# Enumeration: logic, algebraic and geometric methods

Yann Strozecki

Université Paris Sud - Paris $11$
Equipe ALGO

Février $2011$, séminaire MC2 (LIP)

# Enumeration problems

Polynomially balanced predicate $A(x, y)$, decidable in polynomial time.

- $\exists ? y A(x, y)$ : decision problem (class $\mathrm{NP}$)
- $\sharp\{y \mid A(x, y)\}$ : counting problem (class $\sharp\mathrm{P}$)
- $\{y \mid A(x, y)\}$ : enumeration problem (class $\mathrm{EnumP}$)

Example

**Perfect matching:**

- The decision problem is to decide if there is a perfect matching.
- The counting problem is to count the number of perfect matchings.
- The enumeration problem is to list every perfect matching.

# Enumeration problems

Polynomially balanced predicate $A(x,y)$, decidable in polynomial time.

- $\exists ? y A(x,y)$ : decision problem (class $\mathrm{NP}$)
- $\sharp\{y \mid A(x,y)\}$ : counting problem (class $\sharp\mathrm{P}$)
- $\{y \mid A(x,y)\}$ : enumeration problem (class $\mathrm{EnumP}$)

Example

**Perfect matching:**

- The decision problem is to decide if there is a perfect matching.
- The counting problem is to count the number of perfect matchings.
- The enumeration problem is to list every perfect matching.

# Time complexity measures for enumeration

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay
   - incremental polynomial time: $\mathbf{IncP}$ (Circuits of a matroid)
   - polynomial delay: $\mathbf{DelayP}$ (Perfect Matching [Uno])
   - Constant or linear delay
     - A two steps algorithm: preprocessing + generation
     - An ad-hoc RAM model.

# Time complexity measures for enumeration

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay
   - incremental polynomial time: $\mathbf{IncP}$ (Circuits of a matroid)
   - polynomial delay: $\mathbf{DelayP}$ (Perfect Matching [Uno])
   - Constant or linear delay
     - A two steps algorithm: preprocessing + generation
     - An ad-hoc RAM model.

# Enumeration problems

$R$: polynomially balanced binary predicate

$\textsc{Enum}{\cdot}R$

*Input:* $x \in \mathcal{I}$

*Output:* an enumeration of elements in $R(x) = \{y \mid R(x,y)\}$

**Definition**

The problem $\textsc{Enum}{\cdot}R$ belongs to the class $\textsc{Delay}(g,f)$ if there exists an enumeration algorithm that computes $\textsc{Enum}{\cdot}R$ such that, for all input $x$:

- Preprocessing in time $O(g(|x|))$,
- Solutions $y \in R(x)$ are computed successively without repetition with a delay $O(f(|x|))$

$\textsc{Constant-Delay} = \bigcup_k \textsc{Delay}(n^k, 1)$.

# Enumeration complexity classes

**Separation:**

$\mathbf{QueryP} \subsetneq \mathbf{SDelayP} \subseteq \mathbf{DelayP} \subseteq \mathbf{IncP} \subsetneq \mathbf{TotalP} \subsetneq \mathbf{EnumP}.$

# Enumeration complexity classes

**Separation:**

$$\mathbf{QueryP} \subsetneq \mathbf{SDelayP} \subseteq \mathbf{DelayP} \subseteq \mathbf{IncP} \subsetneq \mathbf{TotalP} \subsetneq \mathbf{EnumP}.$$

**Complete problem:**

# Enumeration complexity classes

**Separation:**

$$\mathbf{QueryP} \subsetneq \mathbf{SDelayP} \subseteq \mathbf{DelayP} \subseteq \mathbf{IncP} \subsetneq \mathbf{TotalP} \subsetneq \mathbf{EnumP}.$$

**Complete problem:**

No good notion of reduction out of parsimonious reduction.

# Enumeration complexity classes

**Separation:**

$$\mathbf{QueryP} \subsetneq \mathbf{SDelayP} \subseteq \mathbf{DelayP} \subseteq \mathbf{IncP} \subsetneq \mathbf{TotalP} \subsetneq \mathbf{EnumP}.$$

**Complete problem:**

No good notion of reduction out of parsimonious reduction.

# Arithmetization

Representing a problem by a polynomial:

1. Deciding if there is a perfect matching in randomized parallel logarithmic time [MVV1987].
2. IP = PSPACE [S 1992]

# Arithmetization

Representing a problem by a polynomial:

1. Deciding if there is a perfect matching in randomized parallel logarithmic time [MVV1987].
2. $\mathrm{IP} = \mathrm{PSPACE}$ [S 1992]
3. Polynomial time algorithm to decide wether a number is prime [AKS 2004].

# Arithmetization

Representing a problem by a polynomial:

1. Deciding if there is a perfect matching in randomized parallel logarithmic time [MVV1987].
2. $\mathrm{IP} = \mathrm{PSPACE}$ [S 1992]
3. Polynomial time algorithm to decide wether a number is prime [AKS 2004].
4. Better parametized algorithms for packing and path problems [K 2008].

# Arithmetization

Representing a problem by a polynomial:

1. Deciding if there is a perfect matching in randomized parallel logarithmic time [MVV1987].
2. $\mathrm{IP} = \mathrm{PSPACE}$ [S 1992]
3. Polynomial time algorithm to decide wether a number is prime [AKS 2004].
4. Better parametized algorithms for packing and path problems [K 2008].
5. Enumeration algorithms [S 2010].

# Arithmetization

Representing a problem by a polynomial:

1. Deciding if there is a perfect matching in randomized parallel logarithmic time [MVV1987].
2. $\mathrm{IP} = \mathrm{PSPACE}$ [S 1992]
3. Polynomial time algorithm to decide wether a number is prime [AKS 2004].
4. Better parametized algorithms for packing and path problems [K 2008].
5. Enumeration algorithms [S 2010].

# Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix : cycle covers of a graph.

# Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix : cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.

# Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix : cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph. Randomized algorithm to find the size of a maximal acyclic subhypergraph.

# Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix : cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub $2002$]: spanning hypertrees of a $3$-uniform hypergraph. Randomized algorithm to find the size of a maximal acyclic subhypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.

# Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix : cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub $2002$]: spanning hypertrees of a $3$-uniform hypergraph. Randomized algorithm to find the size of a maximal acyclic subhypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.

Only multilinear polynomials.

# Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix : cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub $2002$]: spanning hypertrees of a $3$-uniform hypergraph. Randomized algorithm to find the size of a maximal acyclic subhypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.

Only multilinear polynomials.

# Of polynomials and automa

Probabilistic Automaton $A = (n, \Sigma, M, \alpha, \eta)$.

A word $w = \sigma_1 \sigma_2 \ldots \sigma_k$ has a probability $A(w) = \alpha(\prod_{i=1}^{k} M(\sigma_i))\eta$.

# Of polynomials and automa

Probabilistic Automaton $A = (n, \Sigma, M, \alpha, \eta)$.

A word $w = \sigma_1 \sigma_2 \dots \sigma_k$ has a probability $A(w) = \alpha(\prod_{i=1}^{k} M(\sigma_i))\eta$.

To an automaton, we associate $\sum_{w \in \Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \dots X_{n,\sigma_n}$

# Of polynomials and automa

Probabilistic Automaton $A = (n, \Sigma, M, \alpha, \eta)$.

A word $w = \sigma_1 \sigma_2 \ldots \sigma_k$ has a probability $A(w) = \alpha(\prod_{i=1}^{k} M(\sigma_i))\eta$.

To an automaton, we associate $\sum_{w \in \Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \ldots X_{n,\sigma_n}$

Alternate form: $\alpha(\prod_{i=1}^{n} \sum_{\sigma \in \Sigma} M(\sigma) X_{i,\sigma})\eta$.

# Of polynomials and automa

Probabilistic Automaton $A = (n, \Sigma, M, \alpha, \eta)$.

A word $w = \sigma_1\sigma_2\ldots\sigma_k$ has a probability $A(w) = \alpha(\prod_{i=1}^{k} M(\sigma_i))\eta$.

To an automaton, we associate $\sum_{w\in\Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \ldots X_{n,\sigma_n}$

Alternate form: $\alpha(\prod_{i=1}^{n} \sum_{\sigma\in\Sigma} M(\sigma) X_{i,\sigma})\eta$.

**Theorem**

*Randomized algorithm to test if two automata have the same language and to produce a witness.*

# Of polynomials and automa

Probabilistic Automaton $A = (n, \Sigma, M, \alpha, \eta)$.

A word $w = \sigma_1\sigma_2 \dots \sigma_k$ has a probability $A(w) = \alpha(\prod_{i=1}^{k} M(\sigma_i))\eta$.

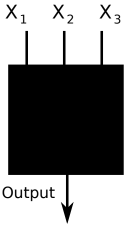
To an automaton, we associate $\sum_{w \in \Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \dots X_{n,\sigma_n}$

Alternate form: $\alpha(\prod_{i=1}^{n} \sum_{\sigma \in \Sigma} M(\sigma) X_{i,\sigma})\eta$.

**Theorem**

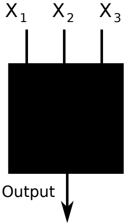
*Randomized algorithm to test if two automata have the same language and to produce a witness.*

# Polynomial given by a black-box




$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

# Polynomial given by a black-box



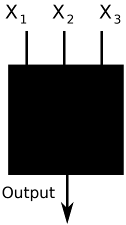

$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

$$X_1 = 1,\ X_2 = 2,\ X_3 = 1$$

$$1 * 2 + 1 * 1 + 2 + 1$$

$$Output = 6$$

# Polynomial given by a black-box



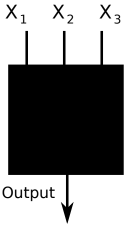

$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

$$X_1 = -1,\ X_2 = 1,\ X_3 = 2$$

$$-1 * 1 + -1 * 2 + 1 + 2$$

$$Output = 0$$

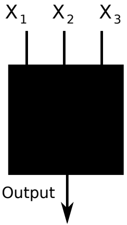
# Polynomial given by a black-box




$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

- Problem: interpolation, compute $P$ from its values.
- Complexity: time and number of calls to the oracle.

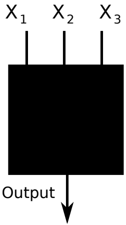
# Polynomial given by a black-box




$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

- Problem: interpolation, compute $P$ from its values.
- Complexity: time and number of calls to the oracle.
- Parameters: number of variables and total degree.

# Polynomial given by a black-box

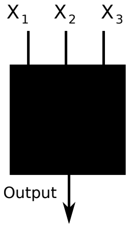



$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

- Problem: interpolation, compute $P$ from its values.
- Complexity: time and number of calls to the oracle.
- Parameters: number of variables and total degree.

**Enumeration problem:** output the monomials one after the other.

# Polynomial given by a black-box




$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

- Problem: interpolation, compute $P$ from its values.
- Complexity: time and number of calls to the oracle.
- Parameters: number of variables and total degree.

**Enumeration problem:** output the monomials one after the other.

# The decision problem

POLYNOMIAL IDENTITY TESTING
*Input:* a polynomial given as a black box.
*Output:* decides if the polynomial is zero.

**Lemma (Schwarz-Zippel)**

*Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.*

# The decision problem

POLYNOMIAL IDENTITY TESTING
*Input:* a polynomial given as a black box.
*Output:* decides if the polynomial is zero.

**Lemma (Schwarz-Zippel)**

*Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.*

No way to make PIT deterministic for black box.

# The decision problem

POLYNOMIAL IDENTITY TESTING
*Input:* a polynomial given as a black box.
*Output:* decides if the polynomial is zero.

**Lemma (Schwarz-Zippel)**

*Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.*

No way to make PIT deterministic for black box.

