

# DATABASE THEORY

## Lecture 2: First-Order Queries

Markus Krötzsch

Knowledge-Based Systems

TU Dresden, 5th April 2022

More recent versions of this slide deck might be available.  
For the most current version of this course, see  
[https://iccl.inf.tu-dresden.de/web/Database\\_Theory/en](https://iccl.inf.tu-dresden.de/web/Database_Theory/en)

# What is a Query?

The relational queries considered so far produced a result table from a database. Other query languages can be completely different, but they usually agree on this:

## Definition 2.1:

- Syntax: a query expression  $q$  is a word from a query language (algebra expression, logical expression, etc.)
- Semantics: a query mapping  $M[q]$  is a function that maps a database instance  $\mathcal{I}$  to a database table  $M[q](\mathcal{I})$

↪ for some semantics, query mappings are not defined on all database instances

# Generic Queries

We only consider queries that do not depend on the concrete names given to constants in the database:

**Definition 2.2:** A query  $q$  is **generic** if, for every bijective renaming function  $\mu : \mathbf{dom} \rightarrow \mathbf{dom}$  and database instance  $\mathcal{I}$ :

$$\mu(M[q](\mathcal{I})) = M[\mu(q)](\mu(\mathcal{I})).$$

In this case,  $M[q]$  is closed under isomorphisms.

# Review: Example from Previous Lecture

Lines:

Line	Type
85	bus
3	tram
F1	ferry
...	...

Stops:

SID	Stop	Accessible
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

Every table has a **schema**:

- Lines[Line:string, Type:string]
- Stops[SID:int, Stop:string, Accessible:bool]
- Connect[From:int, To:int, Line:string]

# First-order Logic as a Query Language

Idea: database instances are finite first-order interpretations

~> use first-order formulae as query language

~> use unnamed perspective (more natural here)

Examples (using schema as in previous lecture):

- Find all bus lines:  $\text{Lines}(x, \text{"bus"})$
- Find all possible types of lines:  $\exists y. \text{Lines}(y, x)$
- Find all lines that depart from an accessible stop:

$$\exists y_{\text{SID}}, y_{\text{Stop}}, y_{\text{To}}. (\text{Stops}(y_{\text{SID}}, y_{\text{Stop}}, \text{"true"}) \wedge \text{Connect}(y_{\text{SID}}, y_{\text{To}}, x_{\text{Line}}))$$

# First-order Logic with Equality: Syntax

Basic building blocks:

- **Predicate names** with an arity  $\geq 0$ :  $p, q$ , Lines, Stops
- **Variables**:  $x, y, z$
- **Constants**:  $a, b, c$
- **Terms** are variables or constants:  $s, t$

Formulae of first-order logic are defined as usual:

$$\varphi ::= p(t_1, \dots, t_n) \mid t_1 \approx t_2 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \forall x.\varphi$$

where  $p$  is an  $n$ -ary predicate,  $t_i$  are terms, and  $x$  is a variable.

- An **atom** is a formula of the form  $p(t_1, \dots, t_n)$
- A **literal** is an atom or a negated atom
- Occurrences of variables in the scope of a quantifier are **bound**;  
other occurrences of variables are **free**

# First-order Logic Syntax: Simplifications

We use the usual shortcuts and simplifications:

- flat conjunctions ( $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$  instead of  $(\varphi_1 \wedge (\varphi_2 \wedge \varphi_3))$ )
- flat disjunctions (similar)
- flat quantifiers ( $\exists x, y, z. \varphi$  instead of  $\exists x. \exists y. \exists z. \varphi$ )
- $\varphi \rightarrow \psi$  as shortcut for  $\neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi$  as shortcut for  $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $t_1 \neq t_2$  as shortcut for  $\neg(t_1 \approx t_2)$

But we always use parentheses to clarify nesting of  $\wedge$  and  $\vee$ :

No “ $\varphi_1 \wedge \varphi_2 \vee \varphi_3$ ”!

# First-order Logic with Equality: Semantics

First-order formulae are evaluated over **interpretations**  $\langle \Delta^I, \cdot^I \rangle$ , where  $\Delta^I$  is the domain. To interpret formulas with free variables, we need a **variable assignment**  $\mathcal{Z} : \text{Var} \rightarrow \Delta^I$ .

- constants  $a$  interpreted as  $a^{I, \mathcal{Z}} = a^I \in \Delta^I$
- variables  $x$  interpreted as  $x^{I, \mathcal{Z}} = \mathcal{Z}(x) \in \Delta^I$
- $n$ -ary predicates  $p$  interpreted as  $p^I \subseteq (\Delta^I)^n$



# First-order Logic with Equality: Semantics

First-order formulae are evaluated over **interpretations**  $\langle \Delta^I, \cdot^I \rangle$ , where  $\Delta^I$  is the domain. To interpret formulas with free variables, we need a **variable assignment**  $\mathcal{Z} : \text{Var} \rightarrow \Delta^I$ .

