G53RDB: Theory of Relational Databases

Lecture 13

Natasha Alechina
School of Computer Science & IT
nza@cs.nott.ac.uk

Plan of the lecture

- Relational calculus and relational algebra
- Deductive databases
- Datalog:
  - syntax
  - non-recursive queries
  - meaning of non-recursive queries

Formulas and relations

- A formula with n free variables defines an n-ary relation: the set of n-tuples which satisfy this formula.
- For example, ∃ x ChildOf(x,y,z) defines a binary relation
  {<**a**, **b**>: <y:**a**,z:**b**> satisfy ∃ z ChildOf(x,y,z)}

ChildOf(x,y,z)

x	y	z
Mary	Tom	Jane
Bill	Tom	Jane

∃ x ChildOf(x,y,z)

y	z
Tom	Jane

Relational calculus and relational algebra

The relationship is really quite simple (if we abstract from many technical details like variable names vs attributes):

- Starting from the same set of relations, all relations we can build using relational algebra operators, we can define by a formula of relational calculus, and vice versa.
- The two formalisms are equally expressive: every query we can define in one, we can define in another.
- Certain things are easier with relational calculus: it is more flexible, closer to natural language. On the other hand, it is much easier to design algorithms for evaluating algebraic expressions.

Relations and formulas

- Suppose we have a set of relations (database tables). We can define new relations using relational algebra operators π, σ, $\times$, $\cup$, $-$.
- I am going to show that for each such relation, we can write a first order formula which is true exactly for the set of tuples in the relation.

Basic relations and formulas

- For each basic n-ary relation **R** (a database table), we introduce a relation name R
- The formula $R(x_1,\ldots,x_n)$ is then true for exactly tuples in **R**.
- Note that instead of $x_1,\ldots,x_n$, we could have used $y_1,\ldots,y_n$.

Relation **R**

$\langle \mathbf{a_1}, \ldots, \mathbf{a_n} \rangle$
$\langle \mathbf{b_1}, \ldots, \mathbf{b_n} \rangle$
$\langle \mathbf{c_1}, \ldots, \mathbf{c_n} \rangle$

Relation name R

$R(x_1,\ldots,x_n)$

Inductive step

- The rest we show by induction.
- We can define basic relations by atomic formulas, so now need to show how we can define union, difference, product, projection and selection of two definable relations.

Union of relations

- Suppose that we have two union compatible relations **R** and **P**.
- Inductive hypothesis: there exists a first order formula $\phi(x_1,\ldots,x_n)$ which defines relation **R** and a first order formula $\psi(x_1,\ldots, x_n)$ which defines relation **P** (a tuple satisfies the formula if, and only if, it belongs to the relation).
- Then $\mathbf{R} \cup \mathbf{P}$ is defined by $\phi(x_1,\ldots,x_n) \vee \psi(x_1,\ldots, x_n)$.
- Indeed, a tuple is in $\mathbf{R} \cup \mathbf{P}$ if, and only if, it belongs to **R** or **P**; by the inductive hypothesis, this holds if, and only if, it satisfies $\phi(x_1,\ldots,x_n)$ or $\psi(x_1,\ldots, x_n)$; which means, if, and only if, it satisfies $\phi(x_1,\ldots,x_n) \vee \psi(x_1,\ldots, x_n)$.

Difference of relations

- Suppose that we have two union compatible relations **R** and **P**.
- Inductive hypothesis: there exists a first order formula $\phi(x_1,\ldots,x_n)$ which defines relation **R** and a first order formula $\psi(x_1,\ldots, x_n)$ which defines relation **P**
- Then $\mathbf{R} - \mathbf{P}$ is defined by $\phi(x_1,\ldots,x_n) \,\&\, \neg\psi(x_1,\ldots, x_n)$.
- Indeed, a tuple is in $\mathbf{R} - \mathbf{P}$ if, and only if, it belongs to **R** and does not belong to **P**; by the inductive hypothesis, this holds if, and only if, it satisfies $\phi(x_1,\ldots,x_n)$ and does not satisfy $\psi(x_1,\ldots, x_n)$; which means, if, and only if, it satisfies $\phi(x_1,\ldots,x_n) \,\&\, \neg\, \psi(x_1,\ldots, x_n)$.