Error **exponentially small** in the size of the integers!

# The decision problem

POLYNOMIAL IDENTITY TESTING
*Input:* a polynomial given as a black box.
*Output:* decides if the polynomial is zero.

**Lemma (Schwarz-Zippel)**

*Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.*

No way to make PIT deterministic for black box.

Error **exponentially small** in the size of the integers!

# Existing interpolation methods

Sparse interpolation = polynomial total time:

- Ben Or and Tiwari ($1988$): evaluation on big power of prime numbers
- Zippel ($1990$): use a dense interpolation on a polynomial with a restricted number of variables

# Existing interpolation methods

Sparse interpolation = polynomial total time:

- Ben Or and Tiwari ($1988$): evaluation on big power of prime numbers
- Zippel ($1990$): use a dense interpolation on a polynomial with a restricted number of variables
- Klivans and Spielman ($2001$): transformation of a multivariate into an univariate one.

# Existing interpolation methods

Sparse interpolation = polynomial total time:

- Ben Or and Tiwari $(1988)$: evaluation on big power of prime numbers
- Zippel $(1990)$: use a dense interpolation on a polynomial with a restricted number of variables
- Klivans and Spielman $(2001)$: transformation of a multivariate into an univariate one.
- Garg and Schost $(2009)$: non black-box but complexity independent from the degree of the polynomial

# Existing interpolation methods

Sparse interpolation = polynomial total time:

- Ben Or and Tiwari $(1988)$: evaluation on big power of prime numbers
- Zippel $(1990)$: use a dense interpolation on a polynomial with a restricted number of variables
- Klivans and Spielman $(2001)$: transformation of a multivariate into an univariate one.
- Garg and Schost $(2009)$: non black-box but complexity independent from the degree of the polynomial

Enumeration complexity: produce the monomials one at a time with a good **delay**.

# Existing interpolation methods

Sparse interpolation = polynomial total time:

- Ben Or and Tiwari ($1988$): evaluation on big power of prime numbers
- Zippel ($1990$): use a dense interpolation on a polynomial with a restricted number of variables
- Klivans and Spielman ($2001$): transformation of a multivariate into an univariate one.
- Garg and Schost ($2009$): non black-box but complexity independent from the degree of the polynomial

Enumeration complexity: produce the monomials one at a time with a good **delay**.

# From finding a monomial to interpolation

Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Substract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

# From finding a monomial to interpolation

Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Substract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

**Drawback:** one has to store the polynomial $Q =$ the sum of the generated monomials.
When there is a call, compute $P - Q$.

# From finding a monomial to interpolation

Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Substract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

**Drawback:** one has to store the polynomial $Q =$ the sum of the generated monomials.
When there is a call, compute $P - Q$.

Incremental delay.

# From finding a monomial to interpolation

Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Substract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

**Drawback:** one has to store the polynomial $Q =$ the sum of the generated monomials.
When there is a call, compute $P - Q$.

Incremental delay.

# Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7 D^4)$ calls, $n$ number of variables and $D$ the total degree

# Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7 D^4)$ calls, $n$ number of variables and $D$ the total degree
- **Question:** is it possible to decrease the number of calls to a more manageable polynomial.

# Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7 D^4)$ calls, $n$ number of variables and $D$ the total degree
- **Question:** is it possible to decrease the number of calls to a more manageable polynomial.
- Yes for polynomial of fixed degree $d$. One can find the "highest" degree polynomial with $O(n^2 D^{d-1})$ calls.

# Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7 D^4)$ calls, $n$ number of variables and $D$ the total degree
- **Question:** is it possible to decrease the number of calls to a more manageable polynomial.
- Yes for polynomial of fixed degree $d$. One can find the "highest" degree polynomial with $O(n^2 D^{d-1})$ calls.
- Yes for polynomial whose each two monomials have distinct supports: $O(n^2)$ calls.

# Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7D^4)$ calls, $n$ number of variables and $D$ the total degree
- **Question:** is it possible to decrease the number of calls to a more manageable polynomial.
- Yes for polynomial of fixed degree $d$. One can find the "highest" degree polynomial with $O(n^2D^{d-1})$ calls.
- Yes for polynomial whose each two monomials have distinct supports: $O(n^2)$ calls.

# Improving the delay

How to achieve a polynomial delay ?

Partial-Monomial
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

# Improving the delay

How to achieve a polynomial delay ?

PARTIAL-MONOMIAL
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

**Theorem**

*There is a randomized algorithm which solves* PARTIAL-MONOMIAL *over* ***multilinear*** *polynomials in time polynomial in $n$ the number of variables and $\log(\epsilon^{-1})$ the error bound.*

# Improving the delay

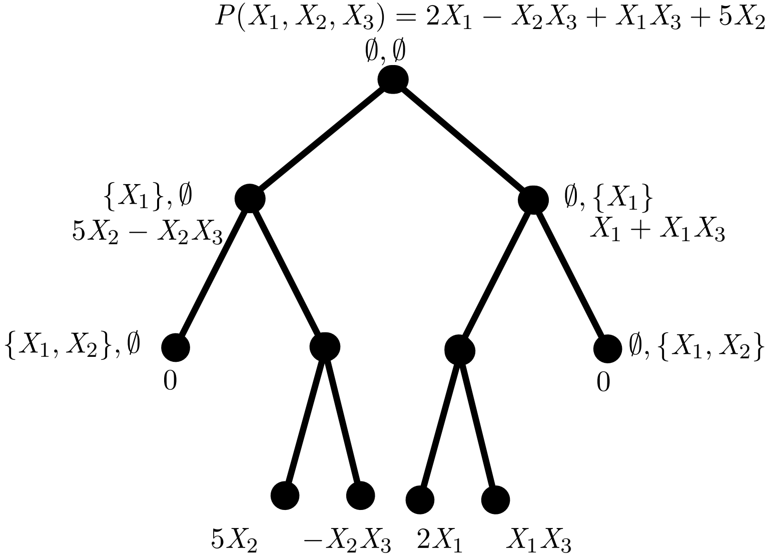
How to achieve a polynomial delay ?

