Mechanically Verified Graph Query Processing

Stefania Dumbrava

ENSIIE & Samovar Laboratory, Évry, France

Joint work with Angela Bonifati (U. Lyon 1) & Emilio Gallego (INRIA πr²)

IMDEA Software Institute - Invited Talk

June 9th, 2020
Graph Data Models

Figure: Interconnected Data

Figure: Basic & Property Graph Data Models
Relational vs. Graph Databases
Graph Queries

- allow extracting & manipulating:
  - particular *labels* or node/edge property *values*
  - *graph patterns* (label-constrained reachability)
Graph Query Example (I)

What are the name of the actors that played in "V for Vendetta"?

Relational Query (SQL)

```
SELECT name FROM Actor
LEFT JOIN ActorMovie
    ON Actor.Id = ActorMovie.ActorId
LEFT JOIN Movie
    ON Movie.Id = ActorMovie.MovieId
WHERE Movie.name = "V for Vendetta"
```

Graph Query (Cypher)

```
MATCH (a:Actor)-[a:ACTED_IN]-(m:Movie)
WHERE m.name = "V for Vendetta"
RETURN a.name
```
$ MATCH (a) -[:ACTED_IN]->(m) <-[:PRODUCED]-(p) -[:PRODUCED]->(n) {title: "V for Vendetta"} ) RETURN *
Graph Database Ecosystem

GRAPH TECHNOLOGY LANDSCAPE 2019

https://graphaware.com/graphaware/2019/02/01/graph-technology-landscape.html
Graph Database Ecosystem

...no standard adopted yet $\rightarrow$ development & interoperability issues

- G-CORE Manifesto$^1$
- GQL Proposal (in-development ISO project as of 2019)

$\Rightarrow$ good use-case for employing formal methods

---

This Talk

Our Work

A machine-verified *incremental* graph query engine.
Our Work

A machine-verified *incremental* graph query engine.

- ...relying on the Coq proof assistant
The Coq Proof Assistant

Functionalities

- *developing* functional programs
- *specifying* logical properties
- *proving* theorems interactively
- *extracting* executable programs $\Rightarrow$ *correct-by-construction code*
The Coq Proof Assistant

### Functionalities

- *developing* functional programs
- *specifying* logical properties
- *proving* theorems interactively
- *extracting* executable programs \(\Rightarrow\) **correct-by-construction code**

### Research Projects and Applications

- certified compilers (**Compcert**)
- formalized mathematics (**Mathematical Components**)
- full-stack certifications (**DeepSpec NSF Project**)
- industrial use (**Airbus, Intel, Chrome, Amazon**)
This Talk

Our Work

A machine-verified *incremental* graph query engine.

- ...relying on the Coq proof assistant

Possible Applications

⇒ *formal testing* of commercial graph query engines

- oracle: the correct-by-construction formal engine specification
  - high-level formalization suitable for proof development
- tests: check the extensional equivalence of query results

Other: *formal certification* of query execution traces, *formal verification* of efficient implementations
This Talk

Our Work

A machine-verified *incremental* graph query engine.

• ...relying on the Coq proof assistant

• ...for regular queries\(^1\) (closure operators as primitive)
  - expressive & tractable\(^2\)
    ⇒ amenable to *graph querying*
  - logic-based language
    ⇒ amenable to *formal verification*

---

\(^1\) Moshe Y. Vardi: *A Theory of Regular Queries*. PODS 2016: 1-9

Datalog

Background

- late 80’s: interplay between AI, LP and DB theory
- extending relational databases with inference
  → *deductive databases*
Datalog

Background

- late 80's: interplay between AI, LP and DB theory
- extending relational databases with inference
  → deductive databases

Expressivity

- logic programming without function symbols
- consists of facts and rules
- finite model semantics
- sound, complete and terminating evaluation
  → bottom-up iteration of a fixpoint operator
Proliferation of highly-interconnected data $\rightarrow$ resurge of interest

**Datalog 2.0 Manifesto:**  [http://www.datalog20.org/](http://www.datalog20.org/)

**Theoretical Applications**

- **powerful abstraction** for querying recursive structures
  $\rightarrow$ renewed academic interest in emerging domains

- **foundational framework** for query languages
  $\rightarrow$ complexity analysis of new (graph query) fragments
Proliferation of highly-interconnected data $\rightarrow$ resurge of interest

