Certified Graph Query Processing

Stefania Dumbrava

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)
Graphs Topologies are pervasive in numerous domains:

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

- *no standard* $\rightarrow$ 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 → 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

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

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

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

**Main Features**

- **terminating** (safety → guaranteed for finite set queries)
- **declarative** (efficient evaluation)
- **uniform** (relations, views, queries, data dependencies)
### Datalog Language

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

### Example: Transitive Closure Computation

\[
\begin{align*}
e(1, 3). \\
e(2, 1). \\
e(4, 2). \\
e(2, 4). \\
tc(X, Y) &\leftarrow e(X, Y). \\
tc(X, Y) &\leftarrow tc(X, Z), tc(Z, Y).
\end{align*}
\]
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

⇒ *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

1. Introduction
2. Regular Datalog
3. Regular Datalog Engine
4. Soundness
5. Conclusions
From Graph Databases to Regular Datalog

How to leverage Datalog to query graph-shaped data?

Figure: DBpedia Snapshot
Graph Databases

$\mathbf{V}$: finite set of constants (nodes).
$\Sigma$: finite set of symbols (edge labels).

**Graph Instance $\mathcal{G}$ over $\Sigma$:**
set of directed labeled edges, $\mathbf{E}$, 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 $\rho$ 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 processing**
  - highly parallelizable
  - optimizable (decidable query containment)
Regular Datalog : Language Syntax

Regular Datalog (RD) Expressions

Terms (Node IDs) \[ t ::= x \mid n \quad \text{where } x \in \mathbb{V}, \ n \in \mathbb{V} \]

Atoms \[ A ::= s(t_1, t_2) \quad \text{where } s \in \Sigma \]

Literals \[ L ::= A \mid A^+ \]

Conjunctive Body \[ B ::= L_1 \land \ldots \land L_n \quad \text{where } n \in \mathbb{N} \]

Disjunctive Body \[ D ::= B_1 \lor \ldots \lor B_n \quad \text{where } n \in \mathbb{N} \]

Clauses \[ C ::= (t_1, t_2) \leftarrow D \]

Programs \[ \Pi ::= \Sigma \rightarrow \{C_1, \ldots, C_n\} \quad \text{where } n \in \mathbb{N} \]

Regular Queries (RQ) over \( G \)

- RD-program \( \Pi \) and a distinguished query clause \( \Omega \) with:
  - head – top-level view \( (V) \)
  - body – disjunctive conjunction of \( \Pi \) literals
Example: Fraud Detection Patterns

(a) Potential Fraud suspect($X, Y$)

(b) Secured Transfer stransfer($X, Y$)

**Figure:** Fraud Detection

\[
suspect(X, Y) \leftarrow ptransfer^+(X, Y), ptransfer^+(Y, X)
\]

\[
ptransfer(X, Y) \leftarrow (transfer + stransfer)(X, Y)
\]

\[
stransfer(X, Y) \leftarrow accredit(Y, X), secures(X, Y), transfers(X, Y)
\]

\[
secures(X, Y) \leftarrow (connected \cdot cmonitor^+ \cdot connected)(X, Y)
\]

\[
cmonitor(X, Y) \leftarrow connected(X, Y), monitor^+(Z, X), monitor^+(Z, Y), accredit(Z, X)
\]
Regular Datalog: Semantics

Interpretations ($\mathcal{G}$)

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

Interpretation Well-Formedness (wfG)

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

- $\text{wfG}(\mathcal{G}) \iff \forall s, \text{is\_closure}(\mathcal{G}(s, \emptyset), \mathcal{G}(s, +))$
- $\text{is\_closure}(g_s, g_p) \iff \forall (n_1, n_2) \in g_p, \exists \rho \in \mathbf{V}^+, \text{path}(g_s, n_1, \rho) \land \text{last}(\rho) = n_2$
- $\text{path}(g, n_1, \rho) \iff \forall i \in \{1 \ldots |\rho|\}, (n_i, n_{i+1}) \in g$
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 (\( MM(\Pi) \))

Intended Model Theoretic Semantics

Datalog programs \( \Pi \) 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 \( \text{adom} \) be the (finite) set of constants in \( \Pi \).
Let \( \mathcal{B}_\Pi = \{ p(c_1, \ldots, c_n) \mid p \in \Sigma, \ c_i \in \text{adom}, \ \text{ar}(p) = n \} \)

**Theorem**

*If \( \Pi \) is safe (all head variables appear in the body) then* \( \mathcal{B}_\Pi \models \Pi \)

**Proof.**

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

**Corollary: Finite Model Property**

\( \mathcal{B}_\Pi \) is a finite set \( \Rightarrow \) minimal models are finite.
Outline

1. Introduction
2. Regular Datalog
3. Regular Datalog Engine
4. Soundness
5. Conclusions
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 $C \triangleq \Pi(s) \equiv (t_1, t_2) \leftarrow \bigvee_{i=1..n} B_i$,

