KAPITEL 3 BERUHT AUF EINEM AUSZUG AUS DEM FOLGENDEN VORTRAG:

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

Example database and two queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Programme		
Cinema	Movietitle	Time
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Resident Evil	20:00
Urania	Resident Evil	21:30
Urania	Resident Evil	23:00

Example database and two queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Programme		
Cinema	Movietitle	Time
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Resident Evil	20:00
Urania	Resident Evil	21:30
Urania	Resident Evil	23:00

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

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

Example database and two queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Programme		
Cinema	Movietitle	Time
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Resident Evil	20:00
Urania	Resident Evil	21:30
Urania	Resident Evil	23:00

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

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

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

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

Example database and two queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Programme		
Cinema	Movietitle	Time
Babylon	Casablanca	17:30
Babylon	Gravity	20:15
Casablanca	Blade Runner	15:30
Casablanca	Alien	18:15
Casablanca	Blade Runner	20:30
Casablanca	Resident Evil	20:30
Kino International	Casablanca	18:00
Kino International	Brazil	20:00
Kino International	Brazil	22:00
Moviemento	Gravity	17:00
Moviemento	Gravity	19:30
Moviemento	Alien	22:00
Urania	Resident Evil	20:00
Urania	Resident Evil	21:30
Urania	Resident Evil	23:00

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

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

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

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

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

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

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

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

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

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 $\sigma := \{M, D\}$

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

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

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

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

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 $\mathbf{dom}^2$
P^D	:	a finite subset of $\mathbf{dom}^3$
dom	:	a fixed, infinite domain of potential db entries

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

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

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

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

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

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

$\varphi_1(y) :=$ *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:

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

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

View D as a finite σ-structure with universe adom(D)!

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

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

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

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

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

View D as a finite σ-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:

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

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

View D as a finite σ-structure with universe $\mathrm{adom}(D)$!

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:

$$\varphi_2(x, y) := \exists z \, \big(P(x, y, z) \wedge M(y, \text{"Sigourney Weaver"}) \big)$$

Conjunctive queries!

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 σ, formulated in L.
Let D be a database of signature σ.

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 σ, formulated in L.
Let D be a database of signature σ.

Task:

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

$$\varphi(D) \ := \ [\![\varphi(x_1, \ldots, x_k)]\!](D) \ :=$$

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

Query evaluation

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

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

Task:

Evaluate $\varphi(x_1, \dots, x_k)$ on D, i.e., compute the set

$$\varphi(D) \;:=\; [\![\varphi(x_1, \dots, x_k)]\!](D) \;:=$$

$$\left\{ \, (a_1, \dots, a_k) \in \mathrm{adom}(D)^k \;:\; (\mathrm{adom}(D), D) \models \varphi \left[\tfrac{a_1 \cdots a_k}{x_1 \cdots x_k} \right] \, \right\}$$

Special case $k = 0$: Boolean queries:

Evaluate $\varphi()$ on D means Decide if $(\mathrm{adom}(D), D) \models \varphi$

Complexity of query evaluation

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

combined complexity

and data complexity

Complexity of query evaluation

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

combined complexity: Measure the complexity of evaluating φ on D in terms of the sizes of φ and D.

and data complexity

Complexity of query evaluation

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

combined complexity: Measure the complexity of evaluating φ on D in terms of the sizes of φ and D.

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

Complexity of query evaluation

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

combined complexity: Measure the complexity of evaluating φ on D in terms of the sizes of φ and D.

and data complexity: Assume the query φ to be fixed. Measure the complexity of evaluating φ 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]

Complexity of query evaluation

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

combined complexity: Measure the complexity of evaluating φ on D in terms of the sizes of φ and D.

and data complexity: Assume the query φ to be fixed. Measure the complexity of evaluating φ 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]

Complexity of query evaluation

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

combined complexity: Measure the complexity of evaluating φ on D in terms of the sizes of φ and D.

and data complexity: Assume the query φ to be fixed. Measure the complexity of evaluating φ 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].

Complexity of query evaluation

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

combined complexity: Measure the complexity of evaluating φ on D in terms of the sizes of φ and D.

and data complexity: Assume the query φ to be fixed. Measure the complexity of evaluating φ 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 $\varphi(x_1, \dots, x_k)$

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 $\varphi(D)$

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 $D \models \varphi$

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:

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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$

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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$
  - 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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$
- **Dynamic setting:**
  Tuples may be inserted into or deleted from D

Overview

Overview

Introduction

Conjunctive Queries on Arbitrary Databases

First-Order Queries on Bounded Degree Databases

Conjunctive queries (CQs)

Conjunctive queries:

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

Conjunctive queries (CQs)

Conjunctive queries:

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

Complexity of query evaluation:

Obvious: static time $\leqslant$ update time $\cdot$ $\|D\|$

Conjunctive queries (CQs)

Conjunctive queries:

$$\varphi(x_1,\ldots,x_\ell) \quad := \quad \exists x_{\ell+1}\cdots\exists x_m \,\big(\, R_1(\overline{x}) \wedge \cdots \wedge R_s(\overline{x}) \,\big)$$

Complexity of query evaluation:

Obvious: static time $\leqslant$ update time $\cdot$ $\|D\|$

Thus:

- constant update time $\Longrightarrow$ static setting has linear data complexity

Conjunctive queries (CQs)

Conjunctive queries:

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

Complexity of query evaluation:

Obvious: static time $\leqslant$ 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:

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

Complexity of query evaluation:

Obvious: static time $\leqslant$ update time $\cdot \ \|D\|$

Thus:

- constant update time $\Longrightarrow$ static setting has linear data complexity
- $n^{O(1)}$ update time $\Longleftrightarrow$ $n^{O(1)}$ 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:

$$\varphi(x_1, \ldots, x_\ell) \quad := \quad \exists x_{\ell+1} \cdots \exists x_m \,\big(\, R_1(\overline{x}) \,\wedge\, \cdots \,\wedge\, R_s(\overline{x}) \,\big)$$

Complexity of query evaluation:

Obvious: static time $\leqslant$ update time $\cdot$ $\|D\|$

Thus:

- constant update time $\Longrightarrow$ static setting has linear data complexity
- $n^{O(1)}$ update time $\Longleftrightarrow$ $n^{O(1)}$ 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

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

- **Input:** data complexity
  - 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 $\overline{a}$ 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($\|D\|$).

Scenario

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

- **Input:** data complexity
  - 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 $\iff$ φ 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.

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.

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 φ 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 $\text{atoms}(x) \subsetneq \text{atoms}(y)$ 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.

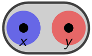
Definition: A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables x, y of φ the following is satisfied:

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

Queries that are not q-hierarchical:

$$\psi_{S\text{-}E\text{-}T}() \quad := \quad \exists x\, \exists y\, \big(\, S(x) \wedge E(x,y) \wedge T(y)\, \big)$$



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.

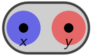
Definition: A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables x, y of φ the following is satisfied:

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

Queries that are not q-hierarchical:

$$\psi_{S\text{-}E\text{-}T}() \quad := \quad \exists x\, \exists y \; \big(\; S(x) \wedge E(x,y) \wedge T(y) \; \big)$$



$$\varphi_{S\text{-}E\text{-}T}(x,y) \quad := \quad S(x) \wedge E(x,y) \wedge T(y)$$

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.

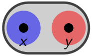
Definition: A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables x, y of φ the following is satisfied:

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

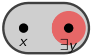
Queries that are not q-hierarchical:

$$\psi_{S\text{-}E\text{-}T}() \;:=\; \exists x\, \exists y\; \big(\, S(x) \wedge E(x,y) \wedge T(y) \,\big)$$



$$\varphi_{S\text{-}E\text{-}T}(x,y) \;:=\; S(x) \wedge E(x,y) \wedge T(y)$$

$$\varphi_{E\text{-}T}(x) \;:=\; \exists y\; \big(\, E(x,y) \wedge T(y) \,\big)$$



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.

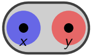
Definition: A CQ $\varphi(z_1, \ldots, z_k)$ is q-hierarchical if for all variables x, y of φ the following is satisfied:

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

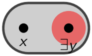
Queries that are not q-hierarchical:

$$\psi_{S\text{-}E\text{-}T}() \quad := \quad \exists x \, \exists y \; \big(\, S(x) \wedge E(x,y) \wedge T(y) \, \big)$$



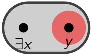
$$\varphi_{S\text{-}E\text{-}T}(x,y) \quad := \quad S(x) \wedge E(x,y) \wedge T(y)$$

$$\varphi_{E\text{-}T}(x) \quad := \quad \exists y \; \big(\, E(x,y) \wedge T(y) \, \big)$$



A q-hierarchical query:

$$\theta_{E\text{-}T}(y) \quad := \quad \exists x \; \big(\, E(x,y) \wedge T(y) \, \big)$$



Scenario

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

- **Input:** data complexity
  - 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 $\overline{a}$ 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 $\iff$ φ 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_ℓ 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_ℓ 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_ℓ 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_ℓ 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.

