

Milestones in Descriptive Complexity Theory

ANU Logic Summer School, 2019

Presented by

Sasha Rubin
School of Computer Science
University of Sydney



THE UNIVERSITY OF
SYDNEY



What is complexity theory?

Analyses and classifies problems by the amount of a resource required for handling them.

Problem:

- defines a computational task
- specifies what the input is and what the output should be

Algorithm:

- a step-by-step recipe to go from input to output
- different from implementation

Complexity analysis:

- bound on the resources used for solving/expressing the problem.

What is complexity theory?

Analyses and classifies problems by the amount of a resource required for handling them.

Which resources?

- Computational complexity theory takes computational resources for **solving** the problem, e.g., time and space.
- Descriptive complexity theory takes logical resources for **expressing** the problem, e.g., variables and quantifiers.

Amazingly, there is a beautiful and tight connection between these two notions of complexity.

The goal of this course is to convince you of this fact.

Logistics

- Ask questions
- 💡 (important idea/fact)
- ⚙️ (do in class)
- 🏠 (do at home)

Assumed notions and notation

- set theoretic predicates and operations: $X \subseteq Y$, $X \subset Y$, $X \cup Y$, $X \cap Y$, $X \setminus Y$, $X \times Y$, $\bar{y} \in X$ or $X(\bar{y})$.
- string predicates and operations: $u \cdot v$, Σ^* .
- functions: $f : X \rightarrow Y$, $f(x) = y$, $\text{dom}(f)$, $\text{rng}(f)$, bijections
- graphs: $G = (V, E)$, paths, circuits
- equivalence relations, e.g., $x \equiv y$:if $f(x) = f(y)$
- linear orderings, e.g., lexicographic ordering

What is DCT?

- Overview of the course

- Logic and Definability

- \exists MSO captures REG on strings

 - Automata primer

 - An excursion into finite model theory

- Fagin's Theorem: \exists SO captures NPTIME

 - Computational complexity crash course

 - Computational complexity of logic problems

 - Sketch proof of Fagin's Theorem

What is DCT?

- Overview of the course

- Logic and Definability

- \exists MSO captures REG on strings

 - Automata primer

 - An excursion into finite model theory

- Fagin's Theorem: \exists SO captures NPTIME

 - Computational complexity crash course

 - Computational complexity of logic problems

 - Sketch proof of Fagin's Theorem

Overview of the course

Example: 3-Colourability (3-COL)

Definition

- A graph $G = (V, E)$ is 3-colourable if there is a mapping $\kappa : V \rightarrow \{r, g, b\}$ such that for all $(v, w) \in E$, $\kappa(v) \neq \kappa(w)$.

Example: 3-Colourability (3-COL)

Definition

- A graph $G = (V, E)$ is 3-colourable if there is a mapping $\kappa : V \rightarrow \{r, g, b\}$ such that for all $(v, w) \in E$, $\kappa(v) \neq \kappa(w)$.

Computational question. What time resources are sufficient for deciding if a given graph is 3-colourable?

- Guess a mapping κ and cycle through every edge to test if the colours are different. (A nondeterministic algorithm accepts an input if there is a guess which leads to acceptance, otherwise it rejects)
- This is a nondeterministic algorithm that runs in time $O(|G|)$.
- This puts 3-COL in the computational complexity class NPTIME.

Example: 3-Colourability (3-COL)

Definition

- A graph $G = (V, E)$ is 3-colourable if there is a mapping $\kappa : V \rightarrow \{r, g, b\}$ such that for all $(v, w) \in E$, $\kappa(v) \neq \kappa(w)$.

Descriptive question. What logical resources are sufficient for describing the set of 3-colourable graphs?

- The property can be expressed by the formula

$$\exists \kappa : V \rightarrow \{r, g, b\} \quad \forall (v, w) \in E \quad \kappa(v) \neq \kappa(w)$$

- This puts 3-COL in the logical complexity class Existential Second-Order Logic (\exists SO).

Milestone 1: Fagin's Theorem

On finite structures:

$$\text{NPTIME} = \exists\text{SO}$$

- A computational problem can be solved by a nondeterministic polynomial time machine iff it can be expressed in existential second order logic.

Milestone 2: Immerman-Livchak-Vardi's Theorem

On finite ordered-structures:

$$\text{PTIME} = \text{FO}[\text{LFP}]$$

A computational problem on finite ordered structures can be computed by a (deterministic) polynomial time machine iff it can be expressed in first order logic extended by least fixed-point operators.

Milestone 3: Büchi-Elgot-Trakhtenbrot's Theorem

On finite strings:

$$\text{REG} = \text{MSO}$$

A computational problem on finite strings can be computed by a finite-state automaton iff it can be expressed in existential monadic second order logic.

Descriptive Complexity: A Logician's Approach to Computation

N. Immerman

A basic issue in computer science is the complexity of problems. If one is doing a calculation once on a medium-sized input, the simplest algorithm may be the best method to use, even if it is not the fastest. However, when one has a subproblem that will have to be solved millions of times, optimization is important. A fundamental issue in theoretical computer science is the computational complexity of problems. How much time and how much memory space is needed to solve a particular problem?

- Here are a few examples of such problems:
1. **Reachability:** Given a directed graph and two specified points s, t , determine if there is a path from s to t . A simple, linear-time algorithm marks s and then continues to mark every vertex at the head of an edge whose tail is marked. When no more vertices can be marked, t is reachable from s iff it has been marked.
 2. **Min-triangulation:** Given a polygon in the plane and a length L , determine if there is a triangulation of the polygon of total length less than or equal to L . Even though there are exponentially many possible triangulations, a dynamic programming algorithm can find an optimal one in $O(n^3)$ steps.
 3. **Three-Colorability:** Given an undirected graph, determine whether its vertices can be colored using three colors with no two adjacent vertices having the same color. Again

there are exponentially many possibilities, but in this case no known algorithm is faster than exponential time.

Computational complexity measures how much time and/or memory space is needed as a function of the input size. Let $\text{TIME}(n)$ be the set of problems that can be solved by algorithms that perform at most $O(n)$ steps for inputs of size n . The complexity class polynomial time (P) is the set of problems that are solvable in time at most some polynomial in n .

$$P = \bigcup_{k \geq 1} \text{TIME}(n^k)$$

Even though the class $\text{TIME}(n)$ is sensitive to the exact machine model used in the computations, the class P is quite robust. The problems Reachability and Min-triangulation are elements of P .

Some important computational problems appear to require more than polynomial time. An interesting class of such problems is contained in **nondeterministic polynomial time** (NP). A **nondeterministic computation** is one that may make arbitrary choices as it works. If any of those choices leads to an accept state, then we say the input is accepted. As an example, let us consider the three-colorability problem. A nondeterministic algorithm traverses the input graph, arbitrarily assigning to each vertex a color: red, yellow, or blue. Then it checks whether each edge joins vertices of different colors. If so, it accepts.

