Logical foundations of databases

Diego Figueira

Gabriele Puppis

CNRS
LaBRI

ESSLLI 2016
Bolzano, Italy
Gabriele Puppis
PhD from Udine (Italy)
post-docs in Oxford
Works in LaBRI, Bordeaux
CNRS researcher

Diego Figueira
PhD from ENS Cachan (France),
post-docs in Warsaw, Edinburgh
Works in LaBRI, Bordeaux
CNRS researcher
First and foremost…

interrupt!
**Databases**

**DBMS** = a collection of data, structured in some way + a way of defining, querying, updating the data inside

meditate between humans, processes & data

* [Abitebou, Hull, Vianu “Foundations of databases”]
**DBMS** = a collection of data, structured in some way + a way of defining, querying, updating the data inside

* [Abitebou, Hull, Vianu “Foundations of databases”]

**Data model**
- how the data is *logically organised*
- mathematical abstraction for representing data
- independent from physical organisation

DBMS also implement: transactions, concurrency, access control, resiliency…

* [Abitebou, Hull, Vianu “Foundations of databases”]
Relational databases, historical outlook

1970–72: E.F. Codd (IBM San Jose research lab) introduces the "relational data model" and two query languages: "relational algebra" and "relational calculus"

1974–75: IBM researchers start implementing
- "System R": first relational database management system (RDBMS).
- SEQUEL: a query language based on relational algebra

1983: IBM "DB2" is released, based on System R.
And UC Berkley released Ingres RDBMS

1979: Oracle Corporation is founded

1981: Codd receives Turing award

Now: multi-billion industry

<table>
<thead>
<tr>
<th>Company</th>
<th>2006 Revenue</th>
<th>2006 Market Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>7.168B</td>
<td>47.1%</td>
</tr>
<tr>
<td>IBM</td>
<td>3.204B</td>
<td>21.1%</td>
</tr>
<tr>
<td>Microsoft</td>
<td>2.654B</td>
<td>17.4%</td>
</tr>
<tr>
<td>Teradata</td>
<td>494.2M</td>
<td>3.2%</td>
</tr>
<tr>
<td>Sybase</td>
<td>486.7M</td>
<td>3.2%</td>
</tr>
<tr>
<td>Other</td>
<td>1.2B</td>
<td>7.8%</td>
</tr>
<tr>
<td>Total</td>
<td>15.2B</td>
<td>100%</td>
</tr>
</tbody>
</table>
Relational data model = data logically organised into relations ("tables").

What’s a relation?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns
**Relational** data model = data logically organised into relations (“tables”).

What’s a **relation**?
- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

\[ \{(1,a,2), (2,b,6), (2,a,1)\} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N} \]
Relational databases

**Relational** data model = data logically organised into relations ("tables").

What’s a relation?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

\[
\{ (1,a,2), (2,b,6), (2,a,1) \} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N}
\]

a “tuple” (a “3-tuple”)
Relational databases

**Relational** data model = data logically organised into relations (“tables”).

What’s a relation?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like:

\[
\{ (1, a, 2), (2, b, 6), (2, a, 1) \} \subseteq \mathbb{N} \times \{a, b\} \times \mathbb{N}
\]

a “tuple” (a “3-tuple”)

() \rightarrow 0-tuple
Relational databases

Relational data model = data logically organised into relations ("tables").

What’s a relation?

- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

like: \( \{ (1,a,2), (2,b,6), (2,a,1) \} \subseteq \mathbb{N} \times \{a,b\} \times \mathbb{N} \)

a “tuple” (a “3-tuple”)

0-tuple
Relational databases

**Relational** data model = data logically organised into relations (“tables”).

What’s a relation?

• a (finite) subset of the cartesian product of sets
• a “table” with rows and columns

DB = A **schema**: names of tables and attributes

An **instance**: data conforming to the schema
Relational data model = data logically organised into relations ("tables").

What’s a relation?

• a (finite) subset of the cartesian product of sets
• a “table” with rows and columns

DB = A schema: names of tables and attributes

- Films (Title:string, Director:string, Actor:string)
- Schedule (Theatre:string, Title:string)

An instance: data conforming to the schema
Relational databases

**Relational** data model = data logically organised into relations (“tables”).

What’s a **relation**?
- a (finite) subset of the cartesian product of sets
- a “table” with rows and columns

DB = A **schema**: names of tables and attributes

- **Films** (Title:string, Director:string, Actor:string)
- **Schedule** (Theatre:string, Title:string)

An **instance**: data conforming to the schema

<table>
<thead>
<tr>
<th>Films</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
<td><strong>Theatre</strong></td>
</tr>
<tr>
<td>8 1/2</td>
<td>Utopia</td>
</tr>
<tr>
<td>Shining</td>
<td>Utopia</td>
</tr>
<tr>
<td>Dr. Strangelove</td>
<td>UGC</td>
</tr>
<tr>
<td>8 femmes</td>
<td>UGC</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Director</strong></td>
<td><strong>Title</strong></td>
</tr>
<tr>
<td>Fellini</td>
<td>Dr. Strangelove</td>
</tr>
<tr>
<td>Kubrick</td>
<td>8 1/2</td>
</tr>
<tr>
<td>Kubrick</td>
<td>Dr. Strangelove</td>
</tr>
<tr>
<td>Ozon</td>
<td>8 femmes</td>
</tr>
</tbody>
</table>
Relational databases

**Relational** data model = data logically organised into relations (“tables”).

⚠️ We assume all elements come from a fixed set of constants or *data values* $U$. 
What is a query $q$?

A mapping that takes a database instance $D$ and returns a relation $q(D) \subseteq U^r$ of fixed arity $r$. 
What is a query $q$?

A mapping that takes a database instance $D$ returns a relation $q(D) \subseteq U^r$ of fixed arity $r$. 

computable!
What is a query $q$?

A mapping that takes a database instance $D$ returns a relation $q(D) \subseteq U^r$ of fixed arity $r$.
What is a query \( q \) ?

A mapping that takes a database instance \( D \) returns a relation \( q(D) \subseteq U^r \) of fixed arity \( r \).

- **Computable!**
- **Generic!** (order independent)
- **Boolean query**: \( r=0 \)
  Either “yes” \( \{ () \} \) or “no” \( \{ \} \)
What is a query $q$?

A mapping that takes a database instance $D$ returns a relation $q(D) \subseteq U^r$ of fixed arity $r$.

What do we care about queries?

expressive power  evaluation  static analysis
The fundamental questions:

How to query the relational data model?
How efficient/expressive is it?
How to query the relational data model?
How efficient/expressive is it?
Query Language

Syntax

Expressions for querying the db, governed by syntactic rules

“Select X from Y”

“y :- ∀x (x ≤ y)”

Semantics

Interpretation of symbols in terms of some structure

Retrieves all strings in column X of table Y

Returns the maximum element of the set.
Relational Algebra (RA)

Syntax:

\[ E := R, S, \ldots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E) \]

where \( M \subseteq \mathbb{N} \)
\( \Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N} \)
Relational Algebra (RA)

Syntax: $E := R, S, \ldots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E)$

where $M \subseteq \mathbb{N}$

$\Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N}$

• $R_1 \cup R_2$: Set union
• $R_1 \times R_2$: Cartesian product
• $R_1 \setminus R_2$: Set difference
Relational Algebra (RA)

Syntax: \( E := R, S, \ldots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E) \)

where \( M \subseteq \mathbb{N} \)
\[ \Theta \subseteq \mathbb{N} \times \{=,\neq\} \times \mathbb{N} \]

- \( R_1 \cup R_2 \): Set union
- \( R_1 \times R_2 \): Cartesian product
- \( R_1 \setminus R_2 \): Set difference
- \( \sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1}=x_{j_1}) \land \cdots \land (x_{i_n}=x_{j_n})\} \) : Selection

[Codd, 1970]
Relational Algebra (RA)

Syntax: \( E := R, S, \ldots | E \cup E | E \setminus E | E \times E | \pi_M(E) | \sigma_\Theta(E) \)

where \( M \subseteq \mathbb{N} \)
\( \Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N} \)

- \( R_1 \cup R_2 \): Set union
- \( R_1 \times R_2 \): Cartesian product
- \( R_1 \setminus R_2 \): Set difference
- \( \sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) \) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\} \): Selection

[Codd, 1970]
Relational Algebra (RA)

Syntax: \[ E := R, S, \ldots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E) \]

where \( M \subseteq \mathbb{N} \) and \( \Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N} \)

- \( R_1 \cup R_2 \) : Set union

- \( R_1 \times R_2 \) : Cartesian product

- \( R_1 \setminus R_2 \) : Set difference

- \( \sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\} : \) Selection

\[ \sigma_{\{1=3, 1 \neq 2\}}(\{(1,2,1), (2,2,2)\}) = \{(1,2,1)\} \]
Relational Algebra (RA)

Syntax: \[ E := R, S, \ldots \mid E \cup E \mid E \setminus E \mid E \times E \mid \pi_M(E) \mid \sigma_\Theta(E) \]

where \( M \subseteq \mathbb{N} \)
\( \Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N} \)

- \( R_1 \cup R_2 \): Set union
- \( R_1 \times R_2 \): Cartesian product
- \( R_1 \setminus R_2 \): Set difference

\[ \sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\} : \text{Selection} \]

\[ \pi_{\{i_1, \ldots, i_n\}}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\} : \text{Projection} \]
Relational Algebra (RA)