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_ℓ 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 $\varphi_{E\text{-}T}(x) \ := \ \exists y \left(E(x,y) \wedge T(y) \right)$

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \, \big(E(x,y) \wedge T(y) \big)$

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_ℓ before accessing $v_{\ell+1}$

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \, \big(\, E(x,y) \wedge T(y) \, \big)$

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_ℓ before accessing $v_{\ell+1}$

Given $n \times n$ matrix M, let

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

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

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \, \big(\, E(x,y) \wedge T(y) \, \big)$

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_ℓ before accessing $v_{\ell+1}$

Given $n \times n$ matrix M, let

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

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

Given n-dim vector v_ℓ, update

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

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

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \, \big(\, E(x,y) \wedge T(y) \, \big)$

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_ℓ 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_ℓ, update

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

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

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

Proof idea for $\varphi_{E\text{-}T}(x) \;:=\; \exists y\,\big(\,E(x,y) \wedge T(y)\,\big)$

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_ℓ before accessing $v_{\ell+1}$

Given $n \times n$ matrix M, let

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

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

Given n-dim vector v_ℓ, update

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

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

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

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

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \, \big(E(x,y) \wedge T(y) \big)$

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_ℓ 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_ℓ, update

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

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

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

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

This solves OMv in total time $O(n^{3-\varepsilon})$ $\lightning$

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 $(u_\ell)^\intercal M v_\ell$ 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 $(u_\ell)^\mathsf{T} M v_\ell$ 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 $(u_\ell)^\top M v_\ell$ 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 φ 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.

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 $(u_\ell)^\mathsf{T} M v_\ell$ 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 φ 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 $\psi_{S\text{-}E\text{-}T} := \exists x \exists y \, \big(S(x) \wedge E(x,y) \wedge T(y) \big)$

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 $u^\mathsf{T} v = 0$

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 $u^\mathsf{T} v = 0$

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.

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 $u^{\mathsf{T}} v = 0$

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

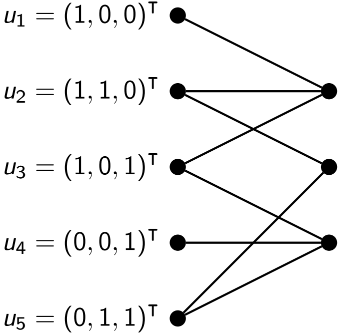
Theorem (Counting): [Berkholz, Keppeler, S., PODS'17]
Let $\epsilon > 0$ and let $\varphi(\overline{x})$ be a CQ whose homomorphic core is not q-hierarchical.
Then, there is no algorithm with arbitrary preprocessing time and $t_u = O(n^{1-\epsilon})$ update time that computes $|\varphi(D)|$ in time $t_c = O(n^{1-\epsilon})$, unless the OV-conjecture or the OuMv-conjecture fails.

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \ \big(\, E(x,y) \wedge T(y) \, \big)$

Proof idea for $\varphi_{E\text{-}T}(x) \ := \ \exists y \ \big(\, E(x,y) \wedge T(y) \, \big)$

A lower bound for counting via OV

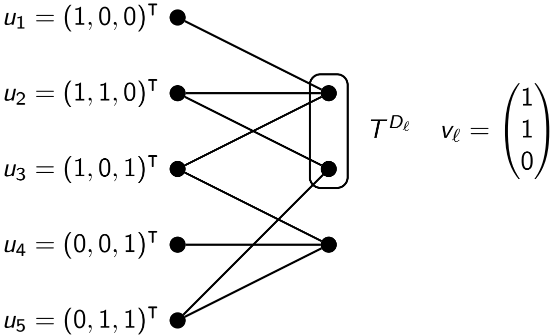
Left: n vertices for the n vectors $u \in U$
Right: $d := \lceil \log^2 n \rceil$ vertices for vector-coordinates



Proof idea for $\varphi_{E\text{-}T}(x) \;:=\; \exists y\,\big(\,E(x,y) \wedge T(y)\,\big)$

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



- for each $v_\ell \in V$: update T^{D_ℓ} in time $d \cdot n^{1-\epsilon} \;=\; \lceil \log^2 n \rceil n^{1-\varepsilon}$

Proof idea for $\varphi_{E\text{-}T}(x) \; := \; \exists y \, \big(\, E(x,y) \wedge T(y) \, \big)$

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

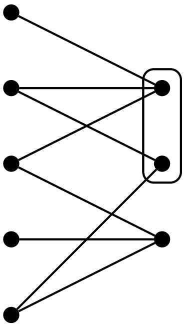
$u_1 = (1,0,0)^\mathsf{T}$

$u_2 = (1,1,0)^\mathsf{T}$

$u_3 = (1,0,1)^\mathsf{T}$

$u_4 = (0,0,1)^\mathsf{T}$

$u_5 = (0,1,1)^\mathsf{T}$



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

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

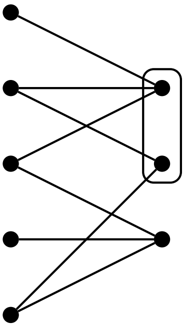
- for each $v_\ell \in V$: update T^{D_ℓ} in time $d \cdot n^{1-\epsilon} \; = \; \lceil \log^2 n \rceil n^{1-\varepsilon}$
- there is $u_i \in U$ with ${u_i}^\mathsf{T} v_\ell = 0 \iff |\varphi_{E\text{-}T}(D_\ell)| < n$.

Proof idea for $\varphi_{E\text{-}T}(x) \; := \; \exists y \, \big(\, E(x,y) \wedge T(y) \, \big)$

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

$u_1 = (1,0,0)^\mathsf{T}$
$u_2 = (1,1,0)^\mathsf{T}$
$u_3 = (1,0,1)^\mathsf{T}$
$u_4 = (0,0,1)^\mathsf{T}$
$u_5 = (0,1,1)^\mathsf{T}$



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

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

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

Scenario

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

- **Input:** data complexity
  - 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 $\overline{a}$ 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** data complexity
  - 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 $\overline{a}$ 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 $\iff$ φ is q-hierarchical.

Scenario
[Berkholz, Keppeler, S., PODS'17]

- **Input:** data complexity
  - Database D arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ
- **Preprocessing:** in time $O(\|D\|)$
  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 $\overline{a}$ 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** data complexity
  - Database D arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ
- **Preprocessing:** in time $O(\|D\|)$
  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 $\overline{a}$ 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** data complexity
  - Database D arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ
- **Preprocessing:** in time $O(\|D\|)$
  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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** data complexity
  - Database D arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ
- **Preprocessing:** in time $O(\|D\|)$
  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 $\varphi(D)$
- **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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** data complexity
  - Database D arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ
- **Preprocessing:** in time $O(\|D\|)$
  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 $\varphi(D)$ 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** data complexity
  - Database D arbitrary
  - query $\varphi(x_1, \ldots, x_k)$ **q-hierarchical** CQ
- **Preprocessing:** in time $O(\|D\|)$
  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 polylog$(\|D\|)$.

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

Scenario

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 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 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 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 $\text{poly}(\varphi)$

  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 $\iff$ φ is q-hierarchical.

Scenario [Berkholz, Keppeler, S., PODS'17]

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 $\overline{a}$ 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 $\bar{a}$ 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 $\iff$ φ is q-hierarchical.

Scenario

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 $\bar{a}$ 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 $\iff$ φ 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

Efficient evaluation of a fragment of CQs

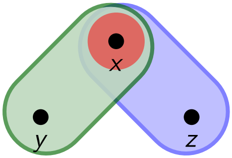
Theorem (Upper bound):

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

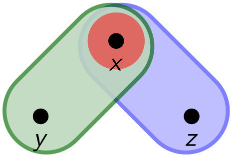
$$\varphi(x, y, z) \ := \ R(x) \wedge E(x, y) \wedge F(x, z)$$



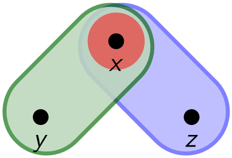
q-hierarchical queries

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

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



q-hierarchical queries



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

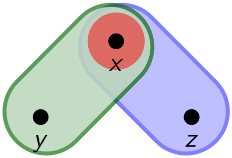
$$|\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$$

- COUNT: store $|N_E^+(v)|$, $|N_F^+(v)|$, $\sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$

q-hierarchical queries

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

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

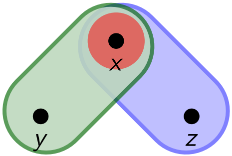


- COUNT: store $|N_E^+(v)|$, $|N_F^+(v)|$, $\sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$
- ENUM: store $N_E^+(v)$, $N_F^+(v)$ as lists with constant access,
  for $v \in R^D$ report $\{v\} \times N_E^+(v) \times N_F^+(v)$

