A tutorial on Database Theory
and a talk on
database query answering under updates

Nicole Schweikardt

Humboldt-Universität zu Berlin

24th Workshop on Logic, Language, Information and Computation (WoLLIC 2017)
London, July 19 & 20, 2017
### Movie

<table>
<thead>
<tr>
<th>Name</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

### Programme

<table>
<thead>
<tr>
<th>Cinema</th>
<th>Movietitle</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babylon</td>
<td>Casablanca</td>
<td>17:30</td>
</tr>
<tr>
<td>Babylon</td>
<td>Gravity</td>
<td>20:15</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Blade Runner</td>
<td>15:30</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Alien</td>
<td>18:15</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Blade Runner</td>
<td>20:30</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Resident Evil</td>
<td>20:30</td>
</tr>
<tr>
<td>Kino International</td>
<td>Casablanca</td>
<td>18:00</td>
</tr>
<tr>
<td>Kino International</td>
<td>Brazil</td>
<td>20:00</td>
</tr>
<tr>
<td>Kino International</td>
<td>Brazil</td>
<td>22:00</td>
</tr>
<tr>
<td>Movimento</td>
<td>Gravity</td>
<td>17:00</td>
</tr>
<tr>
<td>Movimento</td>
<td>Gravity</td>
<td>19:30</td>
</tr>
<tr>
<td>Movimento</td>
<td>Alien</td>
<td>22:00</td>
</tr>
<tr>
<td>Urania</td>
<td>Resident Evil</td>
<td>20:00</td>
</tr>
<tr>
<td>Urania</td>
<td>Resident Evil</td>
<td>21:30</td>
</tr>
<tr>
<td>Urania</td>
<td>Resident Evil</td>
<td>23:00</td>
</tr>
</tbody>
</table>
Example database and two queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Gravity</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programme</th>
<th>Movietitle</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babylon</td>
<td>Casablanca</td>
<td>17:30</td>
</tr>
<tr>
<td>Babylon</td>
<td>Gravity</td>
<td>20:15</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Blade Runner</td>
<td>15:30</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Alien</td>
<td>18:15</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Blade Runner</td>
<td>20:30</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Resident Evil</td>
<td>20:30</td>
</tr>
<tr>
<td>Kino International</td>
<td>Casablanca</td>
<td>18:00</td>
</tr>
<tr>
<td>Kino International</td>
<td>Brazil</td>
<td>20:00</td>
</tr>
<tr>
<td>Kino International</td>
<td>Brazil</td>
<td>22:00</td>
</tr>
<tr>
<td>Movimento</td>
<td>Gravity</td>
<td>17:00</td>
</tr>
<tr>
<td>Movimento</td>
<td>Gravity</td>
<td>19:30</td>
</tr>
<tr>
<td>Movimento</td>
<td>Alien</td>
<td>22:00</td>
</tr>
<tr>
<td>Urania</td>
<td>Resident Evil</td>
<td>20:00</td>
</tr>
<tr>
<td>Urania</td>
<td>Resident Evil</td>
<td>21:30</td>
</tr>
<tr>
<td>Urania</td>
<td>Resident Evil</td>
<td>23:00</td>
</tr>
</tbody>
</table>

Return all titles of movies $y$ in which Sigourney Weaver stars:

$\varphi_1(y) := \text{Movie}(y, "Sigourney Weaver")$
Example database and two queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinema</td>
</tr>
<tr>
<td>Babylon</td>
</tr>
<tr>
<td>Babylon</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Kino International</td>
</tr>
<tr>
<td>Kino International</td>
</tr>
<tr>
<td>Kino International</td>
</tr>
<tr>
<td>Movimento</td>
</tr>
<tr>
<td>Movimento</td>
</tr>
<tr>
<td>Movimento</td>
</tr>
<tr>
<td>Urania</td>
</tr>
<tr>
<td>Urania</td>
</tr>
<tr>
<td>Urania</td>
</tr>
</tbody>
</table>

Return all titles of movies \( y \) in which Sigourney Weaver stars:
\[
\varphi_1(y) := \text{Movie}(y, "Sigourney Weaver")
\]

Return all tuples \((x, y)\) of cinemas \(x\) and movie titles \(y\) such that \(x\) plays movie \(y\) in which Sigourney Weaver stars:
\[
\varphi_2(x, y) := \exists z \left( \text{Programme}(x, y, z) \land \text{Movie}(y, "Sigourney Weaver") \right)
\]
Example database and two queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinema</td>
</tr>
<tr>
<td>Babylon</td>
</tr>
<tr>
<td>Babylon</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Casablanca</td>
</tr>
<tr>
<td>Kino International</td>
</tr>
<tr>
<td>Kino International</td>
</tr>
<tr>
<td>Kino International</td>
</tr>
<tr>
<td>Movimento</td>
</tr>
<tr>
<td>Movimento</td>
</tr>
<tr>
<td>Movimento</td>
</tr>
<tr>
<td>Urania</td>
</tr>
<tr>
<td>Urania</td>
</tr>
<tr>
<td>Urania</td>
</tr>
</tbody>
</table>

Return all titles of movies $y$ in which Sigourney Weaver stars: 

$\varphi_1(y) := \text{Movie}(y, "Sigourney Weaver")$

Return all tuples $(x, y)$ of cinemas $x$ and movie titles $y$ such that $x$ plays movie $y$ in which Sigourney Weaver stars: 

$\varphi_2(x, y) := \exists z \ (\text{Programme}(x, y, z) \land \text{Movie}(y, "Sigourney Weaver"))$

Conjunctive queries!
A logician’s point of view:

\[
\begin{align*}
\text{Movie} & : \text{a 2-ary relation symbol } M \\
\text{Programme} & : \text{a 3-ary relation symbol } P
\end{align*}
\]

Return all titles of movies \( y \) in which Sigourney Weaver stars:

\[
\varphi_1(y) := \text{Movie}(y, "Sigourney Weaver")
\]

Return all tuples \((x, y)\) of cinemas \( x \) and movie titles \( y \) such that \( x \) plays movie \( y \) in which Sigourney Weaver stars:

\[
\varphi_2(x, y) := \exists z \left( \text{Programme}(x, y, z) \land \text{Movie}(y, "Sigourney Weaver") \right)
\]

Conjunctive queries!
Example database and two queries

A logician’s point of view:

\[ Movie \ : \ \text{a 2-ary relation symbol } M \]
\[ Programme \ : \ \text{a 3-ary relation symbol } P \]
\[ \text{database schema} \ : \ \text{relational signature } \sigma \ := \{M, D\} \]

Return all titles of movies \( y \) in which Sigourney Weaver stars:
\[ \varphi_1(y) := Movie(y, "Sigourney Weaver") \]

Return all tuples \((x, y)\) of cinemas \(x\) and movie titles \(y\) such that \(x\) plays movie \(y\) in which Sigourney Weaver stars:
\[ \varphi_2(x, y) := \exists z \ (Programme(x, y, z) \land Movie(y, "Sigourney Weaver")) \]

Conjunctive queries!
Return all titles of movies $y$ in which Sigourney Weaver stars:

$$\varphi_1(y) := \text{Movie}(y, "Sigourney Weaver")$$

Return all tuples $(x, y)$ of cinemas $x$ and movie titles $y$ such that $x$ plays movie $y$ in which Sigourney Weaver stars:

$$\varphi_2(x, y) := \exists z \ (\text{Programme}(x, y, z) \land \text{Movie}(y, "Sigourney Weaver"))$$

Conjunctive queries!
Example database and two queries

A logician’s point of view:

\[ \text{Movie} : \text{a 2-ary relation symbol } M \]
\[ \text{Programme} : \text{a 3-ary relation symbol } P \]

database schema : relational signature \( \sigma := \{ M, D \} \)
a db : \( D = (M^D, P^D) \), where
\( M^D \) : a finite subset of \( \text{dom}^2 \)
\( P^D \) : a finite subset of \( \text{dom}^3 \)
\( \text{dom} \) : a fixed, infinite domain of potential db entries
\( \text{adom}(D) \) : the set of all \( d \in \text{dom} \) that occur in \( M^D \) or \( P^D \)

Return all titles of movies \( y \) in which Sigourney Weaver stars:
\[ \varphi_1(y) := \text{Movie}(y, "Sigourney Weaver") \]

Return all tuples \( (x, y) \) of cinemas \( x \) and movie titles \( y \) such that \( x \) plays movie \( y \) in which Sigourney Weaver stars:
\[ \varphi_2(x, y) := \exists z \left( \text{Programme}(x, y, z) \land \text{Movie}(y, "Sigourney Weaver") \right) \]

Conjunctive queries!
Example database and two queries

A logician’s point of view:

\[ \text{Movie} : \text{a 2-ary relation symbol } M \]
\[ \text{Programme} : \text{a 3-ary relation symbol } P \]

database schema : relational signature \( \sigma := \{ M, D \} \)
a db : \( D = (M^D, P^D) \), where
\[ M^D : \text{a finite subset of } \text{dom}^2 \]
\[ P^D : \text{a finite subset of } \text{dom}^3 \]
\[ \text{dom} : \text{a fixed, infinite domain of potential db entries} \]
\[ \text{adom}(D) : \text{the set of all } d \in \text{dom} \text{ that occur in } M^D \text{ or } P^D \]

View \( D \) as a finite \( \sigma \)-structure with universe \( \text{adom}(D) \)!

Return all titles of movies \( y \) in which Sigourney Weaver stars:
\[ \varphi_1(y) := \text{Movie}(y, "\text{Sigourney Weaver}") \]

Return all tuples \((x, y)\) of cinemas \( x \) and movie titles \( y \) such that \( x \) plays movie \( y \) in which Sigourney Weaver stars:
\[ \varphi_2(x, y) := \exists z \left( \text{Programme}(x, y, z) \land \text{Movie}(y, "\text{Sigourney Weaver}") \right) \]

Conjunctive queries!

Nicole Schweikardt (HU Berlin) Database Theory and Query Answering under Updates 2/ 43
Example database and two queries

A logician’s point of view:

\[ \text{Movie} : \text{a 2-ary relation symbol } M \]
\[ \text{Programme} : \text{a 3-ary relation symbol } P \]
\[ \text{database schema} : \text{relational signature } \sigma := \{M, D\} \]
\[ \text{a db} : D = (M^D, P^D), \text{where} \]
\[ M^D : \text{a finite subset of } \text{dom}^2 \]
\[ P^D : \text{a finite subset of } \text{dom}^3 \]
\[ \text{dom} : \text{a fixed, infinite domain of potential db entries} \]
\[ \text{adom}(D) : \text{the set of all } d \in \text{dom} \text{ that occur in } M^D \text{ or } P^D \]

View \( D \) as a finite \( \sigma \)-structure with universe \( \text{adom}(D) \)!

Return all titles of movies \( y \) in which Sigourney Weaver stars:
\[ \varphi_1(y) := M(y, \text{"Sigourney Weaver"}) \]

Return all tuples \((x, y)\) of cinemas \( x \) and movie titles \( y \) such that \( x \) plays movie \( y \) in which Sigourney Weaver stars:
\[ \varphi_2(x, y) := \exists z \ (\text{Programme}(x, y, z) \land \text{Movie}(y, \text{"Sigourney Weaver"})) \]

Conjunctive queries!
Example database and two queries

A logician’s point of view:

- **Movie**: a 2-ary relation symbol \( M \)
- **Programme**: a 3-ary relation symbol \( P \)

Database schema: relational signature \( \sigma := \{ M, D \} \)

- A db: \( D = (M^D, P^D) \), where
- \( M^D \): a finite subset of \( \text{dom}^2 \)
- \( P^D \): a finite subset of \( \text{dom}^3 \)
- \( \text{dom} \): a fixed, infinite domain of potential db entries
- \( \text{adom}(D) \): the set of all \( d \in \text{dom} \) that occur in \( M^D \) or \( P^D \)

View \( D \) as a finite \( \sigma \)-structure with universe \( \text{adom}(D) \)!

Return all titles of movies \( y \) in which Sigourney Weaver stars:

\[ \varphi_1(y) := M(y, "Sigourney Weaver") \]

Return all tuples \((x, y)\) of cinemas \( x \) and movie titles \( y \) such that \( x \) plays movie \( y \) in which Sigourney Weaver stars:

\[ \varphi_2(x, y) := \exists z \ ( P(x, y, z) \land M(y, "Sigourney Weaver")) \]

Conjunctive queries!

Nicole Schweikardt (HU Berlin) Database Theory and Query Answering under Updates
Query evaluation

Consider a query language $L$ (e.g., SQL, conjunctive queries CQ, first-order logic FO).

Let $\varphi(x_1, \ldots, x_k)$ be a query of signature $\sigma$, formulated in $L$. Let $D$ be a database of signature $\sigma$.

Task:
Evaluate $\varphi(x_1, \ldots, x_k)$ on $D$
Query evaluation

Consider a query language $L$ (e.g., SQL, conjunctive queries CQ, first-order logic FO).

Let $\varphi(x_1, \ldots, x_k)$ be a query of signature $\sigma$, formulated in $L$. Let $D$ be a database of signature $\sigma$.

**Task:** Evaluate $\varphi(x_1, \ldots, x_k)$ on $D$, i.e., compute the set

$$\varphi(D) := \models \varphi(x_1, \ldots, x_k)(D) :=$$

$$\{ (a_1, \ldots, a_k) \in \text{dom}(D)^k : (\text{dom}(D), D) \models \varphi \left[ \frac{a_1 \cdots a_k}{x_1 \cdots x_k} \right] \}$$

Special case $k = 0$: Boolean queries: Evaluate $\varphi()$ on $D$ means Decide if $(\text{dom}(D), D) \models \varphi$.
Query evaluation

Consider a query language $L$ (e.g., SQL, conjunctive queries CQ, first-order logic FO).

Let $\varphi(x_1, \ldots, x_k)$ be a query of signature $\sigma$, formulated in $L$. Let $D$ be a database of signature $\sigma$.

