Finite Model Theory
Unit 3

Dan Suciu

Spring 2018

599c: Finite Model Theory

Unit 3: Logic and Complexity

Resources

- Libkin, *Finite Model Theory*
- Immerman, *Descriptive Complexity* (Ch.3)
- Grädel, Kolaitis, Vardi, *On the Decision Problem for Two-Variable First-Order Logic.*
- Vardi, *Why is Modal Logic so Robustly Decidable?*
- Halpern, Harper, Immerman, Kolaitis, Vardi, Vianu, *On the Unusual Effectiveness of Logic in Computer Science*

Logic and Complexity

Two problems:

- Satisfiability: given φ, does it have a (finite) model $\boldsymbol{A}$?
- Model checking: given a finite $\boldsymbol{A}$ and φ, is $\boldsymbol{A}$ a model of φ?

Trakhtenbrot's Theorem

A sentence φ is finitely satisfiable if there exists a finite model $\boldsymbol{A}$.

Theorem (Trakhtenbrot)

Suppose the vocabulary σ has at least one relation with arity ≥ 2. Then the problem "given φ check if it is finitely satisfiable" is undecidable.

What about unary vocabularies? $\Rightarrow$ Homework!

Before we prove it, let's see some consequences.

Trakthenbrot's Theorem: Consequence 1

Denote $\varphi \equiv_{\text{fin}} \psi$ if φ, ψ are equivalent on all finite structures:

Corollary

If the vocabulary σ has at least one relation with arity ≥ 2, then the following problem is undecidable: "given two sentences φ, ψ, check whether $\varphi \equiv_{fin} \psi$.".

Proof in class

Trakthenbrot's Theorem: Consequence 1

Denote $\varphi \equiv_{\text{fin}} \psi$ if φ, ψ are equivalent on all finite structures:

Corollary

If the vocabulary σ has at least one relation with arity ≥ 2, then the following problem is undecidable: "given two sentences φ, ψ, check whether $\varphi \equiv_{fin} \psi$.".

Proof in class

Proof: Reduce it to UNSAT. Assuming we have an oracle for $\varphi \equiv_{\text{fin}} \psi$, we can check UNSAT by checking if $\varphi \equiv_{\text{fin}} \boldsymbol{F}$.

Trakthenbrot's Theorem: Consequence 2

Let $f : \mathbb{N} \to \mathbb{N}$ a function with the following property:
every finitely satisfiable sentence φ has a model of size $\leq f(|\varphi|)$.

Corollary

If the vocabulary σ has at least one relation with arity ≥ 2, then no computable function f exists with the property above.

Proof in class

Trakthenbrot's Theorem: Consequence 2

Let $f : \mathbb{N} \to \mathbb{N}$ a function with the following property:
every finitely satisfiable sentence φ has a model of size $\leq f(|\varphi|)$.

Corollary

If the vocabulary σ has at least one relation with arity ≥ 2, then no computable function f exists with the property above.

Proof in class

Proof: If we had such an f, then we can check finite satisfiability as follows. Given φ, compute $n = f(|\varphi|)$, and try out all structures of size $\leq n$:

- If one of the structures is a model then answer YES.
- Otherwise answer NO.

Discussion

Simple fact:

Fact

The set of finitely satisfiable sentences φ is recursively enumerable.

Why?

Discussion

Simple fact:

Fact

The set of finitely satisfiable sentences φ is recursively enumerable.

Why?

Proof: for each $n = 1, 2, 3, \ldots$ enumerate all structures $\boldsymbol{A}$ of size $\leq n$, and all $FO[n]$ sentences φ that are true in $\boldsymbol{A}$.

What is $FO[n]$? Is it finite?

Discussion

Simple fact:

Fact

The set of finitely satisfiable sentences φ is recursively enumerable.

Why?

Proof: for each $n = 1, 2, 3, \ldots$ enumerate all structures $\boldsymbol{A}$ of size $\leq n$, and all $FO[n]$ sentences φ that are true in $\boldsymbol{A}$.

What is $FO[n]$? Is it finite?

It is FO restricted to quantifier rank n, and we know it is finite.

Trakthenbrot's Theorem: Consequence 3

"Finiteness is not axiomatizable."

We say that φ is finitely valid, $\models_{\text{fin}} \varphi$, if it holds in every finite model $\boldsymbol{A}$.

Corollary

There is no r.e. set of axioms Σ such that $\Sigma \vdash \varphi$ iff $\models_{fin} \varphi$.

Proof in class

Trakthenbrot's Theorem: Consequence 3

"Finiteness is not axiomatizable."

We say that φ is finitely valid, $\models_{\text{fin}} \varphi$, if it holds in every finite model $\boldsymbol{A}$.

Corollary

There is no r.e. set of axioms Σ such that $\Sigma \vdash \varphi$ iff $\models_{fin} \varphi$.

Proof in class

Proof:

- By the previous fact, the set of finitely satisfiable sentences φ is r.e.
- Hence, the set of finitely valid sentences is co-r.e. (since $\models_{\text{fin}} \varphi$ iff $\neg\varphi$ is not finitely satisfiable).
- Since Σ is r.e. the set $\{\varphi \mid \Sigma \vdash \varphi\}$ is r.e.
- If $\Sigma \vdash \varphi$ iff $\models_{\text{fin}} \varphi$ then this set is both r.e. and co-r.e., hence it is decidable. why?

Proof of Trakhtenbrot's Theorem

By reduction from the Halting Problem:

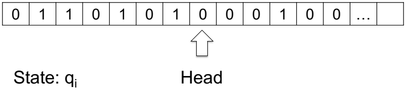
- Given a Turing Machine M, does M halt on the empty input?

The proof consist of the following: given M we will construct a sentence φ_M s.t. M halts iff φ_M is finitely satisfiable.

Review: Turing Machines Basics

$M = (Q, \Sigma, \Delta, q_0, Q_F)$ where:

- $Q = \{q_0, q_1, \ldots, q_m\}$ are the states;
  q_0 is the initial state;
  $Q_F \subseteq Q$ are the final states.
- Σ is the tape alphabet; we take $\Sigma = \{0, 1\}$
- $\Delta \subseteq Q \times \Sigma \times \Sigma \times \{\text{Left}, \text{Right}\} \times Q$ are the transitions.



Review: Turing Machines Basics

A configuration is a triple $c = (w, h, q)$ where:

- $w \subseteq \Sigma^*$ is a tape content.
- $h \in \mathbb{N}$ is the head position.
- $q \in Q$ is a state.

An accepting computation is a sequence $\boldsymbol{C} = c_1, c_2, \ldots, c_T$ where:

- Each c_i is a configuration.
- c_1 is the initial configuration what does that mean?
- c_T is a final configuration what does that mean?
- Forall t, (c_t, c_{t+1}) is a valid transition what does that mean?

Review: Turing Machines Basics

A configuration is a triple $c = (w, h, q)$ where:

- $w \subseteq \Sigma^*$ is a tape content.
- $h \in \mathbb{N}$ is the head position.
- $q \in Q$ is a state.

An accepting computation is a sequence $\boldsymbol{C} = c_1, c_2, \ldots, c_T$ where:

- Each c_i is a configuration.
- c_1 is the initial configuration what does that mean?
- c_T is a final configuration what does that mean?
- Forall t, (c_t, c_{t+1}) is a valid transition what does that mean?

Review: Turing Machines Basics

A configuration is a triple $c = (w, h, q)$ where:

- $w \subseteq \Sigma^*$ is a tape content.
- $h \in \mathbb{N}$ is the head position.
- $q \in Q$ is a state.

An accepting computation is a sequence $\boldsymbol{C} = c_1, c_2, \ldots, c_T$ where:

- Each c_i is a configuration.
- c_1 is the initial configuration what does that mean?
- c_T is a final configuration what does that mean?
- Forall t, (c_t, c_{t+1}) is a valid transition what does that mean?

Review: Turing Machines Basics

A configuration is a triple $c = (w, h, q)$ where:

- $w \subseteq \Sigma^*$ is a tape content.
- $h \in \mathbb{N}$ is the head position.
- $q \in Q$ is a state.

An accepting computation is a sequence $\boldsymbol{C} = c_1, c_2, \dots, c_T$ where:

- Each c_i is a configuration.
- c_1 is the initial configuration what does that mean?
- c_T is a final configuration what does that mean?
- Forall t, (c_t, c_{t+1}) is a valid transition what does that mean?

Review: Turing Machines Basics

A configuration is a triple $c = (w, h, q)$ where:

- $w \subseteq \Sigma^*$ is a tape content.
- $h \in \mathbb{N}$ is the head position.
- $q \in Q$ is a state.

An accepting computation is a sequence $\boldsymbol{C} = c_1, c_2, \ldots, c_T$ where:

- Each c_i is a configuration.
- c_1 is the initial configuration what does that mean?
- c_T is a final configuration what does that mean?
- Forall t, (c_t, c_{t+1}) is a valid transition what does that mean?

Proof Plan

M halts iff

$$\boxed{\exists \boldsymbol{C},\ \boldsymbol{C} \text{ is an accepting computation of } M.}$$

φ is finitely satisfiable iff

$$\boxed{\exists \boldsymbol{A} \text{ such that } \boldsymbol{A} \models \varphi.}$$

This suggests the proof plan:

- Computation $\boldsymbol{C} \equiv$ structure $\boldsymbol{A}$.
- $\boldsymbol{C}$ is an accepting computation iff $\boldsymbol{A}$ is a model of φ.

Proof Plan

M halts iff

$$\boxed{\exists \boldsymbol{C},\ \boldsymbol{C} \text{ is an accepting computation of } M.}$$

φ is finitely satisfiable iff

$$\boxed{\exists \boldsymbol{A} \text{ such that } \boldsymbol{A} \models \varphi.}$$

This suggests the proof plan:

- Computation $\boldsymbol{C} \equiv$ structure $\boldsymbol{A}$.
- $\boldsymbol{C}$ is an accepting computation iff $\boldsymbol{A}$ is a model of φ.

Proof Plan

M halts iff

$$\boxed{\exists \boldsymbol{C},\ \boldsymbol{C} \text{ is an accepting computation of } M.}$$

φ is finitely satisfiable iff

$$\boxed{\exists \boldsymbol{A} \text{ such that } \boldsymbol{A} \models \varphi.}$$

This suggests the proof plan:

- Computation $\boldsymbol{C} \equiv$ structure $\boldsymbol{A}$.
- $\boldsymbol{C}$ is an accepting computation iff $\boldsymbol{A}$ is a model of φ.

Proof Details

Fix a Turing Machine M.

- Describe a vocabulary σ_M and sentence φ_M whose models correspond precisely to accepting computations of M.
- Describe an FO encoding of σ_M and φ_M into a single binary relation.

Proof Details

Fix $M = (Q, \{0,1\}, \Delta, q_0, Q_F)$.
Define: $\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q \in Q})$

Intended meaning:

- $<$ is a total order
- $T_0(t,p), T_1(t,p)$: the tape content at time t position p is 0 or 1.
- $H(t,p)$: the head at time t is on position p.
- $S_q(t)$: the Turning Machine is is stated q at time t.

Proof Details

$$M = (Q, \{0,1\}, \Delta, q_0, Q_F)$$
$$\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q\in Q})$$

The sentence φ_M asserts the following:

- General consistency: $<$ is a total order, every tape has exactly one symbol, the head is on exactly one position, etc.
- At time $t = \min$, the TM is in the initial configuration.
- At time $t = \max$, the TM is in an accepting configuration.
- Every transition from t to $t+1$ is correct

details in class (also next slides)

Proof Details: General Consistency

$$M = (Q, \{0,1\}, \Delta, q_0, Q_F)$$
$$\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q \in Q})$$

- $<$ is a total order.
- Exactly one tape symbol:
  $\forall t, \forall p (T_0(t,p) \vee T_1(t,p)) \wedge \neg(T_0(t,p) \wedge T_1(t,p))$
- Exactly one head position at each time: ...
- Exactly one state at each time: ...

Proof Details: Initial Configuration

$$M = (Q, \{0,1\}, \Delta, q_0, Q_F)$$
$$\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q\in Q})$$

At time $t = \min$, the TM is in the initial configuration:

$$\forall p T_0(\min, p) \wedge H(\min, \min) \wedge S_{q_0}(\min)$$

Note that we can name min by $\exists x \neg \exists y (y < x)$; similarly max.

