Databases and Despriptive Complexity — Part 1:

Using Logical Formulas to Describe Computations

Nicole Schweikardt

Humboldt-Universität zu Berlin

EPIT 2019 — Spring School on Theoretical Computer Science:
Databases, logic and automata

Luminy, 11 April 2019

Overview

Descriptive Complexity

Datalog is poorly expressive

Datalog is highly expressive

Overview

Descriptive Complexity

Datalog is poorly expressive

Datalog is highly expressive

Throughout this talk

- all graphs are finite and directed
- undirected graphs are modeled as directed graphs where an undirected edge $u — v$ is represented by the directed edges $u \rightarrow v$ and $u \leftarrow v$.

Throughout this talk

- all graphs are finite and directed
- undirected graphs are modeled as directed graphs where an undirected edge $u — v$ is represented by the directed edges $u \rightarrow v$ and $u \leftarrow v$.
- graphs $G = (V^G, E^G)$ are represented by databases $\mathbf{I}_G$ of schema $\{V, E\}$ where V and E are interpreted by V^G and E^G, respectively

Throughout this talk

- all graphs are finite and directed
- undirected graphs are modeled as directed graphs where an undirected edge $u — v$ is represented by the directed edges $u \to v$ and $u \leftarrow v$.
- graphs $G = (V^G, E^G)$ are represented by databases $\mathbf{I}_G$ of schema $\{V, E\}$ where V and E are interpreted by V^G and E^G, respectively
- p is a graph property, if for all graphs G, H we have:

  if $G \cong H$, then G has property $p \iff H$ has property p

Throughout this talk

- all graphs are finite and directed
- undirected graphs are modeled as directed graphs where an undirected edge $u — v$ is represented by the directed edges $u \to v$ and $u \leftarrow v$.
- graphs $G = (V^G, E^G)$ are represented by databases $\mathbf{I}_G$ of schema $\{V, E\}$ where V and E are interpreted by V^G and E^G, respectively
- p is a graph property, if for all graphs G, H we have:

  if $G \cong H$, then $\quad G$ has property $p \iff H$ has property p
- An ordered graph G is of the form $(V^G, E^G, <^G)$ where (V^G, E^G) is a graph and $<^G$ is a strict linear order on V^G.

Throughout this talk

- all graphs are finite and directed
- undirected graphs are modeled as directed graphs where an undirected edge $u — v$ is represented by the directed edges $u \to v$ and $u \leftarrow v$.
- graphs $G = (V^G, E^G)$ are represented by databases $\mathbf{I}_G$ of schema $\{V, E\}$ where V and E are interpreted by V^G and E^G, respectively
- p is a graph property, if for all graphs G, H we have:

  if $G \cong H$, then G has property $p \iff H$ has property p
- An ordered graph G is of the form $(V^G, E^G, <^G)$ where (V^G, E^G) is a graph and $<^G$ is a strict linear order on V^G.
- p is a property of ordered graphs, if for all ordered graphs G, H we have:

  if $G \cong H$, then G has property $p \iff H$ has property p

Seminal Results in Descriptive Complexity (1/2)

ESO : existential second-order logic : $\exists R_1 \cdots \exists R_\ell \; \underbrace{\psi(E, R_1, \ldots, R_\ell)}_{\in FO}$

Fagin's Theorem: NP is captured by ESO on graphs.

Seminal Results in Descriptive Complexity (1/2)

ESO : existential second-order logic : $\exists R_1 \cdots \exists R_\ell \underbrace{\psi(E, R_1, \ldots, R_\ell)}_{\in FO}$

Fagin's Theorem: NP is captured by ESO on graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E\}$, upon input of a graph $G = (V^G, E^G)$ it can be decided in nondeterministic polynomial time whether $G \models \varphi$.

Seminal Results in Descriptive Complexity (1/2)

ESO : existential second-order logic : $\exists R_1 \cdots \exists R_\ell \underbrace{\psi(E, R_1, \ldots, R_\ell)}_{\in FO}$

Fagin's Theorem: NP is captured by ESO on graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E\}$, upon input of a graph $G = (V^G, E^G)$ it can be decided in nondeterministic polynomial time whether $G \models \varphi$.

The data complexity of model-checking for ESO-sentences is in NP.

Seminal Results in Descriptive Complexity (1/2)

ESO : existential second-order logic : $\exists R_1 \cdots \exists R_\ell \underbrace{\psi(E, R_1, \ldots, R_\ell)}_{\in FO}$

Fagin's Theorem: NP is captured by ESO on graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E\}$, upon input of a graph $G = (V^G, E^G)$ it can be decided in nondeterministic polynomial time whether $G \models \varphi$.

The data complexity of model-checking for ESO-sentences is in NP.

(2) For every property p of graphs that is decidable in NP, there exists an ESO-sentence φ of signature $\{E\}$ such that for all graphs G we have: $G \models \varphi \iff G$ has property p.

Seminal Results in Descriptive Complexity (1/2)

ESO : existential second-order logic : $\exists R_1 \cdots \exists R_\ell \underbrace{\psi(E, R_1, \ldots, R_\ell)}_{\in FO}$

Fagin's Theorem: NP is captured by ESO on graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E\}$, upon input of a graph $G = (V^G, E^G)$ it can be decided in nondeterministic polynomial time whether $G \models \varphi$.

The data complexity of model-checking for ESO-sentences is in NP.

(2) For every property p of graphs that is decidable in NP, there exists an ESO-sentence φ of signature $\{E\}$ such that for all graphs G we have: $G \models \varphi \iff G$ has property p.

Every NP-property of graphs can be described by an ESO-sentence.

Seminal Results in Descriptive Complexity (2/2)

LPF : least fixed-point logic : extends FO by the ability to define relations inductively

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

Seminal Results in Descriptive Complexity (2/2)

LPF : least fixed-point logic : extends FO by the ability to define relations inductively

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E, <\}$, upon input of an ordered graph $G = (V^G, E^G, <^G)$ it can be decided in polynomial time whether $G \models \varphi$.

The data complexity of model-checking for LFP-sentences is in PTIME.

Seminal Results in Descriptive Complexity (2/2)

LPF : least fixed-point logic : extends FO by the ability to define relations inductively

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E, <\}$, upon input of an ordered graph $G = (V^G, E^G, <^G)$ it can be decided in polynomial time whether $G \models \varphi$.

The data complexity of model-checking for LFP-sentences is in PTIME.

(2) For every property p of ordered graphs that is decidable in PTIME, there exists an LFP-sentence φ of signature $\{E, <\}$ such that for all ordered graphs G we have: $G \models \varphi \iff G$ has property p.

Every PTIME-property of ordered graphs can be described by an LFP-sentence.

Seminal Results in Descriptive Complexity (2/2)

LPF : least fixed-point logic : extends FO by the ability to define relations inductively

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

This means:

(1) For every fixed ESO-sentence φ of signature $\{E, <\}$, upon input of an ordered graph $G = (V^G, E^G, <^G)$ it can be decided in polynomial time whether $G \models \varphi$.

The data complexity of model-checking for LFP-sentences is in PTIME.

(2) For every property p of ordered graphs that is decidable in PTIME, there exists an LFP-sentence φ of signature $\{E, <\}$ such that for all ordered graphs G we have: $G \models \varphi \iff G$ has property p.

Every PTIME-property of ordered graphs can be described by an LFP-sentence.

Later on in this talk, we will prove a variant of the Immerman-Vardi Theorem for Datalog rather than LFP.

The Quest for a Logic Capturing PTIME (1/3)

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

Major open research question: [Chandra & Harel 1982; Gurevich 1988]
Is there a logic L (instead of LFP) such that the Immerman-Vardi theorem can be generalized to arbitrary graphs? I.e.:

Is there a logic L such that PTIME is captured by L on graphs?

The Quest for a Logic Capturing PTIME (1/3)

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

Major open research question: [Chandra & Harel 1982; Gurevich 1988]
Is there a logic L (instead of LFP) such that the Immerman-Vardi theorem can be generalized to arbitrary graphs? I.e.:

Is there a logic L such that PTIME is captured by L on graphs?

Such a logic L would be a great query language: It is guaranteed that

- all queries described by a user can be evaluated in PTIME, and
- all tractable queries can be formulated in the language.

The Quest for a Logic Capturing PTIME (1/3)

Immerman-Vardi Theorem: PTIME is captured by LFP on ordered graphs.

Major open research question: [Chandra & Harel 1982; Gurevich 1988]
Is there a logic L (instead of LFP) such that the Immerman-Vardi theorem can be generalized to arbitrary graphs? I.e.:

Is there a logic L such that PTIME is captured by L on graphs?

Such a logic L would be a great query language: It is guaranteed that

- all queries described by a user can be evaluated in PTIME, and
- all tractable queries can be formulated in the language.

In order to really get this, the notions of "logic" and "capturing PTIME" have to be defined very carefully:

The Quest for a Logic Capturing PTIME (2/3)

- An abstract logic L consists of
  - a set of L[σ]-sentences for each signature σ, and
  - a mapping that associates a property p_φ of σ-structures with each L[σ]-sentence φ.

For every σ-structure G we write $G \models_{\mathrm{L}} \varphi \quad :\Longleftrightarrow \quad G \in p_\varphi$.

The Quest for a Logic Capturing PTIME (2/3)

- An abstract logic L consists of
  - a set of L[σ]-sentences for each signature σ, and
  - a mapping that associates a property p_φ of σ-structures with each L[σ]-sentence φ.

  For every σ-structure G we write $G \models_{\mathsf{L}} \varphi \quad :\Longleftrightarrow \quad G \in p_\varphi$.

- An abstract logic L captures PTIME on graphs if the following 3 conditions are satisfied for the signature $\sigma = \{E\}$:
  1. The set of L[σ]-sentences is decidable.

     It can be decided if the user's input is an admissible query.

The Quest for a Logic Capturing PTIME (2/3)

- An abstract logic L consists of
  - a set of L[σ]-sentences for each signature σ, and
  - a mapping that associates a property p_φ of σ-structures with each L[σ]-sentence φ.

  For every σ-structure G we write $G \models_{\mathrm{L}} \varphi \;:\Longleftrightarrow\; G \in p_\varphi$.

- An abstract logic L captures PTIME on graphs if the following 3 conditions are satisfied for the signature $\sigma = \{E\}$:
  1. The set of L[σ]-sentences is decidable.
     It can be decided if the user's input is an admissible query.
  2. There is an algorithm $\mathbb{B}$ that associates with every sentence $\varphi \in \mathrm{L}[\sigma]$ a PTIME-algorithm $\mathbb{A}_\varphi$ that decides p_φ — i.e., upon input of a graph G, $\mathbb{A}_\varphi$ decides in PTIME whether $G \models_{\mathrm{L}} \varphi$.
     $\mathbb{B}$ is the query optimizer, which produces the query evaluation plan $\mathbb{A}_\varphi$

The Quest for a Logic Capturing PTIME (2/3)

- An abstract logic L consists of
  - a set of L[σ]-sentences for each signature σ, and
  - a mapping that associates a property p_φ of σ-structures with each L[σ]-sentence φ.

  For every σ-structure G we write $G \models_L \varphi \quad :\Longleftrightarrow \quad G \in p_\varphi$.

- An abstract logic L captures PTIME on graphs if the following 3 conditions are satisfied for the signature $\sigma = \{E\}$:
  1. The set of L[σ]-sentences is decidable.
     It can be decided if the user's input is an admissible query.
  2. There is an algorithm $\mathbb{B}$ that associates with every sentence $\varphi \in$ L[σ] a PTIME-algorithm $\mathbb{A}_\varphi$ that decides p_φ — i.e., upon input of a graph G, $\mathbb{A}_\varphi$ decides in PTIME whether $G \models_L \varphi$.
     $\mathbb{B}$ is the query optimizer, which produces the query evaluation plan $\mathbb{A}_\varphi$
  3. For every PTIME-algorithm $\mathbb{A}$ that decides a graph property, there is a sentence $\varphi \in$ L[σ] such that for every graph G we have: $G \models_L \varphi \iff$ $\mathbb{A}$ accepts G. All PTIME graph properties can be expressed in L[σ].

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) ??? satisfies conditions 1 & 2, but not condition 3

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) ??? satisfies conditions 2 & 3, but not condition 1

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines.

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines. Thus there are only countably many PTIME computable graph properties;

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines. Thus there are only countably many PTIME computable graph properties; let $p_0, p_1, p_2, \ldots$ be a list of all these.

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines. Thus there are only countably many PTIME computable graph properties; let $p_0, p_1, p_2, \ldots$ be a list of all these.
  Note: We don't require this list to be recursively enumerable!

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines. Thus there are only countably many PTIME computable graph properties; let $p_0, p_1, p_2, \ldots$ be a list of all these.
  Note: We don't require this list to be recursively enumerable!
- Syntax: $\mathrm{L}[\sigma] := \{0, 1, 2, \ldots\} = \mathbb{N}$. $\mathrm{L}[\sigma]$ is decidable. I.e., condition 1 is met.

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines. Thus there are only countably many PTIME computable graph properties; let $p_0, p_1, p_2, \ldots$ be a list of all these.
  Note: We don't require this list to be recursively enumerable!
- Syntax: $\mathsf{L}[\sigma] := \{0, 1, 2, \ldots\} = \mathbb{N}$. $\mathsf{L}[\sigma]$ is decidable. I.e., condition 1 is met.
- Semantics: for each $n \in \mathsf{L}[\sigma]$ and each graph G let
  $G \models_{\mathsf{L}} n \;:\Longleftrightarrow\; G$ has property p_n.
  All PTIME properties of graphs are expressible. I.e., condition 3 is met.