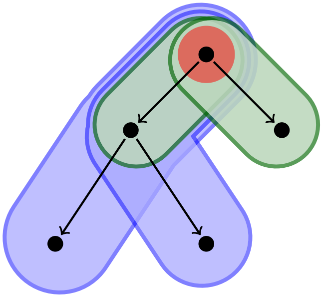
q-hierarchical queries

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

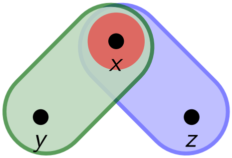
$|\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$



- 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



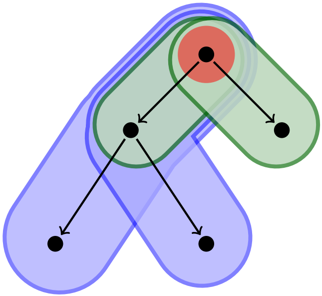
$$\varphi(x,y,z) \ := \ R(x) \wedge E(x,y) \wedge F(x,z)$$

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

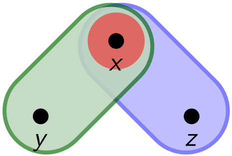
- COUNT: store $|N_E^+(v)|$, $|N_F^+(v)|$, $\sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$
- ENUM: store $N_E^+(v)$, $N_F^+(v)$ as lists with constant access,
  for $v \in R^D$ report $\{v\} \times N_E^+(v) \times N_F^+(v)$

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



$\varphi(x, y, z) \ := \ {\color{red}R(x)} \wedge {\color{green}E(x, y)} \wedge {\color{blue}F(x, z)}$

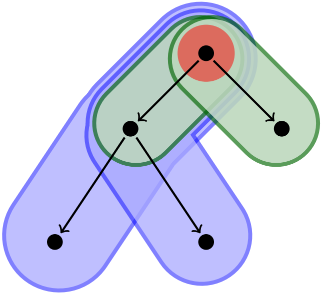
$|\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$

- 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 $\{v\} \times N_E^+(v) \times N_F^+(v)$

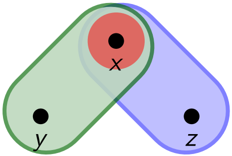
Definition (q-tree):

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

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



q-hierarchical queries



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

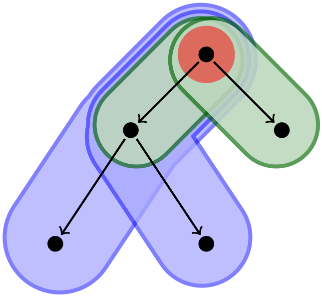
$$|\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$$

- 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 $\{v\} \times N_E^+(v) \times N_F^+(v)$

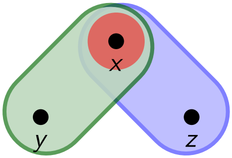
Definition (q-tree):

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 φ: $\{y_1, \ldots, y_r\}$ 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



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

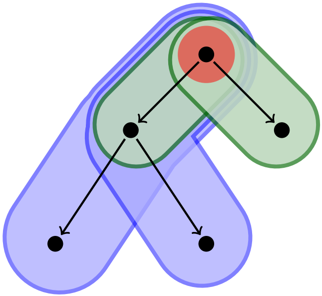
$|\varphi(D)| = \sum_{v \in R^D} |N_E^+(v)| \cdot |N_F^+(v)|$

- 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

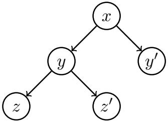
1. for every $R(y_1,\ldots,y_r)$ in φ: $\{y_1,\ldots,y_r\}$ forms a path in T that starts at the root
2. the free variables $\{x_1,\ldots,x_\ell\}$ form a connected subtree that contains the root



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

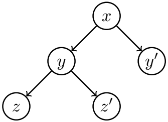
Data structure for q-hierarchical queries

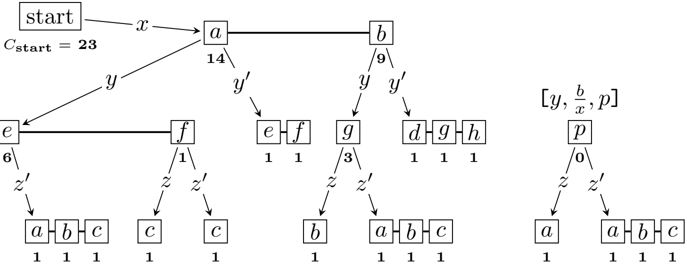
$$\varphi(x, y, z, y', z') = \big(Rxyz \wedge Rxyz' \wedge Exy \wedge Exy' \wedge Sxyz \big)$$



Data structure for q-hierarchical queries

$$\varphi(x,y,z,y',z') = \big(\, Rxyz \,\wedge\, Rxyz' \,\wedge\, Exy \,\wedge\, Exy' \,\wedge\, Sxyz \,\big)$$

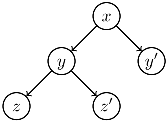


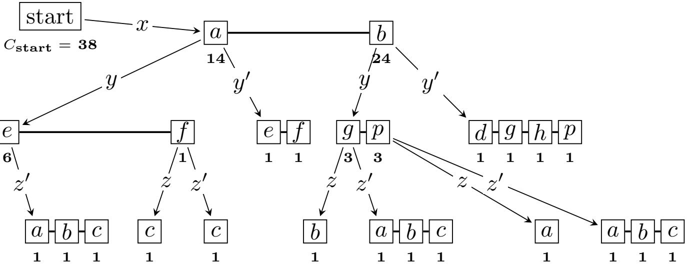


- $S(b,p,a), R(b,p,a), R(b,p,b), R(b,p,c) \in D,\;\; E(b,p) \notin D$

Data structure for q-hierarchical queries

$$\varphi(x, y, z, y', z') = \big(Rxyz \wedge Rxyz' \wedge Exy \wedge Exy' \wedge Sxyz \big)$$





- $S(b,p,a), R(b,p,a), R(b,p,b), R(b,p,c) \in D, \;\; E(b,p) \notin D$
- INSERT $E(b,p)$

Summary

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 $\overline{a}$ 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

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 $\iff$ φ is q-hierarchical.

Summary

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 $\bar{a}$ 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 $\iff$ φ is q-hierarchical.

Ongoing work: Similar results for UCQs & FDs.

Summary

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

- **Input:** combined complexity
  - Database D arbitrary $\text{poly}(\varphi) := \|\varphi\|^{O(1)}$
  - 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 $\overline{a}$ 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

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

$$\exists^{0 \bmod 2} y \;\; \mathit{Movie}(y, \text{"Sigourney Weaver"})$$

FO+MOD queries and FOC($\mathbb{P}$) queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
$\vdots$	$\vdots$

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

$\exists^{0 \bmod 2} y \;\; \mathit{Movie}(y, \text{"Sigourney Weaver"})$

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

FO+MOD queries and FOC($\mathbb{P}$) queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
$\vdots$	$\vdots$

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

$$\exists^{0 \bmod 2} y \;\; Movie(y, \text{"Sigourney Weaver"})$$

In FOC($\mathbb{P}$):

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

FO+MOD $=$ extension of first-order logic with modulo-counting quantifiers $\exists^{i \bmod m} y \; \psi(y, \overline{z})$

FO+MOD queries and FOC($\mathbb{P}$) queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

$$\exists^{0 \bmod 2} y \;\; \mathit{Movie}(y, \text{"Sigourney Weaver"})$$

In FOC($\mathbb{P}$):

$$P_{even}\big(\, \#(y).\mathit{Movie}(y, \text{"Sigourney Weaver"})\,\big)$$

FO+MOD = extension of first-order logic with modulo-counting quantifiers $\exists^{i \bmod m} y \; \psi(y, \overline{z})$

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

FO+MOD queries and FOC($\mathbb{P}$) queries

Movie	
Name	Actor
Alien	Sigourney Weaver
Blade Runner	Harrison Ford
Blade Runner	Sean Young
Brazil	Jonathan Pryce
Brazil	Kim Greist
Casablanca	Humphrey Bogart
Casablanca	Ingrid Bergmann
Gravity	Sandra Bullock
Gravity	George Clooney
Resident Evil	Milla Jovovich
Terminator	Arnold Schwarzenegger
Terminator	Linda Hamilton
Terminator	Michael Biehn
⋮	⋮

Is the number of movies with Sigourney Weaver even?

In FO+MOD:

$$\exists^{0 \bmod 2} y \;\; Movie(y, \text{"Sigourney Weaver"})$$

In FOC($\mathbb{P}$):

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

FO+MOD = extension of first-order logic with modulo-counting quantifiers $\exists^{i \bmod m} y\ \psi(y, \overline{z})$

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

FOC($\mathbb{P}$) = extension of first-order logic with formulas of the form $P(t_1, \ldots, t_r)$ for $P \in \mathbb{P}$ of arity r, and where each t_i is a counting term built using integers, $+$, $\cdot$, and basic counting terms $t(\overline{x})$ of the form $\#\overline{y}.\psi(\overline{x}, \overline{y})$

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 $\leqslant d$

Boolean queries:

- evaluation in linear time (Seese 1996)

Known results for the static setting (i.e., without updates)

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

Boolean queries:

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

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

and the 3-fold exponential blow-up is unavoidable assuming FPT $\neq$ AW[$*$].