Syntax: \[ E := R, S, \ldots | E \cup E | E \setminus E | E \times E | \pi_M(E) | \sigma_\Theta(E) \]

where \( M \subseteq \mathbb{N} \)
\( \Theta \subseteq \mathbb{N} \times \{=, \neq\} \times \mathbb{N} \)

- \( R_1 \cup R_2 \): Set union

\[
\sigma_{\{1=3, 1 \neq 2\}}(\{(1,2,1), (2,2,2)\}) = \{(1,2,1)\}
\]

\[
\pi_{\{1,3\}}(\{(1,2,1), (2,2,2)\}) = \{(1,1), (2,2)\}
\]

- \( R_1 \times R_2 \): Cartesian product

- \( R_1 \setminus R_2 \): Set difference

- \( \sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \ldots \land (x_{i_n} \neq x_{j_n})\} : Selection \)

- \( \pi_{\{i_1, \ldots, i_n\}}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\} : Projection \)
Relational Algebra (RA)

• $R_1 \cup R_2$: Set union
• $R_1 \times R_2$: Cartesian product
• $R_1 \setminus R_2$: Set difference
• $\sigma_{i_1=j_1, \ldots, i_n=j_n}(R) : \{ (x_1, \ldots, x_m) \in R \mid (x_{i_1}=x_{j_1}) \land \cdots \land (x_{i_n}=x_{j_n}) \}$: Selection
• $\pi_{i_1, \ldots, i_n}(R) : \{ (x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R \}$: Projection

Question 1: What is the RA expression for
\[
\{ (v_1, v_2) \mid \text{there are } w_1 \neq w_2 \text{ so that } (v_1, w_1) \in R_1 \text{ and } (v_2, w_2) \in R_2 \}\, ?
\]

Question 2: $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2)) = ?$
Relational Algebra (RA)

- $R_1 \cup R_2$ : Set union
- $R_1 \times R_2$ : Cartesian product
- $R_1 \setminus R_2$ : Set difference

$\sigma_{\{i_1 \neq j_1, \ldots, i_n \neq j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\}$ : Selection

$\pi_{\{i_1, \ldots, i_n\}}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\}$ : Projection

**Question 1:** What is the RA expression for

$\{ (v_1,v_2) \mid \text{there are } w_1 \neq w_2 \text{ so that } (v_1,w_1) \in R_1 \text{ and } (v_2,w_2) \in R_2 \}$ ?

**Answer:** $\pi_{\{1,3\}}(\sigma_{1 \neq 3}(R_1 \times R_2))$

**Question 2:** $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2)) = ?$
Relational Algebra (RA)

- $R_1 \cup R_2$: Set union
- $R_1 \times R_2$: Cartesian product
- $R_1 \setminus R_2$: Set difference
- $\sigma_{i_1=j_1, \ldots, i_n=j_n}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\}$: Selection
- $\pi_{i_1, \ldots, i_n}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\}$: Projection

**Question 1:** What is the RA expression for 
\[
\{ (v_1, v_2) \mid \text{there are } w_1 \neq w_2 \text{ so that } (v_1, w_1) \in R_1 \text{ and } (v_2, w_2) \in R_2 \}\ ?
\]

**Answer:** $\pi_{1,3}(\sigma_{1 \neq 3}(R_1 \times R_2))$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

**Question 2:** $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2)) = ?$

**Answer (only one element):** b
RA = Basic SQL

no domain-specific features, aggregation, etc

Select $X$
From $R_1, ..., R_n$  $\iff$  $\pi_X (\sigma_Z ( R_1 \times ... \times R_n ))$
Where $Z$

... or ...
$\iff$ union

... not in (...)
$\iff$ difference
**RA = Basic SQL**

- no domain-specific features, aggregation, etc

Select $X$
From $R_1, \ldots, R_n$  \iff  $\pi_X (\sigma_Z (R_1 \times \cdots \times R_n))$
Where $Z$

- ... or ...
- ... not in (...)  \iff  union
- ... not in (...)  \iff  difference

\[ \pi_2 (\sigma_1 \neq 3 (R_1 \times R_2)) \sim \]

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$R_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>
RA = Basic SQL

no domain-specific features, aggregation, etc

Select $X$
From $R_1, \ldots, R_n$  $\iff$  $\pi_X (\sigma_Z (R_1 \times \ldots \times R_n))$
Where $Z$

... or ...  $\iff$  union
... not in (...)  $\iff$  difference

$\pi_2 (\sigma_{1 \neq 3}(R_1 \times R_2))  \sim$
Select $R_{1.2}$ as $\text{foo}$
From $R_1$, $R_2$
Where $R_{1.1} \neq R_{2.1}$

<table>
<thead>
<tr>
<th></th>
<th>$R_1$</th>
<th>$R_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

a, b, c are values from the table.
RA = Basic SQL

Select X
From R₁,...,Rₙ ⇔ \( \pi_X( \sigma_Z( R₁ \times \cdots \times Rₙ ) ) \)
Where Z

... or ...

... not in (...

no domain-specific features, aggregation, etc

Select R₁.2 as foo
From R₁, R₂
Where R₁.₁ ≠ R₂.₁

Select R₁.2 as foo
From R₁, R₂
Where R₁.₁ ≠ R₂.₁

\( \pi_2(\sigma_{1 \neq 3}(R₁ \times R₂)) \)  \( \sim \)

\( \pi_2(\sigma_{1 = 3}(R₁ \times R₂)) \)  \( \sim \)

R₁
\[ \begin{array}{|c|}
\hline
a & 3 \\
\hline
b & 2 \\
\hline
c & 4 \\
\hline
b & 3 \\
\hline
a & 2 \\
\hline
\end{array} \]

R₂
\[ \begin{array}{|c|}
\hline
a & 4 \\
\hline
b & 1 \\
\hline
c & 4 \\
\hline
b & 2 \\
\hline
a & 1 \\
\hline
b & 3 \\
\hline
\end{array} \]
\[ \text{Select } X \]
\[ \text{From } R_1, \ldots, R_n \quad \iff \quad \pi_X ( \sigma_Z ( R_1 \times \cdots \times R_n ) ) \]
\[ \text{Where } Z \]

... or ...
... not in (...

\[ \pi_2 ( \sigma_{1 \neq 3} ( R_1 \times R_2 ) ) \sim \]

\[ \pi_2 ( \sigma_{1=3} ( \times \times R_2 ) ) \sim \]

\[ \begin{array}{|l|}
\hline
R_1 & R_2 \\
\hline
a & 3 \\
b & 2 \\
c & 4 \\
b & 3 \\
a & 2 \\
\hline
\end{array} \]
Select $X$
From $R_1, \ldots, R_n$ $\iff$ $\pi_X(\sigma_Z(R_1 \times \cdots \times R_n))$
Where $Z$

... or ...
$\iff$ union

... not in (...)
$\iff$ difference

\[
\pi_2(\sigma_{1 \neq 3}(R_1 \times R_2)) \leadsto \begin{aligned}
&\text{Select } R_1.2 \text{ as } \text{foo} \\
&\text{From } R_1, R_2 \\
&\text{Where } R_1.1 \neq R_2.1
\end{aligned}
\]

\[
\pi_2(\sigma_{1=3}(\bigstar \times R_2)) \leadsto \begin{aligned}
&\text{Select } \text{foo} \\
&\text{From } \bigstar, R_2 \\
&\text{Where } \text{foo} = R_2.2
\end{aligned}
\]
Denotational languages

Algebra $\sim$ *How* to obtain the result

Logics $\sim$ *What* is the property of the result

Procedural

Declarative
Denotational languages

Algebra $\leadsto$ \textit{How} to obtain the result

Logics $\leadsto$ \textit{What} is the property of the result

\{ Relational Algebra
operations on tables \}

Procedural

Declarative
Denotational languages

- **Relational Algebra**
  - operations on tables
  - Algebra \(\sim\) *How* to obtain the result

- **Logics**
  - First Order logic
  - properties on mathematical structures
  - Logics \(\sim\) *What* is the property of the result

- Procedural
- Declarative
Denotational languages

Algebra \sim \textit{How} to obtain the result

Logics \sim \textit{What} is the property of the result

\begin{itemize}
  \item \textbf{Relational Algebra}
    \begin{itemize}
      \item operations on tables
    \end{itemize}
  \item \textbf{First Order logic}
    \begin{itemize}
      \item properties on mathematical structures
    \end{itemize}
\end{itemize}
FO = First-Order logic
A structure is:

\[ A = (D, R_1, \ldots, R_n, f_1, \ldots, f_n) \]

- \(D\) is a non-empty set, the domain
- \(R_i\) is an \(m\)-ary relation for some \(m\) (ie, \(R_i \subseteq D^m\))
- \(f_i\) is an \(n\)-ary function for some \(n\) (ie, \(f_i : D^n \rightarrow D\))
Relational structures

A **structure** is:

\[ A = (D, R_1, \ldots, R_n, f_1, \ldots f_n) \]

*\( D \) is a non-empty set, the domain

*\( R_i \) is an \( m \)-ary relation for some \( m \) (ie, \( R_i \subseteq D^m \))

*\( f_i \) is an \( n \)-ary function for some \( n \) (ie, \( f_i : D^n \rightarrow D \))