PARTIAL-MONOMIAL
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

**Theorem**

*There is a randomized algorithm which solves* PARTIAL-MONOMIAL *over **multilinear** polynomials in time polynomial in $n$ the number of variables and $\log(\epsilon^{-1})$ the error bound.*

# Depth-first traversal of the monomial tree



# Polynomial delay algorithm

**Theorem**

*Let $P$ be a multilinear polynomial with $n$ variables. There is an algorithm which computes the set of monomials of $P$ with probability $1 - \epsilon$ and a delay **polynomial** in $n$ and $\log(\epsilon)^{-1}$.*

- The algorithm can be parallelized.

# Polynomial delay algorithm

**Theorem**

*Let $P$ be a multilinear polynomial with $n$ variables. There is an algorithm which computes the set of monomials of $P$ with probability $1-\epsilon$ and a delay* ***polynomial*** *in $n$ and $\log(\epsilon)^{-1}$.*

- The algorithm can be parallelized.
- It works on finite fields of small characteristic (can be used to speed up computation).

# Polynomial delay algorithm

**Theorem**

*Let $P$ be a multilinear polynomial with $n$ variables. There is an algorithm which computes the set of monomials of $P$ with probability $1-\epsilon$ and a delay* ***polynomial*** *in $n$ and $\log(\epsilon)^{-1}$.*

- The algorithm can be parallelized.
- It works on finite fields of small characteristic (can be used to speed up computation).
- On classes of polynomials given by circuits on which PIT can be derandomized, this algorithm also can be derandomized. STOC 2011, Saraf, Volkovich: deterministic identity testing of depth-$4$ multilinear circuits with bounded top fan-in

# Polynomial delay algorithm

**Theorem**

*Let $P$ be a multilinear polynomial with $n$ variables. There is an algorithm which computes the set of monomials of $P$ with probability $1-\epsilon$ and a delay* ***polynomial*** *in $n$ and $\log(\epsilon)^{-1}$.*

- The algorithm can be parallelized.
- It works on finite fields of small characteristic (can be used to speed up computation).
- On classes of polynomials given by circuits on which PIT can be derandomized, this algorithm also can be derandomized. STOC 2011, Saraf, Volkovich: deterministic identity testing of depth-$4$ multilinear circuits with bounded top fan-in

# Comparison to other algorithms

| | Ben-Or Tiwari | Zippel | KS | My Algorithm |
|---|---|---|---|---|
| Algorithm type | Deterministic | Probabilistic | Probabilistic | Probabilistic |
| Number of calls | $2T$ | $tnD$ | $tn^7D^4$ | $tnD(n + \log(\epsilon^{-1}))$ |
| Total time | Quadratic in $T$ | Quadratic in $t$ | Quadratic in $t$ | Linear in $t$ |
| Enumeration | Exponential | **TotalPP** | **IncPP** | **DelayPP** |
| Size of points | $T\log(n)$ | $\log(nT^2\epsilon^{-1})$ | $\log(nD\epsilon^{-1})$ | $\log(D)$ |

Figure: Comparison of interpolation algorithms on multilinear polynomials

Good total time and best delay, but only on multilinear polynomials.

# Comparison to other algorithms

| | Ben-Or Tiwari | Zippel | KS | My Algorithm |
|---|---|---|---|---|
| Algorithm type | Deterministic | Probabilistic | Probabilistic | Probabilistic |
| Number of calls | $2T$ | $tnD$ | $tn^7D^4$ | $tnD(n + \log(\epsilon^{-1}))$ |
| Total time | Quadratic in $T$ | Quadratic in $t$ | Quadratic in $t$ | Linear in $t$ |
| Enumeration | Exponential | **TotalPP** | **IncPP** | **DelayPP** |
| Size of points | $T \log(n)$ | $\log(nT^2\epsilon^{-1})$ | $\log(nD\epsilon^{-1})$ | $\log(D)$ |

Figure: Comparison of interpolation algorithms on multilinear polynomials

Good total time and best delay, but only on multilinear polynomials.

# Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

Partial-Monomial
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

# Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

PARTIAL-MONOMIAL
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

The polynomial delay algorithm works by repeatedly solving this problem.

# Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

Partial-Monomial
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

The polynomial delay algorithm works by repeatedly solving this problem.

**Proposition**

*The problem* Partial-Monomial *restricted to degree* $2$ *polynomials is* $\mathrm{NP}$*-hard.*

# Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

PARTIAL-MONOMIAL
*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

The polynomial delay algorithm works by repeatedly solving this problem.

**Proposition**

*The problem* PARTIAL-MONOMIAL *restricted to degree* $2$ *polynomials is* NP*-hard.*

Introduction to Enumeration

Enumeration and polynomials

Enumeration and logic

Enumeration and polytopes

# Logic in half a slide

**First order logic(FO):**

- Variables: $x, y, z \ldots$
- The language $\sigma$, relations and functions: $R(x, y)$, $f(z)$
- Unary and binary connectors: $\wedge$, $\vee$, $\neg$
- Quantifiers: $\forall$, $\exists$
- $\varphi \equiv \forall x \exists y E(x, y) \vee E(y, x)$

# Logic in half a slide

**First order logic(FO):**

- Variables: $x, y, z \ldots$
- The language $\sigma$, relations and functions: $R(x, y)$, $f(z)$
- Unary and binary connectors: $\wedge$, $\vee$, $\neg$
- Quantifiers: $\forall$, $\exists$
- $\varphi \equiv \forall x \exists y E(x, y) \vee E(y, x)$

**Theorem (Goldberg)**

*For almost all first order graph property $\varphi$, the graphs of size $n$ which satisfies $\varphi$ can be enumerated with polynomial delay in $n$.*

# Logic in half a slide

**First order logic(FO):**

- Variables: $x, y, z \dots$
- The language $\sigma$, relations and functions: $R(x, y)$, $f(z)$
- Unary and binary connectors: $\wedge$, $\vee$, $\neg$
- Quantifiers: $\forall$, $\exists$
- $\varphi \equiv \forall x \exists y E(x, y) \vee E(y, x)$

**Theorem (Goldberg)**

*For almost all first order graph property $\varphi$, the graphs of size $n$ which satisfies $\varphi$ can be enumerated with polynomial delay in $n$.*

# Enumeration problem defined by a formula

**Second order logic (SO):**
Second order variable: $\mathbf{T}$, denotes unknown relation over the domain.

Let $\Phi(\mathbf{z}, \mathbf{T})$ be a first order formula with free first and second order variables.

