph Databases

V: *finite* set of constants (nodes).
Σ: *finite* set of symbols (edge labels).

Graph Instance $\mathcal{G}$ over Σ:
set of *directed* labeled edges, **E**, where $\mathbf{E} \subseteq \mathbf{V} \times \Sigma \times \mathbf{V}$.

Graph Database $\mathcal{D}(\mathcal{G})$ over $\mathcal{G}$:
$\mathcal{G}$ can be seen as a database $\mathcal{D}(\mathcal{G}) = \{s(n_1, n_2) \mid (n_1, s, n_2) \in \mathbf{E}\}$

Path ρ of length k in $\mathcal{G}$: sequence $n_1 \xrightarrow{s_1} n_2 \ldots n_{k-1} \xrightarrow{s_k} n_k$

Path Label: $\lambda(\rho) = s_1 \ldots s_k \in \Sigma^*$

From Graph Databases to Regular Datalog

Regular Datalog ([Reutter et al., 2017])

- binary **Datalog** limiting recursion to *transitive closure*
  - specify *complex, regular expression patterns*
- *efficient query processsing*
  - highly parallelizable
  - optimizable (decidable query containment)

Regular Datalog : Language Syntax

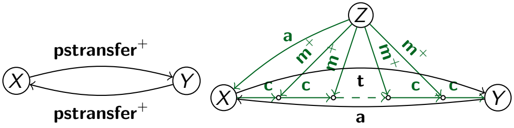
Regular Datalog (RD) Expressions

Terms (Node IDs)	$t ::= x \mid n$	where $x \in \mathbb{V}$, $n \in \mathbf{V}$
Atoms	$A ::= s(t_1, t_2)$	where $s \in \Sigma$
Literals	$L ::= A \mid A^+$	
Conjunctive Body	$B ::= L_1 \wedge \ldots \wedge L_n$	where $n \in \mathbb{N}$
Disjunctive Body	$D ::= B_1 \vee \ldots \vee B_n$	where $n \in \mathbb{N}$
Clauses	$C ::= (t_1, t_2) \leftarrow D$	
Programs	$\Pi ::= \Sigma \to \{C_1, \ldots, C_n\}$	where $n \in \mathbb{N}$

Regular Queries (RQ) over $\mathcal{G}$

- RD-program Π and a distinguished query clause Ω with:
  - head – top-level view (V)
  - body – disjunctive conjunction of Π literals

Example: Fraud Detection Patterns



(a) Potential Fraud $suspect(X, Y)$ (b) Secured Transfer $stransfer(X, Y)$

Figure: Fraud Detection

$$
\begin{array}{lcl}
suspect(X, Y) & \leftarrow & \textbf{pstransfer}^{+}(X, Y), \textbf{pstransfer}^{+}(Y, X) \\
\textbf{pstransfer}(X, Y) & \leftarrow & (\mathbf{t}ransfer + stransfer)(X, Y) \\
stransfer(X, Y) & \leftarrow & \mathbf{a}ccredit(Y, X), secures(X, Y), \mathbf{t}ransfers(X, Y) \\
secures(X, Y) & \leftarrow & (\mathbf{c}onnected \cdot cmonitor^{+} \cdot \mathbf{c}onnected)(X, Y) \\
cmonitor(X, Y) & \leftarrow & \mathbf{c}onnected(X, Y), \mathbf{m}onitor^{+}(Z, X), \mathbf{m}onitor^{+}(Z, Y), \mathbf{a}ccredit(Z, X)
\end{array}
$$

Regular Datalog: Semantics

Interpretations ($\mathcal{G}$)

Modeled as *indexed relations* $(\mathbf{\Sigma} \times \{\Box, +\}) \to \mathcal{P}(\mathbb{C} \times \mathbb{C})$.