Known results for the static setting (i.e., without updates)

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

Boolean queries:

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

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

  and the 3-fold exponential blow-up is unavoidable assuming FPT $\neq$ AW[$*$].

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 $\leqslant d$

Boolean queries:

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

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

  and the 3-fold exponential blow-up is unavoidable assuming FPT $\neq$ AW[$*$].

Non-Boolean queries:

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

Known results for the static setting (i.e., without updates)

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

Boolean queries:

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

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

  and the 3-fold exponential blow-up is unavoidable assuming FPT $\neq$ AW[$*$].

Non-Boolean queries:

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

Boolean queries:

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

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

  and the 3-fold exponential blow-up is unavoidable assuming FPT $\neq$ AW[$*$].

Non-Boolean queries:

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

New: Generalisation to the dynamic setting and FO+MOD [Berkholz, Keppeler, S., ICDT'17] and FOC($\mathbb{P}$) [Kuske, S., LICS'17]

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

- **Input:**
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\mathrm{FOC}(\mathbb{P})[\sigma]$
- **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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$
- **Dynamic setting:**
  Tuples may be inserted into or deleted from D

Scenario

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

- **Input:** data complexity
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\mathsf{FOC}(\mathbb{P})[\sigma]$
- **Preprocessing:** in time $O(\|D\|)$
  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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$
- **Dynamic setting:**
  Tuples may be inserted into or deleted from D

Scenario

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

- **Input:** data complexity
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\mathsf{FOC}(\mathbb{P})[\sigma]$
- **Preprocessing:** in time $O(\|D\|)$

  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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$
  - Enumerate the tuples in $\varphi(D)$
- **Dynamic setting:**

  Tuples may be inserted into or deleted from D

Scenario

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

- **Input:** data complexity
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \dots, x_k)$ in $\mathrm{FOC}(\mathbb{P})[\sigma]$
- **Preprocessing:** in time $O(\|D\|)$
  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

Scenario

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

- **Input:** data complexity
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\mathrm{FOC}(\mathbb{P})[\sigma]$
- **Preprocessing:** in time $O(\|D\|)$
  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:** data complexity
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\mathrm{FOC}(\mathbb{P})[\sigma]$
- **Preprocessing:** in time $O(\|D\|)$
  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 $\varphi(D)$ 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:** data complexity
  - Database D of degree $\leqslant d$
  - query $\varphi(x_1, \ldots, x_k)$ in $\mathsf{FOC}(\mathbb{P})[\sigma]$
- **Preprocessing:** in time $O(\|D\|)$

  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 $\varphi(D)$ 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:** combined complexity
  - Database D of degree $\leqslant d$ — $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] — 3-exp($\|\varphi\| + \lg\lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\varphi(D)$ — 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:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg\lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\varphi(D)$ with constant delay
- **Dynamic setting:** update data structure in time $f(\varphi, d)$

  Tuples may be inserted into or deleted from D

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \dots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg\lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 constant time
  - 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 $f(\varphi, d)$

  Tuples may be inserted into or deleted from D

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg \lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\overline{a}$ 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 $f(\varphi, d)$

  Tuples may be inserted into or deleted from D

Scenario

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg \lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$ 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

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp$(\|\varphi\| + \lg\lg d)$
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$ 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

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg\lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

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

  For Boolean queries:
  - Decide if $D \models \varphi$ in time $O(1)$

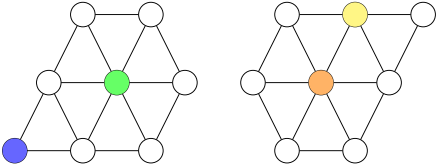
  For k-ary queries:
  - Compute the number of tuples in $\varphi(D)$ in time $O(1)$
  - Test for a given tuple $\overline{a}$ whether $\overline{a} \in \varphi(D)$ in time $O(k^2)$
  - Enumerate the tuples in $\varphi(D)$ with delay $O(k^3)$
- **Dynamic setting:** update data structure in time $f(\varphi, d)$

  Tuples may be inserted into or deleted from D

Proof method: use Hanf normal form for FO+MOD

Hanf normal form for FO+MOD

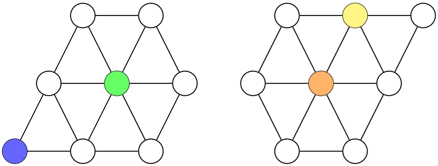
- A type τ with k centres and radius r:



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

Hanf normal form for FO+MOD

- A type τ with k centres and radius r:



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

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

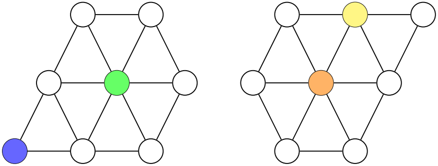
$$N_r^D(\overline{b}) \quad = \quad N_r^D(b_1) \cup \cdots \cup N_r^D(b_k)$$

where

$$N_r^D(b_i) \quad = \quad \{a \in \text{adom}(D) \ : \ \text{dist}^D(b_i, a) \leqslant r\}$$

Hanf normal form for FO+MOD

- A type τ with k centres and radius r:



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

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

$$N_r^D(\overline{b}) \quad = \quad N_r^D(b_1) \cup \cdots \cup N_r^D(b_k)$$

where

$$N_r^D(b_i) \quad = \quad \{a \in \text{adom}(D) \ : \ \text{dist}^D(b_i, a) \leqslant r\}$$

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

$$(D, \overline{a}) \models \text{sph}_\tau(\overline{x}) \quad \iff \quad (\mathcal{N}_r^D(\overline{a}), \overline{a}) \cong \tau$$

Hanf normal form for FO+MOD

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

- sphere-formulas $\mathsf{sph}_\rho(\overline{x})$ and
- Hanf-sentences $\exists^{\geqslant m} y\ \mathsf{sph}_\tau(y)$ and $\exists^{i \bmod m} y\ \mathsf{sph}_\tau(y)$ where τ 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 $\mathrm{sph}_\rho(\overline{x})$ and
- Hanf-sentences $\exists^{\geqslant m} y\ \mathrm{sph}_\tau(y)$ and $\exists^{i \bmod m} y\ \mathrm{sph}_\tau(y)$ where τ 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 $\leqslant d$.

Hanf normal form for FO+MOD

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

- sphere-formulas $\mathrm{sph}_\rho(\overline{x})$ and
- Hanf-sentences $\exists^{\geqslant m} y\ \mathrm{sph}_\tau(y)$ and $\exists^{i \bmod m} y\ \mathrm{sph}_\tau(y)$
  where τ 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 $\leqslant d$.

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

There is an algorithm which receives as input a degree bound $d \geqslant 2$ 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 $f(\varphi, d) = 3\text{-}exp(\|\varphi\| + \lg \lg d)$.

Main result for Boolean queries

Theorem

There is a dynamic algorithm that receives as input

- *a degree bound* $d \geqslant 2$,
- *a Boolean* FO+MOD[σ]*-query* φ, *and*
- *a db* D *of degree* $\leqslant d$,

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* $O(1)$.

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

Main result for Boolean queries

Theorem

There is a dynamic algorithm that receives as input

- *a degree bound* $d \geqslant 2$,
- *a Boolean* FO+MOD[σ]*-query* φ*, and*
- *a db* D *of degree* $\leqslant d$,

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* $O(1)$*.*

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

Proof Idea: *Step 1:* Transform φ into Hanf normal form ψ.

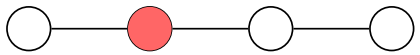
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \; \mathsf{sph}_\tau(y) \;\wedge\; \exists^{0 \bmod 2} y \; \mathsf{sph}_\rho(y)$$

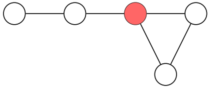
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \ \mathsf{sph}_\tau(y) \ \wedge \ \exists^{0 \bmod 2} y \ \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:



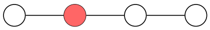
Let ρ be the type with 1 center and radius 2:



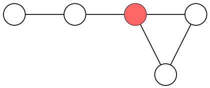
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \ \mathsf{sph}_\tau(y) \ \wedge \ \exists^{0 \bmod 2} y \ \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:



Let ρ be the type with 1 center and radius 2:

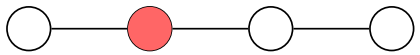


Data structure: `A[`τ`] = 0` , `A[`ρ`] = 0`

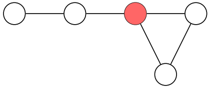
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \; \mathsf{sph}_\tau(y) \;\wedge\; \exists^{0 \bmod 2} y \; \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

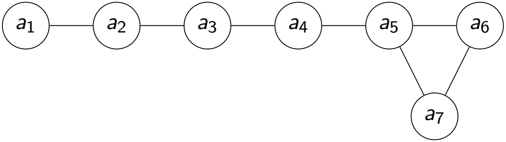


Let ρ be the type with 1 center and radius 2:



Data structure: $\mathtt{A}[\tau] = 0$, $\mathtt{A}[\rho] = 0$

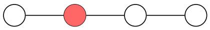
Database:



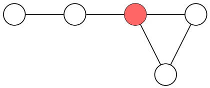
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \; \mathsf{sph}_\tau(y) \;\wedge\; \exists^{0 \bmod 2} y \; \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

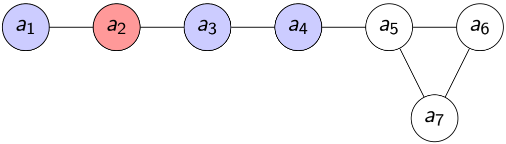


Let ρ be the type with 1 center and radius 2:



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

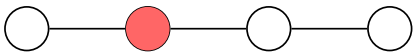
Database:



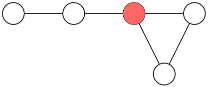
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \ \mathsf{sph}_\tau(y) \ \wedge \ \exists^{0 \bmod 2} y \ \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

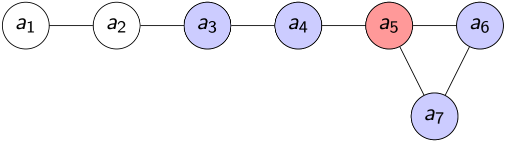


Let ρ be the type with 1 center and radius 2:



Data structure: `A[`τ`] = 1` , `A[`ρ`] = 1`

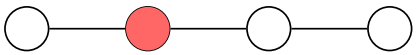
Database:



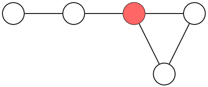
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \ \mathsf{sph}_\tau(y) \ \wedge \ \exists^{0 \bmod 2} y \ \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

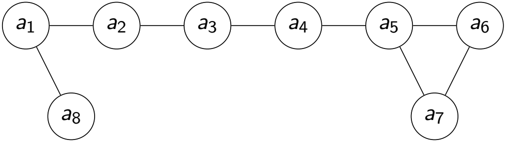


Let ρ be the type with 1 center and radius 2:



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

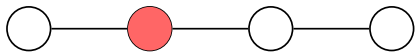
Database:



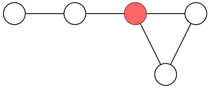
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \; \mathsf{sph}_\tau(y) \;\wedge\; \exists^{0 \bmod 2} y \; \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

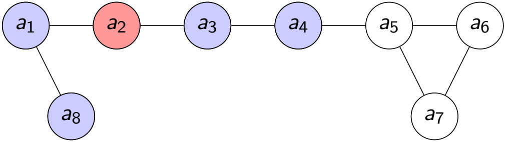


Let ρ be the type with 1 center and radius 2:



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

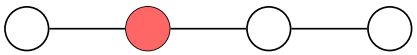
Database:



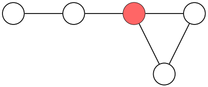
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y\ \mathsf{sph}_\tau(y)\ \wedge\ \exists^{0 \bmod 2} y\ \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

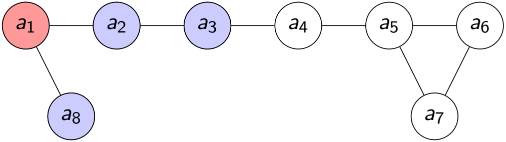


Let ρ be the type with 1 center and radius 2:



Data structure: `A[`τ`] = 1` , `A[`ρ`] = 1`

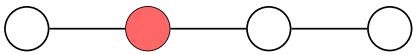
Database:



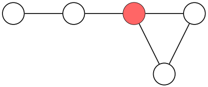
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \; \mathsf{sph}_{\tau}(y) \;\wedge\; \exists^{0 \bmod 2} y \; \mathsf{sph}_{\rho}(y)$$

Let τ be the type with 1 center and radius 2:

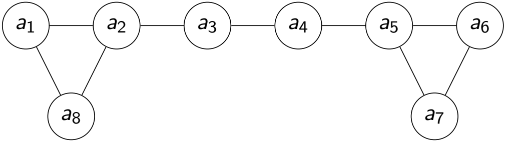


Let ρ be the type with 1 center and radius 2:



Data structure: `A[`τ`] = 1` , `A[`ρ`] = 1`

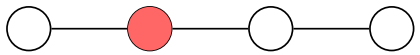
Database:



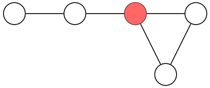
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \ \mathsf{sph}_{\tau}(y) \ \wedge \ \exists^{0 \bmod 2} y \ \mathsf{sph}_{\rho}(y)$$

Let τ be the type with 1 center and radius 2:

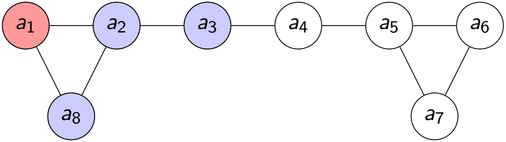


Let ρ be the type with 1 center and radius 2:



Data structure: `A[`τ`] = 0` , `A[`ρ`] = 1`

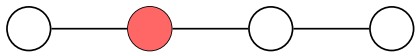
Database:



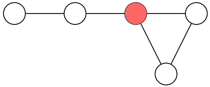
Proof idea (by example)

$$\psi \quad = \quad \exists^{0 \bmod 2} y \ \mathsf{sph}_\tau(y) \ \wedge \ \exists^{0 \bmod 2} y \ \mathsf{sph}_\rho(y)$$

Let τ be the type with 1 center and radius 2:

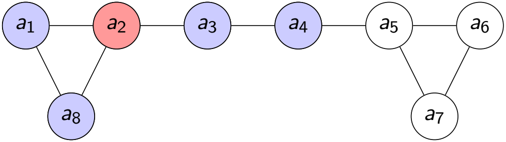


Let ρ be the type with 1 center and radius 2:



Data structure: `A[`τ`]` $= 0$, `A[`ρ`] = 2`

Database:



Main result for Boolean queries

Theorem

There is a dynamic algorithm that receives as input

- *a degree bound* $d \geqslant 2$,
- *a Boolean* $\mathrm{FO{+}MOD}[\sigma]$*-query* φ, *and*
- *a db* D *of degree* $\leqslant d$,

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* $O(1)$.

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

Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

- *a degree bound* $d \geqslant 2$,
- *a* k*-ary* FO+MOD[σ]*-query* $\varphi(\overline{x})$, *and*
- *a db* D *of degree* $\leqslant d$,

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* $O(k^3)$.

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

Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

- *a degree bound $d \geqslant 2$,*
- *a k-ary FO+MOD[σ]-query $\varphi(\overline{x})$, and*
- *a db D of degree $\leqslant d$,*

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 $O(k^3)$.

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

Proof Idea:

Proof idea: Reduction to coloured graphs

Input:

Database D

FO+MOD-query $\varphi(x_1, ..., x_k)$

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+MOD-query $\varphi(x_1, ..., x_k)$

$\Downarrow$

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

σ_k-structure $\mathcal{G}$

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

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

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 $\varphi(x_1, ..., x_k)$

$\Downarrow$

$\sigma_k := \{E, C_1, ..., C_k\}$
σ_k-structure $\mathcal{G}$

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

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

$\Uparrow$

Enumerate:
$\overline{a} \in \varphi(D)$

Same approach as in [Durand, S., Segoufin, PODS'14],
but now we have to take care of updates!

Representing Databases by Coloured Graphs

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

Representing Databases by Coloured Graphs

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

$$\mathsf{sph}_{\tau}(\overline{x}_1,\ldots,\overline{x}_c) \quad \equiv_d \quad \bigwedge_{j\in\{1,\ldots,c\}} \mathsf{sph}_{\tau_j}(\overline{x}_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg\mathsf{dist}_{\leqslant 2r+1}(\overline{x}_j,\overline{x}_{j'})$$

Representing Databases by Coloured Graphs

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

$$\mathrm{sph}_{\tau}(\overline{x}_1,\ldots,\overline{x}_c) \quad \equiv_d \quad \bigwedge_{j\in\{1,\ldots,c\}} \mathrm{sph}_{\tau_j}(\overline{x}_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg\mathrm{dist}_{\leqslant 2r+1}(\overline{x}_j,\overline{x}_{j'})$$

$$\Downarrow$$

$$\varphi_c(z_1,\ldots,z_c) \quad := \quad \bigwedge_{j\in\{1,\ldots,c\}} C_j(z_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg E(z_j,z_{j'})$$

