Enumeration Complexity of Query Problems and Quantifier Elimination

A. Durand[1]

[1]Université Denis Diderot
Paris - France

Complexity of constraints, Dagstuhl, october 2006

Query problems

Definition

- $\mathcal{L} =$ set of first-order σ-formulas
- $\mathcal{S} =$ set of finite σ-structures:

Query evaluation problem

EVAL$(\mathcal{L}, \mathcal{S})$

input: $\varphi \in \mathcal{L}$ and $S \in \mathcal{S}$

output: $\varphi(S) = \{\overline{a} \in D^k \mid (S, \overline{a}) \models \varphi(\overline{x})\}$

Complexity Measure

Problems have two instances: a structure $\mathcal{S}$ and a formula φ (and, often, a non-trivial output $\varphi(\mathcal{S})$).

One can express the complexity of the query problem in terms of:

- $|\mathcal{S}|$, $|\varphi|$ and $|\varphi(\mathcal{S})|$: **Combined Complexity**.
- $|\mathcal{S}|$ and $|\varphi(\mathcal{S})|$: **Data Complexity**.

Parameterized complexity: express the complexity in terms of $|\mathcal{S}|$, $|\varphi|$ and $|\varphi(\mathcal{S})|$ but consider $|\varphi|$ as a parameter.

Complexity of F.O. model-checking problem

Formula without free variables.

EVAL(FO, all)

Input: A signature σ, a σ-structure $\mathcal{S}$ of domain D and a first-order σ-formula φ of $\mathcal{L}$.
Question: Does $\mathcal{S} \models \varphi$?

- unrestricted first-order formulas.
  Combined Complexity: PSPACE $-$ *complete*.
  Data Complexity: in **P**
- existential first-order formulas.
  Combined Complexity: NP $-$ *complete*.
  Data Complexity: in **P**

In all cases : fixed parameter intractable (unless surprise)

Goals

- revisit the complexity of query evaluation by logical methods, in particular quantifier elimination.
- consider query problems as enumeration problems (answer question like : "what is the complexity to output the next tuple?")

three examples

- FO on structures of bounded degree
- Acyclic conjunctive queries (with or without $\neq$)
- FO queries on one unary function (and monadic predicates)

Query as enumeration Problem

Classical Query Evaluation

Total time to output all tuples of the result.

Enumeration problem

ENUM$(\mathcal{L}, \mathcal{S})$

input: $\varphi \in \mathcal{L}$ and $S \in \mathcal{S}$

output: an enumeration $\overline{a}_0, \ldots, \overline{a}_{m-1}$ of $\varphi(S)$

Measures of tractability

Complexity measure

$\text{delay}_i(\varphi, S)$ is the time between the completion of $\overline{a}_{i-1}$ and $\overline{a}_i$.

- $\mathcal{A}$ is *polynomial delay* if $\text{delay}_i(\varphi, S)$ is bounded by $P(|S|)$ for some polynomial P.
- It is said *linear delay* if $\text{delay}_i(\varphi, S)$ is linear in $|S|$.
- $\mathcal{A}$ is *constant delay* if $\text{delay}_i(\varphi, S)$ is bounded by some constant that depends only on φ (and neither on $|S|$ nor on i).

Enumeration algorithm and complexity classes

$\text{ENUM}(\mathcal{L}, \mathcal{S}) \in \text{CONSTANT-DELAY}_{lin}$

if it is computable by a algorithm $\mathcal{A}$ that can be decomposed into two successive phases:

$\mathcal{A}.\textbf{precomp}(\varphi, S)$ which performs a precomputation in time $O_\varphi(S)$

$\mathcal{A}.\textbf{enum}(\varphi, S)$ which enumerate $\varphi(S)$ in constant delay and constant space.

$\text{ENUM}(\mathcal{L}, \mathcal{S}) \in \text{LINEAR-DELAY}_{lin}$ if $\mathcal{A}.\textbf{enum}(\varphi, S)$ is in linear delay.

Does that make sense?

- Enumeration is a well studied field of combinatorics
- It is also well studied from a complexity perspective in graph and hypergraph theory.

Why in the context of query/constraint evaluation?

