LGruDat: Logical Foundations of Databases
Lecture 1 — Introduction

Tadeusz Litak

November 3, 2013

Remember we do have a webpage!

`http://www8.cs.fau.de/ws13:lgrudat`

Go there for all the up-to-date information

Literature? Some reference are here:
`http://www8.cs.fau.de/ws13:lgrudat#literature`

I found (or recalled) a few more:
will add them successively

The lecture will be heavily based on references
I will try to make clear what I took from where ...
... but sometimes it would be unpractical
and sometimes I may simply forget to mention the reference

Any materials for the course are for internal distribution only

Database models

- Relational (our main focus)
  think SQL, DB2, relational algebra and tuple calculus
- Semi-structured
  think XML, DTD, XPath and XQuery
- Graph-based
  think RDF, OrientDB, ontologies and Semantic Web technology
- ...

Towards database $\rightsquigarrow$ logic dictionary?

database scheme	$\rightsquigarrow$	logical signature Σ
		(only relational)
database instance	$\rightsquigarrow$	finite model $\mathfrak{A}, \mathfrak{B}, \mathfrak{C} \dots$
		(just Herbrand models?)
database queries	$\rightsquigarrow$	formulas
		(already in this lecture more details)
query containment	$\rightsquigarrow$	valid implication/entailment
		(over finite models only!)
constraints	$\rightsquigarrow$	axioms ...

"$\rightsquigarrow$" should be read as *we translate as (but there are details one should be careful about)*. In forthcoming lectures, we will slowly go through the whole list

FORC syntax: a reminder

- $\Sigma = (\mathsf{arity}_\Sigma(\cdot) : \mathbf{rel}\Sigma \to \mathbb{N}, \mathbf{const}\Sigma)$.
  - $\mathbf{rel}\Sigma$ is the supply of relational symbols
  - $\mathbf{const}\Sigma$ is the supply of indvidual constants

    Database schemes provide just relation symbols with arities

    We're being more of logicians than of database theorists here
- We abbreviate $P \in \mathbf{rel}\Sigma$ to $P \in \Sigma$
- Let α be a formula in signature/scheme Σ

  Recall the definition of $\mathsf{FORC}(\Sigma)$:

  (first-order) relational calculus

$$\alpha, \beta ::= s = t \mid P(\bar{\mathbf{s}}) \mid \alpha \wedge \beta \mid \neg\alpha \mid \forall v.\alpha$$

  - s and t are either elements of the set of variables $Var := x_1, x_2 \dots$ or of $\mathbf{const}\Sigma$
  - $P \in \Sigma$ and $\bar{\mathbf{s}}$ is of length $\mathsf{arity}_\Sigma(P)$
  - v is a metavariable ranging over variables $x_1, x_2 \dots$

FORC semantics: a reminder

- $\mathfrak{A}$ is a model adequate for Σ if it is of the form $(\mathsf{A}, \{\mathsf{P}^{\mathfrak{A}}\}_{P \in \Sigma}, \{c^{\mathfrak{A}}\}_{c \in \mathbf{const}\Sigma})$

  A is the carrier set or the domain of $\mathfrak{A}$

  For each P, $\mathsf{P}^{\mathfrak{A}}$ is a subset of $\mathsf{A}^{\mathsf{arity}_\Sigma(P)}$

  For each c, $c^{\mathfrak{A}}$ is an element of A
- Two settings:
  - unrestricted when A may be of *arbitrary cardinality*
  - finite when we allow only finite A

    the latter standard for databases, see Kanellakis' overview
- An $\mathfrak{A}$-valuation is a mapping $\kappa : Var \to \mathsf{A}$

- Definition of satisfaction $\mathfrak{A}, \kappa \vDash \alpha$ ...
- ...an exercise (done on blackboard)