Task:
Evaluate $\varphi(x_1, \ldots, x_k)$ on $D$, i.e., compute the set $\varphi(D) := \llbracket \varphi(x_1, \ldots, x_k) \rrbracket(D) :=$

$$\{ (a_1, \ldots, a_k) \in \text{adom}(D)^k : (\text{adom}(D), D) \models \varphi \left[ \begin{array}{c} a_1 \cdots a_k \\ x_1 \cdots x_k \end{array} \right] \}$$

Special case $k = 0$: Boolean queries:
Evaluate $\varphi()$ on $D$ means Decide if $(\text{adom}(D), D) \models \varphi$
Complexity of query evaluation

In his *STOC’82* paper, *Moshe Vardi* introduced the notions:

- combined complexity

and data complexity

Typical results obtained in database theory:

- **Boolean Conjunctive Queries:** data complexity is in AC0, combined complexity is NP-complete [Chandra & Merlin ’77]

- **Boolean First-Order Queries:** data complexity is in AC0, combined complexity is PSPACE-complete [Stockmeyer ’74, Vardi ’82]

- **Boolean Least-Fixed Point Queries:** data complexity is PTIME-complete, combined complexity is EXPTIME-complete [Immerman ’82, Vardi ’82].

CAVEAT: These notions & results cannot handle updates of the db!
Complexity of query evaluation

In his STOC’82 paper, Moshe Vardi introduced the notions

combined complexity: Measure the complexity of evaluating $\varphi$ on $D$ in terms of the sizes of $\varphi$ and $D$.

and data complexity

CAVEAT: These notions & results cannot handle updates of the db!
Complexity of query evaluation

In his STOC’82 paper, Moshe Vardi introduced the notions

combined complexity: Measure the complexity of evaluating $\varphi$ on $D$ in terms of the sizes of $\varphi$ and $D$.

and data complexity: Assume the query $\varphi$ to be fixed. Measure the complexity of evaluating $\varphi$ on $D$ only in terms of the size of $D$.

Typical results obtained in database theory:

▶ Boolean Conjunctive Queries: data complexity is in $\text{AC}^0$, combined complexity is $\text{NP}$-complete [Chandra & Merlin ‘77]

▶ Boolean First-Order Queries: data complexity is in $\text{AC}^0$, combined complexity is $\text{PSPACE}$-complete [Stockmeyer ‘74, Vardi ‘82]

▶ Boolean Least-Fixed Point Queries: data complexity is $\text{PTIME}$-complete, combined complexity is $\text{EXPTIME}$-complete [Immerman ‘82, Vardi ‘82].

CAVEAT: These notions & results cannot handle updates of the db!
Complexity of query evaluation

In his STOC’82 paper, Moshe Vardi introduced the notions

combined complexity: Measure the complexity of evaluating $\varphi$ on $D$ in terms of the sizes of $\varphi$ and $D$.

and data complexity: Assume the query $\varphi$ to be fixed. Measure the complexity of evaluating $\varphi$ on $D$ only in terms of the size of $D$.

Typical results obtained in database theory:

- **Boolean Conjunctive Queries**: data complexity is in $\text{AC}^0$, combined complexity is $\text{NP}$-complete [Chandra & Merlin ’77]

- **Boolean First-Order Queries**: data complexity is in $\text{AC}^0$, combined complexity is $\text{PSPACE}$-complete [Stockmeyer ’74, Vardi ’82]

- **Boolean Least-Fixed Point Queries**: data complexity is $\text{PTIME}$-complete, combined complexity is $\text{EXPTIME}$-complete [Immerman ’82, Vardi ’82].

CAVEAT: These notions & results cannot handle updates of the db!
Complexity of query evaluation

In his STOC’82 paper, Moshe Vardi introduced the notions

**combined complexity:** Measure the complexity of evaluating $\varphi$ on $D$ in terms of the sizes of $\varphi$ and $D$.

and **data complexity:** Assume the query $\varphi$ to be fixed. Measure the complexity of evaluating $\varphi$ on $D$ only in terms of the size of $D$.

Typical results obtained in database theory:

- **Boolean Conjunctive Queries:** data complexity is in $\text{AC}^0$, combined complexity is $\text{NP}$-complete [Chandra & Merlin ’77]

- **Boolean First-Order Queries:** data complexity is in $\text{AC}^0$, combined complexity is $\text{PSPACE}$-complete [Stockmeyer ’74, Vardi ’82]

CAVEAT: These notions & results cannot handle updates of the db!
Complexity of query evaluation

In his STOC’82 paper, Moshe Vardi introduced the notions

combined complexity: Measure the complexity of evaluating $\phi$ on $D$ in terms of the sizes of $\phi$ and $D$.

and data complexity: Assume the query $\phi$ to be fixed. Measure the complexity of evaluating $\phi$ on $D$ only in terms of the size of $D$.

Typical results obtained in database theory:

- **Boolean Conjunctive Queries**: data complexity is in AC$^0$, combined complexity is NP-complete [Chandra & Merlin ’77]
- **Boolean First-Order Queries**: data complexity is in AC$^0$, combined complexity is PSPACE-complete [Stockmeyer ’74, Vardi ’82]
- **Boolean Least-Fixed Point Queries**: data complexity is PTIME-complete, combined complexity is EXPTIME-complete [Immerman ’82, Vardi ’82].

CAVEAT: These notions & results cannot handle updates of the db!
Complexity of query evaluation

In his STOC’82 paper, Moshe Vardi introduced the notions

combined complexity: Measure the complexity of evaluating $\varphi$ on $D$ in terms of the sizes of $\varphi$ and $D$.

and data complexity: Assume the query $\varphi$ to be fixed. Measure the complexity of evaluating $\varphi$ on $D$ only in terms of the size of $D$.

Typical results obtained in database theory:

- **Boolean Conjunctive Queries**: data complexity is in $AC^0$, combined complexity is NP-complete [Chandra & Merlin ’77]
- **Boolean First-Order Queries**: data complexity is in $AC^0$, combined complexity is PSPACE-complete [Stockmeyer ’74, Vardi ’82]
- **Boolean Least-Fixed Point Queries**: data complexity is PTIME-complete, combined complexity is EXPTIME-complete [Immerman ’82, Vardi ’82].

**CAVEAT**: These notions & results cannot handle updates of the db!
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:** Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries: Decide if $D |= \varphi$
  - For $k$-ary queries: Compute the number of tuples in $\varphi(D)$
  - Test for a given tuple $a$ whether $a \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:** update data structure in

  Tuples may be inserted into or deleted from $D$

  Similar results for FO with counting $FOC$ (P) [Kuske, S., LICS'17]

- **Future task:** Revisit other results on FO model checking in the dynamic setting!
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- For Boolean queries:
  - Decide if $D \models \varphi$

- For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$
  - Test for a given tuple $a$ whether $a \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Update data structure in $T$.

Similar results for FO with counting $\text{FOC}$ ($\text{P}$) [Kuske, S., LICS’17].

Future task:
Revisit other results on FO model checking in the dynamic setting!
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  For Boolean queries:
  - Decide if $D \models \varphi$

- For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$
  - Test for a given tuple $a$ whether $a \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

- Dynamic setting:
  Update data structure in $T$
  Tuples may be inserted into or deleted from $D$
  Similar results for FO with counting FOC ($P$)
  [Kuske, S., LICS'17]

Future task:
Revisit other results on FO model checking in the dynamic setting!

– Thank you! –
Nicole Schweikardt (HU Berlin) Database Theory and Query Answering under Updates 5/ 43
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  For Boolean queries:
  - Decide if $D \models \varphi$
  For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$
  - Test for a given tuple $a$ whether $a \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

- Dynamic setting:
  Update data structure in $T$
  Tuples may be inserted into or deleted from $D$
  Similar results for FO with counting FOC

- Future task:
  Revisit other results on FO model checking in the dynamic setting!
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - Query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$

---

---

---
A typical scenario for DB-systems

- **Input:**
  - Database \( D \)
  - query \( \varphi(x_1, \ldots, x_k) \)

- **Preprocessing:**
  Build a suitable data structure that represents \( D \) and \( \varphi(D) \)

- **Output:**
  For Boolean queries:
  - Decide if \( D \models \varphi \)
  For \( k \)-ary queries:
  - Compute the number of tuples in \( \varphi(D) \)
  - Test for a given tuple \( \bar{a} \) whether \( \bar{a} \in \varphi(D) \)
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$
A typical scenario for DB-systems

- **Input:**
  - Database $D$
  - query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
Overview

Introduction

Conjunctive Queries on Arbitrary Databases

First-Order Queries on Bounded Degree Databases
Overview

Introduction

Conjunctive Queries on Arbitrary Databases

First-Order Queries on Bounded Degree Databases
Conjunctive queries (CQs)

Conjunctive queries:

\[ \varphi(x_1, \ldots, x_\ell) := \exists x_{\ell+1} \cdots \exists x_m \left( R_1(\overline{x}) \land \cdots \land R_s(\overline{x}) \right) \]
Conjunctive queries (CQs)

Conjunctive queries:

\[ \varphi(x_1, \ldots, x_\ell) := \exists x_{\ell+1} \cdots \exists x_m \left( R_1(\overline{x}) \land \cdots \land R_s(\overline{x}) \right) \]

Complexity of query evaluation:

Obvious: \( \text{static time} \leq \text{update time} \cdot \|D\| \)
Conjunctive queries (CQs)

Conjunctive queries:

\[ \varphi(x_1, \ldots, x_\ell) := \exists x_{\ell+1} \cdots \exists x_m \left( R_1(x) \land \cdots \land R_s(x) \right) \]

**Complexity of query evaluation:**

Obvious: static time \( \leq \) update time \( \cdot \|D\| \)

Thus:

- constant update time \( \implies \) static setting has linear data complexity
Conjunctive queries (CQs)

Conjunctive queries:

\[ \varphi(x_1, \ldots, x_\ell) := \exists x_{\ell+1} \cdots \exists x_m \left( R_1(x) \land \cdots \land R_s(x) \right) \]

Complexity of query evaluation:

Obvious: static time \( \leq \) update time \( \cdot \|D\| \)

Thus:

- constant update time \( \implies \) static setting has linear data complexity
- \( n^{O(1)} \) update time \( \iff \) \( n^{O(1)} \) static time
Conjunctive queries (CQs)

Conjunctive queries:

\( \varphi(x_1, \ldots, x_\ell) := \exists x_{\ell+1} \cdots \exists x_m (R_1(\bar{x}) \land \cdots \land R_s(\bar{x})) \)

Complexity of query evaluation:

Obvious: static time \( \leq \) update time \( \cdot \|D\| \)

Thus:

- constant update time \( \implies \) static setting has linear data complexity
- \( n^{O(1)} \) update time \( \iff \) \( n^{O(1)} \) static time

For the static setting: tight characterisation of the tractable CQs:

Boolean: [Grohe, Schwentick, Segoufin 2001], [Grohe 2007], [Marx 2010], [Marx 2013]

counting: [Dalmau, Jonsson 2004], [Chen, Mengel 2015], [Greco, Scarcello 2015]

enumeration: [Bulatov et al. 2012], [Bagan, Durand, Grandjean 2007]

I.e.: Update time \( n^{O(1)} \) is well-understood!
Conjunctive queries (CQs)

Conjunctive queries:

\[ \varphi(x_1, \ldots, x_\ell) := \exists x_{\ell+1} \cdots \exists x_m \left( R_1(x) \land \cdots \land R_s(x) \right) \]

Complexity of query evaluation:

Obvious: static time \( \leq \) update time \( \cdot \|D\| \)

Thus:

- constant update time \( \implies \) static setting has linear data complexity
- \( n^{O(1)} \) update time \( \iff \) \( n^{O(1)} \) static time

For the static setting: tight characterisation of the tractable CQs:

- Boolean: [Grohe, Schwentick, Segoufin 2001], [Grohe 2007], [Marx 2010], [Marx 2013]
- counting: [Dalmau, Jonsson 2004], [Chen, Mengel 2015], [Greco, Scarcello 2015]
- enumeration: [Bulatov et al. 2012], [Bagan, Durand, Grandjean 2007]

I.e.: Update time \( n^{O(1)} \) is well-understood!

Interesting: Sub-linear update time
Scenario

- **Input:**
  - Database \( D \) arbitrary
  - query \( \varphi(x_1, \ldots, x_k) \) CQ

- **Preprocessing:**
  Build a suitable data structure that represents \( D \) and \( \varphi(D) \)

- **Output:**
  For Boolean queries:
  - Decide if \( D \models \varphi \)
  For \( k \)-ary queries:
  - Compute the number of tuples in \( \varphi(D) \)
  - Test for a given tuple \( \bar{a} \) whether \( \bar{a} \in \varphi(D) \)
  - Enumerate the tuples in \( \varphi(D) \)

- **Dynamic setting:**
  Tuples may be inserted into or deleted from \( D \)
  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog(\( \|D\| \)).
Scenario

- **Input:**
  - Database $D$, arbitrary
  - query $\varphi(x_1, \ldots, x_k)$, CQ

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  For Boolean queries:
  - Decide if $D \models \varphi$
  For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$
  - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$

  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
**q-hierarchical CQs**

Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

Definition:
A CQ \( \varphi(z_1,\ldots,z_k) \) is q-hierarchical if for all variables \( x, y \) of \( \varphi \) the following is satisfied:

(i) \( \text{atoms}(x) \subseteq \text{atoms}(y) \) or \( \text{atoms}(y) \subseteq \text{atoms}(x) \) or \( \text{atoms}(x) \cap \text{atoms}(y) = \emptyset \), and

(ii) if \( \text{atoms}(x) \subset \text{atoms}(y) \) and \( x \in \text{free}(\varphi) \), then \( y \in \text{free}(\varphi) \).

Queries that are not q-hierarchical:
\[
\psi_{S-E-T}() := \exists x \exists y (S(x) \land E(x,y) \land T(y))
\]
\[
\varphi_{S-E-T}(x,y) := S(x) \land E(x,y) \land T(y)
\]
\[
\varphi_{E-T}(x) := \exists y (E(x,y) \land T(y))
\]

A q-hierarchical query:
\[
\theta_{E-T}(y) := \exists x (E(x,y) \land T(y))
\]
Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

q-hierarchical CQs are hierarchical CQs where, additionally, the quantifiers respect the query’s hierarchical form.

**Definition:**
A CQ $\phi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables $x, y$ of $\phi$ the following is satisfied:

1. $\text{atoms}(x) \subseteq \text{atoms}(y)$ or $\text{atoms}(y) \subseteq \text{atoms}(x)$ or $\text{atoms}(x) \cap \text{atoms}(y) = \emptyset$, and
2. if $\text{atoms}(x) \subset \text{atoms}(y)$ and $x \in \text{free}(\phi)$, then $y \in \text{free}(\phi)$.

