# Materializing Knowledge Bases via Trigger Graphs

Efthimia Tsamoura[1], David Carral[2], Enrico Malizia[3], and Jacopo Urbani[4]

1. Samsung AI Research
2. LIRMM, Inria, University of Montpellier, CNRS
3. University of Bologna
4. Vrije Universiteit Amsterdam

## Definition: Rules

A *rule* is a first-order formula of the form

$$\forall\vec{X}\forall\vec{Y} \bigwedge_{i=1}^{n} P_i(\vec{X}_i, \vec{Y}_i) \rightarrow \exists\vec{Z}.P(\vec{Y}, \vec{Z})$$

where $P$ is an IDB predicate, $\vec{Y} \subseteq \vec{X}$, and $\vec{X}_i \subseteq \vec{X}$ for all $1 \leq i \leq n$

## W.l.o.g. Assumptions

- Rules feature exactly one atom in the head.
- All rules are EDB or IDB:
  - IDB rule: all predicates in the body are IDB.
  - EDB rule: all predicates in the body are EDB.
- We assume that existentially quantified variables do not reoccur across different rules.

## Definition: Rule Application

Consider a rule

$$\rho = \bigwedge_{i=1}^{n} P_i(\vec{X}_i, \vec{Y}_i) \rightarrow \exists \vec{Z}.P(\vec{Y}, \vec{Z}),$$

a fact set $\mathcal{F}$, and a homomorphism $h : \bigwedge_{i=1}^{n} P_i(\vec{X}_i, \vec{Y}_i) \rightarrow \mathcal{F}$.

- We define $Appl(\mathcal{F}, \rho, h) = \mathcal{F} \cup \{h_s(P(\vec{Y}, \vec{Z}))\}$ where $h_s$ is the safe extension of $h$.
- We define $Appl(\mathcal{F}, \rho)$ as the set of facts that includes $Appl(\mathcal{F}, \rho, h)$ for all $h : \bigwedge_{i=1}^{n} P_i(\vec{X}_i, \vec{Y}_i) \rightarrow \mathcal{F}$.

# Execution Graphs: Definition

## Definition: Execution Graphs

An *execution graph* for a rule set $\mathcal{R}$ is an acyclic digraph $G = (V, E, \mathtt{r})$ such that $V$ is a finite set of vertices, $E$ is a finite set of *edges* of the form $v \rightarrow_i u$ with $v, u \in V$ and $i \geq 1$, and the following hold:

- The function $\mathtt{r}$ maps every vertex in $V$ to some rule in $\mathcal{R}$.
- If $\mathtt{r}(v)$ is an EDB rule for some $v \in V$, then $v$ is a root in $G$.
- Consider some $v \in V$ with $\mathtt{r}(v)$ an IDB rule of the form $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \exists \vec{y}.\beta$. Then, for every $1 \leq i \leq n$, there is exactly one vertex $u \in V$ with $u \rightarrow_i v \in E$.

## Remark

Consider an execution graph $G$ for a rule set $\mathcal{R}$.

- For a database $\mathcal{D}$, we define the fact set $G(\mathcal{D})$.
- The execution graph $G$ is a trigger graph for $\mathcal{R}$ if, for all BCQs $\gamma$,

$$\mathcal{R} \cup \mathcal{D} \models \gamma \iff G(\mathcal{D}) \models \gamma.$$

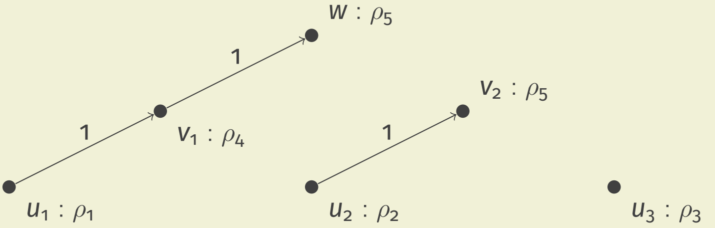
# Execution Graphs: An Example

## Example

Consider the database $\mathcal{D} = \{a(t), b(t), c(t)\}$ and the rule set $\mathcal{R}$:

$$
\begin{array}{lll}
\rho_1 = a(x) \to A(x) & \rho_2 = b(x) \to B(x) & \rho_3 = c(x) \to C(x) \\
\rho_4 = A(x) \to B(x) & \rho_5 = B(x) \to C(x) & \rho_6 = A(x) \to C(x)
\end{array}
$$

