The Finite Model Theory Toolbox of a Database Theoretician

Leonid Libkin
University of Edinburgh

PODS 2009
Finite Model Theory (FMT)

- The main object of study: logics over finite structures
- Foundational role in the study of the theory of relational databases:
  - relational database = finite relational structure
  - query languages are logic based
- Finite model theory was described as the "backbone of database theory" (Vianu, 1995)
- Connections work both ways: much of the motivation for finite model theory research came from databases.
- Many other applications in CS: verification, AI, constraint satisfaction, algorithms, complexity ...
Finite Model Theory (FMT)

- Mature field; 4 textbooks:
  1. Ebbinghaus-Flum "Finite Model Theory", 1994
  2. Immerman "Descriptive Complexity", 1999
  3. Libkin "Elements of ‘Finite Model Theory’, 2004

- Several ‘personal perspective’ surveys:
  1. Fagin (ICDT 1990, LICS 2000)
  2. Kolaitis (LICS 2007)

- Well-developed subfields:
  1. Combinatorial games (Kolaitis PODS 1995)
  2. Logic for PTIME (Kolaitis ICDT 1995, Grohe LICS 2008)
  3. Descriptive complexity, algorithmic model theory, embedded finite models, etc etc

- Surveys they tell you how to prove results, what the main open problems are, etc.
FMT for a database theoretician: Key problems

1. **Expressiveness** of query languages
   - Limited expressiveness (e.g., relational calculus = first-order logic)
   - What is not expressible? When to add new language constructs?
   - Adding new constructs is not free — optimizations!

2. **Complexity** of query languages
   - Do we know the complexity of query evaluation from the logical formalism?

3. **Equivalence** of query languages
   - Can we lower the complexity by changing the syntax?
   - Important in the study of languages for XML.

4. **Satisfiability** (usually, finite satisfiability)
   - Used in: static analysis, incomplete information.
FMT for a database theoretician cont’d

- Many database results involving FMT techniques were shown by people actively working in FMT.
- This was perhaps necessary in the early days of FMT.
- Now the field has built a large arsenal of tools.
- These tools can be used without knowing how they are proved!
- Most of them are actually quite easy to apply.
- They should be a part of the toolbox of every database theoretician.
- Our goal: present finite-model theory as such a toolbox.
Plan

- Expressiveness: Tools for first-order logic
- Expressiveness: Tools that work beyond first-order logic
- Language equivalence (better query evaluation via the composition method)
- Descriptive Complexity
- Satisfiability
Plan

• Expressiveness: Tools for first-order logic
• Expressiveness: Tools that work beyond first-order logic
• Language equivalence (better query evaluation via the composition method)
• Descriptive Complexity
• Satisfiability
First-Order (FO)

- The most fundamental query language — first-order logic (FO)
- Database people often refer to it as relational calculus.
- The core of SQL (minus aggregation – we’ll address it later).
- A basic question: what are the limitations of FO?
- Intuition: FO cannot express:
  1. nontrivial counting properties, and
  2. queries requiring recursion
- We’ll now see some “canonical” examples of inexpressible queries.
Even cardinality

- **Active domain** = set of all elements stored in a database
- A Boolean query

\[
\text{EVEN}(D) = \text{true} \iff |\text{ActiveDomain}(D)| = 0 \pmod{2}
\]
Transitive closure
Transitive closure

\[
\text{trcl}(x, y) :\quad e(x, y)
\]

\[
\text{trcl}(x, y) :\quad e(x, z), \text{trcl}(z, y)
\]
Same Generation
Same Generation

\[ sg(x, x) \ := \ \]  
\[ sg(x, y) \ := \ e(x', x), e(y', y), sg(x', y') \]
Same Generation

\[
\begin{align*}
\text{sg}(x, x) & : - \\
\text{sg}(x, y) & : - \ e(x', x), e(y', y), \text{sg}(x', y')
\end{align*}
\]
A bit of history

- How to prove that these queries are not expressible in FO?
- Classical model theory offers us powerful tools, like compactness.
- They can be used to prove that {\textit{Eve}n} is not FO-definable (also shown in the paper).
- They can also be used to prove that \textit{graph connectivity} (and hence transitive closure) are not definable over \textit{arbitrary} graphs.
- But the proof does not work for \textit{finite} graphs: compactness fails in the finite.
- However, databases are finite!
A bit of history cont’d

- Fagin 1975: Transitive closure is not FO-expressible over finite graphs. Technique: *games*.
- Afterwards (1970s, 1980s): more and more advanced game proofs.
- Require nontrivial combinatorial arguments.
- A notable exception: 0-1 laws (Fagin 1976) – an easily applicable tool.
- 1990s: proper tools are being developed. They do not require complicated combinatorial proofs.
First-Order Logic (FO)

- **Assumption**: databases are graphs – one binary relation $E(\cdot, \cdot)$.
  
  Just to make things simple for the talk.

- **Syntax of FO**:
  
  - Atomic formulae: $E(x, y)$, $x = y$
  - Boolean combinations: $\varphi \lor \psi$, $\varphi \land \psi$, $\neg \varphi$
  - Quantification: $\exists x \, \varphi$, $\forall x \, \varphi$
  
  - $\varphi(\bar{x})$ means that $\bar{x}$ is the tuple of free variables of $\varphi$

- **Quantifier rank $qr(\varphi)$**:
  
  - The depth of quantifier nesting in $\varphi$.
  - Example: the quantifier rank of $\exists x \left( \forall y \, E(x, y) \lor \forall z \, \neg E(x, z) \right)$ is 2 and not 3.
First-Order Logic – Examples

• There are at least $k$ elements

$$\exists x_1 \ldots \exists x_k \bigwedge_{i,j \leq k, \ i \neq j} \neg (x_i = x_j)$$

• There is a path of length $k$ from $x_0$ to $x_k$

$$\exists x_1 \ldots \exists x_{k-1} \bigwedge_{0 \leq i < k} E(x_i, x_{i+1})$$

• There is no cycle of length $k$

$$\neg \exists x_1 \ldots \exists x_k \ E(x_k, x_1) \land \bigwedge_{i < k} E(x_i, x_{i+1})$$
First-Order Logic – Examples

- There are at least $k$ elements

  The number of elements is even – inexpressible.