Proof Details: Final Configuration

$$M = (Q, \{0,1\}, \Delta, q_0, Q_F)$$
$$\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q\in Q})$$

At time $t = \max$, the TM is in the final configuration:

$$\bigvee_{q\in Q_F} S_q(\max)$$

Proof Details: All Transitions are Correct

$$M = (Q, \{0,1\}, \Delta, q_0, Q_F)$$
$$\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q \in Q})$$

Each transition from t to $t+1$ corresponds to one valid $\delta \in \Delta$:

$$\forall t (t < \max \rightarrow \bigvee_{\delta \in \Delta} \mathrm{CHECK}_\delta(t))$$

Proof Details: All Transitions are Correct (Detail)

$$M = (Q, \{0,1\}, \Delta, q_0, Q_F)$$
$$\sigma_M = (<, T_0(\cdot,\cdot), T_1(\cdot,\cdot), H(\cdot,\cdot), (S_q(\cdot))_{q \in Q})$$

Example transition: $\delta = (q_5, 1, 0, \text{Left}, q_3)$
("If in state q_5 and the tape is 1, then write 0, move Left, enter q_3")

$\text{CHECK}_\delta(t) = S_{q_5}(t)$	Check we are in q_5
$\wedge \forall s(\neg H(t,s) \to (T_0(t,s) \leftrightarrow T_0(t+1,s)))$	Leave non-head symbols unchaged
$\wedge \forall s(H(t,s) \to T_1(t,s) \wedge T_0(t+1,s))$	the head was 1 set it to 0
$\wedge H(t+1, s-1)$	move to the left
$\wedge S_{q_3}(t+1)$	enter q_3

Discussion

- A structure s.t. $\boldsymbol{A} \vDash \varphi_M$ is precisely a successful computation of the Turing Machine M.
- How large is $|A|$, the domain of $\boldsymbol{A}$?

Discussion

- A structure s.t. $\boldsymbol{A} \vDash \varphi_M$ is precisely a successful computation of the Turing Machine M.
- How large is $|A|$, the domain of $\boldsymbol{A}$? The number of time steps required by M.
- Is $\boldsymbol{A}$ unique?

Discussion

- A structure s.t. $\boldsymbol{A} \models \varphi_M$ is precisely a successful computation of the Turing Machine M.
- How large is $|A|$, the domain of $\boldsymbol{A}$? The number of time steps required by M.
- Is $\boldsymbol{A}$ unique? Not necessarily. But it is unique when M is deterministic.

Discussion

- A structure s.t. $\boldsymbol{A} \vDash \varphi_M$ is precisely a successful computation of the Turing Machine M.
- How large is $|A|$, the domain of $\boldsymbol{A}$? The number of time steps required by M.
- Is $\boldsymbol{A}$ unique? Not necessarily. But it is unique when M is deterministic.
- Is succ enough, or do we need $<$?

Discussion

- A structure s.t. $\boldsymbol{A} \models \varphi_M$ is precisely a successful computation of the Turing Machine M.
- How large is $|A|$, the domain of $\boldsymbol{A}$? The number of time steps required by M.
- Is $\boldsymbol{A}$ unique? Not necessarily. But it is unique when M is deterministic.
- Is $\mathtt{succ}$ enough, or do we need $<$? $\mathtt{succ}$ is not finitely axiomatizable.

Discussion

- A structure s.t. $\boldsymbol{A} \models \varphi_M$ is precisely a successful computation of the Turing Machine M.
- How large is $|A|$, the domain of $\boldsymbol{A}$? The number of time steps required by M.
- Is $\boldsymbol{A}$ unique? Not necessarily. But it is unique when M is deterministic.
- Is $\mathtt{succ}$ enough, or do we need $<$? $\mathtt{succ}$ is not finitely axiomatizable.
- We still need to reduce the vocabulary σ_M to a vocabulary with a single binary relation E.

FO Reduction

Let $\sigma = \{S_1, \ldots, S_m\}, \tau = \{T_1, \ldots, T_n\}$ be two relational vocabularies.

A query from σ to τ is a function $Q : \text{STRUCT}[\sigma] \rightarrow \text{STRUCT}[\tau]$.

A Boolean query, or a problem, is a function $P : \text{STRUCT}[\sigma] \rightarrow \{0,1\}$.

A First Order Query Q consists of n formulas, $Q = (q_1, \ldots, q_n)$, where each q_j has arity(T_j) free variables; it defines the mapping $Q(\boldsymbol{A}) \stackrel{\text{def}}{=} \boldsymbol{B}$ where:

$$B \stackrel{\text{def}}{=} A \qquad \text{same domain}$$

$$\forall j : \quad T_j^B \stackrel{\text{def}}{=} \{\boldsymbol{b} \mid \boldsymbol{A} \models q_j(\boldsymbol{b})\}$$

Q maps problems on STRUCT$[\tau]$ to problems on STRUCT$[\sigma]$ ("in reverse"): $P \mapsto P \circ Q$, i.e. $\hat{P}(\boldsymbol{A}) \stackrel{\text{def}}{=} P(Q(\boldsymbol{A}))$.

FO Reduction

Let $\sigma = \{S_1, \ldots, S_m\}, \tau = \{T_1, \ldots, T_n\}$ be two relational vocabularies.

A query from σ to τ is a function $Q : \text{STRUCT}[\sigma] \to \text{STRUCT}[\tau]$.

A Boolean query, or a problem, is a function $P : \text{STRUCT}[\sigma] \to \{0, 1\}$.

A First Order Query Q consists of n formulas, $Q = (q_1, \ldots, q_n)$, where each q_j has arity(T_j) free variables; it defines the mapping $Q(\boldsymbol{A}) \stackrel{\text{def}}{=} \boldsymbol{B}$ where:

$$B \stackrel{\text{def}}{=} A \qquad \text{same domain}$$

$$\forall j : \quad T_j^B \stackrel{\text{def}}{=} \{\boldsymbol{b} \mid \boldsymbol{A} \models q_j(\boldsymbol{b})\}$$

Q maps problems on STRUCT$[\tau]$ to problems on STRUCT$[\sigma]$ ("in reverse"): $P \mapsto P \circ Q$, i.e. $\hat{P}(\boldsymbol{A}) \stackrel{\text{def}}{=} P(Q(\boldsymbol{A}))$.

FO Reduction

Let $\sigma = \{S_1, \ldots, S_m\}, \tau = \{T_1, \ldots, T_n\}$ be two relational vocabularies.

A query from σ to τ is a function $Q : \text{STRUCT}[\sigma] \to \text{STRUCT}[\tau]$.

A Boolean query, or a problem, is a function $P : \text{STRUCT}[\sigma] \to \{0, 1\}$.

A First Order Query Q consists of n formulas, $Q = (q_1, \ldots, q_n)$, where each q_j has arity(T_j) free variables; it defines the mapping $Q(\boldsymbol{A}) \stackrel{\text{def}}{=} \boldsymbol{B}$ where:

$$B \stackrel{\text{def}}{=} A \qquad \text{same domain}$$

$$\forall j : \quad T_j^B \stackrel{\text{def}}{=} \{\boldsymbol{b} \mid \boldsymbol{A} \models q_j(\boldsymbol{b})\}$$

Q maps problems on STRUCT$[\tau]$ to problems on STRUCT$[\sigma]$ ("in reverse"): $P \mapsto P \circ Q$, i.e. $\hat{P}(\boldsymbol{A}) \stackrel{\text{def}}{=} P(Q(\boldsymbol{A}))$.

FO Reduction

Let $\sigma = \{S_1, \ldots, S_m\}, \tau = \{T_1, \ldots, T_n\}$ be two relational vocabularies.

A query from σ to τ is a function $Q : \text{STRUCT}[\sigma] \to \text{STRUCT}[\tau]$.

A Boolean query, or a problem, is a function $P : \text{STRUCT}[\sigma] \to \{0, 1\}$.

A First Order Query Q consists of n formulas, $Q = (q_1, \ldots, q_n)$, where each q_j has arity(T_j) free variables; it defines the mapping $Q(\boldsymbol{A}) \stackrel{\text{def}}{=} \boldsymbol{B}$ where:

$$B \stackrel{\text{def}}{=} A \qquad \text{same domain}$$

$$\forall j : \quad T_j^B \stackrel{\text{def}}{=} \{\boldsymbol{b} \mid \boldsymbol{A} \vDash q_j(\boldsymbol{b})\}$$

Q maps problems on STRUCT$[\tau]$ to problems on STRUCT$[\sigma]$ ("in reverse"): $P \mapsto P \circ Q$, i.e. $\hat{P}(\boldsymbol{A}) \stackrel{\text{def}}{=} P(Q(\boldsymbol{A}))$.

FO Reduction

Let $\sigma = \{S_1, \ldots, S_m\}, \tau = \{T_1, \ldots, T_n\}$ be two relational vocabularies.

A query from σ to τ is a function $Q : \mathrm{STRUCT}[\sigma] \to \mathrm{STRUCT}[\tau]$.

A Boolean query, or a problem, is a function $P : \mathrm{STRUCT}[\sigma] \to \{0, 1\}$.

A First Order Query Q consists of n formulas, $Q = (q_1, \ldots, q_n)$, where each q_j has arity(T_j) free variables; it defines the mapping $Q(\boldsymbol{A}) \stackrel{\text{def}}{=} \boldsymbol{B}$ where:

$$B \stackrel{\text{def}}{=} A \qquad\qquad \text{same domain}$$

$$\forall j : \quad T_j^B \stackrel{\text{def}}{=} \{\boldsymbol{b} \mid \boldsymbol{A} \models q_j(\boldsymbol{b})\}$$

Q maps problems on $\mathrm{STRUCT}[\tau]$ to problems on $\mathrm{STRUCT}[\sigma]$ ("in reverse"): $P \mapsto P \circ Q$, i.e. $\hat{P}(\boldsymbol{A}) \stackrel{\text{def}}{=} P(Q(\boldsymbol{A}))$.

FO Reduction

Query $\text{STRUCT}[\sigma] \rightarrow \text{STRUCT}[\tau]$
(Problems on $\text{STRUCT}[\tau]$) $\rightarrow$ (Problems on $\text{STRUCT}[\sigma]$)

Definition

A First Order Reduction is an FO query Q from σ to τ.
It "reduces" a problem P' on τ from the problem $P \stackrel{\text{def}}{=} P' \circ Q$ on σ.

Obviously, P' is at least as hard as P.

Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

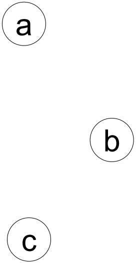
Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$

Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$



R

a	b
a	c
c	a

S

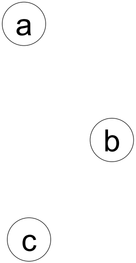
a	c
c	c

Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$



R

a	b
a	c
c	a

S

a	c
c	c

Use small gadgets for R and S

Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$

a

b

c

R

a	b
a	c
c	a

S

a	c
c	c

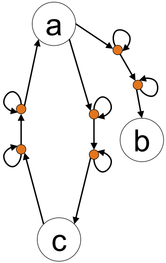
Use small gadgets for R and S

Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$



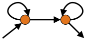
R

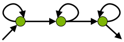
a	b
a	c
c	a

S

a	c
c	c

Use small gadgets for R and S



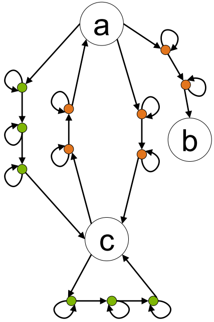


Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$



R

a	b
a	c
c	a

S

a	c
c	c

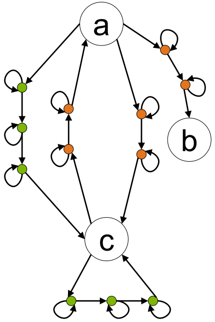
Use small gadgets for R and S

Every Structure is FO-Reducible to a Graph

$\sigma = \{E\}$ a graph.

τ = any vocabulary. For simplicity, assume $\tau = \{R(\cdot,\cdot), S(\cdot,\cdot)\}$.

Question: Given a τ-structure $\boldsymbol{A} = (R^A, S^A)$, encode it as a graph G s.t. you can decode it: $R^A = Q_1(\boldsymbol{G})$, $S^A = Q_2(\boldsymbol{G})$



R

a	b
a	c
c	a

S

a	c
c	c

Use small gadgets for R and S

The query (1) first checks that G is a correct encoding how?, then (2) decodes R and S how?