The Quest for a Logic Capturing PTIME (3/3)

Each of the 3 requirements is crucial:

(3) LFP satisfies conditions 1 & 2, but not condition 3

(1) order-invariant LFP satisfies conditions 2 & 3, but not condition 1

(2) The following abstract logic L satisfies conditions 1 & 3, but not condition 2:

- There are only countably many Turing machines. Thus there are only countably many PTIME computable graph properties; let $p_0, p_1, p_2, \ldots$ be a list of all these.
  Note: We don't require this list to be recursively enumerable!
- Syntax: $\mathsf{L}[\sigma] := \{0, 1, 2, \ldots\} = \mathbb{N}$. $\mathsf{L}[\sigma]$ is decidable. I.e., condition 1 is met.
- Semantics: for each $n \in \mathsf{L}[\sigma]$ and each graph G let
  $G \models_{\mathsf{L}} n \;:\Longleftrightarrow\; G$ has property p_n.
  All PTIME properties of graphs are expressible. I.e., condition 3 is met.
- But we don't have an algorithm $\mathbb{B}$ that associates with every $n \in \mathbb{N}$ a PTIME algorithm $\mathbb{A}_n$ that decides p_n. I.e., condition 2 is not met.

Overview

Descriptive Complexity

Datalog is poorly expressive

Datalog is highly expressive

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

[1] Recall: *adom*(**I**) is the active domain, i.e., the set of all elements of **dom** occurring in **I**.

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

Definition: A query Q of schema $\mathbf{S}$ is closed under homomorphisms if for all DBs $\mathbf{I}$ and $\mathbf{J}$ and all mappings $h : \mathbf{dom} \to \mathbf{dom}$, the following is true

$$\text{if } h(\mathbf{I}) \subseteq \mathbf{J}, \text{ then } h(\,Q(\mathbf{I})\,) \subseteq Q(\mathbf{J}).$$

Easy Observation: Every Datalog query Q is closed under homomorphisms.

[1] Recall: *adom*(**I**) is the active domain, i.e., the set of all elements of **dom** occurring in **I**.

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

Definition: A query Q of schema $\mathbf{S}$ is closed under homomorphisms if for all DBs $\mathbf{I}$ and $\mathbf{J}$ and all mappings $h : \mathbf{dom} \to \mathbf{dom}$, the following is true

$$\text{if } h(\mathbf{I}) \subseteq \mathbf{J}, \text{ then } h(\,Q(\mathbf{I})\,) \subseteq Q(\mathbf{J}).$$

Easy Observation: Every Datalog query Q is closed under homomorphisms.

Examples: The following queries are not closed under homomorphisms — hence, not definable in Datalog.

- EXACTLY-1-IN-R returning "yes" for DB $\mathbf{I}$ $\iff$ relation R has exactly 1 tuple

[1] Recall: $adom(\mathbf{I})$ is the active domain, i.e., the set of all elements of **dom** occurring in $\mathbf{I}$.

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

Definition: A query Q of schema $\mathbf{S}$ is closed under homomorphisms if for all DBs $\mathbf{I}$ and $\mathbf{J}$ and all mappings $h : \mathbf{dom} \to \mathbf{dom}$, the following is true

$$\text{if } h(\mathbf{I}) \subseteq \mathbf{J}, \text{ then } h(Q(\mathbf{I})) \subseteq Q(\mathbf{J}).$$

Easy Observation: Every Datalog query Q is closed under homomorphisms.

Examples: The following queries are not closed under homomorphisms — hence, not definable in Datalog.

- EXACTLY-1-IN-R returning "yes" for DB $\mathbf{I}$ $\iff$ relation R has exactly 1 tuple
- NEQ with[1] $\text{NEQ}(\mathbf{I}) = \{(a, b) : a, b \in adom(\mathbf{I}),\ a \neq b\}$

[1] Recall: $adom(\mathbf{I})$ is the active domain, i.e., the set of all elements of **dom** occurring in $\mathbf{I}$.

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

Definition: A query Q of schema **S** is closed under homomorphisms if for all DBs **I** and **J** and all mappings $h : \mathbf{dom} \to \mathbf{dom}$, the following is true

$$\text{if } h(\mathbf{I}) \subseteq \mathbf{J}, \text{ then } h(Q(\mathbf{I})) \subseteq Q(\mathbf{J}).$$

Easy Observation: Every Datalog query Q is closed under homomorphisms.

Examples: The following queries are not closed under homomorphisms — hence, not definable in Datalog.

- EXACTLY-1-IN-R returning "yes" for DB **I** $\iff$ relation R has exactly 1 tuple
- NEQ with[1] $\text{NEQ}(\mathbf{I}) = \{(a, b) : a, b \in adom(\mathbf{I}),\ a \neq b\}$
- DISCONNECTED returning "yes" for DB $\mathbf{I}_G$ $\iff$ graph G is not connected

[1] Recall: $adom(\mathbf{I})$ is the active domain, i.e., the set of all elements of **dom** occurring in **I**.

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

Definition: A query Q of schema **S** is closed under homomorphisms if for all DBs **I** and **J** and all mappings $h : \mathbf{dom} \to \mathbf{dom}$, the following is true

$$\text{if } h(\mathbf{I}) \subseteq \mathbf{J}, \text{ then } h(Q(\mathbf{I})) \subseteq Q(\mathbf{J}).$$

Easy Observation: Every Datalog query Q is closed under homomorphisms.

Examples: The following queries are not closed under homomorphisms — hence, not definable in Datalog.

- EXACTLY-1-IN-R returning "yes" for DB **I** $\iff$ relation R has exactly 1 tuple
- NEQ with[1] $\text{NEQ}(\mathbf{I}) = \{(a,b) : a,b \in adom(\mathbf{I}),\ a \neq b\}$
- DISCONNECTED returning "yes" for DB $\mathbf{I}_G$ $\iff$ graph G is not connected
- AT-LEAST-2 returning "yes" for DB **I** $\iff$ $|adom(\mathbf{I})| \geqslant 2$

[1] Recall: $adom(\mathbf{I})$ is the active domain, i.e., the set of all elements of **dom** occurring in **I**.

Datalog queries are closed under homomorphisms

For simplicity, throughout this talk Datalog queries don't contain any constants.

Definition: A query Q of schema $\mathbf{S}$ is closed under homomorphisms if for all DBs $\mathbf{I}$ and $\mathbf{J}$ and all mappings $h : \mathbf{dom} \to \mathbf{dom}$, the following is true

$$\text{if } \ h(\mathbf{I}) \subseteq \mathbf{J}, \ \text{ then } \ h(\, Q(\mathbf{I})\,) \subseteq Q(\mathbf{J}).$$

Easy Observation: Every Datalog query Q is closed under homomorphisms.

Examples: The following queries are not closed under homomorphisms — hence, not definable in Datalog.

- EXACTLY-1-IN-R returning "yes" for DB $\mathbf{I}$ $\iff$ relation R has exactly 1 tuple
- NEQ with[1] $\text{NEQ}(\mathbf{I}) = \{(a,b) : a,b \in adom(\mathbf{I}),\ a \neq b\}$
- DISCONNECTED returning "yes" for DB $\mathbf{I}_G$ $\iff$ graph G is not connected
- AT-LEAST-2 returning "yes" for DB $\mathbf{I}$ $\iff$ $|adom(\mathbf{I})| \geqslant 2$

I.e.: Datalog cannot even count to two!

[1] Recall: $adom(\mathbf{I})$ is the active domain, i.e., the set of all elements of **dom** occurring in $\mathbf{I}$.

Overview

Descriptive Complexity

Datalog is poorly expressive

Datalog is highly expressive

Datalog can simulate runs of Turing machines (1/2)

Represent words w of alphabet Σ by databases $\mathbf{I}_w$ of schema $\mathbf{S}_\Sigma$ consisting of a binary relation SUCC and unary relations MIN, MAX and P_α for every $\alpha \in \Sigma$.

Datalog can simulate runs of Turing machines (1/2)

Represent words w of alphabet Σ by databases $\mathbf{I}_w$ of schema $\mathbf{S}_\Sigma$ consisting of a binary relation SUCC and unary relations MIN , MAX and P_α for every $\alpha \in \Sigma$.

Definition: For a word $w = w_0 \cdots w_{n-1}$ with $w_i \in \Sigma$, let $\mathbf{I}_w$ be the database of schema $\mathbf{S}_\Sigma$ with

- $\mathbf{I}_w(\text{SUCC}) = \{(i, i{+}1) : 0 \leqslant i < n{-}1\}$, $\mathbf{I}_w(\text{MIN}) = \{0\}$, $\mathbf{I}_w(\text{MAX}) = \{n{-}1\}$,
- $\mathbf{I}_w(P_\alpha) = \{i \in \{0, \ldots, n{-}1\} : w_i = \alpha\}$ for each letter $\alpha \in \Sigma$.

Example: $\Sigma = \{a, b, c\}$, $w = aaba$, $\rightsquigarrow$ $\mathbf{I}_w$:

SUCC:

0	1
1	2
2	3

MIN:

0

MAX:

3

P_a:

0
1
3

P_b:

2

P_c:

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = $ "yes" $\iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) =$ "yes" $\iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Furthermore, upon input of M and k, the query $Q_{M,k}$ can be constructed in time polynomial in k and the size of M.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^{\ell})}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma
- $w := aa$ (a string of length 2) and $\mathbf{I} := \mathbf{I}_w$.

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma
- $w := aa$ (a string of length 2) and $\mathbf{I} := \mathbf{I}_w$.

$Q_{M,k}(\mathbf{I}_w) =$ "yes"

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation. I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma
- $w := aa$ (a string of length 2) and $\mathbf{I} := \mathbf{I}_w$.

$$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \quad \overset{\text{Simulation Lemma}}{\iff} \quad M \text{ accepts } w \text{ in at most } |w|^k - 1 \text{ steps}$$

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma
- $w := aa$ (a string of length 2) and $\mathbf{I} := \mathbf{I}_w$.

$$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \overset{\text{Simulation Lemma}}{\iff} M \text{ accepts } w \text{ in at most } |w|^k - 1 \text{ steps}$$

$$\iff T \text{ accepts } u \text{ in at most } 2^{(m^\ell)} \text{ steps}$$

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma
- $w := aa$ (a string of length 2) and $\mathbf{I} := \mathbf{I}_w$.

$$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \overset{\text{Simulation Lemma}}{\iff} M \text{ accepts } w \text{ in at most } |w|^k - 1 \text{ steps}$$
$$\iff T \text{ accepts } u \text{ in at most } 2^{(m^\ell)} \text{ steps}$$
$$\iff u \in L.$$

EXPTIME-hardness of Datalog query evaluation (combined complexity)

Fix an arbitrary problem $L \in$ EXPTIME. There is a DTM T and a number ℓ such that, upon input of a string u of length m, T takes at most $2^{(m^\ell)}$ steps to decide if $u \in L$.

Goal: Find a PTIME-computable reduction from L to Datalog query evaluation.
I.e.: For every word u, construct a datalog query Q and a database $\mathbf{I}$ such that $u \in L \iff Q(\mathbf{I}) =$ "yes".

Idea: For input word u choose Q and w as follows:

- Modify T into a deterministic Turing machine M which deletes its input, writes u onto its tape and then simulates T upon input u.
- $m := |u|$, $k := m^\ell + 1$, and choose $Q := Q_{M,k}$ with the Simulation Lemma
- $w := aa$ (a string of length 2) and $\mathbf{I} := \mathbf{I}_w$.

$$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \overset{\text{Simulation Lemma}}{\iff} M \text{ accepts } w \text{ in at most } |w|^k - 1 \text{ steps}$$
$$\iff T \text{ accepts } u \text{ in at most } 2^{(m^\ell)} \text{ steps}$$
$$\iff u \in L.$$

Furthermore, $Q_{M,k}$ and $\mathbf{I}_w$ can be constructed in time polynomial in k, i.e., polynomial in $|u|$. □

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) =$ "yes" $\iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Furthermore, upon input of M and k, the query $Q_{M,k}$ can be constructed in time polynomial in k and the size of M.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = \text{“yes”} \iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Furthermore, upon input of M and k, the query $Q_{M,k}$ can be constructed in time polynomial in k and the size of M.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

(3) Datalog query evaluation is PTIME-complete w.r.t. data complexity.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = $ "yes" $\iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Furthermore, upon input of M and k, the query $Q_{M,k}$ can be constructed in time polynomial in k and the size of M. Moreover, there is a log-space algorithm which, upon input of a string w, constructs the database $\mathbf{I}_w$.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

(3) Datalog query evaluation is PTIME-complete w.r.t. data complexity.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = \text{"yes"} \iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Furthermore, upon input of M and k, the query $Q_{M,k}$ can be constructed in time polynomial in k and the size of M. Moreover, there is a log-space algorithm which, upon input of a string w, constructs the database $\mathbf{I}_w$.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

(3) Datalog query evaluation is PTIME-complete w.r.t. data complexity.

(4) The Boundedness Problem for Datalog is undecidable.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Proof of the Simulation Lemma (1/5)

DTM *M* : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, . . .