Exercise solution

$$
\begin{array}{ll}
\mathfrak{A}, \kappa \vDash s = t & \text{iff } \hat{\kappa}(s) = \hat{\kappa}(t) \\
\mathfrak{A}, \kappa \vDash P(s_1, \dots, s_n) & \text{iff } P(\hat{\kappa}(s_1), \dots, \hat{\kappa}(s_n)) \\
\mathfrak{A}, \kappa \vDash \alpha \wedge \beta & \text{iff } \mathfrak{A}, \kappa \vDash \alpha \ \textsf{and} \ \mathfrak{A}, \kappa \vDash \beta \\
\mathfrak{A}, \kappa \vDash \neg\alpha & \text{iff } \ \textsf{not} \ (\mathfrak{A}, \kappa \vDash \alpha) \\
\mathfrak{A}, \kappa \vDash \forall v.\alpha & \text{iff } \ \textsf{for\_all} \ a \in A, \mathfrak{A}, \kappa[v := a] \vDash \alpha
\end{array}
$$

where

$$
\hat{\kappa}(s) := \begin{cases} \kappa(v) & \text{if } s = v, \\ c^{\mathfrak{A}} & \text{if } s = c. \end{cases}
$$

$$
\kappa[v := a](v') := \begin{cases} \kappa(v) & \text{if } v \neq v', \\ a & \text{if } v' = v. \end{cases}
$$

FORC queries

- Take $\alpha \in \mathsf{FORC}(\Sigma)$
- Let $\overline{\mathbf{v}} = (v_1, \dots, v_n)$ be a sequence of variables s.t.

$$
\forall v, v \in \overline{\mathbf{v}} \quad \text{iff} \quad v \in free(\alpha)
$$

(see Definition 4.2.7 and Section 5.3 in the *Foundations of Databases* book ...
...or Definition 2.2.1 in Kanellakis' overview)

- We will call ϕ of the form

$$
\overline{\mathbf{v}} \mid \alpha
$$

  a (typed) $\mathsf{FORC}(\Sigma)$ query
  (sometimes I will write it in set brackets: $\{\phi\}$)

Effects of queries

- Assume $\mathfrak{A} = (\mathsf{A}, \{\mathsf{P}^{\mathfrak{A}}\}_{P \in \Sigma}, \{c^{\mathfrak{A}}\}_{c \in \mathbf{const}\Sigma})$ is a model adequate for Σ
- Assume $\phi = \{(v_1, \dots, v_n) \mid \alpha\}$ is a $\mathsf{FORC}(\Sigma)$-query
- We define then

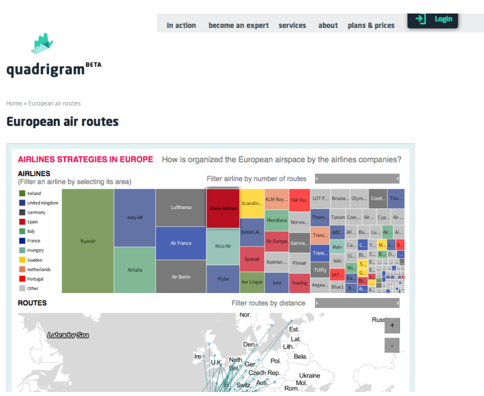
$$
\begin{aligned}
\phi(\mathfrak{A}) &:= \{(\mathsf{a}_1, \dots \mathsf{a}_n) \in \mathsf{A}^n \mid \ \textsf{exists} \ \kappa.\kappa(\overline{\mathbf{v}}) = \overline{\mathsf{a}} \ \textsf{and} \ \mathfrak{A}, \kappa \vDash \alpha\} \\
&= \{(\mathsf{a}_1, \dots \mathsf{a}_n) \in \mathsf{A}^n \mid \ \textsf{for\_all} \ \kappa.\kappa(\overline{\mathbf{v}}) = \overline{\mathsf{a}} \ \textsf{implies} \ \mathfrak{A}, \kappa \vDash \alpha\} \\
&= \{(\mathsf{a}_1, \dots \mathsf{a}_n) \in \mathsf{A}^n \mid \ \textsf{for\_all} \ \kappa.\mathfrak{A}, \kappa[v_1 := \mathsf{a}_1] \dots [v_n := \mathsf{a}_n] \vDash \alpha\} \\
&= \{(\mathsf{a}_1, \dots \mathsf{a}_n) \in \mathsf{A}^n \mid \ \textsf{exists} \ \kappa.\mathfrak{A}, \kappa[v_1 := \mathsf{a}_1] \dots [v_n := \mathsf{a}_n] \vDash \alpha\}
\end{aligned}
$$