A nondeterministic computation can be modeled as a tree whose root is the starting config-

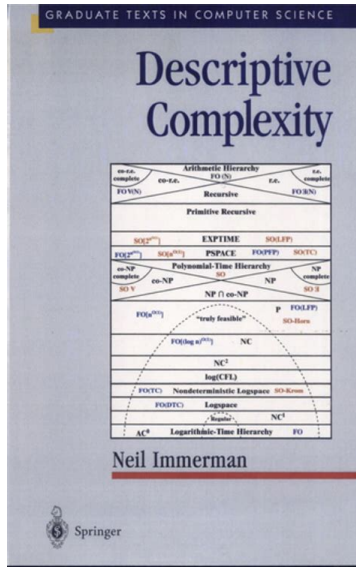
N. Immerman is professor in the Department of Computer Science, University of Massachusetts at Amherst. His e-mail address is immerman@cs.umass.edu.

OCTOBER 1995

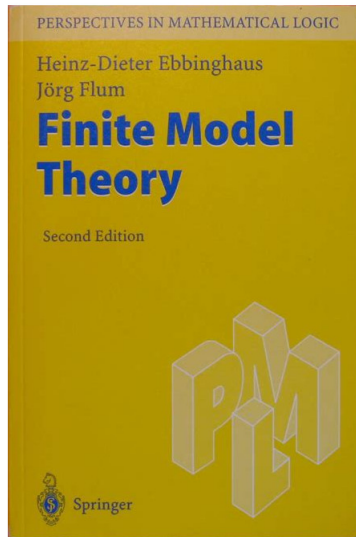
NOTICES OF THE AMS

1127

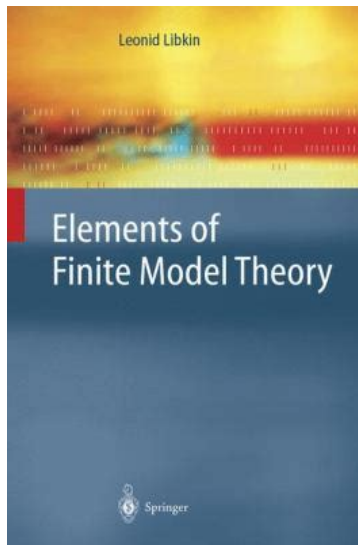
A lot more?



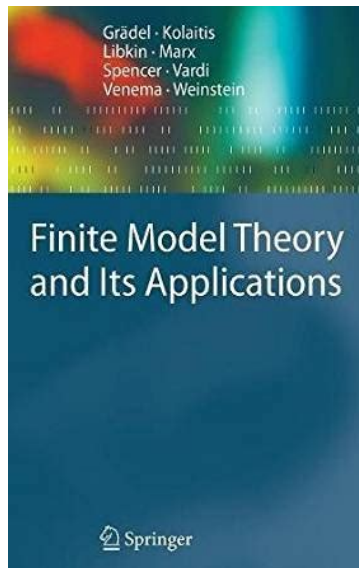
A lot more?



A lot more?



A lot more?



Finite Model Theory

The study of logics on classes of finite structures

Examples of logics

- First-order logic
- Second-order logic
- Logics with fixed-point operators
- Infinitary logics
- Logics with generalised quantifiers

Examples of classes of finite structures

- all finite graphs
- all finite ordered graphs
- all finite planar graphs
- all finite strings
- all finite trees

Reasons for developing FMT

Connections and applications to CS

- Algorithms and computational complexity (this course!)
- Automata theory (this course!)
- Database Theory (SQL, Datalog)
- Verification (in AI, Distributed Computing, Business Process Management)

For its own sake

- Traditional focus of mathematical logic has been on fixed infinite structures (e.g., real-arithmetic), or classes of finite and infinite structures.
- New phenomena emerge when one focuses on classes of finite structures.
- 💡 FO loses its unique status and other logics also feature

Logic and Definability

Basic concepts from logic

Definitions

- **Signature** σ is a finite tuple (R_1, \dots, R_m) of relation symbols of specified arities. (Function symbols can also be handled)
- **σ -structure** $\mathbf{A} = (A, (R_1)^A, \dots, (R_m)^A)$ is a set A , called the domain, and relations R^A on A of the specified arities.

We usually drop superscripts and write (A, R_1, \dots, R_m) .

Basic concepts from logic

Definitions

- **Signature** σ is a finite tuple (R_1, \dots, R_m) of relation symbols of specified arities. (Function symbols can also be handled)
- **σ -structure** $\mathbf{A} = (A, (R_1)^A, \dots, (R_m)^A)$ is a set A , called the domain, and relations R^A on A of the specified arities.

We usually drop superscripts and write (A, R_1, \dots, R_m) .

NB. All σ -structures in this course are finite (i.e., have finite domain), unless otherwise stated.

Structures

Graphs

- Signature (E) with $ar(E) = 2$
- $\mathbf{G} = (V, E)$ where $E \subseteq V^2$

Structures

Graphs

- Signature (E) with $ar(E) = 2$
- $\mathbf{G} = (V, E)$ where $E \subseteq V^2$

Strings over alphabet Σ

- Signature $(<, (P_a)_{a \in \Sigma})$ with $ar(<) = 2$ and $ar(P_a) = 1$.
- For $\Sigma = \{a, b, c\}$, the string $acaaac$ encoded as the structure $\mathbf{S}_{acaaac} := (V, <, P_a, P_b, P_c)$ where $V = \{1, 2, 3, 4, 5\}$, $<$ is the usual order on numbers, and $P_a = \{1, 3, 4\}$, $P_b = \emptyset$, $P_c = \{2, 5\}$.
- Structures $\mathbf{S} = (V, <, (P_a)_{a \in \Sigma})$ where $<$ is a linear order on V , the \bar{P} partition V are called **string structures** or **strings**.


First order (FO) logic

Fix signature σ .

Syntax

- first order variables: $x, y, z \dots$
- formulas:
 - atomic formulas: $x = y$, and $R(\bar{x})$ for R in σ
 - Boolean connectives \wedge, \neg
 - First-order quantification $\exists x, \exists y$

Semantics

- formulas are interpreted on σ -structures \mathbf{A}
- in particular, first-order variables vary over elements of the domain A of the structure.
-  Define $\mathbf{A} \models \varphi(\bar{a})$ where \bar{a} is an assignment of the free variables in φ to elements of A .
- Say that \mathbf{A} **satisfies** φ under assignment α .

