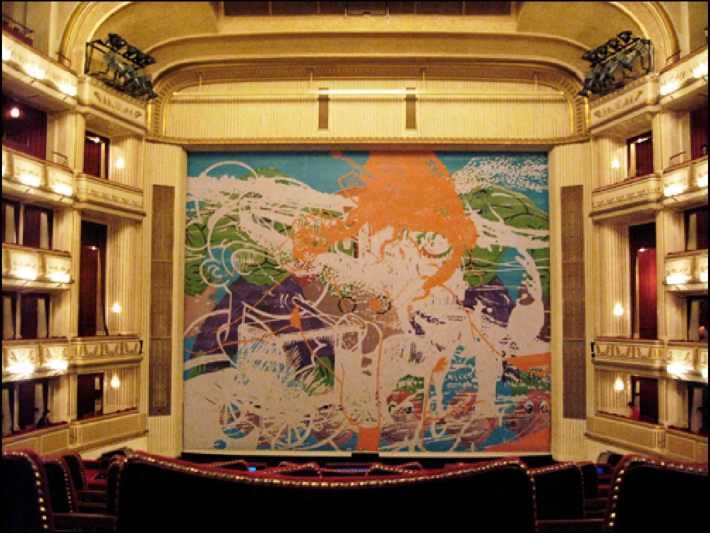
Parameter Compilation

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

TU-Wien – June 2015



Act: Motivation

Model Checking

Model Checking

Basic problem in logic/database theory:

Given a first-order sentence ϕ and a finite structure **B**,
decide if $\mathbf{B} \models \phi$

Model Checking

Basic problem in logic/database theory:

Given a first-order sentence ϕ and a finite structure $\mathbf{B}$,
decide if $\mathbf{B} \models \phi$

Example:
When $\phi_k = \exists v_1 \ldots \exists v_k \bigwedge_{i \neq j} E(v_i, v_j)$ and $\mathbf{G}$ is an undir graph,

Model Checking

Basic problem in logic/database theory:

Given a first-order sentence ϕ and a finite structure $\mathbf{B}$,
decide if $\mathbf{B} \models \phi$

Example:

When $\phi_k = \exists v_1 \ldots \exists v_k \bigwedge_{i \neq j} E(v_i, v_j)$ and $\mathbf{G}$ is an undir graph,

$\mathbf{G} \models \phi_k$ iff $\mathbf{G}$ has a k-clique

Model Checking

Basic problem in logic/database theory:

Given a first-order sentence ϕ and a finite structure **B**,
decide if $\mathbf{B} \models \phi$

Example:
When $\phi_k = \exists v_1 \ldots \exists v_k \bigwedge_{i \neq j} E(v_i, v_j)$ and **G** is an undir graph,

$\mathbf{G} \models \phi_k$ iff **G** has a k-clique

Example:
When $\phi_k = \exists v_1 \ldots \exists v_k \forall y (\bigvee_{i=1}^{k} (y = v_i \vee E(y, v_i))$
and **G** is an undir graph,

Model Checking

Basic problem in logic/database theory:

Given a first-order sentence ϕ and a finite structure **B**,
decide if $\mathbf{B} \models \phi$

Example:
When $\phi_k = \exists v_1 \ldots \exists v_k \bigwedge_{i \neq j} E(v_i, v_j)$ and **G** is an undir graph,

$\mathbf{G} \models \phi_k$ iff **G** has a k-clique

Example:
When $\phi_k = \exists v_1 \ldots \exists v_k \forall y (\bigvee_{i=1}^{k} (y = v_i \vee E(y, v_i))$
and **G** is an undir graph,

$\mathbf{G} \models \phi_k$ iff **G** has a dominating set of size $\leqslant k$

Studying complexity

Studying complexity

General problem intractable — PSPACE-complete.

Studying complexity

General problem intractable — PSPACE-complete.

Restrict to a single first-order sentence: polytime tractable.

Studying complexity

General problem intractable — PSPACE-complete.

Restrict to a single first-order sentence: polytime tractable.

Here, we restrict to a set of first-order sentences Φ.

Studying complexity

General problem intractable — PSPACE-complete.

Restrict to a single first-order sentence: polytime tractable.

Here, we restrict to a set of first-order sentences Φ.

Def: The problem $\mathrm{MC}(\Phi)$ is...