Interpretation Well-Formedness (wfG)

$\mathcal{G}(s, +)$ has to correspond to the transitive closure of $\mathcal{G}(s, \Box)$:

$$\text{wfG}(\mathcal{G}) \iff \forall s, \text{is\_closure}(\mathcal{G}(s, \emptyset), \mathcal{G}(s, +))$$

$$\text{is\_closure}(g_s, g_p) \iff \forall (n_1, n_2) \in g_p, \exists \rho \in \mathbf{V}^+, \text{path}(g_s, n_1, \rho) \wedge \text{last}(\rho) = n_2$$

$$\text{path}(g, n_1, \rho) \iff \forall i \in \{1 \ldots |\rho|\}, (n_i, n_{i+1}) \in g$$

Minimal Model

Example

$$\Pi = \begin{cases} R_1(a). \\ R_2(b). \\ R_3(X) \leftarrow R_2(X) \end{cases}$$

- $\{R_1(a), R_2(b), R_3(a), R_3(b)\}$ - valid (trivial) model
- $\emptyset$, $\{R_1(a), R_2(b)\}$ - not models
- $\{R_1(a), R_2(b), R_3(b)\}$ - intended semantics ($MM(P)$)

Intended Model Theoretic Semantics

Datalog programs Π have an unique minimal model $MM(\Pi)$

$$MM(\Pi) \models \Pi \land (\forall M, M \models \Pi \Rightarrow MM(\Pi) \subseteq M)$$

Existence of a Finite Model

Let *adom* be the (finite) set of constants in Π.
Let $\mathbb{B}_\Pi = \{p(c_1, \ldots, c_n) \mid p \in \mathbf{\Sigma},\ c_i \in \text{adom},\ ar(p) = n\}$

Theorem

If Π is safe (all head variables appear in the body) then

$$\mathbb{B}_\Pi \models \Pi$$

Proof.

Let head $\leftarrow$ body $\in \Pi$ and $\nu : \mathbb{V} \rightarrow \mathbb{C}$.
Safety $\Rightarrow \nu(\text{head}) \in \mathbb{B}_\Pi \vee \mathbb{B}_\Pi \not\models \nu(\text{body})$. □

Corollary: Finite Model Property

$\mathbb{B}_\Pi$ is a finite set $\Rightarrow$ minimal models are finite.

Outline

Regular Datalog: Engine Overview

- stratified, single-pass, bottom-up heuristic
- non-recursive (recursion internalized in closure computation)
- supports both *base* and *incremental* inference
- core component: clause evaluation
  - *forward-chain clausal consequence operator* (`fwd_or_clause`)
  - based on a *matching algorithm*
  - corresponds to computing a *nested-loop join*

Regular Datalog: Base Engine

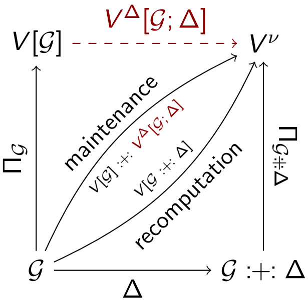
Base Clause Evaluation: Clausal Consequence Operator

For $C \triangleq \Pi(s) \equiv (t_1, t_2) \leftarrow \bigvee_{i=1..n} B_i$,

$T^{\Pi,s}(\mathcal{G}) \equiv \{\sigma(t_1, t_2) \mid \sigma \in \bigcup_{i=1..n} M^{B}_{\mathcal{G}}(B_i)\}$.

Regular Datalog IVM-Engine

Certify Graph Database Incremental View Maintenance (IVM)



$\mathcal{G} \triangleq$ base graph; $\Pi \triangleq$ RD program; $V \triangleq$ top-view; $\Delta \triangleq$ update.

Soundness

If $V[\mathcal{G}] \models \Pi$*, the IVM-engine outputs an incremental view update,* $V^{\Delta}[\mathcal{G};\Delta]$*, such that* $V[\mathcal{G}] :+: V^{\Delta}[\mathcal{G};\Delta] \models \Pi$.

Regular Datalog Updates

Updates

An *update* $\Delta \triangleq (\Delta_+, \Delta_-)$ is a pair of *disjoint* graphs Δ_+, Δ_-.
$\Delta_+ \triangleq$ bulk insertions; $\Delta_+ \triangleq$ bulk deletions.