**Datalog 2.0 Manifesto:**  [http://www.datalog20.org/](http://www.datalog20.org/)

**Practical Applications**

- modular, scalable and extensible *programming language*
  $\rightarrow$ successful industrial implementations

- expressive *reasoning tool*
  $\rightarrow$ (*graph*) knowledge management & analytics
Assume we have the following simple graph database instance:

...and that we want to compute *all the pairs of potential friends*.
Datalog Example

knows(Alice, Bob)
contacted(Bob, Eve)
follows(Alice, Tim)
follows(Tim, Bill)
follows(Bob, Bill)

\[ p\text{friends}(X, Y) \leftarrow \text{knows}(X, Y), \text{connected}(X, Z), \text{connected}(Y, Z) \]
\[ p\text{friends}(X, Y) \leftarrow \text{contacted}(X, Y) \]
\[ \text{connected}(X, Y) \leftarrow \text{follows}(X, Z), \text{connected}(Z, Y) \]
\[ \text{connected}(X, Y) \leftarrow \text{follows}(X, Y) \]

\( X \text{ and } Y \text{ are potential friends (pfriends)} \)
\( \text{if } X \text{ contacted } Y \)
\( \text{or } X \text{ knows } Y \text{ and both are connected to a common person} \)

\( X \text{ is connected to } Y \)
\( \text{if } X \text{ is linked to } Y \text{ through a chain of followers} \)
Datalog Example

knows(Alice, Bob)
contacted(Bob, Eve)
follows(Alice, Tim)
follows(Tim, Bill)
follows(Bob, Bill)

pfriends(X, Y) ← knows(X, Y), connected(X, Z), connected(Y, Z)
pfriends(X, Y) ← contacted(X, Y)
connected(X, Y) ← follows(X, Z), connected(Z, Y)
connected(X, Y) ← follows(X, Y)
Datalog Example

knows(Alice, Bob)
contacted(Bob, Eve)
follows(Alice, Tim)
follows(Tim, Bill)
follows(Bob, Bill)

\[
\begin{align*}
p\text{friends}(X, Y) & \leftarrow \text{knows}(X, Y), \text{connected}(X, Z), \text{connected}(Y, Z) \\
p\text{friends}(X, Y) & \leftarrow \text{contacted}(X, Y) \\
\text{connected}(X, Y) & \leftarrow \text{follows}(X, Z), \text{connected}(Z, Y) \\
\text{connected}(X, Y) & \leftarrow \text{follows}(X, Y)
\end{align*}
\]
Datalog Example

knows(Alice, Bob)  pfriends(Bob, Eve)
contacted(Bob, Eve) connected(Alice, Tim)
f(Alpe, Tim)  connected(Tim, Bill)  connected(Alice, Bill)
f(Bob, Bill)  connected(Bob, Bill)

pfriends(X, Y) ← knows(X, Y), connected(X, Z), connected(Y, Z)
pfriends(X, Y) ← contacted(X, Y)
connected(X, Y) ← follows(X, Z), connected(Z, Y)
connected(X, Y) ← follows(X, Y)
Datalog Example

```
knows(Alice, Bob)  
contacted(Bob, Eve)  
follows(Alice, Tim)  
follows(Tim, Bill)  
follows(Bob, Bill)  

pfriends(Bob, Eve)  
connected(Alice, Tim)  
connected(Tim, Bill)  
connected(Bob, Bill)  

pfriends(X, Y) ← knows(X, Y), connected(X, Z), connected(Y, Z)  
pfriends(X, Y) ← contacted(X, Y)  
connected(X, Y) ← follows(X, Z), connected(Z, Y)  
connected(X, Y) ← follows(X, Y)
```
Path Queries in Datalog

$pfriends(X,Y) \iff \text{knows}(X,Y), \text{connected}(X,Z), \text{connected}(Y,Z)$

$pfriends(X,Y) \iff \text{contacted}(X,Y)$
Path Queries in Datalog

\[
p\text{friends}(X,Y) \leftarrow \text{knows}(X,Y), \text{connected}(X,Z), \text{connected}(Y,Z)
\]
\[
p\text{friends}(X,Y) \leftarrow \text{contacted}(X,Y)
\]
\[
r(X,Y) \leftarrow r^-(Y,X)
\]
\[
s(X,Y) \leftarrow s(X,Z), s(Z,Y)
\]
\[
p\text{friends}(X,Y) \leftarrow (\text{knows} \land (\text{connected} \cdot \text{connected}))(X,Y)
\]
\[
p\text{friends}(X,Y) \leftarrow \text{contacted}(X,Y)
\]
Path Queries in Datalog

pfriends : union of conjunctive two-way regular path queries (UC2RPQ).