Given $\phi \in \Phi$ and a finite struct $\mathbf{B}$,
decide if $\mathbf{B} \models \phi$

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

  $\Rightarrow$ 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

  Here: the query is the parameter

Parameterized tractability

Parameterized tractability

Def: A parameterized problem is a pair (Q, κ) where

- $Q \subseteq \Sigma^*$ is a language
- $\kappa : \Sigma^* \to \Sigma^*$ is a parameterization,
  assumed here to be polytime computable

Parameterized tractability

Def: A parameterized problem is a pair (Q, κ) where

- $Q \subseteq \Sigma^*$ is a language
- $\kappa : \Sigma^* \to \Sigma^*$ is a parameterization,
  assumed here to be polytime computable

Ex: When discussing the problem MC(Φ),
we understand κ to be the projection $\kappa(\phi, \mathbf{B}) = \phi$

Parameterized tractability

Def: A parameterized problem is a pair (Q, κ) where

- $Q \subseteq \Sigma^*$ is a language
- $\kappa : \Sigma^* \to \Sigma^*$ is a parameterization, assumed here to be polytime computable

Ex: When discussing the problem $\mathrm{MC}(\Phi)$, we understand κ to be the projection $\kappa(\phi, \mathbf{B}) = \phi$

Def: A parameterized problem (Q, κ) is FPT ("tractable") if $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ and a language $Q' \in P$ such that

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Parameterized tractability

Def: A parameterized problem is a pair (Q, κ) where

- $Q \subseteq \Sigma^*$ is a language
- $\kappa : \Sigma^* \to \Sigma^*$ is a parameterization, assumed here to be polytime computable

Ex: When discussing the problem $\mathrm{MC}(\Phi)$, we understand κ to be the projection $\kappa(\phi, \mathbf{B}) = \phi$

Def: A parameterized problem (Q, κ) is FPT ("tractable") if $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ and a language $Q' \in P$ such that

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Compilation view: after applying an arbitrary compilation to the parameter, can decide in polytime

Dichotomy for existential positive logic

Dichotomy for existential positive logic

Thm (Chen '14):
Let Φ be a set of $\{\exists, \wedge, \vee\}$-sentences, of bounded arity.

- If there exists $k \geqslant 1$ such that each $\phi \in \Phi$ is logically equivalent to a k-variable sentence, then MC(Φ) is in FPT
- Else, MC(Φ) is W[1]-hard

Dichotomy for existential positive logic

Thm (Chen '14):
Let Φ be a set of $\{\exists, \wedge, \vee\}$-sentences, of bounded arity.

- If there exists $k \geqslant 1$ such that each $\phi \in \Phi$ is logically equivalent to a k-variable sentence, then MC(Φ) is in FPT
- Else, MC(Φ) is W[1]-hard

In first case, can compile each $\phi \in \Phi$ to a k-variable sentence to show FPT inclusion

Dichotomy for existential positive logic

Thm (Chen '14):
Let Φ be a set of $\{\exists, \wedge, \vee\}$-sentences, of bounded arity.

- If there exists $k \geqslant 1$ such that each $\phi \in \Phi$ is logically equivalent to a k-variable sentence, then $\text{MC}(\Phi)$ is in FPT
- Else, $\text{MC}(\Phi)$ is W[1]-hard

In first case, can compile each $\phi \in \Phi$ to a k-variable sentence to show FPT inclusion

Example of first case: define Φ to contain each $\{\exists, \wedge, \vee\}$-sentence over a unary signature;
let us use unary-EP-MC to denote $\text{MC}(\Phi)$

The length of compilation

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure $\mathbf{B}$, decide if $\mathbf{B} \models \phi$

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure **B**, decide if **B** $\models \phi$

- Problem is NP-complete

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure **B**, decide if $\mathbf{B} \models \phi$

- Problem is NP-complete
- But at the same time, problem is FPT:
  $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ & language $Q' \in P$ such that

$$(\phi, \mathbf{B}) \in Q \text{ iff } (c(\phi), (\phi, \mathbf{B})) \in Q'$$

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure $\mathbf{B}$, decide if $\mathbf{B} \models \phi$

- Problem is NP-complete
- But at the same time, problem is FPT:
  $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ & language $Q' \in P$ such that
  $$(\phi, \mathbf{B}) \in Q \text{ iff } (c(\phi), (\phi, \mathbf{B})) \in Q'$$