A **graph** \( G = (V,E) \)

- *\( V \): nodes
- *\( E \subseteq V^2 \): edges (binary relation)
- *(no functions)*

A **group**, like \( (\mathbb{N},+) \)

- *\( \mathbb{N} \): natural numbers
- *(no relations)*
- *+: \( \mathbb{N}^2 \rightarrow \mathbb{N} \) addition (binary function)*
First-order logic
First-order logic

variables $x, y, z, \ldots$
quantifiers: $\exists, \forall$
Boolean connectives: $\neg, \land, \lor$

A language to talk about structures
Variables range over the domain
Atomic formulas: $R(x_1, \ldots, x_m), x=y$
First-order logic

FO

variables $x$, $y$, $z$, …
quantifiers: $\exists$, $\forall$
Boolean connectives: $\neg$, $\land$, $\lor$

A language to talk about structures
Variables range over the domain
Atomic formulas: $R(x_1, \ldots, x_m)$, $x=y$

A graph $G = (V,E)$
- $V$: nodes
- $E \subseteq V^2$: edges (binary relation)
- (no functions)

Language to talk about graphs
Variables range over nodes
Atomic formulas: $E(x,y)$, $x = y$
First-order logic

Variables $x, y, z, \ldots$

Quantifiers: $\exists, \forall$

Boolean connectives: $\neg, \land, \lor$

A language to talk about structures

Variables range over the domain

Atomic formulas: $R(x_1, \ldots, x_m)$, $x = y$

A graph $G = (V, E)$

- $V$: nodes
- $E \subseteq V^2$: edges (binary relation)
- (no functions)

Language to talk about graphs

Variables range over nodes

Atomic formulas: $E(x, y)$, $x = y$

Formulas: Atomic formulas + connectives + quantifiers
“The node x has at least two neighbours”

$\exists y \exists z \ (\neg(y=z) \land E(x,y) \land E(x,z))$
“The node $x$ has at least two neighbours”

$$\varphi(x) = \exists y \exists z \, (\neg(y=z) \land E(x,y) \land E(x,z))$$

$x$ is **free** = not quantified

(a property of a **node** in the **graph**)

The node $x$ has at least two neighbours
“The node $x$ has at least two neighbours”

$\varphi(x) = \exists y \exists z \left( \neg(y=z) \land E(x,y) \land E(x,z) \right)$

x is free = not quantified
(a property of a node in the graph)

“Each node has at least two neighbours”

$\forall x \exists y \exists z \left( \neg(y=z) \land E(x,y) \land E(x,z) \right)$
“The node $x$ has at least two neighbours”

$$\varphi(x) = \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))$$

**x is free** = not quantified
(a property of a node in the graph)

“Each node has at least two neighbours”

$$\psi = \forall x \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))$$

the formula is a sentence
= no free variables
(a property of the graph)
“The node $x$ has at least two neighbours”
\[
\varphi(x) = \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))
\]
x is free = not quantified (a property of a node in the graph)

“Each node has at least two neighbours”
\[
\psi = \forall x \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))
\]

the formula is a sentence = no free variables (a property of the graph)

**Question:**  • How to express in FO

“Every two adjacent nodes have a common neighbour” ?

• Does it have free variables? Is it a sentence?
“The node $x$ has at least two neighbours”

$\varphi(x) = \exists y \exists z \ (\neg(y=z) \land E(x,y) \land E(x,z))$

$x$ is free = not quantified
(a property of a node in the graph)

“Each node has at least two neighbours”

$\psi = \forall x \exists y \exists z \ (\neg(y=z) \land E(x,y) \land E(x,z))$

the formula is a sentence
= no free variables
(a property of the graph)

Question:  
• How to express in FO
   “Every two adjacent nodes have a common neighbour” ?

• Does it have free variables? Is it a sentence?

Answer:  
$\forall x \forall y \left( \neg E(x,y) \lor \exists z \left( (E(x,z) \lor E(z,x)) \land (E(y,z) \lor E(z,y)) \right) \right)$
To evaluate a formula $\phi$ we need a graph $G=(V,E)$ and a **binding** $\alpha$ that maps free variables of $\phi$ to nodes of $G$.

$$G \models_\alpha \phi(x_1,\ldots,x_n) \quad \alpha : \{x_1,\ldots,x_n\} \rightarrow V \quad \text{assigns nodes to free variables}$$
To evaluate a formula $\phi$ we need a graph $G=(V,E)$ and a binding $\alpha$ that maps free variables of $\phi$ to nodes of $G$.

$G \models_\alpha \phi(x_1,\ldots,x_n)$ \hspace{1cm} $\alpha : \{x_1,\ldots,x_n\} \rightarrow V$ \hspace{1cm} assigns nodes to free variables

\[ \text{“} G, \alpha \text{ satisfy } \phi \text{”} \hspace{1cm} \text{“} \phi \text{ is satisfiable} \text{”} \]
To evaluate a formula $\phi$ we need a graph $G=(V,E)$ and a **binding** $\alpha$ that maps free variables of $\phi$ to nodes of $G$.

\[
G \models_{\alpha} \phi(x_1,\ldots,x_n) \quad \alpha : \{x_1,\ldots,x_n\} \longrightarrow V \quad \text{assigns nodes to free variables}
\]

"The node $x$ has at least two neighbours"

$\phi(x) = \exists y \exists z \ (\neg(y=z) \land E(x,y) \land E(x,z))$

$G \models_{\alpha} \phi$ if $\alpha = \{x \mapsto v\}$
To evaluate a formula $\phi$ we need a graph $G=(V,E)$ and a binding $\alpha$ that maps free variables of $\phi$ to nodes of $G$.

$G \models \alpha \phi$ if $\alpha : \{x_1, \ldots, x_n\} \rightarrow V$ assigns nodes to free variables.

“Every node has at least two neighbours”

$\psi = \forall x \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))$

$G \models \emptyset \psi$

“Every node has at least two neighbours”

$\psi = \forall x \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))$

$G \models \emptyset \psi$
First-order logic

Formal Semantics of FO

\[ G \models_\alpha \exists x \phi \quad \text{iff} \quad \text{for some } v \in V \text{ and } \alpha' = \alpha \cup \{x \mapsto v\} \text{ we have } G \models_{\alpha'} \phi \]

\[ G \models_\alpha \forall x \phi \quad \text{iff} \quad \text{for every } v \in V \text{ and } \alpha' = \alpha \cup \{x \mapsto v\} \text{ we have } G \models_{\alpha'} \phi \]

\[ G \models_\alpha \phi \land \psi \quad \text{iff} \quad G \models_\alpha \phi \text{ and } G \models_\alpha \psi \]

\[ G \models_\alpha \neg \phi \quad \text{iff} \quad \text{it is not true that } G \models_\alpha \phi \]

\[ G \models_\alpha x = y \quad \text{iff} \quad \alpha(x) = \alpha(y) \]

\[ G \models_\alpha E(x, y) \quad \text{iff} \quad (\alpha(x), \alpha(y)) \in E \]
Formulas as queries

$\phi(x_1, ..., x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy $\phi$:

$$\phi(G) = \{ (\alpha(x_1), ..., \alpha(x_n)) \mid G \models_\alpha \phi, \ \alpha: \{x_1, ..., x_n\} \rightarrow V \}$$
Formulas as queries

$\phi(x_1, \ldots, x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy $\phi$:

$$\phi(G) = \{ (\alpha(x_1), \ldots, \alpha(x_n)) \mid G \vDash_\alpha \phi, \ \alpha: \{x_1, \ldots, x_n\} \rightarrow V \}$$

“The node $x$ has at least two neighbours”

$$\phi(x) = \exists y \exists z (\neg (y = z) \land E(x,y) \land E(x,z))$$

“Return all nodes with at least two neighbours”
Formulas as queries

$\phi(x_1, \ldots, x_n)$ evaluated on $G=(V,E)$ yields all the bindings that satisfy $\phi$:

$$\phi(G) = \{ (\alpha(x_1), \ldots, \alpha(x_n)) \mid G \models \alpha, \alpha : \{x_1, \ldots, x_n\} \rightarrow V \}$$

“The node x has at least two neighbours”

$\phi(x) = \exists y \exists z (\neg(y=z) \land E(x,y) \land E(x,z))$

$\phi(G) = \{v, v', v''\}$

$\phi(G') = \{v, v'\}$
Formulas as queries

\( \phi(x_1, \ldots, x_n) \) evaluated on \( G=(V,E) \) yields all the bindings that satisfy \( \phi \):

\[
\phi(G) = \{ \alpha(x_1), \ldots, \alpha(x_n) \mid G \vDash \alpha, \alpha: \{x_1, \ldots, x_n\} \rightarrow V \}
\]

“The node \( x \) has at least two neighbours”

\( \phi(x) = \exists y \exists z \ (\neg(y=z) \land E(x,y) \land E(x,z)) \)

\( \phi(G) = \{v, v', v''\} \)

\( \phi(G') = \{v, v'\} \)

“Every node has two neighbours”

\( \psi = \forall x \exists y \exists z \ (\neg(y=z) \land E(x,y) \land E(x,z)) \)

\( \psi(G) = \{()\} \sim \) set with one element: the 0-tuple

\( \psi(G') = \{} \sim \) empty set
Question: Which bindings $\alpha$ verify $G \vDash_\alpha \phi$ for

$$\phi(x,y) = \exists z \ (E(x,z) \land E(z,y))$$

and

$G = \cdots$
Question: Which bindings $\alpha$ verify $G \models_\alpha \phi$ for

$\phi(x,y) = \exists z \ (E(x,z) \land E(z,y))$

and $G = \langle \cdot \rangle$?

