Incremental Maintenance of Graph Views

Stefania Dumbrava

March 9, 2021

This Talk

Our Work

A machine-verified *incremental* graph query engine.

This Talk

Our Work

A machine-verified *incremental* graph query engine.

- ...relying on the Coq proof assistant

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

[2] Juan L. Reutter, et al.: *Regular Queries on Graph Databases*. Th. Comput. Syst. 61(1): 31-83 (2017)

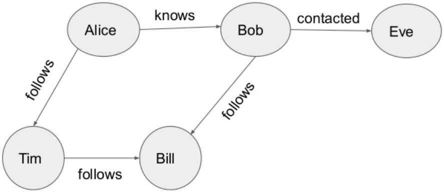
Datalog

- logic programming without function symbols
- consists of facts and rules
- finite model semantics
- sound, complete and terminating evaluation

  $\rightarrow$ *bottom-up iteration of a fixpoint operator*

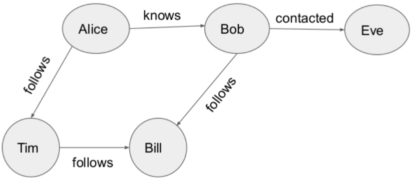
Datalog Example

Assume we have the following simple graph database instance:



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

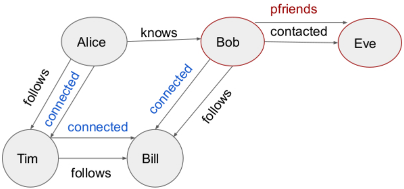
Datalog Example



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

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)

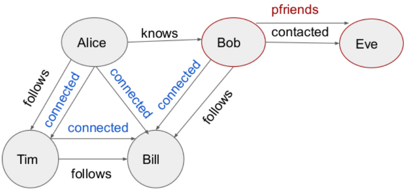
Datalog Example



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



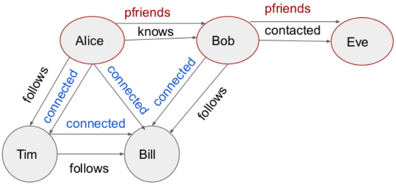
knows(Alice,Bob)
contacted(Bob,Eve)
follows(Alice,Tim)
follows(Tim,Bill)
follows(Bob, Bill)

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

connected(Alice,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) ✓

connected(Alice,Bill) ✓

pfriends(Alice,Bob)

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)

Regular Datalog

- safe: head variables in each clause appear in its body
- recursion is restricted to transitive closure

  ...and internalized into special literals
- distinguished top clause whose head we call view
- all symbols (except the view one) are binary
- stratified: no clauses with body symbols not previously defined

Datalog vs. Regular Datalog

pconnected(X,Y) ← pfriends(X,Y)
pconnected(X,Y) ← pfriends(X,Z), pconnected(Z,Y)
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)

⇩

pconnected(X,Y) ← **pfriends**$^+$(X,Y)
pfriends(X,Y) ← ((knows ∧ (connected · connected$^-$)) ∨ contacted) (X,Y)
connected(X,Y) ← **follows**$^+$(X,Y)

- Regular Datalog corresponds to UC2RPQ under transitive closure
  → expressive: allows for complex graph patterns
  → tractable: parallelizability & decidable query containement

Building Blocks

Graph Databases

finite sets of constants $\mathbf{V}$ (nodes) & symbols Σ (edge labels).

Graph Instance $\mathcal{G}$ over Σ:
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 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: $\lambda(\rho) = s_1 \ldots s_k \in \Sigma^*$

Syntax

Regular Datalog (RD) Expressions

Terms (Node IDs)	$t ::= x \mid n$	where $x \in Var$, $n \in \mathbf{V}$
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
- distinguished query clause Ω whose head is the top-level view

Semantics (I/II)

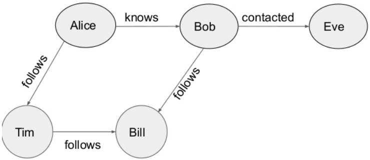
Interpretations ($\mathcal{G}$)

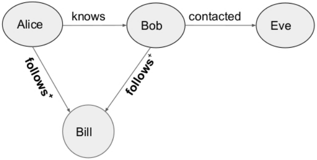
Modeled as *indexed relations* $(\Sigma \times \{\Box, +\}) \to \mathcal{P}(\mathbf{V} \times \mathbf{V})$.