Definability

Definition

- Fix signature σ
- For a sentence $\varphi \in \text{FO}$, write $\text{Mod}(\varphi)$ for the set of σ -structures satisfying φ .
- For a set P of σ -structures, if $P = \text{Mod}(\varphi)$ then we say that φ defines P .

FO definability

Signature of graphs (E) .

- The sentence

$$(\forall x)(\forall y)E(x, y) \rightarrow E(y, x)$$

defines the set of undirected graphs.



- The sentence

$$(\forall x)(\forall y)E(x, y) \rightarrow \neg(x = y)$$

defines the set of graphs with no self-loops.



FO definability

Signature of binary strings $\sigma := (<, P_a, P_b)$.

- The set of all binary-string structures is definable by an FO sentence that says that $<$ linear orders the domain and P_a, P_b partition the domain.
-  " x is the first (resp. last) position"
-  "position y is immediately after position x ", written $y = x + 1$.

FO definability

Signature of binary strings $\sigma := (<, P_a, P_b)$.


- The set of all binary-string structures is definable by an FO sentence that says that $<$ linear orders the domain and P_a, P_b partition the domain.
-  " x is the first (resp. last) position"
-  "position y is immediately after position x ", written $y = x + 1$.
- "the string contains the substring aab "

$$(\exists x)(\exists y)(\exists z)(y = x + 1 \wedge z = y + 1 \wedge P_a(x) \wedge P_a(y) \wedge P_b(z))$$

Second order logic (SO)

Second order logic (SO) extends FO by

- second-order variables $X, Y, Z \dots$ that vary over **relations** on the domain.
- atomic formulas $Z(\bar{x})$, etc.

 Every SO formula can be translated into an equivalent formula in prenex form:

$$(Q_1 X_1)(Q_2 X_2) \cdots (Q_k X_k) \Phi$$

where each $Q_i \in \{\exists, \forall\}$ and the only quantification in Φ is first order.

Fragments of prenex form:

- Existential second order logic (**ESO**): all Q_i s are \exists
- Universal second order logic (**USO**): all Q_i s are \forall



\exists SO definability on undirected graphs

- Fix the signature of graphs (E) .
- Show that the set of 3-colourable undirected graphs is definable in \exists SO, i.e., find an \exists SO sentence Φ such that $\mathbf{G} \models \Phi$ iff \mathbf{G} is a 3-colourable undirected graph.
- Being undirected is FO definable. So we focus on 3-colourability (assuming the graph is undirected).



\exists SO definability on undirected graphs


- Fix the signature of graphs (E) .
- Show that the set of 3-colourable undirected graphs is definable in \exists SO, i.e., find an \exists SO sentence Φ such that $\mathbf{G} \models \Phi$ iff \mathbf{G} is a 3-colourable undirected graph.
- Being undirected is FO definable. So we focus on 3-colourability (assuming the graph is undirected).

$$\begin{aligned} & (\exists X_1)(\exists X_2)(\exists X_3) \\ & (\forall x) \left[\bigvee_i X_i(x) \wedge \bigwedge_{i \neq j} X_i(x) \rightarrow \neg X_j(x) \right] \\ & \wedge \\ & (\forall x)(\forall y) \left[\bigwedge_i E(x, y) \rightarrow \neg (X_i(x) \wedge X_i(y)) \right] \end{aligned}$$

Monadic second order logic (MSO)

Monadic second order logic (MSO) extends FO by

- second-order variables $X, Y, Z \dots$ that vary over **subsets** of the domain.

 Every MSO formula can be translated into an equivalent formula prenex-form:

$$(Q_1 X_1)(Q_2 X_2) \cdots (Q_k X_k) \Phi$$

where each $Q_i \in \{\exists, \forall\}$ and the only quantification in Φ is first order.

Fragments of prenex-form:

- Existential monadic second order logic (**EMSO**): all Q_i s are \exists
- Universal monadic second order logic (**UMSO**): all Q_i s are \forall



Definability

For undirected graphs, which of these can you express in $\exists\text{SO}$, $\exists\text{MSO}$, $\forall\text{MSO}$:

- "there is a path from x to y "
- "the graph is connected"
- "the graph has a perfect matching"
- "the graph has a Hamiltonian cycle"



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"the string is a sequence of a 's followed by a sequence of b 's"



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"the string is a sequence of a 's followed by a sequence of b 's"

$$(\forall x)(\forall y)(P_a(x) \wedge P_b(y) \rightarrow x < y)$$



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"all, and only, odd positions contain a 's"



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"all, and only, odd positions contain a 's"

$$P_a(\min) \wedge (\forall x < \max)(P_a(x) \leftrightarrow \neg P_a(x + 1))$$



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"all odd positions contain a 's"



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"all odd positions contain a 's"

$$(\exists X)[X(\min) \wedge (\forall x < \max)(X(x) \leftrightarrow \neg X(x+1)) \wedge \\ (\forall x)(X(x) \rightarrow P_a(x))]$$

It turns out that this property is not expressible in FO.



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"all positions divisible by 2 or 3 contain a 's"



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"all positions divisible by 2 or 3 contain a 's"

$(\exists X)(\exists Y)$

$[X(\min) \wedge (\forall x < \max)(X(x) \leftrightarrow \neg X(x+1)) \wedge$

$[Y(\min) \wedge (\forall x < \max)(Y(x) \leftrightarrow (\neg Y(x+1) \wedge \neg Y(x+2)))] \wedge$

$(\forall x)(X(x) \vee Y(x) \rightarrow P_a(x))]$

Is this expressible using one existential monadic second-order quantifier?



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"the string contains the same number of a 's as b 's"



\exists SO definability on strings

Fix alphabet $\Sigma = \{a, b\}$. Express in FO, \exists MSO, \exists SO:

"the string contains the same number of a 's as b 's"

$$(\exists R)[\text{func}(R) \wedge \text{dom}(R) = P_a \wedge \text{rng}(R) = P_b \wedge \text{inj}(R)]$$

It turns out that this property is not expressible in MSO.

What is DCT?

- Overview of the course

- Logic and Definability

\exists MSO captures REG on strings

- Automata primer

- An excursion into finite model theory

Fagin's Theorem: \exists SO captures NPTIME

- Computational complexity crash course

- Computational complexity of logic problems

- Sketch proof of Fagin's Theorem



Framework

Fix a signature σ .

- A **problem** P is a set of σ -structures.
 - "the set of all connected graphs"
 - "the set of all graphs with a Hamiltonian circuit"
 - "the set of all strings with the same number of a 's as b 's"
- For a logic \mathcal{L} , a problem P is **\mathcal{L} -definable** if $P = \text{Mod}(\varphi)$ for some $\varphi \in \mathcal{L}$.
- For a complexity class \mathcal{C} , a problem P is **\mathcal{C} -solvable** if the complexity of deciding if a given σ -structure \mathbf{A} is in P is in \mathcal{C} .

