The Logic of Counting Query Answers

Hubie Chen
Univ. del País Vasco & Ikerbasque
San Sebastián, Spain

Simons Institute, Berkeley – November 2016

Joint work with Stefan Mengel,
CRIL-CNRS / Université d'Artois

Act: Overview

Query evaluation

Query evaluation

Basic problem in logic/database theory:

Evaluate a formula $\phi(V)$ on a finite structure $\mathbf{B}$,
that is, determine $\phi(\mathbf{B}) = \{h : V \to B \mid \mathbf{B}, h \models \phi\}$

Query evaluation

Basic problem in logic/database theory:

Evaluate a formula $\phi(V)$ on a finite structure $\mathbf{B}$,
that is, determine $\phi(\mathbf{B}) = \{h : V \to B \mid \mathbf{B}, h \models \phi\}$

Here:

- ϕ first-order formula

Query evaluation

Basic problem in logic/database theory:

Evaluate a formula $\phi(V)$ on a finite structure **B**,
that is, determine $\phi(\mathbf{B}) = \{h : V \to B \mid \mathbf{B}, h \models \phi\}$

Here:

- ϕ first-order formula
- problem of counting query answers — determine $|\phi(\mathbf{B})|$
  ...intractable, in general...

Query evaluation

Basic problem in logic/database theory:

Evaluate a formula $\phi(V)$ on a finite structure $\mathbf{B}$,
that is, determine $\phi(\mathbf{B}) = \{h : V \to B \mid \mathbf{B}, h \models \phi\}$

Here:

- ϕ first-order formula
- problem of counting query answers — determine $|\phi(\mathbf{B})|$
  ...intractable, in general...

Example: $\phi(u, v) = \exists x(E(u, x) \wedge E(x, v))$

Query evaluation

Basic problem in logic/database theory:

Evaluate a formula $\phi(V)$ on a finite structure **B**,
that is, determine $\phi(\mathbf{B}) = \{h : V \to B \mid \mathbf{B}, h \models \phi\}$

Here:

- ϕ first-order formula
- problem of counting query answers — determine $|\phi(\mathbf{B})|$
  ...intractable, in general...

Example: $\phi(u, v) = \exists x(E(u, x) \wedge E(x, v))$

Generalizes model checking — determine if $\mathbf{B} \models \phi$
where ϕ is a sentence

- Here, we have $|\phi(\mathbf{B})| = 1 \quad \Leftrightarrow \quad \mathbf{B} \models \phi$

♯-logic

$\sharp$-logic

Suppose we have a first-order formula ϕ in hand,
and are interested in counting $|\phi(\mathbf{B})|$ for various structs **B**

$\sharp$-logic

Suppose we have a first-order formula ϕ in hand,
and are interested in counting $|\phi(\mathbf{B})|$ for various structs **B**

Motivating question: is there a language/logic in which one can express algorithms for computing the mapping $\mathbf{B} \rightarrow |\phi(\mathbf{B})|$?

$\sharp$-logic

Suppose we have a first-order formula ϕ in hand,
and are interested in counting $|\phi(\mathbf{B})|$ for various structs $\mathbf{B}$

Motivating question: is there a language/logic in which one can express algorithms for computing the mapping $\mathbf{B} \to |\phi(\mathbf{B})|$?

We present such a logic, $\sharp$-logic
— can serve as a target language into which one can compile FO-formulas of interest

$\sharp$-logic

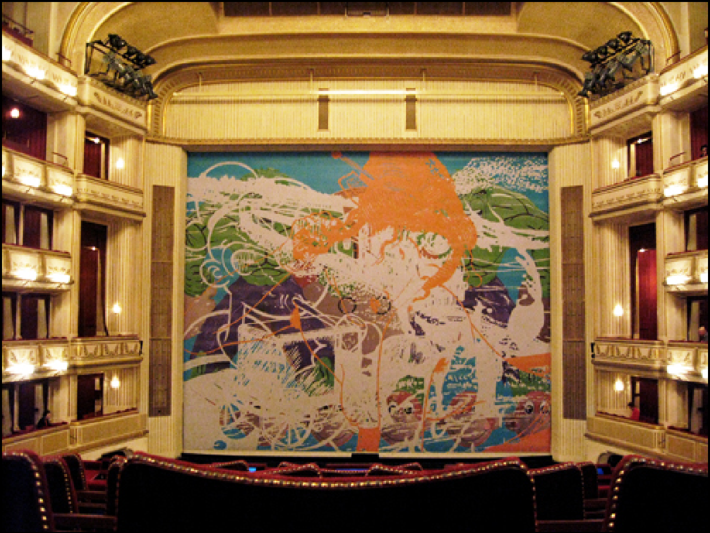
Suppose we have a first-order formula ϕ in hand,
and are interested in counting $|\phi(\mathbf{B})|$ for various structs **B**

Motivating question: is there a language/logic in which one can express algorithms for computing the mapping $\mathbf{B} \to |\phi(\mathbf{B})|$?