Summary of Trakhtenbrot's Theorem

Assume an oracle: given φ, check if φ has a finite model $\boldsymbol{A}$.
We reduce the halting problem for a Turning Machine M.

- Construct the vocabulary σ_M and the sentence φ_M that says "the model $\boldsymbol{A}$ represents an accepting computation of M.
- Consider the FO reduction Q from a graph $\{E\}$ to σ_M, and denote $\psi_M = \varphi \circ Q$. This is a sentence over the vocabulary $\{E\}$.
- Claim: ψ_M is satisfiable iff M terminates. Proof:
  - If M terminates, then there exists a model $\boldsymbol{A} \vDash \varphi_M$. From $\boldsymbol{A}$, we construct a graph encoding $\boldsymbol{G}$ s.t. $Q(\boldsymbol{G}) = \boldsymbol{A}$. This is a model of ψ_M.
  - If ψ_M has a model $\boldsymbol{G}$ then: (a) if $\boldsymbol{G}$ is an invalid encoding, then $Q(\boldsymbol{G})$ returns the empty structure $\boldsymbol{A}$, which is not a model of φ_M. (b) otherwise, $\boldsymbol{G}$ is a valid encoding of some structure $\boldsymbol{A}$, which, in turn, represents an accepting computation.

Discussion

Satisfiability in the finite or in general (finite or infinite) are quite different!

- The problem "given φ, is φ finitely satisfiable?" is r.e. why?
- The problem "given φ, is φ satisfiable?" is co-r.e. why?

The Finite Model Property

Let $L \subseteq FO$ be a subset of FO.

Definition

We say that L has the finite model property, or it is *finitely controllable* if: $\forall \varphi \in L$, φ has a model iff φ has a finite model.

Definition

We say that L has the small model property if there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ s.t.
$\forall \varphi \in L$, φ has a model iff φ has a finite model of size $\leq f(|\varphi|)$.

The Finite Model Property

Let $L \subseteq FO$ be a subset of FO.

Definition

We say that L has the finite model property, or it is *finitely controllable* if:
$\forall \varphi \in L$, φ has a model iff φ has a finite model.

Definition

We say that L has the small model property if there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ s.t.
$\forall \varphi \in L$, φ has a model iff φ has a finite model of size $\leq f(|\varphi|)$.

The Finite Model Property Implies Decidability

Theorem

If L has the small model property then L is decidable.

To check $SAT(\varphi)$ enumerate all structures up to size $f(|\varphi|)$;
if any is a model return YES, if none is a model return NO.

Theorem

If L has the finite model property then L is decidable.

To check $SAT(\varphi)$ enumerate all finite structures $\boldsymbol{A}$ AND all proofs $\vdash \psi$:

- If φ is SAT it is also finitely satisfiable, hence some model $\boldsymbol{A}$ will show up in the first list; answer YES.
- If φ is UNSAT then $\neg\varphi$ will show up in the second list; answer NO.

The Finite Model Property Implies Decidability

Theorem

If L has the small model property then L is decidable.

To check $SAT(\varphi)$ enumerate all structures up to size $f(|\varphi|)$;
if any is a model return YES, if none is a model return NO.

Theorem

If L has the finite model property then L is decidable.

To check $SAT(\varphi)$ enumerate all finite structures $\boldsymbol{A}$ AND all proofs $\vdash \psi$:

- If φ is SAT it is also finitely satisfiable, hence some model $\boldsymbol{A}$ will show up in the first list; answer YES.
- If φ is UNSAT then $\neg\varphi$ will show up in the second list; answer NO.

The Finite Model Property Implies Decidability

Theorem

If L has the small model property then L is decidable.

To check $SAT(\varphi)$ enumerate all structures up to size $f(|\varphi|)$;
if any is a model return YES, if none is a model return NO.

Theorem

If L has the finite model property then L is decidable.

To check $SAT(\varphi)$ enumerate all finite structures $\boldsymbol{A}$ AND all proofs $\vdash \psi$:

- If φ is SAT it is also finitely satisfiable, hence some model $\boldsymbol{A}$ will show up in the first list; answer YES.
- If φ is UNSAT then $\neg\varphi$ will show up in the second list; answer NO.

The Finite Model Property Implies Decidability

Theorem

If L has the small model property then L is decidable.

To check $SAT(\varphi)$ enumerate all structures up to size $f(|\varphi|)$;
if any is a model return YES, if none is a model return NO.

Theorem

If L has the finite model property then L is decidable.

To check $SAT(\varphi)$ enumerate all finite structures $\boldsymbol{A}$ AND all proofs $\vdash \psi$:

- If φ is SAT it is also finitely satisfiable, hence some model $\boldsymbol{A}$ will show up in the first list; answer YES.
- If φ is UNSAT then $\neg\varphi$ will show up in the second list; answer NO.

Application 1: Bernays-Schönfinkel

Let L be the set of sentences with quantifier prefix $\exists^*\forall^*$.
L is called the Bernays-Schönfinkel class.

Theorem

The set of $\exists^*\forall^*$ sentences has the small model property, hence it is decidable.

Proof in class

$\varphi = \exists x_1 \cdots \exists x_m \forall y_1 \cdots \forall y_n \psi$.

Let $\boldsymbol{A}$ be a model of φ. Then there exists values $\boldsymbol{a} = (a_1, \ldots, a_m)$ s.t. $\boldsymbol{A} \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

Let $\boldsymbol{A}_0$ be the structure restricted to the value $a_1, \ldots, a_m$. Then, obviously: $\boldsymbol{A}_0 \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

what is the "small model" function $f(|\varphi|)$?

Application 1: Bernays-Schönfinkel

Let L be the set of sentences with quantifier prefix $\exists^*\forall^*$.
L is called the Bernays-Schönfinkel class.

Theorem

The set of $\exists^*\forall^*$ sentences has the small model property, hence it is decidable.

Proof in class

$\varphi = \exists x_1 \cdots \exists x_m \forall y_1 \cdots \forall y_n \psi$.

Let $\boldsymbol{A}$ be a model of φ. Then there exists values $\boldsymbol{a} = (a_1, \ldots, a_m)$ s.t. $\boldsymbol{A} \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

Let $\boldsymbol{A}_0$ be the structure restricted to the value $a_1, \ldots, a_m$. Then, obviously: $\boldsymbol{A}_0 \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

what is the "small model" function $f(|\varphi|)$?

Application 1: Bernays-Schönfinkel

Let L be the set of sentences with quantifier prefix $\exists^*\forall^*$.
L is called the Bernays-Schönfinkel class.

Theorem

The set of $\exists^*\forall^*$ sentences has the small model property, hence it is decidable.

Proof in class
$\varphi = \exists x_1 \cdots \exists x_m \forall y_1 \cdots \forall y_n \psi$.

Let $\boldsymbol{A}$ be a model of φ. Then there exists values $\boldsymbol{a} = (a_1, \ldots, a_m)$ s.t. $\boldsymbol{A} \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

Let $\boldsymbol{A}_0$ be the structure restricted to the value $a_1, \ldots, a_m$. Then, obviously: $\boldsymbol{A}_0 \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

what is the "small model" function $f(|\varphi|)$?

Application 1: Bernays-Schönfinkel

Let L be the set of sentences with quantifier prefix $\exists^*\forall^*$.
L is called the Bernays-Schönfinkel class.

Theorem

The set of $\exists^*\forall^*$ sentences has the small model property, hence it is decidable.

Proof in class

$\varphi = \exists x_1 \cdots \exists x_m \forall y_1 \cdots \forall y_n \psi$.

Let $\boldsymbol{A}$ be a model of φ. Then there exists values $\boldsymbol{a} = (a_1, \ldots, a_m)$ s.t.
$\boldsymbol{A} \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

Let $\boldsymbol{A}_0$ be the structure restricted to the value $a_1, \ldots, a_m$. Then, obviously:
$\boldsymbol{A}_0 \models \forall y_1 \cdots \forall y_n \psi[\boldsymbol{a}/\boldsymbol{x}]$

what is the "small model" function $f(|\varphi|)$?

Application 2: FO^2

Theorem

FO^2 has the small model property, with an exponential f. More precisely: for any sentence in $\varphi \in FO^2$, if φ is satisfiable then it has a model of size $2^{O(|\varphi|)}$. In particular, FO^2 is decidable.

We omit the proof. Please check Grädel, Kolaitis, Vardi.

Descriptive Complexity

Main topic: correspondence between logics and computational complexity classes.

Descriptive Complexity

Fix a class $\mathcal{C}$ of finite structures.

- Examples: (1) all strings $w \in \{0,1\}^*$; (2) all graphs (V,E); (3) all ordered graphs $(V,E,<)$; (4) all strings representing FO^2 sentences, $\varphi \in \{x,y,R,(,),\rightarrow,\neg,\forall\}^*$.

A *problem* is a function $P : \mathcal{C} \rightarrow \{0,1\}$.

- A *computational complexity class* is the set of problems that can be answered within some fixed computational resources. E.g. LOGSPACE, PTIME, PSPACE, etc.
- A *descriptive complexity class* is the set of problems that can be represented in some fixed logic language L. E.g. FO, FO+Fixpoint, $\exists$SO, SO, etc.

Descriptive Complexity

Fix a class $\mathcal{C}$ of finite structures.

- Examples: (1) all strings $w \in \{0,1\}^*$; (2) all graphs (V, E); (3) all ordered graphs $(V, E, <)$; (4) all strings representing FO^2 sentences, $\varphi \in \{x, y, R, (,), \to, \neg, \forall\}^*$.

A *problem* is a function $P : \mathcal{C} \to \{0,1\}$.

- A *computational complexity class* is the set of problems that can be answered within some fixed computational resources. E.g. LOGSPACE, PTIME, PSPACE, etc.
- A *descriptive complexity class* is the set of problems that can be represented in some fixed logic language L. E.g. FO, FO+Fixpoint, $\exists$SO, SO, etc.

Descriptive Complexity

Fix a class $\mathcal{C}$ of finite structures.

- Examples: (1) all strings $w \in \{0,1\}^*$; (2) all graphs (V, E); (3) all ordered graphs $(V, E, <)$; (4) all strings representing FO^2 sentences, $\varphi \in \{x, y, R, (,), \rightarrow, \neg, \forall\}^*$.

A *problem* is a function $P : \mathcal{C} \rightarrow \{0,1\}$.

- A *computational complexity class* is the set of problems that can be answered within some fixed computational resources. E.g. LOGSPACE, PTIME, PSPACE, etc.
- A *descriptive complexity class* is the set of problems that can be represented in some fixed logic language L. E.g. FO, FO+Fixpoint, $\exists$SO, SO, etc.

Descriptive Complexity

Fix a class $\mathcal{C}$ of finite structures.

- Examples: (1) all strings $w \in \{0,1\}^*$; (2) all graphs (V, E); (3) all ordered graphs $(V, E, <)$; (4) all strings representing FO^2 sentences, $\varphi \in \{x, y, R, (,), \to, \neg, \forall\}^*$.

A *problem* is a function $P : \mathcal{C} \to \{0,1\}$.

- A *computational complexity class* is the set of problems that can be answered within some fixed computational resources. E.g. LOGSPACE, PTIME, PSPACE, etc.

- A *descriptive complexity class* is the set of problems that can be represented in some fixed logic language L. E.g. FO, FO+Fixpoint, $\exists$SO, SO, etc.

Computational Complexity

Very brief review of computational complexity classes:

- AC^0
- LOGSPACE
- NLOGSPACE
- PTIME
- NP
- PSPACE
- (what about NPSACE?)
- EXPTIME
- NEXPTIME

Computational Complexity of Model Checking

The model checking problem is: given $\boldsymbol{A} \in \mathcal{C}, \varphi \in L$, check whether $\boldsymbol{A} \vDash \varphi$.

Vardi's classification of complexity:

Data complexity: φ is fixed, study the complexity as a function of $\boldsymbol{A}$.

Note: different complexity for every φ.

We focus on data complexity

Query complexity: (or expression complexity):

$\boldsymbol{A}$ is fixed, study the complexity as a function of φ.

Combined complexity: both $\boldsymbol{A}, \varphi$ are input.