- There is a path of length $k$ from $x_0$ to $x_k$

  There is a path from $x_0$ to $x_k$ – inexpressible.

- There is no cycle of length $k$

  There is no cycle – inexpressible.
Ehrenfeucht-Fraïssé games

- The tool of choice for the neolithic period of FMT.
- Played on two databases (graphs) $\mathcal{A}$ and $\mathcal{B}$.
- Two players:
  - **Spoiler** (the bad guy): tries to show that $\mathcal{A}$ and $\mathcal{B}$ are different.
  - **Duplicator** (the good guy): tries to show that $\mathcal{A}$ and $\mathcal{B}$ are the same.
- Play for $k$ rounds.
- In each round:
  - the spoiler moves first: selects a database and an element there.
  - the duplicator responds by an element in the other database.
- The duplicator wins if at the end, the played elements form a **partial isomorphism** between $\mathcal{A}$ and $\mathcal{B}$.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

 Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

**A**

**B**

*Spoiler* and *Duplicator* play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.
Ehrenfeucht-Fraïssé game - example 1

Spoiler and Duplicator play for 3 rounds.

The duplicator wins in 3 rounds.
Ehrenfeucht-Fraïssé game - example 2

\[ A \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \rightarrow \right}
Ehrenfeucht-Fraïssé game - example 2

\[ A \quad B \]

\[ V \quad V \]
Ehrenfeucht-Fraïssé game - example 2

A

B

Leonid Libkin
Ehrenfeucht-Fraïssé game - example 2

\[ \mathcal{A} \quad \begin{array}{c}
\text{A}
\end{array} \quad \begin{array}{c}
\vdash
\end{array} \quad \begin{array}{c}
\vdash
\end{array} \quad \begin{array}{c}
\vdash
\end{array} \quad \begin{array}{c}
\vdash
\end{array} \quad \begin{array}{c}
\vdash
\end{array}
\]

\[ \mathcal{B} \quad \begin{array}{c}
\text{B}
\end{array} \quad \begin{array}{c}
\vdash
\end{array} \quad \begin{array}{c}
\vdash
\end{array} \quad \begin{array}{c}
\vdash
\end{array}
\]
Ehrenfeucht-Fraïssé game - example 2

A

B
Ehrenfeucht-Fraïssé game - example 2

\[
\begin{align*}
&\mathcal{A} \quad \bullet \quad \downarrow \quad \downarrow \quad \bullet \quad \downarrow \quad \downarrow \quad \bullet \quad \downarrow \quad \downarrow \\
&\mathcal{B} \quad \bullet \quad \downarrow \quad \downarrow \quad \bullet \quad \downarrow \quad \downarrow
\end{align*}
\]
Ehrenfeucht-Fraïssé game - example 2

A

B
Ehrenfeucht-Fraïssé game - example 2
Ehrenfeucht-Fraïssé game - example 2

The spoiler wins in 3 rounds.
Ehrenfeucht-Fraïssé games and FO

The duplicator has a winning strategy in the $k$-round Ehrenfeucht-Fraïssé game if he can win in $k$ rounds no matter how the spoiler plays.

**Theorem**

The duplicator has a winning strategy in the $k$-round Ehrenfeucht-Fraïssé game on $\mathcal{A}$ and $\mathcal{B}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ cannot be distinguished by FO sentences of quantifier rank up to $k$. 
How to prove that a property \( \mathcal{P} \) is not expressible in FO?

Find families of graphs \( \mathcal{A}_k \) and \( \mathcal{B}_k \), for \( k \in \mathbb{N} \) so that:

1. All \( \mathcal{A}_k \) have property \( \mathcal{P} \);
2. None of \( \mathcal{B}_k \) has property \( \mathcal{P} \);
3. The duplicator has a winning strategy in the \( k \)-round Ehrenfeucht-Fraïssé game on \( \mathcal{A}_k \) and \( \mathcal{B}_k \)

If \( \mathcal{P} \) were expressible by a sentence \( \varphi \) of quantifier rank \( k \),

- \( \mathcal{A}_k \) and \( \mathcal{B}_k \) must agree on \( \varphi \) — by 3,
- but by 1 and 2 they disagree on \( \varphi \).
A surprisingly powerful example

Let $L_n$ be a linear ordering of length $n$.

**Theorem**

*If $m, n \geq 2^k$, then the duplicator has a winning strategy in the $k$-round Ehrenfeucht-Fraïssé game on $L_n$ and $L_m$.***
A surprisingly powerful example

Let $L_n$ be a linear ordering of length $n$.

**Theorem**

If $m, n \geq 2^k$, then the duplicator has a winning strategy in the $k$-round Ehrenfeucht-Fraïssé game on $L_n$ and $L_m$.

**Corollary**

Query **Even** is not expressible over linear orderings.

Because: take $\mathcal{A}_k$ to be $L_{2^k+1}$ and $\mathcal{B}_k$ to be $L_{2^k}$. 
The early 1980s: the era of tricks

The result about \textbf{EVEN} shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.

Edges: to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: *Graph connectivity; graph acyclicity; transitive closure.*

**Edges:** to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about \textbf{EVEN} shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.

\textbf{Edges:} to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about \textbf{EVEN} shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.