Then, the following structure is an execution graph for $\mathcal{R}$:



## Definition

Consider an execution graph $G = (V, E, \mathtt{r})$ for a rule set $\mathcal{R}$, and a database $\mathcal{D}$. Then, for a vertex $v \in V$, we inductively define the fact set $v(\mathcal{D})$ as follows:

- Case 1: if $v$ is a root in $G$, then $v(\mathcal{D}) = \mathit{Appl}(\mathcal{D}, \mathtt{r}(v))$.
- Case 2: $\mathtt{r}(v)$ is an IDB rule of the form $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \exists \vec{y}.\beta$ and, for every $1 \leq i \leq n$, there is exactly one vertex $u_i \in V$ with $u_i \rightarrow_i v \in E$. Then, $v(\mathcal{D})$ is the set that contains $h_s(\beta)$ for every homomorphism $h$ such that $h(\alpha_i) \subseteq u_i(\mathcal{D})$ for all $1 \leq i \leq n$.

We define $G(\mathcal{D}) = \bigcup_{v \in V} v(\mathcal{D}) \cup \mathcal{D}$.

## Remark

Consider a rule set $\mathcal{R}$, a database $\mathcal{D}$, and an execution graph $G$ for $\mathcal{R}$. Then, we can compute $G(\mathcal{D})$ if $G$ is finite.

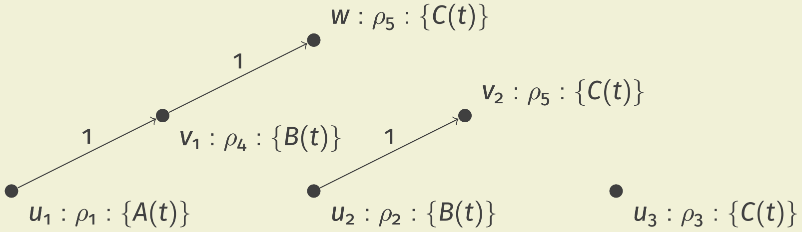
# Evaluating Execution Graphs: Example

## Example

Consider the database $\mathcal{D} = \{a(t), b(t), c(t)\}$ and the rule set $\mathcal{R}$:

$$\rho_1 = a(x) \to A(x) \qquad \rho_2 = b(x) \to B(x) \qquad \rho_3 = c(x) \to C(x)$$
$$\rho_4 = A(x) \to B(x) \qquad \rho_5 = B(x) \to C(x) \qquad \rho_6 = A(x) \to C(x)$$

Then, the following structure is an execution graph for $\mathcal{R}$:




To obtain $G(\mathcal{D})$, we inductively compute $v(\mathcal{D})$ for each vertex $v$.

# Trigger Graphs: Definition

## Definition: Trigger Graphs

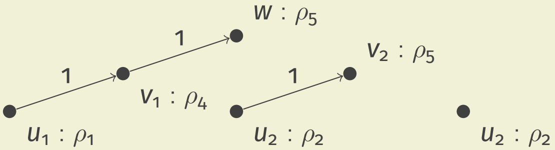
An execution graph $G$ for a rule set $\mathcal{R}$ is a *trigger graph* for $\mathcal{R}$ if, for all databases $\mathcal{D}$ and all queries $\gamma$, we have that $\mathcal{R} \cup \mathcal{D} \models \gamma$ iff $G(\mathcal{D}) \models \gamma$.

## Example

Consider the rule set $\mathcal{R}$:

$$\rho_1 = a(x) \rightarrow A(x) \qquad \rho_2 = b(x) \rightarrow B(x) \qquad \rho_3 = c(x) \rightarrow C(x)$$
$$\rho_4 = A(x) \rightarrow B(x) \qquad \rho_5 = B(x) \rightarrow C(x) \qquad \rho_6 = A(x) \rightarrow B(x)$$

Then, the following structure is a trigger graph for $\mathcal{R}$:



## Definition: Trigger Graphs

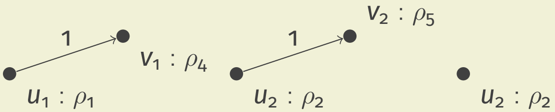
An execution graph $G$ for a rule set $\mathcal{R}$ is a *trigger graph* for $\mathcal{R}$ if, for all databases $\mathcal{D}$ and all queries $\gamma$, we have that $\mathcal{R} \cup \mathcal{D} \models \gamma$ iff $G(\mathcal{D}) \models \gamma$.