Computational Complexity of Model Checking

The model checking problem is: given $\boldsymbol{A} \in \mathcal{C}, \varphi \in L$, check whether $\boldsymbol{A} \vDash \varphi$.

Vardi's classification of complexity:

Data complexity: φ is fixed, study the complexity as a function of $\boldsymbol{A}$.
Note: different complexity for every φ.
We focus on data complexity

Query complexity: (or expression complexity):
$\boldsymbol{A}$ is fixed, study the complexity as a function of φ.

Combined complexity: both $\boldsymbol{A}, \varphi$ are input.

Computational Complexity of Model Checking

The model checking problem is: given $\boldsymbol{A} \in \mathcal{C}, \varphi \in L$, check whether $\boldsymbol{A} \vDash \varphi$.

Vardi's classification of complexity:

Data complexity: φ is fixed, study the complexity as a function of $\boldsymbol{A}$.
Note: different complexity for every φ.
We focus on data complexity

Query complexity: (or expression complexity):
$\boldsymbol{A}$ is fixed, study the complexity as a function of φ.

Combined complexity: both $\boldsymbol{A}, \varphi$ are input.

Computational Complexity of Model Checking

The model checking problem is: given $\boldsymbol{A} \in \mathcal{C}, \varphi \in L$, check whether $\boldsymbol{A} \models \varphi$.

Vardi's classification of complexity:

Data complexity: φ is fixed, study the complexity as a function of $\boldsymbol{A}$.
Note: different complexity for every φ.
We focus on data complexity

Query complexity: (or expression complexity):
$\boldsymbol{A}$ is fixed, study the complexity as a function of φ.

Combined complexity: both $\boldsymbol{A}, \varphi$ are input.

Descriptive Complexity: Overview of Results

- $\text{FO}(+,*) = \text{FO}(<,\text{BIT}) = AC^0$
- $\text{FO}(\text{det-TC},<)$ =LOGSPACE, and $\text{FO}(\text{TC},<)$ =NLOGSPACE; will omit this
- $\text{FO}(\text{LeastFixpoint},<)$ =$\text{FO}(\text{InflationaryFixpoint},<)$ =PTIME
- $\text{FO}(\text{PartialFixpoint},<)$ =PSPACE
- $\exists$SO=NP

All these refer to data complexity. We will briefly discuss expression complexity at the end.

Encodings

- A Turning Machine (or other computational device), accepts a language $L \subseteq \{0,1\}^*$.
- A sentence φ defines a set of models $\subseteq \mathrm{STRUCT}[\sigma]$.
- To compare them, we need some encoding between them.

Encoding STRUCT$[\sigma]$ to $\{0,1\}^*$

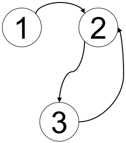
Encode $\boldsymbol{A} = ([n], R_1^A, R_2^A, \ldots)$ as follows:

- Start with 01^n.
- Encode R_i^A using "adjacency matrix", of length $n^{\text{arity}(R_i)}$

Encoding STRUCT[σ] to $\{0,1\}^*$

Encode $\boldsymbol{A} = ([n], R_1^A, R_2^A, \ldots)$ as follows:

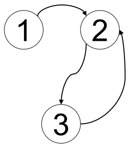
- Start with 01^n.
- Encode R_i^A using "adjacency matrix", of length $n^{\text{arity}(R_i)}$
- Example:



Encoding STRUCT[σ] to $\{0,1\}^*$

Encode $\boldsymbol{A} = ([n], R_1^A, R_2^A, \ldots)$ as follows:

- Start with 01^n.
- Encode R_i^A using "adjacency matrix", of length $n^{\text{arity}(R_i)}$
- Example:

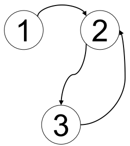


$$\underbrace{0111}_{n=3}\,\underbrace{010001010}_{3\times 3 \text{ matrix}}$$

Encoding STRUCT$[\sigma]$ to $\{0,1\}^*$

Encode $\boldsymbol{A} = ([n], R_1^A, R_2^A, \ldots)$ as follows:

- Start with 01^n.
- Encode R_i^A using "adjacency matrix", of length $n^{\text{arity}(R_i)}$
- Example:



$$\underbrace{0111}_{n=3}\underbrace{010001010}_{3\times 3 \text{ matrix}}$$

- Length of encoding: $n^{1+\text{arity}(R_1)+\text{arity}(R_2)+\cdots} = n^{O(1)} = \text{poly}(n)$.

Encoding $\{0, 1\}^*$ to STRUCT$[\sigma]$

Choose $\sigma = \{U(\cdot)\}$ and encode $w \in \{0, 1\}^*$ as the structure $([n], U)$, where $U \subseteq [n]$.

Descriptive Complexity: Overview of Results

- $\mathrm{FO}(+, *) = \mathrm{FO}(<, \mathrm{BIT}) = AC^0$
- $\mathrm{FO}(\text{det-TC}, <) =$LOGSPACE, and $\mathrm{FO}(\mathrm{TC}, <) =$NLOGSPACE;
- $\mathrm{FO}(\mathrm{LeastFixpoint}, <) =\mathrm{FO}(\mathrm{InflationaryFixpoint}, <) =$PTIME
- $\mathrm{FO}(\mathrm{PartialFixpoint}, <) =$PSPACE
- $\exists$SO=NP

Non-uniform AC^0

Fix $n > 0$. A Boolean circuit C with n inputs is a DAG where:

- Leaves are labeled with input variables $X_1, \ldots, X_n \in \{0, 1\}$.
- Internal nodes are labeled with $\vee, \wedge$ (unbounded fan-in), and $\neg$.
- There is one root node.

$$\text{size}(C) \stackrel{\text{def}}{=} \text{number of gates}$$

$$\text{depth}(C) \stackrel{\text{def}}{=} \text{length of longest path}$$

Non-uniform AC^0

Definition

A language $L \subseteq \{0,1\}^*$ is in non-uniform AC^0 if forall n there exists a circuit C_n s.t.

- C_n computes $L \cap \{0,1\}^n$,
- $\text{size}(C_n) = n^{O(1)}$ (polynomial in n),
- $\text{depth}(C_n) = O(1)$ (constant, indep. on n).

Example: given a graph $G = ([n], E)$, check $\forall x \forall y \exists z (E(x,z) \wedge E(z,y))$
draw C_n (actually C_{n^2}) in class.

Theorem

The data complexity of any $\varphi \in FO$ is in non-uniform AC^0.
This still holds if we include in FO all *interpreted predicate $(+, <, \ldots)$.*
Thus $FO(\mathtt{ALL}) \subseteq AC^0$, where $\mathtt{ALL}$ means all predicates on $[n]$.

Non-uniform AC^0

Definition

A language $L \subseteq \{0,1\}^*$ is in non-uniform AC^0 if forall n there exists a circuit C_n s.t.

- C_n computes $L \cap \{0,1\}^n$,
- $\text{size}(C_n) = n^{O(1)}$ (polynomial in n),
- $\text{depth}(C_n) = O(1)$ (constant, indep. on n).

Example: given a graph $G = ([n], E)$, check $\forall x \forall y \exists z (E(x,z) \wedge E(z,y))$
draw C_n (actually C_{n^2}) in class.

Theorem

The data complexity of any $\varphi \in FO$ is in non-uniform AC^0.
This still holds if we include in FO all *interpreted predicate $(+, <, \ldots)$.*
Thus $FO(\mathcal{ALL}) \subseteq AC^0$, where $\mathcal{ALL}$ means all predicates on $[n]$.

Non-uniform AC^0

Definition

A language $L \subseteq \{0,1\}^*$ is in non-uniform AC^0 if forall n there exists a circuit C_n s.t.

- C_n computes $L \cap \{0,1\}^n$,
- $\text{size}(C_n) = n^{O(1)}$ (polynomial in n),
- $\text{depth}(C_n) = O(1)$ (constant, indep. on n).

Example: given a graph $G = ([n], E)$, check $\forall x \forall y \exists z (E(x,z) \wedge E(z,y))$
draw C_n (actually C_{n^2}) in class.

Theorem

The data complexity of any $\varphi \in FO$ is in non-uniform AC^0.
This still holds if we include in FO all *interpreted predicate $(+, <, \ldots)$.*
Thus $FO(\mathtt{ALL}) \subseteq AC^0$, where $\mathtt{ALL}$ means all predicates on $[n]$.

Discussion

- AC^0 is supposed to be the lowest complexity class, but there's a wrinkle:
- If L is in non-uniform AC^0, is L computable?

Discussion

- AC^0 is supposed to be the lowest complexity class, but there's a wrinkle:
- If L is in non-uniform AC^0, is L computable? NO! E.g. L is the set of all words of length n, where n encodes a Turning Machine that halts on the empty input. Describe C_n in class

Discussion

- AC^0 is supposed to be the lowest complexity class, but there's a wrinkle:
- If L is in non-uniform AC^0, is L computable? NO! E.g. L is the set of all words of length n, where n encodes a Turning Machine that halts on the empty input. Describe C_n in class
- Recall: EVEN is the problem "is the domain size n an even number?". Obviously EVEN $\in FO[\texttt{ALL}]$ why?

Discussion

- AC^0 is supposed to be the lowest complexity class, but there's a wrinkle:
- If L is in non-uniform AC^0, is L computable? NO! E.g. L is the set of all words of length n, where n encodes a Turning Machine that halts on the empty input. Describe C_n in class
- Recall: EVEN is the problem "is the domain size n an even number?". Obviously EVEN $\in FO[\texttt{ALL}]$ why?
- Theorem [Furst-Saxe-Sipser, Ajtai] The xor-function $X_1 \oplus X_2 \oplus \cdots \oplus X_n$ is not in non-uniform AC^0 discuss in class
- PARITY is the problem: given a structure with one unary relation, $([n], U \subseteq [n])$, check whether $|U|$ is even. Then PARITY $\notin FO[\texttt{ALL}]$.

Uniform AC^0

Informally: L is in "uniform" AC^0 if there exists an easily computable function $n \mapsto C_n$ (usually LOGSPACE).

A better definition uses FO. For fixed n, define these relations on $[n]$:

$$+ = \{(x, y, z) \mid x + y = z\}$$
$$* = \{(x, y, z) \mid x * y = z\}$$
$$< = \{(x, y) \mid x < y\}$$
$$\text{BIT} = \{(x, y) \mid \text{the } y\text{'s bit of } x \text{ is } 1\}$$

One can show $FO(+, *) = FO(<, \text{BIT})$ (we omit the proof).

Definition

A language $L \subseteq \{0, 1\}^*$ is in uniform AC^0 if it is definable in $FO(+, *)$; equivalently, it is definable in $FO(<, \text{BIT})$.

Uniform AC^0

Informally: L is in "uniform" AC^0 if there exists an easily computable function $n \mapsto C_n$ (usually LOGSPACE).
A better definition uses FO. For fixed n, define these relations on $[n]$:

$$\begin{aligned} + =& \{(x, y, z) \mid x + y = z\} \\ * =& \{(x, y, z) \mid x * y = z\} \\ < =& \{(x, y) \mid x < y\} \\ \text{BIT} =& \{(x, y) \mid \text{the } y\text{'s bit of } x \text{ is } 1\} \end{aligned}$$

One can show $FO(+, *) = FO(<, \text{BIT})$ (we omit the proof).

Definition

A language $L \subseteq \{0, 1\}^*$ is in uniform AC^0 if it is definable in $FO(+, *)$; equivalently, it is definable in $FO(<, \text{BIT})$.

Uniform AC^0

Informally: L is in "uniform" AC^0 if there exists an easily computable function $n \mapsto C_n$ (usually LOGSPACE).
A better definition uses FO. For fixed n, define these relations on $[n]$:

$$\begin{aligned}
+ =& \{(x,y,z) \mid x+y=z\} \\
* =& \{(x,y,z) \mid x*y=z\} \\
<=& \{(x,y) \mid x<y\} \\
\text{BIT} =& \{(x,y) \mid \text{the } y\text{'s bit of } x \text{ is } 1\}
\end{aligned}$$

One can show $FO(+,*) = FO(<,\text{BIT})$ (we omit the proof).

Definition

A language $L \subseteq \{0,1\}^*$ is in uniform AC^0 if it is definable in $FO(+,*)$; equivalently, it is definable in $FO(<,\text{BIT})$.