Definition

- \mathcal{L} **captures** \mathcal{C} if every problem is \mathcal{L} -definable iff it is \mathcal{C} -solvable.

Machine-independent characterisation of a complexity class.

Automata primer

Automata Primer

Deterministic finite-state automaton (DFA) $M = (\Sigma, Q, q_0, \delta, F)$

- Σ finite alphabet
- Q finite set of states
- $q_0 \in Q$ initial state
- $\delta : Q \times \Sigma \rightarrow Q$ transition function
- $F \subseteq Q$ final states

Acceptance

- A string $w \in \Sigma^*$ is **accepted** by M if there is a path labeled by w from the initial state to a final state.

Automata are language recognisers

- M **recognises** the set of strings $w \in \Sigma^*$ it accepts
- A language L is **regular** if it is recognised by some DFA
- **REG** denotes the set of all regular languages

Regular languages



Examples ($\Sigma = \{a, b\}$)

- all strings
- "the string is a sequence of a 's followed by a sequence of b 's"
- "all, and only, odd positions contain a 's"
- "all odd positions contain a 's"



The set "strings containing the same number of a 's as b 's" is not regular.

Regular languages



Examples ($\Sigma = \{a, b\}$)

- all strings
- "the string is a sequence of a 's followed by a sequence of b 's"
- "all, and only, odd positions contain a 's"
- "all odd positions contain a 's"



The set "strings containing the same number of a 's as b 's" is not regular.

Proof. Suppose there is a DFA M recognising this set.

- for every $n \in \mathbb{N}$ the string a^n labels a path ending in state q_n .
- since $a^n b^n$ is accepted by M , there is a path from q_n labeled b^n to a final state.
- Since M is finite, there exist $n \neq m$ such that $q_n = q_m$.
- Thus M must also accept $a^n b^m$, contradiction.

REG = MSO = \exists MSO on strings

Recall our coding:

- string w over alphabet Σ is coded as the structure $\mathbf{S}_w := (\{1, 2, \dots, |w|\}, <, (P_a)_{a \in \Sigma})$.
- set $L \subseteq \Sigma^*$ is coded as the set $\{\mathbf{S}_w : w \in L\}$ of string structures.

Theorem

The following are equivalent for a set $L \subseteq \Sigma^$ of finite strings:*

- L is recognised by a DFA
- $\{\mathbf{S}_w : w \in L\}$ is definable in \exists MSO
- $\{\mathbf{S}_w : w \in L\}$ is definable in MSO

In short:

REG = MSO = \exists MSO, on finite strings

DFA to \exists MSO

For DFA $M = (\Sigma, Q, q_0, \delta, F)$ with $Q = \{1, 2, \dots, m\}$, we define an \exists MSO formula Φ_M such that M accepts $w \in \Sigma^*$ iff $\mathbf{S}_w \models \Phi_M$.

DFA to \exists MSO

For DFA $M = (\Sigma, Q, q_0, \delta, F)$ with $Q = \{1, 2, \dots, m\}$, we define an \exists MSO formula Φ_M such that M accepts $w \in \Sigma^*$ iff $\mathbf{S}_w \models \Phi_M$.

$$(\exists Q_1) \cdots (\exists Q_m) \text{ Partition} \wedge \text{Start} \wedge \text{Trans} \wedge \text{End}$$

where

- Partition says "the sets \bar{Q} partition the domain"
- Start says "the first state is q_0 ", i.e., $Q_{q_0}(\min)$
- Trans says "the sets \bar{Q} encode the path labeled by the input word, except for the final transition", i.e.,
$$\bigwedge_{i,a} (\forall t < \max) (P_a(t) \wedge Q_i(t) \rightarrow Q_{\delta(i,a)}(t+1))$$
- End says "the last transition of the path results in a final state", i.e., $\bigvee_{\{(i,a):\delta(i,a) \in F\}} (Q_i(\max) \wedge P_a(\max))$

MSO to DFA

Aim: For MSO-formula φ , we define DFA M_φ such that $\mathbf{S}_w \models \varphi$ iff M_φ accepts w .

First Attempt. Define automaton $(\Sigma, Q, q_0, \delta, F)$ where

- states are the sets $Th(u) := \{\varphi \in \text{MSO} : \mathbf{S}_u \models \varphi\}$ for $u \in \Sigma^*$.
- initial state $Th(\lambda)$ where λ is the empty string
- final states are those $Th(u)$ that contain φ
- transitions $\delta(Th(u), a) = Th(ua)$



What's wrong with this?

MSO to DFA

Aim: For MSO-formula φ , we define DFA M_φ such that $\mathbf{S}_w \models \varphi$ iff M_φ accepts w .

First Attempt. Define automaton $(\Sigma, Q, q_0, \delta, F)$ where

- states are the sets $Th(u) := \{\varphi \in \text{MSO} : \mathbf{S}_u \models \varphi\}$ for $u \in \Sigma^*$.
- initial state $Th(\lambda)$ where λ is the empty string
- final states are those $Th(u)$ that contain φ
- transitions $\delta(Th(u), a) = Th(ua)$



What's wrong with this?

- $u \neq v$ implies $Th(u) \neq Th(v)$
- i.e., the automaton has infinitely many states

The road ahead...

- To fix this, we keep track of fewer formulas, but enough to update the states and determine when φ holds
- In particular, we keep only formulas with the same quantifier rank as φ .
- This ensures that there are finitely many states.
- But then we need to show that the transitions are well-defined.

To explain all this, we take...

An excursion into finite model theory

Quantifier-rank

Definition

- The quantifier-rank of φ is the depth of the quantifier-nesting in φ .

Examples

- $E(x, y)$ has qr 0.
- $(\forall x)(\exists y)(\forall z)(E(x, y) \wedge \neg E(y, z))$ has qr 3.
- $[(\exists x)E(x, x)] \wedge [(\forall x)(\exists y)E(x, y)]$ has qr 2.
- $qr(\phi) = r$ implies ϕ is a Bool combination of $\exists X.\psi$ with $qr(\psi) \leq r - 1$, and at least one ψ with $qr(\psi) = r - 1$.

Quantifier-rank

Definition

- The quantifier-rank of φ is the depth of the quantifier-nesting in φ .

Definitions. Fix logic \mathcal{L} .

- $Th_r^{\mathcal{L}}(\mathbf{A}) := \{\varphi \in \mathcal{L} : qr(\varphi) = r, \mathbf{A} \models \varphi\}$
- Write $\mathbf{A} \equiv_r^{\mathcal{L}} \mathbf{B}$ if $Th_r^{\mathcal{L}}(\mathbf{A}) = Th_r^{\mathcal{L}}(\mathbf{B})$, i.e., they satisfy the same \mathcal{L} -sentences of qr r . (equivalence relation)

Quantifier-rank

Definition

- The quantifier-rank of φ is the depth of the quantifier-nesting in φ .