- because it refines complexity results by giving hints on the regularity of the evaluation process.
  - $\text{ENUM}(\mathcal{L}, \mathcal{S}) \in \text{CONSTANT-DELAY}_{lin} \Rightarrow \text{EVAL}(\mathcal{L}, \mathcal{S}) \in O_{|\varphi|}(S + \varphi(S))$ time
  - $\text{ENUM}(\mathcal{L}, \mathcal{S}) \in \text{LINEAR-DELAY}_{lin} \Rightarrow \text{EVAL}(\mathcal{L}, \mathcal{S}) \in O_{|\varphi|}(S \times \varphi(S))$ time
- Gives information on the possibility to restart interrupted process without re-computing the whole query.

Consider two couples $(\mathcal{L}, \mathcal{S})$ and $(\mathcal{L}', \mathcal{S}')$ where:

- $\mathcal{L}, \mathcal{L}' =$ sets of first-order formulas
- $\mathcal{S}, \mathcal{S}' =$ sets of finite structures:

Definition

EVAL$(\mathcal{L}, \mathcal{S})$ *linearly reduces to* EVAL$(\mathcal{L}', \mathcal{S}')$ if there exist two recursive functions $f : \mathcal{S} \times \mathcal{L} \to \mathcal{S}'$ and $g : \mathcal{L} \to \mathcal{L}'$ such that:

- for $(S, \varphi) \in \mathcal{S} \times \mathcal{L}$ and $(S', \varphi') = (f(S, \varphi), g(\varphi))$:

$$\varphi(S) = \varphi'(S')$$

- the function $f : (S, \varphi) \mapsto S'$ is computable in time $O_{|\varphi|}(|S|)$.

Particular case

Sometimes no need to transform the structure or (weaker) $\mathcal{S} = \mathcal{S}'$.

Eliminating variables in formulas

One use the above defined reduction in the context of variable elimination.

Basic scenario

FO_k first-order formulas with k quantified variables.

Given $S \in \mathcal{S}$ and a formula $\varphi \in \mathrm{FO}_k$, we construct $S' \in \mathcal{S}$ and a formula $\varphi' \in \mathrm{FO}_{k-1}$ such that:

$$\varphi(S) = \varphi'(S').$$

The case of relations of bounded degree

$S = \langle D; R_1, \ldots, R_q \rangle$ be relational structure, $k \in \mathbb{N}$.
S is of degree bounded by k : for each relation R_i, any x belongs to at most k tuples of the relation R_i.
BOUND-DEG$_k$: class of structures of degree bounded by k.
BOUND-DEG $= \bigcup_{k \geq 1}$ BOUND-DEG$_k$

Theorem ([Seese-96])

EVAL$(\varphi,$ BOUND-DEG$)$ *can be solved in time* $O_{|\varphi|}(|S|)$ *if* φ *is without free variable.*

Change the language

$\sigma = \{f_1, \ldots, f_q, U_1, \ldots, U_p\}$ *unary* signature
BIJ: class of unary structures all of whose functions are permutations

Bijective first order formulas

Bijective term : $f_1^{\epsilon_1} \dots f_l^{\epsilon_l}(x)$ where $l \geq 0$, x is a variable and each $f_i^{\epsilon_i} \in \{f_i, f_i^{-1}\}$

A bijective atomic formula is of one of the following forms ($\tau(x), \tau_1(x), \tau_2(x)$ are bijective terms):

- either a bijective equality $\tau_1(x) = \tau_2(y)$,
- or $\tau(x) = c$ where c is a constant symbol,
- or $U(\tau(x))$ where U is a monadic predicate,
- or a cardinality statement $\exists_x^k \Psi(x)$ with $\exists_x^k$ interpreted as *"there exist at least k values of x such that"* and $\Psi(x)$ is a Boolean combination of bijective atomic formulas over variable x only.

$\mathrm{FO}_{bij}(\mathrm{qf})$: boolean combination of bijective atomic formulas

FO_{bij}: first order over bijective atomic formulas.

Results

From relations to functions

$(\text{FO}, \text{BOUND-DEG})$ linearly reduces (in fact, equivalent) to $(\text{FO}_{bij}, \text{BIJ})$

Variable elimination [D & Grandjean-05]

Each bijective *first-order* formula is equivalent to a Boolean combination of bijective *atomic* formulas i.e. $\text{EVAL}(\text{FO}_{bij}, \text{BIJ})$ linearly reduces to $\text{EVAL}(\text{FO}_{bij}(\text{qf}), \text{BIJ})$.
When the formula has no free variable then, it is equivalent to a Boolean combination of cardinality statements.

