TECHNISCHE UNIVERSITÄT DRESDEN

# DATABASE THEORY

## Lecture 9: First-Order Expressiveness / Introduction to Datalog

**Markus Krötzsch**

TU Dresden, 9 June 2016

# Overview

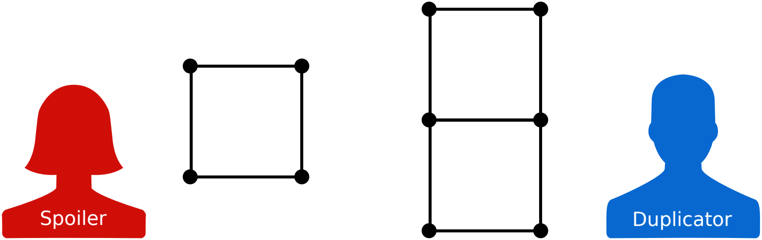


See course homepage [⇒ link] for more information and materials

# Review: EF Games

Ehrenfeucht-Fraïssé games characterise expressivity of FO formulas:

- the quantifier rank needed to distinguish structure corresponds to
- the number of rounds needed by Spoiler to win the game



# Using EF Games to Show FO-Undefinability

How to show that a query mapping $M$ can not be FO-defined:

- Let $C_M$ be the class of all databases recognised by $M$
- Find sequences of databases $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \ldots \in C_M$ and databases $\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \ldots \notin C_M$, such that $\mathcal{I}_i \sim_i \mathcal{J}_i$

$\rightsquigarrow$ for any formula $\varphi$ (however large its quantifier rank $r$), there is a counterexample $\mathcal{I}_r \in C_M$ and $\mathcal{J}_r \notin C_M$ that $\varphi$ cannot distinguish

Problems:

- How to find such sequences of $\mathcal{I}_i$ and $\mathcal{J}_i$?
  $\rightsquigarrow$ No general strategy exists
- Given suitable sequences, how to show that $\mathcal{I}_i \sim_i \mathcal{J}_i$?
  $\rightsquigarrow$ Can be difficult, but doable for some special cases

# Expressiveness on Linear Orders

Let's look at some very simple structures:

**Definition**

A structure $\mathcal{I}$ is a linear order if it has a single binary predicate $\leq$ interpreted as a total, transitive, reflexive and asymmetric relation.

Example:

$\mathcal{L}_6 : 1 \leq 2 \leq 3 \leq 4 \leq 5 \leq 6$

$\mathcal{L}_7 : 1 \leq 2 \leq 3 \leq 4 \leq 5 \leq 6 \leq 7$

Spoiler can win the 3-round EF game:

Spoiler plays $4$ in $\mathcal{L}_7$
- Duplicator plays $4$ in $\mathcal{L}_6$: Spoiler plays $6$ in $\mathcal{L}_7$
  - Duplicator plays $5$ in $\mathcal{L}_6$: Spoiler plays $5$ in $\mathcal{L}_7$ and wins
  - Duplicator plays $6$ in $\mathcal{L}_6$: Spoiler plays $7$ in $\mathcal{L}_7$ and wins
- Duplicator plays $3$ in $\mathcal{L}_6$: symmetric game (flipped horizontally)

# Expressiveness on Linear Orders

Let's look at some very simple structures:

**Definition**

A structure $\mathcal{I}$ is a linear order if it has a single binary predicate $\leq$ interpreted as a total, transitive, reflexive and asymmetric relation.