Definitions. Fix logic \mathcal{L} .

- $Th_r^{\mathcal{L}}(\mathbf{A}) := \{\varphi \in \mathcal{L} : qr(\varphi) = r, \mathbf{A} \models \varphi\}$
- Write $\mathbf{A} \equiv_r^{\mathcal{L}} \mathbf{B}$ if $Th_r^{\mathcal{L}}(\mathbf{A}) = Th_r^{\mathcal{L}}(\mathbf{B})$, i.e., they satisfy the same \mathcal{L} -sentences of qr r . (equivalence relation)

Central properties:

- $\mathbf{A} \models Th_r(\mathbf{B})$ implies $\mathbf{A} \equiv_r^{\mathcal{L}} \mathbf{B}$.
- For every r , there are finitely many r -theories. (Show, by induction on r , that for every m , there are finitely many MSO-formulas on the m variables x_1, \dots, x_m of quantifier rank r up to logical equivalence.)

Distinguishing structures

- $\mathbf{A} = (\mathbb{Z}, <)$ and $\mathbf{B} = (\mathbb{Q}, <)$
- What FO formula distinguishes \mathbf{A} from \mathbf{B} ? What is the quantifier rank?
- Is there a formula of smaller quantifier rank that distinguishes them?

FO Ehrenfeucht-Fraïssé games with r moves

- *Spoiler* and *Duplicator* play on two σ -structures \mathbf{A} and \mathbf{B}
- Each play of the game has r moves. In each move
 - Spoiler picks an element of one of the structures.
 - Duplicator picks an element of the other structure.
- A typical play looks as follows:

Round	1	2	3	...	r
Spoiler	$a_1 \in A$	$b_2 \in B$	$b_3 \in B$	\dots	$a_r \in A$
Duplicator	$b_1 \in B$	$a_2 \in A$	$a_3 \in A$	\dots	$b_r \in B$

- Duplicator wins the play if (\mathbf{A}, \bar{a}) and (\mathbf{B}, \bar{b}) agree on the quantifier-free sentences.
- Duplicator wins the game, if he wins no matter what Spoiler does.



Ehrenfeucht-Fraïssé games capture logical equivalence

Theorem Duplicator wins the FO Ehrenfeucht-Fraïssé r -round game on \mathbf{A}, \mathbf{B} iff $Th_r^{\text{FO}}(\mathbf{A}) = Th_r^{\text{FO}}(\mathbf{B})$.



Ehrenfeucht-Fraïssé games capture logical equivalence

Theorem Duplicator wins the FO Ehrenfeucht-Fraïssé r -round game on \mathbf{A}, \mathbf{B} iff $Th_r^{\text{FO}}(\mathbf{A}) = Th_r^{\text{FO}}(\mathbf{B})$.

Illustration

- $\mathbf{A} = (\mathbb{Z}, <)$ and $\mathbf{B} = (\mathbb{Q}, <)$
- What FO formula distinguishes \mathbf{A} from \mathbf{B} ? What is the quantifier rank?
- Is there a FO formula of smaller quantifier rank that distinguishes them?



Ehrenfeucht-Fraïssé games capture logical equivalence

Theorem Duplicator wins the FO Ehrenfeucht-Fraïssé r -round game on \mathbf{A}, \mathbf{B} iff $Th_r^{\text{FO}}(\mathbf{A}) = Th_r^{\text{FO}}(\mathbf{B})$.

Illustration

- $\mathbf{A} = (\mathbb{Z}, <)$ and $\mathbf{B} = (\mathbb{Q}, <)$
- What FO formula distinguishes \mathbf{A} from \mathbf{B} ? What is the quantifier rank?
- Is there a FO formula of smaller quantifier rank that distinguishes them?

Play the game!

Ehrenfeucht-Fraïssé games for MSO

A similar game captures \equiv_r^{MSO}

- allow players to pick subsets A_i, B_i of the domains
- declare that Duplicator wins the play if $(\mathbf{A}, \bar{a}, \bar{A})$ and $(\mathbf{B}, \bar{b}, \bar{B})$ agree on the quantifier-free sentences.

Theorem Duplicator wins the MSO Ehrenfeucht-Fraïssé r -round game on \mathbf{A}, \mathbf{B} iff $Th_r^{\text{MSO}}(\mathbf{A}) = Th_r^{\text{MSO}}(\mathbf{B})$.

Composing Structures

Definition

- Fix signature $\sigma = (<, R_1, \dots, R_m)$ with order symbol $<$
- Write $\mathbf{A} \triangleleft \mathbf{B}$ for the structure with
 - domain $A \cup B$ (assume A, B disjoint),
 - relation $<$ is $<^A \cup <^B \cup (A \times B)$,
 - and R is $R^A \cup R^B$ for the other relation symbols R in the signature.

Example

- $\mathbf{S}_{ab} \triangleleft \mathbf{S}_{aab} = \mathbf{S}_{abaab}$.

Composing Theories

Composition Theorem

If $A \equiv_r^{\text{MSO}} A'$ and $B \equiv_r^{\text{MSO}} B'$, then $A \triangleleft B \equiv_r^{\text{MSO}} A' \triangleleft B'$.

Composing Theories

Composition Theorem

If $A \equiv_r^{\text{MSO}} A'$ and $B \equiv_r^{\text{MSO}} B'$, then $A \triangleleft B \equiv_r^{\text{MSO}} A' \triangleleft B'$.

Conclude:

- So, in the construction from MSO to DFA, take $Th_{gr(\varphi)}^{\text{MSO}}(\mathbf{S}_u)$ instead of $Th(u)$.
- This finishes the proof that $\text{REG} = \text{MSO} = \exists\text{MSO}$ on strings.

Proving the composition theorem

- To prove the composition theorem, it is enough to establish that if the Duplicator wins the r -round game on $(\mathbf{A}, \mathbf{A}')$ as well as $(\mathbf{B}, \mathbf{B}')$ then she also wins the r -round game on $(\mathbf{A} \triangleleft \mathbf{B}, \mathbf{A}' \triangleleft \mathbf{B}')$
- Intuitively, this is true by having Duplicator play the two games $(\mathbf{A}, \mathbf{A}')$ and $(\mathbf{B}, \mathbf{B}')$ on the side in order to know what to play in the game $(\mathbf{A} \triangleleft \mathbf{B}, \mathbf{A}' \triangleleft \mathbf{B}')$.
- In the next slides we provide some of the formalities.