# Enumeration problem defined by a formula

**Second order logic (SO):**
Second order variable: $\mathbf{T}$, denotes unknown relation over the domain.

Let $\Phi(\mathbf{z}, \mathbf{T})$ be a first order formula with free first and second order variables.

Enum$\cdot\Phi$
*Input:* A $\sigma$-structure $\mathcal{S}$
*Output:* $\Phi(\mathcal{S}) = \{(\mathbf{z}^*, \mathbf{T}^*) : (\mathcal{S}, \mathbf{z}^*, \mathbf{T}^*) \models \Phi(\mathbf{z}, \mathbf{T})\}$

Let $\mathscr{F}$ be a subclass of first order formulas. We denote by Enum$\cdot\mathscr{F}$ the collection of problems Enum$\cdot\Phi$ for $\Phi \in \mathscr{F}$.

# Enumeration problem defined by a formula

**Second order logic (SO):**
Second order variable: $\mathbf{T}$, denotes unknown relation over the domain.

Let $\Phi(\mathbf{z}, \mathbf{T})$ be a first order formula with free first and second order variables.

$\textsc{Enum}{\cdot}\Phi$
*Input:* A $\sigma$-structure $\mathcal{S}$
*Output:* $\Phi(\mathcal{S}) = \{(\mathbf{z}^*, \mathbf{T}^*) : (\mathcal{S}, \mathbf{z}^*, \mathbf{T}^*) \models \Phi(\mathbf{z}, \mathbf{T})\}$

Let $\mathscr{F}$ be a subclass of first order formulas. We denote by $\textsc{Enum}{\cdot}\mathscr{F}$ the collection of problems $\textsc{Enum}{\cdot}\Phi$ for $\Phi \in \mathscr{F}$.

# Example

## Example

Independent sets:

$$IS(T) \equiv \forall x \forall y \;\; T(x) \wedge T(y) \Rightarrow \neg E(x, y).$$

## Example

Hitting sets (vertex covers) of a hypergraph represented by the incidence structure $\langle D, \{ V, E, R \} \rangle$.

$$HS(T) \equiv \forall x \, ( T(x) \Rightarrow \; V(x)) \wedge \forall y \exists x \, E(y) \Rightarrow ( T(x) \wedge R(x, y))$$

# First-order queries with free second order variables

**This presentation**

- **FO** queries with free second-order variables
- Data complexity: the query is fixed
- The complexity in term of the size of the input structure's domain
- Quantifier depth as a parameter: $\textsc{Enum}\cdot\Sigma_1$
- $\textsc{Enum}\cdot\mathrm{IS} \in \textsc{Enum}\cdot\Pi_1$ and $\textsc{Enum}\cdot\mathrm{HS} \in \textsc{Enum}\cdot\Pi_2$

# First-order queries with free second order variables

## This presentation

- **FO** queries with free second-order variables
- Data complexity: the query is fixed
- The complexity in term of the size of the input structure's domain
- Quantifier depth as a parameter: $\textsc{Enum}{\cdot}\Sigma_1$
- $\textsc{Enum}{\cdot}\mathrm{IS} \in \textsc{Enum}{\cdot}\Pi_1$ and $\textsc{Enum}{\cdot}\mathrm{HS} \in \textsc{Enum}{\cdot}\Pi_2$

# Previous results

1. Only first-order free variables and bounded degree structures. Durand-Grandjean'07, Lindell'08, Kazana-Segoufin'10: linear preprocessing + constant delay.
2. Only first-order free variables and acyclic conjunctive formula. Bagan-Durand-Grandjean'07: linear preprocessing + linear delay

**Example**

Enumeration of the $k$-cliques of a graph of bounded degree.

# Previous results

1. Only first-order free variables and bounded degree structures. Durand-Grandjean'07, Lindell'08, Kazana-Segoufin'10: linear preprocessing + constant delay.
2. Only first-order free variables and acyclic conjunctive formula. Bagan-Durand-Grandjean'07: linear preprocessing + linear delay
3. Monadic second order formula and bounded tree-width structure Bagan, Courcelle 2009: almost linear preprocessing + linear delay

**Example**

Typical database query. Simple paths of length $k$.

# Previous results

1. Only first-order free variables and bounded degree structures. Durand-Grandjean'07, Lindell'08, Kazana-Segoufin'10: linear preprocessing + constant delay.
2. Only first-order free variables and acyclic conjunctive formula. Bagan-Durand-Grandjean'07: linear preprocessing + linear delay
3. Monadic second order formula and bounded tree-width structure Bagan, Courcelle 2009: almost linear preprocessing + linear delay

**Example**

Enumeration of the cliques of a bounded tree-width graph.

# A hierarchy result for counting functions

From a formula $\Phi(\mathbf{z}, \mathbf{T})$, one defines the counting function:

$$\#\Phi : \mathcal{S} \mapsto |\Phi(\mathcal{S})|.$$

**Theorem (Saluja, Subrahmanyam, Thakur 1995)**

*On linearly ordered structures:*
$\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2 = \sharp\mathrm{P}$.

Some $\sharp\mathrm{P}$-hard problems in $\#\Sigma_1$ (but existence of FPRAS at this level).