Proof of the Simulation Lemma (1/5)

DTM M : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first $n^k - 1$ steps of M.

Proof of the Simulation Lemma (1/5)

DTM *M* : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first $n^k - 1$ steps of M.

Let $[n] := \{0, \ldots, n-1\} = adom(\mathbf{I}_w)$.

Proof of the Simulation Lemma (1/5)

DTM M : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first $n^k - 1$ steps of M.

Let $[n] := \{0, \dots, n-1\} = adom(\mathbf{I}_w)$.
Use k-tuples over $[n]$ to represent numbers in $\{0, \dots, n^k - 1\}$:

$$\overline{x} = (x_{k-1}, \dots, x_0) \in [n]^k \text{ represents number } nr(\overline{x}) := \sum_{i=0}^{k-1} x_i \cdot n^i.$$

Proof of the Simulation Lemma (1/5)

DTM M : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first $n^k - 1$ steps of M.

Let $[n] := \{0, \ldots, n-1\} = adom(\mathbf{I}_w)$.
Use k-tuples over $[n]$ to represent numbers in $\{0, \ldots, n^k - 1\}$:

$$\overline{x} = (x_{k-1}, \ldots, x_0) \in [n]^k \text{ represents number } nr(\overline{x}) := \sum_{i=0}^{k-1} x_i \cdot n^i.$$

Our Datalog program $P_{M,k}$ will use the following idb-predicates to represent configurations of M on input w at time steps $0, 1, \ldots, n^k - 1$:

- A $2k$-ary predicate HEAD.
- A k-ary predicate STATE_q, for each state q (incl. “halt”, “accept”, “reject”).
- A $2k$-ary predicate TAPE_a, for each tape symbol a.

Proof of the Simulation Lemma (1/5)

DTM M : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first $n^k - 1$ steps of M.

Let $[n] := \{0, \ldots, n-1\} = adom(\mathbf{I}_w)$.
Use k-tuples over $[n]$ to represent numbers in $\{0, \ldots, n^k - 1\}$:

$$\overline{x} = (x_{k-1}, \ldots, x_0) \in [n]^k \text{ represents number } nr(\overline{x}) := \sum_{i=0}^{k-1} x_i \cdot n^i.$$

Our Datalog program $P_{M,k}$ will use the following idb-predicates to represent configurations of M on input w at time steps $0, 1, \ldots, n^k - 1$:

- A $2k$-ary predicate HEAD.
  Intended meaning of $\text{HEAD}(\overline{x}, \overline{y})$: at time $nr(\overline{x})$, M's head is at tape cell $nr(\overline{y})$
- A k-ary predicate STATE_q, for each state q (incl. "halt", "accept", "reject").
- A $2k$-ary predicate TAPE_a, for each tape symbol a.

Proof of the Simulation Lemma (1/5)

DTM M : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first n^k-1 steps of M.

Let $[n] := \{0, \ldots, n-1\} = adom(\mathbf{I}_w)$.
Use k-tuples over $[n]$ to represent numbers in $\{0, \ldots, n^k-1\}$:

$$\overline{x} = (x_{k-1}, \ldots, x_0) \in [n]^k \text{ represents number } nr(\overline{x}) := \sum_{i=0}^{k-1} x_i \cdot n^i.$$

Our Datalog program $P_{M,k}$ will use the following idb-predicates to represent configurations of M on input w at time steps $0, 1, \ldots, n^k-1$:

- A $2k$-ary predicate HEAD.
  Intended meaning of $\text{HEAD}(\overline{x}, \overline{y})$: at time $nr(\overline{x})$, M's head is at tape cell $nr(\overline{y})$
- A k-ary predicate STATE_q, for each state q (incl. "halt", "accept", "reject").
  Intended meaning of $\text{STATE}_q(\overline{x})$: M is in state q at time $nr(\overline{x})$
- A $2k$-ary predicate TAPE_a, for each tape symbol a.

Proof of the Simulation Lemma (1/5)

DTM M : only 1 tape; this is single-sided infinite with tape cells 0, 1, 2, 3, ...

For input string $w = w_0 \cdots w_{n-1}$, we want to simulate the first $n^k - 1$ steps of M.

Let $[n] := \{0, \ldots, n-1\} = adom(\mathbf{I}_w)$.
Use k-tuples over $[n]$ to represent numbers in $\{0, \ldots, n^k - 1\}$:

$$\overline{x} = (x_{k-1}, \ldots, x_0) \in [n]^k \text{ represents number } nr(\overline{x}) := \sum_{i=0}^{k-1} x_i \cdot n^i.$$

Our Datalog program $P_{M,k}$ will use the following idb-predicates to represent configurations of M on input w at time steps $0, 1, \ldots, n^k - 1$:

- A $2k$-ary predicate HEAD.
  Intended meaning of $\text{HEAD}(\overline{x}, \overline{y})$: at time $nr(\overline{x})$, M's head is at tape cell $nr(\overline{y})$
- A k-ary predicate STATE_q, for each state q (incl. "halt", "accept", "reject").
  Intended meaning of $\text{STATE}_q(\overline{x})$: M is in state q at time $nr(\overline{x})$
- A $2k$-ary predicate TAPE_a, for each tape symbol a.
  Intended meaning of $\text{TAPE}_a(\overline{x}, \overline{y})$:
  at time $nr(\overline{x})$ tape cell $nr(\overline{y})$ carries the symbol a .

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\text{HEAD}(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0),\ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_0)$$

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\text{HEAD}(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0), \ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\text{HEAD}(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0),\ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

$$\text{STATE}_{q_0}(\overline{x}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0)$$

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\mathrm{HEAD}(\overline{x}, \overline{y}) \leftarrow \mathrm{MIN}(x_{k-1}), \ldots, \mathrm{MIN}(x_0),\ \mathrm{MIN}(y_{k-1}), \ldots, \mathrm{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

$$\mathrm{STATE}_{q_0}(\overline{x}) \leftarrow \mathrm{MIN}(x_{k-1}), \ldots, \mathrm{MIN}(x_0)$$

- At time 0, tape positions $0, \ldots, n-1$ carry the input string w:

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\mathrm{HEAD}(\overline{x}, \overline{y}) \leftarrow \mathrm{MIN}(x_{k-1}), \ldots, \mathrm{MIN}(x_0),\ \mathrm{MIN}(y_{k-1}), \ldots, \mathrm{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

$$\mathrm{STATE}_{q_0}(\overline{x}) \leftarrow \mathrm{MIN}(x_{k-1}), \ldots, \mathrm{MIN}(x_0)$$

- At time 0, tape positions $0, \ldots, n{-}1$ carry the input string w:

$$\mathrm{TAPE}_a(\overline{x}, \overline{y}) \leftarrow \mathrm{MIN}(x_{k-1}), \ldots, \mathrm{MIN}(x_0),\ \mathrm{MIN}(y_{k-1}), \ldots, \mathrm{MIN}(y_1), \mathrm{P}_a(y_0)$$

  Add this rule for every letter $a \in \Sigma$.

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\text{HEAD}(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0),\ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

$$\text{STATE}_{q_0}(\overline{x}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0)$$

- At time 0, tape positions $0, \ldots, n{-}1$ carry the input string w:

$$\text{TAPE}_a(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0),\ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_1), \text{P}_a(y_0)$$

  Add this rule for every letter $a \in \Sigma$.

- At time 0, tape positions $n, \ldots, n^k{-}1$ carry the blank symbol $\square$:

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\text{HEAD}(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0),\ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

$$\text{STATE}_{q_0}(\overline{x}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0)$$

- At time 0, tape positions $0, \ldots, n-1$ carry the input string w:

$$\text{TAPE}_a(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0),\ \text{MIN}(y_{k-1}), \ldots, \text{MIN}(y_1), \text{P}_a(y_0)$$

  Add this rule for every letter $a \in \Sigma$.

- At time 0, tape positions $n, \ldots, n^k-1$ carry the blank symbol $\Box$:
  For each $i \in \{1, \ldots, k-1\}$ add the rule

$$\text{TAPE}_\Box(\overline{x}, \overline{y}) \leftarrow \text{MIN}(x_{k-1}), \ldots, \text{MIN}(x_0), \text{NOTMIN}(y_i)$$

Proof of the Simulation Lemma (2/5)

Start with $P_{M,k} := \emptyset$ and add rules as follows.

Step 1: Add rules to achieve the intended meaning at time 0.

- At time 0, head is at tape position 0:

$$\text{HEAD}(\overline{x},\overline{y}) \leftarrow \text{MIN}(x_{k-1}),\ldots,\text{MIN}(x_0),\ \text{MIN}(y_{k-1}),\ldots,\text{MIN}(y_0)$$

- At time 0, M is in the starting state q_0:

$$\text{STATE}_{q_0}(\overline{x}) \leftarrow \text{MIN}(x_{k-1}),\ldots,\text{MIN}(x_0)$$

- At time 0, tape positions $0,\ldots,n-1$ carry the input string w:

$$\text{TAPE}_a(\overline{x},\overline{y}) \leftarrow \text{MIN}(x_{k-1}),\ldots,\text{MIN}(x_0),\ \text{MIN}(y_{k-1}),\ldots,\text{MIN}(y_1),\text{P}_a(y_0)$$

  Add this rule for every letter $a \in \Sigma$.

- At time 0, tape positions $n,\ldots,n^k-1$ carry the blank symbol $\Box$:
  For each $i \in \{1,\ldots,k-1\}$ add the rule

$$\text{TAPE}_{\Box}(\overline{x},\overline{y}) \leftarrow \text{MIN}(x_{k-1}),\ldots,\text{MIN}(x_0),\text{NOTMIN}(y_i)$$

  And add the rule

$$\text{NOTMIN}(z) \leftarrow \text{SUCC}(z',z)$$

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "$+1$": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples".

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "$+1$": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate $\textsc{succ}_\ell$ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\textsc{succ}_1(z, z') \leftarrow \textsc{succ}(z, z')$.

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t{+}1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1,\ldots,k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z,z') \leftarrow \text{SUCC}(z,z')$. For each $\ell \in \{1,\ldots k{-}1\}$ we add the rules

$$\text{SUCC}_{\ell+1}(x_\ell,x_{\ell-1},\ldots,x_0,\ y_\ell,y_{\ell-1},\ldots,y_0) \leftarrow \text{MAX}(x_{\ell-1}),\ldots,\text{MAX}(x_0),\text{SUCC}(x_\ell,y_\ell),\text{MIN}(y_{\ell-1}),\ldots,\text{MIN}(y_0)$$

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z, z') \leftarrow \text{SUCC}(z, z')$. For each $\ell \in \{1, \ldots k-1\}$ we add the rules

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ y_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{MAX}(x_{\ell-1}), \ldots, \text{MAX}(x_0), \text{SUCC}(x_\ell, y_\ell), \text{MIN}(y_{\ell-1}), \ldots, \text{MIN}(y_0)$$

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ x_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{SUCC}_\ell(x_{\ell-1}, \ldots, x_0,\ y_{\ell-1}, \ldots, y_0), \text{ADOM}(x_\ell)$$

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z, z') \leftarrow \text{SUCC}(z, z')$. For each $\ell \in \{1, \ldots k-1\}$ we add the rules

  $$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ y_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{MAX}(x_{\ell-1}), \ldots, \text{MAX}(x_0), \text{SUCC}(x_\ell, y_\ell), \text{MIN}(y_{\ell-1}), \ldots, \text{MIN}(y_0)$$

  $$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ x_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{SUCC}_\ell(x_{\ell-1}, \ldots, x_0,\ y_{\ell-1}, \ldots, y_0), \text{ADOM}(x_\ell)$$

  And we add rules for describing the active domain:

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z, z') \leftarrow \text{SUCC}(z, z')$. For each $\ell \in \{1, \ldots k-1\}$ we add the rules

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ y_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{MAX}(x_{\ell-1}), \ldots, \text{MAX}(x_0), \text{SUCC}(x_\ell, y_\ell), \text{MIN}(y_{\ell-1}), \ldots, \text{MIN}(y_0)$$

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ x_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{SUCC}_\ell(x_{\ell-1}, \ldots, x_0,\ y_{\ell-1}, \ldots, y_0), \text{ADOM}(x_\ell)$$

  And we add rules for describing the active domain:
  $\text{ADOM}(z) \leftarrow \text{SUCC}(z, z')$ and $\text{ADOM}(z') \leftarrow \text{SUCC}(z, z')$
  and the rule $\text{ADOM}(z) \leftarrow \text{X}(z)$ for each unary edb-predicate X.

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t{+}1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z, z') \leftarrow \text{SUCC}(z, z')$. For each $\ell \in \{1, \ldots k{-}1\}$ we add the rules

  $$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ y_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{MAX}(x_{\ell-1}), \ldots, \text{MAX}(x_0), \text{SUCC}(x_\ell, y_\ell), \text{MIN}(y_{\ell-1}), \ldots, \text{MIN}(y_0)$$

  $$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ x_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{SUCC}_\ell(x_{\ell-1}, \ldots, x_0,\ y_{\ell-1}, \ldots, y_0), \text{ADOM}(x_\ell)$$

  And we add rules for describing the active domain:
  $\text{ADOM}(z) \leftarrow \text{SUCC}(z, z')$ and $\text{ADOM}(z') \leftarrow \text{SUCC}(z, z')$
  and the rule $\text{ADOM}(z) \leftarrow \text{X}(z)$ for each unary edb-predicate X.