Remark

Similar result proved in [Lindell-06]

Sketch of proof

W.l.o.g. one may suppose that formulas are of the form:

$$\varphi(\overline{x}) \equiv \exists y \ (\alpha_1 \wedge \cdots \wedge \alpha_r)$$

where each α_i is a bijective atomic formula or its negation.

Goal

Try to eliminate variable y.

Sketch of proof

$\varphi(\overline{x})$ can be put under the form:

$$\varphi(\overline{x}) \equiv \exists y\ [\psi(y) \wedge y =_{\epsilon_1} \tau_1(x_{i_1}) \wedge \cdots \wedge y =_{\epsilon_k} \tau_k(x_{i_k})]$$

with $=_\epsilon \in \{=, \neq\}$.

- if one of the terms is of the form $y = \tau_j(x_{i_j})$ then, just substitute y with $\tau_j(x_{i_j})$.
- else, the formula is of the form:

$$\varphi(\overline{x}) \equiv \exists y\ [\psi(y) \wedge \bigwedge_{j \leq k} y \neq \tau_j(x_j)]$$

Sketch of proof

consider $x_1, \ldots, x_k$ are fixed.
Let $h \leq k$ be the number of distinct values among the k terms $\tau_j(x_j)$ that satisfy $\psi(\tau_j(x_j))$.

Observation

$\varphi(\overline{x})$ is true iff the number of b s.t. $(\mathcal{S}, b) \models \psi(y)$ is bigger than h i.e. iff $\exists_y^{h+1} \psi(y)$ is true.

Then formula $\varphi(\overline{x})$ can be substituted with the equivalent formula:

$$\bigvee_{h=0}^{k} \bigvee_{P \subseteq [k], Q \subseteq P, |Q|=h} \left[\bigwedge_{j \in Q} \psi(\tau_j(x_j)) \wedge \bigwedge_{i \in P} \bigvee_{j \in Q} \tau_i(x_i) = \tau_j(x_j) \wedge \bigwedge_{j \in [k] \setminus P} \neg \psi(\tau_j(x_j)) \wedge \exists_y^{h+1} \psi(y) \right.$$

Using variable elimination to enumerate

Enumeration [D & Grandjean-05]

Problems $\text{ENUM}(\text{FO}_{bij}, \text{BIJ})$ and $\text{ENUM}(\text{FO}, \text{BOUND-DEG})$ are in $\text{CONSTANT-DELAY}_{lin}$.

Consequences:

Query problems $\text{EVAL}(\text{FO}_{bij}, \text{BIJ})$ and $\text{EVAL}(\text{FO}, \text{BOUND-DEG})$ are solvable in $O_\varphi(|S| + |\varphi(S)|)$ time and $O_\varphi(|S|)$ space.

hints on the enumeration process

Observation:

The variable elimination procedure reduces the problem to the enumeration of elements in a cartesian product of finite sets with (episodic) "holes".

Inductive evaluation of a formula of the form:

$$\Psi \equiv \Psi_1(\overline{x}) \wedge \Psi_2(y) \wedge \bigwedge_{1 \leq i \leq r} y \neq \tau_i(x_{j_i})$$

hints on the enumeration process

$\mathcal{A}_1$, algorithm for $Q_1 = \{\overline{a} \in D^k : (\mathcal{S}, \overline{a}) \models \Psi_1(\overline{x})\}$

Enumeration Algorithm

Input: $\mathcal{S}, \Psi$
Compute $Q_2 = \{b \in D : (\mathcal{S}, b) \models \Psi_2(y)\}$ and $|Q_2|$
If $|Q_2| \leq r$ **then** *particular case...*
Else
$\mathcal{A}_1$.**precomp**
For $\overline{a} \in \mathcal{A}_1$.**enum**
For $b \in Q_2$
If $(\mathcal{S}, \overline{a}, b) \not\models \bigvee_{1 \leq i \leq r} y = \tau_i(x_{j_i})$ **Output** $(\overline{a}, b)$

Summary of the bounded degree case

- Very simple argument to eliminate variable
- No need to transform the structure during the incremental elimination of variables:
  Queries defined by FO formula can be equivalently defined by a B.C. of atomic bijective formulas).
- Further results [Bagan, D., Grandjean, Olive - 06]:
  - Enumerate relatively to some lexicographic ordering,
  - produce the j^{th} element for some predefined order in constant time (but same result for lex. ordering is open)

The case of acyclic conjunctive queries

Conjunctive Queries

Build over first-order formulas using $\exists$ and $\wedge$ only.