**Corollary**

*On linearly ordered structures:*
$\textsc{Enum}{\cdot}\Sigma_0 \subsetneq \textsc{Enum}{\cdot}\Sigma_1 \subsetneq \textsc{Enum}{\cdot}\Pi_1 \subsetneq \textsc{Enum}{\cdot}\Sigma_2 \subsetneq \textsc{Enum}{\cdot}\Pi_2$.

# A hierarchy result for counting functions

From a formula $\Phi(\mathbf{z}, \mathbf{T})$, one defines the counting function:

$$\#\Phi : \mathcal{S} \mapsto |\Phi(\mathcal{S})|.$$

**Theorem (Saluja, Subrahmanyam, Thakur 1995)**

*On linearly ordered structures:*
$\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2 = \sharp\mathrm{P}.$

Some $\sharp\mathrm{P}$-hard problems in $\#\Sigma_1$ (but existence of FPRAS at this level).

**Corollary**

*On linearly ordered structures:*
$\textsc{Enum}{\cdot}\Sigma_0 \subsetneq \textsc{Enum}{\cdot}\Sigma_1 \subsetneq \textsc{Enum}{\cdot}\Pi_1 \subsetneq \textsc{Enum}{\cdot}\Sigma_2 \subsetneq \textsc{Enum}{\cdot}\Pi_2.$

# The first level: Enum$\cdot\Sigma_0$

**Theorem**

*For $\varphi \in \Sigma_0$, $\textsc{Enum}\cdot\varphi$ can be enumerated with preprocessing $O(|D|^k)$ and delay $O(1)$ where $k$ is the number of free first order variables of $\varphi$ and $D$ is the domain of the input structure.*

**Simple ingredients:**

1. Transformation of a f.o. formula $\Phi(\mathbf{z}, T)$ into a propositional formulas for each $\mathbf{z}$.
2. Solve the propositional formula to obtain a minimal solution for $T$.
3. Gray Code Enumeration to extend $T$.

# Bounded degree structure

**Remark:** The $k$-clique query is definable.
No hope to improve the $O(|D|^k)$ preprocessing.

**Theorem**

*Let $d \in \mathbb{N}$, on $d$-degree bounded input structures, $\textsc{Enum}\cdot\Sigma_0 \in \textsc{Delay}(|D|, 1)$ where $D$ is the domain of the input structure.*

# Bounded degree structure

**Remark:** The $k$-clique query is definable.
No hope to improve the $O(|D|^k)$ preprocessing.

**Theorem**

*Let $d \in \mathbb{N}$, on $d$-degree bounded input structures, $\textsc{Enum}{\cdot}\Sigma_0 \in \textsc{Delay}(|D|, 1)$ where $D$ is the domain of the input structure.*

**Idea of proof:**

- Another transformation: $\Phi(\mathbf{z}, T)$ seen as a propositional formula whose variables are the atoms of $\Phi$.
- From each solution, create a quantifier free formula without free second order variables.
- Enumerate the solutions of this formula thanks to [DG 2007].

# Bounded degree structure

**Remark:** The $k$-clique query is definable.
No hope to improve the $O(|D|^k)$ preprocessing.

**Theorem**

*Let $d \in \mathbb{N}$, on $d$-degree bounded input structures, $\textsc{Enum}{\cdot}\Sigma_0 \in \textsc{Delay}(|D|, 1)$ where $D$ is the domain of the input structure.*

**Idea of proof:**

- Another transformation: $\Phi(\mathbf{z}, T)$ seen as a propositional formula whose variables are the atoms of $\Phi$.
- From each solution, create a quantifier free formula without free second order variables.
- Enumerate the solutions of this formula thanks to [DG 2007].

# Second level: Enum$\cdot\Sigma_1$

**Theorem**

$\textsc{Enum}\cdot\Sigma_1 \subseteq \textsc{DelayP}$. *More precisely,* $\textsc{Enum}\cdot\Sigma_1$ *can be computed with precomputation* $O(|D|^{h+k})$ *and delay* $O(|D|^k)$ *where* $h$ *is the number of free first order variables of the formula,* $k$ *the number of existentially quantified variables and* $D$ *is the domain of the input structure.*

**Idea of Proof:** $\Phi(\mathbf{y}, T) = \exists \mathbf{x} \varphi(\mathbf{x}, \mathbf{y}, T)$

- Substitute values for $\mathbf{x}$. Collection of formulas of the form:

$$\varphi(\mathbf{x}^*, \mathbf{y}, T)$$

- Need to enumerate the union.

# Enumeration of an union

Enumerate the solution of $\text{ENUM}\cdot R$ and $\text{ENUM}\cdot S$.

- Disjoint union
- Union with an order

# Enumeration of an union

Enumerate the solution of $\textsc{Enum}{\cdot}R$ and $\textsc{Enum}{\cdot}S$.

- Disjoint union
- Union with an order
- Union without order

# Enumeration of an union

Enumerate the solution of $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$.

- Disjoint union
- Union with an order
- Union without order

**Idea:** run $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$ in parallel with appropriate priority.

# Enumeration of an union

Enumerate the solution of $\textsc{Enum}{\cdot}R$ and $\textsc{Enum}{\cdot}S$.

- Disjoint union
- Union with an order
- Union without order

**Idea:** run $\textsc{Enum}{\cdot}R$ and $\textsc{Enum}{\cdot}S$ in parallel with appropriate priority.

**Question:** possible to improve this scheme ? A better algorithm for $\textsc{Enum}{\cdot}\mathrm{DNF}(\textsc{l})$?

# Enumeration of an union

Enumerate the solution of $\textsc{Enum}{\cdot}R$ and $\textsc{Enum}{\cdot}S$.

- Disjoint union
- Union with an order
- Union without order

**Idea:** run $\textsc{Enum}{\cdot}R$ and $\textsc{Enum}{\cdot}S$ in parallel with appropriate priority.

**Question:** possible to improve this scheme ? A better algorithm for $\textsc{Enum}{\cdot}\mathrm{DNF}(\textsc{l})$?

# The case Enum$\cdot\Pi_1$

**Proposition**

*Unless* $\mathrm{P} = \mathrm{NP}$, *there is no polynomial delay algorithm for* $\textsc{Enum}\cdot\Pi_1$.