We present such a logic, $\sharp$-logic
— can serve as a target language into which one can compile FO-formulas of interest

$\sharp$-logic enjoys & balances:

- Expressivity: in a precise sense, can express known efficient algorithms for counting query answers in $\sharp$-logic
- Optimizability: minimizing width can be done computably (in an expressive fragment of $\sharp$-logic)



Act: Background

Studying complexity

Studying complexity

Counting query answers: intractable in general

Studying complexity

Counting query answers: intractable in general

Restrict to a single first-order formula: poly-time tractable

Studying complexity

Counting query answers: intractable in general

Restrict to a single first-order formula: poly-time tractable

Here, we seek tractable cases of the general problem
by identifying *classes* of first-order formulas Φ
on which the problem is tractable

Studying complexity

Counting query answers: intractable in general

Restrict to a single first-order formula: poly-time tractable

Here, we seek tractable cases of the general problem
by identifying *classes* of first-order formulas Φ
on which the problem is tractable

Let Φ be a class of first-order formulas

Def: p-count(Φ) is the problem...

Given $\phi(V) \in \Phi$ and a finite struct $\mathbf{B}$,
output $|\phi(\mathbf{B})|$

Parameterized complexity

Parameterized complexity

- Argued: classical complexity notions (eg, poly time) are not satisfactory in the study of query evaluation

Parameterized complexity

- Argued: classical complexity notions (eg, poly time) are not satisfactory in the study of query evaluation
- Typical scenario: short query on BIG structure

Parameterized complexity

- Argued: classical complexity notions (eg, poly time) are not satisfactory in the study of query evaluation
- Typical scenario: short query on BIG structure

  ⇒ we might tolerate
  a non-polynomial, bad dependence on query,
  so long as have good dependence on structure

Parameterized complexity

- Argued: classical complexity notions (eg, poly time) are not satisfactory in the study of query evaluation

- Typical scenario: short query on BIG structure

  ⇒ we might tolerate
  a non-polynomial, bad dependence on query,
  so long as have good dependence on structure

- Parameterized complexity theory: classify problems up to allowing arbitrary dependence on a parameter

Parameterized complexity

- Argued: classical complexity notions (eg, poly time) are not satisfactory in the study of query evaluation
- Typical scenario: short query on BIG structure

  ⇒ we might tolerate
  a non-polynomial, bad dependence on query,
  so long as have good dependence on structure
- Parameterized complexity theory: classify problems up to allowing arbitrary dependence on a parameter

  Here: the query/formula is the parameter

Tractability

Tractability

Let Φ be a class of first-order formulas.

Def: p-count(Φ) is the problem...

Given $\phi(V) \in \Phi$ and a finite struct $\mathbf{B}$,
output $|\phi(\mathbf{B})|$

Tractability

Let Φ be a class of first-order formulas.

Def: p-count(Φ) is the problem...

Given $\phi(V) \in \Phi$ and a finite struct $\mathbf{B}$,
output $|\phi(\mathbf{B})|$

Here: p-count(Φ) is tractable if

$\exists$ an algorithm f and a poly-time algorithm A such that

given $(\phi, \mathbf{B})$, the value $|\phi(\mathbf{B})|$ is computed by $A(f(\phi), \mathbf{B})$

("fixed-parameter tractable")

Classification

Classification Thm (Chen & Mengel, ICDT ’15/PODS ’16):

Let Φ be a class of $\{\exists, \wedge, \vee\}$-formulas (of bounded arity).

Classification

Classification Thm (Chen & Mengel, ICDT '15/PODS '16):

Let Φ be a class of $\{\exists, \wedge, \vee\}$-formulas (of bounded arity).

- If (X), then p-count(Φ) is tractable (in FPT).
- Else, p-count(Φ) is not tractable, unless W[1] = FPT.

Width

Def: The *width* of a FO-formula ϕ is $\max_\psi |\text{free}(\psi)|$, where max is over all subformulas ψ of ϕ

Width

Def: The *width* of a FO-formula ϕ is $\max_{\psi} |\text{free}(\psi)|$,
where max is over all subformulas ψ of ϕ

Obs (Immerman '82, Vardi '95): For each $k \geqslant 1$, $\exists$ poly-time alg for evaluating a FO-sentence of width $\leqslant k$ on a finite struct

Width

Def: The *width* of a FO-formula ϕ is $\max_{\psi} |\text{free}(\psi)|$,
where max is over all subformulas ψ of ϕ

Obs (Immerman '82, Vardi '95): For each $k \geqslant 1$, $\exists$ poly-time alg for evaluating a FO-sentence of width $\leqslant k$ on a finite struct

Obs (Chen '14): The following condition is sufficient for model checking on Φ to be in FPT:

$\exists k \geqslant 1$ and an alg f that computes, for each $\phi \in \Phi$,
a logically equiv sentence $f(\phi)$ of width $\leqslant k$

(By model checking on a class of sentences Φ,
we refer to p-count(Φ))

Model checking

Obs (Chen '14): The following condition is sufficient for model checking on Φ to be in FPT:

$$\exists k \geqslant 1 \text{ and an alg } f \text{ that computes, for each } \phi \in \Phi, \text{ a logically equiv sentence } f(\phi) \text{ of width } \leqslant k$$

Model checking

Obs (Chen '14): The following condition is sufficient for model checking on Φ to be in FPT:

$\exists k \geqslant 1$ and an alg f that computes, for each $\phi \in \Phi$, a logically equiv sentence $f(\phi)$ of width $\leqslant k$

Thm (Chen '14): (building on Kolaitis, Vardi, ...)
Let Φ be *any* class of $\{\exists, \wedge, \vee\}$-sentences (of bounded arity).
If model checking on Φ in FPT, then condition holds,
ie, above condition is exclusive explanation for FPT!