Update Operations

$$\begin{array}{lcl} \mathcal{G} :+: \Delta & \equiv & \mathcal{G} \setminus \Delta_- \cup \Delta_+ \\ \Delta\{s \to (g_+, g_-)\} & \equiv & (\Delta_+\{s \to g_+\}, \Delta_-\{s \to g_- \setminus g_+\}) \end{array}$$

Incremental Δ-Matching

Compute $V^{\Delta}[\mathcal{G};\Delta]$, such that $V[\mathcal{G} :+: \Delta] = V[\mathcal{G}] :+: V^{\Delta}[\mathcal{G};\Delta]$, via delta programs, *distributing deltas over joins* and *factoring*. (based on [Gupta et al, 1993])

Delta Programs ($\delta(V)$)

For a view V, with $V \leftarrow L_1, \ldots, L_n$, $\delta(V) \triangleq \{\delta_i \mid i \in [1, n]\}$.

Each *delta clause* $\delta_i \triangleq V \leftarrow L_1, \ldots, L_{i-1}, L_i^{\Delta}, L_{i+1}^{\nu}, \ldots, L_n^{\nu}$, where:

$L_j^{\nu} \triangleq$ match L_j with $\mathcal{G} \cup \Delta\mathcal{G}$ atoms with the same symbol as L_j

$L_j^{\Delta} \triangleq$ match L_j with $\Delta\mathcal{G}$ atoms with the same symbol as L_j.

Example: Incremental Δ-Matching

Let $V \triangleq r \bowtie s$, where $V(X, Y) \leftarrow r(X, Z), s(Z, Y)$, r^{Δ} and s^{Δ}.

$V^{\Delta} = (r^{\Delta} \bowtie s) \cup (r \bowtie s^{\Delta}) \cup (r^{\Delta} \bowtie s^{\Delta})$.

$V^{\Delta} = (r^{\Delta} \bowtie s) \cup (r^{\nu} \bowtie s^{\Delta})$, where $r^{\nu} = r \cup r^{\Delta}$.

$V^{\Delta} = V_1^{\Delta} \cup V_2^{\Delta}$, where:

$$\delta_1 : V_1^{\Delta} \leftarrow r^{\Delta}(X, Z), s(Z, Y)$$
$$\delta_2 : V_2^{\Delta} \leftarrow r^{\nu}(X, Z), s^{\Delta}(Z, Y).$$

Regular Datalog: Incremental Engine

Incremental Atom Matching

$$M^{A,m}_{\mathcal{G},\Delta}(a) = (\text{if } m \in \{\mathbf{B},\mathbf{F}\} \text{ then } M^{A}_{\mathcal{G}}(a) \text{ else } \emptyset) \cup (\text{if } m \in \{\mathbf{D},\mathbf{F}\} \text{ then } M^{A}_{\Delta}(a) \text{ else } \emptyset)$$

Incremental Body Matching

For a set of body literals $B \triangleq [L_1, \ldots, L_n]$, generates $B_\Delta = \mathit{body\_mask}(B)$

$$\begin{bmatrix} {L_1}^{\mathbf{D}} & {L_2}^{\mathbf{F}} & \ldots & {L_{n-1}}^{\mathbf{F}} & {L_n}^{\mathbf{F}} \\ {L_1}^{\mathbf{B}} & {L_2}^{\mathbf{D}} & \ldots & {L_{n-1}}^{\mathbf{F}} & {L_n}^{\mathbf{F}} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ {L_1}^{\mathbf{B}} & {L_2}^{\mathbf{B}} & \ldots & {L_{n-1}}^{\mathbf{B}} & {L_n}^{\mathbf{D}} \end{bmatrix}$$

Incremental Clausal Maintenance Operator

$$T^{\Pi,s}_{\mathcal{G},\text{supp}}(\Delta) = \begin{cases} T^{\Pi,s}(\mathcal{G} :+: \Delta), & (s \notin \text{supp}) \vee (\Delta_- \cup D) \neq \emptyset \\ \bigcup_{B_m \in B_\Delta} M^{B}_{\mathcal{G},\Delta}(B_m), & \text{otherwise} \end{cases}$$