- constants  $a$  interpreted as  $a^{I, \mathcal{Z}} = a^I \in \Delta^I$
- variables  $x$  interpreted as  $x^{I, \mathcal{Z}} = \mathcal{Z}(x) \in \Delta^I$
- $n$ -ary predicates  $p$  interpreted as  $p^I \subseteq (\Delta^I)^n$

A formula  $\varphi$  can be **satisfied** by  $I$  and  $\mathcal{Z}$ , written  $I, \mathcal{Z} \models \varphi$ :

- $I, \mathcal{Z} \models p(t_1, \dots, t_n)$  if  $\langle t_1^{I, \mathcal{Z}}, \dots, t_n^{I, \mathcal{Z}} \rangle \in p^I$
- $I, \mathcal{Z} \models t_1 \approx t_2$  if  $t_1^{I, \mathcal{Z}} = t_2^{I, \mathcal{Z}}$
- $I, \mathcal{Z} \models \neg \varphi$  if  $I, \mathcal{Z} \not\models \varphi$
- $I, \mathcal{Z} \models \varphi \wedge \psi$  if  $I, \mathcal{Z} \models \varphi$  and  $I, \mathcal{Z} \models \psi$
- $I, \mathcal{Z} \models \varphi \vee \psi$  if  $I, \mathcal{Z} \models \varphi$  or  $I, \mathcal{Z} \models \psi$
- $I, \mathcal{Z} \models \exists x. \varphi$  if there is  $\delta \in \Delta^I$  with  $I, \{x \mapsto \delta\}, \mathcal{Z} \models \varphi$
- $I, \mathcal{Z} \models \forall x. \varphi$  if for all  $\delta \in \Delta^I$  we have  $I, \{x \mapsto \delta\}, \mathcal{Z} \models \varphi$

# First-order Logic Queries

**Definition 2.3:** An  $n$ -ary **first-order query**  $q$  is an expression  $\varphi[x_1, \dots, x_n]$  where  $x_1, \dots, x_n$  are exactly the free variables of  $\varphi$  (in a specific order).

**Definition 2.4:** An **answer** to  $q = \varphi[x_1, \dots, x_n]$  over an interpretation  $\mathcal{I}$  is a tuple  $\langle a_1, \dots, a_n \rangle$  of constants such that

$$\mathcal{I} \models \varphi[x_1/a_1, \dots, x_n/a_n]$$

where  $\varphi[x_1/a_1, \dots, x_n/a_n]$  is  $\varphi$  with each free  $x_i$  replaced by  $a_i$ .

The **result** of  $q$  over  $\mathcal{I}$  is the set of all answers of  $q$  over  $\mathcal{I}$ .

# Boolean Queries

A **Boolean query** is a query of arity 0

↪ we simply write  $\varphi$  instead of  $\varphi[]$

↪  $\varphi$  is a closed formula (a.k.a. sentence)

What does a Boolean query return?

# Boolean Queries

A **Boolean query** is a query of arity 0

→ we simply write  $\varphi$  instead of  $\varphi[]$

→  $\varphi$  is a closed formula (a.k.a. sentence)

What does a Boolean query return?

Two possible cases:

- $\mathcal{I} \not\models \varphi$ , then the result of  $\varphi$  over  $\mathcal{I}$  is  $\emptyset$  (the empty table)
- $\mathcal{I} \models \varphi$ , then the result of  $\varphi$  over  $\mathcal{I}$  is  $\{\langle \rangle\}$  (the unit table)

Interpreted as Boolean check with result true or false (match or no match)

# Domain Dependence

We have defined FO queries over interpretations

↪ How *exactly* do we get from databases to interpretations?

- Constants are just interpreted as themselves:  $a^I = a$
- Predicates are interpreted according to the table contents
- But what is the domain of the interpretation?

# Domain Dependence

We have defined FO queries over interpretations

↪ How *exactly* do we get from databases to interpretations?

- Constants are just interpreted as themselves:  $a^I = a$
- Predicates are interpreted according to the table contents
- But what is the domain of the interpretation?

**Example 2.5:** What should the following queries return?

- (1)  $\neg \text{Lines}(x, \text{"bus"})[x]$
- (2)  $(\text{Connect}(x_1, \text{"42"}, \text{"85"}) \vee \text{Connect}(\text{"57"}, x_2, \text{"85"}))[x_1, x_2]$
- (3)  $\forall y.p(x, y)[x]$

# Domain Dependence

We have defined FO queries over interpretations

→ How *exactly* do we get from databases to interpretations?