Answer: $\alpha = \{ x \mapsto v, y \mapsto v' \}$,

$\alpha = \{ x \mapsto v, y \mapsto v \}$, $\phi(G) = \{ v, v', v'' \} \times \{ v, v', v'' \}$

$\alpha = \{ x \mapsto v', y \mapsto v' \}$,

$\ldots$ and all the rest
FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

- Tables = Relations
- Queries = Formulas
- Rows = Tuples
Formulas as queries

FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

- Tables = Relations
- Queries = Formulas
- Rows = Tuples

**Particular to databases:**
- Use of constants
- No functions
- Finite structure
- Quantification over active domain
FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

\[
\begin{align*}
\text{Tables} &= \text{Relations} \\
\text{Queries} &= \text{Formulas} \\
\text{Rows} &= \text{Tuples}
\end{align*}
\]

**Particular to databases:**
- Use of constants
- No functions
- Finite structure
- Quantification over active domain

\(\vDash_{\text{finite}} \text{ is different from } \vDash\)
FO can serve as a **declarative** query language on relational databases: we express the properties of the answer.

\[
\text{Tables} = \text{Relations} \\
\text{Queries} = \text{Formulas} \\
\text{Rows} = \text{Tuples}
\]

**Particular to databases:**
- Use of constants
- No functions
- Finite structure
- Quantification over active domain

$\text{⊧}_{\text{finite}}$ is different from $\not=$. There are formulas $\phi$ that are satisfiable only on infinite structures. Like which?
Formulas as queries

FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

Tables = Relations
Queries = Formulas
Rows = Tuples

**Particular to databases:**
- Use of constants
- No functions
- Finite structure
- Quantification over active domain

⊧_{\text{finite}}$ is different from $\models$

There are formulas $\phi$ that are satisfiable only on infinite structures.

Like which?

$\phi = "R(x,y) \text{ is an infinite linear order}"$
FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

\[
\text{Tables} = \text{Relations} \\
\text{Queries} = \text{Formulas} \\
\text{Rows} = \text{Tuples}
\]

**Particular to databases:**
- Use of constants
- No functions
- Finite structure
- Quantification over active domain

\[\models_{\text{finite}} \text{ is different from } \models\]

There are formulas \(\phi\) that are satisfiable only on infinite structures.

Like which?

\[\phi = "R(x,y) \text{ is an infinite linear order}"\]

Finite model theory
Formulas as queries

FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

- Tables = Relations
- Rows = Tuples
- Queries = Formulas

[E.F. Codd 1972]
FO can serve as a **declarative** query language on relational databases: we express the properties of the answer

\[
\begin{align*}
\text{Tables} &= \text{Relations} \\
\text{Rows} &= \text{Tuples} \\
\text{Queries} &= \text{Formulas}
\end{align*}
\]

\[
\begin{align*}
\text{RA} &= \ast \text{ FO} \\
\text{How} &= \text{What}
\end{align*}
\]

[E.F. Codd 1972]
FO can serve as a **declarative** query language on relational databases: we express the properties of the answer.

\[
\text{Tables} = \text{Relations} \\
\text{Rows} = \text{Tuples} \\
\text{Queries} = \text{Formulas}
\]

\[
\text{RA} \ = \ \ast \ \text{FO} \\
\text{How} = \text{What}
\]

**RA and FO logic have roughly\(^*\) the same expressive power!**

\(^*\text{FO without functions, with equality, on finite domains, …}\)
Formulas as queries

$\mathbf{RA \subseteq FO}$

- $R_1 \times R_2 \leadsto R_1(x_1, \ldots, x_n) \land R_2(x_{n+1}, \ldots, x_m)$

- $R_1 \cup R_2 \leadsto R_1(x_1, \ldots, x_n) \lor R_2(x_1, \ldots, x_n)$

- $\sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) \leadsto R(x_1, \ldots, x_m) \land (x_{i_1}=x_{j_1}) \land \cdots \land (x_{i_n}=x_{j_n})$

- $\pi_{\{i_1, \ldots, i_n\}}(R) \leadsto \exists (\{x_1, \ldots, x_m\} \setminus \{x_{i_1}, \ldots, x_{i_n}\}). R(x_1, \ldots, x_m)$

- $R_1 \setminus R_2 \leadsto R_1(x_1, \ldots, x_n) \land \neg R_2(x_1, \ldots, x_n)$

- $\ldots$
Formulas as queries

\[ \text{FO} \subseteq \text{RA} \text{ does not hold in general!} \]
Formulas as queries

FO $\subseteq$ RA does not hold in general!

"the complement of R" $\not\in$ RA $\in$ FO : $\neg R(x)$
Formulas as queries

\[ \text{FO} \notin \text{RA} \]

“the complement of R” \( \notin \text{RA} \in \text{FO} : \neg R(x) \)
Formulas as queries

\[ \text{FO } \notin \text{ RA} \]

“the complement of R” \[ \notin \text{ RA} \]
\[ \in \text{ FO : } \neg R(x) \]

\[ \hookrightarrow \] We restrict variables to range over active domain
Formulas as queries

$\text{FO} \notin \text{RA}$

"the complement of R" $\notin \text{RA}$ $\in \text{FO} : \neg R(x)$

$\text{FO}^{\text{act}}$

= \text{FO restricted to active domain}
Formulas as queries

\[ \text{FO} \not\in \text{RA} \]

"the complement of R" \[ \not\in \text{RA} \]
\[ \in \text{FO} : \neg R(x) \]

We restrict variables to range over **active domain**

\[ \text{FO}^{\text{act}} \]
\[ = \]
\[ \text{FO restricted to active domain} \]

\[ \phi_1(x) = \forall y \ E(y,x) \]
\[ \phi_1(G) = \{v_2\} \]

\[ \phi_2(x,y) = \neg E(x,y) \]
\[ \phi_2(G) = \{(v_1,v_1),(v_3,v_1),(v_2,v_3)\} \]

\[ G = \]

\[ \begin{array}{c}
 v_1 \\
 v_2 \\
 v_3 \\
 v_4
\end{array} \]
First-order logic restricted to active domain

Formal Semantics of $\text{FO}^{\text{act}}$

$G \models_{\alpha} \exists x \phi$ \iff for some $v \in \text{ACT}(G)$ and $\alpha' = \alpha \cup \{x \mapsto v\}$ we have $G \models_{\alpha'} \phi$

$G \models_{\alpha} \forall x \phi$ \iff for every $v \in \text{ACT}(G)$ and $\alpha' = \alpha \cup \{x \mapsto v\}$ we have $G \models_{\alpha'} \phi$

$G \models_{\alpha} \phi \land \psi$ \iff $G \models_{\alpha} \phi$ and $G \models_{\alpha} \psi$

$G \models_{\alpha} \neg \phi$ \iff it is not true that $G \models_{\alpha} \phi$

$G \models_{\alpha} x=y$ \iff $\alpha(x) = \alpha(y)$

$G \models_{\alpha} E(x,y)$ \iff $(\alpha(x), \alpha(y)) \in E$

$\text{ACT}(G) = \{v \mid \text{for some } v': (v, v') \in E \text{ or } (v', v) \in E\}$
First-order logic restricted to active domain

\[ \text{FO}^{\text{act}} \subseteq \text{RA} \]
First-order logic restricted to active domain

\[ \text{FO}^{\text{act}} \subseteq \text{RA} \]

Assume:

1. \( \phi \) has variables \( x_1, \ldots, x_n \),
2. \( \phi \) in normal form: \((\exists^* (\neg \exists^*)^* + \text{quantifier-free } \psi(x_1, \ldots, x_n))\)

\[ \exists x_1 \exists x_2 \neg \exists x_3 \exists x_4 . \ ( E(x_1,x_3) \land \neg E(x_4,x_2) ) \lor (x_1=x_3) \]
First-order logic restricted to active domain

\[ \text{FO}^{\text{act}} \subseteq \text{RA} \]

Assume:

1. \( \phi \) has variables \( x_1, \ldots, x_n \),
2. \( \phi \) in normal form: \( (\exists^{*}(\neg\exists)^{*}) + \text{quantifier-free } \psi(x_1, \ldots, x_n) \)

\[ \exists x_1 \exists x_2 \neg\exists x_3 \exists x_4 . (E(x_1, x_3) \land \neg E(x_4, x_2)) \lor (x_1 = x_3) \]

\( \text{Adom} = \text{RA expression for active domain} = "\pi_1(E) \cup \pi_2(E)" \)

\begin{itemize}
  \item \((R(x_{i_1}, \ldots, x_{i_n}))^+ \leadsto R \)
  \item \((\exists x_i \phi(x_{i_1}, \ldots, x_{i_n}))^+ \leadsto \pi_{\{i_1, \ldots, i_n\}\{i\}}(\phi^+) \)
  \item \((x_i = x_j)^+ \leadsto \sigma_{\{i=j\}}(\text{Adom} \times \cdots \times \text{Adom}) \)
  \item \((\psi_1(x_{i_1}, \ldots, x_{i_n}) \land \psi_2(x_{i_1}, \ldots, x_{i_n}))^+ \leadsto \psi_1^+ \cap \psi_2^+ \)
  \item \((\neg \phi(x_{i_1}, \ldots, x_{i_n}))^+ \leadsto \text{Adom} \times \cdots \times \text{Adom} \setminus \phi^+ \)
\end{itemize}
First-order logic restricted to active domain

\[ \mathbf{FO^{act}} \subseteq \mathbf{RA} \]