Discussion

- Main take away: AC^0 is FO.
- The reason is simple: $\vee, \wedge$ have bounded fan-in, $\exists, \forall$ have unbounded fan-in, and the depth is constant.
- But there is a fine print in the equality $AC^0 = FO$:
  - Non-uniform AC^0 can express any predicate on $[n]$, much beyond FO.
  - We define Uniform AC^0 as $FO(+, *)$ or as $FO(<, \text{BIT})$; the choice to restrict to the predicates $+, *$ (or $<, \text{BIT}$) is somewhat arbitrary, yet leads to a natural definition of Uniform AC^0.

Descriptive Complexity: Overview of Results

- FO$(+, *)$ = FO$(<, \text{BIT})$ = AC^0
- FO(det-TC, $<$) =LOGSPACE, and FO(TC, $<$) =NLOGSPACE;
- FO(LeastFixpoint, $<$) =FO(InflationaryFixpoint, $<$) =PTIME
- FO(PartialFixpoint, $<$) =PSPACE
- $\exists$SO=NP

$\exists SO$ and NP

$\exists SO$ consists of sentences $\exists S_1 \cdots \exists S_m \varphi$, where $\varphi \in$FO over vocabulary $\sigma \cup \{S_1, \ldots, S_m\}$.

Theorem (Fagin)

$$\exists SO = NP.$$

In words:

- $\exists SO \subseteq NP$: the data complexity of $\psi \in \exists SO$ is in NP proof in class
- $NP \subseteq \exists SO$: for any problem in NP, there exists a sentence $\psi \in \exists SO$ that expresses precisely that problem; will prove next.

In class Give an $\exists SO$ formula to check if a graph has a Hamiltonean cycle.

$\exists SO$ and NP

$\exists SO$ consists of sentences $\exists S_1 \cdots \exists S_m \varphi$, where $\varphi \in$FO over vocabulary $\sigma \cup \{S_1, \ldots, S_m\}$.

Theorem (Fagin)

$$\exists SO = NP.$$

In words:

- $\exists SO \subseteq NP$: the data complexity of $\psi \in \exists SO$ is in NP proof in class
- $NP \subseteq \exists SO$: for any problem in NP, there exists a sentence $\psi \in \exists SO$ that expresses precisely that problem; will prove next.

In class Give an $\exists SO$ formula to check if a graph has a Hamiltonean cycle.

$\exists SO$ and NP

$\exists SO$ consists of sentences $\exists S_1 \cdots \exists S_m \varphi$, where $\varphi \in$FO over vocabulary $\sigma \cup \{S_1, \ldots, S_m\}$.

Theorem (Fagin)

$$\exists SO = NP.$$

In words:

- $\exists SO \subseteq NP$: the data complexity of $\psi \in \exists SO$ is in NP proof in class
- $NP \subseteq \exists SO$: for any problem in NP, there exists a sentence $\psi \in \exists SO$ that expresses precisely that problem; will prove next.

In class Give an $\exists SO$ formula to check if a graph has a Hamiltonean cycle.

$\exists SO$ and NP

$\exists SO$ consists of sentences $\exists S_1 \cdots \exists S_m \varphi$, where $\varphi \in$FO over vocabulary $\sigma \cup \{S_1, \ldots, S_m\}$.

Theorem (Fagin)

$$\exists SO = NP.$$

In words:

- $\exists SO \subseteq NP$: the data complexity of $\psi \in \exists SO$ is in NP proof in class
- $NP \subseteq \exists SO$: for any problem in NP, there exists a sentence $\psi \in \exists SO$ that expresses precisely that problem; will prove next.

In class Give an $\exists SO$ formula to check if a graph has a Hamiltonean cycle.

$\exists SO$ and NP

$\exists SO$ consists of sentences $\exists S_1 \cdots \exists S_m \varphi$, where $\varphi \in$FO over vocabulary $\sigma \cup \{S_1, \ldots, S_m\}$.

Theorem (Fagin)

$$\exists SO = NP.$$

In words:

- $\exists SO \subseteq NP$: the data complexity of $\psi \in \exists SO$ is in NP proof in class
- $NP \subseteq \exists SO$: for any problem in NP, there exists a sentence $\psi \in \exists SO$ that expresses precisely that problem; will prove next.

In class Give an $\exists SO$ formula to check if a graph has a Hamiltonean cycle.

Proof of $NP \subseteq \exists SO$

Let $L \subseteq \{0,1\}^*$ be a language in NP. This means:
$\exists$ Turing Machine M and $d > 0$ s.t. for any input $w \subseteq \{0,1\}^n$:
M has an accepting computation of length $\leq n^d$ iff $w \in L$.

Define ψ_M s.t. $([n], U) \vDash \psi_M$ iff[1] $U \in L$, as in Trakhtenbrot's theorem:

$$\psi_M = \exists < \exists T_0(\cdot,\cdot) \exists T_1(\cdot,\cdot) \exists H(\cdot,\cdot) \exists S_{q_0}(\cdot) \exists S_{q_1}(\cdot) \cdots \varphi_M$$

where φ_M is as in Trakthenbrot's proof, with two changes:

- Assert that the initial configuration is the string U (i.e. not 0's)
- The time/space can now go up to n^d: encode it using a d-tuple instead of a single value in class;
  T_0, T_1, H now have arity $2d$ and $S_{q_0}, S_{q_1}, \ldots$ have arity d

[1] $U \subseteq [n]$ denotes a string $w \in \{0,1\}^n$ how?.

Proof of $NP \subseteq \exists SO$

Let $L \subseteq \{0,1\}^*$ be a language in NP. This means:
$\exists$ Turing Machine M and $d > 0$ s.t. for any input $w \subseteq \{0,1\}^n$:
M has an accepting computation of length $\leq n^d$ iff $w \in L$.

Define ψ_M s.t. $([n], U) \vDash \psi_M$ iff[1] $U \in L$, as in Trakhtenbrot's theorem:

$$\psi_M = \exists < \exists T_0(\cdot,\cdot) \exists T_1(\cdot,\cdot) \exists H(\cdot,\cdot) \exists S_{q_0}(\cdot) \exists S_{q_1}(\cdot) \cdots \varphi_M$$

where φ_M is as in Trakthenbrot's proof, with two changes:

- Assert that the initial configuration is the string U (i.e. not 0's)
- The time/space can now go up to n^d: encode it using a d-tuple instead of a single value in class;
  T_0, T_1, H now have arity $2d$ and $S_{q_0}, S_{q_1}, \ldots$ have arity d

[1] $U \subseteq [n]$ denotes a string $w \in \{0,1\}^n$ how?.

Proof of $NP \subseteq \exists SO$

Let $L \subseteq \{0,1\}^*$ be a language in NP. This means:
$\exists$ Turing Machine M and $d > 0$ s.t. for any input $w \subseteq \{0,1\}^n$:
M has an accepting computation of length $\leq n^d$ iff $w \in L$.

Define ψ_M s.t. $([n], U) \vDash \psi_M$ iff[1] $U \in L$, as in Trakhtenbrot's theorem:

$$\psi_M = \exists < \exists T_0(\cdot,\cdot)\exists T_1(\cdot,\cdot)\exists H(\cdot,\cdot)\exists S_{q_0}(\cdot)\exists S_{q_1}(\cdot)\cdots\varphi_M$$

where φ_M is as in Trakthenbrot's proof, with two changes:

- Assert that the initial configuration is the string U (i.e. not 0's)
- The time/space can now go up to n^d: encode it using a d-tuple instead of a single value in class;
  T_0, T_1, H now have arity $2d$ and $S_{q_0}, S_{q_1}, \ldots$ have arity d

[1] $U \subseteq [n]$ denotes a string $w \in \{0,1\}^n$ how?.

Proof of $NP \subseteq \exists SO$

Let $L \subseteq \{0,1\}^*$ be a language in NP. This means:
$\exists$ Turing Machine M and $d > 0$ s.t. for any input $w \subseteq \{0,1\}^n$:
M has an accepting computation of length $\leq n^d$ iff $w \in L$.

Define ψ_M s.t. $([n], U) \vDash \psi_M$ iff[1] $U \in L$, as in Trakhtenbrot's theorem:

$$\psi_M = \exists < \exists T_0(\cdot,\cdot) \exists T_1(\cdot,\cdot) \exists H(\cdot,\cdot) \exists S_{q_0}(\cdot) \exists S_{q_1}(\cdot) \cdots \varphi_M$$

where φ_M is as in Trakthenbrot's proof, with two changes:

- Assert that the initial configuration is the string U (i.e. not 0's)
- The time/space can now go up to n^d: encode it using a d-tuple instead of a single value in class;
  T_0, T_1, H now have arity $2d$ and $S_{q_0}, S_{q_1}, \ldots$ have arity d

[1] $U \subseteq [n]$ denotes a string $w \in \{0,1\}^n$ how?.

Discussion

- $\exists SO = NP$ is a very elegant result!
- Main lesson: $\exists SO$ is very expressive.
- This suggests a restriction to monadic existential SO. $\exists MSO$ is also called monadic NP.
- Is there a very easy query that is not expressible in $\exists MSO$?

Discussion

- $\exists SO = NP$ is a very elegant result!
- Main lesson: $\exists SO$ is very expressive.
- This suggests a restriction to monadic existential SO. $\exists MSO$ is also called monadic NP.
- Is there a very easy query that is not expressible in $\exists MSO$? Connectivity!

Detour: Spectra and Counting

The spectrum of a sentence φ is the set of numbers n s.t. φ has a model of size n.

Examples:

- Let $\sigma = \{E\}$ and let φ says "E is a matching of the domain". What is $\texttt{Spec}(\varphi)$?

Detour: Spectra and Counting

The spectrum of a sentence φ is the set of numbers n s.t. φ has a model of size n.

Examples:

- Let $\sigma = \{E\}$ and let φ says "E is a matching of the domain". What is $\texttt{Spec}(\varphi)$? $\{2n \mid n \in \mathbb{N}\}$.

Detour: Spectra and Counting

The spectrum of a sentence φ is the set of numbers n s.t. φ has a model of size n.

Examples:

- Let $\sigma = \{E\}$ and let φ says "E is a matching of the domain". What is $\texttt{Spec}(\varphi)$? $\{2n \mid n \in \mathbb{N}\}$.
- Let $\sigma = (+, *, 0, 1)$ and φ be the axioms of a field. What is $\texttt{Spec}(\varphi)$?

Detour: Spectra and Counting

The spectrum of a sentence φ is the set of numbers n s.t. φ has a model of size n.

Examples:

- Let $\sigma = \{E\}$ and let φ says "E is a matching of the domain". What is $\texttt{Spec}(\varphi)$? $\{2n \mid n \in \mathbb{N}\}$.
- Let $\sigma = (+, *, 0, 1)$ and φ be the axioms of a field. What is $\texttt{Spec}(\varphi)$? $\{p^c \mid p \text{ prime}, c \geq 1\}$.

Detour: Spectra and Counting

The spectrum of a sentence φ is the set of numbers n s.t. φ has a model of size n.

Examples:

- Let $\sigma = \{E\}$ and let φ says "E is a matching of the domain". What is $\texttt{Spec}(\varphi)$? $\{2n \mid n \in \mathbb{N}\}$.
- Let $\sigma = (+, *, 0, 1)$ and φ be the axioms of a field. What is $\texttt{Spec}(\varphi)$? $\{p^c \mid p \text{ prime}, c \geq 1\}$.

We study the decision problem: "given n, check if $n \in \texttt{Spec}(\varphi)$".

Detour: Spectra and Counting

NETIME = problems solvable in time $\bigcup_{c\geq 0} 2^{cn}$
(don't confuse with NEXPTIME = problems solvable in time $\bigcup_{c\geq 0} 2^{n^c}$)

Theorem (Jones&Selman'1972)

If the input n is given in binary: $\{\mathcal{S}pec(\varphi) \mid \varphi \in FO\} = NETIME$

Theorem (Special case of Fagin's theorem, for $\sigma = \emptyset$ why?)

If the input n is given in unary: $\{\mathcal{S}pec(\varphi) \mid \varphi \in FO\} = NP_1$

The counting problem is: given n, count the number of models $\#_n(\varphi)$

Theorem

If the input n is given in unary: $\{n \mapsto \#_n(\varphi) \mid \varphi \in FO\} = \#P_1$
In particular, there exists a sentence φ s.t. $\#_n(\varphi)$ is $\#P_1$-complete.