Model checking

Obs (Chen '14): The following condition is sufficient for model checking on Φ to be in FPT:

$\exists k \geqslant 1$ and an alg f that computes, for each $\phi \in \Phi$,
a logically equiv sentence $f(\phi)$ of width $\leqslant k$

Thm (Chen '14): (building on Kolaitis, Vardi, ...)
Let Φ be *any* class of $\{\exists, \wedge, \vee\}$-sentences (of bounded arity).

If model checking on Φ in FPT, then condition holds,

ie, above condition is exclusive explanation for FPT!

Conceptual point: FO logic is a useful model of computation!

Model checking

Obs (Chen '14): The following condition is sufficient for model checking on Φ to be in FPT:

$\exists k \geqslant 1$ and an alg f that computes, for each $\phi \in \Phi$, a logically equiv sentence $f(\phi)$ of width $\leqslant k$

Thm (Chen '14): (building on Kolaitis, Vardi, ...)
Let Φ be *any* class of $\{\exists, \wedge, \vee\}$-sentences (of bounded arity).
If model checking on Φ in FPT, then condition holds,
ie, above condition is exclusive explanation for FPT!

Conceptual point: FO logic is a useful model of computation!

- If we have tractability at all, we have tractability via putting the sentences in the right "format"

Model checking

Obs (Chen '14): The following condition is sufficient for model checking on Φ to be in FPT:

$$\exists k \geqslant 1 \text{ and an alg } f \text{ that computes, for each } \phi \in \Phi,$$
$$\text{a logically equiv sentence } f(\phi) \text{ of width } \leqslant k$$

Thm (Chen '14): (building on Kolaitis, Vardi, ...)
Let Φ be *any* class of $\{\exists, \wedge, \vee\}$-sentences (of bounded arity).
If model checking on Φ in FPT, then condition holds,
ie, above condition is exclusive explanation for FPT!

Conceptual point: FO logic is a useful model of computation!

- If we have tractability at all, we have tractability via putting the sentences in the right "format"
- FO logic contains the computational primitives needed to express the algorithm witnessing tractability

A logic for counting query answers?

A logic for counting query answers?

Question: is there some analogously useful “logic” for counting query answers?

...that is, in which one can express efficient algorithms?

A logic for counting query answers?

Question: is there some analogously useful “logic” for counting query answers?

...that is, in which one can express efficient algorithms?

In some sense, we are asking for a counting analog of bounded-width logic!

A logic for counting query answers?

Question: is there some analogously useful “logic” for counting query answers?

...that is, in which one can express efficient algorithms?

In some sense, we are asking for a counting analog of bounded-width logic!

Observe...

A logic for counting query answers?

Question: is there some analogously useful “logic” for counting query answers?

...that is, in which one can express efficient algorithms?

In some sense, we are asking for a counting analog of bounded-width logic!

Observe...

- In model checking: given sentence ϕ, struct **B**, want to decide if **B** $\models \phi$

A logic for counting query answers?

Question: is there some analogously useful “logic” for counting query answers?

...that is, in which one can express efficient algorithms?

In some sense, we are asking for a counting analog of bounded-width logic!

Observe...

- In model checking: given sentence ϕ, struct **B**, want to decide if $\mathbf{B} \models \phi$
- In counting answers: given formula $\phi(V)$, struct **B**, want to compute $|\phi(\mathbf{B})|$

A logic for counting query answers?

Question: is there some analogously useful “logic” for counting query answers?

...that is, in which one can express efficient algorithms?

In some sense, we are asking for a counting analog of bounded-width logic!

Observe...

- In model checking: given sentence ϕ, struct **B**, want to decide if $\mathbf{B} \models \phi$
- In counting answers: given formula $\phi(V)$, struct **B**, want to compute $|\phi(\mathbf{B})|$

We want some notion of sentence that, when evaluated on a struct **B**, returns a numerical value (not just true/false)

A logic for counting query answers?

Question: is there some analogously useful "logic" for counting query answers?

...that is, in which one can express efficient algorithms?

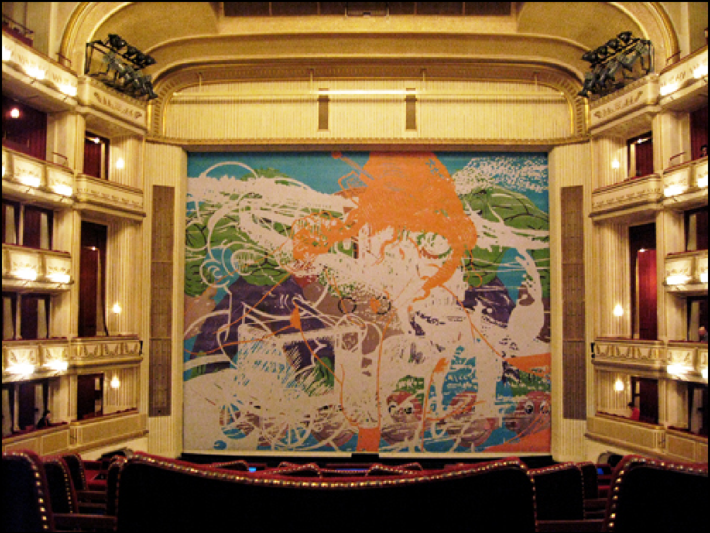
In some sense, we are asking for a counting analog of bounded-width logic!

Observe...