\textbf{Edges:} to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: **Graph connectivity; graph acyclicity; transitive closure.**

**Edges:** to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: **Graph connectivity; graph acyclicity; transitive closure.**

Edges: to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: **Graph connectivity; graph acyclicity; transitive closure.**

**Edges:** to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: **Graph connectivity; graph acyclicity; transitive closure.**

**Edges:** to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about **EVEN** shows that none of the following is definable in FO: **Graph connectivity; graph acyclicity; transitive closure.**

**Edges:** to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about \textbf{EVEN} shows that none of the following is definable in FO: \textit{Graph connectivity; graph acyclicity; transitive closure.}

\textbf{Edges:} to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about \textbf{EVEN} shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.

\textbf{Edges:} to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about \textbf{EVEN} shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.

Edges: to the 2nd successor, modulo the length of the chain.
The early 1980s: the era of tricks

The result about \texttt{EVEN} shows that none of the following is definable in FO: Graph connectivity; graph acyclicity; transitive closure.

\begin{figure}
\centering
\begin{tikzpicture}
\node at (0,0) {\texttt{EVEN}};
\node at (2,0) {$\Rightarrow$};
\end{tikzpicture}
\end{figure}

Connected if \texttt{EVEN} is false; disconnected if \texttt{EVEN} is true.
The era of tricks cont’d

**Acyclicity** is not FO-expressible:

- a similar trick – with one backedge instead of two.

**Transitive closure** is not FO-expressible:

- the symmetric-transitive closure of $G$ is a complete graph iff $G$ is connected.
The 1990s: the era of tools

- The iron age of FMT.
- For more complicated problems, stone (pebble) tools – games – become very hard to use.
- More and more complicated winning conditions are used:
  - Fagin, Ajtai, Vardi, Stockmeyer, Schwentick, Kolaitis, Väänänen, etc
- Fagin, Stockmeyer, Vardi, 1993: Let’s build a library of winning strategies for the duplicator.
- Key idea: locality.
  - Already present in earlier work by Gaifman 1980 and Hanf 1965.
Transitive closure revisited

Degrees of nodes: 0, 1
Transitive closure revisited

Degrees of nodes: $0, 1, \ldots, n$ – depends on the input.
Transitive closure revisited

Degrees of nodes: \(0, 1, \ldots, n\) – depends on the input.

This cannot happen for FO queries!
A useful property: BNDP

- A query $Q$ from graphs to graphs has the **Bounded Number of Degrees Property** if there is a function $f_Q : \mathbb{N} \rightarrow \mathbb{N}$ such that:

  all degrees in $G$ are bounded by $k$

  \[ \Downarrow \]

  the number of different degrees in $Q(G)$ is at most $f_Q(k)$

- We’ve just seen that transitive closure violates the BNDP.
A useful property: BNDP

- A query $Q$ from graphs to graphs has the **Bounded Number of Degrees Property** if there is a function $f_Q : \mathbb{N} \rightarrow \mathbb{N}$ such that:
  
  all degrees in $G$ are bounded by $k$
  
  $\downarrow$
  
  the number of different degrees in $Q(G)$ is at most $f_Q(k)$

- We’ve just seen that transitive closure violates the BNDP.

  **Theorem (Dong, L., Wong’95, L. ’97)**

  *Every FO query has the BNDP.*

- **Corollary:** transitive closure is not FO-definable.
Another application of BNDP – Same Generation

Degrees: 0, 1, 2
Another application of BNDP – Same Generation
Another application of BNDP – Same Generation

Degrees: $1, 2, 4, 8, \ldots, 2^{\text{depth}(G)-1}$

Number of degrees: $\text{depth}(G)$
Another application of BNDP – Same Generation

Violates the BNDP — Hence same-generation is not FO-definable
What makes the BNDP work?

- Locality of FO.
- There are two tools based on locality:
  1. Gaifman-locality (Gaifman 1982)
- Key concept: neighborhood.
- A neighborhood of radius $r$ of $\bar{a}$ in a graph $G$ is denoted by $N_r^G(\bar{a})$.
- It is the subgraph induced by all the nodes of distance $\leq r$ from one of the nodes in $\bar{a}$.
- Nodes $\bar{a}$ are distinguished:
  - if we have an isomorphism $h : N_r^G(a_1, \ldots, a_n) \to N_r^{G'}(b_1, \ldots, b_n)$ then $h(a_1) = b_1$, $\ldots$, $h(a_n) = b_n$. 
Gaifman-locality

Theorem (Gaifman 1982, bound from L., '98)

For every FO formula \( \varphi(\bar{x}) \) of quantifier rank \( k \) and for every graph \( G \):

\[
\mathcal{N}_{2k}^G(\bar{a}) \text{ and } \mathcal{N}_{2k}^G(\bar{b}) \text{ are isomorphic } \Rightarrow \ G \models \varphi(\bar{a}) \leftrightarrow \varphi(\bar{b})
\]
Gaifman-locality

**Theorem (Gaifman 1982, bound from L., '98)**

For every FO formula $\varphi(\bar{x})$ of quantifier rank $k$ and for every graph $G$:

$$N^G_{2k}(\bar{a}) \text{ and } N^G_{2k}(\bar{b}) \text{ are isomorphic } \Rightarrow G \models \varphi(\bar{a}) \leftrightarrow \varphi(\bar{b})$$

**Application:** Transitive closure is not definable in FO.
Gaifman-locality

**Theorem (Gaifman 1982, bound from L., ’98)**

For every FO formula $\varphi(\bar{x})$ of quantifier rank $k$ and for every graph $G$:

$$N_{2^k}^G(\bar{a}) \text{ and } N_{2^k}^G(\bar{b}) \text{ are isomorphic} \Rightarrow G \models \varphi(\bar{a}) \leftrightarrow \varphi(\bar{b})$$

**Application**: Transitive closure is not definable in FO.

If $\varphi(x, y)$ is of quantifier rank $k$ and $r = 2^k$ then both $\varphi(a, b)$ and $\varphi(b, a)$ are true, or both are false.
Hanf-locality