- Auxiliary rules for strict linear order and inequality on k-tuples:

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z, z') \leftarrow \text{SUCC}(z, z')$. For each $\ell \in \{1, \ldots k-1\}$ we add the rules

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ y_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{MAX}(x_{\ell-1}), \ldots, \text{MAX}(x_0), \text{SUCC}(x_\ell, y_\ell), \text{MIN}(y_{\ell-1}), \ldots, \text{MIN}(y_0)$$

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ x_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{SUCC}_\ell(x_{\ell-1}, \ldots, x_0,\ y_{\ell-1}, \ldots, y_0), \text{ADOM}(x_\ell)$$

  And we add rules for describing the active domain:
  $\text{ADOM}(z) \leftarrow \text{SUCC}(z, z')$ and $\text{ADOM}(z') \leftarrow \text{SUCC}(z, z')$
  and the rule $\text{ADOM}(z) \leftarrow \text{X}(z)$ for each unary edb-predicate X.

- Auxiliary rules for strict linear order and inequality on k-tuples:

$$\text{LESS}_k(\overline{x}, \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{y})$$

$$\text{LESS}_k(\overline{x}, \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{z}),\ \text{LESS}_k(\overline{z}, \overline{y})$$

Proof of the Simulation Lemma (3/5)

Step 2: Add rules so that if intended meaning is achieved at time t, then also at $t+1$.

- Auxiliary rules for reasoning about "+1": For each $\ell \in \{1, \ldots, k\}$, use a 2ℓ-ary predicate SUCC_ℓ to represent the "successor on ℓ-tuples". We add to $P_{M,k}$ the rule $\text{SUCC}_1(z, z') \leftarrow \text{SUCC}(z, z')$. For each $\ell \in \{1, \ldots k-1\}$ we add the rules

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ y_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{MAX}(x_{\ell-1}), \ldots, \text{MAX}(x_0), \text{SUCC}(x_\ell, y_\ell), \text{MIN}(y_{\ell-1}), \ldots, \text{MIN}(y_0)$$

$$\text{SUCC}_{\ell+1}(x_\ell, x_{\ell-1}, \ldots, x_0,\ x_\ell, y_{\ell-1}, \ldots, y_0) \leftarrow \text{SUCC}_\ell(x_{\ell-1}, \ldots, x_0,\ y_{\ell-1}, \ldots, y_0), \text{ADOM}(x_\ell)$$

  And we add rules for describing the active domain:
  $\text{ADOM}(z) \leftarrow \text{SUCC}(z, z')$ and $\text{ADOM}(z') \leftarrow \text{SUCC}(z, z')$
  and the rule $\text{ADOM}(z) \leftarrow \text{X}(z)$ for each unary edb-predicate X.

- Auxiliary rules for strict linear order and inequality on k-tuples:

$$\text{LESS}_k(\overline{x}, \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{y})$$

$$\text{LESS}_k(\overline{x}, \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{z}),\ \text{LESS}_k(\overline{z}, \overline{y})$$

$$\text{NEQ}_k(\overline{x}, \overline{y}) \leftarrow \text{LESS}_k(\overline{x}, \overline{y})$$

$$\text{NEQ}_k(\overline{x}, \overline{y}) \leftarrow \text{LESS}_k(\overline{y}, \overline{x})$$

Proof of the Simulation Lemma (4/5)

Now consider each state q and tape symbol a, and let $(q', a', m) := \delta(q, a)$, where δ is the transition function of M.

Proof of the Simulation Lemma (4/5)

Now consider each state q and tape symbol a, and let $(q', a', m) := \delta(q, a)$, where δ is the transition function of M. We add to $P_{M,k}$ the following rules:

$$\text{STATE}_{q'}(\overline{x}') \leftarrow \underbrace{\text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})}_{\text{at time } t := nr(\overline{x}),\ M \text{ is in state } q, \text{ reads symbol } a, \text{ and } nr(\overline{x}') = t+1}$$

Proof of the Simulation Lemma (4/5)

Now consider each state q and tape symbol a, and let $(q', a', m) := \delta(q, a)$, where δ is the transition function of M. We add to $P_{M,k}$ the following rules:

$$\text{STATE}_{q'}(\overline{x}') \leftarrow \underbrace{\text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})}_{\text{at time } t := nr(\overline{x}),\ M \text{ is in state } q, \text{ reads symbol } a, \text{ and } nr(\overline{x}') = t+1}$$

$$\underbrace{\text{TAPE}_{a'}(\overline{x}', \overline{y})}_{\text{at time } t+1, \text{ position } nr(\overline{y}) \text{ carries the letter written at step } t} \leftarrow \overbrace{\text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})}$$

And all other tape positions carry the same letter at time $t+1$ as at time t:
For every tape symbol b add the rule

$$\text{TAPE}_b(\overline{x}', \overline{y}') \leftarrow \text{TAPE}_b(\overline{x}, \overline{y}'),\ \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{NEQ}(\overline{y}, \overline{y}')$$

Proof of the Simulation Lemma (4/5)

Now consider each state q and tape symbol a, and let $(q', a', m) := \delta(q, a)$, where δ is the transition function of M. We add to $P_{M,k}$ the following rules:

$$\text{STATE}_{q'}(\overline{x}') \leftarrow \underbrace{\text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})}_{\text{at time } t := nr(\overline{x}),\ M \text{ is in state } q, \text{ reads symbol } a, \text{ and } nr(\overline{x}') = t+1}$$

$$\underbrace{\text{TAPE}_{a'}(\overline{x}', \overline{y})}_{\text{at time } t+1, \text{ position } nr(\overline{y}) \text{ carries the letter written at step } t} \leftarrow \overbrace{\text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})}$$

And all other tape positions carry the same letter at time $t+1$ as at time t:
For every tape symbol b add the rule

$$\text{TAPE}_b(\overline{x}', \overline{y}') \leftarrow \text{TAPE}_b(\overline{x}, \overline{y}'),\ \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{NEQ}(\overline{y}, \overline{y}')$$

Add similar rules for representing the head movement of M:
$m \in \{0, 1, -1\}$ indicates whether the head stays or moves one position to the right or the left, respectively.

Proof of the Simulation Lemma (5/5)

Recall that we consider M's transition $(q', a', m) := \delta(q, a)$.

- If $m = 0$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})$$

Proof of the Simulation Lemma (5/5)

Recall that we consider M's transition $(q', a', m) := \delta(q, a)$.

- If $m = 0$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})$$

- If $m = 1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}, \overline{y}')$$

Proof of the Simulation Lemma (5/5)

Recall that we consider *M*'s transition $(q', a', m) := \delta(q, a)$.

- If $m = 0$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})$$

- If $m = 1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}, \overline{y}')$$

- If $m = -1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}', \overline{y})$$

Proof of the Simulation Lemma (5/5)

Recall that we consider M's transition $(q', a', m) := \delta(q, a)$.

- If $m = 0$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})$$

- If $m = 1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}, \overline{y}')$$

- If $m = -1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}', \overline{y})$$

To ensure that the Datalog query outputs the correct result, we add the rule

$$\text{GOAL}() \leftarrow \text{STATE}_{\text{accept}}(\overline{x})$$

Proof of the Simulation Lemma (5/5)

Recall that we consider M's transition $(q', a', m) := \delta(q, a)$.

- If $m = 0$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}) \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y})$$

- If $m = 1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}, \overline{y}')$$

- If $m = -1$, we add to $P_{M,k}$ the rule

$$\text{HEAD}(\overline{x}', \overline{y}') \leftarrow \text{SUCC}_k(\overline{x}, \overline{x}'),\ \text{STATE}_q(\overline{x}),\ \text{HEAD}(\overline{x}, \overline{y}),\ \text{TAPE}_a(\overline{x}, \overline{y}),\ \text{SUCC}_k(\overline{y}', \overline{y})$$

To ensure that the Datalog query outputs the correct result, we add the rule

$$\text{GOAL}() \leftarrow \text{STATE}_{\text{accept}}(\overline{x})$$

This finally completes the construction of the Datalog program $P_{M,k}$.
It is straightforward to verify that this proves the Simulation Lemma. □

Datalog can simulate runs of Turing machines (2/2)

Simulation Lemma: For every deterministic Turing machine M with input alphabet Σ and every integer $k \geqslant 1$, there is a Datalog program $P_{M,k}$ with edb-predicates $\mathbf{S}_\Sigma$ and a 0-ary idb-predicate GOAL, such that the following is true for the Datalog query $Q_{M,k} := (P_{M,k}, \text{GOAL})$ and for every non-empty word $w \in \Sigma^*$:

$Q_{M,k}(\mathbf{I}_w) = \text{“yes”} \iff$ upon input w, M stops in an accepting state after at most $|w|^k - 1$ steps.

Furthermore, upon input of M and k, the query $Q_{M,k}$ can be constructed in time polynomial in k and the size of M. Moreover, there is a log-space algorithm which, upon input of a string w, constructs the database $\mathbf{I}_w$.

Easy consequences:

(1) Datalog captures PTIME on database-respresentations of strings.[2]

(2) Datalog query evaluation is EXPTIME-complete w.r.t. combined complexity.

(3) Datalog query evaluation is PTIME-complete w.r.t. data complexity.

(4) The Boundedness Problem for Datalog is undecidable.

[2] This is the variant of the Immerman-Vardi Theorem promised at the beginning of the talk.

Databases and Despriptive Complexity — Part 2:

A Toolkit for Proving Limitations of the Expressive Power of Logics

Nicole Schweikardt

Humboldt-Universität zu Berlin

EPIT 2019 — Spring School on Theoretical Computer Science:
Databases, logic and automata

Luminy, 11 April 2019

In this talk

- Consider finite directed graphs $G = (V^G, E^G)$.

  Sometimes, nodes are additionally labeled (i.e., colored) by a symbol from a finite alphabet Σ.

In this talk

- Consider finite directed graphs $G = (V^G, E^G)$.

  Sometimes, nodes are additionally labeled (i.e., colored) by a symbol from a finite alphabet Σ.

- p is a graph property, if the following is true:

  if $G \cong H$, then $\quad G$ has property $p \iff H$ has property p

In this talk

- Consider finite directed graphs $G = (V^G, E^G)$.

  Sometimes, nodes are additionally labeled (i.e., colored) by a symbol from a finite alphabet Σ.

- p is a graph property, if the following is true:

  if $G \cong H$, then $\quad G$ has property $p \iff H$ has property p

- q is a k-ary graph query, if the following is true:

  if $\pi : G \cong H$, then for all $a_1, \ldots, a_k \in V^G$,

  $$(a_1, \ldots, a_k) \in q(G) \iff (\pi(a_1), \ldots, \pi(a_k)) \in q(H)$$

In this talk

- Consider finite directed graphs $G = (V^G, E^G)$.

  Sometimes, nodes are additionally labeled (i.e., colored) by a symbol from a finite alphabet Σ.

- p is a graph property, if the following is true:

  if $G \cong H$, then G has property $p \iff H$ has property p

- q is a k-ary graph query, if the following is true:

  if $\pi : G \cong H$, then for all $a_1, \ldots, a_k \in V^G$,

  $$(a_1, \ldots, a_k) \in q(G) \iff (\pi(a_1), \ldots, \pi(a_k)) \in q(H)$$

- I.e., graph properties and queries are closed under isomorphisms.

Logics expressing graph properties and queries

Classical logics like, e.g.

- FO (first-order logic: Boolean combinations + quantification over nodes)

express graph properties and queries in a straightforward way.

Example:

- $q(G) := \{\ x \in V^G\ :\ x \text{ lies on a triangle}\ \}$ is expressed in FO via

$$\varphi(x) \ := \ \exists y\, \exists z\, \big(\ E(x,y) \ \wedge\ E(y,z)\ \wedge\ E(z,x)\ \big)$$

Logics expressing graph properties and queries

Classical logics like, e.g.

- FO (first-order logic: Boolean combinations + quantification over nodes)
- EMSO (existential monadic second-order logic: FO + existential quantification over sets of nodes)

express graph properties and queries in a straightforward way.

Example:

- $q(G) := \{ \, x \in V^G \; : \; x \text{ lies on a triangle} \, \}$ is expressed in FO via

$$\varphi(x) \;:=\; \exists y \, \exists z \, \big(\, E(x,y) \,\wedge\, E(y,z) \,\wedge\, E(z,x) \, \big)$$

- $p \;=\; \{ \, G \; : \; G \text{ is 3-colorable} \, \}$ is expressed in EMSO via

$$\exists R \, \exists B \, \exists G \Big(\; \forall x \, \big(R(x) \vee B(x) \vee G(x)\big) \;\; \wedge \;\; \forall x \, \forall y \, \big(\, E(x,y) \rightarrow \; \neg \big((R(x) \wedge R(y)) \,\vee\, (B(x) \wedge B(y)) \,\vee\, (G(x) \wedge G(y))\big) \, \big) \Big)$$

Question

How can we prove that

certain properties or queries

are NOT expressible in a particular logic?

Overview

Overview

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.
- The asymptotic probability of p is $\mu(p) := \lim_{n \to \infty} \mu_n(p)$ (if the limit exists).

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.
- The asymptotic probability of p is $\mu(p) := \lim_{n\to\infty} \mu_n(p)$ (if the limit exists).
- A logic L is said to have the zero-one law, if for every L-definable graph property p, the asymptotic probability $\mu(p)$ exists and is either 0 or 1.