**Proof** Direct encoding of SAT.

Hardness even:

- on the class of bounded degree structure
- if all clauses but one have at most two occurences of a second-order free variable

# Tractable cases

Problem $\textsc{Enum}{\cdot}\Phi$ with $\Phi \in \Sigma_i$: transformation of $\Phi$ into a propositional formula $\tilde{\Phi}$.

**Corollary**

*Let $\Phi(\mathbf{z}, T)$ be a formula, such that, for all $\sigma$ structures, all propositional formulas $\tilde{\Phi}$ are either Horn, anti-Horn, affine or bijunctive. Then $\textsc{Enum}{\cdot}\Phi \subseteq \textsc{Delay}\mathrm{P}$.*

# Tractable cases

Problem $\textsc{Enum}{\cdot}\Phi$ with $\Phi \in \Sigma_i$: transformation of $\Phi$ into a propositional formula $\tilde{\Phi}$.

**Corollary**

*Let $\Phi(\mathbf{z}, T)$ be a formula, such that, for all $\sigma$ structures, all propositional formulas $\tilde{\Phi}$ are either Horn, anti-Horn, affine or bijunctive. Then $\textsc{Enum}{\cdot}\Phi \subseteq \textsc{Delay}\mathrm{P}$.*

**Example:** independent sets and hitting sets respectively bijunctive and Horn.

# Tractable cases

Problem $\textsc{Enum}{\cdot}\Phi$ with $\Phi \in \Sigma_i$: transformation of $\Phi$ into a propositional formula $\tilde{\Phi}$.

**Corollary**

*Let $\Phi(\mathbf{z}, T)$ be a formula, such that, for all $\sigma$ structures, all propositional formulas $\tilde{\Phi}$ are either Horn, anti-Horn, affine or bijunctive. Then $\textsc{Enum}{\cdot}\Phi \subseteq \textsc{Delay}\mathrm{P}$.*

**Example:** independent sets and hitting sets respectively bijunctive and Horn.

# Further tractable cases

Add a maximization/minimization operator.

$\textsc{Enum}{\cdot}Max_T\varphi(T)$ is the problem of enumerating all maximal models (for inclusion) of $\varphi$.

# Further tractable cases

Add a maximization/minimization operator.

$\text{ENUM}\cdot Max_T \varphi(T)$ is the problem of enumerating all maximal models (for inclusion) of $\varphi$.

We can represent ENUM·MAX-INDSET. Reduction to ENUM·MAX-2SAT: **polynomial delay**.

# Further tractable cases

Add a maximization/minimization operator.

$\textsc{Enum}\cdot Max_T \varphi(T)$ is the problem of enumerating all maximal models (for inclusion) of $\varphi$.

We can represent $\textsc{Enum}\cdot\textsc{Max}$-INDSET. Reduction to $\textsc{Enum}\cdot\textsc{Max}$-2SAT: **polynomial delay**.

We can represent $\textsc{Enum}\cdot\textsc{Min}$-HITTINGSET. Reduction to $\textsc{Enum}\cdot\textsc{Min}$-HORNSAT: hard problem.

# Further tractable cases

Add a maximization/minimization operator.

$\textsc{Enum}{\cdot}Max_T\varphi(T)$ is the problem of enumerating all maximal models (for inclusion) of $\varphi$.

We can represent $\textsc{Enum}{\cdot}\textsc{Max}\text{-INDSET}$. Reduction to $\textsc{Enum}{\cdot}\textsc{Max}\text{-2SAT}$: **polynomial delay**.

We can represent $\textsc{Enum}{\cdot}\textsc{Min}\text{-HITTINGSET}$. Reduction to $\textsc{Enum}{\cdot}\textsc{Min}\text{-HORNSAT}$: hard problem.

Introduction to Enumeration

Enumeration and polynomials

Enumeration and logic

Enumeration and polytopes

# Verification problems

Want to compare and separate two systems:

1. Regular automata: equality of language, linear time.
2. Probabilistic automata: equality of language, cubic time, better randomized algorithm.

# Verification problems

Want to compare and separate two systems:

1. Regular automata: equality of language, linear time.
2. Probabilistic automata: equality of language, cubic time, better randomized algorithm.
3. Non-deterministic automata. Equivalence $\mathrm{PSPACE}$-complete. Approximation for the edit distance with moves. ustat$_k(w)$: the density vector of all the $n-k+1$ subwords of length $k$ of the word $w$.

# Verification problems

Want to compare and separate two systems:

1. Regular automata: equality of language, linear time.
2. Probabilistic automata: equality of language, cubic time, better randomized algorithm.
3. Non-deterministic automata. Equivalence $\mathrm{PSPACE}$-complete. Approximation for the edit distance with moves. $\mathsf{ustat}_k(w)$: the density vector of all the $n-k+1$ subwords of length $k$ of the word $w$.
4. Markov decision processes.

# Verification problems

Want to compare and separate two systems:

1. Regular automata: equality of language, linear time.
2. Probabilistic automata: equality of language, cubic time, better randomized algorithm.
3. Non-deterministic automata. Equivalence $\mathrm{PSPACE}$-complete. Approximation for the edit distance with moves. $\mathsf{ustat}_k(w)$: the density vector of all the $n-k+1$ subwords of length $k$ of the word $w$.
4. Markov decision processes.

# Probabilistic automata

The polynomial for the automaton $A$:

$$\sum_{w \in \Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \ldots X_{n,\sigma_n}$$

Possible to change $n$, the size of the generated words.

NP-hard to approximate the bounded maximal distance: $\max_{w \in \Sigma^n} |A(w) - B(w)|$.

# Probabilistic automata

The polynomial for the automaton $A$:

$$\sum_{w \in \Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \dots X_{n,\sigma_n}$$

Possible to change $n$, the size of the generated words.