- In model checking: given sentence ϕ, struct **B**, want to decide if $\mathbf{B} \models \phi$
- In counting answers: given formula $\phi(V)$, struct **B**, want to compute $|\phi(\mathbf{B})|$

We want some notion of sentence that, when evaluated on a struct **B**, returns a numerical value (not just true/false)

Will give a logic: $\sharp$-logic



Act: $\sharp$-logic

$\sharp$-logic

$\sharp$-logic

Each $\sharp$-*formula* ψ has a set of free variables, $\mathsf{free}(\psi)$.

$\sharp$-logic

Each $\sharp$-*formula* ψ has a set of free variables, $\mathrm{free}(\psi)$.

When $\mathrm{free}(\psi) = \varnothing$, we say that ψ is a $\sharp$-*sentence*.

$\sharp$-logic

Each $\sharp$-*formula* ψ has a set of free variables, $\text{free}(\psi)$.

When $\text{free}(\psi) = \varnothing$, we say that ψ is a $\sharp$-*sentence*.

Obs: For each $k \geqslant 1$, evaluating a $\sharp$-sentence of width $\leqslant k$ on a finite struct is polytime computable

$\sharp$-logic

Each $\sharp$-*formula* ψ has a set of free variables, free(ψ).

When free$(\psi) = \emptyset$, we say that ψ is a $\sharp$-*sentence*.

Obs: For each $k \geqslant 1$, evaluating a $\sharp$-sentence of width $\leqslant k$ on a finite struct is polytime computable

Def: A $\sharp$-sentence ψ represents a FO formula $\phi(V)$ if, for each struct **B**, evaluating ψ on **B** gives the value $|\phi(\mathbf{B})|$

$\sharp$-logic

Each $\sharp$-*formula* ψ has a set of free variables, free(ψ).

When free$(\psi) = \emptyset$, we say that ψ is a $\sharp$-*sentence*.

Obs: For each $k \geqslant 1$, evaluating a $\sharp$-sentence of width $\leqslant k$ on a finite struct is polytime computable

Def: A $\sharp$-sentence ψ represents a FO formula $\phi(V)$ if, for each struct $\mathbf{B}$, evaluating ψ on $\mathbf{B}$ gives the value $|\phi(\mathbf{B})|$

Obs: The following condition is sufficient for counting answers on Φ (p-count(Φ)) to be in FPT:

$\exists k \geqslant 1$ and an alg f that computes, for each $\phi \in \Phi$, a $\sharp$-sentence representation $f(\phi)$ of width $\leqslant k$

$\sharp$-logic

Each $\sharp$-*formula* ψ has a set of free variables, free(ψ).

When free$(\psi) = \emptyset$, we say that ψ is a $\sharp$-*sentence*.

Obs: For each $k \geqslant 1$, evaluating a $\sharp$-sentence of width $\leqslant k$ on a finite struct is polytime computable

Def: A $\sharp$-sentence ψ represents a FO formula $\phi(V)$ if, for each struct $\mathbf{B}$, evaluating ψ on $\mathbf{B}$ gives the value $|\phi(\mathbf{B})|$

Obs: The following condition is sufficient for counting answers on Φ (p-count(Φ)) to be in FPT:

$\exists k \geqslant 1$ and an alg f that computes, for each $\phi \in \Phi$, a $\sharp$-sentence representation $f(\phi)$ of width $\leqslant k$

A Main Thm: On classes Φ of $\{\exists, \wedge, \vee\}$-formulas, this condition is exclusive explanation for FPT!

Example

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ ("casting")

Relative to a struct **B**...

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ ("casting")

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0, depending on whether $\mathbf{B}, h \models E(v, y)$

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ (“casting”)

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0, depending on whether $\mathbf{B}, h \models E(v, y)$
- $[\mathbf{B}, Py\psi_E](g : \{v\} \to B)$ gives the num of exts $\{v, y\} \to B$ of g satisfying $E(v, y)$

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ ("casting")

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0, depending on whether $\mathbf{B}, h \models E(v, y)$
- $[\mathbf{B}, Py\psi_E](g : \{v\} \to B)$ gives the num of exts $\{v, y\} \to B$ of g satisfying $E(v, y)$
- $[\mathbf{B}, Pz\psi_F](g : \{v\} \to B)$ gives the num of exts $\{v, z\} \to B$ of g satisfying $F(v, z)$

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ (“casting”)

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0, depending on whether $\mathbf{B}, h \models E(v, y)$
- $[\mathbf{B}, Py\psi_E](g : \{v\} \to B)$ gives the num of exts $\{v, y\} \to B$ of g satisfying $E(v, y)$
- $[\mathbf{B}, Pz\psi_F](g : \{v\} \to B)$ gives the num of exts $\{v, z\} \to B$ of g satisfying $F(v, z)$
- $[\mathbf{B}, Py\psi_E \times Pz\psi_F](g : \{v\} \to B)$ gives

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ ("casting")

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0, depending on whether $\mathbf{B}, h \models E(v, y)$
- $[\mathbf{B}, Py\psi_E](g : \{v\} \to B)$ gives the num of exts $\{v, y\} \to B$ of g satisfying $E(v, y)$
- $[\mathbf{B}, Pz\psi_F](g : \{v\} \to B)$ gives the num of exts $\{v, z\} \to B$ of g satisfying $F(v, z)$
- $[\mathbf{B}, Py\psi_E \times Pz\psi_F](g : \{v\} \to B)$ gives ...the product of the previous two quantities...