Assume:

1. \( \phi \) has variables \( x_1, \ldots, x_n \),

2. \( \phi \) in normal form: \((\exists^* (\neg \exists)^*)^* + \) quantifier-free \( \psi(x_1, \ldots, x_n) \)

\( \exists x_1 \exists x_2 \neg \exists x_3 \exists x_4 . (E(x_1, x_3) \wedge \neg E(x_4, x_2)) \vee (x_1 = x_3) \)

\( \text{Adom} = \) RA expression for active domain = “\( \pi_1(E) \cup \pi_2(E) \)”

\[ \begin{align*}
    \bullet \ (R(x_{i_1}, \ldots, x_{i_n}))^+ & \leadsto R \\
    \bullet \ (\exists x_i \phi(x_{i_1}, \ldots, x_{i_n}))^+ & \leadsto \pi_{\{i_1, \ldots, i_n\}\{i\}}(\phi^+) \\
    \bullet \ (x_i = x_j)^+ & \leadsto \sigma_{\{i = j\}}(\text{Adom} \times \cdots \times \text{Adom}) \\
    \bullet \ (\psi_1(x_{i_1}, \ldots, x_{i_n}) \wedge \psi_2(x_{i_1}, \ldots, x_{i_n}))^+ & \leadsto \psi_1^+ \cap \psi_2^+ \\
    \bullet \ (\neg \phi(x_{i_1}, \ldots, x_{i_n}))^+ & \leadsto \text{Adom} \times \cdots \times \text{Adom} \setminus \phi^+
\end{align*} \]

Translation

\[ A \cap B = (A \cup B) \setminus A \setminus B \]
$\text{FO}^{\text{act}}$ is equivalent to $\text{RA}$
Question 1: How is $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2))$ expressed in FO? Remember: $R_1, R_2$ are binary

Question 2: How is $\exists y, z . (R_1(x, y) \land R_1(y, z) \land x \neq z)$ expressed in RA? Remember: The signature is the same as before ($R_1, R_2$ binary)

- $R_1 \cup R_2$
- $R_1 \times R_2$
- $R_1 \setminus R_2$
- $\sigma_{\{i_1 \neq j_1, \ldots, i_n \neq j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\}$
- $\pi_{\{i_1, \ldots, i_n\}}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\}$
**Question 1:** How is $\pi_2(\sigma_{1=3}(\pi_2(\sigma_{1=3}(R_1 \times R_2)) \times R_2))$ expressed in FO?

**Remember:** $R_1, R_2$ are binary

**Answer:** $\exists x_2. \left( \exists x_1, x_4. (R_1(x_1, x_2) \land R_2(x_1, x_4)) \land R_2(x_2, x_5) \right)$

**Question 2:** How is $\exists y, z. (R_1(x, y) \land R_1(y, z) \land x \neq z)$ expressed in RA?

**Remember:** The signature is the same as before ($R_1, R_2$ binary)

- $R_1 \cup R_2$
- $R_1 \times R_2$
- $R_1 \setminus R_2$
- $\sigma_{\{i_1 \neq j_1, \ldots, i_n \neq j_n\}}(R) := \{(x_1, \ldots, x_m) \in R \mid (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\}$
- $\pi_{\{i_1, \ldots, i_n\}}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\}$
**Question 1:** How is $\pi_2(\sigma_1=3(\pi_2(\sigma_1=3(R_1 \times R_2)) \times R_2))$ expressed in FO?

**Remember:** $R_1, R_2$ are binary

**Answer:** $\exists x_2 . \left( \exists x_1, x_4 . (R_1(x_1, x_2) \land R_2(x_1, x_4)) \land R_2(x_2, x_5) \right)$

**Question 2:** How is $\exists y, z . (R_1(x, y) \land R_1(y, z) \land x \neq z)$ expressed in RA?

**Remember:** The signature is the same as before ($R_1, R_2$ binary)

- $R_1 \cup R_2$
- $R_1 \times R_2$
- $R_1 \setminus R_2$
- $\sigma_{\{i_1=j_1, \ldots, i_n=j_n\}}(R) := \{(x_1, \ldots, x_m) \in R | (x_{i_1} \neq x_{j_1}) \land \cdots \land (x_{i_n} \neq x_{j_n})\}$
- $\pi_{\{i_1, \ldots, i_n\}}(R) := \{(x_{i_1}, \ldots, x_{i_n}) \mid (x_1, \ldots, x_m) \in R\}$

**Answer:** $\pi_1(\sigma_{\{2=3, 1 \neq 4\}}(R_1 \times R_1))$
Logic = Algebra = Programming language

FO = RA = SQL
Logic = Algebra = Programming language

- FO over active domain
- RA on finite domains
- SQL very basic
Evaluation problem: Given a query \( Q \), a database instance \( db \), and a tuple \( t \), is \( t \in Q(db) \) ?

\( \Rightarrow \) How hard is it to retrieve data?
Algorithmic problems for query languages

**Evaluation problem**: Given a query $Q$, a database instance $db$, and a tuple $t$, is $t \in Q(db)$?

→ How hard is it to retrieve data?

**Emptiness problem**: Given a query $Q$, is there a database instance $db$ so that $Q(db) \neq \emptyset$?

→ Does $Q$ make sense? Is it a contradiction? (Query optimization)
Algorithmic problems for query languages

**Evaluation problem:** Given a query $Q$, a database instance $db$, and a tuple $t$, is $t \in Q(db)$?

→ How hard is it to retrieve data?

**Emptiness problem:** Given a query $Q$, is there a database instance $db$ so that $Q(db) \neq \emptyset$?

→ Does $Q$ make sense? Is it a contradiction? (Query optimization)

**Equivalence problem:** Given queries $Q_1$, $Q_2$, is $Q_1(db) = Q_2(db)$ for all database instances $db$?

→ Can we safely replace a query with another? (Query optimization)
Complexity theory

What can be **mechanized?** \(\sim\) **decidable/undecidable**

How **hard** is it to mechanise? \(\sim\) **complexity classes**
Complexity theory

What can be **mechanized**? $\sim$ decidable/undecidable

How **hard** is it to mechanise? $\sim$ complexity classes
Complexity theory

What can be **mechanized**?  \(\sim\) **decidable/undecidable**

How **hard** is it to mechanise?  \(\sim\) **complexity classes**

- usage of resources:
  - time
  - memory
Complexity theory

What can be **mechanized**?  $\sim$ decidable/undecidable

How **hard** is it to mechanise?  $\sim$ complexity classes

- usage of resources:  
  - time
  - memory

Algorithm $\text{Alg}$ is **TIME**-bounded by a function $f: \mathbb{N} \rightarrow \mathbb{N}$ if $\text{Alg}(\text{input})$ uses less than $f(|\text{input}|)$ units of TIME.
What can be **mechanized**? \( \leadsto \) decidable/undecidable

How **hard** is it to mechanise? \( \leadsto \) complexity classes

- usage of resources:
  - time
  - memory

Algorithm \( \text{Alg} \) is **TIME**-bounded by a function \( f: \mathbb{N} \rightarrow \mathbb{N} \) if

\( \text{Alg}(\text{input}) \) uses less than \( f(|\text{input}|) \) units of **TIME**.
Complexity theory

What can be **mechanized**? \(\sim\) decidable/undecidable

How **hard** is it to mechanise? \(\sim\) complexity classes

---

**Algorithm** \(\text{Alg}\) is **TIME**-bounded by a function \(f: \mathbb{N} \rightarrow \mathbb{N}\) if \(\text{Alg}(\text{input})\) uses less than \(f(|\text{input}|)\) units of **TIME**.

**SPACE**

\[ f \]
Complexity theory

What can be **mechanized**? \(\approx\) decidable/undecidable

How **hard** is it to mechanise? \(\approx\) complexity classes

usage of resources: • time • memory

Algorithm \(\text{Alg}\) is \(\text{TIME}\)-bounded by a function \(f: \mathbb{N} \rightarrow \mathbb{N}\) if \(\text{Alg}(\text{input})\) uses less than \(f(|\text{input}|)\) units of \(\text{TIME}\).

\[
\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \ldots
\]
Complexity theory

What can be **mechanized**? \( \sim \) decidable/undecidable

How **hard** is it to mechanise? \( \sim \) complexity classes

usage of resources:
- time
- memory

Algorithm \( \text{Alg} \) is **TIME**-bounded by a function \( f : \mathbb{N} \rightarrow \mathbb{N} \) if \( \text{Alg}(\text{input}) \) uses less than \( f(|\text{input}|) \) units of **TIME**.

**SPACE**-bounded by a polynomial

\( \text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \cdots \)

**SPACE**-bounded by a polynomial

**SPACE**-bounded by \( \log(n) \)
Algorithmic problems for FO

**Evaluation problem:** Given a FO formula $\phi(x_1, ..., x_n)$, a graph $G$, and a binding $\alpha$, does $G \models_\alpha \phi$?