We can infer that there is no polytime computable c (otherwise would have unary-EP-MC in PTIME)

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure **B**, decide if $\mathbf{B} \models \phi$

- Problem is NP-complete
- But at the same time, problem is FPT:
  $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ & language $Q' \in P$ such that

$$(\phi, \mathbf{B}) \in Q \text{ iff } (c(\phi), (\phi, \mathbf{B})) \in Q'$$

We can infer that there is no polytime computable c (otherwise would have unary-EP-MC in PTIME)

But, why not? Two potential explanations:

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure $\mathbf{B}$, decide if $\mathbf{B} \models \phi$

- Problem is NP-complete
- But at the same time, problem is FPT:
  $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ & language $Q' \in P$ such that
  $$(\phi, \mathbf{B}) \in Q \text{ iff } (c(\phi), (\phi, \mathbf{B})) \in Q'$$

We can infer that there is no polytime computable c (otherwise would have unary-EP-MC in PTIME)

But, why not? Two potential explanations:

- For any c (satisfying above), c is not polynomial length

The length of compilation

Problem unary-EP-MC: given EP ϕ over a unary signature and structure **B**, decide if $\mathbf{B} \models \phi$

- Problem is NP-complete
- But at the same time, problem is FPT:
  $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ & language $Q' \in P$ such that
  $$(\phi, \mathbf{B}) \in Q \text{ iff } (c(\phi), (\phi, \mathbf{B})) \in Q'$$

We can infer that there is no polytime computable c (otherwise would have unary-EP-MC in PTIME)

But, why not? Two potential explanations:

- For any c (satisfying above), c is not polynomial length
- There exists a c (satisfying above) of polynomial length, but not polytime computable

The length of compilation

The length of compilation

Reason: there is no compilation *c* of polynomial length (proved in Chen '14)

The length of compilation

Reason: there is no compilation c of polynomial length (proved in Chen '14)

We say that a fn $c : \Sigma^* \to \Sigma^*$ has *polynomial length* if $\exists$ a poly p such that, for all $x \in \Sigma^*$: $|c(x)| \leqslant p(|x|)$

The length of compilation

Reason: there is no compilation c of polynomial length (proved in Chen '14)

We say that a fn $c : \Sigma^* \to \Sigma^*$ has *polynomial length* if $\exists$ a poly p such that, for all $x \in \Sigma^*$: $|c(x)| \leqslant p(|x|)$

Here:

- Think of poly length compilation as positive result

The length of compilation

Reason: there is no compilation c of polynomial length (proved in Chen '14)

We say that a fn $c : \Sigma^* \to \Sigma^*$ has *polynomial length* if $\exists$ a poly p such that, for all $x \in \Sigma^*$: $|c(x)| \leqslant p(|x|)$

Here:

- Think of poly length compilation as positive result
- Will give a framework where we can prove negative results — superpoly *length* lower bounds on compilations

The length of compilation

Reason: there is no compilation c of polynomial length (proved in Chen '14)

We say that a fn $c : \Sigma^* \to \Sigma^*$ has *polynomial length* if $\exists$ a poly p such that, for all $x \in \Sigma^*$: $|c(x)| \leqslant p(|x|)$

Here:

- Think of poly length compilation as positive result
- Will give a framework where we can prove negative results — superpoly *length* lower bounds on compilations

Please note:

The length of compilation

Reason: there is no compilation c of polynomial length (proved in Chen '14)

We say that a fn $c : \Sigma^* \to \Sigma^*$ has *polynomial length* if $\exists$ a poly p such that, for all $x \in \Sigma^*$: $|c(x)| \leqslant p(|x|)$

Here:

- Think of poly length compilation as positive result
- Will give a framework where we can prove negative results — superpoly *length* lower bounds on compilations

Please note:

- (Chen '05) “Parameterized compilability” — relax the notion of positive result; let c be “FPT-length”

The length of compilation

Reason: there is no compilation c of polynomial length (proved in Chen '14)

We say that a fn $c : \Sigma^* \to \Sigma^*$ has *polynomial length* if $\exists$ a poly p such that, for all $x \in \Sigma^*$: $|c(x)| \leqslant p(|x|)$

Here:

- Think of poly length compilation as positive result
- Will give a framework where we can prove negative results — superpoly *length* lower bounds on compilations

Please note:

- (Chen '05) "Parameterized compilability" — relax the notion of positive result; let c be "FPT-length"
- (Chen '15) "Parameter compilation" — framework for distinguishing between polynomial / non-polynomial length compilations

Act: Parameter compilation

Framework

Framework

Contribution: Give framework for understanding length of compilations — so that, post-compilation, can solve in polytime

Framework

Contribution: Give framework for understanding length of compilations — so that, post-compilation, can solve in polytime

Inspired by and closely related to framework
by Cadoli, Donini, Liberatore & Schaerf ’02
— see our paper for more details/discussion

Framework — motivation

For a problem, if many instances share a feature in common, it may be fruitful to compile this feature into a format that allows for faster decision

Framework — motivation

For a problem, if many instances share a feature in common, it may be fruitful to compile this feature into a format that allows for faster decision

Examples:

- Deciding connectivity of vertex pairs in graphs:
  If many instances may share the same graph G,
  may wish to compile G

Framework — motivation

For a problem, if many instances share a feature in common, it may be fruitful to compile this feature into a format that allows for faster decision

Examples:

- Deciding connectivity of vertex pairs in graphs:
  If many instances may share the same graph G,
  may wish to compile G

- Model checking / query evaluation:
  If a query ϕ will be posed to many databases,
  may wish to compile ϕ

The base class

The base class

Def: A parameterized problem (Q, κ) is in FPT if $\exists$ computable fn $c : \Sigma^* \rightarrow \Sigma^*$ and a language $Q' \in P$ such that

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Compilation view: after applying an arbitrary compilation to the parameter, can decide in polytime

The base class

Def: A parameterized problem (Q, κ) is in FPT
if $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ and a language $Q' \in P$ such that

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Compilation view: after applying an arbitrary compilation to the parameter, can decide in polytime

Def: A parameterized problem (Q, κ) is in poly-comp-PTIME
if c can be taken to be polynomial length
("poly-length compilable to PTIME")

The base class

Def: A parameterized problem (Q, κ) is in FPT
if $\exists$ computable fn $c : \Sigma^* \to \Sigma^*$ and a language $Q' \in P$ such that

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Compilation view: after applying an arbitrary compilation to the parameter, can decide in polytime

Def: A parameterized problem (Q, κ) is in poly-comp-PTIME
if c can be taken to be polynomial length
("poly-length compilable to PTIME")

Idea: Q is decidable in polytime (in $|x|$),
modulo knowledge of $c(\kappa(x))$ — slice-wise advice

Reduction

Reduction

Def: Param problem (Q, κ) *poly-comp reduces* to param problem (Q', κ') if exists:

- $g(x) = f(c(\kappa(x)), x)$ with polytime computable f, poly-length computable c
- poly-length computable $s : \Sigma^* \to \wp(\Sigma^*)$

such that

- $x \in Q \Leftrightarrow g(x) \in Q'$
- $\kappa'(g(x)) \in s(\kappa(x))$

Reduction

Def: Param problem (Q, κ) *poly-comp reduces* to param problem (Q', κ') if exists:

- $g(x) = f(c(\kappa(x)), x)$ with polytime computable f, poly-length computable c
- poly-length computable $s : \Sigma^* \to \wp(\Sigma^*)$

such that

- $x \in Q \Leftrightarrow g(x) \in Q'$
- $\kappa'(g(x)) \in s(\kappa(x))$

Note that $s(x)$ must be a set of size poly in $|x|$

Reduction

Def: Param problem (Q, κ) *poly-comp reduces* to param problem (Q', κ') if exists:

- $g(x) = f(c(\kappa(x)), x)$ with polytime computable f, poly-length computable c
- poly-length computable $s : \Sigma^* \to \wp(\Sigma^*)$

such that

- $x \in Q \Leftrightarrow g(x) \in Q'$
- $\kappa'(g(x)) \in s(\kappa(x))$

Note that $s(x)$ must be a set of size poly in $|x|$

Fact: This is a restricted version of FPT many-one reduction!

Reduction

Def: Param problem (Q, κ) *poly-comp reduces* to param problem (Q', κ') if exists:

- $g(x) = f(c(\kappa(x)), x)$ with polytime computable f, poly-length computable c
- poly-length computable $s : \Sigma^* \to \wp(\Sigma^*)$

such that

- $x \in Q \Leftrightarrow g(x) \in Q'$
- $\kappa'(g(x)) \in s(\kappa(x))$

Note that $s(x)$ must be a set of size poly in $|x|$

Fact: This is a restricted version of FPT many-one reduction!