\[
\text{pfriends}(X,Y) \leftarrow \text{knows}(X,Y), \text{connected}(X,Z), \text{connected}(Y,Z) \\
\text{pfriends}(X,Y) \leftarrow \text{contacted}(X,Y) \\
\]
Regular Datalog

- recursion is restricted to transitive closure
- distinguished top clause whose head we call view
Datalog vs. Regular Datalog

\[
\text{pfriends}(X,Y) \leftarrow \text{knows}(X,Y), \text{connected}(X,Z), \text{connected}(Y,Z) \\
pfriends(X,Y) \leftarrow \text{contacted}(X,Y) \\
\text{connected}(X,Y) \leftarrow \text{follows}(X,Z), \text{connected}(Z,Y) \\
\text{connected}(X,Y) \leftarrow \text{follows}(X,Y) \\
pfriends(X,Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^-)) \lor \text{contacted})(X,Y) \\
\text{connected}(X,Y) \leftarrow \text{follows}^+(X,Y)
\]
Regular Datalog

- recursion is restricted to transitive closure
- distinguished top clause whose head we call view
- corresponds to UC2RPQ closed under transitive closure
Datalog vs. Regular Datalog

\[ p\text{connected}(X,Y) \leftarrow pfriends(X,Y) \]
\[ p\text{connected}(X,Y) \leftarrow pfriends(X,Z), p\text{connected}(Z,Y) \]
\[ pfriends(X,Y) \leftarrow knows(X,Y), connected(X,Z), connected(Y,Z) \]
\[ pfriends(X,Y) \leftarrow contacted(X,Y) \]
\[ connected(X,Y) \leftarrow follows(X,Z), connected(Z,Y) \]
\[ connected(X,Y) \leftarrow follows(X,Y) \]

\[ p\text{connected}(X,Y) \leftarrow pfriends^+(X,Y) \]
\[ pfriends(X,Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^\neg)) \lor \text{contacted}) (X,Y) \]
\[ connected(X,Y) \leftarrow follows^+(X,Y) \]
Regular Datalog

- recursion is restricted to **transitive closure**
- distinguished top clause whose head we call **view**
- corresponds to **UC2RPQ closed under transitive closure**
  - expressive: allows for complex graph patterns
  - tractable: parallelizability & decidable query containment
Building Blocks

Graph Databases

*finite* sets of constants $V$ (nodes) & symbols $\Sigma$ (edge labels).

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

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

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

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

### Regular Datalog (RD) Expressions

<table>
<thead>
<tr>
<th>Term Type</th>
<th>Definition</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terms (Node IDs)</td>
<td>$t ::= x</td>
<td>n$</td>
</tr>
<tr>
<td>Atoms</td>
<td>$A ::= s(t_1, t_2)$</td>
<td>$s \in \Sigma$</td>
</tr>
<tr>
<td>Literals</td>
<td>$L ::= A</td>
<td>A^+$</td>
</tr>
<tr>
<td>Conjunctive Body</td>
<td>$B ::= L_1 \land \ldots \land L_n$</td>
<td>$n \in \mathbb{N}$</td>
</tr>
<tr>
<td>Disjunctive Body</td>
<td>$D ::= B_1 \lor \ldots \lor B_n$</td>
<td>$n \in \mathbb{N}$</td>
</tr>
<tr>
<td>Clauses</td>
<td>$C ::= (t_1, t_2) \leftarrow D$</td>
<td></td>
</tr>
<tr>
<td>Programs</td>
<td>$\Pi : \Sigma \rightarrow {C_1, \ldots, C_n}$</td>
<td>$n \in \mathbb{N}$</td>
</tr>
</tbody>
</table>

### Regular Queries (RQ) over $G$

- RD-program $\Pi$
- distinguished query clause $\Omega$ whose head is the top-level view
Interpretations ($\mathcal{G}$)

Modeled as *indexed relations* $(\Sigma \times \{\Box, +\}) \rightarrow \mathcal{P}(\mathbf{V} \times \mathbf{V})$. 
Interpretations ($\mathcal{G}$)

Modeled as \textit{indexed relations} \((\Sigma \times \{\square, +\}) \rightarrow \mathcal{P}(V \times V)\).