- Constants are just interpreted as themselves:  $a^I = a$
- Predicates are interpreted according to the table contents
- But what is the domain of the interpretation?

**Example 2.5:** What should the following queries return?

- (1)  $\neg \text{Lines}(x, \text{"bus"})[x]$
- (2)  $(\text{Connect}(x_1, \text{"42"}, \text{"85"}) \vee \text{Connect}(\text{"57"}, x_2, \text{"85"}))[x_1, x_2]$
- (3)  $\forall y.p(x, y)[x]$

→ Answers depend on the interpretation domain, not just on the database contents

# Natural Domain

First possible solution: the natural domain

Natural domain semantics (ND):

- fix the interpretation domain to **dom** (infinite)
- query answers might be infinite (not a valid result table)  
     $\leadsto$  query result undefined for such cases



# Natural Domain: Examples

Query answers under natural domain semantics:

(1)  $\neg\text{Lines}(x, \text{"bus"})[x]$

Undefined on all databases

# Natural Domain: Examples

Query answers under natural domain semantics:

(1)  $\neg \text{Lines}(x, \text{"bus"})[x]$

Undefined on all databases

(2)  $(\text{Connect}(x_1, \text{"42"}, \text{"85"}) \vee \text{Connect}(\text{"57"}, x_2, \text{"85"}))[x_1, x_2]$

Undefined on databases with matching  $x_1$  or  $x_2$  in Connect, otherwise empty

# Natural Domain: Examples

Query answers under natural domain semantics:

(1)  $\neg \text{Lines}(x, \text{"bus"})[x]$

Undefined on all databases

(2)  $(\text{Connect}(x_1, \text{"42"}, \text{"85"}) \vee \text{Connect}(\text{"57"}, x_2, \text{"85"}))[x_1, x_2]$

Undefined on databases with matching  $x_1$  or  $x_2$  in Connect, otherwise empty

(3)  $\forall y.p(x, y)[x]$

Empty on all databases

# Active Domain

Alternative: restrict to constants that are really used

↪ active domain

- for a database instance  $\mathcal{I}$ ,  $\mathbf{adom}(\mathcal{I})$  is the set of constants used in relations of  $\mathcal{I}$
- for a query  $q$ ,  $\mathbf{adom}(q)$  is the set of constants in  $q$
- $\mathbf{adom}(\mathcal{I}, q) = \mathbf{adom}(\mathcal{I}) \cup \mathbf{adom}(q)$

Active domain semantics (AD):

consider database instance as interpretation over  $\mathbf{adom}(\mathcal{I}, q)$

# Active Domain: Examples

Query answers under active domain semantics:

(1)  $\neg \text{Lines}(x, \text{"bus"})[x]$

Let  $q' = \text{Lines}(x, \text{"bus"})[x]$ . The answer is  $\mathbf{adom}(\mathcal{I}, q) \setminus M[q'](\mathcal{I})$

# Active Domain: Examples

Query answers under active domain semantics:

(1)  $\neg \text{Lines}(x, \text{"bus"})[x]$

Let  $q' = \text{Lines}(x, \text{"bus"})[x]$ . The answer is  $\mathbf{adom}(\mathcal{I}, q) \setminus M[q'](\mathcal{I})$

(2)  $(\underbrace{\text{Connect}(x_1, \text{"42"}, \text{"85"})}_{\varphi_1[x_1]} \vee \underbrace{\text{Connect}(\text{"57"}, x_2, \text{"85"})}_{\varphi_2[x_2]})[x_1, x_2]$

The answer is  $M[\varphi_1](\mathcal{I}) \times \mathbf{adom}(\mathcal{I}, q) \cup \mathbf{adom}(\mathcal{I}, q) \times M[\varphi_2](\mathcal{I})$

# Active Domain: Examples

Query answers under active domain semantics:

(1)  $\neg \text{Lines}(x, \text{"bus"})[x]$

Let  $q' = \text{Lines}(x, \text{"bus"})[x]$ . The answer is  $\mathbf{adom}(\mathcal{I}, q) \setminus M[q'](\mathcal{I})$

(2)  $(\underbrace{\text{Connect}(x_1, \text{"42"}, \text{"85"})}_{\varphi_1[x_1]} \vee \underbrace{\text{Connect}(\text{"57"}, x_2, \text{"85"})}_{\varphi_2[x_2]})[x_1, x_2]$

The answer is  $M[\varphi_1](\mathcal{I}) \times \mathbf{adom}(\mathcal{I}, q) \cup \mathbf{adom}(\mathcal{I}, q) \times M[\varphi_2](\mathcal{I})$

(3)  $\forall y.p(x, y)[x]$

$\leadsto$  exercise (to think about)

# Domain Independence

Observation: some queries do not depend on the domain

- $\text{Stops}(x, y, \text{"true"})[x, y]$
- $(x \approx a)[x]$
- $p(x) \wedge \neg q(x)[x]$
- $r(x) \wedge \forall y. (q(x, y) \rightarrow p(x, y))[x]$  (exercise: why?)