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.
- The asymptotic probability of p is $\mu(p) := \lim_{n\to\infty} \mu_n(p)$ (if the limit exists).
- A logic L is said to have the zero-one law, if for every L-definable graph property p, the asymptotic probability $\mu(p)$ exists and is either 0 or 1.

Theorem:

- FO has the zero-one law. (Glebskii et al. 1969; Fagin 1976)

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.
- The asymptotic probability of p is $\mu(p) := \lim_{n\to\infty} \mu_n(p)$ (if the limit exists).
- A logic L is said to have the zero-one law, if for every L-definable graph property p, the asymptotic probability $\mu(p)$ exists and is either 0 or 1.

Theorem:

- FO has the zero-one law. (Glebskii et al. 1969; Fagin 1976)
- $L^{\omega}_{\infty,\omega}$ has the zero-one law. (Kolaitis, Vardi 1992])
  Thus, also the fixed point logics LFP and PFP have the zero-one law.

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.
- The asymptotic probability of p is $\mu(p) := \lim_{n\to\infty} \mu_n(p)$ (if the limit exists).
- A logic L is said to have the zero-one law, if for every L-definable graph property p, the asymptotic probability $\mu(p)$ exists and is either 0 or 1.

Theorem:

- FO has the zero-one law. (Glebskii et al. 1969; Fagin 1976)
- $L^{\omega}_{\infty,\omega}$ has the zero-one law. (Kolaitis, Vardi 1992])
  Thus, also the fixed point logics LFP and PFP have the zero-one law.

Example: The property of having an even number of nodes or edges is not definable in a logic that has the zero-one law (since $\mu(p)$ doesn't exist, resp., is equal to 0.5).

Zero-One Laws

- Let p be a graph property.
- Let $\mu_n(p)$ be the probability that a graph chosen uniformly at random from the set of all graphs on n vertices has property p.
- The asymptotic probability of p is $\mu(p) := \lim_{n\to\infty} \mu_n(p)$ (if the limit exists).
- A logic L is said to have the zero-one law, if for every L-definable graph property p, the asymptotic probability $\mu(p)$ exists and is either 0 or 1.

Theorem:

- FO has the zero-one law. (Glebskii et al. 1969; Fagin 1976)
- $L^{\omega}_{\infty,\omega}$ has the zero-one law. (Kolaitis, Vardi 1992])
  Thus, also the fixed point logics LFP and PFP have the zero-one law.

Example: The property of having an even number of nodes or edges is not definable in a logic that has the zero-one law (since $\mu(p)$ doesn't exist, resp., is equal to 0.5).

Note: There are properties with $\mu(p) \in \{0, 1\}$ which cannot be expressed in FO.
Example: Connectivity.

Overview

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$,

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator,

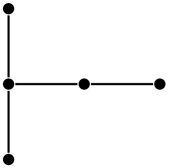
The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

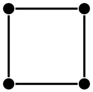
The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

$\mathcal{A}$:



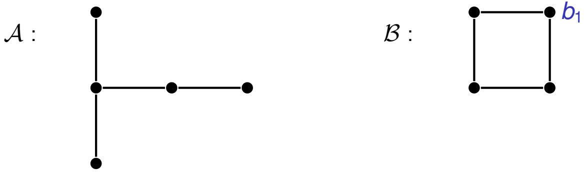
$\mathcal{B}$:



Each round $i \in \{1, \ldots, r\}$ is played as follows:

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

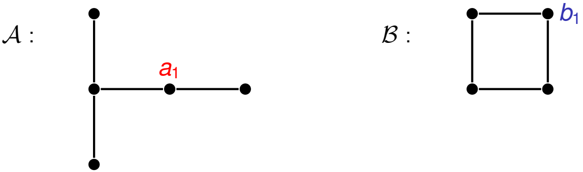


Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

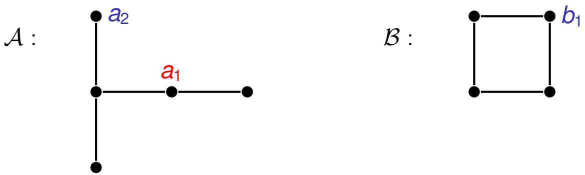


Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

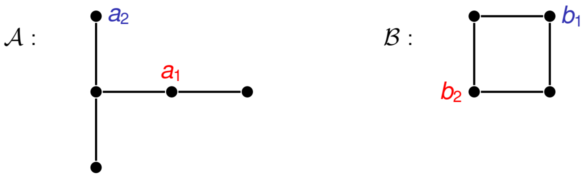


Each round $i \in \{1, \dots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

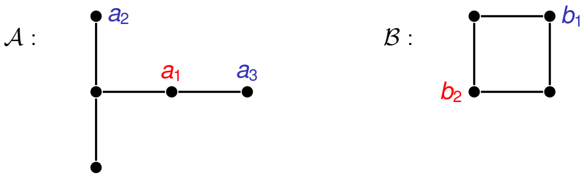


Each round $i \in \{1, \dots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

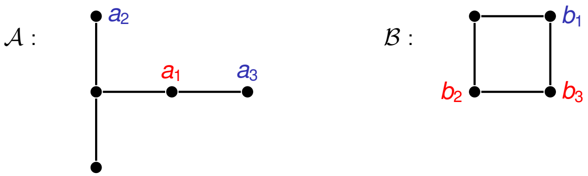


Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.

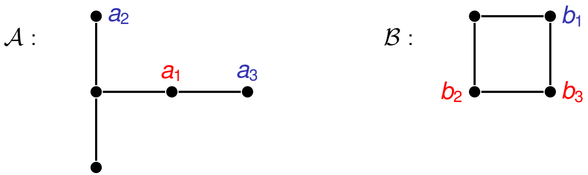


Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.



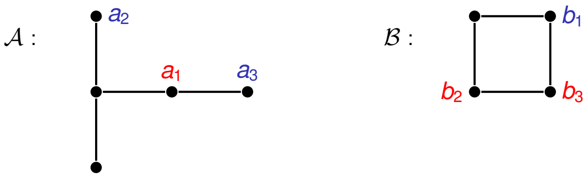
Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

After r rounds, vertices $a_1, \ldots, a_r$ have been chosen in $\mathcal{A}$, and vertices $b_1, \ldots, b_r$ have been chosen in $\mathcal{B}$.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.



Each round $i \in \{1, \ldots, r\}$ is played as follows:

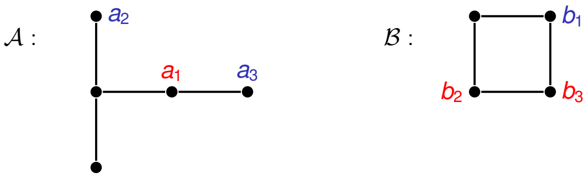
1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

After r rounds, vertices $a_1, \ldots, a_r$ have been chosen in $\mathcal{A}$, and vertices $b_1, \ldots, b_r$ have been chosen in $\mathcal{B}$.

Duplicator wins, iff the mapping $(a_i \mapsto b_i)$ is an isomorphism on the induced subgraphs $\mathcal{A}_{|\{a_1,\ldots,a_r\}}$ and $\mathcal{B}_{|\{b_1,\ldots,b_r\}}$.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.



Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

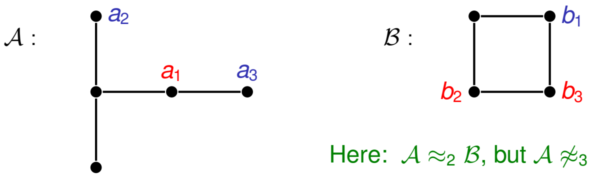
After r rounds, vertices $a_1, \ldots, a_r$ have been chosen in $\mathcal{A}$, and vertices $b_1, \ldots, b_r$ have been chosen in $\mathcal{B}$.

Duplicator wins, iff the mapping $(a_i \mapsto b_i)$ is an isomorphism on the induced subgraphs $\mathcal{A}_{|\{a_1,\ldots,a_r\}}$ and $\mathcal{B}_{|\{b_1,\ldots,b_r\}}$.

Write $\mathcal{A} \approx_r \mathcal{B}$ iff Duplicator has a winning strategy.

The Ehrenfeucht-Fraïssé game

is played on 2 graphs: $\mathcal{A}$ & $\mathcal{B}$, by 2 players: Spoiler & Duplicator, in r rounds.



Here: $\mathcal{A} \approx_2 \mathcal{B}$, but $\mathcal{A} \not\approx_3 \mathcal{B}$.

Each round $i \in \{1, \ldots, r\}$ is played as follows:

1. Spoiler chooses a vertex in one of the two graphs,
2. Duplicator chooses a vertex in the other graph.

After r rounds, vertices $a_1, \ldots, a_r$ have been chosen in $\mathcal{A}$, and vertices $b_1, \ldots, b_r$ have been chosen in $\mathcal{B}$.

Duplicator wins, iff the mapping $(a_i \mapsto b_i)$ is an isomorphism on the induced subgraphs $\mathcal{A}_{|\{a_1,\ldots,a_r\}}$ and $\mathcal{B}_{|\{b_1,\ldots,b_r\}}$.

Write $\mathcal{A} \approx_r \mathcal{B}$ iff Duplicator has a winning strategy.

Ehrenfeucht-Fraïssé Theorem

Theorem:

$\mathcal{A} \approx_r \mathcal{B} \iff \mathcal{A}$ and $\mathcal{B}$ satisfy the same FO-sentences of quantifier depth $\leqslant r$.

Ehrenfeucht-Fraïssé Theorem

Theorem:

$\mathcal{A} \approx_r \mathcal{B} \iff \mathcal{A}$ and $\mathcal{B}$ satisfy the same FO-sentences of quantifier depth $\leqslant r$.

Corollary

A graph property p is not FO-expressible, if the following is true:
For every r there are graphs $\mathcal{A}_r$ and $\mathcal{B}_r$ such that

- *$\mathcal{A}_r$ has property p,*
- *$\mathcal{B}_r$ doesn't have property p, and*
- *$\mathcal{A}_r \approx_r \mathcal{B}_r$.*

Ehrenfeucht-Fraïssé Theorem

Theorem:

$\mathcal{A} \approx_r \mathcal{B} \iff \mathcal{A}$ and $\mathcal{B}$ satisfy the same FO-sentences of quantifier depth $\leqslant r$.

Corollary

A graph property p is not FO-expressible, if the following is true:
For every r there are graphs $\mathcal{A}_r$ and $\mathcal{B}_r$ such that

- *$\mathcal{A}_r$ has property p,*
- *$\mathcal{B}_r$ doesn't have property p, and*
- *$\mathcal{A}_r \approx_r \mathcal{B}_r$.*

Examples:

- The property of being a linear order of even cardinality is not FO-expressible.

Ehrenfeucht-Fraïssé Theorem

Theorem:

$\mathcal{A} \approx_r \mathcal{B} \iff \mathcal{A}$ and $\mathcal{B}$ satisfy the same FO-sentences of quantifier depth $\leqslant r$.

Corollary

A graph property p is not FO-expressible, if the following is true:
For every r there are graphs $\mathcal{A}_r$ and $\mathcal{B}_r$ such that

- *$\mathcal{A}_r$ has property p,*
- *$\mathcal{B}_r$ doesn't have property p, and*
- *$\mathcal{A}_r \approx_r \mathcal{B}_r$.*

Examples:

- The property of being a linear order of even cardinality is not FO-expressible.
- Connectivity is not EMSO-expressible (Fagin, 1975)

Ehrenfeucht-Fraïssé Theorem

Theorem:

$\mathcal{A} \approx_r \mathcal{B} \iff \mathcal{A}$ and $\mathcal{B}$ satisfy the same FO-sentences of quantifier depth $\leqslant r$.

Corollary

A graph property p is not FO-expressible, if the following is true:
For every r there are graphs $\mathcal{A}_r$ and $\mathcal{B}_r$ such that

- $\mathcal{A}_r$ *has property p,*
- $\mathcal{B}_r$ *doesn't have property p, and*
- $\mathcal{A}_r \approx_r \mathcal{B}_r$.

Examples:

- The property of being a linear order of even cardinality is not FO-expressible.
- Connectivity is not EMSO-expressible (Fagin, 1975); not even on linearly ordered graphs (Schwentick, 1996).

Ehrenfeucht-Fraïssé Theorem

Theorem:

$\mathcal{A} \approx_r \mathcal{B} \iff \mathcal{A}$ and $\mathcal{B}$ satisfy the same FO-sentences of quantifier depth $\leqslant r$.

Corollary

A graph property p is not FO-expressible, if the following is true:
For every r there are graphs $\mathcal{A}_r$ and $\mathcal{B}_r$ such that

- $\mathcal{A}_r$ *has property p,*
- $\mathcal{B}_r$ *doesn't have property p, and*
- $\mathcal{A}_r \approx_r \mathcal{B}_r$.

Examples:

- The property of being a linear order of even cardinality is not FO-expressible.
- Connectivity is not EMSO-expressible (Fagin, 1975);
  not even on linearly ordered graphs (Schwentick, 1996).

Note: Finding winning strategies for Duplicator often requires highly non-trivial combinatorial arguments.

Overview

Logical Reductions (1/2)

Use known non-expressibility results for showing new non-expressibility results!

Logical Reductions (1/2)

Use known non-expressibility results for showing new non-expressibility results!