\begin{itemize}
  \item Alice knows Bob contacted Eve
  \item Tim follows Bill
  \item Bill follows Alice
  \item Eve follows Bob
\end{itemize}
Semantics (I/II)

### Interpretations ($\mathcal{G}$)

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

### Interpretation Well-Formedness (wf\(G\))

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

- \(\text{is\_closure}(g_s, g_p) \iff \forall (n_1, n_2) \in g_p, \exists \rho \in V^+, \text{path}(g_s, n_1, \rho) \land \text{last}(\rho) = n_2\)
- \(\text{path}(g, n_1, \rho) \iff \forall i \in \{1 \ldots |\rho|\}, (n_i, n_{i+1}) \in g\)
- \(\text{wf\(G\)}(\mathcal{G}) \iff \forall s, \text{is\_closure}(\mathcal{G}(s, \Box), \mathcal{G}(s, +))\)
## Semantics (II/II)

<table>
<thead>
<tr>
<th>Literal Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $L \triangleq s^l(n_1, n_2)$, $G \models L \iff (n_1, n_2) \in G(s, l)$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clause Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $C \triangleq (t_1, t_2) \leftarrow (L_{1,1} \land \ldots \land L_{1,n}) \lor \ldots \lor (L_{m,1} \land \ldots \land L_{m,n})$, $G \models_s L \iff \forall \eta, \bigvee_{i=1..m}(\bigwedge_{j=1..n} G \models \eta(L_{i,j})) \Rightarrow G \models \eta(s(t_1, t_2))$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $\Pi \triangleq \Sigma \rightarrow { C_1, \ldots, C_n }$, $G \models_{\Sigma} \Pi \iff \forall s \in \Sigma, G \models_s \Pi(s)$.</td>
</tr>
</tbody>
</table>
View Computation: Clausal Evaluation

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

- match each conjunctive body $B_i$ against the interpretation
  - extend substitutions $M^A_G(a)$ matching a given atom $a$
  - ...uniformly, to the entire body as $M^B_G(B_i)$
- collect the substitutions matching the entire disjunctive body

Clausal Consequence Operator

$$T^{\Pi,s}(G) \triangleq \{ \sigma(t_1, t_2) \mid \sigma \in \bigcup_{i=1..n} M^B_G(B_i) \}$$
**Stratification Condition**

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:

\[
\max_{r \in \text{sym}(B)} \sigma(r) < \sigma(s)
\]
View Computation: Stratified Program Evaluation

\[ \Pi_1 = \{ \text{connected}(X, Y) \leftarrow \text{follows}^+(X, Y) \} \]

\[ \Pi_2 = \{ \text{pfriends}(X, Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^-)) \lor \text{contacted})(X, Y) \} \]

\[ \Pi_3 = \{ \text{pconnected}(X, Y) \leftarrow \text{pfriends}^+(X, Y) \} \]
View Computation: Stratified Program Evaluation

\[ \Pi_1 = \{ \text{connected}(X, Y) \leftarrow \text{follows}^+(X, Y) \} \]

\[ \Pi_2 = \{ \text{pfriends}(X, Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^{-})) \lor \text{contacted})(X, Y) \} \]

\[ \Pi_3 = \{ \text{pconnected}(X, Y) \leftarrow \text{pfriends}^+(X, Y) \} \]

\[ \Delta_1 = T^{\Pi_1, \text{connected}}(\emptyset) = \text{connected} \rightarrow \{(Alice, Bill), (Bob, Bill)\} \]
**View Computation: Stratified Program Evaluation**

\[ \Pi_1 = \{ \text{connected}(X, Y) \leftarrow \text{follows}^+(X, Y) \} \]

\[ \Pi_2 = \{ \text{pfriends}(X, Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^-)) \lor \text{contacted})(X, Y) \} \]

\[ \Pi_3 = \{ \text{pconnected}(X, Y) \leftarrow \text{pfriends}^+(X, Y) \} \]

\[ \Delta_1 = T^{\Pi_1, \text{connected}}(\emptyset) = \text{connected} \rightarrow \{(Alice, Bill), (Bob, Bill)\} \]

\[ \Delta_2 = T^{\Pi_2, \text{pfriends}}(\Delta_1) = \text{pfriends} \rightarrow \{(Alice, Bob), (Bob, Eve)\} \]
View Computation: Stratified Program Evaluation