**Queries that are not q-hierarchical:**

$\psi_{S-E-T}(x) := \exists x \exists y (S(x) \land E(x, y) \land T(y))$

$\phi_{S-E-T}(x, y) := S(x) \land E(x, y) \land T(y)$

$\phi_{E-T}(x) := \exists y (E(x, y) \land T(y))$

A q-hierarchical query:

$\theta_{E-T}(y) := \exists x (E(x, y) \land T(y))$
q-hierarchical CQs

Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

q-hierarchical CQs are hierarchical CQs where, additionally, the quantifiers respect the query’s hierarchical form.

Definition: A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables $x, y$ of $\varphi$ the following is satisfied:

(i) $\text{atoms}(x) \subseteq \text{atoms}(y)$ or $\text{atoms}(y) \subseteq \text{atoms}(x)$ or
$\text{atoms}(x) \cap \text{atoms}(y) = \emptyset$, and

(ii) if $\text{atoms}(x) \not\subseteq \text{atoms}(y)$ and $x \in \text{free}(\varphi)$, then $y \in \text{free}(\varphi)$. 
q-hierarchical CQs

Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

q-hierarchical CQs are hierarchical CQs where, additionally, the quantifiers respect the query’s hierarchical form.

Definition: A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables $x, y$ of $\varphi$ the following is satisfied:

(i) $\text{atoms}(x) \subseteq \text{atoms}(y)$ or $\text{atoms}(y) \subseteq \text{atoms}(x)$ or $\text{atoms}(x) \cap \text{atoms}(y) = \emptyset$, and

(ii) if $\text{atoms}(x) \varsubsetneq \text{atoms}(y)$ and $x \in \text{free}(\varphi)$, then $y \in \text{free}(\varphi)$.

Queries that are not q-hierarchical:

$$\psi_{S\cdot E\cdot T}() := \exists x \exists y \left( S(x) \land E(x, y) \land T(y) \right)$$
Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

**q-hierarchical CQs** are hierarchical CQs where, additionally, the quantifiers respect the query’s hierarchical form.

**Definition:** A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables $x, y$ of $\varphi$ the following is satisfied:

1. $\text{atoms}(x) \subseteq \text{atoms}(y)$ or $\text{atoms}(y) \subseteq \text{atoms}(x)$ or $\text{atoms}(x) \cap \text{atoms}(y) = \emptyset$, and
2. if $\text{atoms}(x) \subsetneq \text{atoms}(y)$ and $x \in \text{free}(\varphi)$, then $y \in \text{free}(\varphi)$.

**Queries that are not q-hierarchical:**

$$\psi_{S-E-T}() := \exists x \exists y \left( S(x) \land E(x, y) \land T(y) \right)$$
$$\varphi_{S-E-T}(x, y) := S(x) \land E(x, y) \land T(y)$$
q-hierarchical CQs

Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

q-hierarchical CQs are hierarchical CQs where, additionally, the quantifiers respect the query’s hierarchical form.

**Definition:** A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables $x, y$ of $\varphi$ the following is satisfied:

(i) $\text{atoms}(x) \subseteq \text{atoms}(y)$ or $\text{atoms}(y) \subseteq \text{atoms}(x)$ or $\text{atoms}(x) \cap \text{atoms}(y) = \emptyset$, and

(ii) if $\text{atoms}(x) \not\subset \text{atoms}(y)$ and $x \in \text{free}(\varphi)$, then $y \in \text{free}(\varphi)$.

**Queries that are not q-hierarchical:**

\[
\psi_{S-E-T}() := \exists x \exists y \left( S(x) \land E(x, y) \land T(y) \right)
\]

\[
\varphi_{S-E-T}(x, y) := S(x) \land E(x, y) \land T(y)
\]

\[
\varphi_{E-T}(x) := \exists y \left( E(x, y) \land T(y) \right)
\]
q-hierarchical CQs

Dalvi & Suciu (PODS’07) introduced the hierarchical CQs to characterise the Boolean CQs that can be answered in PTIME on probabilistic dbs.

q-hierarchical CQs are hierarchical CQs where, additionally, the quantifiers respect the query’s hierarchical form.

**Definition:** A CQ \( \varphi(z_1, \ldots, z_k) \) is q-hierarchical if for all variables \( x, y \) of \( \varphi \) the following is satisfied:

(i) \( \text{atoms}(x) \subseteq \text{atoms}(y) \) or \( \text{atoms}(y) \subseteq \text{atoms}(x) \) or \( \text{atoms}(x) \cap \text{atoms}(y) = \emptyset \), and

(ii) if \( \text{atoms}(x) \not\subseteq \text{atoms}(y) \) and \( x \in \text{free}(\varphi) \), then \( y \in \text{free}(\varphi) \).

**Queries that are not q-hierarchical:**

\[
\psi_{S-E-T}() := \exists x \exists y ( S(x) \land E(x, y) \land T(y) )
\]

\[
\varphi_{S-E-T}(x, y) := S(x) \land E(x, y) \land T(y)
\]

\[
\varphi_{E-T}(x) := \exists y ( E(x, y) \land T(y) )
\]

**A q-hierarchical query:**

\[
\theta_{E-T}(y) := \exists x ( E(x, y) \land T(y) )
\]
**Scenario**

- **Input:**
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ CQ

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
  
  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

**Main result:** This is possible $\iff$ $\varphi$ is $q$-hierarchical.
Intractability result for enumerating CQs that are not q-hierarchical

... is subject to suitable algorithmic conjecture
Intractability result for enumerating CQs that are not q-hierarchical
...is subject to suitable algorithmic conjecture

**The OMv-problem:** [Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and
a stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$
Intractability result for enumerating CQs that are not q-hierarchical
...is subject to suitable algorithmic conjecture

**The OMv-problem:** [Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and
a stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$

**OMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves
the OMv-problem in total time $O(n^{3-\epsilon})$
Intractability result for enumerating CQs that are not q-hierarchical

... is subject to suitable algorithmic conjecture

**The OMv-problem:** [Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and
a stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$

**OMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OMv-problem in total time $O(n^{3-\epsilon})$

Fails for offline algorithms if we receive all vectors at once: fast matrix multiplication!
Intractability result for enumerating CQs that are not q-hierarchical... is subject to suitable algorithmic conjecture

**The OMv-problem:** [Henzinger et al., STOC’15]

Input: a Boolean $n \times n$-matrix $M$ and
a stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors

Task: output $Mv_\ell$ before accessing $v_{\ell+1}$

OMv-Conjecture: For every $\epsilon > 0$, there is no algorithm that solves
the OMv-problem in total time $O(n^{3-\epsilon})$

Fails for offline algorithms if we receive all vectors at once: fast matrix multiplication!

**Theorem (Enumeration):** [Berkholz, Keppeler, S., PODS’17]

Let $\epsilon > 0$ and let $\varphi(x)$ be a self-join free CQ that is
not q-hierarchical.

Then, there is no algorithm with arbitrary preprocessing time and
$t_u = O(n^{1-\epsilon})$ update time that enumerates $\varphi(D)$ with
t$d = O(n^{1-\epsilon})$ delay, unless the OMv-conjecture fails.
Intractability result for enumerating CQs that are not q-hierarchical

...is subject to suitable algorithmic conjecture

**The OMv-problem:** [Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and a stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$

**OMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OMv-problem in total time $O(n^{3-\epsilon})$.

Fails for offline algorithms if we receive all vectors at once: fast matrix multiplication!

**Theorem (Enumeration):** [Berkholz, Keppeler, S., PODS’17]

Let $\epsilon > 0$ and let $\varphi(\overline{x})$ be a self-join free CQ that is not q-hierarchical.

Then, there is no algorithm with arbitrary preprocessing time and $t_u = O(n^{1-\epsilon})$ update time that enumerates $\varphi(D)$ with $t_d = O(n^{1-\epsilon})$ delay, unless the OMv-conjecture fails.

Proof idea for $\varphi_{E-T}(x) := \exists y \left( E(x, y) \land T(y) \right)$
Proof idea for $\varphi_{E-T}(x) := \exists y \ (E(x, y) \land T(y))$

A lower bound for enumerating via OMv

**Input:** Boolean $n \times n$ matrix $M$ and stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors.

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$
Proof idea for $\varphi_{E-T}(x) := \exists y \ (E(x, y) \land T(y))$

A lower bound for enumerating via OMv

**Input:** Boolean $n \times n$ matrix $M$ and stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors.

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$

Given $n \times n$ matrix $M$, let

$E^{D_0} := \{(i, j) \in [n]^2 : M(i, j) = 1\}, \quad T^{D_0} := \emptyset$

Create data structure for $D_0$ in time $n^2 \cdot n^{1-\epsilon}$. 
Proof idea for $\varphi_{E-T}(x) := \exists y \left( E(x, y) \land T(y) \right)$

A lower bound for enumerating via OMv

**Input:** Boolean $n \times n$ matrix $M$ and stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors.

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$

Given $n \times n$ matrix $M$, let

$$E^{D_0} := \{ (i, j) \in [n]^2 : M(i, j) = 1 \}, \quad T^{D_0} := \emptyset$$

Create data structure for $D_0$ in time $n^2 \cdot n^{1-\epsilon}$.

Given $n$-dim vector $v_\ell$, update

$$T^{D_\ell} := \{ i \in [n] : v_\ell(i) = 1 \}.$$

in time $n \cdot n^{1-\epsilon}$. 

Proof idea for \( \varphi_{E \cdot T}(x) := \exists y \left( E(x, y) \land T(y) \right) \)

A lower bound for enumerating via OMv

**Input:** Boolean \( n \times n \) matrix \( M \) and stream \( v_1, \ldots, v_n \) of \( n \)-dimensional Boolean vectors.

**Task:** output \( Mv_\ell \) before accessing \( v_{\ell+1} \)

Given \( n \times n \) matrix \( M \), let

\[ E^{D_0} := \{ (i,j) \in [n]^2 : M(i,j) = 1 \}, \quad T^{D_0} := \emptyset \]

Create data structure for \( D_0 \) in time \( n^2 \cdot n^{1-\epsilon} \).

Given \( n \)-dim vector \( v_\ell \), update

\[ T^{D_\ell} := \{ i \in [n] : v_\ell(i) = 1 \}. \]

in time \( n \cdot n^{1-\epsilon} \). For \( u_\ell := Mv_\ell \) we have:

\[ \varphi_{E \cdot T}(D_\ell) = \{ i \in [n] : u_\ell(i) = 1 \} \]
Proof idea for $\varphi_{E-T}(x) := \exists y \ (E(x, y) \land T(y))$

A lower bound for enumerating via OMv

**Input:** Boolean $n \times n$ matrix $M$ and stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors.

**Task:** output $Mv_\ell$ before accessing $v_{\ell+1}$

Given $n \times n$ matrix $M$, let

- $E^{D_0} := \{(i, j) \in [n]^2 : M(i, j) = 1\}$,
- $T^{D_0} := \emptyset$

Create data structure for $D_0$ in time $n^2 \cdot n^{1-\epsilon}$.

Given $n$-dim vector $v_\ell$, update

- $T^{D_\ell} := \{ i \in [n] : v_\ell(i) = 1\}$.

in time $n \cdot n^{1-\epsilon}$. For $u_\ell := Mv_\ell$ we have:

- $\varphi_{E-T}(D_\ell) = \{ i \in [n] : u_\ell(i) = 1\}$

and can output $u_\ell$ after enumerating $\varphi_{E-T}(D_\ell)$ in time $n \cdot n^{1-\epsilon}$. 
Proof idea for $\varphi_{E-T}(x) := \exists y \left( E(x, y) \land T(y) \right)$

A lower bound for enumerating via OMv

**Input:** Boolean $n \times n$ matrix $M$ and stream $v_1, \ldots, v_n$ of $n$-dimensional Boolean vectors.

**Task:** output $Mv_{\ell}$ before accessing $v_{\ell+1}$

Given $n \times n$ matrix $M$, let

$E^{D_0} := \{ (i, j) \in [n]^2 : M(i, j) = 1 \}$, $T^{D_0} := \emptyset$

Create data structure for $D_0$ in time $n^2 \cdot n^{1-\epsilon}$.

Given $n$-dim vector $v_{\ell}$, update

$T^{D_\ell} := \{ i \in [n] : v_{\ell}(i) = 1 \}$

in time $n \cdot n^{1-\epsilon}$. For $u_{\ell} := Mv_{\ell}$ we have:

$\varphi_{E-T}(D_\ell) = \{ i \in [n] : u_{\ell}(i) = 1 \}$

and can output $u_{\ell}$ after enumerating $\varphi_{E-T}(D_\ell)$ in time $n \cdot n^{1-\epsilon}$.

This solves OMv in total time $O(n^{3-\epsilon})$. 
Intractability result for **Boolean** CQs that are not q-hierarchical

**The OuMv-problem:** [Henzinger et al., STOC’15]

Input: a Boolean $n \times n$-matrix $M$ and
a stream $u_1, v_1, \ldots, u_n, v_n$ of $n$-dimensional Boolean vectors

Task: output $(u_\ell)^T M v_\ell$ before accessing $u_{\ell+1}$, $v_{\ell+1}$

**OuMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OuMv-problem in total time $O(n^{3-\epsilon})$
Intractability result for Boolean CQs that are not q-hierarchical

**The OuMv-problem:** [Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and a stream $u_1, v_1, \ldots, u_n, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $(u_\ell)^T M v_\ell$ before accessing $u_{\ell+1}, v_{\ell+1}$

**OuMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OuMv-problem in total time $O(n^{3-\epsilon})$

Fails for offline algorithms if we receive all vectors at once: fast matrix multiplication!
Intractability result for Boolean CQs that are not q-hierarchical

**The OuMv-problem:** [Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and

- a stream $u_1, v_1, \ldots, u_n, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $(u_\ell)^T M v_\ell$ before accessing $u_{\ell+1}, v_{\ell+1}$

**OuMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OuMv-problem in total time $O(n^{3-\epsilon})$

Fails for offline algorithms if we receive all vectors at once: fast matrix multiplication!

**Theorem (Boolean):** [Berkholz, Keppeler, S., PODS’17]

Fix an $\epsilon > 0$ and let $\varphi$ be a Boolean CQ whose homomorphic core is not q-hierarchically.