$\mathrm{NP}$-hard to approximate the bounded maximal distance: $\max_{w \in \Sigma^n} |A(w) - B(w)|$.

**Proposition**

*Let $P$ be a multilinear polynomial given by a black box. There is no polynomial delay algorithm to produce the monomials in decreasing order of coefficient unless $\mathrm{P} = \mathrm{NP}$.*

# Probabilistic automata

The polynomial for the automaton $A$:

$$\sum_{w \in \Sigma^n} A(w) X_{1,\sigma_1} X_{2,\sigma_2} \dots X_{n,\sigma_n}$$

Possible to change $n$, the size of the generated words.

$\mathrm{NP}$-hard to approximate the bounded maximal distance:
$\max_{w \in \Sigma^n} |A(w) - B(w)|$.

**Proposition**

*Let $P$ be a multilinear polynomial given by a black box. There is no polynomial delay algorithm to produce the monomials in decreasing order of coefficient unless $\mathrm{P} = \mathrm{NP}$.*

# The polytope separation

Let $K_1, K_2$ be two polytopes. We want to produce a point in $K_1 \triangle K_2$.
Enumeration = **uniform sampling**.

Different representation of polytopes:

1. Convex hull of a set of given points: $\mathcal{V}$-polytope
2. A set of linear inequalities: $\mathcal{H}$-polytope
3. A way to test if a point is in the polytope: strong membership oracle (SMO).

# The polytope separation

Let $K_1, K_2$ be two polytopes. We want to produce a point in $K_1 \bigtriangleup K_2$.
Enumeration = **uniform sampling**.

Different representation of polytopes:

1. Convex hull of a set of given points: $\mathcal{V}$-polytope
2. A set of linear inequalities: $\mathcal{H}$-polytope
3. A way to test if a point is in the polytope: strong membership oracle (SMO).

# A simple solution

For $K_1, K_2$ $\mathcal{H}$-polytopes:
Negate a constraint of $K_1$, add it to $K_2$ then decide whether the system has a solution.

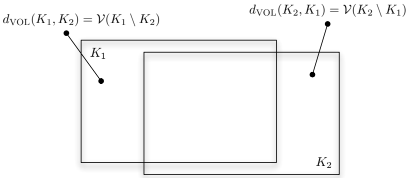
**Problem:** we are dealing with other representations.
For non-deterministic automaton, a $\mathcal{V}$-polytope: the $\mathrm{ustat}_k$ of accepted words.

# A simple solution

For $K_1, K_2$ $\mathcal{H}$-polytopes:
Negate a constraint of $K_1$, add it to $K_2$ then decide whether the system has a solution.

**Problem:** we are dealing with other representations.
For non-deterministic automaton, a $\mathcal{V}$-polytope: the $\mathsf{ustat}_k$ of accepted words.

# Distance and random walk



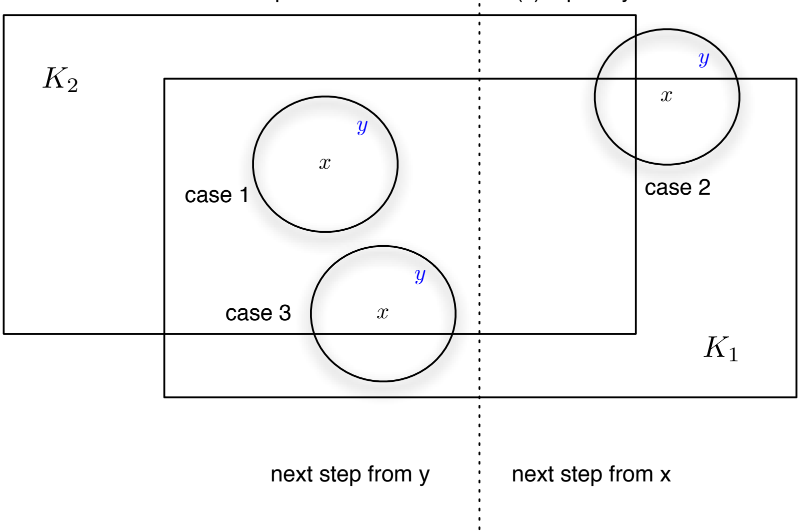

$$d_{vol}(K_1, K_2) = \max\left\{\frac{d_{\mathrm{VOL}}(K_1, K_2)}{\mathcal{V}(K_1)}, \frac{d_{\mathrm{VOL}}(K_2, K_1)}{\mathcal{V}(K_2)}\right\}$$

**Theorem**

*Let $K_1$ and $K_2$ be two polytopes, given as SMOs. For all $\epsilon > 0$, if $d_{vol}(K_1, K_2) \geq \varepsilon$, then the* Polytope Separator *outputs a point $x$ in $K_1 \triangle K_2$ with probability greater than $2/3$. Moreover, the running time of this algorithm is polynomial in $n$ and $\epsilon^{-1}$.*

# The ball walk

We use the Ball Walk... we pick at random unif. in B(x) a point y : 3 cases



# Approximate verification: witness generation

**Theorem**

*Given two regular expressions $r_1, r_2$ on words and $\epsilon$, if $d_{vol}(H_{r_1}, H_{r_2}) \geq \lambda$, we can generate $\epsilon$-separating words in polynomial time in the dimension and $1/\lambda$.*

Thanks!

Thanks!
Thanks,

Thanks!
Thanks, thanks,

Thanks!
Thanks, thanks, thanks,

Thanks!
Thanks, thanks, thanks, thanks,

Thanks!

Thanks, thanks, thanks, thanks, thanks,

Thanks!

Thanks, thanks, thanks, thanks, thanks, thanks,

Thanks!

Thanks, thanks, thanks, thanks, thanks, thanks, thanks,

Thanks!

Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks,

Thanks!
Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks,

Thanks!

Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks

Thanks!

Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks

Let's all do enumeration