Proofs

Definition

- $G_0(\mathbf{A}, \mathbf{B})$ if $\mathbf{A} \equiv_0 \mathbf{B}$.
- $G_{r+1}(\mathbf{A}, \mathbf{B})$ if the following two conditions hold:
 - forth: for every $A' \subseteq A$ there exists $B' \subseteq B$ such that $G_r((\mathbf{A}, A'), (\mathbf{B}, B'))$
 - back: for every $B' \subseteq B$ there exists $A' \subseteq A$, such that $G_r((\mathbf{A}, A'), (\mathbf{B}, B'))$.

Informally, in the $(r + 1)$ -round game, every move by Spoiler can be "matched" by Duplicator so that Duplicator can survive the remaining r -round game.

Proofs

Theorem

- If $G_r(\mathbf{A}, \mathbf{A}')$ and $G_r(\mathbf{B}, \mathbf{B}')$ then $G_r(\mathbf{A} \triangleleft \mathbf{B}, \mathbf{A}' \triangleleft \mathbf{B}')$

Proof

- Let $X \subseteq A \cup B$ (the other case is symmetric).
- There exists $Y_1 \subseteq A'$ such that $G_{r-1}((\mathbf{A}, X \cap A), (\mathbf{A}', Y_1))$
- There exists $Y_2 \subseteq B'$ such that $G_{r-1}((\mathbf{B}, X \cap B), (\mathbf{B}', Y_2))$
- By induction, $G_{r-1}((\mathbf{A} \triangleleft \mathbf{B}, X), (\mathbf{A}' \triangleleft \mathbf{B}', Y_1 \cup Y_2))$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Rightarrow

- Say $G_r(\mathbf{A}, \mathbf{B})$ and $qr(\varphi) = r$. We will show that $\mathbf{A} \models \varphi$ implies $\mathbf{B} \models \varphi$ (the other case is symmetric).

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Rightarrow

- Say $G_r(\mathbf{A}, \mathbf{B})$ and $qr(\varphi) = r$. We will show that $\mathbf{A} \models \varphi$ implies $\mathbf{B} \models \varphi$ (the other case is symmetric).
- It is enough to prove it for $\varphi = \exists X.\psi$ with $qr(\psi) = r - 1$ (why?)

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Rightarrow

- Say $G_r(\mathbf{A}, \mathbf{B})$ and $qr(\varphi) = r$. We will show that $\mathbf{A} \models \varphi$ implies $\mathbf{B} \models \varphi$ (the other case is symmetric).
- It is enough to prove it for $\varphi = \exists X.\psi$ with $qr(\psi) = r - 1$ (why?)
- Since $\mathbf{A} \models \varphi$, there is some A_1 such that $\mathbf{A} \models \psi(A_1)$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Rightarrow

- Say $G_r(\mathbf{A}, \mathbf{B})$ and $qr(\varphi) = r$. We will show that $\mathbf{A} \models \varphi$ implies $\mathbf{B} \models \varphi$ (the other case is symmetric).
- It is enough to prove it for $\varphi = \exists X.\psi$ with $qr(\psi) = r - 1$ (why?)
- Since $\mathbf{A} \models \varphi$, there is some A_1 such that $\mathbf{A} \models \psi(A_1)$.
- Matching the choice of $A_1 \subseteq A$, there exists $B_1 \subseteq B$ such that $G_{r-1}((\mathbf{A}, A_1), (\mathbf{B}, B_1))$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Rightarrow

- Say $G_r(\mathbf{A}, \mathbf{B})$ and $qr(\varphi) = r$. We will show that $\mathbf{A} \models \varphi$ implies $\mathbf{B} \models \varphi$ (the other case is symmetric).
- It is enough to prove it for $\varphi = \exists X.\psi$ with $qr(\psi) = r - 1$ (why?)
- Since $\mathbf{A} \models \varphi$, there is some A_1 such that $\mathbf{A} \models \psi(A_1)$.
- Matching the choice of $A_1 \subseteq A$, there exists $B_1 \subseteq B$ such that $G_{r-1}((\mathbf{A}, A_1), (\mathbf{B}, B_1))$.
- By induction, $(\mathbf{A}, A_1) \equiv_{r-1} (\mathbf{B}, B_1)$. Thus they agree on ψ .

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.
- By induction, this is the same as $(\mathbf{A}, A') \equiv_{r-1} (\mathbf{B}, B')$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.
- By induction, this is the same as $(\mathbf{A}, A') \equiv_{r-1} (\mathbf{B}, B')$.
- Let $\Psi(X') = \bigwedge Th_{r-1}((\mathbf{A}, A'))[A'/X']$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.
- By induction, this is the same as $(\mathbf{A}, A') \equiv_{r-1} (\mathbf{B}, B')$.
- Let $\Psi(X') = \bigwedge Th_{r-1}((\mathbf{A}, A'))[A'/X']$.
- Then $qr(\Psi) = r$ and $\mathbf{A} \models \exists X' \Psi(X')$, so $\mathbf{B} \models \exists X' \Psi(X')$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.
- By induction, this is the same as $(\mathbf{A}, A') \equiv_{r-1} (\mathbf{B}, B')$.
- Let $\Psi(X') = \bigwedge Th_{r-1}((\mathbf{A}, A'))[A'/X']$.
- Then $qr(\Psi) = r$ and $\mathbf{A} \models \exists X' \Psi(X')$, so $\mathbf{B} \models \exists X' \Psi(X')$.
- Say $\mathbf{B} \models \Psi(B')$ for some $B' \subseteq B$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.
- By induction, this is the same as $(\mathbf{A}, A') \equiv_{r-1} (\mathbf{B}, B')$.
- Let $\Psi(X') = \bigwedge Th_{r-1}((\mathbf{A}, A'))[A'/X']$.
- Then $qr(\Psi) = r$ and $\mathbf{A} \models \exists X' \Psi(X')$, so $\mathbf{B} \models \exists X' \Psi(X')$.
- Say $\mathbf{B} \models \Psi(B')$ for some $B' \subseteq B$.
- So $(\mathbf{B}, B') \models Th_{r-1}(\mathbf{A}, A')$.

Proofs

Theorem $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$.

Note: the case $r = 0$ is by definition.