**Satisfiability problem:** Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models_\alpha \phi$?

**Equivalence problem:** Given FO formulae $\phi, \psi$, is $G \models_\alpha \phi$ iff $G \models_\alpha \psi$ for all graphs $G$ and bindings $\alpha$?
Algorithmic problems for FO

**Evaluation problem:** Given a FO formula $\phi(x_1, ..., x_n)$, a graph $G$, and a binding $\alpha$, does $G \models_\alpha \phi$?

**Satisfiability problem:** Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models_\alpha \phi$?

**Equivalence problem:** Given FO formulae $\phi, \psi$, is $G \models_\alpha \phi$ iff $G \models_\alpha \psi$ for all graphs $G$ and bindings $\alpha$?

DECIDABLE $\iff$ foundations of the database industry
Algorithmic problems for FO

**Evaluation problem:** Given a FO formula $\phi(x_1, ..., x_n)$, a graph $G$, and a binding $\alpha$, does $G \models_\alpha \phi$?

**Satisfiability problem:** Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models_\alpha \phi$?

**UNDECIDABLE ⇝ both for $\models$ and $\models_{\text{finite}}$**

**Equivalence problem:** Given FO formulae $\phi, \psi$, is $G \models_\alpha \phi$ iff $G \models_\alpha \psi$ for all graphs $G$ and bindings $\alpha$?

DECIDABLE ⇝ foundations of the database industry
Algorithmic problems for FO

**Evaluation problem:** Given a FO formula $\phi(x_1, ..., x_n)$, a graph $G$, and a binding $\alpha$, does $G \models^\alpha \phi$?

DECIDABLE $\iff$ foundations of the database industry

**Satisfiability problem:** Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models^\alpha \phi$?

💀 UNDECIDABLE $\iff$ both for $\models$ and $\models_{finite}$

**Equivalence problem:** Given FO formulae $\phi, \psi$, is $G \models^\alpha \phi$ iff $G \models^\alpha \psi$ for all graphs $G$ and bindings $\alpha$?

💀 UNDECIDABLE $\iff$ by reduction to the satisfiability problem
Algorithmic problems for FO

Satisfiability problem: Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models \alpha \phi$?

💀 **UNDECIDABLE** $\iff$ both for $\models$ and $\models_{\text{finite}}$  [Trakhtenbrot ’50]
Satisfiability problem: Given a FO formula \( \phi \), is there a graph \( G \) and binding \( \alpha \), such that \( G \vDash_{\alpha} \phi \) ?

💀 **UNDECIDABLE** \( \iff \) both for \( \vDash \) and \( \vDash_{\text{finite}} \) [Trakhtenbrot ’50]

Proof: By reduction from the Domino (aka Tiling) problem.
Algorithmic problems for FO

Satisfiability problem: Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \vDash \alpha \phi$?

💀 UNDECIDABLE $\iff$ both for $\not\vDash$ and $\vDash_{\text{finite}}$ [Trakhtenbrot ’50]

Proof: By reduction from the Domino (aka Tiling) problem.

Reduction from $P$ to $P'$: Algorithm that solves $P$ using a $O(1)$ procedure 
" $P'(x)$ " that returns the truth value of $P'(x)$. 
The (undecidable) Domino problem

Input: 4-sided dominos:

- [Image of 4-sided dominos]
The (undecidable) Domino problem

**Input:** 4-sided dominos:  

**Output:** Is it possible to form a white-bordered rectangle? (of any size)
The (undecidable) Domino problem

DOMINO

Input: 4-sided dominos:

Output: Is it possible to form a white-bordered rectangle? (of any size)

Rules: sides must match,
you can’t rotate the dominos, but you can ‘clone’ them.
The (undecidable) Domino problem

Domino - Why is it undecidable?

It can easily encode *halting* computations of Turing machines:

```
0 0 q q q0 0
0 l1 l1 l 0 0
0 0 q q q0 0
0 1 1 r0 r0 0
0 1 1 r r q0 0
0 q q q q q q q q
0 0 q q q q q q q
0 l1 l1 l 0 0
0 0 q q q q q q q
0 0 q q q q q q q
```

...
The (undecidable) Domino problem

**Domino - Why is it undecidable?**

It can easily encode *halting* computations of Turing machines:

(head is elsewhere, symbol is not modified)
The (undecidable) Domino problem

Domino - Why is it undecidable?

It can easily encode *halting* computations of Turing machines:

- (head is elsewhere, symbol is not modified)
- (head is here, symbol is rewritten, head moves right)
The (undecidable) Domino problem

Domino - Why is it undecidable?

It can easily encode *halting* computations of Turing machines:

- (head is elsewhere, symbol is not modified)
- (head is here, symbol is rewritten, head moves right)
- (head is here, symbol is rewritten, head moves left)
The (undecidable) Domino problem

**Domino - Why is it undecidable?**

It can easily encode *halting* computations of Turing machines:

- (head is elsewhere, symbol is not modified)
- (head is here, symbol is rewritten, head moves right)
- (head is here, symbol is rewritten, head moves left)
- (initial configuration)
The (undecidable) Domino problem

Domino - Why is it undecidable?

It can easily encode halting computations of Turing machines:

- Initial configuration
- Halting configuration

- Head is elsewhere, symbol is not modified
- Head is here, symbol is rewritten, head moves right
- Head is here, symbol is rewritten, head moves left
Domino $\iff$ Sat-FO (domino has a solution iff $\phi$ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...
1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...
1. There is a grid: $H(\ ,\ )$ and $V(\ ,\ )$ are relations representing bijections such that...
1. There is a grid: $H(\cdot, \cdot)$ and $V(\cdot, \cdot)$ are relations representing bijections such that...
Domino $\leftrightarrow$ Sat-FO (domino has a solution iff $\phi$ satisfiable)

1. There is a grid: $H(,)$ and $V(,)$ are relations representing bijections such that...

2. Assign one domino to each node: a unary relation

$$D_x(x)$$

for each domino $\boxed{\times}$
Domino $\iff$ Sat-FO (domino has a solution iff $\phi$ satisfiable)

1. There is a grid: $H(\ ,\ )$ and $V(\ ,\ )$ are relations representing bijections such that...

2. Assign one domino to each node:
   a unary relation
   
   $D_a(x)$

   for each domino

3. Match the sides $\forall x,y$
   if $H(x,y)$, then $D_a(x) \land D_b(y)$
   for some dominos $a,b$ that ‘match’ horizontally (Idem vertically)
Domino \( \iff \) Sat-FO (domino has a solution iff \( \phi \) satisfiable)

1. There is a grid: \( H( , ) \) and \( V( , ) \) are relations representing bijections such that...

2. Assign one domino to each node: a unary relation

\[
D_x(x)
\]

for each domino \( \Box \)

3. Match the sides \( \forall x,y \)

if \( H(x,y) \), then \( D_a(x) \land D_b(y) \)

for some dominos \( a,b \) that ‘match’ horizontally \( (\text{Idem vertically}) \)

4. Borders are white.
Algorithmic problems for FO

**Evaluation problem:** Given a FO formula $\phi(x_1, \ldots, x_n)$, a graph $G$, and a binding $\alpha$, does $G \models_\alpha \phi$?

**Satisfiability problem:** Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models_\alpha \phi$?

**Equivalence problem:** Given FO formulae $\phi, \psi$, is $G \models_\alpha \phi$ iff $G \models_\alpha \psi$ for all graphs $G$ and bindings $\alpha$?

---

**DECIDABLE $\iff$** foundations of the database industry

**UNDECIDABLE $\iff$** by reduction to the satisfiability problem

💀 UNDECIDABLE $\iff$ both for $\models$ and $\models_{\text{finite}}$
Algorithmic problems for FO

Equivalence problem: Given FO formulae $\phi, \psi$, is

$$G \vDash_\alpha \phi \iff G \vDash_\alpha \psi$$

for all graphs $G$ and bindings $\alpha$?

💀 UNDECIDABLE $\iff$ by reduction from the satisfiability problem
Algorithmic problems for FO

φ is **satisfiable** iff φ is **not equivalent** to ⊥

Satisfiability problem undecidable \(\iff\) Equivalence problem undecidable

---

**Equivalence problem**: Given FO formulae \(\phi, \psi\), is

\[ G \models_{\alpha} \phi \iff G \models_{\alpha} \psi \]

for all graphs G and bindings \(\alpha\)?

💀 **UNDECIDABLE** \(\iff\) by reduction from the satisfiability problem
Algorithmic problems for FO

\[ \phi \text{ is satisfiable } \iff \phi \text{ is not equivalent to } \bot \]

Satisfiability problem undecidable \( \Rightarrow \) Equivalence problem undecidable

Actually, there are reductions in both senses:

\[ \phi(x_1,\ldots,x_n) \text{ and } \psi(y_1,\ldots,y_m) \text{ are equivalent } \iff \]

- \( n=m \)
- \( (x_1=y_1) \land \cdots \land (x_n=y_n) \land \phi(x_1,\ldots,x_n) \land \neg \psi(y_1,\ldots,y_n) \) is unsatisfiable
- \( (x_1=y_1) \land \cdots \land (x_n=y_n) \land \psi(x_1,\ldots,x_n) \land \neg \phi(y_1,\ldots,y_n) \) is unsatisfiable

Equivalence problem: Given FO formulae \( \phi, \psi \), is

\[ G \vDash_{\alpha} \phi \iff G \vDash_{\alpha} \psi \]

for all graphs \( G \) and bindings \( \alpha \) ?