This explains why it is hard to compute $\mu_n(\varphi)$ exactly!
Notice: no "natural" hard problem is known for $\#P_1$.

Detour: Spectra and Counting

NETIME = problems solvable in time $\bigcup_{c\geq 0} 2^{cn}$
(don't confuse with NEXPTIME = problems solvable in time $\bigcup_{c\geq 0} 2^{n^c}$)

Theorem (Jones&Selman'1972)

If the input n is given in binary: $\{\mathit{Spec}(\varphi) \mid \varphi \in FO\} = NETIME$

Theorem (Special case of Fagin's theorem, for $\sigma = \emptyset$ why?)

If the input n is given in unary: $\{\mathit{Spec}(\varphi) \mid \varphi \in FO\} = NP_1$

The counting problem is: given n, count the number of models $\#_n(\varphi)$

Theorem

If the input n is given in unary: $\{n \mapsto \#_n(\varphi) \mid \varphi \in FO\} = \#P_1$
In particular, there exists a sentence φ *s.t.* $\#_n(\varphi)$ *is* $\#P_1$*-complete.*

This explains why it is hard to compute $\mu_n(\varphi)$ exactly!
Notice: no "natural" hard problem is known for $\#P_1$.

Detour: Spectra and Counting

NETIME = problems solvable in time $\bigcup_{c\geq 0} 2^{cn}$
(don't confuse with NEXPTIME = problems solvable in time $\bigcup_{c\geq 0} 2^{n^c}$)

Theorem (Jones&Selman'1972)

If the input n is given in binary: $\{Spec(\varphi) \mid \varphi \in FO\} = NETIME$

Theorem (Special case of Fagin's theorem, for $\sigma = \emptyset$ why?)

If the input n is given in unary: $\{Spec(\varphi) \mid \varphi \in FO\} = NP_1$

The counting problem is: given n, count the number of models $\#_n(\varphi)$

Theorem

If the input n is given in unary: $\{n \mapsto \#_n(\varphi) \mid \varphi \in FO\} = \#P_1$
In particular, there exists a sentence φ s.t. $\#_n(\varphi)$ is $\#P_1$-complete.

This explains why it is hard to compute $\mu_n(\varphi)$ exactly!
Notice: no "natural" hard problem is known for $\#P_1$.

Detour: Spectra and Counting

NETIME = problems solvable in time $\bigcup_{c\geq 0} 2^{cn}$
(don't confuse with NEXPTIME = problems solvable in time $\bigcup_{c\geq 0} 2^{n^c}$)

Theorem (Jones&Selman'1972)

If the input n is given in binary: $\{\mathit{Spec}(\varphi) \mid \varphi \in FO\} = NETIME$

Theorem (Special case of Fagin's theorem, for $\sigma = \emptyset$ why?)

If the input n is given in unary: $\{\mathit{Spec}(\varphi) \mid \varphi \in FO\} = NP_1$

The counting problem is: given n, count the number of models $\#_n(\varphi)$

Theorem

If the input n is given in unary: $\{n \mapsto \#_n(\varphi) \mid \varphi \in FO\} = \#P_1$
In particular, there exists a sentence φ s.t. $\#_n(\varphi)$ is $\#P_1$-complete.

This explains why it is hard to compute $\mu_n(\varphi)$ exactly!
Notice: no "natural" hard problem is known for $\#P_1$.

Detour: Spectra and Counting

NETIME = problems solvable in time $\bigcup_{c\geq 0} 2^{cn}$
(don't confuse with NEXPTIME = problems solvable in time $\bigcup_{c\geq 0} 2^{n^c}$)

Theorem (Jones&Selman'1972)

If the input n is given in binary: $\{\mathit{Spec}(\varphi) \mid \varphi \in FO\} = NETIME$

Theorem (Special case of Fagin's theorem, for $\sigma = \emptyset$ why?)

If the input n is given in unary: $\{\mathit{Spec}(\varphi) \mid \varphi \in FO\} = NP_1$

The counting problem is: given n, count the number of models $\#_n(\varphi)$

Theorem

If the input n is given in unary: $\{n \mapsto \#_n(\varphi) \mid \varphi \in FO\} = \#P_1$
In particular, there exists a sentence φ s.t. $\#_n(\varphi)$ is $\#P_1$-complete.

This explains why it is hard to compute $\mu_n(\varphi)$ exactly!
Notice: no "natural" hard problem is known for $\#P_1$.

Descriptive Complexity: Overview of Results

- $FO(+, *) = FO(<, BIT) = AC^0$
- $FO(\text{det-TC}, <) =$LOGSPACE, and $FO(TC, <) =$NLOGSPACE;
- $FO(\text{LeastFixpoint}, <) = FO(\text{InflationaryFixpoint}, <) =$PTIME
- $FO(\text{PartialFixpoint}, <) =$PSPACE
- $\exists$SO=NP

Fixpoints

Let U be a finite set, and $f : 2^U \rightarrow 2^U$.

- A *fixpoint* is a set $X \subseteq U$ s.t. $f(X) = X$.
- A *least fixpoint* is a fixpoint X_0 s.t. for any fixpoint X, $X_0 \subseteq X$.
- When it exists, the least fixpoint is unique why?; denote it $\texttt{lfp}(f)$.

Fixpoints (cont'd)

Fix finite U, $f : 2^U \rightarrow 2^U$. Define $f^0 \stackrel{\text{def}}{=} \emptyset$, $f^{n+1} \stackrel{\text{def}}{=} f(f^n)$, $f^\infty \stackrel{\text{def}}{=} \bigcup_n f^n$.

Theorem (Tarski-Knaster)

If f is monotone ($X \subseteq Y \rightarrow f(X) \subseteq f(Y)$) then $\mathtt{lfp}(f) = f^\infty$.

Definition (Partial Fixpoint)

If $f^{n+1} = f^n$ for $n \geq 0$, then $\mathtt{pfp}(f) \stackrel{\text{def}}{=} f^n$ is called the *partial fixpoint* of f.

f is *inflationary* if $X \subseteq f(X)$; then f^∞ is a fixpoint why?

Definition (Inflationary Fixpoint)

The *inflationary fixpoint* of f is $\mathtt{ifp}(f) \stackrel{\text{def}}{=} g^\infty$, where $g(X) \stackrel{\text{def}}{=} X \cup f(X)$.

When f is monotone, $\mathtt{lfp}(f) = \mathtt{ifp}(f) = \mathtt{pfp}(f) = f^\infty$.

Fixpoints (cont'd)

Fix finite U, $f : 2^U \to 2^U$. Define $f^0 \stackrel{\text{def}}{=} \emptyset$, $f^{n+1} \stackrel{\text{def}}{=} f(f^n)$, $f^\infty \stackrel{\text{def}}{=} \bigcup_n f^n$.

Theorem (Tarski-Knaster)

If f is monotone ($X \subseteq Y \to f(X) \subseteq f(Y)$) then $\mathtt{lfp}(f) = f^\infty$.

Definition (Partial Fixpoint)

If $f^{n+1} = f^n$ for $n \geq 0$, then $\mathtt{pfp}(f) \stackrel{\text{def}}{=} f^n$ is called the *partial fixpoint* of f.

f is *inflationary* if $X \subseteq f(X)$; then f^∞ is a fixpoint why?

Definition (Inflationary Fixpoint)

The *inflationary fixpoint* of f is $\mathtt{ifp}(f) \stackrel{\text{def}}{=} g^\infty$, where $g(X) \stackrel{\text{def}}{=} X \cup f(X)$.

When f is monotone, $\mathtt{lfp}(f) = \mathtt{ifp}(f) = \mathtt{pfp}(f) = f^\infty$.

Fixpoints (cont'd)

Fix finite U, $f : 2^U \to 2^U$. Define $f^0 \stackrel{\text{def}}{=} \emptyset$, $f^{n+1} \stackrel{\text{def}}{=} f(f^n)$, $f^\infty \stackrel{\text{def}}{=} \bigcup_n f^n$.

Theorem (Tarski-Knaster)

If f is monotone ($X \subseteq Y \to f(X) \subseteq f(Y)$) then $\mathtt{lfp}(f) = f^\infty$.

Definition (Partial Fixpoint)

If $f^{n+1} = f^n$ for $n \geq 0$, then $\mathtt{pfp}(f) \stackrel{\text{def}}{=} f^n$ is called the *partial fixpoint* of f.

f is *inflationary* if $X \subseteq f(X)$; then f^∞ is a fixpoint why?

Definition (Inflationary Fixpoint)

The *inflationary fixpoint* of f is $\mathtt{ifp}(f) \stackrel{\text{def}}{=} g^\infty$, where $g(X) \stackrel{\text{def}}{=} X \cup f(X)$.

When f is monotone, $\mathtt{lfp}(f) = \mathtt{ifp}(f) = \mathtt{pfp}(f) = f^\infty$.

Fixpoints (cont'd)

Fix finite U, $f : 2^U \to 2^U$. Define $f^0 \stackrel{\text{def}}{=} \emptyset$, $f^{n+1} \stackrel{\text{def}}{=} f(f^n)$, $f^\infty \stackrel{\text{def}}{=} \bigcup_n f^n$.

Theorem (Tarski-Knaster)

If f is monotone ($X \subseteq Y \to f(X) \subseteq f(Y)$) then $\mathtt{lfp}(f) = f^\infty$.

Definition (Partial Fixpoint)

If $f^{n+1} = f^n$ for $n \geq 0$, then $\mathtt{pfp}(f) \stackrel{\text{def}}{=} f^n$ is called the *partial fixpoint* of f.

f is *inflationary* if $X \subseteq f(X)$; then f^∞ is a fixpoint why?

Definition (Inflationary Fixpoint)

The *inflationary fixpoint* of f is $\mathtt{ifp}(f) \stackrel{\text{def}}{=} g^\infty$, where $g(X) \stackrel{\text{def}}{=} X \cup f(X)$.

When f is monotone, $\mathtt{lfp}(f) = \mathtt{ifp}(f) = \mathtt{pfp}(f) = f^\infty$.

Fixpoints (cont'd)

Fix finite U, $f : 2^U \to 2^U$. Define $f^0 \stackrel{\text{def}}{=} \emptyset$, $f^{n+1} \stackrel{\text{def}}{=} f(f^n)$, $f^\infty \stackrel{\text{def}}{=} \bigcup_n f^n$.

Theorem (Tarski-Knaster)

If f is monotone ($X \subseteq Y \to f(X) \subseteq f(Y)$) then $\mathtt{lfp}(f) = f^\infty$.

Definition (Partial Fixpoint)

If $f^{n+1} = f^n$ for $n \geq 0$, then $\mathtt{pfp}(f) \stackrel{\text{def}}{=} f^n$ is called the *partial fixpoint* of f.

f is *inflationary* if $X \subseteq f(X)$; then f^∞ is a fixpoint why?

Definition (Inflationary Fixpoint)

The *inflationary fixpoint* of f is $\mathtt{ifp}(f) \stackrel{\text{def}}{=} g^\infty$, where $g(X) \stackrel{\text{def}}{=} X \cup f(X)$.

When f is monotone, $\mathtt{lfp}(f) = \mathtt{ifp}(f) = \mathtt{pfp}(f) = f^\infty$.

Fixpoints (cont'd)

Fix finite U, $f : 2^U \to 2^U$. Define $f^0 \stackrel{\text{def}}{=} \emptyset$, $f^{n+1} \stackrel{\text{def}}{=} f(f^n)$, $f^\infty \stackrel{\text{def}}{=} \bigcup_n f^n$.

Theorem (Tarski-Knaster)

If f is monotone ($X \subseteq Y \to f(X) \subseteq f(Y)$) then $\mathtt{lfp}(f) = f^\infty$.

Definition (Partial Fixpoint)

If $f^{n+1} = f^n$ for $n \geq 0$, then $\mathtt{pfp}(f) \stackrel{\text{def}}{=} f^n$ is called the *partial fixpoint* of f.

f is *inflationary* if $X \subseteq f(X)$; then f^∞ is a fixpoint why?

Definition (Inflationary Fixpoint)

The *inflationary fixpoint* of f is $\mathtt{ifp}(f) \stackrel{\text{def}}{=} g^\infty$, where $g(X) \stackrel{\text{def}}{=} X \cup f(X)$.