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ ("casting")

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0,
  depending on whether $\mathbf{B}, h \models E(v, y)$
- $[\mathbf{B}, Py\psi_E](g : \{v\} \to B)$ gives
  the num of exts $\{v, y\} \to B$ of g satisfying $E(v, y)$
- $[\mathbf{B}, Pz\psi_F](g : \{v\} \to B)$ gives
  the num of exts $\{v, z\} \to B$ of g satisfying $F(v, z)$
- $[\mathbf{B}, Py\psi_E \times Pz\psi_F](g : \{v\} \to B)$ gives
  ...the product of the previous two quantities...
  ...which is the num of exts $\{v, y, z\} \to B$ of g satisfying ϕ

Example

Consider the formula $\phi(v, y, z) = E(v, y) \wedge F(v, z)$

Set $\psi_E = C(E(v, y))$, $\psi_F = C(F(v, z))$ ("casting")

Relative to a struct **B**...

- $[\mathbf{B}, \psi_E](h : \{v, y\} \to B)$ is 1 or 0, depending on whether $\mathbf{B}, h \models E(v, y)$
- $[\mathbf{B}, Py\psi_E](g : \{v\} \to B)$ gives the num of exts $\{v, y\} \to B$ of g satisfying $E(v, y)$
- $[\mathbf{B}, Pz\psi_F](g : \{v\} \to B)$ gives the num of exts $\{v, z\} \to B$ of g satisfying $F(v, z)$
- $[\mathbf{B}, Py\psi_E \times Pz\psi_F](g : \{v\} \to B)$ gives ...the product of the previous two quantities... ...which is the num of exts $\{v, y, z\} \to B$ of g satisfying ϕ
- So if we take the previous $\sharp$-formula and project v, get representation $Pv(Py\psi_E \times Pz\psi_F)$ of ϕ

$\sharp$-logic: a summary

$\sharp$-logic: a summary

- Casting

  $C(\phi)$ is a $\sharp$-formula if ϕ is a FO-formula

$\sharp$-logic: a summary

- Casting

  $C(\phi)$ is a $\sharp$-formula if ϕ is a FO-formula

- Projection

  $Pv\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula and...

  $\text{free}(Pv\phi) = \text{free}(\phi)\backslash\{v\}$, $\text{closed}(Pv\phi) = \{v\} \cup \text{closed}(\phi)$

$\sharp$-logic: a summary

- **Casting**

  $C(\phi)$ is a $\sharp$-formula if ϕ is a FO-formula

- **Projection**

  $Pv\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula and...

  $\text{free}(Pv\phi) = \text{free}(\phi)\backslash\{v\}$, $\text{closed}(Pv\phi) = \{v\} \cup \text{closed}(\phi)$

- **Expansion**

  $Ev\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula, $v \notin \text{free}(\phi) \cup \text{closed}(\phi)$

  $\text{free}(Ev\phi) = \{v\} \cup \text{free}(\phi)$, $\text{closed}(Ev\phi) = \text{closed}(\phi)$

$\sharp$-logic: a summary

- Casting

  $C(\phi)$ is a $\sharp$-formula if ϕ is a FO-formula

- Projection

  $Pv\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula and...

  $\text{free}(Pv\phi) = \text{free}(\phi)\backslash\{v\}$, $\text{closed}(Pv\phi) = \{v\} \cup \text{closed}(\phi)$

- Expansion

  $Ev\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula, $v \notin \text{free}(\phi) \cup \text{closed}(\phi)$

  $\text{free}(Ev\phi) = \{v\} \cup \text{free}(\phi)$, $\text{closed}(Ev\phi) = \text{closed}(\phi)$

- Multiplication and addition

  $\phi \times \phi'$, $\phi + \phi'$ are $\sharp$-formulas if ϕ, ϕ' are $\sharp$-formulas with...

$\sharp$-logic: a summary

- Casting

  $C(\phi)$ is a $\sharp$-formula if ϕ is a FO-formula

- Projection

  $Pv\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula and...

  $\text{free}(Pv\phi) = \text{free}(\phi)\backslash\{v\}$, $\text{closed}(Pv\phi) = \{v\} \cup \text{closed}(\phi)$

- Expansion

  $Ev\phi$ is a $\sharp$-formula if ϕ is a $\sharp$-formula, $v \notin \text{free}(\phi) \cup \text{closed}(\phi)$

  $\text{free}(Ev\phi) = \{v\} \cup \text{free}(\phi)$, $\text{closed}(Ev\phi) = \text{closed}(\phi)$

- Multiplication and addition

  $\phi \times \phi'$, $\phi + \phi'$ are $\sharp$-formulas if ϕ, ϕ' are $\sharp$-formulas with...

- Constants

  Each $n \in \mathbb{Z}$ is a $\sharp$-formula

  $\text{free}(n) = \text{closed}(n) = \varnothing$

Width minimization

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Define $\phi'(a, b, c) = \phi(b, c, a)$.

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Define $\phi'(a, b, c) = \phi(b, c, a)$.

In general, ϕ and ϕ' are not logically equivalent...

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Define $\phi'(a, b, c) = \phi(b, c, a)$.

In general, ϕ and ϕ' are not logically equivalent...

...but they are counting equivalent...

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Define $\phi'(a, b, c) = \phi(b, c, a)$.

In general, ϕ and ϕ' are not logically equivalent...

...but they are counting equivalent...

Def: Two $\{\exists, \wedge\}$-formulas $\phi(V)$, $\phi'(V)$ are counting equivalent if, for each finite struct $\mathbf{B}$, it holds that $|\phi(\mathbf{B})| = |\phi'(\mathbf{B})|$

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Define $\phi'(a, b, c) = \phi(b, c, a)$.