\( \blackice \text{ UNDECIDABLE } \Rightarrow \) by reduction from the satisfiability problem
Algorithmic problems for FO

**Evaluation problem:** Given a FO formula $\phi(x_1, \ldots, x_n)$, a graph $G$, and a binding $\alpha$, does $G \models_\alpha \phi$?

DECIDABLE $\iff$ foundations of the database industry

**Satisfiability problem:** Given a FO formula $\phi$, is there a graph $G$ and binding $\alpha$, such that $G \models_\alpha \phi$?

💀 UNDECIDABLE $\iff$ both for $\models$ and $\models_{finite}$

**Equivalence problem:** Given FO formulae $\phi, \psi$, is $G \models_\alpha \phi$ iff $G \models_\alpha \psi$ for all graphs $G$ and bindings $\alpha$?

💀 UNDECIDABLE $\iff$ by reduction to the satisfiability problem
Evaluation problem for FO

Input:

\[ \phi(x_1, \ldots, x_n) \]

\[ G = (V, E) \]

\[ \alpha = \{x_1, \ldots, x_n\} \rightarrow V \]

Output:

\[ G \vDash_\alpha \phi \, ? \]
Evaluation problem for FO

\[
\phi(x_1,\ldots,x_n)
\]

\[
G = (V,E)
\]

\[
\alpha = \{x_1,\ldots,x_n\} \longrightarrow V
\]

Input: 

\[
\begin{align*}
\phi(x_1,\ldots,x_n) \\
G = (V,E) \\
\alpha = \{x_1,\ldots,x_n\} \longrightarrow V
\end{align*}
\]

Output: 

\[
G \models \phi \ ?
\]

Encoding of \(G = (V, E)\)

- each node is coded with a bit string of size \(\log(|V|)\),
- edge set is encoded by its tuples, e.g. \((100,101), (010, 010), \ldots\)

Cost of coding: \(||G|| = |E| \cdot 2 \cdot \log(|V|) \approx |V| \text{ (mod a polynomial)}\)
Evaluation problem for FO

Input: \[ \phi(x_1, \ldots, x_n) \]
\[ G = (V, E) \]
\[ \alpha = \{x_1, \ldots, x_n\} \rightarrow V \]

Output: \[ G \models_\alpha \phi ? \]

Encoding of \( G = (V, E) \)

- each node is coded with a bit string of size \( \log(|V|) \),
- edge set is encoded by its tuples, e.g. \((100,101), (010, 010), \ldots\)

Cost of coding: \[ ||G|| = |E| \cdot 2 \cdot \log(|V|) \approx |V| \pmod{\text{a polynomial}} \]

Encoding of \( \alpha = \{x_1, \ldots, x_n\} \rightarrow V \)

- each node is coded with a bit string of size \( \log(|V|) \),

Cost of coding: \[ ||\alpha|| = n \cdot \log(|V|) \]
Input: \[ \phi(x_1, \ldots, x_n) \]
\[ G = (V, E) \]
\[ \alpha = \{x_1, \ldots, x_n\} \rightarrow V \]

Output: \[ G \models_{\alpha} \phi ? \]
Evaluation problem for FO

Input: \( \phi(x_1, \ldots, x_n) \)
\[ G = (V, E) \]
\[ \alpha = \{x_1, \ldots, x_n\} \rightarrow V \]

Output: \( G \models_\alpha \phi \)?

- If \( \phi(x_1, \ldots, x_n) = E(x_i, x_j) \):
  answer YES if \( (\alpha(x_i), \alpha(x_j)) \in E \)

- If \( \phi(x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n) \land \psi'(x_1, \ldots, x_n) \):
  answer YES if \( G \models_\alpha \psi \) and \( G \models_\alpha \psi' \)

- If \( \phi(x_1, \ldots, x_n) = \neg \psi(x_1, \ldots, x_n) \):
  answer NO if \( G \models_\alpha \psi \)

- If \( \phi(x_1, \ldots, x_n) = \exists y \cdot \psi(x_1, \ldots, x_n, y) \):
  answer YES if for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \)
  we have \( G \models_{\alpha'} \psi \).
Evaluation problem for FO

Input: \( \phi(x_1, \ldots, x_n) \)
\( G = (V, E) \)
\( \alpha = \{x_1, \ldots, x_n\} \rightarrow V \)

Output: \( G \vDash_\alpha \phi \)?

- If \( \phi(x_1, \ldots, x_n) = E(x_i, x_j) \):
  answer YES iff \( (\alpha(x_i), \alpha(x_j)) \in E \)

- If \( \phi(x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n) \land \psi'(x_1, \ldots, x_n) \):
  answer YES iff \( G \vDash_\alpha \psi \) and \( G \vDash_\alpha \psi' \)

- If \( \phi(x_1, \ldots, x_n) = \neg \psi(x_1, \ldots, x_n) \):
  answer NO iff \( G \vDash_\alpha \psi \)

- If \( \phi(x_1, \ldots, x_n) = \exists y . \psi(x_1, \ldots, x_n, y) \):
  answer YES iff for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \) we have \( G \vDash_{\alpha'} \psi \).

**Question:**
How much space does it take?
Evaluation problem for FO

Input: \( \phi(x_1, \ldots, x_n) \)
\( G = (V, E) \)
\( \alpha = \{x_1, \ldots, x_n\} \rightarrow V \)

Output: \( G \vDash_{\alpha} \phi ? \)

- If \( \phi(x_1, \ldots, x_n) = E(x_i, x_j) \):
  answer YES iff \( (\alpha(x_i), \alpha(x_j)) \in E \)

- If \( \phi(x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n) \land \psi'(x_1, \ldots, x_n) \):
  answer YES iff \( G \vDash_{\alpha} \psi \) and \( G \vDash_{\alpha} \psi' \)

- If \( \phi(x_1, \ldots, x_n) = \neg \psi(x_1, \ldots, x_n) \):
  answer NO iff \( G \vDash_{\alpha} \psi \)

- If \( \phi(x_1, \ldots, x_n) = \exists y . \psi(x_1, \ldots, x_n, y) \):
  answer YES iff for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \)
  we have \( G \vDash_{\alpha'} \psi \).

**Question:** How much space does it take?
Evaluation problem for FO

Input: \[
\begin{pmatrix}
\phi(x_1, \ldots, x_n) \\
G = (V, E) \\
\alpha = \{x_1, \ldots, x_n\} \rightarrow V
\end{pmatrix}
\]

Output: \( G \vDash_\alpha \phi ? \)

- If \( \phi(x_1, \ldots, x_n) = E(x_i, x_j) \):
  answer YES iff \((\alpha(x_i), \alpha(x_j)) \in E \)

- If \( \phi(x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n) \land \psi'(x_1, \ldots, x_n) \):
  answer YES iff \( G \vDash_\alpha \psi \) and \( G \vDash_\alpha \psi' \)

- If \( \phi(x_1, \ldots, x_n) = \neg \psi(x_1, \ldots, x_n) \):
  answer NO iff \( G \vDash_\alpha \psi \)

- If \( \phi(x_1, \ldots, x_n) = \exists y . \psi(x_1, \ldots, x_n, y) \):
  answer YES iff for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \)
  we have \( G \vDash_{\alpha'} \psi \).

Question:
How much space does it take?

use 4 pointers \( \Rightarrow \) LOGSPACE

\( \Rightarrow \) MAX( SPACE( \( G \vDash_\alpha \psi \)), SPACE( \( G \vDash_\alpha \psi' \)) )
Evaluation problem for FO

Input: \[ \begin{align*} 
\phi(x_1,\ldots,x_n) \\
G = (V,E) \\
\alpha = \{x_1,\ldots,x_n\} \rightarrow V 
\end{align*} \]

Output: \[ G \models_\alpha \phi ? \]

- If \( \phi(x_1,\ldots,x_n) = E(x_i,x_j) \):
  answer YES iff \( (\alpha(x_i),\alpha(x_j)) \in E \)
  \( \implies \) use 4 pointers \( \Rightarrow \) LOGSPACE

- If \( \phi(x_1,\ldots,x_n) = \psi(x_1,\ldots,x_n) \land \psi'(x_1,\ldots,x_n) \):
  answer YES iff \( G \models_\alpha \psi \) and \( G \models_\alpha \psi' \)
  \( \implies \) \( \text{MAX}( \text{SPACE}(G \models_\alpha \psi), \text{SPACE}(G \models_\alpha \psi')) \) \( \Rightarrow \text{SPACE}(G \models_\alpha \psi) \)

- If \( \phi(x_1,\ldots,x_n) = \neg \psi(x_1,\ldots,x_n) \):
  answer NO iff \( G \models_\alpha \psi \)

- If \( \phi(x_1,\ldots,x_n) = \exists y . \psi(x_1,\ldots,x_n,y) \):
  answer YES iff for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \)
  we have \( G \models_{\alpha'} \psi \).