Conjunctive Queries with inequalities

Build over first-order formulas using $\exists$, $\wedge$ and $\neq$ only.

Remarks

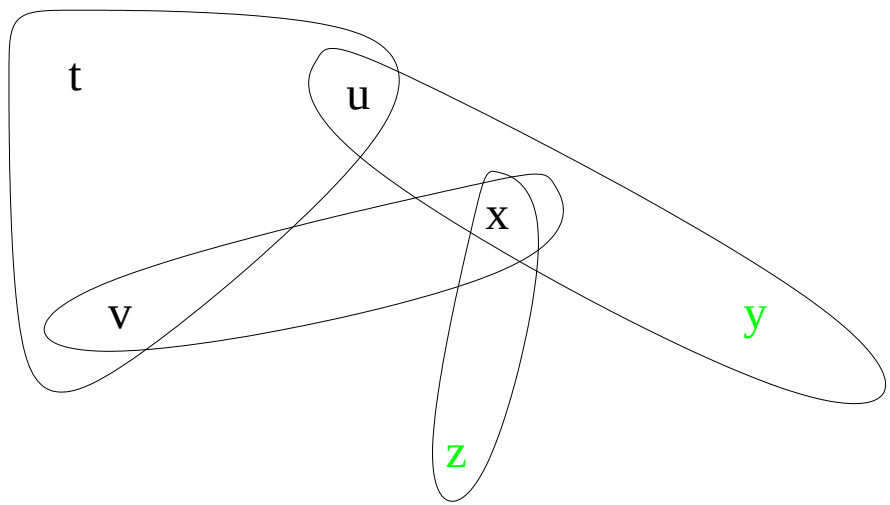
- Query evaluation is hard in general (NP-complete).
- Look for tractable subcases.

Relational Queries - Hypergraph representation

The formula:

$$\varphi(y,z) \equiv \exists t \exists u \exists v \exists x : R(t,u,v) \wedge S(v,x) \wedge T(x,z) \wedge R(u,x,y)$$

can be seen as:



Acyclic Hypergraph

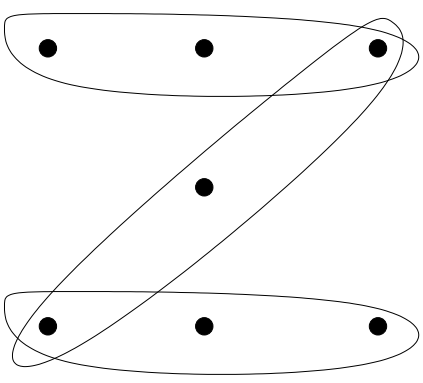
Most general notion: α-acyclicity.

An hypergraph is α-*acyclic* if the application of the following rules as long as possible outputs the empty hypergraph:

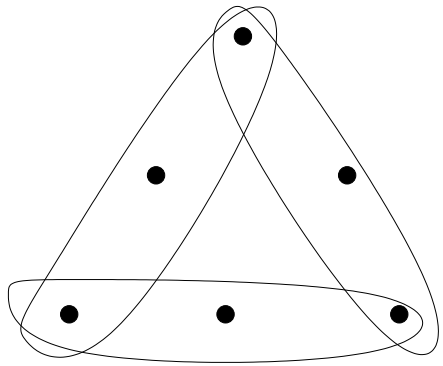
1. Remove hyperedges contained in other hyperedges;
2. Remove vertices that appear in at most one hyperedge.

A conjunctive query is acyclic if its associated hypergraph is acyclic.

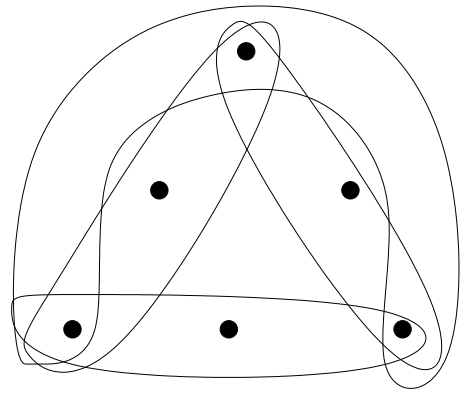
Acyclic hypergraph/query: examples



Acyclic



Cyclic



Acyclic

Large class of hypergraph/queries.