\[ \Pi_1 = \{ \text{connected}(X, Y) \leftarrow \text{follows}^+(X, Y) \} \]

\[ \Pi_2 = \{ \text{pfriends}(X, Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^-)) \lor \text{contacted})(X, Y) \} \]

\[ \Pi_3 = \{ \text{pconnected}(X, Y) \leftarrow \text{pfriends}^+(X, Y) \} \]

\[ \Delta_1 = T_{\Pi_1, \text{connected}}(\emptyset) = \text{connected} \rightarrow \{(Alice, Bill), (Bob, Bill)\} \]

\[ \Delta_2 = T_{\Pi_2, \text{pfriends}}(\Delta_1) = \text{pfriends} \rightarrow \{(Alice, Bob), (Bob, Eve)\} \]

\[ \Delta_3 = T_{\Pi_3, \text{pconnected}}(\Delta_1 \cup \Delta_2) = \text{pconnected} \rightarrow \{(Alice, Bob), (Bob, Eve), (Alice, Eve)\} \]
View Maintenance Computation (I/II)

\[ \Pi_1 = \{ \text{connected}(X, Y) \leftarrow \text{follows}^+(X, Y) \} \]

\[ \Pi_2 = \{ \text{pfriends}(X, Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^-)) \lor \text{contacted})(X, Y) \} \]

\[ \Pi_3 = \{ \text{pconnected}(X, Y) \leftarrow \text{pfriends}^+(X, Y) \} \]
Modeling 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 Application

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

$\Delta\{s \rightarrow (g_+, g_-)\} \triangleq (\Delta_+\{s \rightarrow g_+\}, \Delta_-\{s \rightarrow g_- \setminus g_+\})$
View Maintenance Computation (II/II)

\( \Pi_1 = \{ \text{connected}(X, Y) \leftarrow \text{follows}^+(X, Y) \} \)

\( \Pi_2 = \{ \text{pfriends}(X, Y) \leftarrow ((\text{knows} \land (\text{connected} \cdot \text{connected}^-)) \lor \text{contacted})(X, Y) \} \)

\( \Pi_3 = \{ \text{pconnected}(X, Y) \leftarrow \text{pfriends}^+(X, Y) \} \)

\( \Delta_1' = T_{\Pi_1, \text{connected}}^{\Pi_1}(\Delta_1) = \Delta_1\{\text{connected} \rightarrow (\{\text{Bill, Bob}\}, \emptyset)\} \)

... where \( \text{supp} = \{\text{knows}, \text{contacted}, \text{follows}^+, \text{connected}, \text{pconnected}\} \)
View Computation vs. Maintenance

For each clause \( C \triangleq (t_1, t_2) \leftarrow \bigvee_{i=1..n} B_i \) corresponding to \( \Pi(s) \)

**Incremental Clausal Consequence Operator**

\[
T_{\Pi,s}^{\Pi,s}(\Delta) = \begin{cases} 
T_{\Pi,s}^{\Pi,s}(G :+ \Delta), & (s \notin \text{supp}) \lor (\Delta_+ \cup D) \neq \emptyset \\
\bigcup_{i=1..n} M_{B_i,\Delta}^B G, & \text{otherwise}
\end{cases}
\]
Full Engine Overview

- stratified, single-pass, bottom-up heuristic
- non-recursive (recursion internalized in closure computation)
- supports both view computation and incremental maintenance

- core component: clause evaluation
  - forward-chain clausal consequence operator (fwd_or_clause)
  - based on a matching algorithm
  - corresponds to computing a nested-loop join
Fixpoint fwd_program \( \Pi G \) supp \( \Delta \) \( \Sigma_{\triangleright} \) \( \Sigma_{\triangleleft} \) : edelta :=

match \( \Sigma_{\triangleleft} \) with
| [::] => \( \Delta \)
| [:: s & ss] =>
  let (arg, body) := \( \Pi s \) in
  let \( \Delta' := \text{fwd_or_clause } G \) supp \( \Delta \) s arg body \( \text{in} \)
  let \( \Delta' := \text{compute_closures } G \) \( \Delta' \) s \( \text{in} \)
  fwd_program \( \Pi G \) supp \( \Delta' \) (s \( \cup \) s+ \( \cup \) \( \Sigma_{\triangleright} \)) ss
Regular Datalog: Engine Characterization