## Example

Consider the rule set $\mathcal{R}$:

$$\rho_1 = a(x) \to A(x) \qquad \rho_2 = b(x) \to B(x) \qquad \rho_3 = c(x) \to C(x)$$
$$\rho_4 = A(x) \to B(x) \qquad \rho_5 = B(x) \to C(x) \qquad \rho_6 = A(x) \to B(x)$$

The following structure $G$ is NOT a trigger graph for $\mathcal{R}$:




For instance, $\mathcal{R} \cup \{a(t)\} \models C(t)$ whilst $G(\{a(t)\}) \not\models C(t)$.

# Trigger Graphs: Output

**Remark**

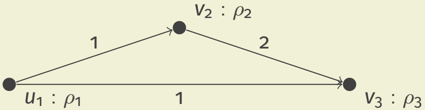
The output of a trigger graph $G$ for a rule set $\mathcal{R}$ on an input database $\mathcal{D}$ may not be a model for $\mathcal{R} \cup \mathcal{D}$.

**Example**

Consider following rule set $\mathcal{R}$:

$$\rho_1 = r(x) \to \exists y.R(x,y) \quad \rho_2 = R(x,y) \to \exists z.R(y,z) \quad \rho_3 = R(x,y) \wedge R(y,z) \to R(y,x)$$

Then, the following structure $G$ is a trigger graph for $\mathcal{R}$:




The set $G(r(a)) = \{r(a), R(a,n), R(n,m), R(n,a)\}$ is not a model for $\mathcal{R} \cup \{r(a)\}$.

**Proposition**

Given a trigger graph $G$ for a rule set $\mathcal{R}$ and a database $\mathcal{D}$, the core of $G(\mathcal{D})$ is a finite universal model for $\mathcal{R} \cup \mathcal{D}$.

## Remark

Not all rule sets admit (finite) trigger graphs. For instance, neither $\{R(x,y) \to \exists z.R(y,z)\}$ nor $\{R(x,y) \wedge R(y,z) \to R(x,z)\}$ admit such graphs!

## Theorem (+ Definition)

A rule set $\mathcal{R}$ admits a trigger graph iff it is universally bounded; that is, if there is some $k$ such that, for every database $\mathcal{D}$ and every BCQ $\gamma$, we have that $\mathcal{R} \cup \mathcal{D} \models \gamma$ iff $Chase_k(\mathcal{R}, \mathcal{D}) \models \gamma$,

Proof Sketch:

$\Longrightarrow$ If a rule set $\mathcal{R}$ admits a trigger graph $G$ of depth $k$, then $\mathcal{R}$ is universally bounded by $k$.

$\Longleftarrow$ If a rule set $\mathcal{R}$ is bound by $k$, then we can construct a "maximal" trigger graph of depth $k$ that derives all of the facts produced by the chase up to the $k$-th step.

## Corollary

A rule set admits a trigger graph iff it is FUS/FO-rewritable and terminates with respect to the skolem chase.

# Trigger Graphs for Linear Rule Sets

**Algorithm 1** tglinear($\mathcal{R}$)

Let $G$ be an empty EG
**for each** $\varphi \in \mathcal{H}(\mathcal{R})$ **do**
  $\Gamma = (V, E, \mathsf{r})$ is an empty $EG$; $\mu$ is the empty mapping
  **for each** $\varphi_1 \rightarrow_\rho \varphi_2 \in$ chaseGraph$(\mathcal{R}, \{\varphi\})$ **do**
    **add** a fresh node $u$ to $V$ with $\mathsf{r}(u) := \rho$
    $\mu(u) := \varphi_1 \rightarrow_\rho \varphi_2$
  **for each** $v, u \in V$ **do**
    **if** $\mu(v) = \varphi_1 \rightarrow_\rho \varphi_2$ **and** $\mu(u) = \varphi_2 \rightarrow_{\rho'} \varphi_3$ **then**
      **add** $v \rightarrow_1 u$ to $E$
  $G := G \cup \Gamma$
**return** $G$