In contrast, all example queries on the previous few slides are not domain independent

Domain independent semantics (DI):

consider only domain independent queries

use any domain  $\mathbf{adom}(\mathcal{I}, q) \subseteq \Delta^{\mathcal{I}} \subseteq \mathbf{dom}$  for interpretation



# How to Compare Query Languages

We have seen three ways of defining FO query semantics  
→ how to compare them?

# How to Compare Query Languages

We have seen three ways of defining FO query semantics

→ how to compare them?

**Definition 2.6:** The set of query mappings that can be described in a query language  $L$  is denoted  $\mathbf{QM}(L)$ .

- $L_1$  is **subsumed by**  $L_2$ , written  $L_1 \sqsubseteq L_2$ , if  $\mathbf{QM}(L_1) \subseteq \mathbf{QM}(L_2)$
- $L_1$  is **equivalent to**  $L_2$ , written  $L_1 \equiv L_2$ , if  $\mathbf{QM}(L_1) = \mathbf{QM}(L_2)$

We will also compare query languages under named perspective with query languages under unnamed perspective.

This is possible since there is an easy one-to-one correspondence between query mappings of either kind (see exercise).

# Equivalence of Relational Query Languages

**Theorem 2.7:** The following query languages are equivalent:

- Relational algebra RA
- FO queries under active domain semantics AD
- Domain independent FO queries DI

This holds under named and under unnamed perspective.

To prove it, we will show:

$$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}} \sqsubseteq AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$$

$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}}$

For a given RA query  $q[a_1, \dots, a_n]$ ,

we recursively construct a DI query  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  as follows:

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if  $q = R$  with signature  $R[a_1, \dots, a_n]$ , then  $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$

$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}}$

For a given RA query  $q[a_1, \dots, a_n]$ ,

we recursively construct a DI query  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  as follows:

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if  $q = R$  with signature  $R[a_1, \dots, a_n]$ , then  $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$
- if  $n = 1$  and  $q = \{a_1 \mapsto c\}$ , then  $\varphi_q = (x_{a_1} \approx c)$

$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}}$

For a given RA query  $q[a_1, \dots, a_n]$ ,  
we recursively construct a DI query  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  as follows:

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if  $q = R$  with signature  $R[a_1, \dots, a_n]$ , then  $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$
- if  $n = 1$  and  $q = \{\{a_1 \mapsto c\}\}$ , then  $\varphi_q = (x_{a_1} \approx c)$
- if  $q = \sigma_{a_i=c}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx c)$

$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}}$

For a given RA query  $q[a_1, \dots, a_n]$ ,

we recursively construct a DI query  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  as follows:

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if  $q = R$  with signature  $R[a_1, \dots, a_n]$ , then  $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$
- if  $n = 1$  and  $q = \{\{a_1 \mapsto c\}\}$ , then  $\varphi_q = (x_{a_1} \approx c)$
- if  $q = \sigma_{a_i=c}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx c)$
- if  $q = \sigma_{a_i=a_j}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx x_{a_j})$

$$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}}$$

For a given RA query  $q[a_1, \dots, a_n]$ ,

we recursively construct a DI query  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  as follows:

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if  $q = R$  with signature  $R[a_1, \dots, a_n]$ , then  $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$
- if  $n = 1$  and  $q = \{\{a_1 \mapsto c\}\}$ , then  $\varphi_q = (x_{a_1} \approx c)$
- if  $q = \sigma_{a_i=c}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx c)$
- if  $q = \sigma_{a_i=a_j}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx x_{a_j})$
- if  $q = \delta_{b_1, \dots, b_n \rightarrow a_1, \dots, a_n} q'$ , then
$$\varphi_q = \exists y_{b_1}, \dots, y_{b_n}. (x_{a_1} \approx y_{b_1}) \wedge \dots \wedge (x_{a_n} \approx y_{b_n}) \wedge \varphi_{q'}[y_{B_1}, \dots, y_{B_n}]$$



$RA_{\text{named}} \sqsubseteq DI_{\text{unnamed}}$

For a given RA query  $q[a_1, \dots, a_n]$ ,

we recursively construct a DI query  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  as follows:

We assume without loss of generality that all attribute lists in RA expressions respect the global order of attributes.