- Write $G \leftrightarrow_r G'$ if there exists a bijection $f : G \rightarrow G'$ such that $N_r^G(a)$ and $N_r^{G'}(f(a))$ are isomorphic for every node $a$.
- Locally two graphs look the same, up to a bijection $f$. 
Hanf-locality

• Write $G \xleftrightarrow{r} G'$ if there exists a bijection $f : G \rightarrow G'$ such that $N_r^G(a)$ and $N_r^{G'}(f(a))$ are isomorphic for every node $a$.
• Locally two graphs look the same, up to a bijection $f$.

\begin{center}
\textbf{Theorem (Fagin, Stockmeyer, Vardi, '93, bound from L.'98)}
\end{center}

\textit{For every FO sentence $\varphi$ of quantifier rank $k$,}

\[ \text{if } G \xleftrightarrow{2k} G' \text{ then } G \models \varphi \iff G' \models \varphi \]

• Can be extended to arbitrary queries, but most often this notion is used for Boolean queries.
Hanf-locality: application

- If $m > 2r + 1$ then $G \equiv_r G'$ (all $r$-neighborhoods are the same).
- Hence no FO sentence $\varphi$ defines connectivity: as long as $m > 2^{qr(\varphi) + 1} + 1$, graphs $G$ and $G'$ cannot be distinguished by $\varphi$. 
Summary: locality notions

A query $Q$ is:

- **Hanf-local** if there is $r \geq 0$ so that $G \cong_r G'$ implies $Q(G) = Q(G')$.
  - for Boolean queries; a natural extension to non-Boolean queries exists.

- **Gaifman-local** if there is $r \geq 0$ such that if $N_r^G(\bar{a})$ and $N_r^G(\bar{b})$ are isomorphic, then $\bar{a} \in Q(G) \iff \bar{b} \in Q(G)$.

- has the **BNDP** if there is function $f_Q : \mathbb{N} \rightarrow \mathbb{N}$ so that for $G$ with all degrees $\leq k$, the number of different degrees in $Q(G)$ is $\leq f_Q(k)$.

---

**Theorem (L. '97)**

Hanf-local $\Rightarrow$ Gaifman-local $\Rightarrow$ BNDP.

---

**Corollary**

FO queries are Hanf-local, Gaifman-local, and have the BNDP.
Counting: towards 0-1 laws

- How to prove that nontrivial counting properties are not expressible?
- **Even**: roughly half of databases have the property, and half don’t.
- FO cannot exhibit such a behavior.
Counting: towards 0-1 laws

- How to prove that nontrivial counting properties are not expressible?
- **Even**: roughly half of databases have the property, and half don’t.
- FO cannot exhibit such a behavior.

- Pick a database “at random”.
- Check if it satisfies a property $\mathcal{P}$.
- What’s the probability of that?
- If $\mathcal{P}$ is FO-definable, it is 0 or 1: **0-1 law**.
- Need to formalize: ‘pick a database at random’.
Towards 0-1 laws

- For each $n$ look at graphs with nodes $1, \ldots, n$.
- For a property $\mathcal{P}$, let
  \[ \mu_n(\mathcal{P}) = \frac{|\{\text{graphs on } 1, \ldots, n \text{ that satisfy } \mathcal{P}\}|}{|\{\text{graphs on } 1, \ldots, n\}|} \]
- Proportion of graphs on $1, \ldots, n$ satisfy $\mathcal{P}$, or
- Probability that a randomly picked graph on $1, \ldots, n$ — with respect to the uniform distribution — satisfies $\mathcal{P}$.
- Asymptotic probabilities:
  \[ \mu(\mathcal{P}) = \lim_{n \to \infty} \mu_n(\mathcal{P}) \]
Asymptotic probabilities: examples

- $\mu(\text{EVEN})$ – does not exist: $\mu_n(\text{EVEN}) = \begin{cases} 
1, & \text{if } n \text{ is even} \\
0, & \text{if } n \text{ is odd}
\end{cases}$

- $\mu(\text{exists isolated node}) = 0$. $\exists x \forall y \neg E(x, y)$

- $\mu(\text{diameter } \leq 2) = 1$. $\forall x \forall y \exists z E(x, z) \land E(y, z)$

- $\mu(\text{graph is connected}) = 1$.

- Two sets $A$ and $B$ with $B \subseteq A$.
  - $\text{Parity}$ is true iff $|B|$ is even.
  - $\mu(\text{Parity}) = \frac{1}{2}$. 

$\text{Leonid Libkin}$

$\text{FMT toolbox}$
0-1 law

**Theorem (Fagin 1976)**

If \( \mathcal{P} \) is FO-definable, then \( \mu(\mathcal{P}) \) exists and equals 0 or 1.
Theorem (Fagin 1976)

If $\mathcal{P}$ is FO-definable, then $\mu(\mathcal{P})$ exists and equals 0 or 1.

- If you like truly beautiful proofs, this is the one for you!
- Immediate corollaries: **EVEN** and **PARITY** are not FO-definable.
- Warning: the result does not hold when we consider specific classes of structures.
- For example, 0-1 law fails over ordered graphs:
  - $\mu(\text{there is an edge between the first and the last element}) = \frac{1}{2}$. 
Plan

- Expressiveness: Tools for first-order logic
- **Expressiveness**: Tools that work beyond first-order logic
- Language equivalence (better query evaluation via the composition method)
- Descriptive Complexity
- Satisfiability
FO extensions

- **Ordering**: elements stored in a database are typically ordered, and order comparisons can be used in queries.
- **Counting and aggregation**: we all know it from SQL; a very common feature in database queries.
- **Fixed points**: for many years a popular topic in database research (Datalog). Now also part of SQL-3.
- **Interpreted operations**: e.g., arithmetic operations such as $x^2 + y \leq x \cdot z$ in queries.
Ordering on the domain

- Can transitive closure be expressed over ordered graphs? What about connectivity? acycliclicity? etc.
- We know that \texttt{Even} is not expressible.
- Queries such as transitive closure do not refer to ordering.
- Order-invariant queries: can use an ordering, but it does not matter which ordering is used.
- Order-invariant formulae over graphs: \( \varphi(\bar{x}) \) over \( E(\cdot, \cdot), < \) so that

\[
(G, <_1) \models \varphi(\bar{a}) \iff (G, <_2) \models \varphi(\bar{a})
\]

for every two orderings \( <_1 \) and \( <_2 \) on the nodes.
- Defines an order-invariant query \( Q_\varphi \):

\[
\bar{a} \in Q_\varphi(G) \iff (G, <) \models \varphi(\bar{a}) \text{ for some ordering } <
\]
Order-invariant queries

• A mysterious class:
  • only makes sense in the finite;
  • a non-r.e. class of queries;
  • locality techniques do not seem to help: with \(<\) everything is a neighborhood of radius 1.