In general, ϕ and ϕ' are not logically equivalent...

...but they are counting equivalent...

Def: Two $\{\exists, \wedge\}$-formulas $\phi(V)$, $\phi'(V)$ are counting equivalent if, for each finite struct **B**, it holds that $|\phi(\mathbf{B})| = |\phi'(\mathbf{B})|$

Thm: Two $\{\exists, \wedge\}$-formulas $\phi(V)$, $\phi'(V)$ are counting equivalent iff their variables can be renamed such that they become logically equivalent

Width minimization

Observation: consider a formula $\phi(a, b, c)$.

Define $\phi'(a, b, c) = \phi(b, c, a)$.

In general, ϕ and ϕ' are not logically equivalent...

...but they are counting equivalent...

Def: Two $\{\exists, \wedge\}$-formulas $\phi(V)$, $\phi'(V)$ are counting equivalent if, for each finite struct $\mathbf{B}$, it holds that $|\phi(\mathbf{B})| = |\phi'(\mathbf{B})|$

Thm: Two $\{\exists, \wedge\}$-formulas $\phi(V)$, $\phi'(V)$ are counting equivalent iff their variables can be renamed such that they become logically equivalent

Note: follows that there is an algorithm that decides counting equivalence

Width minimization

Thm (width minimization): There exists an alg *f* that, given a $\sharp$-formula where only $\{\exists, \wedge, \vee\}$-queries are casted, outputs a “logically equivalent” $\sharp$-formula of minimum width

Width minimization

Thm (width minimization): There exists an alg f that, given a $\sharp$-formula where only $\{\exists, \wedge, \vee\}$-queries are casted, outputs a "logically equivalent" $\sharp$-formula of minimum width

Idea of alg:

Width minimization

Thm (width minimization): There exists an alg f that, given a $\sharp$-formula where only $\{\exists, \wedge, \vee\}$-queries are casted, outputs a "logically equivalent" $\sharp$-formula of minimum width

Idea of alg:

- Show that each $\sharp$-formula ϕ can be normalized to the form $\sum_i (\text{integer}) C(\psi_i)$ without increasing width where each ψ_i is a $\{\exists, \wedge\}$-query

Width minimization

Thm (width minimization): There exists an alg f that, given a $\sharp$-formula where only $\{\exists, \wedge, \vee\}$-queries are casted, outputs a "logically equivalent" $\sharp$-formula of minimum width

Idea of alg:

- Show that each $\sharp$-formula ϕ can be normalized to the form $\sum_i$ (integer)$C(\psi_i)$ without increasing width where each ψ_i is a $\{\exists, \wedge\}$-query
- May then enforce that the ψ_i are counting inequivalent

Width minimization

Thm (width minimization): There exists an alg f that, given a $\sharp$-formula where only $\{\exists, \wedge, \vee\}$-queries are casted, outputs a "logically equivalent" $\sharp$-formula of minimum width

Idea of alg:

- Show that each $\sharp$-formula ϕ can be normalized to the form $\sum_i (\text{integer}) C(\psi_i)$ without increasing width where each ψ_i is a $\{\exists, \wedge\}$-query
- May then enforce that the ψ_i are counting inequivalent
- Then, find a min width representation of each $C(\psi_i)$

Width minimization

Thm (width minimization): There exists an alg f that, given a $\sharp$-formula where only $\{\exists, \wedge, \vee\}$-queries are casted, outputs a "logically equivalent" $\sharp$-formula of minimum width

Idea of alg:

- Show that each $\sharp$-formula ϕ can be normalized to the form $\sum_i$ (integer)$C(\psi_i)$ without increasing width where each ψ_i is a $\{\exists, \wedge\}$-query
- May then enforce that the ψ_i are counting inequivalent
- Then, find a min width representation of each $C(\psi_i)$
- But to justify this...

Independence

Thm (independence): For any "linear combination" $\sum_i a_i|\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure **D** such that $\sum_i a_i|\psi_i(\mathbf{D})| \neq 0$

Independence

Thm (independence): For any "linear combination" $\sum_i a_i|\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure **D** such that $\sum_i a_i|\psi_i(\mathbf{D})| \neq 0$

Proof idea:

Independence

Thm (independence): For any "linear combination" $\sum_i a_i |\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure $\mathbf{D}$ such that $\sum_i a_i |\psi_i(\mathbf{D})| \neq 0$

Proof idea:

- We restrict to a certain subsum

Independence

Thm (independence): For any "linear combination" $\sum_i a_i|\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure **D** such that $\sum_i a_i|\psi_i(\mathbf{D})| \neq 0$

Proof idea:

- We restrict to a certain subsum
- We view each ψ_i as equal to the product of its components, and introduce a variable X_c for each component c

Independence

Thm (independence): For any "linear combination" $\sum_i a_i|\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure **D** such that $\sum_i a_i|\psi_i(\mathbf{D})| \neq 0$

Proof idea:

- We restrict to a certain subsum
- We view each ψ_i as equal to the product of its components, and introduce a variable X_c for each component c
- We view $\sum_i a_i|\psi_i|$ as $\sum_i a_i \prod_{c \in \psi_i} X_c$, a non-zero multivariate polynomial

Independence

Thm (independence): For any "linear combination" $\sum_i a_i|\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure **D** such that $\sum_i a_i|\psi_i(\mathbf{D})| \neq 0$