Product of relations

- Suppose that we have two relations **R** and **P**.
- Inductive hypothesis: there exists a first order formula $\phi(x_1,\ldots,x_n)$ which defines relation **R** and a first order formula $\psi(y_1,\ldots, y_k)$ which defines relation **P**
- Then $\mathbf{R} \times \mathbf{P}$ is defined by $\phi(x_1,\ldots,x_n) \,\&\, \psi(y_1,\ldots, y_k)$.
- Indeed, a tuple $<\mathbf{a_1},\ldots, \mathbf{a_n},\mathbf{b_1},\ldots, \mathbf{b_k}>$ is in $\mathbf{R} \times \mathbf{P}$ if, and only if, first n values belong to **R** and next k values belong to **P**; by the inductive hypothesis, this holds if, and only if, $<\mathbf{a_1},\ldots, \mathbf{a_n}>$ it satisfies $\phi(x_1,\ldots,x_n)$ and $<\mathbf{b_1}, \ldots, \mathbf{b_k}>$ satisfies $\psi(y_1,\ldots, y_k)$; which means, if, and only if, it satisfies $\phi(x_1,\ldots,x_n) \,\&\, \psi(y_1,\ldots, y_k)$.

Projection

- Suppose we have a relation **R**, and by the inductive hypothesis, there exists a first order formula $\phi(x_1,\ldots,x_n, y_1,\ldots, y_k)$ which defines **R**. Suppose we want to project just on the attribute positions corresponding to xs.
- Then $\pi_{x1,\ldots,xn}$ **R** is defined by $\exists\, y_1 \ldots \exists\, y_n\, \phi$:

Relation **R**

x_1 …	x_n	y_1 …	y_k
$\mathbf{a_1}$ …	$\mathbf{a_n}$	$\mathbf{b_1}$ …	$\mathbf{b_k}$
$\mathbf{c_1}$ …	$\mathbf{c_n}$	$\mathbf{d_1}$ …	$\mathbf{d_k}$
$\mathbf{e_1}$ …	$\mathbf{e_n}$	$\mathbf{f_1}$ …	$\mathbf{f_k}$

Selection

- Suppose we have a relation **R**, and by the inductive hypothesis, there exists a first order formula $\phi(x_1,\ldots,x_n)$ which defines **R**. Suppose we want to select tuples which satisfy property α.
- If α can be expressed in first order logic, as some condition $\alpha(x_1,\ldots,x_n)$, then
- σ_α **R** is defined by $\alpha(x_1,\ldots,x_n) \,\&\, \phi(x_1,\ldots,x_n)$.
- All we need to express α is to add to our first order language comparison operators and constants for database values. We already have &, $\neg$ and $\vee$.

So…

- Every relation definable in relational algebra, is also definable in first order logic.
- The opposite direction (everything definable in first order logic, is definable in relational algebra) also holds, but is a lot harder to prove.
- The two formalisms (relational algebra and first order logic) have the same expressive power.

Deductive databases

- Deductive databases are very similar to relational databases, but their query languages have an additional capability: they can express recursive queries.
- The syntax is closely based on predicate logic

Datalog

- One of the query languages used in deductive databases is called Datalog.
- Its syntax is very similar to predicate logic/relational calculus
- It is also very similar to logical programming (Prolog), but there are some important differences.

Syntax of Datalog

- Similarly to relational calculus, Datalog statements are built from atoms.
- ***Relational atom***: if R is a relation name of arity n and $t_1,\ldots,t_n$ are variables or constants, then $R(t_1,\ldots,t_n)$ is a relational atom
- ***Arithmetic atom:*** comparison between two arithmetic expressions, for example $x > 5$, $x \geq y$, etc.

Datalog syntax: rules

- A Datalog ***rule*** is an expression of the form

$$R_1 \leftarrow R_2 \text{ AND } \ldots \text{ AND } R_n$$

where $n \geq 1$, R_1 is a relational atom, and $R_2,\ldots, R_n$ are relational or arithmetic atoms, possibly preceded by NOT.

- R_1 is called the ***head*** of the rule and $R_2,\ldots, R_n$ the ***body*** of the rule.
- $R_2,\ldots, R_n$ are called ***subgoals***.

Example

- Suppose we have a relation Person over schema (Name, Age, Address, Telephone). Then the following Datalog rule will define a relation which contains names of people aged over 18:

$$\text{Adult}(x) \leftarrow \text{Person}(x,y,z,u) \text{ AND } y \geq 18$$

Anonymous variables

Adult(x) ← Person(x,y,z,u) AND y ≥ 18

- Note that z and u appear only once in the rule, and their names do not matter (thay are not compared to any other variable and do not appear in the definition of Adult).
- In such cases (so that a programmer does not have to invent new names for dummy variables) a variable may be replaced by an underscore:

Adult(x) ← Person(x,y,_,_) AND y ≥ 18

Datalog query

- A ***Datalog query*** is a finite set of Datalog rules
- If there is only one relation which appears as a head of a rule in the query, the tuples in that relation are taken as the answer to the query.
- For example,

Parent(x,y) ← Mother(x,y)

Parent(x,y) ← Father(x,y)

defines Parent relation (using relations Father and Mother)

- If there is more than one relation appearing as a head, one of them is the main predicate to be defined and others are auxiliary.