Example:

$\mathcal{L}_7 : 1 \leq 2 \leq 3 \leq 4 \leq 5 \leq 6 \leq 7$

$\mathcal{L}_8 : 1 \leq 2 \leq 3 \leq 4 \leq 5 \leq 6 \leq 7 \leq 8$

Spoiler cannot win the 3-round EF game:

Spoiler plays $4$ in $\mathcal{L}_8$: Duplicator plays $4$ in $\mathcal{L}_7$
- Spoiler plays $6$ in $\mathcal{L}_8$: Duplicator plays $6$ in $\mathcal{L}_7$; spoiler cannot win
- Spoiler plays $7$ in $\mathcal{L}_8$: Duplicator plays $6$ in $\mathcal{L}_7$; spoiler cannot win

Other cases similar: Spoiler never wins

**Theorem**

The following are equivalent:

- $\mathcal{L}_m \sim_r \mathcal{L}_n$
- either (1) $m = n$, or (2) $m \geq 2^r - 1$ and $n \geq 2^r - 1$

Proof: see board

# FO-Definability of Parity

## Theorem

Parity is not FO-definable for linear orders, hence it is not FO-definable for arbitrary databases.

Proof:

- Suppose for a contradiction that Parity is FO-definable by some query $\varphi$.
- Let $r$ be the quantifier rank of $\varphi$.
- Consider databases $\mathcal{L}_m$ and $\mathcal{L}_n$ with $m = 2^r$ and $n = 2^r + 1$.
- We know that $\mathcal{L}_m \sim_r \mathcal{L}_n$, and therefore $\mathcal{L}_m \equiv_r \mathcal{L}_n$.
- Hence, $\mathcal{L}_m \models \varphi$ if and only if $\mathcal{L}_n \models \varphi$.
- But $\mathcal{L}_m \in \text{Parity}$ while $\mathcal{L}_n \notin \text{Parity}$.
- Therefore, $\varphi$ does not FO-define Parity. Contradiction.

# FO-Definability of Connectivity

The Connectivity problem over finite graphs is as follows:

- Input: A finite graph (relational structure with one binary relation "edge")
- Output: "true" if there is an (undirected) path between any pair of vertices

**Theorem**

Connectivity is not FO-definable.

Proof:

- Suppose for a contradiction that Connectivity is FO-definable using a query $\varphi$.
- We show that this would make Parity FO-definable on linear orders.
- For a linear order $\mathcal{L}$ with order predicate $\leq$, we define a finite graph $\mathcal{G}(\mathcal{L})$ over a binary predicate "edge" such that $\mathcal{G}(\mathcal{L})$ is connected if and only if $\mathcal{L}$ has an even number of elements.

# Defining a Graph From a Linear Order

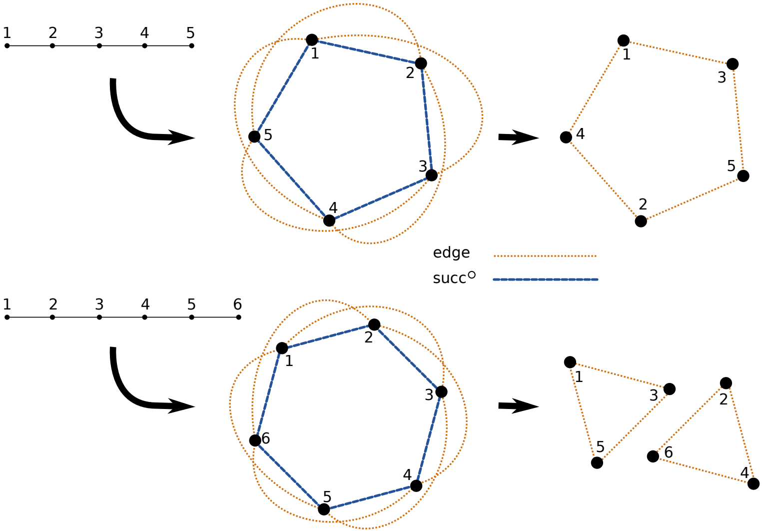
We use abbreviations for the following FO formulas:

$$\begin{aligned}
\text{succ}[x, y] &= (x \leq y) \land \neg(y \leq x) \land && y \text{ is the successor of } x \\
&\quad \forall z.(z \leq x \lor y \leq z) && \\
\text{min}[x] &= \forall z.x \leq z && x \text{ is the first element} \\
\text{max}[x] &= \forall z.z \leq x && x \text{ is the last element} \\
\text{succ}^\circ[x, y] &= \text{succ}[x, y] \lor (\text{max}[x] \land \text{min}[y]) && \text{circular version of succ}
\end{aligned}$$