- $\mathcal{H}(\mathcal{R})$ is a maximal set of facts formed over the predicates in $\mathcal{R}$, where no $\varphi_1 \in \mathcal{H}(\mathcal{R})$ is pattern isomorphic to another $\varphi_2 \in \mathcal{H}(\mathcal{R})$.
- Let *chaseGraph*$(\mathcal{R}, \mathcal{D})$ as the acyclic graph having as nodes the facts in the chase of $(\mathcal{R}, \mathcal{D})$ and having an edge from $\varphi_1$ to $\varphi_2$ labeled with rule $\rho \in \mathcal{R}$ if $\varphi_2$ is obtained from $\varphi_1$ by executing $\rho$.

# Minimizing Trigger Graphs for Linear Rule Sets

Consider a linear rule set $\mathcal{R}$, a trigger graph $G = (V, E, \mathtt{r})$ for $\mathcal{R}$, and some $u, v \in V$.

**Definition**

- For a database $\mathcal{D}$, a homomorphism from $u(\mathcal{D})$ into $v(\mathcal{D})$ is *preserving* if it is the identity over the set of nulls in any fact set associated with an ancestor of $u$.
- The vertex $v \in V$ is dominated by another vertex $u \in V$ if there is a preserving homomorphism from $v(\{\varphi\})$ to $u(\{\varphi\})$ for all $\varphi \in \mathcal{H}(P)$.

**Lemma**

There is a *preserving homomorphism* from $u(\mathcal{D})$ into $v(\mathcal{D})$ for all databases $\mathcal{D}$ iff there is a preserving homomorphism from $u(\{\varphi\})$ into $v(\{\varphi\})$ for all $\varphi \in \mathcal{H}(P)$.

**Lemma**

Assume that $v$ is dominated by $u$ and $u$ is not a successor of $v$. Then, the following transformation produces a trigger graph for $\mathcal{R}$:

- Add an edge $u \rightarrow_1 w$ for every edge of the form $v \rightarrow_1 w \in E$.
- Remove the vertix $v$ and all edges where this vertex occurs from $G$.

The above process can be exhaustively repeated to obtain a minimal trigger graph for $\mathcal{R}$.

# Trigger Graphs: Practical Contribution

## Empirical Claim

Trigger graphs can be used to develop a very efficient implementation of the chase algorithm.

Some (cherry-picked) results that support our claim:

| | VLog | RDFOx | GLog |
|---|---|---|---|
| LUBM | 170s | 115s | 16s |
| DBpedia | 41s | 198s | 19s |
| Claros | 431s | 2373s | 122s |

## Remarks

- The above theories only contain Datalog rules.
- When using trigger graphs, we only remove duplicates at the end.

# Trigger Graphs: Future Work

**Remark**

We can only compute finite trigger graphs for universally bounded rule sets. For instance, rule sets as simple as $\{R(x,y) \rightarrow \exists z.R(y,z)\}$ nor $\{R(x,y) \wedge R(y,z) \rightarrow R(x,z)\}$ do not admit trigger graphs!

One possible solution is to reconsider the definition of *trigger graphs*:

**Hypotheses**

- Every DL/BTS rule set admits a cyclic trigger graph in which all vertices occurring in a cycle are labelled with Datalog rules.
- All FUS rule sets admit acyclic trigger graphs that are complete for fact entailment.

"DL = Description Logics" and "BTS = Bounded Treewidth Set"

# Trigger Graphs: Future Work

The use of cyclic trigger graphs can allow us to implement well-known optimisations:

## Example: Computing Transitivity

Consider the rule set

$$\begin{aligned}\mathcal{R} = \{\rho_1 &= r(x,y) \rightarrow R(x,y),\\ \rho_2 &= R(x,y) \wedge R(y,z) \rightarrow R(x,y)\}.\end{aligned}$$

Then, $G = (V, E, \mathrm{r})$ is a trigger graph for $\mathcal{R}$.

$$\begin{aligned}V &= \{v_1, v_2\}\\ E &= \{(v_1, v_1, v_2), (v_1, v_2, v_2)\}\\ \mathrm{r} &= \{v_1 \mapsto \rho_1, v_2 \mapsto \rho_2\}\end{aligned}$$

# Thank you for your attention!