Remark: Adding an edge to a cyclic hypergraph may result in an acyclic hypergraph.

The case of acyclic conjunctive queries

ACQ Acyclic Conjunctive Queries.

$\text{ACQ}_{\neq}$ Acyclic Conjunctive Queries with inequalities (free use of $\neq$)

Known results

ACQ solvable in $O(|\varphi| \times |S| \times |\varphi(S)|)$ times [Yannakakis-81]

$\text{ACQ}_{\neq}$ solvable in $O_{|\varphi|}(|S| \times |\varphi(S)| \times \log^2(|S|))$ times [Papadimitriou & Yannakakis-99]

Goal

Try to revisits results around evaluation of acyclic queries using "quantifier elimination"-like methods.

Changing the vocabulary again

Conjunctive functional query

built over formulas of the form:

$$\varphi(x_1,\ldots,x_a) \equiv \exists y_1 \ldots \exists y_b : \bigwedge_{i=1}^{h} f_i(z_i) = g_i(t_i) \wedge \bigwedge_{i=1}^{k} U_i(v_i)$$

with $z_i, t_i, v_i \in \{x_1,\ldots,x_a,y_1,\ldots,y_b\}$ where f_i ,g_i are unary functions and U_i are monadic predicates.

Associated graph

To each conjunctive functional query φ is associated a graph $G_\varphi = (V,E)$ defined by: $V = var(\varphi)$ and for all distinct $x,y \in V$, $(x,y) \in E$ iff φ contains at least one atom of the form $f(x) = g(y)$ for some $f,g \in \sigma \cup \{Id\}$.

Acyclic functional query

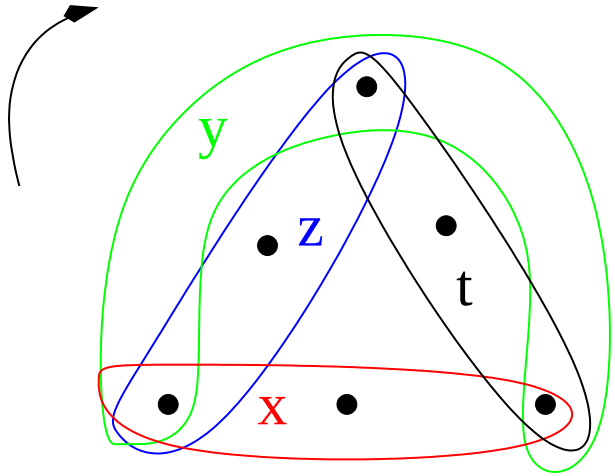
Acyclic functional query

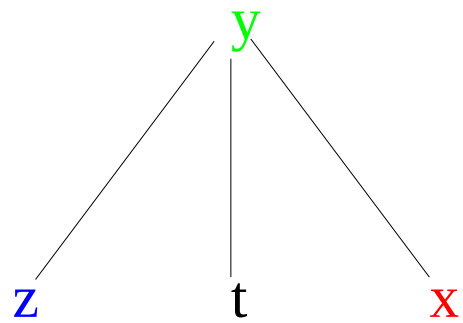
A c.f. query φ is *acyclic* if its associated graph G_φ is acyclic.
F-ACQ (resp. F-ACQ$_\neq$) class of acyclic functional queries (resp. with free use of inequalities $\neq$)

From relational to functional queries

ACQ (resp. ACQ$_\neq$) linearly reduces to F-ACQ (resp. F-ACQ$_\neq$)

Preservation of acyclicity





Hyperedges become elements of the unary functional structures, functions f_1, f_2, f_3 describe their components.

$$f_1(x) = f_1(y) \wedge f_3(x) = f_3(y)$$
$$f_1(z) = f_1(y) \wedge f_3(z) = f_2(y)$$
$$f_1(t) = f_2(y) \wedge f_3(t) = f_3(y)$$

How to evaluate queries in F-ACQ$_{\neq}$?

- Reduce this problem to some abstract covering problem
- Arguments easier to see on the "dual" language : fragment of FO built over $\forall$, $\wedge$, $\vee$, $\neq$.