• But quite remarkably:

\textbf{Theorem (Grohe, Schwentick, 2000)}

Order-invariant queries are Gaifman-local and have the BNDP.

• Corollary: Transitive closure, connectivity, etc are not expressible even with order.

• Hanf-locality for order-invariance: open (partial results by Niemistö).
Adding counting and aggregation to the language

- Standard SQL feature.
- Assume domain of 2 sorts:
  - usual database entries (graph nodes);
  - numbers (for examples, $\mathbb{Q}$).
- Add counting terms and operations:
  - $\#\bar{x}.\varphi$ – how many $\bar{x}$ satisfy $\varphi$.
  - $P_{\text{property}}(\cdot)$ testing the property of numbers.
- Examples:
  - $\exists x \ P_{\text{even}}(\#y.E(x,y))$ – there is a node of even degree.
  - Degree of $x$ is (degree of $y$)$^2$:
    $P_{n=m^2}(\#z.E(x,y), \#z.E(y,z))$
Adding counting and aggregation to the language

- aggregates and grouping by example: sum up all even degrees in a graph
  - in SQL:  
    ```sql
    SELECT SUM(R.C)
    FROM (SELECT E.A, COUNT(E.B) AS C
          FROM E
          GROUP BY E.A
          HAVING MOD(COUNT(E.B), 2) = 0) R
    ```
  - in logic:  
    $$\text{Aggr}_{\text{SUM}} \times (P_{\text{even}}(\#y.E(x, y)), \#y.E(x, y))$$
Adding counting and aggregation to the language

- aggregates and grouping by example: sum up all even degrees in a graph
  - in SQL:  
  ```
  SELECT SUM(R.C)
  FROM (SELECT E.A, COUNT(E.B) AS C
      FROM E
      GROUPBY E.A
      HAVING MOD(COUNT(E.B),2) = 0) R
  ```
  - in logic:  
  ```
  Aggr_{SUM} \times (P_{even}(\#y.E(x,y)), \#y.E(x,y))
  ```
- Formally: $F$ is an aggregate (e.g., SUM, COUNT...)
- $aggr_{term}(\bar{x}) = Aggr_{F}\bar{y} (\varphi(\bar{x},\bar{y}), t(\bar{x},\bar{y}))$
- Semantics:
  - Find all $\bar{y}_1, \ldots, \bar{y}_k$ so that $\varphi(\bar{x},\bar{y}_i)$ holds
  - Calculate $v_i = t(\bar{x},\bar{y}_i)$
  - $aggr_{term}(\bar{x})$ is $F(\{v_1, \ldots, v_k\})$
Expressiveness of aggregation

- Question: which arithmetic predicates and which aggregate functions to add?
- Let’s be generous: add them all.
- But still look at queries over graph nodes (e.g., transitive closure).

Theorem (Hella, L., Nurmonen, Wong’99, improved L.’01)

Queries expressed in the aggregate language with arbitrary arithmetic and aggregates are local: i.e., Hanf-local, Gaifman-local, and have the BNDP.

- In particular, the usual SQL (select-from-where-groupby-having) cannot express transitive closure.
Aggregation and order

- What if we have an order on graph nodes? Can we recover locality?
- No, even in a minimalistic setting:
  - Arithmetic: $<$, $+$, $\times$
  - Aggregation: SUM
- If such an aggregate language cannot express transitive closure over ordered graphs, then some complexity classes are separated:
  - $TC^0$ and NLOGSPACE
  - big open problem in complexity theory
Recursion and Datalog

• Have seen it already:
  • transitive closure:
    
    $trcl(x, y) \; :\; e(x, y)$
    $trcl(x, y) \; :\; e(x, z), trcl(z, y)$

  • same-generation:
    
    $sg(x, x) \; :\;$
    $sg(x, y) \; :\; e(x', x), e(y', y), sg(x', y')$

• Now available in the latest SQL standard: WITH RECURSIVE.
  • But without negation.
  • With negation, several semantics exist.
Datalog: expressive power

- Without negation, queries are monotone.
- Even with negation and inflationary semantics:

  \begin{center}
  \textbf{Theorem (Blass, Kozen, Gurevich, 1985)}
  \end{center}

  \textit{Datalog has the 0-1 law.}

- This is \textbf{without order}. What if order is added?
- Then Datalog (with negation) captures \textbf{PTIME}.
- To prove bounds, one needs to separate complexity classes again.
- But without order, it can be separated from \textbf{NP}: 3-colorability is not expressible in Datalog with negation (Dawar, ’98).
  - A useful result (recent application in the work on schema mappings)
Extensions: summary

First-Order:

• cannot do recursive (fixed-point) queries;

• cannot count; this continues to hold with the order on the domain.