Prop: poly-comp-PTIME is closed under poly-comp reduction

[(Q, κ) reduces to $(Q', \kappa') \in$ poly-comp-PTIME implies $(Q, \kappa) \in$ poly-comp-PTIME]

Chopped classes

Chopped classes

Def: A param problem (Q, κ) is in chopped-$\mathcal{C}$ if
$\exists$ polytime computable f, poly-length computable c
where $g(x) = f(c(\kappa(x)), x)$ has:

- $x \in Q$ iff $g(x) \in Q'$ where $Q' \in \mathcal{C}$
- $|g(x)| \leqslant p(|\kappa(x)|)$ for a polynomial p

Chopped classes

Def: A param problem (Q, κ) is in chopped-$\mathcal{C}$ if
$\exists$ polytime computable f, poly-length computable c
where $g(x) = f(c(\kappa(x)), x)$ has:

- $x \in Q$ iff $g(x) \in Q'$ where $Q' \in \mathcal{C}$
- $|g(x)| \leqslant p(|\kappa(x)|)$ for a polynomial p

Example problem: Minimal model checking

$$\{(\phi, y) \mid \ \phi \text{ is a prop formula, } y \text{ is a minimal model of } \phi\ \}$$

Chopped classes

Def: A param problem (Q, κ) is in chopped-$\mathcal{C}$ if
$\exists$ polytime computable f, poly-length computable c
where $g(x) = f(c(\kappa(x)), x)$ has:

- $x \in Q$ iff $g(x) \in Q'$ where $Q' \in \mathcal{C}$
- $|g(x)| \leqslant p(|\kappa(x)|)$ for a polynomial p

Example problem: Minimal model checking

$$\{(\phi, y) \mid \phi \text{ is a prop formula, } y \text{ is a minimal model of } \phi\}$$

- In coNP

Chopped classes

Def: A param problem (Q, κ) is in chopped-$\mathcal{C}$ if
$\exists$ polytime computable f, poly-length computable c
where $g(x) = f(c(\kappa(x)), x)$ has:

- $x \in Q$ iff $g(x) \in Q'$ where $Q' \in \mathcal{C}$
- $|g(x)| \leqslant p(|\kappa(x)|)$ for a polynomial p

Example problem: Minimal model checking

$$\{(\phi, y) \mid \phi \text{ is a prop formula, } y \text{ is a minimal model of } \phi\}$$

- In coNP
- In chopped-coNP under $\kappa(\phi, y) = \phi$:

Chopped classes

Def: A param problem (Q, κ) is in chopped-$\mathcal{C}$ if
$\exists$ polytime computable f, poly-length computable c
where $g(x) = f(c(\kappa(x)), x)$ has:

- $x \in Q$ iff $g(x) \in Q'$ where $Q' \in \mathcal{C}$
- $|g(x)| \leqslant p(|\kappa(x)|)$ for a polynomial p

Example problem: Minimal model checking

$$\{(\phi, y) \mid \phi \text{ is a prop formula, } y \text{ is a minimal model of } \phi\}$$

- In coNP
- In chopped-coNP under $\kappa(\phi, y) = \phi$:
  Take $g(\phi, y) = (\phi, y)$ if y is an assignment to vars of ϕ,
  a *no* instance otherwise

Chopped classes — facts

Fact: poly-comp-PTIME $=$ chopped-PTIME

Chopped classes — facts

Fact: poly-comp-PTIME $=$ chopped-PTIME

Thm: chopped-NP is not contained in poly-comp-PTIME, unless the PH collapses.

Chopped classes — facts

Fact: poly-comp-PTIME $=$ chopped-PTIME

Thm: chopped-NP is not contained in poly-comp-PTIME, unless the PH collapses.

Follows from Karp-Lipton plus...

Thm: If chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$, then $\mathcal{C} \subseteq \mathcal{C}'/$poly

Chopped classes — facts

Fact: poly-comp-PTIME $=$ chopped-PTIME

Thm: chopped-NP is not contained in poly-comp-PTIME, unless the PH collapses.

Follows from Karp-Lipton plus...

Thm: If chopped-$\mathcal{C} \subseteq$ chopped-$\mathcal{C}'$, then $\mathcal{C} \subseteq \mathcal{C}'/$poly

Note that the chopped classes stratify FPT...