Question:
How much space does it take?
Evaluation problem for FO

Input: \[ \phi(x_1,\ldots,x_n) \]
\[ G = (V,E) \]
\[ \alpha = \{x_1,\ldots,x_n\} \longrightarrow V \]

Output: \[ G \models_\alpha \phi ? \]

- If \( \phi(x_1,\ldots,x_n) = E(x_i,x_j) \):
  answer YES iff \((\alpha(x_i),\alpha(x_j)) \in E\)
  use 4 pointers \( \mapsto \) LOGSPACE

- If \( \phi(x_1,\ldots,x_n) = \psi(x_1,\ldots,x_n) \land \psi'(x_1,\ldots,x_n) \):
  answer YES iff \( G \models_\alpha \psi \) and \( G \models_\alpha \psi' \)
  \( \mapsto \) MAX( SPACE(\( G \models_\alpha \psi \)), SPACE(\( G \models_\alpha \psi' \)) )

- If \( \phi(x_1,\ldots,x_n) = \neg \psi(x_1,\ldots,x_n) \):
  answer NO iff \( G \models_\alpha \psi \)
  \( \mapsto \) SPACE(\( G \models_\alpha \psi \))

- If \( \phi(x_1,\ldots,x_n) = \exists y . \psi(x_1,\ldots,x_n,y) \):
  answer YES iff for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \)
  we have \( G \models_{\alpha'} \psi \)
  \( \mapsto \) 2\cdot\log(|G|) + SPACE(\( G \models_{\alpha'} \psi \))

Question:
How much space does it take?
Evaluation problem for FO

Input: \( \phi(x_1, \ldots, x_n) \)
\[ G = (V, E) \]
\[ \alpha = \{x_1, \ldots, x_n\} \rightarrow V \]

Output: \( G \models_{\alpha} \phi ? \)

• If \( \phi(x_1, \ldots, x_n) = E(x_i, x_j) \):
  answer YES iff \((\alpha(x_i), \alpha(x_j)) \in E\)

• If \( \phi(x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n) \land \psi'(x_1, \ldots, x_n) \):
  answer YES iff \( G \models_{\alpha} \psi \) and \( G \models_{\alpha} \psi' \)

• If \( \phi(x_1, \ldots, x_n) = \neg \psi(x_1, \ldots, x_n) \):
  answer NO iff \( G \models_{\alpha} \psi \)

• If \( \phi(x_1, \ldots, x_n) = \exists y . \psi(x_1, \ldots, x_n, y) \):
  answer YES iff for some \( v \in V \) and \( \alpha' = \alpha \cup \{y \mapsto v\} \)
  we have \( G \models_{\alpha'} \psi \).

Question:
How much space does it take?

\[ 2 \cdot \log(|G|) + \cdots + 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|) \text{ space} \]
\[ \leq |\phi| \text{ times} \]
Evaluation problem for FO \textbf{in PSPACE}

Input: \[
\begin{align*}
\phi(x_1,\ldots,x_n) \\
G = (V,E) \\
\alpha = \{x_1,\ldots,x_n\} \rightarrow V
\end{align*}
\]

Output: \(G \vDash_\alpha \phi ?\)

- If \(\phi(x_1,\ldots,x_n) = E(x_i,x_j)\):
  answer YES iff \((\alpha(x_i),\alpha(x_j)) \in E\)

- If \(\phi(x_1,\ldots,x_n) = \psi(x_1,\ldots,x_n) \land \psi'(x_1,\ldots,x_n)\):
  answer YES iff \(G \vDash_\alpha \psi\) and \(G \vDash_\alpha \psi'\)

- If \(\phi(x_1,\ldots,x_n) = \neg \psi(x_1,\ldots,x_n)\):
  answer NO iff \(G \vDash_\alpha \neg \psi\)

- If \(\phi(x_1,\ldots,x_n) = \exists y . \psi(x_1,\ldots,x_n,y)\):
  answer YES iff for some \(v \in V\) and \(\alpha' = \alpha \cup \{y \mapsto v\}\)
  we have \(G \vDash_\alpha' \psi\).

Question: How much space does it take?

\[2 \cdot \log(|G|) + \cdots + 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|)\]

\(\leq |\phi|\) times
Problem: Usual scenario in database

Input:

A database of size $10^6$
A query of size 100
Problem: Usual scenario in database

A database of size $10^6$

A query of size 100

Input: query +
Problem: Usual scenario in database

Input:  • query  +
Problem: Usual scenario in database

Input: • query +

But we don’t distinguish this in the analysis:

\[
\text{TIME}(2|\text{query}| + |\text{data}|) = \text{TIME}(|\text{query}| + 2|\text{data}|)
\]
Query and data play very different roles.

Separation of concerns: How the resources grow with respect to

- the size of the data
- the query size
Combined complexity: input size is $|\text{query}| + |\text{data}|$

Query complexity ($|\text{data}|$ fixed): input size is $|\text{query}|$

Data complexity ($|\text{query}|$ fixed): input size is $|\text{data}|$
Combined complexity: input size is $|\text{query}| + |\text{data}|$

Query complexity ($|\text{data}|$ fixed): input size is $|\text{query}|$

Data complexity ($|\text{query}|$ fixed): input size is $|\text{data}|$

$O(2^{|\text{query}|} + |\text{data}|)$ is exponential in combined complexity

$O(|\text{query}| + 2^{|\text{data}|})$ is exponential in combined complexity

$O(|\text{query}| + 2^{|\text{data}|})$ is linear in query complexity

$O(2^{|\text{query}|} + |\text{data}|)$ is linear in data complexity
Question

What is the data, query and combined complexity for the evaluation problem for FO?

Remember:

**data** complexity, input size: $|\text{data}|$

**query** complexity, input size: $|\text{query}|$

**combined** complexity, input size: $|\text{data}| + |\text{query}|$

$|\phi| \cdot 2 \cdot \log(|G|) + k \cdot \log(|\alpha| + |G|)$ space
Question

What is the data, query and combined complexity for the evaluation problem for FO?

Remember:  
- **data** complexity, input size: $|\text{data}|$
- **query** complexity, input size: $|\text{query}|$
- **combined** complexity, input size: $|\text{data}| + |\text{query}|$

$$|\phi| \cdot 2 \cdot \log(|G|) + k \cdot \log(|\alpha|+|G|) \text{ space}$$

$O(\log(|\text{data}|) \cdot |\text{query}|) \text{ space}$

**PSPACE** combined and query complexity

**LOGSPACE** data complexity
Evaluation pb for FO is PSPACE-complete

PSPACE-complete problem: **QBF**
(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)
Evaluation pb for FO is PSPACE-complete (combined complexity)

PSPACE-complete problem: **QBF**

(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

\[ \exists p \ \forall q \ . \ (p \lor \neg q) \ \text{where } p,q \text{ range over } \{T,F\} \]
Theorem: Evaluation for FO is PSPACE-complete (combined c.)

PSPACE-complete problem: QBF (satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

$$\exists p \ \forall q . \ (p \lor \neg q)$$ where p,q range over \{T,F\}
Evaluation pb for FO is PSPACE-complete (combined complexity)

PSPACE-complete problem: **QBF**
(satisfaction of Quantified Boolean Formulas)

QBF = a boolean formula with quantification over the truth values (T,F)

\[ \exists p \forall q . (p \lor \neg q) \]  where p,q range over \{T,F\}

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

Polynomial reduction **QBF \sim FO** :

1. Given \( \psi \in QBF \),
   let \( \psi'(x) \) be the replacement of each ‘p’ with ‘p=x’ in \( \psi \).

2. Note: \( \exists x \psi' \) holds in a 2-element graph iff \( \psi \) is QBF-satisfiable

3. Test if \( G \vDash_\emptyset \psi' \) for \( G=(\{v,v'\},\{\}) \)
Evaluation pb for FO is PSPACE-complete

PSPACE-complete problem: $\text{QBF}$
(satisfaction of Quantified Boolean Formulas)

$\text{QBF} = \text{a boolean formula with quantification over the truth values (T,F)}$

$$\exists p \forall q . (p \lor \neg q)$$
where $p, q$ range over \{T, F\}

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

Polynomial reduction $\text{QBF} \sim \text{FO}$:

$$\psi'(x) = \exists p \forall q . ((p=x) \lor \neg (q=x))$$

1. Given $\psi \in \text{QBF}$, let $\psi'(x)$ be the replacement of each ‘$p$’ with ‘$p=x$’ in $\psi$.

2. Note: $\exists x \psi'$ holds in a 2-element graph iff $\psi$ is QBF-satisfiable

3. Test if $G \vDash \psi'$ for $G=\{v, v'\}, \{\}$
Evaluation pb for FO is PSPACE-complete (combined complexity)

PSPACE-complete problem: $\text{QBF}$

(satisfaction of Quantified Boolean Formulas)

$\text{QBF} = \text{a boolean formula with quantification over the truth values (T,F)}$

$$\exists p \forall q . (p \lor \neg q) \quad \text{where p,q range over \{T,F\}}$$

Theorem: Evaluation for FO is PSPACE-complete (combined c.)

Polynomial reduction $\text{QBF} \sim \text{FO}$:

1. Given $\psi \in \text{QBF}$, let $\psi'(x)$ be the replacement of each ‘p’ with ‘p=x’ in $\psi$.

2. Note: $\exists x \psi'$ holds in a 2-element graph iff $\psi$ is QBF-satisfiable.

3. Test if $G \vDash \psi'$ for $G=(\{v,v'\},\{}$)

(freely available at http://webdam.inria.fr/Alice/)

Chapters 1, 2, 3