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, \( \Sigma_\uparrow \), are a well-formed \( \Pi \)-slice
- \( \Delta \) only modifies \( \Sigma_\uparrow \), i.e., \( \text{sym}(\Delta) \subseteq \Sigma_\uparrow \)
- \( \mathcal{G} :+: \Delta \models_{\Sigma_\uparrow} \Pi \)

Then, it outputs \( \Delta_\mathcal{O} \), such that \( \mathcal{G} :+: \Delta_\mathcal{O} \models_{\Sigma} \Pi \).
Incremental View Maintenance (IVM)

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

**Soundness**

If \( V[\mathcal{G}] \models \Pi_{\mathcal{G}} \), the engine outputs an incremental update \( V^{\Delta} \) s.t.

\[ V[\mathcal{G}] :+ : V^{\Delta} \models \Pi_{\mathcal{G} \triangleq \Delta} \]
Incremental $\Delta$-Matching

Compute $V^\Delta$ s.t. $V[\mathcal{G} :+\Delta] = V[\mathcal{G}] :+V^\Delta$ with a *delta program*.\(^1\)

Idea: distributing deltas over joins and factoring.

Incremental $\Delta$-Matching

Compute $V^\Delta$ s.t $V[G :+: \Delta] = V[G] :+: V^\Delta$ with a *delta program*\(^1\)

Idea: distributing deltas over joins and factoring.

**Delta Program**

For $V \leftarrow L_1, \ldots, L_n$, a *delta program* is a set of clauses of the form

$$V \leftarrow L_1, \ldots, L_{i-1}, L_i^\Delta, L_{i+1}^\nu, \ldots, L_n^\nu$$

---

Incremental $\Delta$-Matching

Compute $V^\Delta$ s.t $V[G ::= \Delta] = V[G ::= V^\Delta]$ with a *delta program*\(^1\)

Idea: distributing deltas over joins and factoring.

**Delta Programs**

For $V \leftarrow L_1, \ldots, L_n$, a *delta program* is a set of clauses of the form

$$V \leftarrow L_1, \ldots, L_{i-1}, L_i^\Delta, L_{i+1}^\nu, \ldots, L_n^\nu,$$

where $L_j$ is matched against $G$ atoms with its symbol.

Incremental $\Delta$-Matching

Compute $V^\Delta$ s.t. $V[G :+: \Delta] = V[G] :+: V^\Delta$ with a delta program\(^1\)

Idea: distributing deltas over joins and factoring.

### Delta Programs

For $V \leftarrow L_1, \ldots, L_n$, a delta program is a set of clauses of the form

$$V \leftarrow L_1, \ldots, L_{i-1}, L_i^\Delta, L_{i+1}^\nu, \ldots, L_n^\nu,$$

where

- $L_j$ is matched against $G$ atoms with its symbol.
- $L_j^\Delta$ is matched against $\Delta$ atoms with its symbol.

Incremental $\Delta$-Matching

Compute $V^\Delta$ s.t $V[\mathcal{G} :+\Delta] = V[\mathcal{G}] :+ V^\Delta$ with a delta program

Idea: distributing deltas over joins and factoring.

**Delta Programs**

For $V \leftarrow L_1, \ldots, L_n$, a delta program is a set of clauses of the form

$$V \leftarrow L_1, \ldots, L_{i-1}, L_i^\Delta, L_i^\nu, L_{i+1}^\nu, \ldots, L_n^\nu,$$

where

$L_j$ is matched against $\mathcal{G}$ atoms with its symbol.
$L_j^\Delta$ is matched against $\Delta$ atoms with its symbol.
$L_j^\nu$ is matched against $\mathcal{G} :+\Delta$ atoms with its symbol.

---

Example: Incremental $\Delta$-Matching

\[ \text{pfriends}(X, Y) \leftarrow \text{knows}(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y) \]

(a) Initial Graph $G$
Example: Incremental $\Delta$-Matching

\[ pfriends(X, Y) \leftarrow \text{knows}(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y) \]

(a) Initial Graph $\mathcal{G}$

\[ pfriends[\mathcal{G}] = \{(V_6, V_0), (V_3, V_0)\} \]
Example: Incremental $\Delta$-Matching

$$\text{pfriends}(X, Y) \leftarrow \text{knows}(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y)$$