When f is monotone, $\mathtt{lfp}(f) = \mathtt{ifp}(f) = \mathtt{pfp}(f) = f^\infty$.

Fixpoint Logics

Let $R \notin \sigma$ be a new relational symbol. Define three new formulas:

$$[\mathtt{lfp}_{R,\boldsymbol{x}}\varphi(R, \boldsymbol{x})][\boldsymbol{t}]$$
$$[\mathtt{ifp}_{R,\boldsymbol{x}}\varphi(R, \boldsymbol{x})][\boldsymbol{t}]$$
$$[\mathtt{pfp}_{R,\boldsymbol{x}}\varphi(R, \boldsymbol{x})][\boldsymbol{t}]$$

where $|\boldsymbol{x}| = |\boldsymbol{t}| = \text{arity}(R)$; $\boldsymbol{x}$ are free in φ, and bound in $[\mathtt{lfp}_{R,\boldsymbol{x}}(\cdots)]$. Their meaning in a structure $\boldsymbol{A}$ is this. Define the function:

$$f(R) = \{\boldsymbol{a} \mid (\boldsymbol{A}, R) \vDash \varphi[\boldsymbol{a}/\boldsymbol{x}]\}$$

Then the formulas "mean" $\mathtt{lfp}(f), \mathtt{ifp}(f), \mathtt{pfp}(f)$ respectively[2].

Three new logics: $FO(\mathtt{lfp}), FO(\mathtt{ifp}), FO(\mathtt{pfp})$.

[2] For $\mathtt{lfp}$ we must ensure that φ is monotone. See homework.

Discussion

- This is horrible syntax. Here is how we check if a, b are connected in a graph $G = (V, E)$:

$$[\mathtt{lfp}_{T,x,y}(E(x,y) \vee \exists z(E(x,z) \wedge T(z,y)))](a,b)$$

  Now you really love datalog, were we write:

$$\begin{aligned} T(x,y) &\leftarrow E(x,y) \\ T(x,y) &\leftarrow E(x,z), T(z,y) \\ \text{Answer}() &\leftarrow T(a,b) \end{aligned}$$

- We made a few arbitrary choices: allow free variables? allow simultaneous recursion? It turns out these don't add expressive power, so use them if needed.
- Gurevitch and Shelah proved $FO(\mathtt{lfp}) = FO(\mathtt{ifp})$;
  We will only discuss `ifp` and `pfp`.

Detour: The Win-Move Game

The game is played by two players on a graph G. A pebble is placed initially on a node, then players take turn, and each player may move the pebble along an edge. The player who can't move loses. Write a query to compute the positions from which the *first* player has a winning strategy.

Detour: The Win-Move Game

The game is played by two players on a graph G. A pebble is placed initially on a node, then players take turn, and each player may move the pebble along an edge. The player who can't move loses. Write a query to compute the positions from which the *first* player has a winning strategy.

$$S(x) \leftarrow \exists y(E(x,y) \wedge \neg S(y)) \qquad \text{or } [\mathtt{pfp}_{S,x} \exists y(E(x,y) \wedge \neg S(y))](x)$$

Detour: The Win-Move Game

The game is played by two players on a graph G. A pebble is placed initially on a node, then players take turn, and each player may move the pebble along an edge. The player who can't move loses. Write a query to compute the positions from which the *first* player has a winning strategy.

$$S(x) \leftarrow \exists y(E(x,y) \wedge \neg S(y)) \qquad \text{or } [\mathtt{pfp}_{S,x} \exists y(E(x,y) \wedge \neg S(y))](x)$$

This is not monotone, hence may not have a fixpoint!

Detour: The Win-Move Game

The game is played by two players on a graph G. A pebble is placed initially on a node, then players take turn, and each player may move the pebble along an edge. The player who can't move loses. Write a query to compute the positions from which the *first* player has a winning strategy.

$$S(x) \leftarrow \exists y(E(x,y) \wedge \neg S(y)) \qquad \text{or } [\texttt{pfp}_{S,x} \exists y(E(x,y) \wedge \neg S(y))](x)$$

This is not monotone, hence may not have a fixpoint!
When it has a fixpoint, then it can obtain as:

$$S(x) \leftarrow \exists y(E(x,y) \wedge (\forall z E(y,z) \rightarrow S(z)))$$

Or:

$$[\texttt{lfp}_{S,x} \exists y(E(x,y) \wedge (\forall z E(y,z) \rightarrow S(z)))](x)$$

$FO[\mathtt{ifp}]$ captures PTIME

Theorem

(1) $FO[ifp] \subseteq PTIME$ and (2) $FO[ifp, <] = PTIME$.

Proof in class:

1. $FO[\mathtt{ifp}] \subseteq PTIME$ Show that the data complexity is PTIME.
2. $PTIME \subseteq FO[\mathtt{ifp}, <]$. Given a PTIME language $L \subseteq \{0,1\}^*$, write an $FO(\mathtt{ifp}, <)$-formula φ s.t. on any input structure $([n], U, <)$, φ is true iff $U \in L$. Note: we are given the order $<$ for free.

$FO[\texttt{pfp}]$ captures PSPACE

Theorem

(1) $FO[pfp] \subseteq PSPACE$ *and (2)* $FO[pfp, <] = PSPACE$*.*

Proof in class:

1. $FO[\texttt{pfp}] \subseteq PSPACE$. The hard part is negation: Immerman proved that $\neg\texttt{pfp}$ can be rewritten as some $\texttt{pfp}$, and this implied that $PSPACE$ is closed under negation.

2. $PSPACE \subseteq FO[\texttt{pfp}, <]$. Given a PSPACE language $L \subseteq \{0,1\}^*$, write an $FO(\texttt{pfp}, <)$-formula φ s.t. on any input structure $([n], U, <)$, φ is true iff $U \in L$. Note: we can't use the time any more.

Discussion

- Do we need order, e.g. could be the case that $FO(\texttt{lfp}) = PTIME$ (without $<$)?

Discussion

- Do we need order, e.g. could be the case that $FO(\texttt{lfp}) = PTIME$ (without $<$)? Yes: $FO(\texttt{lfp}) \subseteq L^{\omega}_{\infty\omega}$ and cannot express EVEN.

Discussion

- Do we need order, e.g. could be the case that $FO(\mathtt{lfp}) = PTIME$ (without $<$)? Yes: $FO(\mathtt{lfp}) \subseteq L^{\omega}_{\infty\omega}$ and cannot express EVEN.
- Clearly $FO(\mathtt{ifp}) \subseteq FO(\mathtt{pfp})$. Could they be equal?

Discussion

- Do we need order, e.g. could be the case that $FO(\texttt{lfp}) = PTIME$ (without $<$)? Yes: $FO(\texttt{lfp}) \subseteq L^{\omega}_{\infty\omega}$ and cannot express EVEN.
- Clearly $FO(\texttt{ifp}) \subseteq FO(\texttt{pfp})$. Could they be equal?
  - If $FO(\texttt{ifp}) = FO(\texttt{pfp})$ then they remain $=$ after adding $<$, hence $PTIME = PSPACE$.

Discussion

- Do we need order, e.g. could be the case that $FO(\texttt{lfp}) = PTIME$ (without $<$)? Yes: $FO(\texttt{lfp}) \subseteq L^{\omega}_{\infty\omega}$ and cannot express EVEN.
- Clearly $FO(\texttt{ifp}) \subseteq FO(\texttt{pfp})$. Could they be equal?
  - If $FO(\texttt{ifp}) = FO(\texttt{pfp})$ then they remain = after adding $<$, hence $PTIME = PSPACE$.
  - Abiteboul and Vianu proved the converse: if $PTIME = PSPACE$ then $FO(\texttt{ifp}) = FO(\texttt{pfp})$. The proof uses the FO^k types in a very clever way discuss in class.

Discussion

- Do we need order, e.g. could be the case that $FO(\mathtt{lfp}) = PTIME$ (without $<$)? Yes: $FO(\mathtt{lfp}) \subseteq L^{\omega}_{\infty\omega}$ and cannot express EVEN.
- Clearly $FO(\mathtt{ifp}) \subseteq FO(\mathtt{pfp})$. Could they be equal?
  - If $FO(\mathtt{ifp}) = FO(\mathtt{pfp})$ then they remain $=$ after adding $<$, hence $PTIME = PSPACE$.
  - Abiteboul and Vianu proved the converse: if $PTIME = PSPACE$ then $FO(\mathtt{ifp}) = FO(\mathtt{pfp})$. The proof uses the FO^k types in a very clever way discuss in class.
- If we could use some game to separate $FO(\mathtt{ifp}) \neq FO(\mathtt{pfp})$, then we have proven $PTIME \neq PSPACE$!

Discussion

- Do we need order, e.g. could be the case that $FO(\texttt{lfp}) = PTIME$ (without $<$)? Yes: $FO(\texttt{lfp}) \subseteq L^{\omega}_{\infty\omega}$ and cannot express EVEN.
- Clearly $FO(\texttt{ifp}) \subseteq FO(\texttt{pfp})$. Could they be equal?
  - If $FO(\texttt{ifp}) = FO(\texttt{pfp})$ then they remain $=$ after adding $<$, hence $PTIME = PSPACE$.
  - Abiteboul and Vianu proved the converse: if $PTIME = PSPACE$ then $FO(\texttt{ifp}) = FO(\texttt{pfp})$. The proof uses the FO^k types in a very clever way discuss in class.
- If we could use some game to separate $FO(\texttt{ifp}) \neq FO(\texttt{pfp})$, then we have proven $PTIME \neq PSPACE$!
- Main open problem in FMT: find a logic for $PTIME$ (no order)

Descriptive Complexity: Overview of Results

- $\text{FO}(+, *) = \text{FO}(<, \text{BIT}) = AC^0$
- $\text{FO}(\text{det-TC}, <) =$LOGSPACE, and $\text{FO}(\text{TC}, <) =$NLOGSPACE; will omit this
- $\text{FO}(\text{LeastFixpoint}, <) =\text{FO}(\text{InflationaryFixpoint}, <) =$PTIME
- $\text{FO}(\text{PartialFixpoint}, <) =$PSPACE
- $\exists$SO=NP

Next: combined complexity.

Combined Complexity

We sill study both FO and the restriction to the quantifier prefix $\exists^*$.
$\exists^*$ is important in databases: Unions of Conjunctive Queries with negation.

UCQ with negation (same as non-recursive datalog with negation):

$$\texttt{Answer} \leftarrow E(x,y) \wedge E(y,z) \wedge E(z,y)$$
$$\texttt{Answer} \leftarrow \neg E(x,y) \wedge \neg E(y,z) \wedge \neg E(z,y)$$

what does it say?

Combined Complexity

We sill study both FO and the restriction to the quantifier prefix $\exists^*$.
$\exists^*$ is important in databases: Unions of Conjunctive Queries with negation.

UCQ with negation (same as non-recursive datalog with negation):

$$\texttt{Answer} \leftarrow E(x,y) \wedge E(y,z) \wedge E(z,y)$$
$$\texttt{Answer} \leftarrow \neg E(x,y) \wedge \neg E(y,z) \wedge \neg E(z,y)$$

what does it say?
In the $\exists^*$ fragment:

$$\exists x \exists y \exists z (E(x,y) \wedge E(y,z) \wedge E(z,y) \vee \neg E(x,y) \wedge \neg E(y,z) \wedge \neg E(z,y))$$

Special case: Conjunctive Query (CQ) means no $\vee$ and no $\neg$.

Combined Complexity

Theorem

The combined complexity of the $\exists^*$ fragment of FO is in NP.

Theorem

The combined complexity of FO is in PSPACE.

In class: give a algorithm that runs in NP (PSPACE) and does this: given $\boldsymbol{A}, \varphi$, checks if $\boldsymbol{A} \vDash \varphi$.

Combined Complexity

Theorem

The combined complexity of the $\exists^*$ fragment of FO is in NP.

Theorem

The combined complexity of FO is in PSPACE.

In class: give a algorithm that runs in NP (PSPACE) and does this: given $\boldsymbol{A}, \varphi$, checks if $\boldsymbol{A} \models \varphi$.

Can we design better algorithms?

Combined Complexity

No better algorithm is possible!

Then there exists a structure $\boldsymbol{A}$ such that:

Theorem

The expression complexity for CQ (a subset of $\exists^*$-FO) is NP-complete.

Theorem

The expression complexity for FO is PSPACE-complete.