We now define the formula $\psi$ that derives edges from a linear order:

$$\forall x, y.\text{edge}(x, y) \leftrightarrow \exists z.\text{succ}^\circ[x, z] \land \text{succ}^\circ[z, y]$$

# Illustration: Graphs From Linear Orders



# Completing the Proof

Observation:
The graph $\mathcal{G}(\mathcal{L})$ is connected if and only if $\mathcal{L}$ has odd parity.

Therefore, if $\varphi$ FO-defines $\textsc{Connectivity}$ on graphs with predicate edge, then $\neg(\varphi \wedge \psi)$ FO-defines $\textsc{Parity}$ on linear orders.

Since $\textsc{Parity}$ is not FO-definable, no such $\varphi$ can exist.

# Beyond Linear Orders: Locality

Intuition: Duplicator can win an EF game if selected nodes have the same "neighbourhood"
$\leadsto$ let's define this for graphs (structures with binary predicates)

## Definition

Consider a graph $\mathcal{G}$. For a natural number $d \geq 0$ and a vertex $v$, the $d$-neighbourhood of $v$, $N(v, d)$, is defined inductively:

- $N(v, 0) = \{v\}$
- $N(v, d + 1) = N(v, d) \cup$
  $\{w \mid w \text{ is a direct neighbour of some } w' \in N(v, d)\}$

Two vertices $v$ and $w$ have the same $d$-type if the subgraphs $\mathcal{G}|_{N(v,d)}$ and $\mathcal{G}|_{N(w,d)}$ are isomorphic.
Two graphs are $d$-equivalent if, for every $d$-type, they have the same number of $d$-neighbourhoods of this type.

# Locality and FO-definability

A special case of Gaifman's Locality Theorem of first-order logic:

**Theorem**

For every integer $r \geq 1$:

- if $\mathcal{G}_1$ is $3^{r-1}$-equivalent to $\mathcal{G}_2$
- then $\mathcal{G}_1 \sim_r \mathcal{G}_2$, and thus $\mathcal{G}_1 \equiv_r \mathcal{G}_2$

$\leadsto$ Intuition: FO can only express local properties

How to show that a query mapping $M$ can not be FO-defined:

- Let $C_M$ be the class of all databases recognised by $M$
- Find sequences of graphs $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, \ldots \in C_M$ and graphs $\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \ldots \notin C_M$, such that $\mathcal{I}_i$ is $i$-equivalent to $\mathcal{J}_i$

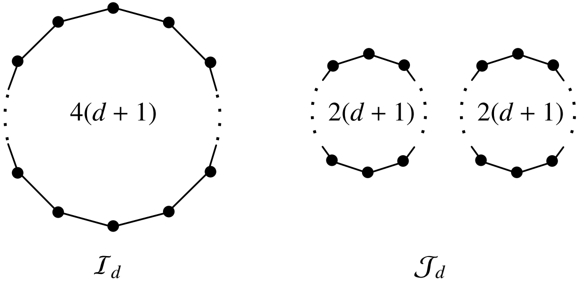
$\leadsto$ for any formula $\varphi$ (however large its quantifier rank $r$), there is a counterexample $\mathcal{I}_{3^{r-1}} \in C_M$ and $\mathcal{J}_{3^{r-1}} \notin C_M$ that $\varphi$ cannot distinguish

# CONNECTIVITY is not FO-definable (Proof 2)

**Theorem**

CONNECTIVITY is not FO-definable.