Semantics (I/II)

Interpretations ($\mathcal{G}$)

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





Semantics (I/II)

Interpretations ($\mathcal{G}$)

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

Interpretation Well-Formedness (wfG)

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

$$
\begin{aligned}
\mathrm{wfG}(\mathcal{G}) &\iff \forall s, \mathrm{is\_closure}(\mathcal{G}(s, \Box), \mathcal{G}(s, +)) \\
\mathrm{is\_closure}(g_s, g_p) &\iff \forall (n_1, n_2) \in g_p, \exists \rho \in \mathbf{V}^+, \mathrm{path}(g_s, n_1, \rho) \wedge \mathrm{last}(\rho) = n_2 \\
\mathrm{path}(g, n_1, \rho) &\iff \forall i \in \{1 \ldots |\rho|\}, (n_i, n_{i+1}) \in g
\end{aligned}
$$

Semantics (II/II)

Literal Satisfaction

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

Clause Satisfaction

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

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

Program Satisfaction

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

View Computation: Clausal Evaluation

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

- match each conjunctive body B_i against the interpretation
  - extend substitutions $M^A_{\mathcal{G}}(a)$ matching a given atom a
  - ...uniformly, to the entire body as $M^B_{\mathcal{G}}(B_i)$
- collect the substitutions matching the entire disjunctive body

Clausal Consequence Operator

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

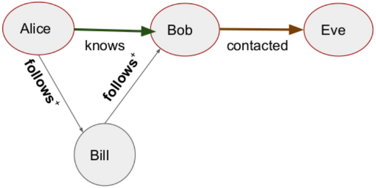
View Computation: Stratified Program Evaluation

Stratification Condition

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

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

View Computation: Stratified Program Evaluation

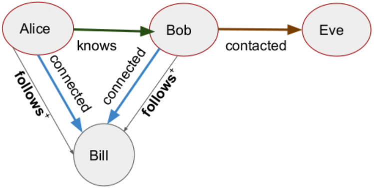


$$\Pi_1 = \{\ connected(X,Y) \leftarrow follows^{+}(X,Y)\ \}$$

$$\Pi_2 = \{\ pfriends(X,Y) \leftarrow ((knows \wedge (connected \cdot connected^{-})) \vee contacted)(X,Y)\ \}$$

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

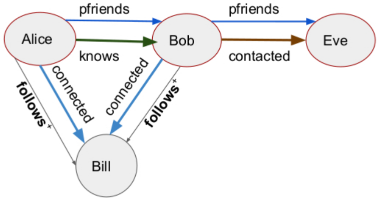
View Computation: Stratified Program Evaluation



$$\Pi_1 = \{ \; connected(X, Y) \leftarrow follows^+(X, Y) \; \}$$
$$\Pi_2 = \{ \; pfriends(X, Y) \leftarrow ((knows \wedge (connected \cdot connected^-)) \vee contacted)(X, Y) \; \}$$
$$\Pi_3 = \{ \; pconnected(X, Y) \leftarrow pfriends^+(X, Y) \; \}$$

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

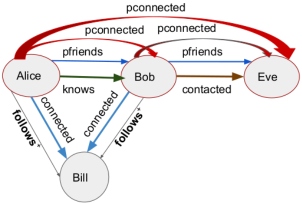
View Computation: Stratified Program Evaluation



$$\Pi_1 = \{\ \ connected(X, Y) \leftarrow follows^+(X, Y)\ \ \}$$
$$\Pi_2 = \{\ \ pfriends(X, Y) \leftarrow ((knows \wedge (connected \cdot connected^-)) \vee contacted)(X, Y)\ \ \}$$
$$\Pi_3 = \{\ \ pconnected(X, Y) \leftarrow pfriends^+(X, Y)\ \ \}$$

$$\Delta_1 = T^{\Pi_1, connected}(\emptyset) \quad = connected \rightarrow \{(Alice, Bill), (Bob, Bill)\}$$
$$\Delta_2 = T^{\Pi_2, pfriends} \quad (\Delta_1) = pfriends \rightarrow \{(Alice, Bob), (Bob, Eve)\}$$

View Computation: Stratified Program Evaluation



$$\Pi_1 = \{\ connected(X, Y) \leftarrow follows^+(X, Y)\ \}$$
$$\Pi_2 = \{\ pfriends(X, Y) \leftarrow ((knows \wedge (connected \cdot connected^-)) \vee contacted)(X, Y)\ \}$$
$$\Pi_3 = \{\ pconnected(X, Y) \leftarrow pfriends^+(X, Y)\ \}$$

$$\Delta_1 = T^{\Pi_1, connected}(\emptyset) = connected \rightarrow \{(Alice, Bill), (Bob, Bill)\}$$
$$\Delta_2 = T^{\Pi_2, pfriends}(\Delta_1) = pfriends \rightarrow \{(Alice, Bob), (Bob, Eve)\}$$
$$\Delta_3 = T^{\Pi_3, pconnected}(\Delta_1 \cup \Delta_2) = pconnected \rightarrow \{(Alice, Bob), (Bob, Eve), (Alice, Eve)\}$$