$T^{\Pi,s}(G) \equiv \{\sigma(t_1, t_2) \mid \sigma \in \bigcup_{i=1..n} M_G^B(B_i)\}.$
Certify Graph Database Incremental View Maintenance (IVM)

\[ V[\mathcal{G}] \rightarrow V' \]

\[ \Pi, \Delta \]

\[ \mathcal{G} \triangleq \text{base graph}; \quad \Pi \triangleq \text{RD program}; \quad V \triangleq \text{top-view}; \quad \Delta \triangleq \text{update}. \]

**Soundness**

If \( V[\mathcal{G}] \models \Pi \), the IVM-engine outputs an incremental view update, \( V^\Delta[\mathcal{G}; \Delta] \), such that \( V[\mathcal{G}] \vdash V^\Delta[\mathcal{G}; \Delta] \models \Pi \).
Regular Datalog Updates

Updates

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

Update Operations

$\mathcal{G} :+: \Delta \equiv \mathcal{G} \setminus \Delta_- \cup \Delta_+$

$\Delta\{s \rightarrow (g_+, g_-)\} \equiv (\Delta_+\{s \rightarrow g_+\}, \Delta_-\{s \rightarrow g_- \setminus g_+\})$
Incremental $\Delta$-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 $L_j$. 
Let $V \triangleq r \Join s$, where $V(X, Y) \leftarrow r(X, Z), s(Z, Y)$, $r^\Delta$ and $s^\Delta$.

$V^\Delta = (r^\Delta \Join s) \cup (r \Join s^\Delta) \cup (r^\Delta \Join s^\Delta)$.

$V^\Delta = (r^\Delta \Join s) \cup (r^\nu \Join s^\Delta)$, where $r^\nu = r \cup r^\Delta$.

$V^\Delta = V^\Delta_1 \cup V^\Delta_2$, where:

$$
\delta_1 : V^\Delta_1 \leftarrow r^\Delta(X, Z), s(Z, Y)
$$

$$
\delta_2 : V^\Delta_2 \leftarrow r^\nu(X, Z), s^\Delta(Z, Y).
$$
Regular Datalog: Incremental Engine

Incremental Atom Matching

\[ M_{G,\Delta}^{A,m}(a) = (\text{if } m \in \{B, F\} \text{ then } M_{G}^{A}(a) \text{ else } \emptyset) \cup (\text{if } m \in \{D, F\} \text{ then } M_{\Delta}^{A}(a) \text{ else } \emptyset) \]

Incremental Body Matching

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

\[
\begin{bmatrix}
L_1^D & L_2^F & \ldots & L_{n-1}^F & L_n^F \\
L_1^B & L_2^D & \ldots & L_{n-1}^F & L_n^F \\
& \cdots & \cdots & \cdots & \cdots \\
L_1^B & L_2^B & \ldots & L_{n-1}^B & L_n^D
\end{bmatrix}
\]

Incremental Clausal Maintenance Operator

\[ T_{G,\text{supp}}^{\Pi,s}(\Delta) = \left\{ \begin{array}{ll}
T_{\Pi,s}(G :+\Delta), & (s \notin \text{supp}) \lor (\Delta \cup D) \neq \emptyset \\
\bigcup_{B_m \in B_{\Delta}} M_{G,\Delta}^{B}(B_m), & \text{otherwise}
\end{array} \right. \]
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

1. Introduction
2. Regular Datalog
3. Regular Datalog Engine
4. Soundness
5. Conclusions
Regular Datalog: Stratification Conditions

**Stratified Programs**

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

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

**Well-Formed Program Slices**

A symbol set $\Sigma$ is a *well-formed slice* of $\Pi$ if:

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

- \( \Pi \) – a safe, stratifiable, Regular Datalog program
- \( \Sigma \) – its set of symbols
- \( \mathcal{G} \) – a graph instance
- \( \Delta \) – an update

The IVM-engine cumulatively processes symbols in \( \Sigma \), such that if:

- the already processed symbols, \( \Sigma_\triangleright \), are a well-formed \( \Pi \)-slice
- \( \Delta \) only modifies \( \Sigma_\triangleright \), i.e., \( \text{sym}(\Delta) \subseteq \Sigma_\triangleright \)
- \( \mathcal{G} :+\: \Delta \models \Sigma_\triangleright \Pi \)

Then, it outputs \( \Delta_0 \), such that \( \mathcal{G} :+\: \Delta_0 \models \Sigma \Pi \).
Key Lemmas (I/II)

Lemma (Clause Modularity Satisfaction)

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

\[
\mathcal{G} :+: \Delta \models_s C \iff \mathcal{G} \models_s C.
\]

Lemma (Program Modularity Satisfaction)

Assume \( \Sigma \) a well-formed slice of \( \Pi \) and \( s \notin \Sigma \). Let

\[
\Delta' = (\Delta'_+, \Delta'_-), \text{ where } \Delta'_+ = \Delta_+ \cup \{s(t_1, t_2) \mid (t_1, t_2) \in g\} \text{ and } \\
\Delta'_- = \Delta_- \setminus \{s(t_1, t_2) \mid (t_1, t_2) \in g\}.
\]

Then:

\[
\mathcal{G} :+: \Delta' \models_{\{s\} \cup \Sigma} \Pi \iff \mathcal{G} :+: \Delta' \models_s \Pi(s) \land \mathcal{G} :+: \Delta \models_{\Sigma} \Pi
\]
**Key Lemmas (II/II)**

**Lemma (Clausal Maintenance Soundness)**

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

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

**Lemma (Δ-Body Matching Soundness)**

Let $B$ a conjunctive body; $\sigma$ a substitution.