Regular Datalog: Engine Overview

```
Fixpoint fwd_program Π G supp Δ Σ▷ Σ◁ : edelta :=
  match Σ◁ with
  | [::]       => Δ
  | [:: s & ss] =>
    let (arg, body) := Π s                            in
    let Δ′ := fwd_or_clause G supp Δ s arg body in
    let Δ′ := compute_closures G Δ′ s                 in
    fwd_program Π G supp Δ′ (s ∪ s+ ∪ Σ▷) ss
```

Outline

Regular Datalog: Stratification Conditions

Stratified Programs

A program Π is *stratified* if: there exists a mapping $\sigma : \mathbf{\Sigma} \rightarrow [1, n]$ such that, for all s in $\mathbf{\Sigma}$, the $\Pi(s)$ clause $(t_1, t_2) \leftarrow B$ satisfies:

$$\max_{r \in \mathrm{sym}(B)} \sigma(r) < \sigma(s)$$

Well-Formed Program Slices

A symbol set Σ is a *well-formed slice* of Π if:

$$\text{for all } s \text{ in } \Sigma,\ \mathrm{sym}(\Pi(s)) \subseteq \Sigma$$

Regular Datalog: Engine Characterization

Theorem (Soundness)

- Π – *a safe, stratifiable, Regular Datalog program*
- $\boldsymbol{\Sigma}$ – *its set of symbols*
- $\mathcal{G}$ – *a graph instance*
- Δ – *an update*

The IVM-engine cumulatively processes symbols in $\boldsymbol{\Sigma}$*, such that if:*

- *the already processed symbols,* $\Sigma_{\triangleright}$*, are a well-formed* Π*-slice*
- Δ *only modifies* $\Sigma_{\triangleright}$*, i.e.,* $\mathrm{sym}(\Delta) \subseteq \Sigma_{\triangleright}$
- $\mathcal{G} :+: \Delta \models_{\Sigma_{\triangleright}} \Pi$

Then, it outputs Δ_O*, such that* $\mathcal{G} :+: \Delta_O \models_{\boldsymbol{\Sigma}} \Pi$.

Key Lemmas (I/II)

Lemma (Clause Modularity Satisfaction)

Assume $s \notin \mathrm{sym}(\Delta)$ *and also* $\mathrm{sym}(C) \cap \mathrm{sym}(\Delta) = \emptyset$. *Then:*

$$\mathcal{G} \mathrel{:+:} \Delta \models_s C \iff \mathcal{G} \models_s C.$$

Lemma (Program Modularity Satisfaction)

Assume Σ *a well-formed slice of* Π *and* $s \notin \Sigma$. *Let* $\Delta' = (\Delta'_+, \Delta'_-)$, *where* $\Delta'_+ = \Delta_+ \cup \{s(t_1, t_2) \mid (t_1, t_2) \in g\}$ *and* $\Delta'_- = \Delta_- \setminus \{s(t_1, t_2) \mid (t_1, t_2) \in g\}$. *Then:*

$$\mathcal{G} \mathrel{:+:} \Delta' \models_{\{s\} \cup \Sigma} \Pi \iff \mathcal{G} \mathrel{:+:} \Delta' \models_s \Pi(s) \wedge \mathcal{G} \mathrel{:+:} \Delta \models_\Sigma \Pi$$

Key Lemmas (II/II)

Lemma (Clausal Maintenance Soundness)

Assume: $\Pi(s)$ *is a safe clause,* $\mathcal{G} \models_{\Sigma} \Pi$*;* $\Sigma_{\triangleright}$ *is well-formed wrt closures;* $\Sigma_{\triangleright}$ *is a well-formed slice of* Π*;* $s \notin \Sigma_{\triangleright}$*;* $\mathrm{sym}(\Pi(s)) \subseteq \Sigma_{\triangleright}$*;* $\mathrm{sym}(\Delta) \subseteq \Sigma_{\triangleright}$*;* $\mathcal{G} :+: \Delta \models_{\Sigma_{\triangleright}} \Pi$*.*

Then: $\mathcal{G} :+: \Delta_s \models_{\{s\} \cup \Sigma_{\triangleright}} \Pi$, *where* $\Delta_s = T^{\Pi,s}_{\mathcal{G},\mathrm{supp}}(\Delta)$.