- if  $q = R$  with signature  $R[a_1, \dots, a_n]$ , then  $\varphi_q = R(x_{a_1}, \dots, x_{a_n})$
- if  $n = 1$  and  $q = \{\{a_1 \mapsto c\}\}$ , then  $\varphi_q = (x_{a_1} \approx c)$
- if  $q = \sigma_{a_i=c}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx c)$
- if  $q = \sigma_{a_i=a_j}(q')$ , then  $\varphi_q = \varphi_{q'} \wedge (x_{a_i} \approx x_{a_j})$
- if  $q = \delta_{b_1, \dots, b_n \rightarrow a_1, \dots, a_n} q'$ , then

$$\varphi_q = \exists y_{b_1}, \dots, y_{b_n}. (x_{a_1} \approx y_{b_1}) \wedge \dots \wedge (x_{a_n} \approx y_{b_n}) \wedge \varphi_{q'}[y_{B_1}, \dots, y_{B_n}]$$

(Here we assume that the  $a_1, \dots, a_n$  in  $\delta_{b_1, \dots, b_n \rightarrow a_1, \dots, a_n}$  are written in the order of attributes, while  $b_1, \dots, b_n$  might be in another order. We use  $\{B_1, \dots, B_n\} = \{b_1, \dots, b_n\}$  to denote the ordered version of the  $b_i$  attributes.  $\varphi_{q'}[y_{B_1}, \dots, y_{B_n}]$  is like  $\varphi_{q'}$  but using variables  $y_{B_i}$ .)

# RA<sub>named</sub> $\sqsubseteq$ DI<sub>unnamed</sub> (cont'd)

Remaining cases:

- if  $q = \pi_{a_1, \dots, a_n}(q')$  for a subquery  $q'[b_1, \dots, b_m]$  with  
 $\{b_1, \dots, b_m\} = \{a_1, \dots, a_n\} \cup \{c_1, \dots, c_k\}$ ,  
then  $\varphi_q = \exists x_{c_1}, \dots, x_{c_k} \cdot \varphi_{q'}$

# RA<sub>named</sub> $\sqsubseteq$ DI<sub>unnamed</sub> (cont'd)

Remaining cases:

- if  $q = \pi_{a_1, \dots, a_n}(q')$  for a subquery  $q'[b_1, \dots, b_m]$  with  $\{b_1, \dots, b_m\} = \{a_1, \dots, a_n\} \cup \{c_1, \dots, c_k\}$ ,  
then  $\varphi_q = \exists x_{c_1}, \dots, x_{c_k} \cdot \varphi_{q'}$
- if  $q = q_1 \bowtie q_2$ , then  $\varphi_q = \varphi_{q_1} \wedge \varphi_{q_2}$

# RA<sub>named</sub> $\sqsubseteq$ DI<sub>unnamed</sub> (cont'd)

Remaining cases:

- if  $q = \pi_{a_1, \dots, a_n}(q')$  for a subquery  $q'[b_1, \dots, b_m]$  with  $\{b_1, \dots, b_m\} = \{a_1, \dots, a_n\} \cup \{c_1, \dots, c_k\}$ ,  
then  $\varphi_q = \exists x_{c_1}, \dots, x_{c_k} \cdot \varphi_{q'}$
- if  $q = q_1 \bowtie q_2$ , then  $\varphi_q = \varphi_{q_1} \wedge \varphi_{q_2}$
- if  $q = q_1 \cup q_2$ , then  $\varphi_q = \varphi_{q_1} \vee \varphi_{q_2}$

# RA<sub>named</sub> $\sqsubseteq$ DI<sub>unnamed</sub> (cont'd)

Remaining cases:

- if  $q = \pi_{a_1, \dots, a_n}(q')$  for a subquery  $q'[b_1, \dots, b_m]$  with  $\{b_1, \dots, b_m\} = \{a_1, \dots, a_n\} \cup \{c_1, \dots, c_k\}$ ,  
then  $\varphi_q = \exists x_{c_1}, \dots, x_{c_k} \cdot \varphi_{q'}$
- if  $q = q_1 \bowtie q_2$ , then  $\varphi_q = \varphi_{q_1} \wedge \varphi_{q_2}$
- if  $q = q_1 \cup q_2$ , then  $\varphi_q = \varphi_{q_1} \vee \varphi_{q_2}$
- if  $q = q_1 - q_2$ , then  $\varphi_q = \varphi_{q_1} \wedge \neg \varphi_{q_2}$

## RA<sub>named</sub> $\sqsubseteq$ DI<sub>unnamed</sub> (cont'd)

Remaining cases:

- if  $q = \pi_{a_1, \dots, a_n}(q')$  for a subquery  $q'[b_1, \dots, b_m]$  with  $\{b_1, \dots, b_m\} = \{a_1, \dots, a_n\} \cup \{c_1, \dots, c_k\}$ ,  
then  $\varphi_q = \exists x_{c_1}, \dots, x_{c_k} \cdot \varphi_{q'}$
- if  $q = q_1 \bowtie q_2$ , then  $\varphi_q = \varphi_{q_1} \wedge \varphi_{q_2}$
- if  $q = q_1 \cup q_2$ , then  $\varphi_q = \varphi_{q_1} \vee \varphi_{q_2}$
- if  $q = q_1 - q_2$ , then  $\varphi_q = \varphi_{q_1} \wedge \neg \varphi_{q_2}$

One can show that  $\varphi_q[x_{a_1}, \dots, x_{a_n}]$  is domain independent and equivalent to  $q$   
 $\rightsquigarrow$  exercise

$$DI_{\text{unnamed}} \sqsubseteq AD_{\text{unnamed}}$$

This is easy to see

$$DI_{\text{unnamed}} \sqsubseteq AD_{\text{unnamed}}$$

This is easy to see:

- Consider an FO query  $q$  that is domain independent
- The semantics of  $q$  is the same for any domain  $\mathbf{adom} \subseteq \Delta^I \subseteq \mathbf{dom}$
- In particular, the semantics of  $q$  is the same under active domain semantics
- Hence, for every DI query, there is an equivalent AD query



$AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$

Consider an AD query  $q = \varphi[x_1, \dots, x_n]$ .

For an arbitrary attribute name  $a$ , we can construct an RA expression  $E_{a, \mathbf{adom}}$  such that

$$E_{a, \mathbf{adom}}(I) = \{\{a \mapsto c\} \mid c \in \mathbf{adom}(I, q)\}$$

$\leadsto$  exercise

$AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$

Consider an AD query  $q = \varphi[x_1, \dots, x_n]$ .

For an arbitrary attribute name  $a$ , we can construct an RA expression  $E_{a, \text{adom}}$  such that

$$E_{a, \text{adom}}(\mathcal{I}) = \{\{a \mapsto c\} \mid c \in \text{adom}(\mathcal{I}, q)\}$$

$\leadsto$  exercise

For every variable  $x$ , we use a distinct attribute name  $a_x$

- if  $\varphi = R(t_1, \dots, t_m)$  with signature  $R[a_1, \dots, a_m]$  with variables  $x_1 = t_{v_1}, \dots, x_n = t_{v_n}$  and constants  $c_1 = t_{w_1}, \dots, c_k = t_{w_k}$ ,

$$AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$$

Consider an AD query  $q = \varphi[x_1, \dots, x_n]$ .

For an arbitrary attribute name  $a$ , we can construct an RA expression  $E_{a, \mathbf{adom}}$  such that

$$E_{a, \mathbf{adom}}(\mathcal{I}) = \{\{a \mapsto c\} \mid c \in \mathbf{adom}(\mathcal{I}, q)\}$$

$\leadsto$  exercise

For every variable  $x$ , we use a distinct attribute name  $a_x$

- if  $\varphi = R(t_1, \dots, t_m)$  with signature  $R[a_1, \dots, a_m]$  with variables  $x_1 = t_{v_1}, \dots, x_n = t_{v_n}$  and constants  $c_1 = t_{w_1}, \dots, c_k = t_{w_k}$ ,  
then  $E_\varphi = \delta_{a_{v_1} \dots a_{v_n} \rightarrow a_{x_1} \dots a_{x_n}} (\pi_{a_{v_1}, \dots, a_{v_n}} (\sigma_{a_{w_1}=c_1} (\dots \sigma_{a_{w_k}=c_k} (R) \dots)))$
- if  $\varphi = (x \approx c)$

$AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$

Consider an AD query  $q = \varphi[x_1, \dots, x_n]$ .

For an arbitrary attribute name  $a$ , we can construct an RA expression  $E_{a, \mathbf{adom}}$  such that

$$E_{a, \mathbf{adom}}(\mathcal{I}) = \{ \{a \mapsto c\} \mid c \in \mathbf{adom}(\mathcal{I}, q) \}$$

$\leadsto$  exercise

For every variable  $x$ , we use a distinct attribute name  $a_x$

- if  $\varphi = R(t_1, \dots, t_m)$  with signature  $R[a_1, \dots, a_m]$  with variables  $x_1 = t_{v_1}, \dots, x_n = t_{v_n}$  and constants  $c_1 = t_{w_1}, \dots, c_k = t_{w_k}$ ,  
then  $E_\varphi = \delta_{a_{v_1} \dots a_{v_n} \rightarrow a_{x_1} \dots a_{x_n}} (\pi_{a_{v_1}, \dots, a_{v_n}} (\sigma_{a_{w_1}=c_1} (\dots \sigma_{a_{w_k}=c_k} (R) \dots)))$
- if  $\varphi = (x \approx c)$ , then  $E_\varphi = \{ \{a_x \mapsto c\} \}$
- if  $\varphi = (x \approx y)$

$$AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$$

Consider an AD query  $q = \varphi[x_1, \dots, x_n]$ .