Proof idea:

- We restrict to a certain subsum
- We view each ψ_i as equal to the product of its components, and introduce a variable X_c for each component c
- We view $\sum_i a_i|\psi_i|$ as $\sum_i a_i \prod_{c \in \psi_i} X_c$, a non-zero multivariate polynomial
- Invoke fact: if non-zero n-var polynomial evaluated on tuples in $T_1 \times \cdots \times T_n$ where each T_i sufficiently big, returns non-zero value on at least one tuple

Independence

Thm (independence): For any "linear combination" $\sum_i a_i|\psi_i|$ where each $a_i \neq 0$ and the ψ_i are counting inequivalent, $\{\exists, \wedge\}$, there exists a structure $\mathbf{D}$ such that $\sum_i a_i|\psi_i(\mathbf{D})| \neq 0$

Proof idea:

- We restrict to a certain subsum
- We view each ψ_i as equal to the product of its components, and introduce a variable X_c for each component c
- We view $\sum_i a_i|\psi_i|$ as $\sum_i a_i \prod_{c \in \psi_i} X_c$, a non-zero multivariate polynomial
- Invoke fact: if non-zero n-var polynomial evaluated on tuples in $T_1 \times \cdots \times T_n$ where each T_i sufficiently big, returns non-zero value on at least one tuple
- Would like to control the values of components independently

Controlling the components

Say the components $\theta_1, \ldots, \theta_n$ ($\{\exists, \wedge\}$-queries) are in play

Controlling the components

Say the components $\theta_1, \ldots, \theta_n$ ($\{\exists, \wedge\}$-queries) are in play

- Previously shown: $\exists$ struct $\mathbf{C}$ such that values $|\theta_1(\mathbf{C})|$, …, $|\theta_n(\mathbf{C})|$ all different

Controlling the components

Say the components $\theta_1, \ldots, \theta_n$ ($\{\exists, \wedge\}$-queries) are in play

- Previously shown: $\exists$ struct **C** such that values $|\theta_1(\mathbf{C})|$, …, $|\theta_n(\mathbf{C})|$ all different
- But would like: for *any* values $(t_1, \ldots, t_n) \in \mathbb{N}^n$, exists struct **D** such that $\forall i$: $|\theta_i(\mathbf{D})| = t_i$

Controlling the components

Say the components $\theta_1, \ldots, \theta_n$ ($\{\exists, \wedge\}$-queries) are in play

- Previously shown: $\exists$ struct **C** such that values $|\theta_1(\mathbf{C})|$, . . ., $|\theta_n(\mathbf{C})|$ all different
- But would like: for *any* values $(t_1, \ldots, t_n) \in \mathbb{N}^n$, exists struct **D** such that $\forall i$: $|\theta_i(\mathbf{D})| = t_i$
- Use notion of univar polynomial *p* acting on a struct **B**

Controlling the components

Say the components $\theta_1, \ldots, \theta_n$ ($\{\exists, \wedge\}$-queries) are in play

- Previously shown: $\exists$ struct **C** such that values $|\theta_1(\mathbf{C})|$, ..., $|\theta_n(\mathbf{C})|$ all different
- But would like: for *any* values $(t_1, \ldots, t_n) \in \mathbb{N}^n$, exists struct **D** such that $\forall i$: $|\theta_i(\mathbf{D})| = t_i$
- Use notion of univar polynomial p acting on a struct **B**

  Key property: for any component θ, any struct **B**, and any univar polynomial p (over $\mathbb{N}$),

$$|\theta(p(\mathbf{B}))| = p(|\theta(\mathbf{B})|)$$

Controlling the components

Say the components $\theta_1, \ldots, \theta_n$ ($\{\exists, \wedge\}$-queries) are in play

- Previously shown: $\exists$ struct $\mathbf{C}$ such that values $|\theta_1(\mathbf{C})|$, ..., $|\theta_n(\mathbf{C})|$ all different
- But would like: for *any* values $(t_1, \ldots, t_n) \in \mathbb{N}^n$, exists struct $\mathbf{D}$ such that $\forall i$: $|\theta_i(\mathbf{D})| = t_i$
- Use notion of univar polynomial p acting on a struct $\mathbf{B}$

  Key property: for any component θ, any struct $\mathbf{B}$, and any univar polynomial p (over $\mathbb{N}$),

  $$|\theta(p(\mathbf{B}))| = p(|\theta(\mathbf{B})|)$$

- To get the struct $\mathbf{D}$ as described, take a poly p such that $|\theta_i(\mathbf{C})| \mapsto t_i$, and set $\mathbf{D} = p(\mathbf{C})$

Visiting Lovász

Let's discuss structs over a signature τ;
let $\mathrm{str}[\tau]$ denote the class of finite structs over τ

Visiting Lovász

Let's discuss structs over a signature τ;
let $\mathrm{str}[\tau]$ denote the class of finite structs over τ

Def: Let $R(\mathbf{A})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{B} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Visiting Lovász

Let's discuss structs over a signature τ;
let $\mathrm{str}[\tau]$ denote the class of finite structs over τ

Def: Let $R(\mathbf{A})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{B} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Our independence theorem gives (via Chandra-Merlin):

Thm: If $\mathbf{A}_1, \ldots, \mathbf{A}_k$ pairwise non-isomorphic structs,
$R(\mathbf{A}_1), \ldots, R(\mathbf{A}_k)$ are linearly independent