Modeling Updates

Updates

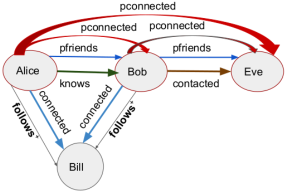
An *update* $\Delta \equiv (\Delta_+, \Delta_-)$ is a pair of *disjoint* graphs Δ_+, Δ_-.

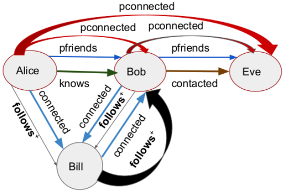
- $\Delta_+ \equiv$ bulk insertions
- $\Delta_- \equiv$ bulk deletions

Update Application

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

View Maintenance Computation





$$\Pi_1 = \{ \; connected(X, Y) \leftarrow follows^+(X, Y) \; \}$$
$$\Pi_2 = \{ \; pfriends(X, Y) \leftarrow ((knows \wedge (connected \cdot connected^-)) \vee contacted)(X, Y) \; \}$$
$$\Pi_3 = \{ \; pconnected(X, Y) \leftarrow pfriends^+(X, Y) \; \}$$

$$\Delta'_1 = T_{\mathrm{supp}}^{\Pi_1, connected}(\Delta_1) = \Delta_1\{connected \rightarrow (\{(Bill, Bob)\}, \emptyset)\}$$

... where supp $= \{knows, contacted, follows^+, connected, pconnected\}$

View Computation vs. Maintenance

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

Incremental Clausal Consequence Operator

$$T^{\Pi,s}_{\mathcal{G},\mathrm{supp}}(\Delta) = \begin{cases} T^{\Pi,s}(\mathcal{G} \mathrel{:+:} \Delta), & (s \notin \mathrm{supp}) \vee (\Delta_- \cup D) \neq \emptyset \\ \bigcup_{i=1..n} M^{B}_{\mathcal{G},\Delta}(B_i), & \text{otherwise} \end{cases}$$