For an arbitrary attribute name  $a$ , we can construct an RA expression  $E_{a, \text{adom}}$  such that

$$E_{a, \text{adom}}(\mathcal{I}) = \{\{a \mapsto c\} \mid c \in \text{adom}(\mathcal{I}, q)\}$$

$\leadsto$  exercise

For every variable  $x$ , we use a distinct attribute name  $a_x$

- if  $\varphi = R(t_1, \dots, t_m)$  with signature  $R[a_1, \dots, a_m]$  with variables  $x_1 = t_{v_1}, \dots, x_n = t_{v_n}$  and constants  $c_1 = t_{w_1}, \dots, c_k = t_{w_k}$ ,  
then  $E_\varphi = \delta_{a_{v_1} \dots a_{v_n} \rightarrow a_{x_1} \dots a_{x_n}} (\pi_{a_{v_1}, \dots, a_{v_n}} (\sigma_{a_{w_1}=c_1} (\dots \sigma_{a_{w_k}=c_k} (R) \dots)))$
- if  $\varphi = (x \approx c)$ , then  $E_\varphi = \{\{a_x \mapsto c\}\}$
- if  $\varphi = (x \approx y)$ , then  $E_\varphi = \sigma_{a_x=a_y} (E_{a_x, \text{adom}} \bowtie E_{a_y, \text{adom}})$
- other forms of equality atoms are similar

# $AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$ (cont'd)

Remaining cases:

- if  $\varphi = \neg\psi$

# $AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$ (cont'd)

Remaining cases:

- if  $\varphi = \neg\psi$ , then  $E_\varphi = (E_{a_{x_1}, \text{adom}} \bowtie \dots \bowtie E_{a_{x_n}, \text{adom}}) - E_\psi$
- if  $\varphi = \varphi_1 \wedge \varphi_2$

# $AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$ (cont'd)

Remaining cases:

- if  $\varphi = \neg\psi$ , then  $E_\varphi = (E_{a_{x_1}, \text{adom}} \bowtie \dots \bowtie E_{a_{x_n}, \text{adom}}) - E_\psi$
- if  $\varphi = \varphi_1 \wedge \varphi_2$ , then  $E_\varphi = E_{\varphi_1} \bowtie E_{\varphi_2}$
- if  $\varphi = \exists y. \psi$  where  $\psi$  has free variables  $y, x_1, \dots, x_n$



# $AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$ (cont'd)

Remaining cases:

- if  $\varphi = \neg\psi$ , then  $E_\varphi = (E_{a_{x_1}, \text{adom}} \bowtie \dots \bowtie E_{a_{x_n}, \text{adom}}) - E_\psi$
- if  $\varphi = \varphi_1 \wedge \varphi_2$ , then  $E_\varphi = E_{\varphi_1} \bowtie E_{\varphi_2}$
- if  $\varphi = \exists y. \psi$  where  $\psi$  has free variables  $y, x_1, \dots, x_n$ ,  
then  $E_\varphi = \pi_{a_{x_1}, \dots, a_{x_n}} E_\psi$

The cases for  $\forall$  and  $\exists$  can be constructed from the above

$\leadsto$  exercise

## $AD_{\text{unnamed}} \sqsubseteq RA_{\text{named}}$ (cont'd)

Remaining cases:

- if  $\varphi = \neg\psi$ , then  $E_\varphi = (E_{a_{x_1}, \text{adom}} \bowtie \dots \bowtie E_{a_{x_n}, \text{adom}}) - E_\psi$
- if  $\varphi = \varphi_1 \wedge \varphi_2$ , then  $E_\varphi = E_{\varphi_1} \bowtie E_{\varphi_2}$
- if  $\varphi = \exists y. \psi$  where  $\psi$  has free variables  $y, x_1, \dots, x_n$ ,  
then  $E_\varphi = \pi_{a_{x_1}, \dots, a_{x_n}} E_\psi$

The cases for  $\forall$  and  $\exists$  can be constructed from the above

$\leadsto$  exercise

**A note on order:** The translation yields an expression  $E_\varphi[a_{x_1}, \dots, a_{x_n}]$ . For this to be equivalent to the query  $\varphi[x_1, \dots, x_n]$ , we must choose the attribute names such that their global order is  $a_{x_1}, \dots, a_{x_n}$ . This is clearly possible, since the names are arbitrary and we have infinitely many names available.