Visiting Lovász

Let's discuss structs over a signature τ;
let $\mathrm{str}[\tau]$ denote the class of finite structs over τ

Def: Let $R(\mathbf{A})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{B} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Our independence theorem gives (via Chandra-Merlin):

Thm: If $\mathbf{A}_1, \ldots, \mathbf{A}_k$ pairwise non-isomorphic structs,
$R(\mathbf{A}_1), \ldots, R(\mathbf{A}_k)$ are linearly independent

Cor: $R(\mathbf{A}) = R(\mathbf{A}')$ iff $\mathbf{A}, \mathbf{A}'$ are isomorphic

Visiting Lovász

Let's discuss structs over a signature τ;
let $\mathrm{str}[\tau]$ denote the class of finite structs over τ

Def: Let $R(\mathbf{A})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{B} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Our independence theorem gives (via Chandra-Merlin):

Thm: If $\mathbf{A}_1, \ldots, \mathbf{A}_k$ pairwise non-isomorphic structs,
$R(\mathbf{A}_1), \ldots, R(\mathbf{A}_k)$ are linearly independent

Cor: $R(\mathbf{A}) = R(\mathbf{A}')$ iff $\mathbf{A}, \mathbf{A}'$ are isomorphic

Def (from Lovász '67): Let $L(\mathbf{B})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{A} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Visiting Lovász

Let's discuss structs over a signature τ;
let $\mathrm{str}[\tau]$ denote the class of finite structs over τ

Def: Let $R(\mathbf{A})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{B} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Our independence theorem gives (via Chandra-Merlin):

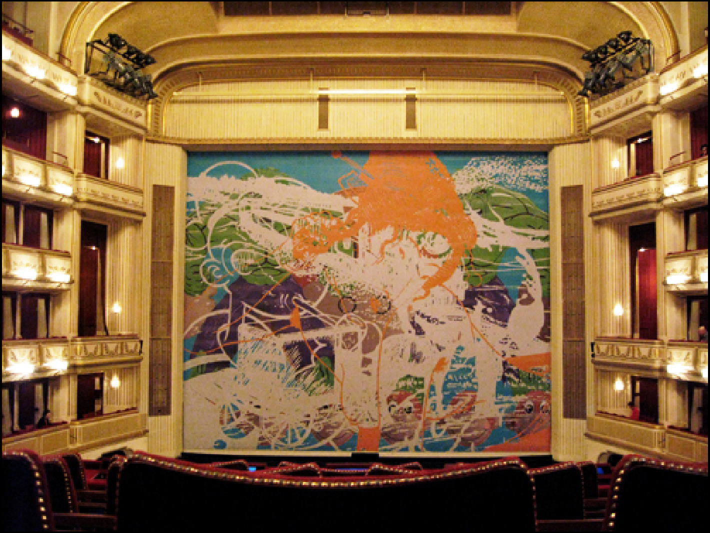
Thm: If $\mathbf{A}_1, \ldots, \mathbf{A}_k$ pairwise non-isomorphic structs,
$R(\mathbf{A}_1), \ldots, R(\mathbf{A}_k)$ are linearly independent

Cor: $R(\mathbf{A}) = R(\mathbf{A}')$ iff $\mathbf{A}, \mathbf{A}'$ are isomorphic

Def (from Lovász '67): Let $L(\mathbf{B})$ be the vector in $\mathbb{Q}^{\mathrm{str}[\tau]}$ that

maps a struct $\mathbf{A} \in \mathrm{str}[\tau]$ to the num of homoms $\mathbf{A} \to \mathbf{B}$

Thm (Lovász '67): $L(\mathbf{B}) = L(\mathbf{B}')$ iff $\mathbf{B}, \mathbf{B}'$ are isomorphic



Act: Reflection

Discussion

Discussion

Logics with counting mechanisms: previously considered in finite model theory, descriptive complexity, database theory...

Discussion

Logics with counting mechanisms: previously considered in finite model theory, descriptive complexity, database theory...

- Example — counting logic of [Immerman and Lander ’90]

Discussion

Logics with counting mechanisms: previously considered in finite model theory, descriptive complexity, database theory...

- Example — counting logic of [Immerman and Lander '90]
- Typical motivation — extend FO logic (or some logic) to capture properties not originally expressible

Discussion

Logics with counting mechanisms: previously considered in finite model theory, descriptive complexity, database theory...

- Example — counting logic of [Immerman and Lander '90]
- Typical motivation — extend FO logic (or some logic) to capture properties not originally expressible

Motivation here somewhat different; we wanted "simple" logics that allow for the direct expression of efficient algorithms

Discussion

Logics with counting mechanisms: previously considered in finite model theory, descriptive complexity, database theory...

- Example — counting logic of [Immerman and Lander '90]
- Typical motivation — extend FO logic (or some logic) to capture properties not originally expressible

Motivation here somewhat different; we wanted "simple" logics that allow for the direct expression of efficient algorithms

Our $\sharp$-logic balances...

- Expressivity
- Computability: there is an algorithm for width minimization, so width is well-characterized (in some sense) (Width minimization not computable in positive FO [Bova & Chen '14])

Open issues

Open issues

Open: Are there Ehrenfeucht-Fraïssé style games for understanding expressibility in ♯-logic?

Open issues

Open: Are there Ehrenfeucht-Fraïssé style games for understanding expressibility in $\sharp$-logic?

Open: We focused on $\{\exists, \wedge, \vee\}$-formulas;
what can one say about FO logic in general?

What can one say about other logics?