Then, there is no algorithm with arbitrary preprocessing time and $t_u = O(n^{1-\epsilon})$ update time that answers $\varphi(D)$ in time $t_a = O(n^{2-\epsilon})$, unless the OuMv-conjecture fails.
Intractability result for **Boolean** CQs that are not q-hierarchical

**The OuMv-problem:**  
[Henzinger et al., STOC’15]

**Input:** a Boolean $n \times n$-matrix $M$ and a stream $u_1, v_1, \ldots, u_n, v_n$ of $n$-dimensional Boolean vectors

**Task:** output $(u_\ell)^T M v_\ell$ before accessing $u_{\ell+1}, v_{\ell+1}$

**OuMv-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OuMv-problem in total time $O(n^{3-\epsilon})$

Fails for offline algorithms if we receive all vectors at once: fast matrix multiplication!

**Theorem (Boolean):**  
[Berkholz, Keppeler, S., PODS’17]

Fix an $\epsilon > 0$ and let $\varphi$ be a Boolean CQ whose homomorphic core is not q-hierarchical.

Then, there is no algorithm with arbitrary preprocessing time and $t_u = O(n^{1-\epsilon})$ update time that answers $\varphi(D)$ in time $t_a = O(n^{2-\epsilon})$, unless the OuMv-conjecture fails.

Proof idea for $\psi_{S-E-T} := \exists x \exists y \left( S(x) \land E(x, y) \land T(y) \right)$
Intractability result for counting CQs that are not q-hierarchical

The OV-problem: [cf. R. Williams, 2005]

Input: two sets $U$ and $V$ of $n$ Boolean vectors of dimension $d := \lceil \log^2 n \rceil$

Task: decide if there exist $u \in U$ and $v \in V$ with $u^\top v = 0$

OV-Conjecture: For every $\epsilon > 0$, there is no algorithm that solves the OV-problem in time $O(n^{2-\epsilon})$.
Intractability result for counting CQs that are not q-hierarchical

**The OV-problem:** [cf. R. Williams, 2005]

**Input:** two sets $U$ and $V$ of $n$ Boolean vectors of dimension $d := \lceil \log^2 n \rceil$

**Task:** decide if there exist $u \in U$ and $v \in V$ with $u^\top v = 0$

**OV-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OV-problem in time $O(n^{2-\epsilon})$

**Theorem (Counting):** [Berkholz, Keppeler, S., PODS’17]

Let $\epsilon > 0$ and let $\varphi(\vec{x})$ be a CQ whose homomorphic core is not q-hierarchical.

Then, there is no algorithm with arbitrary preprocessing time and $t_u = O(n^{1-\epsilon})$ update time that computes $|\varphi(D)|$ in time $t_c = O(n^{1-\epsilon})$, unless the OV-conjecture or the OuMv-conjecture fails.
Intractability result for counting CQs that are not q-hierarchical

**The OV-problem:** [cf. R. Williams, 2005]

**Input:** two sets $U$ and $V$ of $n$ Boolean vectors of dimension $d := \lceil \log^2 n \rceil$

**Task:** decide if there exist $u \in U$ and $v \in V$ with $u^\top v = 0$

**OV-Conjecture:** For every $\epsilon > 0$, there is no algorithm that solves the OV-problem in time $O(n^{2-\epsilon})$

**Theorem (Counting):** [Berkholz, Keppeler, S., PODS’17]

Let $\epsilon > 0$ and let $\varphi(\overline{x})$ be a CQ whose homomorphic core is not q-hierarchical.

Then, there is no algorithm with arbitrary preprocessing time and $t_u = O(n^{1-\epsilon})$ update time that computes $|\varphi(D)|$ in time $t_c = O(n^{1-\epsilon})$, unless the OV-conjecture or the OuMv-conjecture fails.

Proof idea for $\varphi_{E-T}(x) := \exists y \ (E(x, y) \land T(y))$
Proof idea for $\varphi_{E \cdot T}(x) := \exists y \ (E(x, y) \land T(y))$

A lower bound for counting via OV

Left: $n$ vertices for the $n$ vectors $u \in U$
Right: $d := \lceil \log^2 n \rceil$ vertices for vector-coordinates

$u_1 = (1, 0, 0)^T$
$u_2 = (1, 1, 0)^T$
$u_3 = (1, 0, 1)^T$
$u_4 = (0, 0, 1)^T$
$u_5 = (0, 1, 1)^T$
Proof idea for $\varphi_{E-T}(x) := \exists y \left( E(x, y) \land T(y) \right)$

A lower bound for counting via OV

Left: $n$ vertices for the $n$ vectors $u \in U$
Right: $d := \lceil \log^2 n \rceil$ vertices for vector-coordinates

$u_1 = (1, 0, 0)^T$
$u_2 = (1, 1, 0)^T$
$u_3 = (1, 0, 1)^T$
$u_4 = (0, 0, 1)^T$
$u_5 = (0, 1, 1)^T$

for each $v_\ell \in V$: update $T^{D_\ell}$ in time $d \cdot n^{1-\epsilon} = \lceil \log^2 n \rceil n^{1-\epsilon}$
Proof idea for \( \varphi_{E-T}(x) := \exists y \ (E(x, y) \land T(y)) \)

A lower bound for counting via OV

**Left:** \( n \) vertices for the \( n \) vectors \( u \in U \)

**Right:** \( d := \lceil \log^2 n \rceil \) vertices for vector-coordinates

\[
\begin{align*}
u_1 &= (1, 0, 0)^T \\
u_2 &= (1, 1, 0)^T \\
u_3 &= (1, 0, 1)^T \\
u_4 &= (0, 0, 1)^T \\
u_5 &= (0, 1, 1)^T
\end{align*}
\]

\[
T^{D_\ell} \quad v_\ell = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}
\]

\[
|\varphi_{E-T}(D_\ell)| = 4 = |\{ \ u_i : \ u_i^T v_\ell \neq 0 \}| \]

- for each \( v_\ell \in V \): update \( T^{D_\ell} \) in time \( d \cdot n^{1-\epsilon} = \lceil \log^2 n \rceil n^{1-\epsilon} \)
- there is \( u_i \in U \) with \( u_i^T v_\ell = 0 \) \( \iff \ |\varphi_{E-T}(D_\ell)| < n \).
Proof idea for $\varphi_{E-T}(x) := \exists y \ (E(x, y) \land T(y))$

A lower bound for counting via OV

Left: $n$ vertices for the $n$ vectors $u \in U$
Right: $d := \lceil \log^2 n \rceil$ vertices for vector-coordinates

$u_1 = (1, 0, 0)^T$
$u_2 = (1, 1, 0)^T$
$u_3 = (1, 0, 1)^T$
$u_4 = (0, 0, 1)^T$
$u_5 = (0, 1, 1)^T$

$\varphi_{E-T}(D_\ell) = 4 = |\{ u_i : u_i^T v_\ell \neq 0 \}|$

$T^{D_\ell} \quad v_\ell = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$

▶ for each $v_\ell \in V$: update $T^{D_\ell}$ in time $d \cdot n^{1-\varepsilon} = \lceil \log^2 n \rceil n^{1-\varepsilon}$
▶ there is $u_i \in U$ with $u_i^T v_\ell = 0 \iff |\varphi_{E-T}(D_\ell)| < n.$
▶ finished for all $v_\ell \in V$ within time $n \cdot \lceil \log^2 n \rceil n^{1-\varepsilon} = n^{2-\varepsilon'}$
Scenario

- **Input:**
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ CQ

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
Scenario

- **Input:**
  - Database $D$ (arbitrary)
  - query $\varphi(x_1, \ldots, x_k)$ (q-hierarchical CQ)

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
  After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
Scenario

▷ **Input:**
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ q-hierarchical CQ

▷ **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

▷ **Output:**
  For Boolean queries:
  - Decide if $D \models \varphi$
  For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$
  - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

▷ **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$

  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
Scenario

- **Input:**
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ q-hierarchical CQ

- **Preprocessing:**
  - Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  - Tuples may be inserted into or deleted from $D$
  - After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
Scenario

▶ Input:
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ q-hierarchical CQ

▶ Preprocessing:
  Build a suitable data structure that represents $D$ and $\varphi(D)$

▶ Output:
  For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\overline{a}$ whether $\overline{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

▶ Dynamic setting:
  Tuples may be inserted into or deleted from $D$

  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

Main result: This is possible $\iff$ $\varphi$ is q-hierarchical.
Scenario

- **Input:**
  - Database $D$, arbitrary
  - Query $\varphi(x_1, \ldots, x_k)$, q-hierarchical CQ

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $a$ whether $a \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$

  After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
Scenario

Input:
- Database $D$ arbitrary
- query $\varphi(x_1, \ldots, x_k)$ q-hierarchical CQ

Preprocessing:
Build a suitable data structure that represents $D$ and $\varphi(D)$

Output:
For Boolean queries:
- Decide if $D \models \varphi$ in constant time
For $k$-ary queries:
- Compute the number of tuples in $\varphi(D)$ in constant time
- Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
- Enumerate the tuples in $\varphi(D)$ with constant delay

Dynamic setting:
Tuples may be inserted into or deleted from $D$

After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($\|D\|$).

Main result: This is possible $\iff$ $\varphi$ is q-hierarchical.
Scenario

▶ Input:
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ q-hierarchical CQ

▶ Preprocessing:
  Build a suitable data structure that represents $D$ and $\varphi(D)$ in time $O(||D||)$

▶ Output:
  For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

▶ Dynamic setting:
  Update data structure in constant time
  Tuples may be inserted into or deleted from $D$

After every update we want to update the data structure and report the new query result quickly: in time constant or polylog($||D||$).

Main result: This is possible $\iff \varphi$ is q-hierarchical.
Scenario

▶ Input:
- Database $D$ arbitrary
- query $\varphi(x_1, \ldots, x_k)$ $\mathsf{q}$-hierarchical CQ

▶ Preprocessing:
Build a suitable data structure that represents $D$ and $\varphi(D)$

▶ Output:
For Boolean queries:
- Decide if $D \models \varphi$ in constant time
For $k$-ary queries:
- Compute the number of tuples in $\varphi(D)$ in constant time
- Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
- Enumerate the tuples in $\varphi(D)$ with constant delay

▶ Dynamic setting:
update data structure in constant time
Tuples may be inserted into or deleted from $D$

After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.

Main result: This is possible $\iff$ $\varphi$ is $\mathsf{q}$-hierarchical.
Scenario

- **Input:**
  - Database $D$
  - Query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  - Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  - Update data structure in time $\text{poly}(\varphi)$

  Tuples may be inserted into or deleted from $D$

  After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.

[Refs: Berkholz, Keppeler, S., PODS'17]
-whole page-Main result: This is possible $\iff \varphi$ is q-hierarchical.
Scenario

- **Input:**
  - Database $D$ arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ

- **Preprocessing:**
  - Build a suitable data structure that represents $D$ and $\varphi(D)$ in time $\text{poly}(\varphi) \cdot \|D\|$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\overline{a}$ whether $\overline{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  - Update data structure in time $\text{poly}(\varphi)$
  - Tuples may be inserted into or deleted from $D$

After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$. 

**Main result:** This is possible $\iff$ $\varphi$ is q-hierarchical.
Scenario

- **Input:**
  - Database $D$
  - Query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  - Build a suitable data structure that represents $D$ and $\varphi(D)$ in time $\text{poly}(\varphi) \| D\|

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $\text{poly}(\varphi)$
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  - Update data structure in time $\text{poly}(\varphi)$
  - Tuples may be inserted into or deleted from $D$

  After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.

Main result: This is possible $\iff \varphi$ is $q$-hierarchical.
Scenario

- **Input:**
  - Database $D$ arbitrary 
  - query $\varphi(x_1, \ldots, x_k)$ q-hierarchical CQ

- **Preprocessing:** in time $\text{poly}(\varphi)\|D\|
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $a$ whether $a \in \varphi(D)$ in time $\text{poly}(\varphi)$
    - Enumerate the tuples in $\varphi(D)$ with delay $\text{poly}(\varphi)$

- **Dynamic setting:** update data structure in time $\text{poly}(\varphi)$
  Tuples may be inserted into or deleted from $D$

After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.

**Main result:** This is possible $\iff \varphi$ is q-hierarchical.
Efficient evaluation of a fragment of CQs

**Theorem (Upper bound):**

For every CQ that is q-hierarchical, there is a dynamic data structure that has **constant update time** and allows to

- answer a Boolean CQ,
- count the number of result tuples,
- enumerate the result relation with constant delay.
Efficient evaluation of a fragment of CQs

**Theorem (Upper bound):**
For every CQ that is q-hierarchical, there is a dynamic data structure that has constant update time and allows to

- answer a Boolean CQ,
- count the number of result tuples,
- enumerate the result relation with constant delay.
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]
q-hierarchical queries

\[ \varphi(x, y, z) \coloneqq R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)| \]
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N_+^E(v)| \cdot |N_+^F(v)| \]

- **COUNT:** store \( |N_+^E(v)|, |N_+^F(v)|, \sum_{v \in R^D} |N_+^E(v)| \cdot |N_+^F(v)| \)
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)| \]

- **COUNT:** store \( |N_E^+(v)|, |N_F^+(v)|, \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)| \)
- **ENUM:** store \( N_E^+(v), N_F^+(v) \) as lists with constant access, for \( v \in R^D \) report \( \{v\} \times N_E^+(v) \times N_F^+(v) \)
q-hierarchical queries

$$\varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z)$$

$$|\varphi(D)| = \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)|$$

- **COUNT:** store $|N^+_E(v)|$, $|N^+_F(v)|$, $\sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)|$

- **ENUM:** store $N^+_E(v)$, $N^+_F(v)$ as lists with constant access, for $v \in R^D$ report $\{v\} \times N^+_E(v) \times N^+_F(v)$
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)| \]

- **COUNT**: store \( |N^+_E(v)|, |N^+_F(v)|, \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)| \)
- **ENUM**: store \( N^+_E(v), N^+_F(v) \) as lists with constant access, for \( v \in R^D \) report \( \{v\} \times N^+_E(v) \times N^+_F(v) \)

**Definition (q-tree):**

A q-tree \( T \) for a CQ \( \varphi(x_1, \ldots, x_\ell) \) is a rooted tree with \( V(T) = \text{vars}(\varphi) \) and
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)| \]

- **COUNT**: store \( |N^+_E(v)|, |N^+_F(v)|, \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)| \)

- **ENUM**: store \( N^+_E(v), N^+_F(v) \) as lists with constant access, for \( v \in R^D \) report \( \{v\} \times N^+_E(v) \times N^+_F(v) \)

**Definition (q-tree):**

A q-tree \( T \) for a CQ \( \varphi(x_1, \ldots, x_\ell) \) is a rooted tree with \( V(T) = \text{vars}(\varphi) \) and