Datalog and logic programming (Prolog)

- Datalog programs look very similar to logic programs.
- Main differences are:
  - there are no functional symbols in Datalog
  - the use of programs is different:
    - Datalog program is a mapping from the database to the set of new relations; it is assumed that the program is small and the database is large
    - Logic program contains (a small amount of) data as part of the program; it would just list "facts" such as

Mother(Jane, Mary) ←

Meaning of Datalog rules

- First approximation (non-recursive queries):
  - take the values of variables which make the body of the rule true (make each subgoal true; NOT R is true if R is false)
  - see what values the variables of the head take;
  - add the resulting tuple to the predicate in the head of the rule.

Example

- Suppose we have a relation Person over schema (Name, Age, Address, Telephone). Then the following Datalog rule will define a relation which contains names of people aged over 18:

Adult(x) ← Person(x,y,z,u) AND y ≥ 18

- We take all <name, age, addr, tel> in Person for which it is also true that age ≥ 18, and add <name> to Adult.
- Adult relation is the same as $\pi_{\text{Name}}\, \sigma_{\text{Age} \geq 18}$ (Person).

Another example

Parent(x,y) ← Mother(x,y)

Parent(x,y) ← Father(x,y)

- suppose Mother contains tuples <Jane,Mary> and <Jane,Bill> and Father contains tuples <Tom,Mary>, <Tom,Bill>.
- All four tuples are added to Parent.

Example with negation

- Suppose we have a relation Person over schema (Name, Age, Address, Telephone).

  Child(x) ← Person(x,y,z,u) AND NOT(y ≥ 18)
- We take all <name, age, addr, tel> in Person for which it is also true that NOT(age ≥ 18), and add <name> to Adult.
- NOT(age ≥ 18) is true if age ≥ 18 is false, so we add all tuples where age < 18.

Safe queries

- We want the result of a query to be a finite relation.
- To ensure this, the following ***safety condition*** is required:

every variable that appears anywhere in the rule must appear in some non-negated relational subgoal.

- The reason for this is that infinitely many values may satisfy an arithmetical subgoal (e.g. x > 0) and infinitely many values are NOT in some finite table of a relation R.

Datalog and relational algebra

- Every relation definable in relational algebra is definable in Datalog.
- Again we assume that we have a relational name (predicate symbol) R for every basic relation **R**.
- Then for every operation of relational algebra, we show how to write a corresponding Datalog query.

Union

- Union of **R** and **S**:

$U(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$
$U(x_1,\ldots,x_n) \leftarrow S(x_1,\ldots,x_n)$

Difference

- Difference of **R** and **S**:

$D(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$ AND NOT $S(x_1,\ldots,x_n)$

Product

- Product of **R** and **S**:

$P(x_1,\ldots,x_n,y_1,\ldots,y_k) \leftarrow R(x_1,\ldots,x_n)$ AND $S(y_1,\ldots,y_k)$

Projection

- Suppose we want to project **R** on attributes $x_1,\ldots,x_n$.

$P(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n,y_1,\ldots,y_k)$

or

$P(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n,\_,\ldots,\_)$

Selection

- Simple case: all conditions in the selection are connected by AND, for example $\sigma_{\textbf{Age > 18 AND Address = "London"}}$ (Person)

Answer(x,y,z,u) ← Person(x,y,z,u) AND y > 18 AND z = "London"

- If conditions are connected with OR, need more than one rule. For example, $\sigma_{\textbf{Age > 18 OR Address = "London"}}$ (Person)

Answer(x,y,z,u) ← Person(x,y,z,u) AND y > 18

Answer(x,y,z,u) ← Person(x,y,z,u) AND z = "London"

Compound queries

- To translate an arbitrary algebraic expression, create a new predicate for every node in the query tree.
- For example, to do $\sigma_{\text{Name1 = Name2}}$ (R × P):
  - Define predicate S = R × P
  - Define $\sigma_{\text{Name1 = Name2}}$ (S)

Informal coursework

- A database of fictitious company contains three relations:
  - GOODS over schema {Producer, ProductCode, Description}
  - DELIVERY over schema {Producer, ProductCode, Branch#, Stock#}
  - STOCK over schema {Branch#, Stock#, Size, Colour, SellPrice, CostPrice, DateIn, DateOut}.

Define in Datalog

- Query 1: find all producers who supply goods.
- Query 2: find all producers who have delivered goods to any branch of the company.
- Query 3: find SellPrice and CostPrice of all goods delivered to branch L1 still in stock (here, L1 is a value in the attribute domain of Branch#, and products in stock have value InStock for the DateOut attribute).
- Query 4: find Producer, ProductCode, Description for all goods sold at the same day they arrived at any branch.
- Query 5: find Branch#, Size, Colour, SellPrice for all dresses which have not yet been sold (dress is a value in the attribute domain of Description).

Reading

- Ullman, Widom, chapter 10, or
- Abiteboul, Hull, Vianu chapter 12.