...updating $\mathcal{G}$ with $\Delta = \{\text{connected}(V_1, V_2), \text{knows}(V_2, V_0), \text{connected}(V_4, V_5)\}$

$$\text{pfriends}[\mathcal{G} \vdash: \Delta] = \text{pfriends}[\mathcal{G}] \cup ?$$
Example: Incremental $\Delta$-Matching

\[ pfriends(X, Y) \leftarrow knows(X, Y), connected(Z, X), connected(Z, Y) \]

...updating $\mathcal{G}$ with $\Delta = \{connected(V_1, V_2), knows(V_2, V_0), connected(V_4, V_5)\}$

\[ pfriends[\mathcal{G} :+\Delta] = pfriends[\mathcal{G}] \cup ? \]

\[ pfriends^\Delta(X, Y) \leftarrow knows^\Delta(X, Y), connected(Z, X), connected(Z, Y) \]
\[ pfriends^\Delta(X, Y) \leftarrow knows^\nu(X, Y), connected^\Delta(Z, X), connected(Z, Y) \]
\[ pfriends^\Delta(X, Y) \leftarrow knows^\nu(X, Y), connected^\nu(Z, X), connected^\Delta(Z, Y) \]
Example: Incremental $\Delta$-Matching

\[ pfriends(X, Y) \leftarrow \text{knows}(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y) \]

(a) Initial Graph $\mathcal{G}$

(b) Updated Graph

...updating $\mathcal{G}$ with $\Delta = \{\text{connected}(V_1, V_2), \text{knows}(V_2, V_0), \text{connected}(V_4, V_5)\}$

\[ pfriends[\mathcal{G} :+\Delta] = pfriends[\mathcal{G}] \cup \emptyset \cup \ldots \]

\[ pfriends^\Delta(X, Y) \leftarrow \text{knows}^\Delta(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y) \]
Example: Incremental $\Delta$-Matching

\[ pfriends(X, Y) \leftarrow knows(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y) \]

...updating $\mathcal{G}$ with $\Delta = \{\text{connected}(V_1, V_2), \text{knows}(V_2, V_0), \text{connected}(V_4, V_5)\}$

\[ pfriends[\mathcal{G} \leftarrow \Delta] = pfriends[\mathcal{G}] \cup \{(V_2, V_0)\} \cup \ldots \]

\[ pfriends^\Delta(X, Y) \leftarrow \text{knows}^\Delta(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y) \]

\[ pfriends^\Delta(X, Y) \leftarrow \text{knows}^\nu(X, Y), \text{connected}^\Delta(Z, X), \text{connected}(Z, Y) \]
Example: Incremental $\Delta$-Matching

\[
pfriends(X, Y) \leftarrow \text{knows}(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y)
\]

\[
\begin{align*}
V_1 & \quad V_2 \\
V_0 & \quad V_3 \\
V_6 & \quad V_4 & \quad V_5 \\
V_4 & & & & \quad \text{(a) Initial Graph } G
\end{align*}
\]

\[
\begin{align*}
V_1 & \quad V_2 \\
V_0 & \quad V_3 \\
V_6 & \quad V_4 & \quad V_5 \\
V_5 & & & & \quad \text{(b) Updated Graph}
\end{align*}
\]

...updating $G$ with $\Delta = \{ \text{connected}(V_1, V_2), \text{knows}(V_2, V_0), \text{connected}(V_4, V_5) \}$

\[
pfriends[G :+\Delta] = pfriends[G] \cup \{(V_2, V_0)\} \cup \{(V_0, V_2), (V_0, V_5)\}
\]

\[
pfriends^\Delta(X, Y) \leftarrow \text{knows}^\Delta(X, Y), \text{connected}(Z, X), \text{connected}(Z, Y)
\]

\[
pfriends^\Delta(X, Y) \leftarrow \text{knows}^\nu(X, Y), \text{connected}^\Delta(Z, X), \text{connected}(Z, Y)
\]

\[
pfriends^\Delta(X, Y) \leftarrow \text{knows}^\nu(X, Y), \text{connected}^\nu(Z, X), \text{connected}^\Delta(Z, Y)
\]

37 / 42
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/
Related Work

**Datalog Setting:**

- **Certified Standard and Stratified Datalog Engines**
  [Dumbrava, PhD Thesis 2016], [Benzaken et al., ITP 2017]

**Graph Setting:**

- A Mechanized Formalization of GraphQL [Diaz et al., 2020]
- Incremental Graph Computation for RPQ [Fan et al., 2017]

**Relational Setting:**

- 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]
Perspectives

- support for more expressive graph models
- support for transactions & dynamic queries
- refinement methodology to formally test commercial engines