Proof: counterexample for quantifier rank $r$: set $d = 3^r$



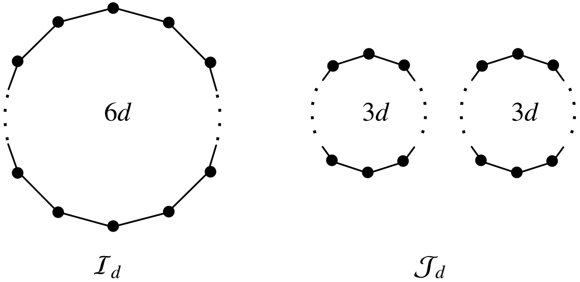

- the only $d$-type is a path of $2d + 1$ nodes
- $\mathcal{I}_d$ and $\mathcal{J}_d$ are $d$-equivalent

# 2-Colourability

## Theorem

2-Colourability is not FO-definable.

Proof: counterexample for quantifier rank $r$: set $d = 3^r$ (odd number)



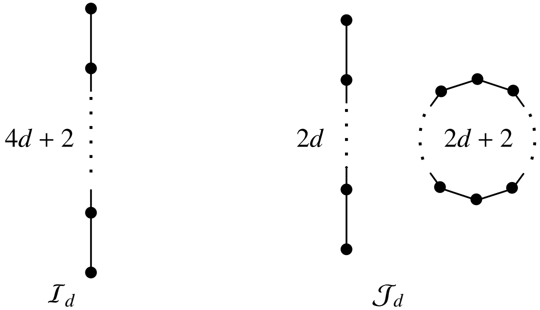

- the only $d$-type is a path of $2d + 1$ nodes
- $\mathcal{I}_d$ and $\mathcal{J}_d$ are $d$-equivalent

# Acyclicity

## Theorem

Acyclicity is not FO-definable.

Proof: counterexample for quantifier rank $r$: set $d = 3^r$




- $d$-types are paths of $\leq 2d + 1$ nodes
- $\mathcal{I}_d$ and $\mathcal{J}_d$ are $d$-equivalent

# Summary: Limits of FO-Queries

FO queries (and hence Relational Calculus) cannot express properties that require a “global” view:

- properties where one needs to follow paths
- properties where one needs to count elements

Remember Lecture 1?

“Stops at distance 2 from Helmholtzstr.”

$$R_2 = \delta_{\mathsf{To}\to\mathsf{From}}(\pi_{\mathsf{To}}(\mathsf{Connect} \bowtie R_1))$$

What about all stops reachable from Helmholtzstr.?

# Summary: Limits of FO-Queries

FO queries (and hence Relational Calculus) cannot express properties that require a "global" view:

- properties where one needs to follow paths
- properties where one needs to count elements

Remember Lecture 1?

"Stops at distance 2 from Helmholtzstr."

$$R_2 = \delta_{\mathsf{To}\to\mathsf{From}}(\pi_{\mathsf{To}}(\mathsf{Connect} \bowtie R_1))$$

What about all stops reachable from Helmholtzstr.?

$\leadsto$ Not expressible in Relational Calculus

Yet, all examples we saw are in $\mathrm{P}$

$\leadsto$ Is there another query language that could help us?

# Introduction to Datalog

# Introduction to Datalog

Datalog introduces recursion into database queries

- Use deterministic rules to derive new information from given facts
- Inspired by logic programming (Prolog)
- However, no function symbols and no negation
- Studied in AI (knowledge representation) and in databases (query language)

Example: transitive closure $C$ of a binary relation $r$

$$C(x, y) \leftarrow r(x, y)$$
$$C(x, z) \leftarrow C(x, y) \wedge r(y, z)$$

Intuition:

- some facts of the form $r(x, y)$ are given as input, and the rules derive new conclusions $C(x, y)$
- variables range over all possible values (implicit universal quantifier)

# Syntax of Datalog

Recall: A term is a constant or a variable. An atom is a formula of the form $R(t_1, \ldots, t_n)$ with $R$ a predicate symbol (or relation) of arity $n$, and $t_1, \ldots, t_n$ terms.