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*

Full 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
```

Regular Datalog: Engine Characterization

Theorem (Soundness)

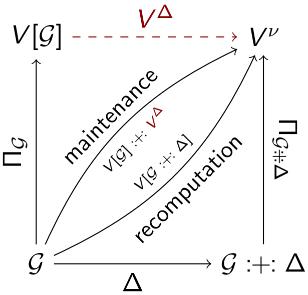
- Π – *a safe, stratifiable, Regular Datalog program*
- $\boldsymbol{\Sigma}$ – *its set of symbols*
- $\mathcal{G}$ – *a graph instance*
- Δ – *an update*

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* $\mathcal{G} :\!+\!: \Delta_O \models_{\boldsymbol{\Sigma}} \Pi$.

Incremental View Maintenance (IVM)



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

Soundness

If $V[\mathcal{G}] \models \Pi_{\mathcal{G}}$, *the engine outputs an incremental update* V^{Δ} *s.t*

$$V[\mathcal{G}] \mathrel{:+:} V^{\Delta} \models \Pi_{\mathcal{G} \ddagger \Delta}$$

Incremental Δ-Matching

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

Idea: distributing deltas over joins and factoring.

[1] Ashish Gupta, et al.: *Maintaining Views Incrementally*. SIGMOD 1993: 157-166

Incremental Δ-Matching

Compute V^{Δ} s.t $V[\mathcal{G} :+: \Delta] = V[\mathcal{G}] :+: V^{\Delta}$ 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

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

[1] Ashish Gupta, et al.: *Maintaining Views Incrementally*. SIGMOD 1993: 157-166

Incremental Δ-Matching

Compute V^{Δ} s.t $V[\mathcal{G} :+: \Delta] = V[\mathcal{G}] :+: V^{\Delta}$ 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

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

$\mathbf{L_j}$ is matched against $\mathcal{G}$ atoms with its symbol.

[1] Ashish Gupta, et al.: *Maintaining Views Incrementally*. SIGMOD 1993: 157-166

Incremental Δ-Matching

Compute V^{Δ} s.t $V[\mathcal{G} :+: \Delta] = V[\mathcal{G}] :+: V^{\Delta}$ 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

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

L_j is matched against $\mathcal{G}$ atoms with its symbol.

$\mathbf{L_j^{\Delta}}$ is matched against Δ atoms with its symbol.

[1] Ashish Gupta, et al.: *Maintaining Views Incrementally*. SIGMOD 1993: 157-166

Incremental Δ-Matching

Compute V^{Δ} s.t $V[\mathcal{G} :+: \Delta] = V[\mathcal{G}] :+: V^{\Delta}$ 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

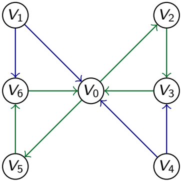
$$V \leftarrow L_1, \ldots, L_{i-1}, L_i^{\Delta}, \mathbf{L}^{\nu}_{\mathbf{i+1}}, \ldots, \mathbf{L}^{\nu}_{\mathbf{n}}, \text{ where}$$

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

[1] Ashish Gupta, et al.: *Maintaining Views Incrementally*. SIGMOD 1993: 157-166

Example: Incremental Δ-Matching

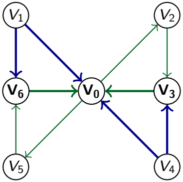
$$\textit{pfriends}(X, Y) \quad \leftarrow \quad \textit{knows}(X, Y), \textit{connected}(Z, X), \textit{connected}(Z, Y)$$



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

Example: Incremental Δ-Matching

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

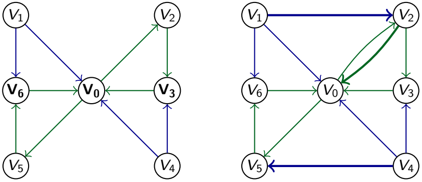


(a) Initial Graph $\mathcal{G}$

$$\textit{pfriends}[\mathcal{G}] = \{(\mathbf{V_6}, \mathbf{V_0}), (\mathbf{V_3}, \mathbf{V_0})\}$$

Example: Incremental Δ-Matching

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



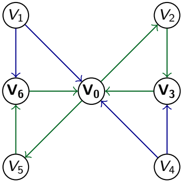
(a) Initial Graph $\mathcal{G}$ (b) Updated Graph

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

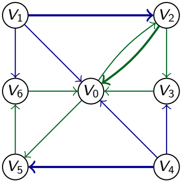
$$\textit{pfriends}[\mathcal{G} :+: \Delta] = \textit{pfriends}[\mathcal{G}] \cup ?$$

Example: Incremental Δ-Matching

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



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



(b) Updated Graph

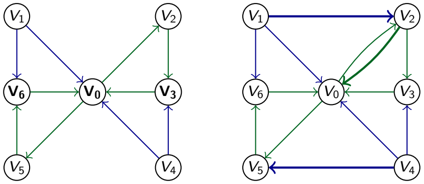
...updating $\mathcal{G}$ with $\Delta = \{\textbf{connected}(V_1, V_2), \textbf{knows}(V_2, V_0), \textbf{connected}(V_4, V_5)\}$

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

$$\begin{aligned}
\textit{pfriends}^{\Delta}(X, Y) &\leftarrow \textbf{knows}^{\Delta}(X, Y), \textit{connected}(Z, X), \textit{connected}(Z, Y)\\
\textit{pfriends}^{\Delta}(X, Y) &\leftarrow \textbf{knows}^{\nu}(X, Y), \textbf{connected}^{\Delta}(Z, X), \textit{connected}(Z, Y)\\
\textit{pfriends}^{\Delta}(X, Y) &\leftarrow \textbf{knows}^{\nu}(X, Y), \textbf{connected}^{\nu}(Z, X), \textbf{connected}^{\Delta}(Z, Y)
\end{aligned}$$

Example: Incremental Δ-Matching

$$pfriends(X, Y) \quad \leftarrow \quad knows(X, Y), connected(Z, X), connected(Z, Y)$$



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

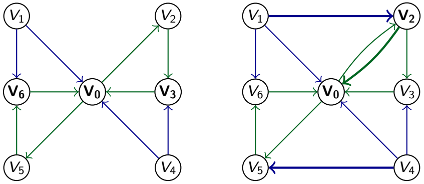
...updating $\mathcal{G}$ with $\Delta = \{\mathbf{connected}(V_1, V_2), \mathbf{knows}(V_2, V_0), \mathbf{connected}(V_4, V_5)\}$

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

$$pfriends^{\Delta}(X, Y) \quad \leftarrow \quad \mathbf{knows}^{\Delta}(X, Y), connected(Z, X), connected(Z, Y)$$

Example: Incremental Δ-Matching

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



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

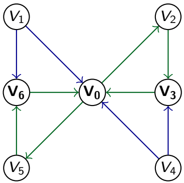
...updating $\mathcal{G}$ with $\Delta = \{\textbf{connected}(V_1, V_2), \textbf{knows}(V_2, V_0), \textbf{connected}(V_4, V_5)\}$

$$\textit{pfriends}[\mathcal{G} :+: \Delta] = \textit{pfriends}[\mathcal{G}] \cup \{(\mathbf{V_2}, \mathbf{V_0})\} \cup \ldots$$

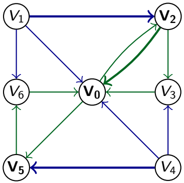
$$\textit{pfriends}^{\Delta}(X, Y) \quad \leftarrow \quad \textbf{knows}^{\Delta}(X, Y), \textit{connected}(Z, X), \textit{connected}(Z, Y)$$
$$\textit{pfriends}^{\Delta}(X, Y) \quad \leftarrow \quad \textbf{knows}^{\nu}(X, Y), \textbf{connected}^{\Delta}(Z, X), \textit{connected}(Z, Y)$$

Example: Incremental Δ-Matching

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



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



(b) Updated Graph

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

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

$$\begin{array}{lcl}
\textit{pfriends}^{\Delta}(X, Y) & \leftarrow & \mathbf{knows}^{\Delta}(X, Y), \textit{connected}(Z, X), \textit{connected}(Z, Y) \\
\textit{pfriends}^{\Delta}(X, Y) & \leftarrow & \mathbf{knows}^{\nu}(X, Y), \mathbf{connected}^{\Delta}(Z, X), \textit{connected}(Z, Y) \\
\textit{pfriends}^{\Delta}(X, Y) & \leftarrow & \mathbf{knows}^{\nu}(X, Y), \mathbf{connected}^{\nu}(Z, X), \mathbf{connected}^{\Delta}(Z, Y)
\end{array}$$

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

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/

Perspectives

- extension to more expressive graph models
- support for Datalog extensions
- handling complex transactions
- refinement methodology to formally test commercial engines

Thank you !

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]

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) $\leq$ WD runtimes (very dense)
- engine works best on bulk updates with small support size
  symbol-level maintenance granularity

Incremental Δ-Matching

Let $V \equiv r \bowtie s$, 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:

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

Regular Datalog in Coq