Proof of \Leftarrow

- Say $\mathbf{A} \equiv_r \mathbf{B}$.
- Let $A' \subseteq A$ (the other case is symmetric).
- We must find $B' \subseteq B$ that "matches" A' for Duplicator to survive the $(r - 1)$ -game, i.e., $G_{r-1}((\mathbf{A}, A'), (\mathbf{B}, B'))$.
- By induction, this is the same as $(\mathbf{A}, A') \equiv_{r-1} (\mathbf{B}, B')$.
- Let $\Psi(X') = \bigwedge Th_{r-1}((\mathbf{A}, A'))[A'/X']$.
- Then $qr(\Psi) = r$ and $\mathbf{A} \models \exists X' \Psi(X')$, so $\mathbf{B} \models \exists X' \Psi(X')$.
- Say $\mathbf{B} \models \Psi(B')$ for some $B' \subseteq B$.
- So $(\mathbf{B}, B') \models Th_{r-1}(\mathbf{A}, A')$.
- So $Th_{r-1}(\mathbf{B}, B') = Th_{r-1}(\mathbf{A}, A')$, i.e., $(\mathbf{B}, B') \equiv_{r-1} (\mathbf{A}, A')$.

What have we done?

We defined a notion $G_r(\mathbf{A}, \mathbf{B})$ and proved:

1. If $G_r(\mathbf{A}, \mathbf{A}')$ and $G_r(\mathbf{B}, \mathbf{B}')$ then $G_r(\mathbf{A} \triangleleft \mathbf{B}, \mathbf{A}' \triangleleft \mathbf{B}')$
2. $G_r(\mathbf{A}, \mathbf{B})$ iff $\mathbf{A} \equiv_r \mathbf{B}$ (Ehrenfeucht-Fraïssé Theorem)

And so conclude:

3. If $\mathbf{A} \equiv_r \mathbf{A}'$ and $\mathbf{B} \equiv_r \mathbf{B}'$ then $\mathbf{A} \triangleleft \mathbf{B} \equiv_r \mathbf{A}' \triangleleft \mathbf{B}'$
(Composition Theorem)

Consequences of Ehrenfeucht-Fraïssé games

Logics capturing models of computation

- $\text{REG} = \text{MSO}$ on strings
- there is an analogous characterisation of FO on strings (star-free regular sets)

Consequences of Ehrenfeucht-Fraïssé games

Logics capturing models of computation

- $\text{REG} = \text{MSO}$ on strings
- there is an analogous characterisation of FO on strings (star-free regular sets)

Decidability

- MSO-satisfiability on finite strings is decidable.

Consequences of Ehrenfeucht-Fraïssé games


Logics capturing models of computation

- REG = MSO on strings
- there is an analogous characterisation of FO on strings (star-free regular sets)

Decidability

- MSO-satisfiability on finite strings is decidable.

Non-definability


- The set $\{\mathbf{S}_w : \#_a(w) = \#_b(w)\}$ is not MSO-definable.
-  The following classes of graphs are not MSO-definable:
 - graphs with a perfect-matching
 - graphs with a Hamiltonian cycle
- Connectivity is not \exists MSO definable. (Hard)

Takeaway

Theorem If $A \equiv_r A'$ and $B \equiv_r B'$ then $A \triangleleft B \equiv_r A' \triangleleft B'$

In general, a "composition theorem" says that the theory of a composed structure is determined by, and computable from, the theories of its parts.

- Feferman-Vaught for FO and generalised products
- Lauchli-Shelah-Gurevich for MSO and generalised sums
- Good read: "Algorithmic uses of the Feferman-Vaught Theorem" by J.A. Makowsky

 Show that MSO has no composition theorem for cartesian-product.

What is DCT?

- Overview of the course

- Logic and Definability

\exists MSO captures REG on strings

- Automata primer

- An excursion into finite model theory

Fagin's Theorem: \exists SO captures NPTIME

- Computational complexity crash course

- Computational complexity of logic problems

- Sketch proof of Fagin's Theorem



Framework

Fix a signature σ .

- A **property** P is a set of σ -structures.
- For a logic \mathcal{L} , a property P is **\mathcal{L} -definable** if $P = \text{Mod}(\varphi)$ for some $\varphi \in \mathcal{L}$.
- For a complexity class \mathcal{C} , a property P is **\mathcal{C} -solvable** if the complexity of deciding "does \mathbf{A} have property P " is in \mathcal{C} .

Example

- The property "the set of all 3-colourable graphs" is $\exists\text{SO}$ -definable and NPTIME -solvable.



Definition

- \mathcal{L} **captures** \mathcal{C} if every property is \mathcal{L} -definable iff it is \mathcal{C} -solvable.

Fagin's Theorem: $\exists\text{SO}$ captures NPTIME

Computational complexity crash course

Computational complexity

Algorithms (Actually, Turing Machines)

- Deterministic
- Nondeterministic ("guess" operator)

Some complexity classes

- PTIME, PSPACE, EXPTIME, EXPSPACE
- NPTIME, NPSPACE

Deterministic case

Algorithms

- an input induces a single execution.
- algorithm accepts the input if this execution is successful (i.e., outputs "Yes").

Running time

- The running time of Alg is the function $t : \mathbb{N} \rightarrow \mathbb{N}$ where $t(n)$ is the largest number of steps used by Alg on the execution of any input of length n .
- $\text{DTIME}(t(n))$ is the set of problems solvable by deterministic algorithms that run in time $O(t(n))$.

Complexity classes

- $\text{PTIME} = \bigcup_k \text{DTIME}(n^k)$
- $\text{EXPTIME} = \bigcup_k \text{DTIME}(2^{n^k})$

Nondeterministic case

Algorithms

- an input may induce more than one execution.
- algorithm accepts the input if **at least one** execution is successful.

Running time

- The running time of Alg is the function $t : \mathbb{N} \rightarrow \mathbb{N}$ where $t(n)$ is the largest number of steps used by Alg **on any execution of any input of length n .**
- $\text{NTIME}(t(n))$ is the set of problems solvable by non-deterministic algorithms that run in time $O(t(n))$.

Complexity classes

- $\text{NPTIME} = \bigcup_k \text{NTIME}(n^k)$

Computational complexity of logic problems

Two algorithmic problems for logic

Satisfiability problem

- In: formula $\varphi \in \mathcal{L}$
- Out: "Yes" if there exists $\mathbf{A} \models \varphi$, and "No" otherwise.

(Recall that φ is valid iff $\neg\varphi$ is not satisfiable.)

Model-checking problem

- In: formula $\varphi \in \mathcal{L}$, structure \mathbf{A}
- Out: "Yes" if $\mathbf{A} \models \varphi$, and "No" otherwise.

Comments about satisfiability

For arbitrary structures:

- Validity of FO is computably enumerable, but not computable.
- Satisfiability of FO is not computably enumerable.

When restricted to finite structures:

- Satisfiability of first-order logic is computably enumerable, but not computable (Trakhtenbrot).
- In particular, there is no sound and complete effective proof system for FO on finite structures.


Model-checking vs Satisfiability

In FMT, model-checking plays a more central role than satisfiability. Why?

- A property is \mathcal{C} -solvable iff it is \mathcal{L} -definable.
- The \Leftarrow direction talks about model-checking not satisfiability.