Prop: If each lang in $\mathcal{C}$ is computable, then chopped-$\mathcal{C}$ is in FPT

Completeness

Completeness

Prop: Let $\mathcal{C}$ be a complexity class; assume that Q is $\mathcal{C}$-complete under many-one polytime reduction. Then, (Q, len) is complete for chopped-$\mathcal{C}$.

Here, len is the parameterization $\text{len}(x) = 1^n$ giving the length of a string, in unary

Completeness: examples

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G)$ = number of nodes in graph G

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G)$ = number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F)$ = number of variables in F

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G) =$ number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F) =$ number of variables in F
3. (CIRCUIT-SAT, $\mu + \nu$)
   where $(\mu + \nu)(C)$ = total number of gates in C

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G) =$ number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F) =$ number of variables in F
3. (CIRCUIT-SAT, $\mu + \nu$)
   where $(\mu + \nu)(C)$ = total number of gates in C
4. (d-HITTING SET, π_2), for each $d \geqslant 2$
   where $\pi_2(H, k) = k$

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G)$ = number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F)$ = number of variables in F
3. (CIRCUIT-SAT, $\mu + \nu$)
   where $(\mu + \nu)(C)$ = total number of gates in C
4. (d-HITTING SET, π_2), for each $d \geqslant 2$
   where $\pi_2(H, k) = k$

Here, d-HITTING SET is the problem of deciding, given (H, k) where H is a hypergraph where each edge has size $\leqslant d$, if there's a hitting set of size $\leqslant k$

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G)$ = number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F)$ = number of variables in F
3. (CIRCUIT-SAT, $\mu + \nu$)
   where $(\mu + \nu)(C)$ = total number of gates in C
4. (d-HITTING SET, π_2), for each $d \geqslant 2$
   where $\pi_2(H, k) = k$

Here, d-HITTING SET is the problem of deciding, given (H, k) where H is a hypergraph where each edge has size $\leqslant d$, if there's a hitting set of size $\leqslant k$

Note: Can show

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G)$ = number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F)$ = number of variables in F
3. (CIRCUIT-SAT, $\mu + \nu$)
   where $(\mu + \nu)(C)$ = total number of gates in C
4. (d-HITTING SET, π_2), for each $d \geqslant 2$
   where $\pi_2(H, k) = k$

Here, d-HITTING SET is the problem of deciding, given (H, k) where H is a hypergraph where each edge has size $\leqslant d$, if there's a hitting set of size $\leqslant k$

Note: Can show

- unary-EP-MC is chopped-NP-hard

Completeness: examples

Prop: The following problems are chopped-NP-complete:

1. (Hamiltonian Path, γ)
   where $\gamma(G)$ = number of nodes in graph G
2. (3-SAT, ν)
   where $\nu(F)$ = number of variables in F
3. (CIRCUIT-SAT, $\mu + \nu$)
   where $(\mu + \nu)(C)$ = total number of gates in C
4. (d-HITTING SET, π_2), for each $d \geqslant 2$
   where $\pi_2(H, k) = k$

Here, d-HITTING SET is the problem of deciding, given (H, k) where H is a hypergraph where each edge has size $\leqslant d$, if there's a hitting set of size $\leqslant k$

Note: Can show

- unary-EP-MC is chopped-NP-hard
- Minimal model checking is chopped-co-NP-complete

Kernelization

In the first three problems just given, if the parameter is bounded, the number of problem instances is bounded

Kernelization

In the first three problems just given, if the parameter is bounded, the number of problem instances is bounded

Example: in 3-SAT, if the number n of variables is bounded, problem instances can only talk about 3-clauses on n variables

Kernelization

In the first three problems just given, if the parameter is bounded, the number of problem instances is bounded

Example: in 3-SAT, if the number n of variables is bounded, problem instances can only talk about 3-clauses on n variables

In the d-HITTING SET problem, this is not true:
for a fixed k, hypergraphs H may have arbitrary size

Kernelization

In the first three problems just given, if the parameter is bounded, the number of problem instances is bounded

Example: in 3-SAT, if the number n of variables is bounded, problem instances can only talk about 3-clauses on n variables

In the d-HITTING SET problem, this is not true:
for a fixed k, hypergraphs H may have arbitrary size

But this problem is in chopped-NP, due to...