Extensions:

• with Counting/aggregation:
  • cannot do fixed-point queries

• with fixed-points:
  • cannot count

• But only without ordering on the domain:
  • with ordering, bounds on the as hard as separating complexity classes
Complex constraints

- Graph nodes: numbers. Query “does a graph lie on a circle?”:
  \[ \exists r \exists a \exists b \ \forall x \forall y \ E(x, y) \rightarrow (x - a)^2 + (y - b)^2 = r^2 \]

- What is the power of such extensions? Can graph connectivity be expressed?
- Look at queries that talk about proper graph properties (formally, isomorphism types of graphs). Known as generic queries.
- The answer depends on the class of numbers and arithmetic operations.
- Graph connectivity is expressible over \( \mathbb{N} \) with arithmetic \( 0, 1, +, \times, < \) (easy) but is not expressible over \( \mathbb{R} \) with arithmetic (much harder; Benedikt, L., '98).
- Results come from the field of embedded FMT: finite structures (graphs) living inside infinite ones (e.g., the real ordered field).
  - Survey: Chap. 5 of “Finite Model Theory and its Applications”. 
Plan

- Expressiveness: Tools for first-order logic
- Expressiveness: Tools that work beyond first-order logic
- Language equivalence (better query evaluation via the composition method)
- Descriptive Complexity
- Satisfiability
Language equivalence: games come back

- The focus of FMT applications in databases switches in the 21st century from *inexpressibility results* to proving *language equivalence*.
- **Goal**: start with a benchmark of expressiveness, and find a language with good complexity of query evaluation.
- Usually in the context of data that comes with a nice structure.
- **XML**: labeled *trees* with some nodes carrying data values.

For the talk, use **words**:

- to keep pictures and notations simple;
- the paper deals with trees as well.
Language equivalence: games come back

For words/trees, benchmark expressiveness is typically MSO.

- **Monadic Second Order Logic** — adds quantification over sets to FO: $\exists X_1 \forall X_2 \ldots \varphi(X_1, X_2, \ldots)$ where $\varphi$ is FO.
- Same expressiveness as automata.
- But problematic complexity of query evaluation.

**Question**: is it possible to achieve better complexity simply by syntactic manipulations?
Language equivalence: games come back

For words/trees, benchmark expressiveness is typically **MSO**.

- **Monadic Second Order Logic** — adds quantification over sets to FO: \( \exists X_1 \forall X_2 \ldots \varphi(X_1, X_2, \ldots) \) where \( \varphi \) is FO.

- Same expressiveness as **automata**.

- But **problematic** complexity of query evaluation.

**Question**: is it possible to achieve better complexity simply by syntactic manipulations?

**Yes!** Key technique: **composing Ehrenfeucht-Fraïssé games**.

Such composition tells us how queries on substructures combine for evaluating queries on the whole structure.
Changing syntax to lower complexity: LTL

Syntax: \( \varphi := a \ (\in \Sigma) \mid \varphi \lor \varphi' \mid \neg \varphi \mid X\varphi \mid \varphi U \varphi' \)
Changing syntax to lower complexity: LTL

Syntax:

\[ \varphi := a \ (\in \Sigma) \mid \varphi \lor \varphi' \mid \neg \varphi \mid X \varphi \mid \varphi U \varphi' \]

Semantics:

\[ a, \ a \in \Sigma \]
Changing syntax to lower complexity: LTL

Syntax: \[ \varphi ::= a \ (\in \Sigma) \ | \ \varphi \lor \varphi' \ | \ \neg \varphi \ | \ X\varphi \ | \ \varphi U \varphi' \]

Semantics:
Changing syntax to lower complexity: LTL

Syntax: \[ \varphi ::= a \ (\in \Sigma) \mid \varphi \lor \varphi' \mid \neg \varphi \mid X\varphi \mid \varphi U \varphi' \]

Semantics:
Changing syntax to lower complexity

FO or MSO evaluation over trees and words with linear data complexity implies non-elementary query complexity.

Need another – but equivalent – logic!

Over words, $LTL = FO$ (Kamp, 1969)

What is the query complexity of evaluating FO and LTL over words? With linear data complexity, it is:

- non-elementary for FO (a stack of exponentials, Frick, Grohe, ’03)
- linear for LTL
Words as databases

A word $w$ over $\Sigma = \{a_1, \ldots, a_m\}$ is a database with relations $E(\cdot, \cdot)$, $L_1(\cdot), \ldots, L_m(\cdot)$:

- $E$ is the ordering of positions;
- $L_i$’s define labelings.

$w = a_1 a_2 a_1 a_2$:

positions 0, 1, 2, 3; positions 0, 2, 3 labeled $a_1$; position 1 labeled $a_2$
Each MSO sentence \( \varphi \) defines a language

\[
L(\varphi) = \{ w \in \Sigma^* \mid w \models \varphi \}
\]

**Theorem (Büchi, Elgot, Trakhtenbrot 1960)**

MSO-definability = Regular languages

A similar result holds for trees as well – both binary and unranked.

We now show how to go from MSO to automata.
Types

Each FO sentence is a disjunction of types.

- **Rank-**$k$** type** $tp_k(D)$: set of all sentences of quantifier rank $k$ true in a database $D$.

- Types are finite objects, definable in the logic: finitely many distinct FO sentences of quantifier rank $k$, up to logical equivalence.

Another way of looking at Ehrenfeucht-Fraïssé games:

$$tp_k(D) = tp_k(D')$$

Equivalent:

Duplicator has a winning strategy in the $k$-round game on $D$ and $D'$.

For MSO, the same is true, but the game is slightly more complex: players can play both points and sets.
From MSO to automata via composition

Rank-\( k \) type of \( w \) uniquely determines rank-\( k \) type of \( w \cdot a \).

If \( tp_k(w) = tp_k(w') \), then \( tp_k(w \cdot a) = tp_k(w' \cdot a) \): compose games!
From MSO to automata via composition

Rank-$k$ type of $w$ uniquely determines rank-$k$ type of $w \cdot a$.

If $tp_k(w) = tp_k(w')$, then $tp_k(w \cdot a) = tp_k(w' \cdot a)$: compose games!

$w$

| duplicator wins
| in $k$ rounds

$w'$
From MSO to automata via composition

Rank-$k$ type of $w$ uniquely determines rank-$k$ type of $w \cdot a$.