1. for every \( R(y_1, \ldots, y_r) \) in \( \varphi \): \( \{y_1, \ldots, y_r\} \) forms a path in \( T \) that starts at the root
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)| \]

- **COUNT:** store \(|N_E^+(v)|, |N_F^+(v)|, \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|\)
- **ENUM:** store \(N_E^+(v), N_F^+(v)\) as lists with constant access, for \(v \in R^D\) report \(\{v\} \times N_E^+(v) \times N_F^+(v)\)

**Definition (q-tree):**
A q-tree \(T\) for a CQ \(\varphi(x_1, \ldots, x_\ell)\) is a rooted tree with \(V(T) = \text{vars}(\varphi)\) and

1. for every \(R(y_1, \ldots, y_r)\) in \(\varphi\): \(\{y_1, \ldots, y_r\}\) forms a path in \(T\) that starts at the root
2. the free variables \(\{x_1, \ldots, x_\ell\}\) form a connected subtree that contains the root
q-hierarchical queries

\[ \varphi(x, y, z) := R(x) \land E(x, y) \land F(x, z) \]

\[ |\varphi(D)| = \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)| \]

- **COUNT**: store \( |N^+_E(v)|, |N^+_F(v)|, \sum_{v \in R^D} |N^+_E(v)| \cdot |N^+_F(v)| \)

- **ENUM**: store \( N^+_E(v), N^+_F(v) \) as lists with constant access, for \( v \in R^D \) report \( \{v\} \times N^+_E(v) \times N^+_F(v) \)

**Definition (q-tree):**

A q-tree \( T \) for a CQ \( \varphi(x_1, \ldots, x_\ell) \) is a rooted tree with \( V(T) = \text{vars}(\varphi) \) and

1. for every \( R(y_1, \ldots, y_r) \) in \( \varphi \): \( \{y_1, \ldots, y_r\} \) forms a path in \( T \) that starts at the root

2. the free variables \( \{x_1, \ldots, x_\ell\} \) form a connected subtree that contains the root

**Lemma**: A CQ \( \varphi(\overline{x}) \) is q-hierarchical \( \iff \) every connected component of \( \varphi(\overline{x}) \) has a q-tree.
Data structure for q-hierarchical queries

\[ \varphi(x, y, z, y', z') = (R_{xyz} \land R_{xyz'} \land E_{xy} \land E_{xy'} \land S_{xyz}) \]
Data structure for q-hierarchical queries

\[ \varphi(x, y, z, y', z') = (R_{xyz} \land R_{xyz'} \land E_{xy} \land E_{xy'} \land S_{xyz}) \]

- \( S(b, p, a), R(b, p, a), R(b, p, b), R(b, p, c) \in D, \ E(b, p) \notin D \)
Data structure for q-hierarchical queries

$$\varphi(x, y, z, y', z') = (R_{xyz} \land R_{xyz'} \land E_{xy} \land E_{xy'} \land S_{xyz})$$

$\varphi(x, y, z, y', z') =$

$\begin{array}{c}
\begin{array}{c}
\text{start} \\
c'_{\text{start}} = 38
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
x \\
a \quad 14
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
y \\
y' \\
a \quad 6
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
z \\
z' \\
a \quad 1
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
e \quad 3
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
f \quad 3
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
g \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
h \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
p \quad 3
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
d \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
a \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
b \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
c \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
a \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
b \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
c \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
a \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
b \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
c \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
a \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
b \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
c \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
a \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
b \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
c \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
a \quad 1
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
S(b, p, a), R(b, p, a), R(b, p, b), R(b, p, c) \in D, E(b, p) \notin D
\end{array}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\text{INSERT } E(b, p)
\end{array}
\end{array}$
Summary

Input:
- Database $D$ arbitrary
- query $\varphi(x_1, \ldots, x_k)$ $\mathbf{q}$-hierarchical CQ

Preprocessing:
Build a suitable data structure that represents $D$ and $\varphi(D)$ in time $\text{poly}(\varphi) \| D \|

Output:
- For Boolean queries:
  - Decide if $D \models \varphi$ in time $O(1)$
- For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
  - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $\text{poly}(\varphi)$
  - Enumerate the tuples in $\varphi(D)$ with delay $\text{poly}(\varphi)$

Dynamic setting: update data structure in time $\text{poly}(\varphi)$
Tuples may be inserted into or deleted from $D$

After every update we want to update the data structure and report the new query result quickly: in time constant or $\text{polylog}(\|D\|)$.
Main result: This is possible $\iff$ $\varphi$ is q-hierarchical.
Summary

- **Input:**
  - Database \( D \) arbitrary
  - query \( \varphi(x_1, \ldots, x_k) \) \textbf{q-hierarchical} CQ

- **Preprocessing:**
  - Build a suitable data structure that represents \( D \) and \( \varphi(D) \)
  - in time \( \text{poly}(\varphi) \| D \| \)

- **Output:**
  - For Boolean queries:
    - Decide if \( D \models \varphi \)
      - in time \( O(1) \)
  - For \( k \)-ary queries:
    - Compute the number of tuples in \( \varphi(D) \)
      - in time \( O(1) \)
    - Test for a given tuple \( \bar{a} \) whether \( \bar{a} \in \varphi(D) \)
      - in time \( \text{poly}(\varphi) \)
    - Enumerate the tuples in \( \varphi(D) \)
      - with delay \( \text{poly}(\varphi) \)

- **Dynamic setting:**
  - update data structure in time \( \text{poly}(\varphi) \)
  - Tuples may be inserted into or deleted from \( D \)
  - After every update we want to update the data structure and report the new query result quickly: in time constant or \( \text{polylog}(\| D \|) \).

- **Main result:** This is possible \( \iff \varphi \) is \textbf{q-hierarchical}.

- **Ongoing work:** Similar results for UCQs & FDs.
Summary

- **Input:**
  - Database $D$
  - Query $\varphi(x_1, \ldots, x_k)$

- **Preprocessing:**
  - Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\overline{a}$ whether $\overline{a} \in \varphi(D)$ in time $\text{poly}(\varphi)$
    - Enumerate the tuples in $\varphi(D)$ with delay $\text{poly}(\varphi)$

- **Dynamic setting:**
  - Update data structure in time $\text{poly}(\varphi)$
  - Tuples may be inserted into or deleted from $D$

**Related work:** [Idris, Ugarte, Vansummeren, SIGMOD’17]: q-hierarchical queries are also efficient in practice!
Overview

Introduction

Conjunctive Queries on Arbitrary Databases

First-Order Queries on Bounded Degree Databases
FO+MOD queries and FOC(\(\mathbb{P}\)) queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

\(\exists y \mod 2 \ y \ Movie(y, "Sigourney Weaver")\)
FO+MOD queries and FOC(\(\mathbb{P}\)) queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

\[ \exists^{0 \mod 2} y \ Movie(y, "Sigourney Weaver") \]

\(\text{FO+MOD} = \) extension of first-order logic with modulo-counting quantifiers \(\exists^{i \mod m} y \ \psi(y, \bar{z})\)
FO+MOD queries and FOC(\(\mathbb{P}\)) queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

Is the number of movies with Sigourney Weaver even?

In FO+MOD:
\[
\exists^{0 \mod 2} y \; \text{Movie}(y, "Sigourney Weaver")
\]

In FOC(\(\mathbb{P}\)):
\[
P_{even}(\#(y).\text{Movie}(y, "Sigourney Weaver"))
\]

\(\text{FO+MOD} = \text{extension of first-order logic with modulo-counting quantifiers}\)

\[
\forall i \mod m y \; \psi(y, \overline{z})
\]
FO+MOD queries and FOC(\(P\)) queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

\[ \exists^{0 \mod 2} y \ Movie(y, "Sigourney Weaver") \]

In FOC(\(P\)):

\[ P_{even}(\#(y).Movie(y, "Sigourney Weaver")) \]

**FO+MOD** = extension of first-order logic with modulo-counting quantifiers

\[ \exists^{i \mod m} y \ \psi(y, \bar{z}) \]

Let \(P\) be a collection of numerical predicates. E.g., \(P\) may contain the predicates \([P_{even}] = \{i \in \mathbb{Z} : i \text{ is even}\}\) and \([P_{\leq}] = \{(i,j) \in \mathbb{Z}^2 : i \leq j\}\).
FO+MOD queries and FOC(\(P\)) queries

<table>
<thead>
<tr>
<th>Movie</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alien</td>
<td>Sigourney Weaver</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Blade Runner</td>
<td>Sean Young</td>
</tr>
<tr>
<td>Brazil</td>
<td>Jonathan Pryce</td>
</tr>
<tr>
<td>Brazil</td>
<td>Kim Greist</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Humphrey Bogart</td>
</tr>
<tr>
<td>Casablanca</td>
<td>Ingrid Bergmann</td>
</tr>
<tr>
<td>Gravity</td>
<td>Sandra Bullock</td>
</tr>
<tr>
<td>Gravity</td>
<td>George Clooney</td>
</tr>
<tr>
<td>Resident Evil</td>
<td>Milla Jovovich</td>
</tr>
<tr>
<td>Terminator</td>
<td>Arnold Schwarzenegger</td>
</tr>
<tr>
<td>Terminator</td>
<td>Linda Hamilton</td>
</tr>
<tr>
<td>Terminator</td>
<td>Michael Biehn</td>
</tr>
</tbody>
</table>

Is the number of movies with Sigourney Weaver even?

**In FO+MOD:**

\[ \exists^{0 \mod 2} \ y \ \text{Movie}(y, "Sigourney Weaver") \]

**In FOC(\(P\)):**

\[ P_{\text{even}}(\#(y).\text{Movie}(y, "Sigourney Weaver")) \]

***FO+MOD = extension of first-order logic with modulo-counting quantifiers*** \[ \exists^{i \mod m} y \ \psi(y, \bar{z}) \]

Let \(P\) be a collection of numerical predicates. E.g., \(P\) may contain the predicates \([P_{\text{even}}] = \{ i \in \mathbb{Z} : i \text{ is even} \}\) and \([P_{\leq}] = \{(i, j) \in \mathbb{Z}^2 : i \leq j\}\).

**FOC(\(P\)) = extension of first-order logic with formulas of the form**

\(P(t_1, \ldots, t_r)\) for \(P \in P\) of arity \(r\), and where each \(t_i\) is a **counting term** built using integers, +, \(\cdot\), and basic counting terms \(t(\bar{x})\) of the form \(\#\bar{y}.\psi(\bar{x}, \bar{y})\).
Bounded degree databases

Graph $G = (V, E)$:

degree of a node $v$ : the number of neighbours of $v$ in $G$
degree of $G$ : $\max \{\text{degree}(v) : v \in V\}$

Database $D$:

degree of $D$ : degree of the Gaifman graph of $D$

Gaifman graph of $D$:

the graph $G = (V, E)$ with $V = \text{adom}(D)$ and an edge between two distinct nodes $a, b \in V$ iff some tuple in some relation of $D$ contains $a$ and $b$
Known results for the static setting (i.e., without updates)

FO query evaluation on dbs of degree $\leq d$

Boolean queries:
  ▶ evaluation in linear time \hfill (Seese 1996)

Non-Boolean queries:
  ▶ enumeration with constant delay and linear-time preprocessing
  ▶ delay $f(\varphi, d) = 3$-exp$(|\varphi| + \lg \lg d)$ (Kazana, Segoufin 2011)
Known results for the static setting (i.e., without updates)

FO query evaluation on dbs of degree $\leq d$

Boolean queries:

- evaluation in linear time \( (\text{Seese 1996}) \)
- evaluation in time \( f(\varphi, d)\|D\|, \) for \( (\text{Frick, Grohe 2004}) \)

\[
f(\varphi, d) = 2^{d^{2O(\|\varphi\|)}} = 3\text{-exp}(\|\varphi\| + \lg \lg d)
\]

and the 3-fold exponential blow-up is unavoidable assuming \( \text{FPT} \neq \text{AW}[\ast] \).
Known results for the static setting (i.e., without updates)

FO query evaluation on dbs of degree $\leq d$

**Boolean queries:**

- evaluation in linear time (Seese 1996)
- evaluation in time $f(\varphi, d)\|D\|$, for (Frick, Grohe 2004)

\[
f(\varphi, d) = 2^{d^{2O(\|\varphi\|)}} = 3\text{-exp}(\|\varphi\| + \lg \lg d)
\]

and the 3-fold exponential blow-up is unavoidable assuming $\text{FPT} \neq \text{AW}[\ast]$.

**Non-Boolean queries:**

- enumeration with constant delay and linear-time preprocessing (Durand, Grandjean 2007)
Known results for the static setting (i.e., without updates)

FO query evaluation on dbs of degree $\leq d$

**Boolean queries:**

- evaluation in linear time \(\text{(Seese 1996)}\)
- evaluation in time \(f(\varphi, d)\|D\|\), for \(\text{(Frick, Grohe 2004)}\)

\[
f(\varphi, d) = 2^{d^{2^O(\|\varphi\|)}} = 3\text{-exp}(\|\varphi\| + \lg \lg d)
\]

and the 3-fold exponential blow-up is unavoidable assuming \(\text{FPT} \neq \text{AW[\ast]}\).

**Non-Boolean queries:**

- enumeration with constant delay and linear-time preprocessing \(\text{(Durand, Grandjean 2007)}\)
- delay \(f(\varphi, d)\) and preprocessing \(f(\varphi, d)\|D\|\), where \(f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)\) \(\text{(Kazana, Segoufin 2011)}\)
Known results for the static setting (i.e., without updates)

FO query evaluation on dbs of degree $\leq d$

**Boolean queries:**
- evaluation in linear time \( (\text{Seese 1996}) \)
- evaluation in time \( f(\varphi, d)\|D\| \), for \( (\text{Frick, Grohe 2004}) \)
  \[
  f(\varphi, d) = 2^{d^{2^{O(\|\varphi\|)}}} = 3\text{-exp}(\|\varphi\| + \lg \lg d)
  \]
  and the 3-fold exponential blow-up is unavoidable assuming \( \text{FPT} \neq \text{AW[*]} \).