Example:

- Show that the property of being **acyclic** is not FO-definable.
- Use that we already know that being a linear order of even cardinality is not FO-definable.

Logical Reductions (1/2)

Use known non-expressibility results for showing new non-expressibility results!

Example:

- Show that the property of being **acyclic** is not FO-definable.
- Use that we already know that being a linear order of even cardinality is not FO-definable.
- Assume, for contradiction, that acyclicity is FO-definable by a formula φ_{acyclic}.

Logical Reductions (1/2)

Use known non-expressibility results for showing new non-expressibility results!

Example:

- Show that the property of being **acyclic** is not FO-definable.
- Use that we already know that being a linear order of even cardinality is not FO-definable.
- Assume, for contradiction, that acyclicity is FO-definable by a formula φ_{acyclic}.
- Transform φ_{acyclic} into a formula ψ_{even} which, when evaluated in a linear order $\mathcal{A}$, simulates the evaluation of φ_{acyclic} on a graph $G_{\mathcal{A}}$ with

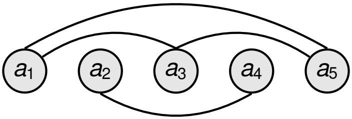
$$G_{\mathcal{A}} \text{ acyclic} \iff \mathcal{A} \text{ has even cardinality.}$$

Logical Reductions (2/2)

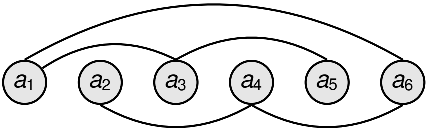
Transform φ_{acyclic} into a formula ψ_{even} which, when evaluated in a linear order $\mathcal{A}$, simulates the evaluation of φ_{acyclic} on a graph $G_{\mathcal{A}}$ with

$$G_{\mathcal{A}} \text{ acyclic} \iff \mathcal{A} \text{ has even cardinality.}$$

$|A| = 5 \implies G_{\mathcal{A}}$:



$|A| = 6 \implies G_{\mathcal{A}}$:



Overview

Introduction

Zero-One Laws

Ehrenfeucht-Fraïssé games

Logical Reductions

Locality Results

Reductions to known results in circuit complexity

The "Algebraic" Approach

Final Remarks

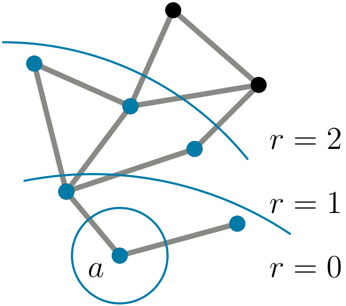
Neighborhoods

Graph $G = (V, E)$

Distance *dist*(u, v) : length of a shortest path between u, v in G.

Shell $S_r(a)$ of nodes at distance exactly r from a.

Ball $N_r(a)$ of radius r at a in G.



Neighborhoods

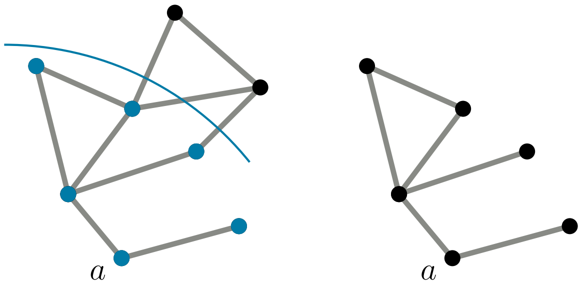
Graph $G = (V, E)$

Distance $dist(u, v)$: length of a shortest path between u, v in G.

Shell $S_r(a)$ of nodes at distance exactly r from a.

Ball $N_r(a)$ of radius r at a in G.

Neighborhood $\mathcal{N}_r(a)$ of radius r at a in G.



Gaifman-local queries

- For a list $a = a_1, \ldots, a_k$ of nodes, $N_r^G(a) = N_r^G(a_1) \cup \cdots \cup N_r^G(a_k)$.
- The r-neighborhood $\mathcal{N}_r^G(a)$ is the structure $(G_{|N_r^G(a)}, a)$ consisting of the induced subgraph of G on $N_r^G(a)$, together with the distinguished nodes a.

Definition: Let q be a k-ary graph query. Let $f : \mathbb{N} \to \mathbb{N}$.
q is called $f(n)$-local if there is an n_0 such that for every $n \geqslant n_0$ and every graph G with $|V^G| = n$, the following is true for all k-tuples a and b of nodes:

$$\text{if} \quad \mathcal{N}_{f(n)}^G(a) \cong \mathcal{N}_{f(n)}^G(b) \quad \text{then} \quad a \in q(G) \iff b \in q(G).$$

Gaifman-locality of FO

Theorem:

- For every graph query q that is FO-definable, there is a constant c such that q is c-local.

(Hella, Libkin, Nurmonen 1990s; Gaifman '82)

Gaifman-locality of FO

Theorem:

- For every graph query q that is FO-definable,
  there is a constant c such that q is c-local.
  (Hella, Libkin, Nurmonen 1990s; Gaifman '82)

- For every graph query q that is FO-definable on ordered graphs
  (for short: q is definable in $<$-invariant FO),
  there is a constant c such that q is c-local.
  (Grohe, Schwentick '98)

Gaifman-locality of FO

Theorem:

- For every graph query q that is FO-definable, there is a constant c such that q is c-local.
  (Hella, Libkin, Nurmonen 1990s; Gaifman '82)

- For every graph query q that is FO-definable on ordered graphs (for short: q is definable in $<$-invariant FO), there is a constant c such that q is c-local.
  (Grohe, Schwentick '98)

- For every graph query q that is FO-definable on graphs with arbitrary numerical predicates (for short: q is definable in Arb-invariant FO), there is a constant c such that q is $(\log n)^c$-local.
  (Anderson, van Melkebeek, S., Segoufin '11)

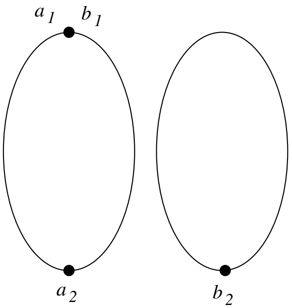
Use locality for proving non-expressibility

Example: The reachability query

$$\text{REACH}(G) := \{ (a_1, a_2) : \text{there is a directed path from } a_1 \text{ to } a_2 \text{ in } G \}$$

is not $\frac{n}{5}$-local an thus cannot be expressed in Arb-invariant FO.

Proof: Consider the graph G:



Use locality for proving non-expressibility

Similarly, one obtains that the following queries are not definable in Arb-invariant FO:

- Does node x lie on a cycle?
- Does node x belong to a connected component that is acyclic?
- Is node x reachable from a node that belongs to a triangle?
- Do nodes x and y have the same distance to node z?

Proof of Gaifman-locality theorem (1/5)

For every query q expressible by Arb-invariant FO, there is a $c \in \mathbb{N}$ such that q is $(\log n)^c$-local.

Proof of Gaifman-locality theorem (1/5)

For every query q expressible by Arb-invariant FO, there is a $c \in \mathbb{N}$ such that q is $(\log n)^c$-local.

Idea: Use known lower bounds in circuit complexity!

Proof of Gaifman-locality theorem (1/5)

For every query q expressible by Arb-invariant FO*, there is a* $c \in \mathbb{N}$ *such that q is* $(\log n)^c$*-local.*

Idea: Use known lower bounds in circuit complexity!

- Let q be expressible by an Arb-invariant FO formula.
- Then, q can be computed by an AC^0 circuit family $\mathcal{C}$ (Immerman '87).

Proof of Gaifman-locality theorem (1/5)

For every query q expressible by Arb-invariant FO, *there is a* $c \in \mathbb{N}$ *such that q is* $(\log n)^c$*-local.*

Idea: Use known lower bounds in circuit complexity!

- Let q be expressible by an Arb-invariant FO formula.
- Then, q can be computed by an AC^0 circuit family $\mathcal{C}$ (Immerman '87).
- Assume that q is *not* $(\log n)^c$-local (for any $c \in \mathbb{N}$), and modify $\mathcal{C}$ to obtain an AC^0 circuit family computing

$$\text{PARITY} := \{w \in \{0,1\}^* : |w|_1 \text{ is even}\}.$$

- This contradicts known lower bounds in circuit complexity theory (Håstad'86).

Proof of Gaifman-locality theorem (2/5)

How to compute a graph query $q(x)$ by an AC^0 circuit family $\mathcal{C}$?

- Represent graph $G = (V, E)$ by a bitstring $\beta(G)$ corresponding to an adjacency matrix for G.

Proof of Gaifman-locality theorem (2/5)

How to compute a graph query $q(x)$ by an AC^0 circuit family $\mathcal{C}$?

- Represent graph $G = (V, E)$ by a bitstring $\beta(G)$ corresponding to an adjacency matrix for G.
- Represent a node $a \in V$ by the bitstring $\beta(a)$ of the form 0^*10^*, carrying the 1 at position i iff node a corresponds to the i-th row/column of the adjacency matrix.

Proof of Gaifman-locality theorem (2/5)

How to compute a graph query $q(x)$ by an AC^0 circuit family $\mathcal{C}$?

- Represent graph $G = (V, E)$ by a bitstring $\beta(G)$ corresponding to an adjacency matrix for G.
- Represent a node $a \in V$ by the bitstring $\beta(a)$ of the form 0^*10^*, carrying the 1 at position i iff node a corresponds to the i-th row/column of the adjacency matrix.
- Let $Rep(G, a)$ be the set of all bitstrings $\beta(G)\beta(a)$, corresponding to all adjacency matrices of G (i.e., all ways of embedding V in $\{1, \ldots, |V|\}$). Thus, $Rep(G, a)$ is the set of all bitstrings representing (G, a).

Proof of Gaifman-locality theorem (2/5)

How to compute a graph query $q(x)$ by an AC^0 circuit family $\mathcal{C}$?

- Represent graph $G = (V, E)$ by a bitstring $\beta(G)$ corresponding to an adjacency matrix for G.
- Represent a node $a \in V$ by the bitstring $\beta(a)$ of the form 0^*10^*, carrying the 1 at position i iff node a corresponds to the i-th row/column of the adjacency matrix.
- Let $Rep(G, a)$ be the set of all bitstrings $\beta(G)\beta(a)$, corresponding to all adjacency matrices of G (i.e., all ways of embedding V in $\{1, \ldots, |V|\}$). Thus, $Rep(G, a)$ is the set of all bitstrings representing (G, a).
- A unary graph query $q(x)$ is computed by a circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ iff the following is true:
  for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $\quad a \in q(G) \iff C_{|\gamma|}$ accepts γ.

Proof of Gaifman-locality theorem (2/5)

How to compute a graph query $q(x)$ by an AC^0 circuit family $\mathcal{C}$?

- Represent graph $G = (V, E)$ by a bitstring $\beta(G)$ corresponding to an adjacency matrix for G.
- Represent a node $a \in V$ by the bitstring $\beta(a)$ of the form 0^*10^*, carrying the 1 at position i iff node a corresponds to the i-th row/column of the adjacency matrix.
- Let $Rep(G, a)$ be the set of all bitstrings $\beta(G)\beta(a)$, corresponding to all adjacency matrices of G (i.e., all ways of embedding V in $\{1, \ldots, |V|\}$). Thus, $Rep(G, a)$ is the set of all bitstrings representing (G, a).
- A unary graph query $q(x)$ is computed by a circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ iff the following is true: for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $\quad a \in q(G) \iff C_{|\gamma|}$ accepts γ.
- *Known:* A unary graph query $q(x)$ is definable in Arb-invariant FO $\iff$ it is computed by a circuit family of constant depth and polynomial size.
  (implicit in Immerman'87)

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO.

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

For contradiction, assume $q(x)$ is not $(\log n)^c$-local, for any $c \in \mathbb{N}$.

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

For contradiction, assume $q(x)$ is not $(\log n)^c$-local, for any $c \in \mathbb{N}$.
Thus: For all c, n_0 there exist $n > n_0$, $G = (V, E)$ with n nodes, $a, b \in V$ such that

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

For contradiction, assume $q(x)$ is not $(\log n)^c$-local, for any $c \in \mathbb{N}$.
Thus: For all c, n_0 there exist $n > n_0$, $G = (V, E)$ with n nodes, $a, b \in V$ such that for $m := (\log n)^c$, $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$, but $a \in q(G)$ and $b \notin q(G)$.

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

For contradiction, assume $q(x)$ is not $(\log n)^c$-local, for any $c \in \mathbb{N}$.
Thus: For all c, n_0 there exist $n > n_0$, $G = (V, E)$ with n nodes, $a, b \in V$ such that for $m := (\log n)^c$, $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$, but $a \in q(G)$ and $b \notin q(G)$.

For simplicity, consider the special case that $dist(a, b) > 2m$.

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

For contradiction, assume $q(x)$ is not $(\log n)^c$-local, for any $c \in \mathbb{N}$.
Thus: For all c, n_0 there exist $n > n_0$, $G = (V, E)$ with n nodes, $a, b \in V$ such that for $m := (\log n)^c$, $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$, but $a \in q(G)$ and $b \notin q(G)$.

For simplicity, consider the special case that $dist(a, b) > 2m$.