If $tp_k(w) = tp_k(w')$, then $tp_k(w \cdot a) = tp_k(w' \cdot a)$: compose games!
From MSO to automata via composition

Rank-\(k\) type of \(w\) uniquely determines rank-\(k\) type of \(w \cdot a\).

If \(tp_k(w) = tp_k(w')\), then \(tp_k(w \cdot a) = tp_k(w' \cdot a)\): compose games!

\(w\)
\(a\)

duplicator still wins

in \(k\) rounds

\(w'\)
\(a\)
From MSO to automata: automata compute types

The rank-$k$ type of $w$ uniquely determines the rank-$k$ type of $w \cdot a$.

**Deterministic Automaton** for sentence $\varphi$:

- **States** are rank-$k$ types;
- **Initial state**: the type of the empty word;
- **Final states**: those types whose disjunction forms $\varphi$.
- **Transition** $\delta(\tau, a)$: the type of $w \cdot a$ if the type of $w$ is $\tau$.

After reading $w$, the state of the automaton is $\text{tp}_k(w)$. 
Language for extracting positions in words?

We need a language for extracting positions in trees:

- Information extraction from XML document;
- Work by Gottlob, Koch, and colleagues; Lixto system

We demonstrate the idea on words; it works for trees as well.
Language for extracting positions in words?

We need a language for extracting positions in trees:

- Information extraction from XML document;
- Work by Gottlob, Koch, and colleagues; Lixto system

We demonstrate the idea on words; it works for trees as well.

Use the composition method:

1) \( \text{tp}_k(w) = \text{tp}_k(w') \)
\( \text{tp}_k(u) = \text{tp}_k(u') \) \( \Rightarrow \) \( \text{tp}_k(w \cdot a \cdot u) = \text{tp}_k(w' \cdot a \cdot u') \)

2) \( \text{tp}_k(u) = \text{tp}_k(w) \) \( \Rightarrow \) \( \text{tp}_k(w^{-1}) = \text{tp}_k(u^{-1}) \)
Language for extracting positions in words

How to express MSO (or FO) $\varphi(x)$ over words?

Idea: for $w \cdot a \cdot u$, compute

1. $tp_k(w)$ going forward from the first position;
2. $tp_k(u^{-1})$ going backwards from the last position;
3. These types tell us whether the $a$ position is selected.

Express this in **Datalog**. Compute $tp_k(w)$ going forward – use predicates $U_\tau$ for types:

$$U_{\tau a}(x) \leftarrow \text{First}(x), L_a(x); \quad a \in \Sigma$$
$$U_{\tau'}(x) \leftarrow \text{Succ}(y, x), L_a(x), U_\tau(y); \quad a \in \Sigma, \; \delta(\tau, a) = \tau'$$
Datalog program cont’d

- Types going forward:

\[
\begin{align*}
U_{\tau a}(x) & : = \text{First}(x), L_a(x); \quad a \in \Sigma \\
U_{\tau'}(x) & : = \text{Succ}(y, x), L_a(x), U_{\tau}(y); \quad a \in \Sigma, \delta(\tau, a) = \tau'
\end{align*}
\]

- Types going backwards \( V_{\tau} \): symmetric.

- **Answer** – for all triples \((\tau, a, \tau')\) saying that \(a\) is selected, add:

\[
\text{Answer}(x) : = U_{\tau}(y), \text{Succ}(y, x), P_a(x), \text{Succ}(x, z), V_{\tau'}(z)
\]

- We used **Monadic Datalog**: all idb predicates are monadic.
  - Edb predicates: successor, labelings, First and Last.

- It captures MSO over words.

- Complexity of evaluating program \(P\) on \(w\):

\[
O(||P|| \cdot |w|)
\]
Review of the journey

Composition technique suggested using monadic datalog:

- captures MSO;
- has very good complexity bounds.

The approach works for trees and yields many XML languages:

- Monadic datalog captures MSO for trees, with the same complexity – one needs to add predicates for the root, leaves, first and last children of nodes (Gottlob, Koch, ’01)
- Other approaches:
  - ETL – Efficient tree logic (Neven, Schwentick, ’00)
  - Temporal logics with good query evaluation properties (Schlingloff ’92, Marx ’04, Barceló, L., ’05)
  - Dialects of XPath (Marx ’04)
Plan

• Expressiveness: Tools for first-order logic
• Expressiveness: Tools that work beyond first-order logic
• Language equivalence (better query evaluation via the composition method)
• Descriptive Complexity
• Satisfiability
Descriptive complexity

- Machine-independent characterization of complexity classes.
- A query language tells you a lot about the complexity.
- If your logic captures a complexity class, you have even more information:
  - complexity cannot be lowered.

First result:

**Theorem (Fagin 1974)**

\[
NP = \text{Existential Second-Order Logic (ESO)}
\]

- ESO = \( \exists R_1 \ldots \exists R_k \varphi(R_1, \ldots, R_k, E) \)
- 3-colorability: \( \exists R \exists G \exists B \varphi \)
  - \( \varphi \) says that \( R, G, B \) partition the set of nodes and endpoints of an edge cannot be in the same set.
Descriptive complexity – other classes

- FO is contained in $AC^0$
  - $AC^0$ – constant parallel time: constant time with polynomially many processors. Suggests very efficient parallel algorithms.
  - Complexity of the relational calculus.
  - Uniform version of $AC^0$ is captured by FO with $<, +, \times$ on the finite universe.
- Basic SQL (FO+simple arithmetic+aggregation) is contained in $TC^0$
  - $TC^0$ – constant parallel time with more complex gates (including majority gates). Probably a small subset of $DLOGSPACE$ but not yet separated from $NP$!
- FO+transitive closure is contained in $NLOGSPACE$.
- FO+fixed-points (including Datalog) is contained in $PTIME$
  - Over ordered databases, captures $PTIME$. 
Descriptive complexity – other classes