**Non-Boolean queries:**
- enumeration with constant delay and linear-time preprocessing \( (\text{Durand, Grandjean 2007}) \)
- delay \( f(\varphi, d) \) and preprocessing \( f(\varphi, d)\|D\| \),
  where \( f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d) \) \( (\text{Kazana, Segoufin 2011}) \)

Similar results for other classes of databases
Known results for the static setting (i.e., without updates)

FO query evaluation on dbs of degree \( \leq d \)

**Boolean queries:**

- evaluation in linear time \((\text{Seese } 1996)\)
- evaluation in time \(f(\varphi, d)\|D\|\), for \((\text{Frick, Grohe } 2004)\)

\[
f(\varphi, d) = 2^{d^{2^O(\|\varphi\|)}} = 3\text{-exp}(\|\varphi\| + \lg \lg d)
\]

and the 3-fold exponential blow-up is unavoidable assuming \(\text{FPT} \neq \text{AW}[\ast]\).

**Non-Boolean queries:**

- enumeration with constant delay and linear-time preprocessing \((\text{Durand, Grandjean } 2007)\)
- delay \(f(\varphi, d)\) and preprocessing \(f(\varphi, d)\|D\|\), where \(f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)\) \((\text{Kazana, Segoufin } 2011)\)

**New:** Generalisation to the dynamic setting and \(\text{FO+MOD}\)

[Berkholz, Keppeler, S., ICDT’17] and \(\text{FOC(}\mathbb{P}\text{)}\) \([\text{Kuske, S., LICS’17}]\)
Scenario

[Scenario] [Berkholz, Keppeler, S., ICDT’17], [Kuske, S., LICS’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\text{FOC}(\mathbb{P})[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
Scenario

[Berkholz, Keppeler, S., ICDT'17], [Kuske, S., LICS’17]

▶ Input:
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in FOC($\mathbb{P})[\sigma]$

▶ Preprocessing:
  Build a suitable data structure that represents $D$ and $\varphi(D)$

▶ Output:
  For Boolean queries:
    - Decide if $D \models \varphi$
  For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

▶ Dynamic setting:
  Tuples may be inserted into or deleted from $D$
Scenario

[Berkholz, Keppeler, S., ICDT’17], [Kuske, S., LICS’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - Query $\varphi(x_1, \ldots, x_k)$ in $\FOC(\mathbb{P})[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$

- Data complexity

- Preprocessing:
  in time $O(\|D\|)$

- Output:
  - For Boolean queries:
    in constant time

- Dynamic setting:
  - Thank you!
Scenario

[Berkholz, Keppeler, S., ICDT’17], [Kuske, S., LICS’17]

▶ **Input:**
- Database $D$ of degree $\leq d$
- query $\varphi(x_1, \ldots, x_k)$ in FOC($\mathbb{P}$)[$\sigma$]

▶ **Preprocessing:**
Build a suitable data structure that represents $D$ and $\varphi(D)$

▶ **Output:**
- For Boolean queries:
  - Decide if $D \models \varphi$ in constant time
- For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$ in constant time
  - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$

▶ **Dynamic setting:**
Tuples may be inserted into or deleted from $D$

data complexity

in time $O(\|D\|)$

in constant time

in constant time

– Thank you! –
Scenario

[Berkholz, Keppeler, S., ICDT'17], [Kuske, S., LICS'17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\text{FOC}(\mathbb{P})[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\overline{a}$ whether $\overline{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
Scenario [Berkholz, Keppeler, S., ICDT'17], [Kuske, S., LICS'17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\text{FOC}(P)[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$ in time $O(\|D\|)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  Tuples may be inserted into or deleted from $D$
Scenario

[Berkholz, Keppeler, S., ICDT’17], [Kuske, S., LICS’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\text{FOC}(\mathbb{P})[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  Update data structure in constant time

Tuples may be inserted into or deleted from $D$
Scenario

[Berkholz, Keppeler, S., ICDT'17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+$\text{MOD}[\sigma]$

- **Preprocessing:**
  - Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in constant time
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  - Update data structure in constant time
  - Tuples may be inserted into or deleted from $D$

combined complexity $f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)$
in time $f(\varphi, d)\|D\|$
Scenario

[BERKHOlz, KEppeler, S., ICDT’17]

▶ Input:
- Database $D$ of degree $\leq d$
- query $\varphi(x_1, \ldots, x_k)$ in $\text{FO}+\text{MOD}[\sigma]$

▶ combined complexity

$\varphi$, $d$

$f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)$

▶ Preprocessing:
Build a suitable data structure that represents $D$ and $\varphi(D)$

▶ Output:
For Boolean queries:
- Decide if $D \models \varphi$ in constant time

For $k$-ary queries:
- Compute the number of tuples in $\varphi(D)$ in constant time
- Test for a given tuple $a$ whether $a \in \varphi(D)$ in constant time
- Enumerate the tuples in $\varphi(D)$ with constant delay

▶ Dynamic setting:
update data structure in time $f(\varphi, d)$

Tuples may be inserted into or deleted from $D$

Similar results for $\text{FO}$ with counting $\text{FOC}[\text{P}]$ (Kuske, S., LICS’17).

Future task:
Revisit other results on $\text{FO}$ model checking in the dynamic setting!

– Thank you! –

Nicole Schweikardt (HU Berlin) Database Theory and Query Answering under Updates
Scenario

[Note: The scenario is extracted from Berkholz, Keppeler, S., ICDT'17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - Query $\varphi(x_1, \ldots, x_k)$ in FO+$\text{MOD}[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in constant time
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  - Update data structure in time $f(\varphi, d)$
  - Tuples may be inserted into or deleted from $D$

---

Similar results for FO with counting FOC \cite{Kuske, LICS'17}.

Future task:
Revisit other results on FO model checking in the dynamic setting!
Scenario

[Berkholz, Keppeler, S., ICDT’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\text{FO} + \text{MOD}[\sigma]$

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in constant time
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  Update data structure in time $f(\varphi, d)$
  Tuples may be inserted into or deleted from $D$

- Thank you!
Scenario

[BERKHOLZ, KEPPELER, S., ICDT’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[$\sigma$]

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
    - Enumerate the tuples in $\varphi(D)$ with constant delay

- **Dynamic setting:**
  - Update data structure in time $f(\varphi, d)$
  - Tuples may be inserted into or deleted from $D$

**Combined complexity**

- $f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)$
- $f(\varphi, d)\|D\|$
Scenario

[Berkholz, Keppeler, S., ICDT'17]

Input:
- Database $D$ of degree $\leq d$
- Query $\varphi(x_1, \ldots, x_k)$ in FO$\pm$MOD$[\sigma]$

Preprocessing:
Build a suitable data structure that represents $D$ and $\varphi(D)$

Output:
- For Boolean queries:
  - Decide if $D \models \varphi$ in time $O(1)$
- For $k$-ary queries:
  - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
  - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
  - Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$

Dynamic setting:
- Update data structure in time $f(\varphi, d)$
  - Tuples may be inserted into or deleted from $D$

combined complexity
- $f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \log \log d)$
- in time $f(\varphi, d)\|D\|$

Future task:
Revisit other results on FO model checking in the dynamic setting!
Scenario

[Berkholz, Keppeler, S., ICDT’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[$\sigma$]
  
  \[ f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d) \]

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
    - Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$

- **Dynamic setting:**
  - Update data structure in time $f(\varphi, d)$
  - Tuples may be inserted into or deleted from $D$

**Proof method:** use Hanf normal form for FO+MOD
Hanf normal form for FO+MOD

- A type $\tau$ with $k$ centres and radius $r$:

Example type with $k = 4$ centres and radius $r = 1$
Hanf normal form for FO+MOD

- A type $\tau$ with $k$ centres and radius $r$:

  Example type with $k = 4$ centres and radius $r = 1$

- $N_r^D(b)$ is the induced substructure of $D$ on

  $$N_r^D(b) = N_r^D(b_1) \cup \cdots \cup N_r^D(b_k)$$

  where

  $$N_r^D(b_i) = \{ a \in \text{adom}(D) : \text{dist}^D(b_i, a) \leq r \}$$
Hanf normal form for FO+MOD

- A type $\tau$ with $k$ centres and radius $r$:

- $\mathcal{N}_r^D(b)$ is the induced substructure of $D$ on

  \[
  \mathcal{N}_r^D(b) = \mathcal{N}_r^D(b_1) \cup \cdots \cup \mathcal{N}_r^D(b_k)
  \]

  where

  \[
  \mathcal{N}_r^D(b_i) = \{a \in \text{adom}(D) : \text{dist}^D(b_i, a) \leq r\}
  \]

- Sphere-formula $\text{sph}_{\tau}(x)$:

  \[
  (D, \overline{a}) \models \text{sph}_{\tau}(\overline{x}) \iff (\mathcal{N}_r^D(\overline{a}), \overline{a}) \cong \tau
  \]
A Hanf normal form $\psi(\overline{x})$ is a Boolean combination of

- sphere-formulas $\text{sph}_\rho(\overline{x})$ and
- Hanf-sentences $\exists \geq m y \text{sph}_\tau(y)$ and $\exists^{i \mod m} y \text{sph}_\tau(y)$

where $\tau$ is a type with 1 centre and radius $r$. 
Hanf normal form for FO+MOD

A Hanf normal form $\psi(\overline{x})$ is a Boolean combination of

- sphere-formulas $\text{sph}_\rho(\overline{x})$ and
- Hanf-sentences $\exists \geq m y \text{sph}_\tau(y)$ and $\exists^{i \mod m} y \text{sph}_\tau(y)$

where $\tau$ is a type with 1 centre and radius $r$.

Two queries $\varphi(\overline{x})$ and $\psi(\overline{x})$ are $d$-equivalent iff

$$(D, \overline{a}) \models \varphi \iff (D, \overline{a}) \models \psi$$

for all dbs $D$ of degree $\leq d$. 

Theorem (Heimberg, Kuske, S., LICS'16)

There is an algorithm which receives as input a degree bound $d \geq 2$ and a FO+MOD $\sigma$-formula $\varphi(\overline{x})$, and constructs a $d$-equivalent formula $\psi(\overline{x})$ in Hanf normal form.

The algorithm's runtime is $f(\varphi, d) = 3\exp(|\varphi| + \log \log d)$. 

Nicole Schweikardt (HU Berlin) Database Theory and Query Answering under Updates 28/43
Hanf normal form for FO+MOD

A Hanf normal form $\psi(\bar{x})$ is a Boolean combination of

- sphere-formulas $\text{sph}_\rho(\bar{x})$ and
- Hanf-sentences $\exists^\geq m y \text{sph}_\tau(y)$ and $\exists^{\text{mod}} m y \text{sph}_\tau(y)$

where $\tau$ is a type with 1 centre and radius $r$.

Two queries $\varphi(\bar{x})$ and $\psi(\bar{x})$ are $d$-equivalent iff

$$(D, \bar{a}) \models \varphi \iff (D, \bar{a}) \models \psi$$

for all dbs $D$ of degree $\leq d$.

Theorem (Heimberg, Kuske, S., LICS’16)

There is an algorithm which receives as input a degree bound $d \geq 2$ and a FO+MOD[$\sigma$]-formula $\varphi(\bar{x})$, and constructs a $d$-equivalent formula $\psi(\bar{x})$ in Hanf normal form.

The algorithm’s runtime is $f(\varphi, d) = 3 \text{-exp}(|\varphi| + \lg \lg d)$. 

Main result for Boolean queries

**Theorem**

*There is a dynamic algorithm that receives as input*

- a degree bound $d \geq 2$,
- a Boolean $\text{FO} + \text{MOD}[\sigma]$-query $\varphi$, and
- a db $D$ of degree $\leq d$,

*and computes*

- *within $f(\varphi, d) \cdot \|D\|$ preprocessing time* a data structure
- *that can be updated in time $f(\varphi, d)$*

*and allows to return the query result $\varphi(D)$ with answer time $O(1)$.*

\[
f(\varphi, d) = 3\cdot \exp(\|\varphi\| + \lg \lg d)
\]
Main result for Boolean queries

Theorem

There is a dynamic algorithm that receives as input

- a degree bound \( d \geq 2 \),
- a Boolean \( \text{FO} + \text{MOD}[\sigma] \)-query \( \varphi \), and
- a db \( D \) of degree \( \leq d \),

and computes

- within \( f(\varphi, d) \| D \| \) preprocessing time a data structure
- that can be updated in time \( f(\varphi, d) \)

and allows to return the query result \( \varphi(D) \) with answer time \( O(1) \).

\[
f(\varphi, d) = 3\text{-exp}(\| \varphi \| + \lg \lg d)
\]

Proof Idea: Step 1: Transform \( \varphi \) into Hanf normal form \( \psi \).
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \text{sph}_\tau(y) \land \exists^{0 \mod 2} y \text{sph}_\rho(y) \]
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \ \text{sph}_\tau(y) \land \exists^{0 \mod 2} y \ \text{sph}_\rho(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

\[
\begin{array}{c}
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array}
\]

Let \( \rho \) be the type with 1 center and radius 2:

\[
\begin{array}{c}
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\text{\textbullet} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\textbullet} \\
\end{array}
\]
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \ sph_\tau(y) \land \exists^{0 \mod 2} y \ sph_\rho(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

Let \( \rho \) be the type with 1 center and radius 2:

Data structure: \( A[\tau] = 0 \), \( A[\rho] = 0 \)
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \ sph_\tau(y) \land \exists^{0 \mod 2} y \ sph_\rho(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

\[
\begin{array}{c}
\circ \quad \bullet \quad \circ \quad \circ
\end{array}
\]

Let \( \rho \) be the type with 1 center and radius 2:

\[
\begin{array}{c}
\circ \quad \circ \quad \bullet \quad \circ \quad \circ \quad \circ
\end{array}
\]

Data structure: \( A[\tau] = 0 \), \( A[\rho] = 0 \)

Database:

\[
\begin{array}{c}
a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7
\end{array}
\]
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \ \text{sph}_\tau(y) \land \exists^{0 \mod 2} y \ \text{sph}_\rho(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

\[
\begin{array}{ccc}
\circ & \bullet & \circ \\
\end{array}
\]

Let \( \rho \) be the type with 1 center and radius 2:

\[
\begin{array}{ccc}
\circ & \circ & \bullet \\
\circ & \circ & \circ \\
\end{array}
\]

Data structure: \[ A[\tau] = 1, \quad A[\rho] = 0 \]

Database:
Proof idea (by example)

$$\psi = \exists^{0 \mod 2} y \ sph_\tau(y) \land \exists^{0 \mod 2} y \ sph_\rho(y)$$

Let $\tau$ be the type with 1 center and radius 2:

Let $\rho$ be the type with 1 center and radius 2:

Data structure: $A[\tau] = 1$, $A[\rho] = 1$

Database:
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \ spher_\tau(y) \land \exists^{0 \mod 2} y \ spher_\rho(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

Let \( \rho \) be the type with 1 center and radius 2:

Data structure: \( A[\tau] = 1 \), \( A[\rho] = 1 \)

Database:
Proof idea (by example)

\[ \psi = \exists_{0 \mod 2} y \ sph_{\tau}(y) \land \exists_{0 \mod 2} y \ sph_{\rho}(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

\[ \text{Data structure: } A[\tau] = 0, \ A[\rho] = 1 \]

Database:
Proof idea (by example)

\[ \psi = \exists^{0 \mod 2} y \ \text{sph}_\tau(y) \land \exists^{0 \mod 2} y \ \text{sph}_\rho(y) \]

Let \( \tau \) be the type with 1 center and radius 2:

\[
\begin{array}{c}
\circ & \bullet & \circ & \circ
\end{array}
\]

Let \( \rho \) be the type with 1 center and radius 2:

\[
\begin{array}{c}
\circ & \circ & \bullet & \circ & \circ
\end{array}
\]

Data structure: \( A[\tau] = 1 \) , \( A[\rho] = 1 \)

Database:
Proof idea (by example)

\[
\psi = \exists^{0 \mod 2} y \ \text{sph}_\tau(y) \land \exists^{0 \mod 2} y \ \text{sph}_\rho(y)
\]

Let $\tau$ be the type with 1 center and radius 2:

Let $\rho$ be the type with 1 center and radius 2:

Data structure: $A[\tau] = 1$, $A[\rho] = 1$

Database:
Proof idea (by example)

\( \psi = \exists y \text{mod} 2 \, y \text{sph}_\tau(y) \land \exists y \text{mod} 2 \, y \text{sph}_\rho(y) \)

Let \( \tau \) be the type with 1 center and radius 2:

\[ \text{Data structure: } A[\tau] = 0, \quad A[\rho] = 1 \]

Database:
Proof idea (by example)

$$\psi = \exists^{0 \mod 2} y \text{sph}_\tau(y) \land \exists^{0 \mod 2} y \text{sph}_\rho(y)$$

Let $\tau$ be the type with 1 center and radius 2:

Let $\rho$ be the type with 1 center and radius 2:

Data structure: $A[\tau] = 0$, $A[\rho] = 2$

Database:
Main result for Boolean queries

**Theorem**

There is a dynamic algorithm that receives as input
- a degree bound $d \geq 2$,
- a Boolean FO+$\text{MOD}[\sigma]$-query $\varphi$, and
- a db $D$ of degree $\leq d$,

and computes
- within $f(\varphi, d)\|D\|$ preprocessing time a data structure
- that can be updated in time $f(\varphi, d)$

and allows to return the query result $\varphi(D)$ with answer time $O(1)$.

$$f(\varphi, d) = 3\text{-exp}\left(\|\varphi\| + \log \log d\right)$$
Main result for enumeration

Theorem
There is a dynamic algorithm that receives as input

- a degree bound \( d \geq 2 \),
- a \( k \)-ary \( \text{FO} + \text{MOD}[\sigma] \)-query \( \varphi(\overline{x}) \), and
- a db \( D \) of degree \( \leq d \),

and computes

- within \( f(\varphi, d)\|D\| \) preprocessing time a data structure
- that can be updated in time \( f(\varphi, d) \)

and allows to enumerate \( \varphi(D) \) with delay \( O(k^3) \).

\[
f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)
\]
Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

- a degree bound \( d \geq 2 \),
- a \( k \)-ary FO+MOD[\( \sigma \)]-query \( \varphi(\overline{x}) \), and
- a db \( D \) of degree \( \leq d \),

and computes

- within \( f(\varphi, d) \|D\| \) preprocessing time a data structure
- that can be updated in time \( f(\varphi, d) \)

and allows to enumerate \( \varphi(D) \) with delay \( O(k^3) \).

\[
f(\varphi, d) = 3 \cdot \exp(\|\varphi\| + \log \log d)
\]

Proof Idea:
Proof idea: Reduction to coloured graphs

Input:
Database $D$
$\text{FO+MOD-quer} \varphi(x_1, \ldots, x_k)$

Same approach as in [Durand, S., Segoufin, PODS’14], but now we have to take care of updates!
Proof idea: Reduction to coloured graphs

Input:

Database $D$

FO+$\text{MOD}$-query $\varphi(x_1, \ldots, x_k)$

$\sigma_k := \{E, C_1, \ldots, C_k\}$

$\sigma_k$-structure $\mathcal{G}$

$\psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j)$

$\overline{v} \in \psi_k(\mathcal{G})$

Same approach as in [Durand, S., Segoufin, PODS’14], but now we have to take care of updates!
Proof idea: Reduction to coloured graphs

Input:
Database $D$
FO+MOD-query $\varphi(x_1, \ldots, x_k)$

Enumerate:
$a \in \varphi(D)$

$\sigma_k := \{E, C_1, \ldots, C_k\}$
$\sigma_k$-structure $G$

$\psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j)$

$v \in \psi_k(G)$

Same approach as in [Durand, S., Segoufin, PODS’14],
but now we have to take care of updates!
Representing Databases by Coloured Graphs

\[ \varphi(x_1, \ldots, x_k) \equiv_d \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \quad \& \text{sentences} \]
Representing Databases by Coloured Graphs

$$\varphi(x_1, \ldots, x_k) \equiv_d \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \ & \text{sentences}$$

$$\text{sph}_{\tau}(\bar{x}_1, \ldots, \bar{x}_c) \equiv_d \bigwedge_{j \in \{1, \ldots, c\}} \text{sph}_{\tau_j}(\bar{x}_j) \ & \bigwedge_{j \neq j'} \neg \text{dist}_{\leq 2r+1}(\bar{x}_j, \bar{x}_{j'})$$
Representing Databases by Coloured Graphs

\[ \varphi(x_1, \ldots, x_k) \equiv d \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \quad \& \text{sentences} \]

\[ \text{sph}_{\tau}(\overline{x}_1, \ldots, \overline{x}_c) \equiv d \bigwedge_{j \in \{1, \ldots, c\}} \text{sph}_{\tau_j}(\overline{x}_j) \quad \& \bigwedge_{j \neq j'} \neg \text{dist} \leq 2r + 1(\overline{x}_j, \overline{x}_{j'}) \]

\[ \varphi_c(z_1, \ldots, z_c) := \bigwedge_{j \in \{1, \ldots, c\}} C_j(z_j) \quad \& \bigwedge_{j \neq j'} \neg E(z_j, z_{j'}) \]
Representing Databases by Coloured Graphs

$$\varphi(x_1, \ldots, x_k) \equiv_d \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \quad \& \quad \text{sentences}$$

$$\text{sph}_{\tau}(x_1, \ldots, x_c) \equiv_d \bigwedge_{j \in \{1, \ldots, c\}} \text{sph}_{\tau_j}(x_j)$$

$$\varphi_c(z_1, \ldots, z_c) := \bigwedge_{j \in \{1, \ldots, c\}} C_j(z_j)$$

$$C_j^{G_D} := \{ v_{\bar{a}} : \bar{a} \in \text{adom}(D)^{|x_j|}, (\mathcal{N}_r^{D}(\bar{a}), \bar{a}) \cong \tau_j \}$$

$$\bigwedge_{j \neq j'} \neg \text{dist} \leq 2r+1(x_j, x_{j'})$$

$$\bigwedge_{j \neq j'} \neg \text{E}(z_j, z_{j'})$$
Representing Databases by Coloured Graphs

$$\varphi(x_1, \ldots, x_k) \equiv_d \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \quad \& \quad \text{sentences}$$

$$\text{sph}_{\tau}(\bar{x}_1, \ldots, \bar{x}_c) \equiv_d \bigwedge_{j \in \{1, \ldots, c\}} \text{sph}_{\tau_j}(\bar{x}_j)$$

$$\varphi_c(z_1, \ldots, z_c) := \bigwedge_{j \in \{1, \ldots, c\}} C_j(z_j)$$

$$C_j^{GD} := \{ v_{\bar{a}} : \bar{a} \in \text{dom}(D)|x_j|, (\mathcal{N}^D_r(\bar{a}), \bar{a}) \cong \tau_j \}$$

$$V := \bigcup_{j \in \{1, \ldots, c\}} C_j^{GD}$$

$$\forall \text{dist} \leq 2r + 1(\bar{x}_j, \bar{x}_{j'})$$

$$\land \bigwedge_{j \neq j'} \neg E(z_j, z_{j'})$$
Representing Databases by Coloured Graphs

\[ \varphi(x_1, \ldots, x_k) \equiv_d \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \quad & \text{\& sentences} \]

\[ \text{sph}_\tau(\bar{x}_1, \ldots, \bar{x}_c) \equiv_d \bigwedge_{j \in \{1, \ldots, c\}} \text{sph}_{\tau_j}(\bar{x}_j) \quad & \bigwedge_{j \neq j'} \neg \text{dist}_{\leq 2r+1}((\bar{x}_j, \bar{x}_{j'})} \]

\[ \varphi_c(z_1, \ldots, z_c) := \bigwedge_{j \in \{1, \ldots, c\}} C_j(z_j) \quad & \bigwedge_{j \neq j'} \neg E(z_j, z_{j'}) \]

\[ C_j^{GD} := \{ v_{\bar{a}} : \bar{a} \in \text{adom}(D)^{|x_j|}, (\mathcal{N}_r^{D}(\bar{a}), \bar{a}) \models \tau_j \} \]

\[ V := \bigcup_{j \in \{1, \ldots, c\}} C_j^{GD} \]

\[ E^{GD} := \{ (v_{\bar{a}}, v_{\bar{b}}) \in V^2 : \text{dist}^D(\bar{a}, \bar{b}) \leq 2r + 1 \} \]
Representing Databases by Coloured Graphs

\[ \varphi(x_1, \ldots, x_k) \equiv \bigvee_{i \in I} \text{sph}_{\tau_i}(x_1, \ldots, x_k) \ & \text{& sentences} \]

\[ \text{sph}_{\tau}(\bar{x}_1, \ldots, \bar{x}_c) \equiv \bigwedge_{j \in \{1, \ldots, c\}} \text{sph}_{\tau_j}(\bar{x}_j) \ & \bigwedge \neg \text{dist} \leq 2r + 1(\bar{x}_j, \bar{x}_{j'}) \]

\[ \varphi_c(z_1, \ldots, z_c) := \bigwedge_{j \in \{1, \ldots, c\}} C_j(z_j) \ & \bigwedge \neg E(z_j, z_{j'}) \]

\[ C_j^{GD} := \{ v_{\bar{a}} : \bar{a} \in \text{adom}(D)|^{x_j}, (\mathcal{N}_r^D(\bar{a}), \bar{a}) \cong \tau_j \} \]

\[ V := \bigcup_{j \in \{1, \ldots, c\}} C_j^{GD} \ & \ E^{GD} := \{(v_{\bar{a}}, v_{\bar{b}}) \in V^2 : \text{dist}^D(\bar{a}, \bar{b}) \leq 2r + 1 \} \]

\[(\bar{a}_1, \ldots, \bar{a}_c) \in \text{sph}_{\tau}(D) \iff (v_{\bar{a}_1}, \ldots, v_{\bar{a}_c}) \in \varphi_c(G_D) \]
Updating the graph (1)

Claim

If $D_{\text{new}}$ is obtained from $D_{\text{old}}$ by one update step, then $G_{D_{\text{new}}}$ can be obtained from $G_{D_{\text{old}}}$ by $d^{O(k^2 r + k \|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\| k^2 d^{2r+2})}$. 

▶ Assume, an update command $\text{update } R(a)$ is received
▶ Let $r' := r + (k - 1)(2r + 1)$
▶ Let $D' \in \{D_{\text{old}}, D_{\text{new}}\}$ be the database where $a$ occurs in $R$.
▶ Let $U := N_{D'} r'(a)$
▶ $C_{Dj} := \{v_b : b \in \text{dom}(D) \cap x_j \text{ with } (N_{D'} b, b) \sim \tau_j\}$
▶ Updating the colours:

Before:
$C_{G_{D_{\text{old}}j}}$

1: for $j = 1$ to $c$
2: for every tuple $b \in \bigcup_{\ell=1}^k U_{\ell}$
3: if $(N_{D_{\text{new}}} r(b), b) \sim \tau_j$ then $C_{j} \leftarrow C_{j} \cup \{v_b\}$
4: else $C_{j} \leftarrow C_{j} \setminus \{v_b\}$

Afterwards:
$C_{G_{D_{\text{new}}j}}$
Updating the graph (1)

Claim

If $D_{\text{new}}$ is obtained from $D_{\text{old}}$ by one update step, then $G_{D_{\text{new}}}$ can be obtained from $G_{D_{\text{old}}}$ by $d^{O(k^2 r + k \|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\| k^2 d^2 r + 2)}$.

- Assume, an update command update $R(\overline{a})$ is received
Updating the graph (1)

Claim

If $D_{new}$ is obtained from $D_{old}$ by one update step, then $G_{D_{new}}$ can be obtained from $G_{D_{old}}$ by $d^{O(k^2 r + k \|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\| k^2 d^2 r + 2)}$.

- Assume, an update command update $R(\overline{a})$ is received
- Let $r' := r + (k - 1)(2r + 1)$
Updating the graph (1)

Claim

If $D_{new}$ is obtained from $D_{old}$ by one update step, then $G_{D_{new}}$ can be obtained from $G_{D_{old}}$ by $d^{O(k^2 r + k \|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\| k^2 d^{2r+2})}$.

- Assume, an update command update $R(\bar{a})$ is received
- Let $r' := r + (k - 1)(2r + 1)$
- Let $D' \in \{D_{old}, D_{new}\}$ be the database where $\bar{a}$ occurs in $R$. 
Updating the graph (1)

Claim

If $D_{\text{new}}$ is obtained from $D_{\text{old}}$ by one update step, then $G_{D_{\text{new}}}$ can be obtained from $G_{D_{\text{old}}}$ by $d^{O(k^2r+k\|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\|k^2d^2r+2)}$.

- Assume, an update command $\text{update } R(\bar{a})$ is received
- Let $r' := r + (k - 1)(2r + 1)$
- Let $D' \in \{D_{\text{old}}, D_{\text{new}}\}$ be the database where $\bar{a}$ occurs in $R$.
- Let $U := N_{r'}^D(\bar{a})$
Updating the graph (1)

Claim

If \( D_{\text{new}} \) is obtained from \( D_{\text{old}} \) by one update step, then \( G_{D_{\text{new}}} \) can be obtained from \( G_{D_{\text{old}}} \) by \( d^{\mathcal{O}(k^2 r + k \| \sigma \|)} \) update steps and additional computing time \( 2^{\mathcal{O}(\| \sigma \| k^2 d^{2r+2})} \).