## Definition

A Datalog rule is an expression of the form:

$$H \leftarrow B_1 \wedge \ldots \wedge B_m$$

where $H$ and $B_1, \ldots, B_m$ are atoms. $H$ is called the head or conclusion; $B_1 \wedge \ldots \wedge B_m$ is called the body or premise. A rule with empty body ($m = 0$) is called a fact. A ground rule is one without variables (i.e., all terms are constants).

A set of Datalog rules is a Datalog program.

# Datalog: Example

$$
\begin{aligned}
\text{father}(\text{alice}, \text{bob}) & \\
\text{mother}(\text{alice}, \text{carla}) & \\
\text{mother}(\text{evan}, \text{carla}) & \\
\text{father}(\text{carla}, \text{david}) & \\
\text{Parent}(x, y) & \leftarrow \text{father}(x, y) \\
\text{Parent}(x, y) & \leftarrow \text{mother}(x, y) \\
\text{Ancestor}(x, y) & \leftarrow \text{Parent}(x, y) \\
\text{Ancestor}(x, z) & \leftarrow \text{Parent}(x, y) \wedge \text{Ancestor}(y, z) \\
\text{SameGeneration}(x, x) & \\
\text{SameGeneration}(x, y) & \leftarrow \text{Parent}(x, v) \wedge \text{Parent}(y, w) \wedge \text{SameGeneration}(v, w)
\end{aligned}
$$

# Datalog Semantics by Deduction

What does a Datalog program express?
Usually we are interested in entailed ground atoms

What can be entailed? Informally:

- Restrict to set of constants that occur in program (finite)
  $\leadsto$ universe $\mathcal{U}$
- Variables can represent arbitrary constants from this set
  $\leadsto$ ground substitutions map variables to constants
- A rule can be applied if its body is satisfied for some ground substitution
  Example: rule Parent$(x, y) \leftarrow$ mother$(x, y)$ can be applied to mother(alice, carla) under substitution $\{x \mapsto \text{alice}, y \mapsto \text{carla}\}$
- If a rule is applicable under some ground substitution, then the according instance of the rule head is entailed.

# Datalog Semantics by Deduction (2)

An inductive definition of what can be derived:

**Definition**

Consider a Datalog program $P$. The set of ground atoms that can be derived from $P$ is the smallest set of atoms $A$ for which there is a rule $H \leftarrow B_1 \wedge \ldots \wedge B_n$ and a ground substitution $\theta$ such that

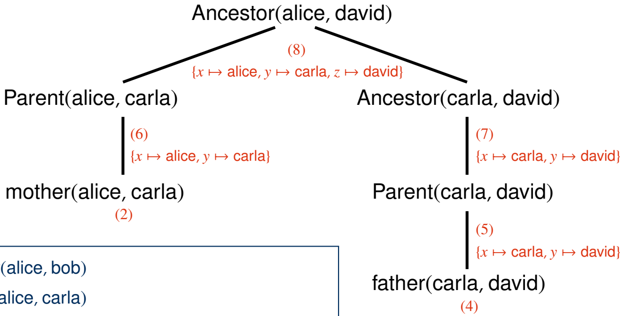
- $A = H\theta$, and
- for each $i \in \{1, \ldots, n\}$, $B_i\theta$ can be derived from $P$.

Notes:

- $n = 0$ for ground facts, so they can always be derived (induction base)
- if variables in the head do not occur in the body, they can be any constant from the universe

# Datalog Deductions as Proof Trees

We can think of deductions as tree structures:




(1) father(alice, bob)
(2) mother(alice, carla)
(3) mother(evan, carla)
(4) father(carla, david)
(5) $\text{Parent}(x, y) \leftarrow \text{father}(x, y)$
(6) $\text{Parent}(x, y) \leftarrow \text{mother}(x, y)$
(7) $\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y)$
(8) $\text{Ancestor}(x, z) \leftarrow \text{Parent}(x, y) \wedge \text{Ancestor}(y, z)$

# Datalog Semantics by Least Fixed Point

Instead of using substitutions, we can also ground programs:

**Definition**

The grounding $\mathrm{ground}(P)$ of a Datalog program $P$ is the set of all ground rules that can be obtained from rules in $P$ by uniformly replacing variables with constants from the universe.