- $\text{NP} = \text{ESO}$.
- Second-order logic = \textit{Polynomial hierarchy} (between $\text{PTIME}$ and $\text{PSPACE}$)
- $\text{FO} + \text{partial fixed-point} \ (\text{need not converge: if it does not converge, the result is empty})$ – contained in $\text{PSPACE}$
  - Captures $\text{PSPACE}$ over ordered databases
- These classes typically appear when deals with schemas, queries, etc
  - small objects compared to databases
  - e.g., conjunctive query containment is $\text{NP}$-complete;
  - e.g., composing schema mappings in data exchange is $\text{NP}$-complete;
  - e.g., equivalence of DTDs is $\text{PSPACE}$-complete
Plan

- Expressiveness: Tools for first-order logic
- Expressiveness: Tools that work beyond first-order logic
- Language equivalence (better query evaluation via the composition method)
- Descriptive Complexity
- Satisfiability
Satisfiability

- Satisfiability problem:

  **INPUT:** FO sentence $\varphi$
  **QUESTION:** does it it have a model ($\mathcal{A} \models \varphi$)?

- Undecidable, but the complement is r.e. (recursively enumerable):
  - The complement is **validity**: Given $\varphi$, is it true in every model $\mathcal{A}$?
  - Because $\varphi$ is satisfiable iff $\neg \varphi$ is not valid
  - Validity is r.e. due to the completeness of FO.
Satisfiability

- Satisfiability problem:

  **Input:** FO sentence $\varphi$
  **Question:** does it it have a model ($\mathcal{A} \models \varphi$)?

- Undecidable, but the complement is r.e. (recursively enumerable):
  - The complement is validity: Given $\varphi$, is it true in every model $\mathcal{A}$?
  - Because $\varphi$ is satisfiable iff $\lnot \varphi$ is not valid
  - Validity is r.e. due to the completeness of FO.

- Finite satisfiability: Given $\varphi$, does it have a finite model?

- Finite validity: Given $\varphi$, is it true in all finite models?

- Completeness fails in the finite, so we cannot get r.e. as before.

  **Theorem (Trakhtenbrot 1950)**

  *Finite validity is not r.e.*
Satisfiability: application

- Static analysis problems (schemas, queries, etc)
- **certain answers**: key concept for
  - incomplete information;
  - query answering using views;
  - data integration and exchange

- Databases with incomplete information:

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>x</td>
<td>A</td>
</tr>
<tr>
<td>y</td>
<td>x</td>
<td>B</td>
</tr>
<tr>
<td>y</td>
<td>CS1</td>
<td>A-</td>
</tr>
<tr>
<td>Jones</td>
<td>CS2</td>
<td>z</td>
</tr>
</tbody>
</table>
Satisfiability: application

- Static analysis problems (schemas, queries, etc)
- **certain answers**: key concept for
  - incomplete information;
  - query answering using views;
  - data integration and exchange

- Databases with incomplete information:

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>x</td>
<td>A</td>
</tr>
<tr>
<td>y</td>
<td>x</td>
<td>B</td>
</tr>
<tr>
<td>y</td>
<td>CS1</td>
<td>A-</td>
</tr>
<tr>
<td>Jones</td>
<td>CS2</td>
<td>z</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>CS4</td>
<td>A</td>
</tr>
<tr>
<td>Smith</td>
<td>CS4</td>
<td>B</td>
</tr>
<tr>
<td>Smith</td>
<td>CS1</td>
<td>A-</td>
</tr>
<tr>
<td>Jones</td>
<td>CS2</td>
<td>B+</td>
</tr>
<tr>
<td>Jones</td>
<td>CS3</td>
<td>B-</td>
</tr>
</tbody>
</table>

- All such databases representing a table $T$: $\text{Rep}(T)$
Certain answers

- Answers to a query that do not depend on the interpretation of missing information:

$$\text{certain}(Q, T) = \bigcap_{R \in \text{Rep}(T)} Q(R)$$

- What if $Q$ is an FO sentence and $T = \emptyset$?
- $\text{certain}(Q, T) = \text{true}$ iff $Q$ is valid! Hence:

**Corollary**

*For FO queries, computing certain answers is undecidable.*

- Hence we (database people) are interested in classes with decidable satisfiability.
Decidable classes

- Conjunctive queries and their unions
  - finding certain answers - PTIME
  - plays very important in data integration/exchange
- Other classes go via finite model property: if there is a model, there must be a finite one.
- Two important classes:
  - Bernays-Schönfinkel class: $\exists x_1 \ldots \exists x_m \forall y_1 \ldots \forall y_k \alpha$
  - $FO^2$ – formulae with 2 variables $x$ and $y$.
- In both cases, satisfiability is NEXPTIME-complete.
- Both results have been used in database theory (e.g. recently in the XML context).
Conclusion

- Quite likely, if you need to use FMT in database research, you need one of the techniques described in this tutorial (paper).
- No need to read textbooks/proofs – just use the toolbox!
  - Unless you want to work in FMT.
Conclusion

- Quite likely, if you need to use FMT in database research, you need one of the techniques described in this tutorial (paper).
- No need to read textbooks/proofs – just use the toolbox!
  - Unless you want to work in FMT.
- But if you want to read an FMT book, ask me after the talk and I’ll tell you which one to buy.
The future of FMT

- We’ve learned how to work with both stone and iron age tools.
- It’s time to go back to games and start putting stones together.
- Middle Ages tools: actively developed by Ben Rossman (the rest of the world is trying to catch up). Solved 3 long-standing open problems:
  1. successor-invariance
  2. preservation under homomorphisms in the finite
  3. strictness of the $\text{FO}^k$ hierarchy over ordered structures.
- Next step, $N$ years away: modern era tools.
  - Main application: separating complexity classes.

**Good news** – most of the database theory tasks are easily doable with stone and iron age tools.