Representing Databases by Coloured Graphs

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

$$\mathrm{sph}_{\tau}(\overline{x}_1,\ldots,\overline{x}_c) \quad \equiv_d \quad \bigwedge_{j\in\{1,\ldots,c\}} \mathrm{sph}_{\tau_j}(\overline{x}_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg\mathrm{dist}_{\leqslant 2r+1}(\overline{x}_j,\overline{x}_{j'})$$

$$\Downarrow$$

$$\varphi_c(z_1,\ldots,z_c) \quad := \quad \bigwedge_{j\in\{1,\ldots,c\}} C_j(z_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg E(z_j,z_{j'})$$

$$C_j^{\mathcal{G}_D} := \{v_{\overline{a}} : \overline{a}\in \mathrm{adom}(D)^{|x_j|}, (\mathcal{N}_r^D(\overline{a}),\overline{a})\cong \tau_j\}$$

Representing Databases by Coloured Graphs

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

$$\mathrm{sph}_{\tau}(\overline{x}_1,\dots,\overline{x}_c) \quad \equiv_d \quad \bigwedge_{j\in\{1,\dots,c\}} \mathrm{sph}_{\tau_j}(\overline{x}_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg\mathrm{dist}_{\leqslant 2r+1}(\overline{x}_j,\overline{x}_{j'})$$

$$\Downarrow$$

$$\varphi_c(z_1,\dots,z_c) \quad := \quad \bigwedge_{j\in\{1,\dots,c\}} C_j(z_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg E(z_j,z_{j'})$$

$$C_j^{\mathcal{G}_D} := \{ v_{\overline{a}} : \overline{a} \in \mathrm{adom}(D)^{|x_j|}, (\mathcal{N}_r^D(\overline{a}),\overline{a}) \cong \tau_j \}$$

$$V := \bigcup_{j\in\{1,\dots,c\}} C_j^{\mathcal{G}_D}$$

Representing Databases by Coloured Graphs

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

$$\mathrm{sph}_{\tau}(\overline{x}_1,\dots,\overline{x}_c) \quad \equiv_d \quad \bigwedge_{j\in\{1,\dots,c\}} \mathrm{sph}_{\tau_j}(\overline{x}_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg\mathrm{dist}_{\leqslant 2r+1}(\overline{x}_j,\overline{x}_{j'})$$

$$\Downarrow$$

$$\varphi_c(z_1,\dots,z_c) \quad := \quad \bigwedge_{j\in\{1,\dots,c\}} C_j(z_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg E(z_j,z_{j'})$$

$$C_j^{\mathcal{G}_D} := \{v_{\overline{a}} : \overline{a}\in \mathrm{adom}(D)^{|x_j|}, (\mathcal{N}_r^D(\overline{a}),\overline{a}) \cong \tau_j\}$$

$$V := \bigcup_{j\in\{1,\dots,c\}} C_j^{\mathcal{G}_D} \qquad E^{\mathcal{G}_D} := \{(v_{\overline{a}},v_{\overline{b}})\in V^2 : \mathrm{dist}^D(\overline{a},\overline{b}) \leqslant 2r+1\}$$

Representing Databases by Coloured Graphs

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

$$\mathrm{sph}_{\tau}(\overline{x}_1,\ldots,\overline{x}_c) \quad \equiv_d \quad \bigwedge_{j\in\{1,\ldots,c\}} \mathrm{sph}_{\tau_j}(\overline{x}_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg\mathrm{dist}_{\leqslant 2r+1}(\overline{x}_j,\overline{x}_{j'})$$

$$\Downarrow$$

$$\varphi_c(z_1,\ldots,z_c) \quad := \quad \bigwedge_{j\in\{1,\ldots,c\}} C_j(z_j) \quad \wedge \quad \bigwedge_{j\neq j'} \neg E(z_j,z_{j'})$$

$$C_j^{\mathcal{G}_D} := \{v_{\overline{a}} : \overline{a} \in \mathrm{adom}(D)^{|x_j|}, (\mathcal{N}_r^D(\overline{a}),\overline{a}) \cong \tau_j\}$$

$$V := \bigcup_{j\in\{1,\ldots,c\}} C_j^{\mathcal{G}_D} \qquad E^{\mathcal{G}_D} := \{(v_{\overline{a}},v_{\overline{b}}) \in V^2 : \mathrm{dist}^D(\overline{a},\overline{b}) \leqslant 2r+1\}$$

$$(\overline{a}_1,\ldots,\overline{a}_c) \in \mathrm{sph}_{\tau}(D) \quad \Longleftrightarrow \quad (v_{\overline{a}_1},\ldots,v_{\overline{a}_c}) \in \varphi_c(\mathcal{G}_D)$$

Updating the graph (1)

Claim

If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ by $d^{\mathcal{O}(k^2 r + k\|\sigma\|)}$ update steps and additional computing time $2^{\mathcal{O}(\|\sigma\| k^2 d^{2r+2})}$.

Updating the graph (1)

Claim

If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ 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` $R(\overline{a})$ is received

Updating the graph (1)

Claim

If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ 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` $R(\overline{a})$ 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 $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ 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` $R(\overline{a})$ is received
- Let $r' := r + (k-1)(2r+1)$
- Let $D' \in \{D_{old}, D_{new}\}$ be the database where $\overline{a}$ occurs in R.

Updating the graph (1)

Claim

If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ 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` $R(\overline{a})$ is received
- Let $r' \;:=\; r + (k-1)(2r+1)$
- Let $D' \in \{D_{old}, D_{new}\}$ be the database where $\overline{a}$ occurs in R.
- Let $U := N_{r'}^{D'}(\overline{a})$

Updating the graph (1)

Claim

If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ 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` $R(\overline{a})$ is received
- Let $r' \ := \ r + (k-1)(2r+1)$
- Let $D' \in \{D_{old}, D_{new}\}$ be the database where $\overline{a}$ occurs in R.
- Let $U := N_{r'}^{D'}(\overline{a})$
- $C_j^D := \{v_{\overline{b}} : \overline{b} \in \text{adom}(D)^{|x_j|} \text{ with } (\mathcal{N}_r^D(\overline{b}), \overline{b}) \cong \tau_j\}$

Updating the graph (1)

Claim

If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ 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` $R(\overline{a})$ is received
- Let $r' \; := \; r + (k-1)(2r+1)$
- Let $D' \in \{D_{old}, D_{new}\}$ be the database where $\overline{a}$ occurs in R.
- Let $U := N_{r'}^{D'}(\overline{a})$
- $C_j^D := \{v_{\overline{b}} : \overline{b} \in \text{adom}(D)^{|x_j|} \text{ with } (\mathcal{N}_r^D(\overline{b}), \overline{b}) \cong \tau_j\}$
- Updating the colours: Before: $C_j = C_j^{\mathcal{G}_{D_{old}}}$

1: **for** $j = 1$ **to** c **do**
2:     **for** every tuple $\overline{b} \in \bigcup_{\ell=1}^{k} U^\ell$ **do**
3:         **if** $(\mathcal{N}_r^{D_{new}}(\overline{b}), \overline{b}) \cong \tau_j$ **then** $C_j \leftarrow C_j \cup \{v_{\overline{b}}\}$
4:         **else** $C_j \leftarrow C_j \setminus \{v_{\overline{b}}\}$

Afterwards: $C_j = C_j^{\mathcal{G}_{D_{new}}}$

Updating the graph (2)

- $E^{\mathcal{G}_D} := \{(v_{\overline{a}}, v_{\overline{b}}) \in V^2 : \text{dist}^D(\overline{a}, \overline{b}) \leqslant 2r + 1\}$

Updating the graph (2)

- $E^{\mathcal{G}_D} := \{(v_{\overline{a}}, v_{\overline{b}}) \in V^2 : \mathrm{dist}^D(\overline{a}, \overline{b}) \leqslant 2r+1\}$
- Updating the edges:

Before: $E = E^{\mathcal{G}_{D_{\boldsymbol{old}}}}$

```
for every tuple b̄ ∈ ⋃_{ℓ=1}^{k} U^ℓ do
    for every tuple b̄' ∈ ⋃_{ℓ=1}^{k} U^ℓ do
        if condition (1), (2) and (3) holds then
            E ← E ∪ {(v_b̄, v_b̄')}
        else
            E ← E \ {(v_b̄, v_b̄')}
```

Afterwards: $E = E^{\mathcal{G}_{D_{\boldsymbol{new}}}}$

Updating the graph (2)

- $E^{\mathcal{G}_D} := \{(v_{\overline{a}}, v_{\overline{b}}) \in V^2 : \text{dist}^D(\overline{a}, \overline{b}) \leqslant 2r + 1\}$
- Updating the edges: Before: $E = E^{\mathcal{G}_{D_{\textit{old}}}}$

1: **for** every tuple $\overline{b} \in \bigcup_{\ell=1}^{k} U^\ell$ **do**
2:     **for** every tuple $\overline{b}' \in \bigcup_{\ell=1}^{k} U^\ell$ **do**
3:         **if** condition (1), (2) and (3) holds **then**
4:             $E \leftarrow E \cup \{(v_{\overline{b}}, v_{\overline{b}'})\}$
5:         **else**
6:             $E \leftarrow E \setminus \{(v_{\overline{b}}, v_{\overline{b}'})\}$