The structure $\boldsymbol{A}$ is the same in both. We will prove them together.

Review of SAT and QBF

The SAT problem is: given a Boolean formula $F(X_1, \ldots, X_n)$ check if it has a satisfying assignment.

The QBF problem is: given a quantified Boolean formula $Q_1X_1, Q_2X_2, \ldots F(X_1, \ldots, X_n)$, check if it is true.
E.g. $\forall X_1 \exists X_2 \forall X_3 (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$.

SAT is the special case $\exists X_1 \cdots \exists X_n F(X_1, \ldots, X_n)$.

Theorem

(1) SAT is NP-complete. (2) QBF is SPACE-complete. These hold even if F is a 3CNF.

Review of SAT and QBF

The SAT problem is: given a Boolean formula $F(X_1, \ldots, X_n)$ check if it has a satisfying assignment.

The QBF problem is: given a quantified Boolean formula $Q_1X_1, Q_2X_2, \ldots F(X_1, \ldots, X_n)$, check if it is true.
E.g. $\forall X_1 \exists X_2 \forall X_3 (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$.

SAT is the special case $\exists X_1 \cdots \exists X_n F(X_1, \ldots, X_n)$.

Theorem

(1) SAT is NP-complete. (2) QBF is SPACE-complete. These hold even if F is a 3CNF.

Review of SAT and QBF

The SAT problem is: given a Boolean formula $F(X_1, \ldots, X_n)$ check if it has a satisfying assignment.

The QBF problem is: given a quantified Boolean formula $Q_1X_1, Q_2X_2, \ldots F(X_1, \ldots, X_n)$, check if it is true.
E.g. $\forall X_1 \exists X_2 \forall X_3 (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$.

SAT is the special case $\exists X_1 \cdots \exists X_n F(X_1, \ldots, X_n)$.

Theorem

(1) SAT is NP-complete. (2) QBF is SPACE-complete. These hold even if F is a 3CNF.

Review of SAT and QBF

The SAT problem is: given a Boolean formula $F(X_1, \ldots, X_n)$ check if it has a satisfying assignment.

The QBF problem is: given a quantified Boolean formula $Q_1X_1, Q_2X_2, \ldots F(X_1, \ldots, X_n)$, check if it is true.
E.g. $\forall X_1 \exists X_2 \forall X_3 (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$.

SAT is the special case $\exists X_1 \cdots \exists X_n F(X_1, \ldots, X_n)$.

Theorem

(1) SAT is NP-complete. (2) QBF is SPACE-complete. These hold even if F is a 3CNF.

Review of SAT and QBF

The SAT problem is: given a Boolean formula $F(X_1, \ldots, X_n)$ check if it has a satisfying assignment.

The QBF problem is: given a quantified Boolean formula $Q_1X_1, Q_2X_2, \ldots F(X_1, \ldots, X_n)$, check if it is true.
E.g. $\forall X_1 \exists X_2 \forall X_3 (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_2 \vee X_3)$.

SAT is the special case $\exists X_1 \cdots \exists X_n F(X_1, \ldots, X_n)$.

Theorem

(1) SAT is NP-complete. (2) QBF is SPACE-complete. These hold even if F is a 3CNF.

Proof

In a 3CNF there are 4 kinds of 3-clauses:

$$X \vee Y \vee Z \qquad \neg X \vee Y \vee Z \qquad \neg X \vee \neg Y \vee Z \qquad \neg X \vee \neg Y \vee \neg Z$$

Consider the structure $\boldsymbol{A}$ with domain $\{0, 1\}$ and with four relations:

$$R_0 = \begin{array}{|c|c|c|} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \qquad R_1 = \begin{array}{|c|c|c|} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{array} \qquad R_2 = \begin{array}{|c|c|c|} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{array} \qquad R_3 = \begin{array}{|c|c|c|} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array}$$

SAT to CQ by example:

$$(X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \exists x_1 \exists x_2 \exists x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

QBE to FO by example:

$$\forall X_1 \exists X_2 \forall X_3 (X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \forall x_1 \exists x_2 \forall x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

In both cases: F is SAT iff $\boldsymbol{A} \models \varphi$ why?

Proof

In a 3CNF there are 4 kinds of 3-clauses:

$$X \vee Y \vee Z \qquad \neg X \vee Y \vee Z \qquad \neg X \vee \neg Y \vee Z \qquad \neg X \vee \neg Y \vee \neg Z$$

Consider the structure $\boldsymbol{A}$ with domain $\{0,1\}$ and with four relations:

$$R_0 = \begin{array}{|c|c|c|}\hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline\end{array} \qquad R_1 = \begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ \hline\end{array} \qquad R_2 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \hline\end{array} \qquad R_3 = \begin{array}{|c|c|c|}\hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \hline\end{array}$$

SAT to CQ by example:

$$(X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \exists x_1 \exists x_2 \exists x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

QBE to FO by example:

$$\forall X_1 \exists X_2 \forall X_3 (X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \forall x_1 \exists x_2 \forall x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

In both cases: F is SAT iff $\boldsymbol{A} \models \varphi$ why?

Proof

In a 3CNF there are 4 kinds of 3-clauses:

$$X \vee Y \vee Z \qquad \neg X \vee Y \vee Z \qquad \neg X \vee \neg Y \vee Z \qquad \neg X \vee \neg Y \vee \neg Z$$

Consider the structure $\boldsymbol{A}$ with domain $\{0, 1\}$ and with four relations:

$$R_0 = \begin{array}{|c|c|c|} \hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline \end{array} \qquad R_1 = \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ \hline \end{array} \qquad R_2 = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \hline \end{array} \qquad R_3 = \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \hline \end{array}$$

SAT to CQ by example:

$$(X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \exists x_1 \exists x_2 \exists x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

QBE to FO by example:

$$\forall X_1 \exists X_2 \forall X_3 (X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \forall x_1 \exists x_2 \forall x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

In both cases: F is SAT iff $\boldsymbol{A} \models \varphi$ why?

Proof

In a 3CNF there are 4 kinds of 3-clauses:

$$X \vee Y \vee Z \qquad \neg X \vee Y \vee Z \qquad \neg X \vee \neg Y \vee Z \qquad \neg X \vee \neg Y \vee \neg Z$$

Consider the structure $\boldsymbol{A}$ with domain $\{0, 1\}$ and with four relations:

$$R_0 = \begin{array}{|c|c|c|}\hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline\end{array} \qquad R_1 = \begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ \hline\end{array} \qquad R_2 = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \hline\end{array} \qquad R_3 = \begin{array}{|c|c|c|}\hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \hline\end{array}$$

SAT to CQ by example:

$(X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \exists x_1 \exists x_2 \exists x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$

QBE to FO by example:

$\forall X_1 \exists X_2 \forall X_3 (X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \forall x_1 \exists x_2 \forall x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$

In both cases: F is SAT iff $\boldsymbol{A} \models \varphi$ why?

Proof

In a 3CNF there are 4 kinds of 3-clauses:

$$X \vee Y \vee Z \qquad \neg X \vee Y \vee Z \qquad \neg X \vee \neg Y \vee Z \qquad \neg X \vee \neg Y \vee \neg Z$$

Consider the structure $\boldsymbol{A}$ with domain $\{0, 1\}$ and with four relations:

$$R_0 = \begin{array}{|c|c|c|} \hline 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline \end{array} \qquad R_1 = \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ \hline \end{array} \qquad R_2 = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \hline \end{array} \qquad R_3 = \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \hline \end{array}$$

SAT to CQ by example:

$$(X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \exists x_1 \exists x_2 \exists x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

QBE to FO by example:

$$\forall X_1 \exists X_2 \forall X_3 (X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4) \mapsto \forall x_1 \exists x_2 \forall x_3 \exists x_4 R_0(x_1, x_2, x_3) \wedge R_1(x_3, x_1, x_4) \wedge R_0(x_2, x_3, x_4)$$

In both cases: F is SAT iff $\boldsymbol{A} \models \varphi$ why?

Discussion

- Data complexity of FO is AC^0 very low!
- For database fans: the expression and combined complexity of CQ (and hence select-from-where SQL queries) is NP-complete.
- Expression complexity and combined complexity of FO are $PSPACE$-complete very high!
- We omit the expression complexity of extensions of FO (hint: they get even higher).

Representing Strings

Fix an alphabet Σ, e.g. $\Sigma = \{a, b, c\}$.
A word $w \in \Sigma^*$ can be encoded as a structure over the alphabet $\sigma = (<, P_a(\cdot), P_b(\cdot), P_c(\cdot))$.
In class represent *aabaca*.

A sentence φ defines a language $\{w \mid w \vDash \varphi\}$.
E.g. $\forall x \forall y (x < y \wedge P_a(x) \wedge P_a(y) \rightarrow \exists z (x < y < z \wedge P_b(z)))$
Assuming alphabet $\{a, b\}$ it says "between any two a's there is a b": $b^*.(a.b^+)^*.(a|\varepsilon)$

- What languages can be define in FO?
- What languages can be define in MSO?

Representing Strings

Fix an alphabet Σ, e.g. $\Sigma = \{a, b, c\}$.
A word $w \in \Sigma^*$ can be encoded as a structure over the alphabet $\sigma = (<, P_a(\cdot), P_b(\cdot), P_c(\cdot))$.
In class represent *aabaca*.

A sentence φ defines a language $\{w \mid w \vDash \varphi\}$.
E.g. $\forall x \forall y (x < y \wedge P_a(x) \wedge P_a(y) \rightarrow \exists z (x < y < z \wedge P_b(z)))$
Assuming alphabet $\{a, b\}$ it says "between any two a's there is a b":
$b^*.(a.b^+)^*.(a|\varepsilon)$

- What languages can be define in FO?
- What languages can be define in MSO?

Representing Strings

Fix an alphabet Σ, e.g. $\Sigma = \{a, b, c\}$.
A word $w \in \Sigma^*$ can be encoded as a structure over the alphabet $\sigma = (<, P_a(\cdot), P_b(\cdot), P_c(\cdot))$.
In class represent *aabaca*.

A sentence φ defines a language $\{w \mid w \vDash \varphi\}$.
E.g. $\forall x \forall y(x < y \wedge P_a(x) \wedge P_a(y) \rightarrow \exists z(x < y < z \wedge P_b(z)))$
Assuming alphabet $\{a, b\}$ it says "between any two a's there is a b":
$b^*.(a.b^+)^*.(a|\varepsilon)$

- What languages can be define in FO?
- What languages can be define in MSO?

Regular Expressions

Fix an alphabet Σ. Regular expressions are:

$$
\begin{aligned}
E ::= & \emptyset | \varepsilon | a \in \Sigma \\
& E \cup E | E.E \\
& C(E) \qquad\qquad\qquad\qquad \text{complement} \\
& E^*
\end{aligned}
$$

E is called *star-free* if it is equivalent to an expression without $*$.
In class assuming $\Sigma = \{a, b\}$, which expressions are star-free?

$C(\emptyset)$ $\qquad b^*.(a.b^*)^*$ $\qquad (a.b)^*$ $\qquad (a.a)^*$

FO on Strings

Theorem

A language L is star-free iff it is defined in FO.

MSO on Strings

Theorem

A language L is regular iff it is defined in MSO.

Proof

TBD (or, better, in class)

Applications

- There exists a regular language which is not star-free. which one?
- SAT for MSO on strings is decidable. what is the complexity?
- The data complexity for MSO on strings is linear time! what is the data complexity of MSO?
- On strings: $\exists MSO = \forall MSO = MSO$ why?

Courcelle's Theorem

Let $\mathcal{C}$ be a class of structures with bounded tree-width. discuss tw in class; we will return to it.

Theorem (Courcelle)

Every formula in $\varphi \in MSO$ can be evaluated in linear time over structures of bounded tree-width.

This is an amazing result! Caveats:

- The expression complexity is horrible (non-elementary).
- We need a tree decomposition of the structure (i.e. database) $\boldsymbol{A}$: this is NP-complete in general.
- If we have a promise that the treewidth is $\leq k$, then we can compute a TD in time $O(n^k)$; but "real" databases rarely have bounded tw.

Discussion

- MSO is very powerful in general: Monadic NP.
- But over strings it can only express regular languages: linear time.
- Even over trees, or “tree-like” structures MSO is still in linear time.
- Problem: data in real life is not “tree-like”!