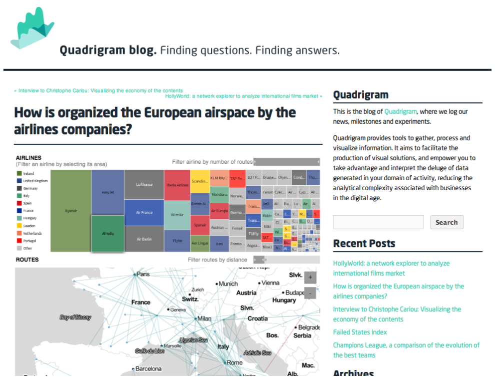
- These equalities need to be proved . . .
- by showing that whenever
  - $free(\alpha) = \{v_1, \ldots, v_n\}$
  - $\kappa(v_1) = \kappa'(v_1), \ \ldots, \ \kappa(v_n) = \kappa'(v_n)$

  then $\mathfrak{A}, \kappa \models \alpha$ iff $\mathfrak{A}, \kappa' \models \alpha$
- Note we obtain boolean queries as the limit case

Example (simplest possible—see introduction to Libkin's book)

- Assume **rel**Σ consists only of $\{Flight\}$ with $\mathsf{arity}_\Sigma(Flight) = 2$
  constΣ consists of all names of European cities you can think of
- Each airline has a corresponding instance
- Elements of the domain (carrier set) are cities
- A pair of cities is in the denotation of *Flight*
  in the instance of the database generated by a given airline
  iff
  the airline flies between them
- To make our life simpler and our slides more appealing,
  we use data from a project based on the Quadrigram tool

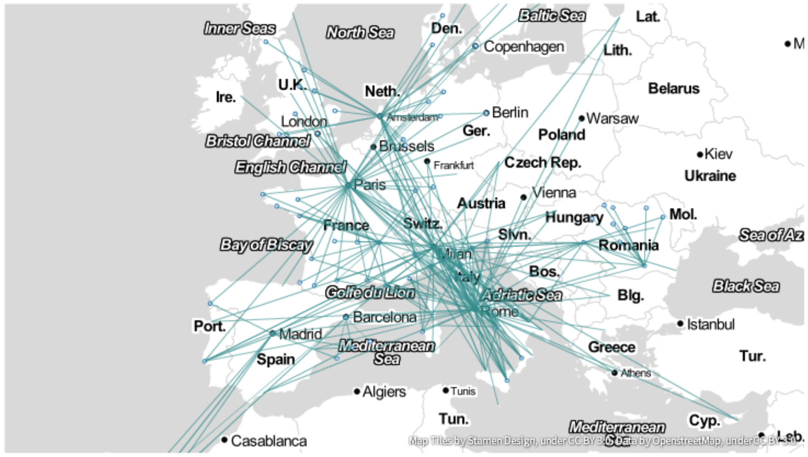




RyanAir

Malev

Alitalia



Iberia

ThomsonFly

EasyJet

Lufthansa

. . . you get the idea . . .

An aside: first encounter with constraints?

- *Flight* is in all likelihood a symmetric relation
- Few airlines tend to fly to a city without return flights
  . . . although it is imaginable the outgoing flight goes somewhere else . . .
  . . . we can neglect this "open jaw" complication . . .
- We would model this with a symmetric (undirected) graph . . .
  (a structure we will encounter often, by the way)
- . . . or enforce with a constraint.
  In a FO notation:

$$Flight\,(x,y) -> Flight\,(y,x)$$

  or with a more explicit universal closure:

$$\forall x,y.Flight\,(x,y) -> Flight\,(y,x)$$

First example query

Find all *cities* ***with*** *a direct flight to Warszawa (Warsaw)*

FO notation:

$$x \mid Flight\,(x, Warsaw)$$

SQL-like query for comparision:

```
SELECT Flight.origin FROM Flight
         WHERE Flight.destination = 'Warsaw';
```