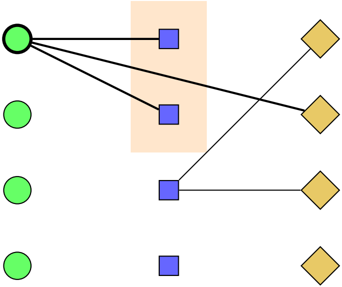
Afterwards: $E = E^{\mathcal{G}_{D_{\textit{new}}}}$

- Conditions:

(1) There is a $j \in \{1, \ldots, c\}$ such that $\overline{b} \in C_j^{\mathcal{G}}$
(2) There is a $j' \in \{1, \ldots, c\}$ such that $\overline{b}' \in C_{j'}^{\mathcal{G}}$
(3) $\text{dist}^{D_{\textit{new}}}(\overline{b}, \overline{b}') \leqslant 2r + 1$

Enumeration with delay $O(k^3 d)$

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

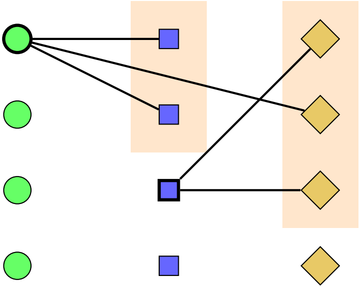


for all $u_1 \in C_1^{\mathcal{G}}$ **do**
    $\texttt{Enum}(u_1)$.
Output $\texttt{EOE}$.

function $\textsc{Enum}(u_1, \ldots, u_i)$
    if $i = k$ **then**
        Output $(u_1, \ldots, u_i)$
    else
        for all $u_{i+1} \in C_{i+1}^{\mathcal{G}}$ **do**
            if $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ **then**
                $\texttt{Enum}(u_1, \ldots, u_i, u_{i+1})$

Enumeration with delay $O(k^3 d)$

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

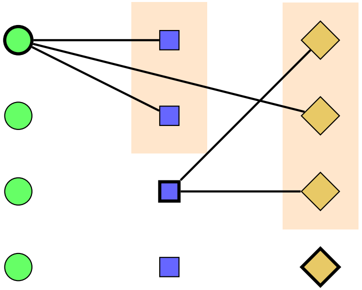


for all $u_1 \in C_1^{\mathcal{G}}$ **do**
  $\texttt{Enum}(u_1)$.
Output $\texttt{EOE}$.

function $\textsc{Enum}(u_1, \ldots, u_i)$
  if $i = k$ **then**
    Output $(u_1, \ldots, u_i)$
  else
    for all $u_{i+1} \in C_{i+1}^{\mathcal{G}}$ **do**
      if $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ **then**
        $\texttt{Enum}(u_1, \ldots, u_i, u_{i+1})$

Enumeration with delay $O(k^3 d)$

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

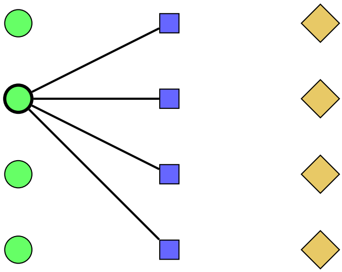


for all $u_1 \in C_1^{\mathcal{G}}$ **do**
  $\texttt{Enum}(u_1)$.
Output $\texttt{EOE}$.

function $\textsc{Enum}(u_1,\ldots,u_i)$
  if $i = k$ **then**
    Output $(u_1,\ldots,u_i)$
  else
    for all $u_{i+1} \in C_{i+1}^{\mathcal{G}}$ **do**
      if $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ **then**
        $\texttt{Enum}(u_1,\ldots,u_i,u_{i+1})$

Enumeration with delay $O(k^3 d)$

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



for all $u_1 \in C_1^{\mathcal{G}}$ **do**
    $\texttt{Enum}(u_1)$.
Output $\texttt{EOE}$.

function $\textsc{Enum}(u_1, \ldots, u_i)$
    if $i = k$ **then**
        Output $(u_1, \ldots, u_i)$
    else
        for all $u_{i+1} \in C_{i+1}^{\mathcal{G}}$ **do**
            if $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ **then**
                $\texttt{Enum}(u_1, \ldots, u_i, u_{i+1})$

Problem: Too few blue nodes

Handling small colours

A colour $\ell \in \{1, \ldots, k\}$ is small $\quad :\iff \quad \left|C_\ell^{\mathcal{G}}\right| \leqslant dk$

Handling small colours

A colour $\ell \in \{1, \ldots, k\}$ is small $\quad :\Longleftrightarrow \quad \left|C_\ell^{\mathcal{G}}\right| \leqslant dk$

W.l.o.g. let $I = \{1, \ldots, s\}$ be the set of small colours (with $s \leqslant k$).

Handling small colours

A colour $\ell \in \{1, \ldots, k\}$ is small $\quad :\Longleftrightarrow \quad \left|C_\ell^{\mathcal{G}}\right| \leqslant dk$

W.l.o.g. let $I = \{1, \ldots, s\}$ be the set of small colours (with $s \leqslant k$).

$$\mathcal{S} \quad := \quad \left\{ (u_1, \ldots, u_s) \ \in \ C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}} \ : \ \begin{array}{c} (u_j, u_{j'}) \notin E^{\mathcal{G}}, \\ \text{for all } j \neq j' \end{array} \right\}$$

The set $\mathcal{S}$ can be computed in time $O((dk)^k)$.

Handling small colours

$$\text{A colour } \ell \in \{1, \ldots, k\} \text{ is small} \quad :\Longleftrightarrow \quad \left| C_\ell^{\mathcal{G}} \right| \leqslant dk$$

W.l.o.g. let $I = \{1, \ldots, s\}$ be the set of small colours (with $s \leqslant k$).

$$\mathcal{S} \quad := \quad \left\{ (u_1, \ldots, u_s) \ \in \ C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}} \ : \ \begin{array}{c} (u_j, u_{j'}) \notin E^{\mathcal{G}}, \\ \text{for all } j \neq j' \end{array} \right\}$$

The set $\mathcal{S}$ can be computed in time $O((dk)^k)$.

$$\overline{s} \in \mathcal{S} \quad \Longleftrightarrow \quad \text{ex. } \overline{a} \text{ such that } (\overline{s}, \overline{a}) \in \varphi(D)$$

The enumeration procedure

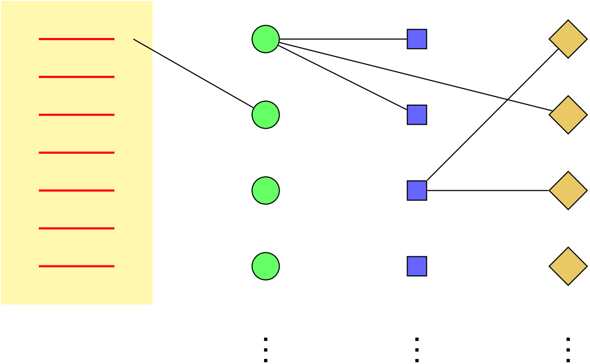
for all $(u_1, \ldots, u_s) \in \mathcal{S}$ **do**
    $\texttt{Enum}(u_1, \ldots, u_s)$.
Output the end-of-enumeration message $\texttt{EOE}$.

function $\textsc{Enum}(u_1, \ldots, u_i)$
    if $i = k$ **then**
        output the tuple $(u_1, \ldots, u_i)$
    else
        for all $u_{i+1} \in C^{\mathcal{G}}_{i+1}$ **do**
            if $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ **then**
                $\texttt{Enum}(u_1, \ldots, u_i, u_{i+1})$

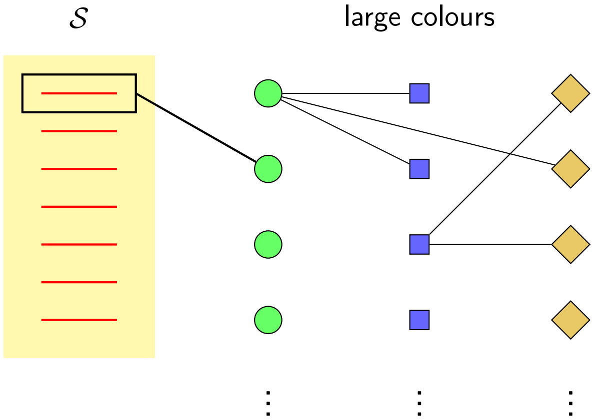
where $N^{\mathcal{G}}(u_j) \ := \ \{ v \in V^{\mathcal{G}} \ : \ (u_j, v) \in E^{\mathcal{G}} \}$.