> ***Key Lemma:***
> *Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.*
> *Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.*
> *Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.*

Proof of Gaifman-locality theorem (3/5)

Let $q(x)$ be a unary graph query expressible in Arb-invariant FO. Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a circuit family of constant depth d and polynomial size $p(n)$ computing q.
I.e., for all $G = (V, E)$, $a \in V$, $\gamma \in Rep(G, a)$: $a \in q(G) \iff C_{|\gamma|}$ accepts γ.

For contradiction, assume $q(x)$ is not $(\log n)^c$-local, for any $c \in \mathbb{N}$.
Thus: For all c, n_0 there exist $n > n_0$, $G = (V, E)$ with n nodes, $a, b \in V$ such that for $m := (\log n)^c$, $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$, but $a \in q(G)$ and $b \notin q(G)$.

For simplicity, consider the special case that $dist(a, b) > 2m$.

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Theorem: *(Håstad '86)*

There exist $\ell, m_0 > 0$ such that for all $m \geqslant m_0$, no circuit of depth d and size $2^{\ell \cdot m^{1/(d-1)}}$ computes parity on m bits.

Contradiction for $c = 2d$, since $2^{\ell \cdot m^{1/(d-1)}} > 2^{\ell \cdot (\log n)^2} = n^{\ell \log n} > p(n)$. □

Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.

Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$. Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.

Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.

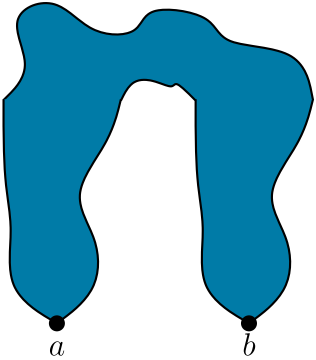
Proof of Gaifman-locality theorem (4/5)

Key Lemma:

*Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.*

Proof:

Consider $w \in \{0, 1\}^m$.



Proof of Gaifman-locality theorem (4/5)

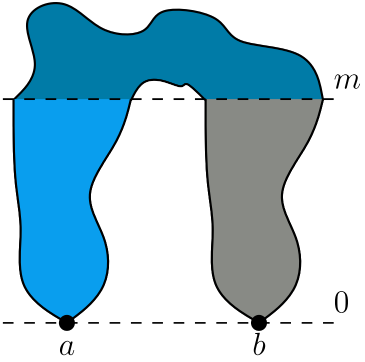
Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$. Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.

Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.



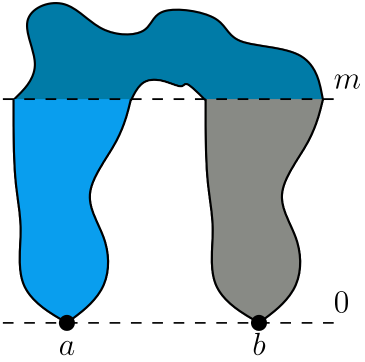
Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.
For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:



Proof of Gaifman-locality theorem (4/5)

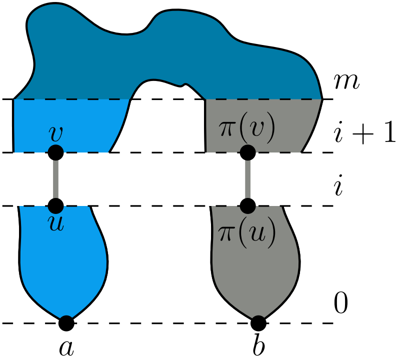
Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:



Proof of Gaifman-locality theorem (4/5)

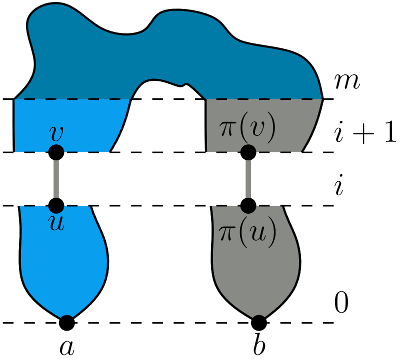
Key Lemma:
*Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.*

Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*



Proof of Gaifman-locality theorem (4/5)

Key Lemma:

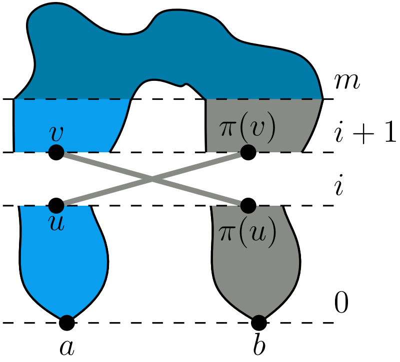
Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*



Proof of Gaifman-locality theorem (4/5)

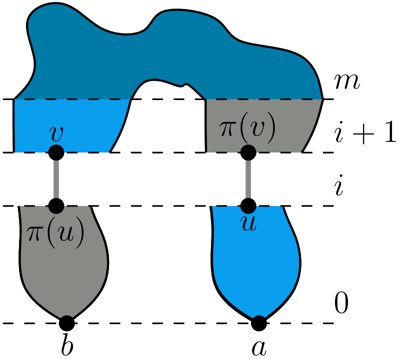
Key Lemma:
Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$. Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*



Proof of Gaifman-locality theorem (4/5)

Key Lemma:

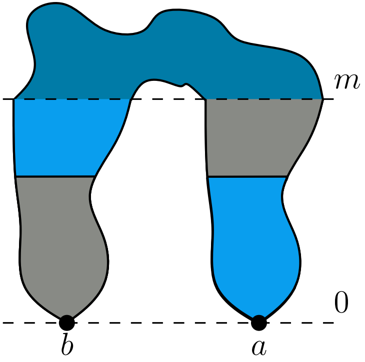
Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*



Proof of Gaifman-locality theorem (4/5)

Key Lemma:

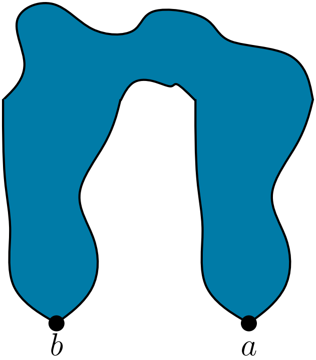
Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.



Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}^G_m(a) \cong \mathcal{N}^G_m(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

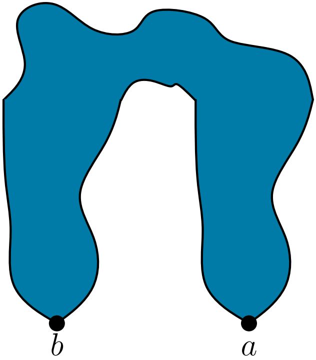
Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \dots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*

The resulting graph $G_w \cong G$.



Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$. Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.

Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

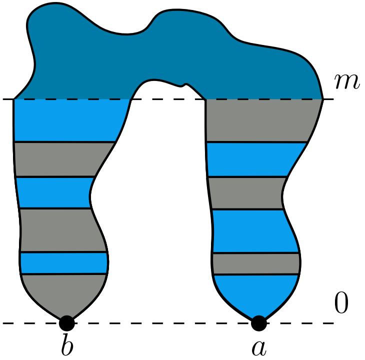
Proof:

Consider $w \in \{0, 1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*

The resulting graph $G_w \cong G$.



Proof of Gaifman-locality theorem (4/5)

Key Lemma:
Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

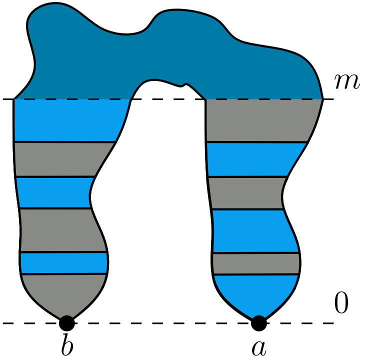
Consider $w \in \{0,1\}^m$.

For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*

The resulting graph $G_w \cong G$.

$$(G_w, a) \cong \begin{cases} (G, a), & \text{if } |w|_1 \text{ even} \\ (G, b), & \text{if } |w|_1 \text{ odd} \end{cases}$$



Proof of Gaifman-locality theorem (4/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

Proof:

Consider $w \in \{0,1\}^m$.

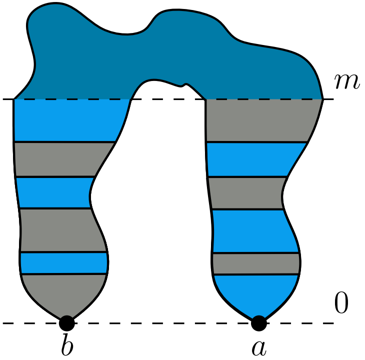
For $i \in \{0, 1, \ldots, m-1\}$ with $w_i = 1$:

> *Swap the endpoints of the edges leaving $N_i(a)$ with the corresponding endpoints of the edges leaving $N_i(b)$.*

The resulting graph $G_w \cong G$.

$$(G_w, a) \cong \begin{cases} (G, a), & \text{if } |w|_1 \text{ even} \\ (G, b), & \text{if } |w|_1 \text{ odd} \end{cases}$$

Circuit C distinguishes these cases.



Proof of Gaifman-locality theorem (5/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$. Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$. Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

How to obtain $\tilde{C}$ from C?

- Consider C for a fixed input string $\gamma \in Rep(G, a)$.
- Fix all input bits (as in γ) that do *not* correspond to potential edges between the shells S_i and S_{i+1}, for $i < m$.

Proof of Gaifman-locality theorem (5/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

How to obtain $\tilde{C}$ from C?

- Consider C for a fixed input string $\gamma \in Rep(G, a)$.
- Fix all input bits (as in γ) that do *not* correspond to potential edges between the shells S_i and S_{i+1}, for $i < m$.
- For all $i < m$ and all $u \in S_i(a)$, $v \in S_{i+1}(a)$ consider the potential edges $e = \{u, v\}$, $e' = \{\pi(u), \pi(v)\}$, $\tilde{e} = \{u, \pi(v)\}$, $\tilde{e}' = \{\pi(u), v\}$.

Proof of Gaifman-locality theorem (5/5)

Key Lemma:

Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

How to obtain $\tilde{C}$ from C?

- Consider C for a fixed input string $\gamma \in Rep(G, a)$.
- Fix all input bits (as in γ) that do *not* correspond to potential edges between the shells S_i and S_{i+1}, for $i < m$.
- For all $i < m$ and all $u \in S_i(a)$, $v \in S_{i+1}(a)$ consider the potential edges $e = \{u, v\}$, $e' = \{\pi(u), \pi(v)\}$, $\tilde{e} = \{u, \pi(v)\}$, $\tilde{e}' = \{\pi(u), v\}$.
- Replace input gates of C as follows:

$$e \text{ by } (e \wedge \neg w_i) \qquad e' \text{ by } (e' \wedge \neg w_i)$$
$$\tilde{e} \text{ by } (e \wedge w_i) \qquad \tilde{e}' \text{ by } (e' \wedge w_i)$$

Proof of Gaifman-locality theorem (5/5)

Key Lemma:
Let $m \in \mathbb{N}$, $G = (V, E)$, $a, b \in V$ such that $\mathcal{N}_m^G(a) \cong \mathcal{N}_m^G(b)$ and $dist(a, b) > 2m$.
Let circuit C accept all strings in $Rep(G, a)$ and reject all strings in $Rep(G, b)$.
Then there is a circuit $\tilde{C}$ of the same size & depth as C computing parity on m bits.

How to obtain $\tilde{C}$ from C?

- Consider C for a fixed input string $\gamma \in Rep(G, a)$.
- Fix all input bits (as in γ) that do *not* correspond to potential edges between the shells S_i and S_{i+1}, for $i < m$.
- For all $i < m$ and all $u \in S_i(a)$, $v \in S_{i+1}(a)$ consider the potential edges $e = \{u, v\}$, $e' = \{\pi(u), \pi(v)\}$, $\tilde{e} = \{u, \pi(v)\}$, $\tilde{e}' = \{\pi(u), v\}$.
- Replace input gates of C as follows:

$$e \text{ by } (e \wedge \neg w_i) \qquad e' \text{ by } (e' \wedge \neg w_i)$$
$$\tilde{e} \text{ by } (e \wedge w_i) \qquad \tilde{e}' \text{ by } (e' \wedge w_i)$$

- This yields a circuit $\tilde{C}$ of the same size and depth as C which, on input $w \in \{0, 1\}^m$ does the same as C on input (G_w, a).
  Thus, $\tilde{C}$ accepts iff $|w|_1$ is even. □

Hanf-local graph properties

- Let $G = (V^G, E^G)$ and $H = (V^H, E^H)$ be two graphs.
- Let $r \in \mathbb{N}$.
- $G \rightleftarrows_r H \quad :\Longleftrightarrow \quad$ there is a bijection $\beta : V^G \to V^H$ such that for every $a \in V^G$

$$N_r^G(a) \;\cong\; N_r^H(\beta(a))$$

Hanf-local graph properties

- Let $G = (V^G, E^G)$ and $H = (V^H, E^H)$ be two graphs.
- Let $r \in \mathbb{N}$.
- $G \rightleftarrows_r H \quad :\Longleftrightarrow \quad$ there is a bijection $\beta : V^G \to V^H$ such that for every $a \in V^G$

$$N_r^G(a) \cong N_r^H(\beta(a))$$

Definition

A graph property p is Hanf $f(n)$-local if there is an n_0 such that for all graphs G and H of size $n \geqslant n_0$ the following is true:

If $G \rightleftarrows_{f(n)} H$ then G has property p iff H has property p.