Assume $\text{sym}(B) \cap \text{sym}(\Delta_{\perp}) = \emptyset$ (no deletions scheduled for $B$).

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

---

**Introduction**

**Regular Datalog**

**Regular Datalog Engine**

**Soundness**

**Conclusions**
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: \( |\text{supp}(G)| = 82 \) (WD), \( |\text{supp}(G)| = 27 \) (SNB)
- instance & workload sizes: \( |G| = 10^3 \), \( |\mathcal{W}| = 10 \) UC2RPQ
- \( \rho_{\text{supp}} = \frac{|\text{supp}(\Delta_+)|}{|\text{supp}(G)|} \in \{0.05, 0.1, 0.15, 0.2, 0.25\} \)
- \( \rho = \frac{\lvert\Delta_+\rvert}{\lvert G'\rvert} * 100 \)
- Time Gain = FVM − IVM, Ratio Gain = 100 − \( \frac{100*\text{IVM}}{\text{FVM}} \)
## Experiments

<table>
<thead>
<tr>
<th>$\rho_{\text{supp}}$</th>
<th>$\rho$</th>
<th>FVM</th>
<th>IVM</th>
<th>Time Gain</th>
<th>Ratio Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>1.4%</td>
<td>558.7</td>
<td>484.75</td>
<td>73.95</td>
<td>13.23%</td>
</tr>
<tr>
<td>0.1</td>
<td>3.67%</td>
<td>561.89</td>
<td>472.7</td>
<td>89.19</td>
<td>15.87%</td>
</tr>
<tr>
<td>0.15</td>
<td>17.93%</td>
<td>562.67</td>
<td>475.96</td>
<td>86.71</td>
<td>15.41%</td>
</tr>
<tr>
<td>0.2</td>
<td>9.7%</td>
<td>562.13</td>
<td>476.4</td>
<td>85.73</td>
<td>15.25%</td>
</tr>
<tr>
<td>0.25</td>
<td>18.26%</td>
<td>563.4</td>
<td>482.64</td>
<td>80.76</td>
<td>14.33%</td>
</tr>
</tbody>
</table>

**Table:** $W_{WD}$ Runtimes (ms) for Varying Support Update Size ($\rho_{\text{supp}}$)

<table>
<thead>
<tr>
<th>$\rho_{\text{supp}}$</th>
<th>$\rho$</th>
<th>FVM</th>
<th>IVM</th>
<th>Time Gain</th>
<th>Ratio Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>10.89%</td>
<td>18.75</td>
<td>10.88</td>
<td>7.87</td>
<td>41.97%</td>
</tr>
<tr>
<td>0.1</td>
<td>19.3%</td>
<td>17.77</td>
<td>10.55</td>
<td>7.22</td>
<td>40.63%</td>
</tr>
<tr>
<td>0.15</td>
<td>10.77%</td>
<td>17.55</td>
<td>11.68</td>
<td>5.82</td>
<td>33.25%</td>
</tr>
<tr>
<td>0.2</td>
<td>26.09%</td>
<td>17.17</td>
<td>11.71</td>
<td>5.46</td>
<td>31.79%</td>
</tr>
<tr>
<td>0.25</td>
<td>28.34%</td>
<td>14.71</td>
<td>11</td>
<td>3.71</td>
<td>25.22%</td>
</tr>
</tbody>
</table>

**Table:** $W_{SNB}$ Runtimes (ms) for Varying Support Update Size ($\rho_{\text{supp}}$)
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

1. Introduction
2. Regular Datalog
3. Regular Datalog Engine
4. Soundness
5. Conclusions
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.

[https://github.com/VerDIILog/](https://github.com/VerDIILog/)
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


Additional References


Variables ($V \Sigma : finType$).

**Inductive** $L := \square \mid +$.  
**Inductive** $egraph := \text{EGraph of } \{\text{set } V * V\}$.  
**Inductive** $lrel := \text{LRel of } \{\text{ffun } \Sigma * L \to egraph\}$

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

**Inductive** $program := \text{Program of } \{\text{ffun } \Sigma \to clause T \Sigma L\}$. 
### Literal Satisfaction

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

### Clause Satisfaction

For $C \triangleq (t_1, t_2) \leftarrow (L_{1,1} \land \ldots \land L_{1,n}) \lor \ldots \lor (L_{m,1} \land \ldots \land L_{m,n})$, $G \models_s L \iff \forall \eta, \bigvee_{i=1\ldots m}(\land_{j=1\ldots n} G \models \eta(L_{i,j})) \Rightarrow G \models \eta(s(t_1, t_2))$.

### Program Satisfaction

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