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

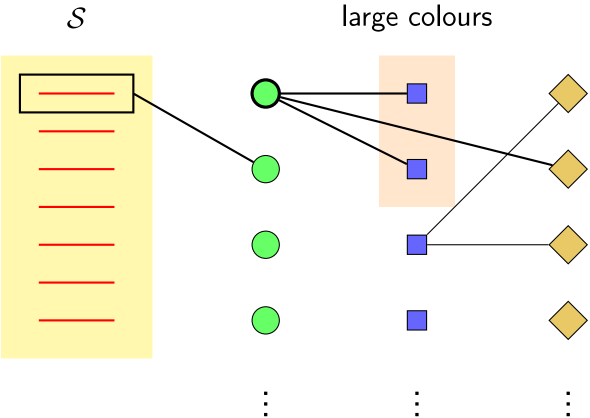
$\mathcal{S}$ large colours



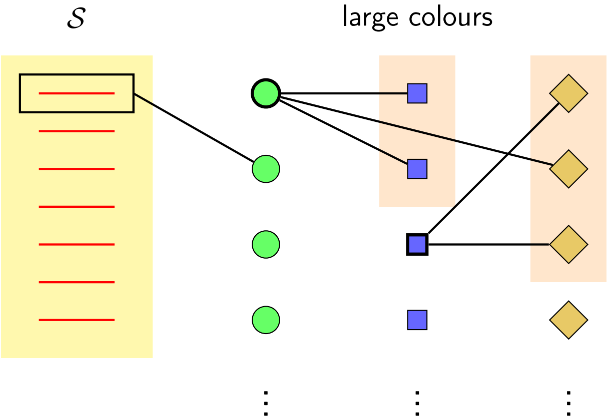
$$\psi_k(x_1,\ldots,x_k) \quad := \quad \bigwedge_{i=1}^{k} C_i(x_i) \ \wedge \ \bigwedge_{i\neq j} \neg E(x_i,x_j)$$



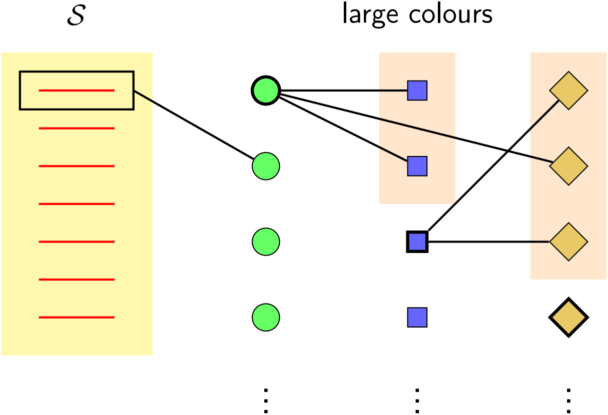
$$\psi_k(x_1,\dots,x_k) \quad := \quad \bigwedge_{i=1}^{k} C_i(x_i) \ \wedge \ \bigwedge_{i\neq j} \neg E(x_i,x_j)$$



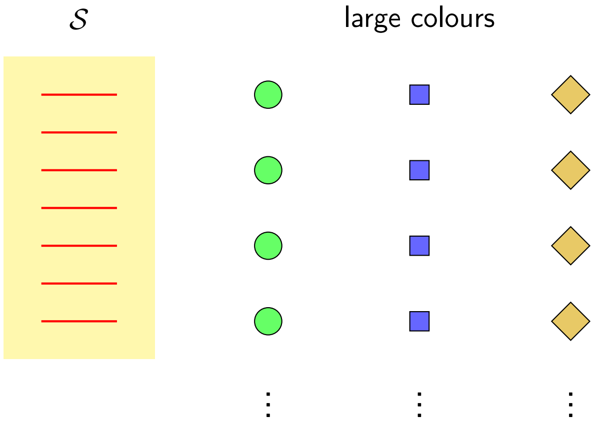
$$\psi_k(x_1,\ldots,x_k) \quad := \quad \bigwedge_{i=1}^{k} C_i(x_i) \ \wedge \ \bigwedge_{i\neq j} \neg E(x_i,x_j)$$



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



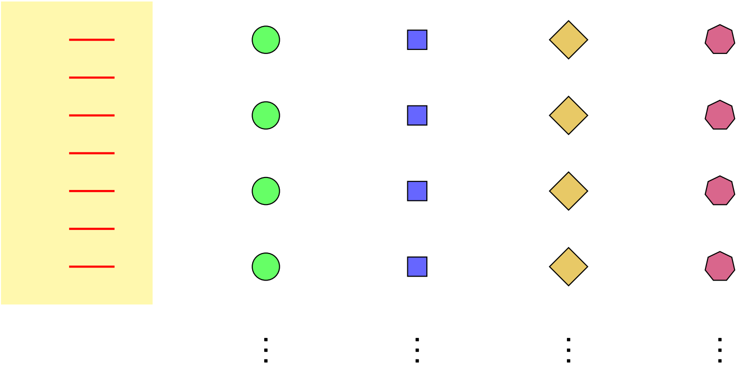
$$\psi_k(x_1,\ldots,x_k) \quad := \quad \bigwedge_{i=1}^{k} C_i(x_i) \ \wedge \ \bigwedge_{i\neq j} \neg E(x_i,x_j)$$



`update step:` Insert a node into a colour C_ℓ with $|C_\ell^{\mathcal{G}}| = dk$

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

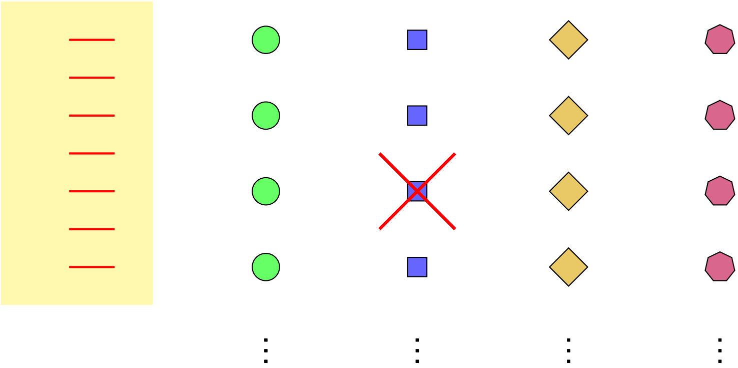
$\mathcal{S}$ large colours



`update step:` Insert a node into a colour C_ℓ with $|C_\ell^{\mathcal{G}}| = dk$

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

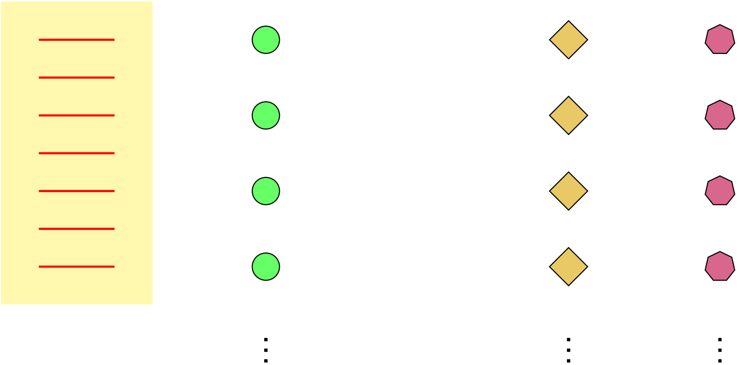
$\mathcal{S}$ large colours



update step: Delete a node from a colour C_ℓ with $|C_\ell^{\mathcal{G}}| = dk+1$

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

$\mathcal{S}$ large colours



`update step:` Delete a node from a colour C_ℓ with $|C_\ell^{\mathcal{G}}| = dk+1$

Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

- *a degree bound* $d \geqslant 2$,
- *a* k*-ary* FO+MOD[σ]*-query* $\varphi(\overline{x})$, *and*
- *a db* D *of degree* $\leqslant d$,

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 $d \geqslant 2$,*
- *a k-ary $\mathrm{FO}+\mathrm{MOD}[\sigma]$-query $\varphi(\overline{x})$, and*
- *a db D of degree $\leqslant d$,*

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 $\cancel{O(k^3)}$ $f(\varphi, d)$.

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

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.

Main result for enumeration

Theorem

There is a dynamic algorithm that receives as input

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

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* $\cancel{O(k^3)}$ $\cancel{f(\varphi, d)}$ $O(k^3)$.

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

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:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg \lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\bar{a}$ 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

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD$[\sigma]$ 3-exp$(\|\varphi\| + \lg\lg d)$
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$ 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 FOC$(\mathbb{P})$ [Kuske, S., LICS'17].

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD[σ] 3-exp($\|\varphi\| + \lg \lg d$)
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\bar{a}$ 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 FOC($\mathbb{P}$) [Kuske, S., LICS'17].

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

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

- **Input:** combined complexity
  - Database D of degree $\leqslant d$ $f(\varphi, d) =$
  - query $\varphi(x_1, \ldots, x_k)$ in FO+MOD$[\sigma]$ 3-exp$(\|\varphi\| + \lg \lg d)$
- **Preprocessing:** in time $f(\varphi, d)\|D\|$

  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 $\overline{a}$ whether $\overline{a} \in \varphi(D)$ 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 FOC$(\mathbb{P})$ [Kuske, S., LICS'17].

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

– Thank you! –