Thank you!
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: $|\text{supp}(\mathcal{G})| = 82$ (WD), $|\text{supp}(\mathcal{G})| = 27$ (SNB)
- instance & workload sizes: $|\mathcal{G}| = 1K$, $|\mathcal{W}| = 10$ UC2RPQ
- $\rho_{\text{supp}} = \frac{|\text{supp}(\Delta^+)\cup\text{supp}(\mathcal{G})|}{|\text{supp}(\mathcal{G})|} \in \{0.05, 0.1, 0.15, 0.2, 0.25\}$
- $\rho = \frac{|\Delta^+|}{|\mathcal{G}'|} \times 100$
- Time Gain = FVM - IVM, Ratio Gain = $100 - \frac{100 \times \text{IVM}}{\text{FVM}}$
### Experiments

**Table: $\mathcal{W}_{WD}$ 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>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: $\mathcal{W}_{SNB}$ 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>
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
Incremental $\Delta$-Matching

Let $V \overset{\Delta}{=} r \times s$, where $V(X, Y) \leftarrow r(X, Z), s(Z, Y)$, $r^\Delta$ and $s^\Delta$.

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

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

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

$V_1^\Delta \leftarrow r^\Delta(X, Z), s(Z, Y)$

$V_2^\Delta \leftarrow r^\nu(X, Z), s^\Delta(Z, Y)$. 
Variables \((V \Sigma : \text{finType})\).

\textbf{Inductive} \quad L := \Box \mid +.

\textbf{Inductive} \quad \text{egraph} := \text{EGraph of} \{\text{set} \ V \times \ V\}.

\textbf{Inductive} \quad \text{lrel} := \text{LRel of} \{\text{ffun} \ \Sigma \times L \to \text{egraph}\}.

\textbf{Record} \quad \text{atom} := \text{Atom} \{\text{syma} : \Sigma; \ \text{arga} : T \times T\}.

\textbf{Record} \quad \text{lit} := \text{Lit} \{\text{tagl} : L; \ \text{atoml} : \text{atom}\}.

\textbf{Record} \quad \text{cbody} := \text{CBody} \{\text{litb} : \text{seq} \ \text{lit}\}.

\textbf{Record} \quad \text{clause} := \text{Clause} \{\text{headc} : T \times T; \ \text{bodyc} : \text{seq} \ \text{cbody}\}.

\textbf{Inductive} \quad \text{program} := \text{Program of} \{\text{ffun} \ \Sigma \to \text{clause} \ T \Sigma L\}.
Incremental View Maintenance

**Incremental Atom Matching**

\[ M^A_m(a) = \begin{cases} M^A_m(a) \text{ if } m \in \{B, F\} \\ \emptyset \text{ otherwise} \end{cases} \cup \begin{cases} M^\Delta_m(a) \text{ if } m \in \{D, F\} \\ \emptyset \text{ otherwise} \end{cases} \]

**Incremental Body Matching**

For a set of body literals \( B \triangleq [L_1, \ldots, L_n] \), generates \( B_\Delta \):

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

**Incremental Clausal Maintenance Operator**

\[
T_{\mathcal{G},\Delta}^{\Pi,s}(\Delta) = \begin{cases} T_{\mathcal{G},\Delta}^{\Pi,s}(\mathcal{G} \vdash \Delta), & \text{if } s \notin \text{supp} \land (\Delta - \cup D) \neq \emptyset \\
\cup_{B_m \in B_\Delta} M^B_m(B_m), & \text{otherwise}
\end{cases}
\]
## Graph Query Complexity

<table>
<thead>
<tr>
<th>Query Fragment</th>
<th>Evaluation</th>
<th>Containment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPQ</td>
<td>NLOGSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>RPQ$_C$</td>
<td>#P–complete</td>
<td>Undefined</td>
</tr>
<tr>
<td>2RPQ</td>
<td>NLOGSPACE-complete</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>C2RPQ</td>
<td>NLOGSPACE-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>UC2RPQ</td>
<td>NLOGSPACE-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>nUC2RPQ</td>
<td>NLOGSPACE-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>UCN2RPQs</td>
<td>NLOGSPACE-complete</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td>RQ</td>
<td>NLOGSPACE-complete</td>
<td>2EXPSPACE-complete</td>
</tr>
</tbody>
</table>

**Table:** Graph query valuation and containment data complexity.