Lemma (Δ-Body Matching Soundness)

Let B *a conjunctive body;* σ *a substitution.*
Assume $\mathrm{sym}(B) \cap \mathrm{sym}(\Delta_{-}) = \emptyset$ *(no deletions scheduled for* B*).*

Then: for all $\sigma \in M^{B}_{\mathcal{G},\Delta}(B)$, *there exists* $\overline{B}$, *s.t* $\sigma(B) = \overline{B}$.

Experiments

Goal: confirm extracted engine's IVM runtime $<$ its FVM runtime

Setting:

- gMark synthetic datasets and query workloads:
  - WD, the Waterloo SPARQL Diversity Test Suite (Wat-Div)
  - SNB, the LDBC Social Network Benchmark
- schema size: $|supp(\mathcal{G})| = 82$ (WD), $|supp(\mathcal{G})| = 27$ (SNB)
- instance & workload sizes: $|\mathcal{G}| = 1K$, $|\mathcal{W}| = 10$ UC2RPQ
- $\rho_{supp} = \frac{|supp(\Delta_+)|}{|supp(\mathcal{G})|} \in \{0.05, 0.1, 0.15, 0.2, 0.25\}$
- $\rho = \frac{|\Delta_+|}{|\mathcal{G}'|} * 100$
- Time Gain $=$ FVM $-$ IVM, Ratio Gain $= 100 - \frac{100*\text{IVM}}{\text{FVM}}$

Experiments

ρ_{supp}	ρ	FVM	**IVM**	Time Gain	Ratio Gain
0.05	1.4%	558.7	484.75	73.95	13.23%
0.1	3.67%	561.89	472.7	89.19	15.87%
0.15	17.93%	562.67	475.96	86.71	15.41%
0.2	9.7%	562.13	476.4	85.73	15.25%
0.25	18.26%	563.4	482.64	80.76	14.33%

Table: $\mathcal{W}_{WD}$ Runtimes (ms) for Varying Support Update Size (ρ_{supp})

ρ_{supp}	ρ	FVM	**IVM**	Time Gain	Ratio Gain
0.05	10.89%	18.75	10.88	7.87	41.97%
0.1	19.3%	17.77	10.55	7.22	40.63%
0.15	10.77%	17.55	11.68	5.82	33.25%
0.2	26.09%	17.17	11.71	5.46	31.79%
0.25	28.34%	14.71	11	3.71	25.22%

Table: $\mathcal{W}_{SNB}$ Runtimes (ms) for Varying Support Update Size (ρ_{supp})

Experiments - Insights

- *absolute time gain (ms)* of running IVM vs. FVM:
  always > 0
- *relative ratio gain (%)* is always better for sparser graphs
  SNB runtimes (less dense) $<<$ WD runtimes (very dense)
- engine works best on bulk updates with small support size
  symbol-level maintenance granularity

Outline

Main Results

- *certified graph query evaluation & maintenance engine*
  - 1062 loc (definitions) + 734 loc (proofs)
  - extracted OCaml engine tested on realistic graph databases
- *machine-checked proofs* of foundational database results
  - mathematical representation of core engine components
- promising to certify a *graph query language standard*

Angela Bonifati, **Stefania Dumbrava**, Emilio Jesus Gallego Arias
Certified Graph View Maintenance with Regular Datalog.
Th. and Practice of Logic Programming, 18(3-4):372–389, 2018.

https://github.com/VerDILog/

Related Work

- Incremental Graph Computation for RPQ
  [Fan et al. 2017]
- Certifying SQL Semantics
  [Chu et al. 2017], [Benzaken et al 2019]
- Verified Relational Algebra Query Compilers
  [Auerbach et al 2017]
- Verified Relational Data Model
  [Benzaken et al 2014]
- **Certified Standard and Stratified Datalog Engines**
  [Dumbrava, 2016], [Benzaken et al 2017]