Iberia

$\cdots = \emptyset$

Malev

$\cdots = \{$Budapest$\}$

SpainAir

$\cdots = \{\text{Barcelona}, \text{Madrid}\}$

Second example query

Find all *cities* ***with more than one*** *outgoing flight*

FO notation:

$$x \mid \exists y_1, y_2 . Flight\,(x, y_1) \wedge Flight\,(x, y_2) \wedge y_1 \neq y_2$$

SQL-like query for comparision:

```
SELECT F.origin FROM Flight F WHERE
  (SELECT COUNT(Flight.destination) > 1 FROM Flight
   WHERE Flight.origin = F.origin );
```

In some case, the effect will be easy to guess ...

AigleAzur

$\cdots = \{$Paris$\}$

In some cases, just slightly more complicated ...

Malev

$\cdots = \{$Budapest, London$\}$

(if this dodgy connection map of ours can be trusted)

And in some cases ...

RyanAir

. . . . . . do you really want to know?

Recall again the shape of the query

FO notation:

$$x \mid \exists y_1, y_2. \mathit{Flight}\,(x, y_1) \land \mathit{Flight}\,(x, y_2) \land y_1 \neq y_2$$

SQL-like query:

```
SELECT F.origin FROM Flight F  WHERE
  (SELECT  COUNT(Flight.destination)  > 1 FROM Flight
   WHERE Flight.origin =  F.origin );
```

Note we used an SQL construct without a FO counterpart:

counting

In the example in question, it did not matter:
in FO, we could use (in-)equality instead

But what would you do with SQL queries like this one:

```
SELECT F1.origin , F2.destination
           FROM Flight F1, Flight F2  WHERE
                  (SELECT  COUNT(Flight.destination) FROM Flight
                              WHERE Flight.origin =  F1.origin)
                                     =
                       (SELECT  COUNT(Flight.origin)  FROM Flight
                              WHERE Flight.destination =
                                                 F2.destination  );
```

(can you see what this query is doing, btw?)

Can we show this query is inexpressible
in plain relational calculus (w/o counting)?

Is the latter exactly as expressive as FOL?

And are there queries even worse than that?
I.e., not only inexpressible in FOL,
but also in SQL with counting?

Start with something simple . . .

Find all pairs *of cities* ***with*** *a direct flight between them*

FO notation:

$$(x, y) \mid Flight\,(x, y)$$

SQL notation:

```
SELECT * FROM Flight;
```

Rather trivial so far . . .

Up one gear . . .

Find all pairs *of cities* ***with exactly one*** *interchange*

FO notation:

$$(x, y) \mid \exists z.(Flight\,(x, z) \wedge Flight\,(z, y))$$

SQL notation:

```
SELECT F1.origin, F2.destination FROM Flight F1, Flight F2 WHERE
    (F1.destination = F2.origin);
```

Not much more complicated . . .

Up one more gear . . .

Find all pairs *of cities* ***with at most one*** *interchange*

FO notation:

$$(x,y) \mid Flight\,(x,y) \vee \exists z.(Flight\,(x,z) \wedge Flight\,(z,y))$$

SQL notation:

```
SELECT F1.origin, F2.destination FROM Flight F1, Flight F2 WHERE
    (F1.destination = F2.origin)
        OR ((F1.origin = F2.origin)
                AND (F1.destination = F2.destination));
```

So far so good ...

... you probably see how to handle

***Find all pairs** of cities connected **with at most** 2 interchanges*

... but how about reachability?

That is:

***Find all pairs** of cities connected*
***with finitely many** interchanges*

Sure, in some cases whatever can be reached, can be reached with at most one interchange ...

AigleAzur

... in some cases, at most two interchanges would do ...

Malev

...but we want a query which would do the job in any instance!

RyanAir

- Is there any such FOL/SQL expression?
- If not, how can we mathematically prove there is none?
- This is a typical question we will be concerned with

Other examples

- Query containment
- Query equivalence

- Query non-emptiness
- Our tools are mainly those of finite model theory
- Our future highlights: `http://www8.cs.fau.de/ws13:lgrudat`