We illustrate the proof on the following simple case.
Simple? only two variables, no negation, no free variable

$$\varphi \equiv \forall x \forall y : f_1(x) = g_1(y) \vee \ldots \vee f_k(x) = g_k(y).$$

We have to evaluate φ over a functional structure $\mathcal{F}$

Definition

Let $P \subseteq [k]$ and $(c_i)_{i \in P} \in D^P$. The tuple $(c_i)_{i \in P}$ is a *sample of* $\bar{f} = (f_1, \ldots, f_k)$ *over* X if

$$X \subseteq \bigcup_{i \in P} f_i^{-1}(c_i).$$

This sample is *minimal* if, moreover, for all $j \in P$:

$$X \not\subseteq \bigcup_{i \in P \setminus \{j\}} f_i^{-1}(c_i).$$

Lemma

There are at most $k!$ *minimal samples of* $\bar{f}$ *over* X *and their set can be computed in time* $O_k(|X|)$.

Evaluating φ over $\mathcal{F}$

First remark that :

$$\forall x \in D, \forall y \in D : f_1(x) = g_1(y) \vee \ldots \vee f_k(x) = g_k(y)$$

means: for each $x \in D$,

$$D = \bigcup_{i \leq k} g_i^{-1}(f_i(x)).$$

In other words, for each $x \in D$, there exists a sample $(c_i)_{i \in P}$ of $\overline{g}$ over D that "agrees" with values of $\overline{f}(x)$:

$$D = \bigcup_{i \in P} g_i^{-1}(c_i) \text{ and } \bigwedge_{i \in P} f_i(x) = c_i.$$

But: P can be chosen among the minimal ones!

Evaluating φ over $\mathcal{F}$ (continued)

Algorithm

- Compute (once!) the set of all (up to $k!$) minimal samples of $\overline{g}$ over D.
- For all $x \in D$, check if there is one minimal sample $(P, (c_i)_{i \in P})$ such that:

$$D = \bigcup_{i \in P} g_i^{-1}(c_i) \text{ and } \bigwedge_{i \in P} f_i(x) = c_i.$$

Cost: the total number of steps is $O_k(|D|)$ i.e. linear in $|D|$.

Logical views of the reduction

Variable elimination

φ logically equivalents to the one-variable formula:

$$\varphi' \equiv \forall y_1 \bigvee_{h \leq k \,!\, j \leq k} \bigwedge (P_j^h(y_1) \rightarrow f_j(y_1) = c_j^h(y_1))$$

where P_j^h always true if j belongs to the support of the h^{th} minimal sample and c_j^h always equals the j^{th} member of the h^{th} minimal sample.

NEWS: the structure has changed. Predicates P_j^h and functions c_j^h are now part of it.

$\text{ACQ}_{\neq}$ queries

The principle above can be applied inductively to acyclic functional (non necessarily boolean) queries with negation.

Theorem ([D. & Grandjean - 05])

F-$\text{ACQ}_{\neq}$ *(hence* $\text{ACQ}_{\neq}$*) can be solved in* $O_{|\varphi|}(|S| \times |\varphi(S)|)$ *times*

- The result remains true for a number of variants ([D. & Grandjean - 05], [Bagan - 06]).

What about enumeration?

Theorem

Enumeration of query results of $\mathrm{ACQ}, \mathrm{ACQ}_{\neq}$ *(and* $\mathrm{F\text{-}ACQ}, \mathrm{F\text{-}ACQ}_{\neq}$*) can be done with linear (in the size of the structure) delay.*

- Seems difficult to obtain a better (i.e. constant delay) result except in particular cases.

The case of one unary function

quasi-unary formulas

formulas over a signature containing *one* unary function and monadic predicates.
QU: class of quasi unary strucutres.

Theorem (D. & Olive-06)

- FO *linearly reduces to* FO(*qf*) *on quasi-unary structures.*
- ENUM(FO, QU) $\in$ CONSTANT-DELAY$_{lin}$

Remark: ENUM(MSO, BOUNDEDTW) $\in$ LINEAR-DELAY$_{lin}$
[Courcelle-06], [Bagan-06], [Frick-??]

Conclusion

- Give several examples of "variable elimination" methods for complexity of query evaluation (revisit old results and prove new ones)
- Derives (sometimes) information on the enumeration complexity of the problem.

Questions

- limits of the approach ?
- Work on reducing the size of constants
- Separates (modulo hypothesis) complexity classes for enumeration pbs
- importance of the order in enumeration algorithm ?
- importance of memory space ?
- ....