Contributions

- **Language Formalization**
  (*syntax* + *finite model-theoretic semantics*)
  - *new* parametric, normalized, indexed representation
  - *new* core theory of updates
  - *first* certified graph query language
- **Inference Engine Mechanization**
  (*evaluation* + *maintenance*)
  - among *early contributions* in graph view maintenance
  - most mainstream commercial engines do not provide concepts for defining graph views/maintenance
- **Soundness Certification**
  (proof that the engine output is *correct*)
  - compact, compositional proofs → *limited effort* + *reusability*
  - *correct-by-construction* engine executable on *realistic* graphs

References

- [Benzaken et al., 2014] Benzaken, V., Contejean, E, **Dumbrava, S**. 2014. A Coq Formalization of the Relational Data Model. In *ESOP*, 189–208.
- [Dumbrava, 2016] **Dumbrava, S**. 2016. A Coq Formalization of Relational and Deductive Databases - and Mechanizations of Datalog. *PhD Thesis*, University of Paris-Saclay, France
- [Benzaken et al., 2017] Benzaken, V., Contejean, E, **Dumbrava, S**. 2017. Certifying Standard and Stratified Datalog Inference Engines in SSReflect. In *ITP*, 171–188.
- [Dumbrava al., 2018] **Dumbrava, S.**, Bonifati, A., Nazabal, A., Vuillemont, R. 2018. Approximate Evaluation of Complex Queries on Property Graphs. (under submission)
- [Bonifati al., 2018] Bonifati, A., **Dumbrava, S**. 2018. Graph Queries: From Theory to Practice. In Sigmod Record 47(4) (to appear)

Additional References

- [Angles et al., 2017] Angles, R., Arenas, M., Barcelo, P., Hogan, A. Reutter, J.L., and Vrgoc, D. 2017. Foundations of Modern Query Languages for Graph Databases. In *ACM Comput. Surv.*. Vol.50. 68:1–68:40.
- [Auerbach et al., 2017] Auerbach, J. S., Hirzel, M., Mandel, L., Shinnar, A., and Siméon, J. 2017. Handling Environments in a Nested Relational Algebra with Combinators and an Implementation in a Verified Query Compiler. In *SIGMOD*. 1555–1569.
- [Bagan et al., 2016] Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., and Advokaat, N. 2017. gMark: Schema-driven generation of graphs and queries. *IEEE Transactions on Knowledge and Data Engineering 29*, 856–869.
- [Gupta et al., 1993] Gupta, A., Mumick, I.S., Subrahmanian, V.S. 1993. Maintaining Views Incrementally. In *Sigmod Rec. 22*, 157–166.
- [Reutter et al., 2017] Reutter, J.L., Romero, M., and Vardi, M.Y. 2017. Regular Queries on Graph Databases. In *Theory of Computing Systems 61*, 31–83.

Coq Specification: Syntax (Extra)

```
Variables (V Σ : finType).

Inductive L := □ | +.
Inductive egraph := EGraph of {set V * V}.
Inductive lrel  := LRel of {ffun Σ * L -> egraph}

Record atom     := Atom   { syma : Σ; arga : T * T }.
Record lit      := Lit    { tagl : L; atoml: atom }.
Record cbody    := CBody  { litb : seq lit }.
Record clause := Clause { headc : T * T; bodyc : seq cbody }.

Inductive program := Program of {ffun Σ -> clause T Σ L}.
```

Regular Datalog: Semantics (Extra)

Literal Satisfaction

For $L \triangleq s^l(n_1, n_2)$, $\mathcal{G} \models L \iff (n_1, n_2) \in \mathcal{G}(s, l)$.

Clause Satisfaction

For $C \triangleq (t_1, t_2) \leftarrow (L_{1,1} \wedge \ldots \wedge L_{1,n}) \vee \ldots \vee (L_{m,1} \wedge \ldots \wedge L_{m,n})$,

$\mathcal{G} \models_s L \iff \forall \eta, \bigvee_{i=1..m}(\bigwedge_{j=1..n} \mathcal{G} \models \eta(L_{i,j})) \Rightarrow \mathcal{G} \models \eta(s(t_1, t_2))$.

Program Satisfaction

For $\Pi \triangleq \Sigma \rightarrow \{C_1, \ldots, C_n\}$, $\mathcal{G} \models_\Sigma \Pi \iff \forall\ s \in \Sigma, \mathcal{G} \models_s \Pi(s)$.