Model-checking vs Satisfiability

In FMT, model-checking plays a more central role than satisfiability. Why?

- A property is \mathcal{C} -solvable iff it is \mathcal{L} -definable.
- The \Leftarrow direction talks about model-checking not satisfiability.
-  Does it say that the complexity of model-checking formulas from \mathcal{L} is in \mathcal{C} ?

Model-checking vs Satisfiability

In FMT, model-checking plays a more central role than satisfiability. Why?

- A property is \mathcal{C} -solvable iff it is \mathcal{L} -definable.
- The \Leftarrow direction talks about model-checking not satisfiability.
-  Does it say that the complexity of model-checking formulas from \mathcal{L} is in \mathcal{C} ?


Not quite.

- It says that the complexity measured in terms of the size structure (not the formula) is in \mathcal{C} .
- This is called the **structure complexity** of model-checking \mathcal{C} .

Algorithm for model-checking FO

Theorem The structure-complexity of model-checking FO is in PTIME.

- Cycle through all possible instantiations in \mathbf{A} of the quantifiers of φ .
- E.g., $(\forall x)(\exists y)E(x, y)$ has a double-loop that scans through the pairs $(a, b) \in A^2$ and checks $E(a, b)$.
- This is a deterministic polynomial time algorithm (where the exponent depends on the formula, which is assumed to be fixed).

 Show that the structure complexity can be solved in time $O(|\varphi| \times |A|^w)$ where w is maximum number of free variables in subformulas of φ .

Algorithm for model-checking \exists SO

Nondeterministic algorithm

- For \exists SO formula $\Phi := \exists Z_1 \exists Z_2 \cdots \exists Z_m \varphi$,
- For a structure \mathbf{A} ,
- Guess $A_1 \subseteq A^{r_1}, \dots, A_m \subseteq A^{r_m}$,
- Check if $(\mathbf{A}, A_1, \dots, A_m) \models \varphi$ using the deterministic algorithm before.

Algorithm for model-checking \exists SO

Nondeterministic algorithm

- For \exists SO formula $\Phi := \exists Z_1 \exists Z_2 \cdots \exists Z_m \varphi$,
- For a structure \mathbf{A} ,
- Guess $A_1 \subseteq A^{r_1}, \dots, A_m \subseteq A^{r_m}$,
- Check if $(\mathbf{A}, A_1, \dots, A_m) \models \varphi$ using the deterministic algorithm before.

Complexity for fixed formula Φ .

- Guessing takes time polynomial in $|A|$, i.e., $O(m \times |A|^{\max r_i})$
- Checking takes time polynomial in $|A|$
- So the structure-complexity of model-checking \exists SO is in NPTIME

Sketch proof of Fagin's Theorem

Fagin's Theorem

Fix a signature σ .

Theorem \exists SO captures NPTIME

1. Structure complexity of model-checking \exists SO is in NPTIME
2. Every set of σ -structures computable for which membership is NPTIME is \exists SO-definable.


Hang on, how do Turing machines work on structures?

We need to decide how to encode σ -structures \mathbf{A} as strings, so we can give them as input to Turing machines. Let's do graphs (V, E) .

- Fix an ordering $v_1 < v_2 < \dots < v_n$ of the vertices.
- This induces the lexicographic ordering on pairs of vertices:
 $(v, w) < (v', w')$ if $v < v'$, or $v = v'$ and $w < w'$.
- i.e., the first pair is (v_1, v_1) , the second is (v_2, v_1) , the third is (v_3, v_1) , until (v_n, v_1) , and then (v_1, v_2) , etc.
- Code the graph (V, E) as the string

$$0^n 1 \cdot enc(E)$$

where $enc(E)$ is the bit-string of length n^2 with a 1 in position i iff the lexicographically i th tuple (v, w) is in E .

-  What are the encodings of $(\{a, b, c\}, \{(a, b), (b, c)\})$?

NPTIME problems are \exists SO-definable

Given

- a set P of σ -structures
- a nondeterministic Turing machine M running in time n^k deciding if a given encoding of a given σ -structure is in P ,

there exists an \exists SO formula φ_M such that M accepts an encoding of \mathbf{A} iff $\mathbf{A} \models \varphi_M$.

NPTIME problems are \exists SO-definable

Recall (from week 1) that satisfiability of FO in arbitrary structures is undecidable.

- Given TM M , it built FO-formula φ_M such that M halts on empty input iff φ_M is satisfiable.
- The signature of φ_M included a binary symbol $<$ satisfying the axioms of the order on the natural numbers.
- The order allowed one to talk about time t and space p . Then FO can axiomatise the following relations:
 - $\text{State}_q(t)$: at time t , machine state is q
 - $\text{Pos}(t, p)$: at time t , machine head is in position p on the tape
 - $\text{Tape}_a(t, p)$: at time t , symbol a is written at position p of the tape

and express the machine halts.

NPTIME problems are \exists SO-definable

The idea is similar except that now the formula φ_M is to be interpreted on a given structure \mathbf{A} .

So, the formula has to also express that:

- the input to the TM is the encoding of \mathbf{A} ,
- there is an ordering that can code a polynomial number of steps/cells.

It is boring, but one can adapt φ_M so that:

- M accepts an encoding of \mathbf{A} iff $\mathbf{A} \models \varphi_M$

💡 The main insight is that one can use \exists SO to express the existence of an order L on A , and then the formula can use the lexicographic order on k -tuples of A to count up to $|A|^k$, and thus has access to all the time/space needed by a polynomial-time machine.

Consequences of Fagin's theorem

Corollaries:

- $\text{coNPTIME} = \forall\text{SO}$
- So: $\text{NPTIME} \neq \text{coNPTIME}$ iff $\exists\text{SO} \neq \forall\text{SO}$
- Showing $\exists\text{SO} \neq \forall\text{SO}$ would also imply $\text{NPTIME} \neq \text{PTIME}$.
- We do know $\exists\text{MSO} \neq \forall\text{MSO}$.

Time to wrap up

What have we seen in this course?

- Logic for defining properties of finite strings, graphs, etc.
- Algorithms for deciding if a given structures satisfies a fixed formula, e.g., PTIME for FO, NPTIME for \exists SO, automata for MSO on strings.
- Connections between describing and solving.
- Tools from finite model theory: Ehrenfeucht-Fraïssé games, Composition theorem
- Undefinability results (property X is not definable in logic \mathcal{L})



If you remember one thing from this course

There is a tight connection between

- computational resources needed for solving problems, and
- logical resources needed for describing problems.

References

- Phokion Kolaitis' ESSLLI slides on FMT
- Erich Grädel's chapter "Finite model theory and descriptive complexity theory".
- Leonid Libkin's book "Elements of finite model theory"