# How to find DI queries?

Domain independent queries are arguably most intuitive, since their result does not depend on special assumptions.

~> How can we check if a query is in DI?

# How to find DI queries?

Domain independent queries are arguably most intuitive, since their result does not depend on special assumptions.

↪ How can we check if a query is in DI? Unfortunately, we can't:

**Theorem 2.8:** Given a FO query  $q$ , it is undecidable if  $q \in \text{DI}$ .

↪ find decidable sufficient conditions for a query to be in DI

# A Normal Form for Queries

We first define a normal form for FO queries:

## Safe-Range Normal Form (SRNF)

- Rename variables apart (distinct quantifiers bind distinct variables, bound variables distinct from free variables)
- Eliminate all universal quantifiers:  $\forall y.\psi \mapsto \neg\exists y.\neg\psi$
- Push negations inwards:
  - $\neg(\varphi \wedge \psi) \mapsto (\neg\varphi \vee \neg\psi)$
  - $\neg(\varphi \vee \psi) \mapsto (\neg\varphi \wedge \neg\psi)$
  - $\neg\neg\psi \mapsto \psi$

# Safe-Range Queries

Let  $\varphi$  be a formula in SRNF. The set  $rr(\varphi)$  of **range-restricted variables** of  $\varphi$  is defined recursively:

$$rr(R(t_1, \dots, t_n)) = \{x \mid x \text{ a variable among the } t_1, \dots, t_n\}$$

$$rr(x \approx a) = \{x\}$$

$$rr(x \approx y) = \emptyset$$

$$rr(\varphi_1 \wedge \varphi_2) = \begin{cases} rr(\varphi_1) \cup \{x, y\} & \text{if } \varphi_2 = (x \approx y) \text{ and } \{x, y\} \cap rr(\varphi_1) \neq \emptyset \\ rr(\varphi_1) \cup rr(\varphi_2) & \text{otherwise} \end{cases}$$

$$rr(\varphi_1 \vee \varphi_2) = rr(\varphi_1) \cap rr(\varphi_2)$$

$$rr(\exists y. \psi) = \begin{cases} rr(\psi) \setminus \{y\} & \text{if } y \in rr(\psi) \\ \text{throw new NotSafeException}() & \text{if } y \notin rr(\psi) \end{cases}$$

$$rr(\neg\psi) = \emptyset \quad \text{if } rr(\psi) \text{ is defined (no exception)}$$

# Safe-Range Queries

**Definition 2.9:** An FO query  $q = \varphi[x_1, \dots, x_n]$  is a **safe-range query** if

$$\text{rr}(\text{SRNF}(\varphi)) = \{x_1, \dots, x_n\}.$$

Safe-range queries are domain independent.

# Safe-Range Queries

**Definition 2.9:** An FO query  $q = \varphi[x_1, \dots, x_n]$  is a **safe-range query** if

$$\text{rr}(\text{SRNF}(\varphi)) = \{x_1, \dots, x_n\}.$$

Safe-range queries are domain independent.

One can show a much stronger result:

**Theorem 2.10:** The following query languages are equivalent:

- Safe-range queries SR
- Relational algebra RA
- FO queries under active domain semantics AD
- Domain independent FO queries DI



# Tuple-Relational Calculus

There are more equivalent ways to define a relational query language

Example: Codd's tuple calculus

- Based on named perspective
- Use first-order logic, but variables range over sorted tuples (rows) instead of values
- Use expressions like  $x : \text{From,To,Line}$  to declare sorts of variables in queries
- Use expressions like  $x.\text{From}$  to access a specific value of a tuple
- Example: Find all lines that depart from an accessible stop

$$\{x : \text{Line} \mid \exists y : \text{SID,Stop,Accessible}.(\text{Stops}(y) \wedge y.\text{Accessible} \approx \text{"true"} \\ \wedge \exists z : \text{From,To,Line}.(\text{Connect}(z) \wedge z.\text{From} \approx y.\text{SID} \\ \wedge z.\text{Line} \approx x.\text{Line}))\}$$

# Summary and Outlook

First-order logic gives rise to a relational query language

The problem of domain dependence can be solved in several ways

All common definitions lead to equivalent calculi

~> “relational calculus”

## Open questions:

- How hard is it to actually answer such queries? (next lecture)
- How can we study the expressiveness of query languages?
- Are there interesting query languages that are not equivalent to RA?