Kernelization

In the first three problems just given, if the parameter is bounded, the number of problem instances is bounded

Example: in 3-SAT, if the number n of variables is bounded, problem instances can only talk about 3-clauses on n variables

In the d-HITTING SET problem, this is not true:
for a fixed k, hypergraphs H may have arbitrary size

But this problem is in chopped-NP, due to...

Def: A param problem (Q, κ) has a polynomial kernelization if $\exists$ polytime computable $K : \Sigma^* \to \Sigma^*$, polynomial p such that

$$x \in Q \Leftrightarrow K(x) \in Q \quad \text{and} \quad |K(x)| \leqslant p(|\kappa(x)|)$$

Kernelization

In the first three problems just given, if the parameter is bounded, the number of problem instances is bounded

Example: in 3-SAT, if the number n of variables is bounded, problem instances can only talk about 3-clauses on n variables

In the d-HITTING SET problem, this is not true:
for a fixed k, hypergraphs H may have arbitrary size

But this problem is in chopped-NP, due to...

Def: A param problem (Q, κ) has a polynomial kernelization if $\exists$ polytime computable $K : \Sigma^* \to \Sigma^*$, polynomial p such that

$$x \in Q \Leftrightarrow K(x) \in Q \quad \text{and} \quad |K(x)| \leqslant p(|\kappa(x)|)$$

Prop: If a param problem (Q, κ) with $Q \in \mathrm{NP}$ has a polynomial kernelization, then (Q, κ) is in chopped-NP

Wrap-up

Wrap-up

We initiated a theory of compilability that makes use of notions/concepts from parameterized complexity
— with connections to classical notions such as FPT, kernelization, ...

Wrap-up

We initiated a theory of compilability that makes use of notions/concepts from parameterized complexity
— with connections to classical notions such as FPT, kernelization, ...

For the future:

- Try to classify problems of interest / established parameterized problems according to their compilability

Wrap-up

We initiated a theory of compilability that makes use of notions/concepts from parameterized complexity
— with connections to classical notions such as FPT, kernelization, ...

For the future:

- Try to classify problems of interest / established parameterized problems according to their compilability
- What can we say about color coding (embedding under bounded treewidth)?

A closing meditation

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

The study of when param problems have
polynomial kernelizations (and how good these can be)
led/lead to a rich, deep body of work

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

The study of when param problems have polynomial kernelizations (and how good these can be) led/lead to a rich, deep body of work

Characterization: (Q, κ) is in FPT iff it can be “compiled” to a PTIME language Q', via a computable c, so that:

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

The study of when param problems have polynomial kernelizations (and how good these can be) led/lead to a rich, deep body of work

Characterization: (Q, κ) is in FPT iff it can be “compiled” to a PTIME language Q', via a computable c, so that:

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Here we initiated a theory for understanding when we have polynomial-length compilations

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

The study of when param problems have
polynomial kernelizations (and how good these can be)
led/lead to a rich, deep body of work

Characterization: (Q, κ) is in FPT iff it can be "compiled" to a PTIME language Q', via a computable c, so that:

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Here we initiated a theory for understanding when we have
polynomial-length compilations
Is there an entire area to be discovered here?

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

The study of when param problems have
polynomial kernelizations (and how good these can be)
led/lead to a rich, deep body of work

Characterization: (Q, κ) is in FPT iff it can be "compiled" to a PTIME language Q', via a computable c, so that:

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Here we initiated a theory for understanding when we have
polynomial-length compilations
Is there an entire area to be discovered here?

"Kernelization is just one technique in parameterized complexity and its systematic study opened up a whole new world of research questions.

A closing meditation

Many characterizations of FPT (see e.g. Flum/Grohe, Chap 1)

Characterization: (Q, κ) in FPT iff (Q, κ) has a kernelization

The study of when param problems have
polynomial kernelizations (and how good these can be)
led/lead to a rich, deep body of work

Characterization: (Q, κ) is in FPT iff it can be "compiled" to a PTIME language Q', via a computable c, so that:

$$x \in Q \text{ iff } (c(\kappa(x)), x) \in Q'$$

Here we initiated a theory for understanding when we have
polynomial-length compilations
Is there an entire area to be discovered here?

"Kernelization is just one technique in parameterized complexity and its systematic study opened up a whole new world of research questions. Could it be that exploring other basic techniques turns out to be as fruitful as the study of kernelization?" — Dániel Marx, '12 survey