Hanf-locality of FO

Theorem:

- For every graph property p that is FO-definable, there is a constant c such that p is Hanf c-local.

(Fagin, Stockmeyer, Vardi '95; Hanf '65)

Hanf-locality of FO

Theorem:

- For every graph property p that is FO-definable,
  there is a constant c such that p is Hanf c-local.
  (Fagin, Stockmeyer, Vardi '95; Hanf '65)

- For every property of strings or trees that is definable in $<$-invariant FO,
  there is a constant c such that p is Hanf c-local.
  (Benedikt, Segoufin '09)

Hanf-locality of FO

Theorem:

- For every graph property p that is FO-definable, there is a constant c such that p is Hanf c-local.
  (Fagin, Stockmeyer, Vardi '95; Hanf '65)

- For every property of strings or trees that is definable in $<$-invariant FO, there is a constant c such that p is Hanf c-local.
  (Benedikt, Segoufin '09)

- For every property of strings that is definable in Arb-invariant FO(*Succ*), there is a constant c such that p is Hanf $(\log n)^c$-local.
  (Anderson, van Melkebeek, S., Segoufin '11)

Hanf-locality of FO

Theorem:

- For every graph property p that is FO-definable, there is a constant c such that p is Hanf c-local.
  (Fagin, Stockmeyer, Vardi ’95; Hanf ’65)

- For every property of strings or trees that is definable in $<$-invariant FO, there is a constant c such that p is Hanf c-local.
  (Benedikt, Segoufin ’09)

- For every property of strings that is definable in Arb-invariant FO($Succ$), there is a constant c such that p is Hanf $(\log n)^c$-local.
  (Anderson, van Melkebeek, S., Segoufin ’11)

Example: The class of all strings of the form $c^* a\, c^* b\, c^*$ is not definable in Arb-invariant FO($Succ$).

Overview

Introduction

Zero-One Laws

Ehrenfeucht-Fraïssé games

Logical Reductions

Locality Results

Reductions to known results in circuit complexity

The "Algebraic" Approach

Final Remarks

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

- Seen already in this talk:

  Proof of poly-logarithmic Gaifman-locality of graph queries definable in Arb-invariant FO.

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

- Seen already in this talk:

  Proof of poly-logarithmic Gaifman-locality of graph queries definable in Arb-invariant FO.

- Rossman's proof of the strictness of the bounded variable hierarchy of FO on finite ordered graphs (Rossman '08):

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

- Seen already in this talk:

  Proof of poly-logarithmic Gaifman-locality of graph queries definable in Arb-invariant FO.

- Rossman's proof of the strictness of the bounded variable hierarchy of FO on finite ordered graphs (Rossman '08):
  - Precise (stronger) statement: The existence of a k-clique cannot be expressed by an Arb-invariant FO-sentence using only $\lfloor k/4 \rfloor$ variables.

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

- Seen already in this talk:

  Proof of poly-logarithmic Gaifman-locality of graph queries definable in Arb-invariant FO.

- Rossman's proof of the strictness of the bounded variable hierarchy of FO on finite ordered graphs (Rossman '08):
  - Precise (stronger) statement: The existence of a k-clique cannot be expressed by an Arb-invariant FO-sentence using only $\lfloor k/4 \rfloor$ variables.
  - Main ingredients of the proof:

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

- Seen already in this talk:

  Proof of poly-logarithmic Gaifman-locality of graph queries definable in Arb-invariant FO.

- Rossman's proof of the strictness of the bounded variable hierarchy of FO on finite ordered graphs (Rossman '08):
  - Precise (stronger) statement: The existence of a k-clique cannot be expressed by an Arb-invariant FO-sentence using only $\lfloor k/4 \rfloor$ variables.
  - Main ingredients of the proof:

    (1) Note that for every k-variable Arb-invariant FO-sentence φ there exists a constant depth circuit family $(C_n)_n$ of size n^k such that C_n evaluates φ on graphs of size n.

Reductions to known results in circuit complexity

Idea: Use known lower bounds in circuit complexity to show non-expressibility in certain logics.

Examples:

- Seen already in this talk:

  Proof of poly-logarithmic Gaifman-locality of graph queries definable in Arb-invariant FO.

- Rossman's proof of the strictness of the bounded variable hierarchy of FO on finite ordered graphs (Rossman '08):
  - Precise (stronger) statement: The existence of a k-clique cannot be expressed by an Arb-invariant FO-sentence using only $\lfloor k/4 \rfloor$ variables.
  - Main ingredients of the proof:
    (1) Note that for every k-variable Arb-invariant FO-sentence φ there exists a constant depth circuit family $(C_n)_n$ of size n^k such that C_n evaluates φ on graphs of size n.
    (2) Prove a new lower bound of $\omega(n^{k/4})$ on the size of constant-depth circuits solving the k-clique problem on n-vertex graphs.

Overview

Introduction

Zero-One Laws

Ehrenfeucht-Fraïssé games

Logical Reductions

Locality Results

Reductions to known results in circuit complexity

The “Algebraic” Approach

Final Remarks

The "Algebraic" Approach

Let L_1 and L_2 be logics, and let C be a class of structures.

Goal: Show that L_1 can define exactly the same properties of C-structures as L_2.

Approach:

The "Algebraic" Approach

Let L_1 and L_2 be logics, and let C be a class of structures.

Goal: Show that L_1 can define exactly the same properties of C-structures as L_2.

Approach:

(0) Identify a suitable set of operations $\mathcal{O}$ on structures in C.

The "Algebraic" Approach

Let L_1 and L_2 be logics, and let C be a class of structures.

Goal: Show that L_1 can define exactly the same properties of C-structures as L_2.

Approach:

(0) Identify a suitable set of operations $\mathcal{O}$ on structures in C.

(1) Show that a property p of C-structures is definable in L_1 iff it is closed under every operation $op \in \mathcal{O}$.

The "Algebraic" Approach

Let L_1 and L_2 be logics, and let C be a class of structures.

Goal: Show that L_1 can define exactly the same properties of C-structures as L_2.

Approach:

(0) Identify a suitable set of operations $\mathcal{O}$ on structures in C.

(1) Show that a property p of C-structures is definable in L_1 iff it is closed under every operation $op \in \mathcal{O}$. I.e., for every $\mathcal{A} \in C$:

$$\mathcal{A} \text{ has property } p \iff op(\mathcal{A}) \text{ has property } p.$$

The "Algebraic" Approach

Let L_1 and L_2 be logics, and let C be a class of structures.

Goal: Show that L_1 can define exactly the same properties of C-structures as L_2.

Approach:

(0) Identify a suitable set of operations $\mathcal{O}$ on structures in C.

(1) Show that a property p of C-structures is definable in L_1 iff it is closed under every operation $op \in \mathcal{O}$. I.e., for every $\mathcal{A} \in C$:

$$\mathcal{A} \text{ has property } p \iff op(\mathcal{A}) \text{ has property } p.$$

(2) Show that a property p of C-structures is closed under every operation $op \in \mathcal{O}$ iff it is definable in L_2.

An example

Theorem (Benedikt, Segoufin, '09):
A string-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*).

An example

Theorem (Benedikt, Segoufin, '09):

A string-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*).

Main ingredients of the proof:

- Use a result by Beauquier and Pin (1989) stating that a string-language is definable in FO(*Succ*) iff it is aperiodic and closed under swaps.

An example

Theorem (Benedikt, Segoufin, '09):
A string-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*).

Main ingredients of the proof:

- Use a result by Beauquier and Pin (1989) stating that a string-language is definable in FO(*Succ*) iff it is aperiodic and closed under swaps.
  - A string language L is aperiodic iff there exists a number $\ell \in \mathbb{N}$ such that for all strings u, x, v we have

$$u\, x^{\ell}\, v \in L \quad \Longleftrightarrow \quad u\, x^{\ell+1}\, v \in L.$$

An example

Theorem (Benedikt, Segoufin, ’09):
A string-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*).

Main ingredients of the proof:

- Use a result by Beauquier and Pin (1989) stating that a string-language is definable in FO(*Succ*) iff it is aperiodic and closed under swaps.
  - A string language L is aperiodic iff there exists a number $\ell \in \mathbb{N}$ such that for all strings u, x, v we have

$$u\, x^{\ell}\; v \in L \quad\iff\quad u\, x^{\ell+1}\; v \in L.$$

  - L is closed under swaps iff for all strings u, v, e, x, y, z such that e, f are idempotents (i.e., for all u, v we have $uev \in L$ iff $ue^2v \in L$), we have

$$u\; e\,x\,f\; y\; e\,z\,f\; v \;\in\; L \quad\iff\quad u\; e\,z\,f\; y\; e\,x\,f\; v \;\in\; L.$$

An example

Theorem (Benedikt, Segoufin, '09):

A string-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*).

Main ingredients of the proof:

- Use a result by Beauquier and Pin (1989) stating that a string-language is definable in FO(*Succ*) iff it is aperiodic and closed under swaps.
  - A string language L is aperiodic iff there exists a number $\ell \in \mathbb{N}$ such that for all strings u, x, v we have

$$u\, x^{\ell}\, v \in L \iff u\, x^{\ell+1}\, v \in L.$$

  - L is closed under swaps iff for all strings u, v, e, x, y, z such that e, f are idempotents (i.e., for all u, v we have $uev \in L$ iff $ue^2v \in L$), we have

$$u\; e\,x\,f\; y\; e\,z\,f\; v \;\in\; L \iff u\; e\,z\,f\; y\; e\,x\,f\; v \;\in\; L.$$

- Show that every string-language definable in $<$-invariant FO(*Succ*) is aperiodic and closed under swaps.

  (For this, you can use Ehrenfeucht-Fraïssé games.)

Some further results proved using this method

Theorem:

- A tree-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*). (Benedikt, Segoufin '09)
  (They use aperiodicity and closure under guarded swaps.)

Some further results proved using this method

Theorem:

- A tree-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*). (Benedikt, Segoufin '09)
  (They use aperiodicity and closure under guarded swaps.)

- A colored finite set is definable in $+$-invariant FO iff it is definable in FO_{card} (i.e., FO with predicates testing the cardinality of the universe modulo fixed numbers). (S., Segoufin '10)

Some further results proved using this method

Theorem:

- A tree-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*). (Benedikt, Segoufin '09)
  (They use aperiodicity and closure under guarded swaps.)

- A colored finite set is definable in $+$-invariant FO iff it is definable in FO_{card} (i.e., FO with predicates testing the cardinality of the universe modulo fixed numbers). (S., Segoufin '10)

- A regular string- or tree-language is definable in $+$-invariant FO(*Succ*) iff it is definable in $\text{FO}_{\text{card}}(\mathit{succ})$. (S., Segoufin '10 and Harwath, S. '12)
  (They use closure under transfers and closure under guarded swaps.)

Some further results proved using this method

Theorem:

- A tree-language is definable in $<$-invariant FO(*Succ*) iff it is definable in FO(*Succ*). (Benedikt, Segoufin '09)
  (They use aperiodicity and closure under guarded swaps.)

- A colored finite set is definable in $+$-invariant FO iff it is definable in FO_{card} (i.e., FO with predicates testing the cardinality of the universe modulo fixed numbers). (S., Segoufin '10)

- A regular string- or tree-language is definable in $+$-invariant FO(*Succ*) iff it is definable in $\text{FO}_{\text{card}}(\mathit{succ})$. (S., Segoufin '10 and Harwath, S. '12)
  (They use closure under transfers and closure under guarded swaps.)

- A regular string-language is definable in Arb-invariant FO(*Succ*) iff it is definable in $\text{FO}_{\text{card}}(\mathit{Succ})$. (Anderson, van Melkebeek, S., Segoufin '11)

Overview

Gaifman-locality

If $\mathcal{N}_r^G(a) \cong \mathcal{N}_r^G(b)$ then $\big(a \in q(G) \iff b \in q(G)\big)$.

Known:

- Queries definable in order-invariant FO are Gaifman-local with respect to a constant locality radius. (Grohe, Schwentick '98)
- Queries definable in Arb-invariant FO are Gaifman-local with respect to a poly-logarithmic locality radius. (Anderson, Melkebeek, S., Segoufin '11)

Open Question:

- How about addition-invariant FO — is it Gaifman-local with respect to a constant locality radius?

Hanf-locality

A graph property *p* is Hanf-local w.r.t. locality radius *r*, if any two graphs having the same *r*-neighbourhood types with the same multiplicities, are not distinguished by *p*.

Known:

- Properties of graphs definable in FO are Hanf-local w.r.t. a constant locality radius. (Fagin, Stockmeyer, Vardi '95)
- Properties of strings or trees definable by order-invariant FO are Hanf-local w.r.t. a constant locality radius. (Benedikt, Segoufin '09)
- Properties of strings definable by Arb-invariant FO are Hanf-local w.r.t. a poly-logarithmic locality radius. (Anderson, van Melkebeek, S., Segoufin '11)

Open Question:

- Do these results generalise from strings to arbitrary finite graphs?

Decidable Characterisations

Open Question:

Are there decidable characterisations of

- order-invariant FO?
- addition-invariant FO?
- $(+, \times)$-invariant FO?

Known:

- On finite strings and trees: order-invariant FO $\equiv$ FO. (Benedikt, Segoufin '10)
- On finite coloured sets: addition-invariant FO $\equiv$ FO enriched by "cardinality modulo" quantifiers. (S., Segoufin '10)

Thank You!