- Assume, an update command update \( R(\bar{a}) \) is received
- Let \( r' := r + (k - 1)(2r + 1) \)
- Let \( D' \in \{D_{\text{old}}, D_{\text{new}}\} \) be the database where \( \bar{a} \) occurs in \( R \).
- Let \( U := N^{D'}_{r'}(\bar{a}) \)
- \( C^D_j := \{v_{\bar{b}} : \bar{b} \in \text{adom}(D)^{|x_j|} \text{ with } (N^D_r(\bar{b}), \bar{b}) \approx \tau_j\} \)
Updating the graph (1)

Claim

If $D_{\text{new}}$ is obtained from $D_{\text{old}}$ by one update step, then $G_{D_{\text{new}}}$ can be obtained from $G_{D_{\text{old}}}$ by $d^{O(k^2 r + k \|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\| k^2 d^2 r^2 + 2)}$.

- Assume, an update command update $R(\bar{a})$ is received
- Let $r' := r + (k - 1)(2r + 1)$
- Let $D' \in \{D_{\text{old}}, D_{\text{new}}\}$ be the database where $\bar{a}$ occurs in $R$.
- Let $U := N_{r'}^{D'}(\bar{a})$
- Let $C_j^D := \{v_b : b \in \text{dom}(D) \mid x_j \text{ with } (N_r^D(b), b) \equiv \tau_j\}$

Updating the colours:

1. for $j = 1$ to $c$ do
2. for every tuple $\bar{b} \in \bigcup_{\ell=1}^{k} U^\ell$ do
3. if $(N_r^{D_{\text{new}}}(\bar{b}), \bar{b}) \equiv \tau_j$ then $C_j \leftarrow C_j \cup \{v_{\bar{b}}\}$
4. else $C_j \leftarrow C_j \setminus \{v_{\bar{b}}\}$

Afterwards: $C_j = C_j^{G_{D_{\text{old}}}}$
Updating the graph (2)

\[ E^{GD} := \{(v_a, v_b) \in V^2 : \text{dist}^D(a, b) \leq 2r + 1\} \]
Updating the graph (2)

\[ E^{GD} := \{ (v_{\overline{a}}, v_{\overline{b}}) \in V^2 : \text{dist}^D(\overline{a}, \overline{b}) \leq 2r + 1 \} \]

Updating the edges:

Before: \( E = E^{GD}_{\text{old}} \)

1. for every tuple \( \overline{b} \in \bigcup_{\ell=1}^{k} U^\ell \) do
2. \hspace{1em} for every tuple \( \overline{b}' \in \bigcup_{\ell=1}^{k} U^\ell \) do
3. \hspace{2em} if condition (1), (2) and (3) holds then
4. \hspace{3em} \( E \leftarrow E \cup \{(v_{\overline{b}}, v_{\overline{b}'})\} \)
5. \hspace{2em} else
6. \hspace{3em} \( E \leftarrow E \setminus \{(v_{\overline{b}}, v_{\overline{b}'})\} \)

Afterwards: \( E = E^{GD}_{\text{new}} \)
Updating the graph (2)

\[ E^{GD} := \{(v_a, v_b) \in V^2 : \text{dist}^D(a, b) \leq 2r + 1\} \]

Updating the edges:

Before: \( E = E^{GD}_{old} \)

1: for every tuple \( \bar{b} \in \bigcup_{\ell=1}^k U^\ell \) do
2: for every tuple \( \bar{b}' \in \bigcup_{\ell=1}^k U^\ell \) do
3: if condition (1), (2) and (3) holds then
4: \( E \leftarrow E \cup \{(v_{\bar{b}}, v_{\bar{b}'})\} \)
5: else
6: \( E \leftarrow E \setminus \{(v_{\bar{b}}, v_{\bar{b}'})\} \)

Afterwards: \( E = E^{GD}_{new} \)

Conditions:

(1) There is a \( j \in \{1, \ldots, c\} \) such that \( \bar{b} \in C^G_j \)
(2) There is a \( j' \in \{1, \ldots, c\} \) such that \( \bar{b}' \in C^G_{j'} \)
(3) \( \text{dist}^{D_{new}}(\bar{b}, \bar{b}') \leq 2r + 1 \)
Enumeration with delay $O(k^3d)$

$$
\psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j)
$$

for all $u_1 \in C_1^G$ do

$\text{Enum}(u_1)$.

Output EOE.

function $\text{Enum}(u_1, \ldots, u_i)$

if $i = k$ then

Output $(u_1, \ldots, u_i)$

else

for all $u_{i+1} \in C_{i+1}^G$ do

if $u_{i+1} \notin \bigcup_{j=1}^{i} N^G(u_j)$ then

$\text{Enum}(u_1, \ldots, u_i, u_{i+1})$
Enumeration with delay $O(k^3 d)$

$$
\psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j)
$$

for all $u_1 \in C_1^G$ do

Enum($u_1$).

Output EOE.

function Enum($u_1, \ldots, u_i$)

if $i = k$ then

Output ($u_1, \ldots, u_i$)

else

for all $u_{i+1} \in C_{i+1}^G$ do

if $u_{i+1} \notin \bigcup_{j=1}^{i} N^G(u_j)$ then

Enum($u_1, \ldots, u_i, u_{i+1}$)

endif

endif

endif
Enumeration with delay $O(k^3d)$

$$\psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j)$$

for all $u_1 \in C_1^G$ do

Enum($u_1$).

Output EOE.

function Enum($u_1, \ldots, u_i$)

if $i = k$ then

Output ($u_1, \ldots, u_i$)

else

for all $u_{i+1} \in C_{i+1}^G$ do

if $u_{i+1} \notin \bigcup_{j=1}^{i} N^G(u_j)$ then

Enum($u_1, \ldots, u_i, u_{i+1}$)
Enumeration with delay $O(k^3d)$

\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]

Problem: Too few blue nodes

for all $u_1 \in C_1^G$ do
- \( \text{Enum}(u_1) \).
- Output EOE.

function \( \text{Enum}(u_1, \ldots, u_i) \)
  if $i = k$ then
  Output \((u_1, \ldots, u_i)\)
  else
  for all $u_{i+1} \in C_{i+1}^G$ do
    if $u_{i+1} \notin \bigcup_{j=1}^{i} N^G(u_j)$ then
      \( \text{Enum}(u_1, \ldots, u_i, u_{i+1}) \)
Handling small colours

A colour $\ell \in \{1, \ldots, k\}$ is small $\iff |C^G_\ell| \leq dk$
Handling small colours

A colour $\ell \in \{1, \ldots, k\}$ is small $\iff |C^G_\ell| \leq dk$

W.l.o.g. let $I = \{1, \ldots, s\}$ be the set of small colours (with $s \leq k$).
Handling small colours

A colour \( \ell \in \{1, \ldots, k\} \) is small \( \iff \) \(|C_\ell^G| \leq dk\)

W.l.o.g. let \( I = \{1, \ldots, s\} \) be the set of small colours (with \( s \leq k \)).

\[
S := \left\{ (u_1, \ldots, u_s) \in C_1^G \times \cdots \times C_s^G : (u_j, u_{j'}) \notin E^G, \text{ for all } j \neq j' \right\}
\]

The set \( S \) can be computed in time \( O((dk)^k) \).
Handling small colours

A colour $\ell \in \{1, \ldots, k\}$ is small $\iff |C_{\ell}^G| \leq dk$

W.l.o.g. let $I = \{1, \ldots, s\}$ be the set of small colours (with $s \leq k$).

$$S := \left\{(u_1, \ldots, u_s) \in C_1^G \times \cdots \times C_s^G : (u_j, u_{j'}) \notin E^G, \text{ for all } j \neq j'\right\}$$

The set $S$ can be computed in time $O((dk)^k)$.

$$\bar{s} \in S \iff \text{ex. } \bar{a} \text{ such that } (\bar{s}, \bar{a}) \in \varphi(D)$$
The enumeration procedure

1: for all \((u_1, \ldots, u_s) \in S\) do
2: \(\text{Enum}(u_1, \ldots, u_s)\).
3: Output the end-of-enumeration message \(\text{EOE}\).

5: function \(\text{Enum}(u_1, \ldots, u_i)\)
6: if \(i = k\) then
7: output the tuple \((u_1, \ldots, u_i)\)
8: else
9: for all \(u_{i+1} \in C_{i+1}\) do
10: if \(u_{i+1} \notin \bigcup_{j=1}^{i} N^G(u_j)\) then
11: \(\text{Enum}(u_1, \ldots, u_i, u_{i+1})\)

where \(N^G(u_j) := \{v \in V^G : (u_j, v) \in E^G\}\).
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]

**update step:** Insert a node into a colour \( C_\ell \) with \( |C_\ell^G| = dk \)
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]

**update step:** Insert a node into a colour \( C_\ell \) with \( |C_\ell^G| = dk \)
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]

large colours

update step: Delete a node from a colour \( C_\ell \) with \( |C_\ell^G| = dk + 1 \)
\[ \psi_k(x_1, \ldots, x_k) := \bigwedge_{i=1}^{k} C_i(x_i) \land \bigwedge_{i \neq j} \neg E(x_i, x_j) \]

**update step:** Delete a node from a colour \( C_{\ell} \) with \( |C_{\ell}^{G}| = dk + 1 \)
Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

1. a degree bound \(d \geq 2\),
2. a \(k\)-ary \(\text{FO} + \text{MOD}[\sigma]\)-query \(\varphi(\overline{x})\), and
3. a db \(D\) of degree \(\leq d\),

and computes

1. within \(f(\varphi, d)\|D\|\) preprocessing time a data structure
2. that can be updated in time \(f(\varphi, d)\)

and allows to enumerate \(\varphi(D)\) with delay \(O(k^3)\).

\[
f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d)\]

Proof Idea:

Nicole Schweikardt (HU Berlin) Database Theory and Query Answering under Updates 42/43
Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

- a degree bound $d \geq 2$,
- a $k$-ary $\text{FO} + \text{MOD}[\sigma]$-query $\varphi(\overline{x})$, and
- a db $D$ of degree $\leq d$,

and computes

- within $f(\varphi, d)\|D\|$ preprocessing time a data structure
- that can be updated in time $f(\varphi, d)$

and allows to enumerate $\varphi(D)$ with delay $O(k^3) f(\varphi, d)$.

\[
f(\varphi, d) = 3\cdot \exp(\|\varphi\| + \lg \lg d)
\]

For enumeration with delay $O(k^3)$: Use the skip-pointers that were introduced by [Durand, S., Segoufin, PODS’14] for the static setting and lift the approach to the dynamic setting.
Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

- a degree bound \( d \geq 2 \),
- a \( k \)-ary \( \text{FO}+\text{MOD}[\sigma] \)-query \( \varphi(\overline{x}) \), and
- a db \( D \) of degree \( \leq d \),

and computes

- within \( f(\varphi, d)\|D\| \) preprocessing time a data structure
- that can be updated in time \( f(\varphi, d) \)

and allows to enumerate \( \varphi(D) \) with delay \( O(k^3) \) \( f(\|\varphi\|/d)/O(k^3) \).

\[ f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d) \]

For enumeration with delay \( O(k^3) \): Use the skip-pointers that were introduced by [Durand, S., Segoufin, PODS’14] for the static setting and lift the approach to the dynamic setting.
Summary [Berkholz, Keppeler, S., ICDT'17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - Query $\varphi(x_1, \ldots, x_k)$ in FO$\!+$MOD$[\sigma]$ in time $f(\varphi, d) = 3$-exp($|\varphi| + \lg \lg d$)

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries: Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
    - Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$

- **Dynamic setting:**
  Update data structure in time $f(\varphi, d)$
  Tuples may be inserted into or deleted from $D$

---

Future task: Revisit other results on FO model checking in the dynamic setting!
Summary [Berkholz, Keppeler, S., ICDT’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - Query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[$\sigma$]

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - For Boolean queries:
    - Decide if $D \models \varphi$ in time $O(1)$
  - For $k$-ary queries:
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
    - Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$

- **Dynamic setting:**
  - Update data structure in time $f(\varphi, d)$
  - Tuples may be inserted into or deleted from $D$

Similar results for FO with counting $\text{FOC}(\mathbb{P})$ [Kuske, S., LICS’17].
Summary

[Berkholz, Keppeler, S., ICDT’17]

- **Input:**
  - Database $D$ of degree $\leq d$
  - Query $\varphi(x_1, \ldots, x_k)$ in $\text{FO} + \text{MOD}[\sigma]$

- **Combined complexity**
  \[ f(\varphi, d) = 3\text{-exp}(\|\varphi\| + \lg \lg d) \]

- **Preprocessing:**
  Build a suitable data structure that represents $D$ and $\varphi(D)$

- **Output:**
  - **For Boolean queries:**
    - Decide if $D \models \varphi$ in time $O(1)$
  - **For $k$-ary queries:**
    - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
    - Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
    - Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$

- **Dynamic setting:**
  - Update data structure in time $f(\varphi, d)$
  - Tuples may be inserted into or deleted from $D$

Similar results for FO with counting $\text{FOC}(\mathbb{P})$ [Kuske, S., LICS’17].

**Future task:** Revisit other results on FO model checking in the dynamic setting!
Summary

Input:
- Database $D$ of degree $\leq d$
- query $\varphi(x_1, \ldots, x_k)$ in $\text{FO} + \text{MOD}[\sigma]$

combined complexity

$\varphi(D)$

$\|D\|$ in time $f(\varphi, d)\|D\|$ in time $3\cdot \exp(\|\varphi\| + \lg \lg d)$

Preprocessing:
Build a suitable data structure that represents $D$ and $\varphi(D)$

Output:
For Boolean queries:
- Decide if $D \models \varphi$ in time $O(1)$
For $k$-ary queries:
- Compute the number of tuples in $\varphi(D)$ in time $O(1)$
- Test for a given tuple $\bar{a}$ whether $\bar{a} \in \varphi(D)$ in time $O(k^2)$
- Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$

Dynamic setting: update data structure in time $f(\varphi, d)$
Tuples may be inserted into or deleted from $D$

Similar results for FO with counting $\text{FOC}(\mathbb{P})$ [Kuske, S., LICS’17].

Future task: Revisit other results on FO model checking in the dynamic setting!

– Thank you! –