Derivations are described by the immediate consequence operator $T_P$ that maps sets of ground facts $I$ to sets of ground facts $T_P(I)$:

- $T_P(I) = \{H \mid H \leftarrow B_1 \wedge \ldots \wedge B_n \in \mathrm{ground}(P) \text{ and } B_1, \ldots, B_n \in I\}$
- Least fixed point of $T_P$: smallest set $L$ such that $T_P(L) = L$
- Bottom-up computation: $T_P^0 = \emptyset$ and $T_P^{i+1} = T_P(T_P^i)$
- The least fixed point of $T_P$ is $T_P^\infty = \bigcup_{i \geq 0} T_P^i$ (exercise)

Observation: Ground atom $A$ is derived from $P$ if and only if $A \in T_P^\infty$

# Datalog Semantics by Least Model

We can also read Datalog rules as universally quantified implications

Example: Ancestor$(x, z) \leftarrow$ Parent$(x, y) \wedge$ Ancestor$(y, z)$
corresponds to implication

$$\forall x, y, z.\text{Parent}(x, y) \wedge \text{Ancestor}(y, z) \rightarrow \text{Ancestor}(x, z).$$

A set of FO implications may have many models
$\leadsto$ consider least model over the domain defined by the universe

**Theorem**

A fact is entailed by the least model of a Datalog program if and only if it can be derived from the Datalog program.

# Datalog Semantics: Overview

There three equivalent ways of defining Datalog semantics:

- Proof-theoretic: What can be proven deductively?
- Operational: What can be computed bottom up?
- Model-theoretic: What is true in the least model?

In each case, we restrict to the universe of given constants.
$\leadsto$ similar to active domain semantics in databases

# Datalog as a Query Language

How can we use Datalog to query databases?
$\rightsquigarrow$ View database as set of ground facts
$\rightsquigarrow$ Specify which predicate yields the query result

## Definition

A Datalog query is a pair $\langle R, P \rangle$, where $P$ is a Datalog program and $R$ is the answer predicate.
Results of the query: $R$-facts entailed by $P$

Datalog queries distinguish "given" relations from "derived" ones:

- predicates that occur in a head of $P$ are intensional database (IDB) predicates
- predicates that only occur in bodies are extensional database (EDB) predicates

Requirement: database relations used as EDB predicates only

# Datalog as a Generalisation of CQs

A conjunctive query $\exists y_1, \ldots, y_m. A_1 \wedge \ldots \wedge A_\ell$ with answer variables $x_1, \ldots, x_n$ can be expressed as a Datalog query $\langle \mathsf{Ans}, P \rangle$ where $P$ has the single rule:

$$\mathsf{Ans}(x_1, \ldots, x_n) \leftarrow A_1 \wedge \ldots \wedge A_\ell$$

Unions of CQs can also be expressed (exercise)

Intuition: Datalog generalises UCQs with recursion

Open questions:

- How hard is it to answer Datalog queries?
- Can Datalog express all queries in $\mathrm{P}$?
- What about query containment and equivalence?

# Summary and Outlook

FO-queries can only express "local" properties

Possible proof techniques:

- Ehrenfeucht-Fraïssé Games
- Locality Theorems
- For more approaches see
  Chapter 17 of [Abiteboul, Hull, Vianu 1994]

Datalog can overcome some of these limitations

Next topics:

- Complexity and expressive power of Datalog
- Implementation techniques for Datalog