```
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}.
```

Incremental View Maintenance

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 $B \equiv [L_1, \ldots, L_n]$, generates B_Δ:

$$\begin{bmatrix} {L_1}^{\mathbf{D}} & {L_2}^{\mathbf{F}} & \ldots & {L_{n-1}}^{\mathbf{F}} & {L_n}^{\mathbf{F}} \\ {L_1}^{\mathbf{B}} & {L_2}^{\mathbf{D}} & \ldots & {L_{n-1}}^{\mathbf{F}} & {L_n}^{\mathbf{F}} \\ \hdotsfor{5} \\ {L_1}^{\mathbf{B}} & {L_2}^{\mathbf{B}} & \ldots & {L_{n-1}}^{\mathbf{B}} & {L_n}^{\mathbf{D}} \end{bmatrix}$$

Incremental Clausal Maintenance Operator

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

Graph Query Complexity

Query Fragment	*Evaluation*	*Containment*
RPQ	NLOGSPACE-complete	PSPACE-complete
RPQ_C	$\#P-$complete	*Undefined*
2RPQ	NLOGSPACE-complete	PSPACE-complete
C2RPQ	NLOGSPACE-complete	EXPSPACE-complete
UC2RPQ	NLOGSPACE-complete	EXPSPACE-complete
nUC2RPQ	NLOGSPACE-complete	EXPSPACE-complete
UCN2RPQs	NLOGSPACE-complete	EXPSPACE-complete
RQ	NLOGSPACE-complete	2EXPSPACE-complete

Table: Graph query valuation and containment data complexity.

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., 2019] **Dumbrava, S.**, Bonifati, A., Nazabal, A., Vuillemont, R. 2019. Approximate Evaluation of Complex Queries on Property Graphs. In SUM, 250-265.
- [Bonifati al., 2018] Bonifati, A., **Dumbrava, S**. 2018. Graph Queries: From Theory to Practice. In Sigmod Record 47(4), 5–12.

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 Trans. on Knowledge and Data Engineering 29*, 856–869.
- [Diaz et al., 2020] Tomas Díaz, Federico Olmedo, Eric Tanter. 2020. A Mechanized Formalization of GQL. In *CPP*, 201–214.
- [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 *Th. of Computing Systems 61*, 31–83.