CS 838: Foundations of Data Management **Spring 2016**

Lecture 1: Conjunctive Queries

Instructor: Paris Koutris 1/21/2016

Before we start, let us set up some basic notation.

A *database schema* $\mathbf{R}$ is a set of relations: we will typically use the symbols $R, S, T, \dots$ for relations. The *arity* of a relation R is the number of attributes in the relation. This is the *unnamed* perspective of a relational schema, since we do not associate a name with each attribute in the relation. Contrast this to defining a relational schema as $R(A, B)$, where we can refer to each of the attributes as $R.A$ and $R.B$ respectively.

The attributes in a relation can take values from the same domain **dom**, which is a countably infinite set. We can assume that each attribute takes values from a different domain (in this case we would denote the domain as $\mathbf{dom}(A)$), but from a theoretical perspective it almost always suffices to have a single shared domain. A *constant* is an element of the domain **dom**.

Let R be a relation of arity m. A *fact* over R is an expression of the form $R(a_1, \dots, a_m)$, where $a_i \in \mathbf{dom}$ for every $i = 1, \dots, m$. An *instance* of the relation R is a finite set of facts over R. A *database instance* I over a database schema $\mathbf{R}$ is a union of relational instances over the relations $R \in \mathbf{R}$. In this case, we denote R^I the instance of relation $R \in \mathbf{R}$. Given a database instance I, we define the *active domain* of I, denoted $\mathbf{adom}(I)$ as the set of all constants occurring in I.

1.1 Basics of Conjunctive Queries

Conjunctive queries are the simplest form of queries that can be expressed over a database, but as we will see they have many interesting properties and a deep theory behind them.

There are many ways to define a conjunctive query, and we will do it from a logical perspective first, using *Datalog* notation. Syntactically, a *conjunctive query* q (or simply CQ) is an expression of the form

$$q(x_1, \dots, x_k) : -R_1(\vec{y_1}), \dots, R_n(\vec{y_n}) \tag{1.1}$$

where $n \geq 0$, $R_i \in \mathbf{R}$ for every $i = 1, \dots, n$ and q is a fresh relation name. The expressions $\vec{x}, \vec{y_1}, \dots, \vec{y_n}$ are called free tuples, and contain either variables or constants. We will typically name the variables $x, y, z, \dots$, and the constants $a, b, c, \dots$ The tuples $\vec{y_i}$ must match the arities of the corresponding relation. Also, every variable in $\vec{x} = \langle x_1, \dots, x_k \rangle$ must appear at least once in $\vec{y_1}, \dots, \vec{y_n}$.

The expression $q(x_1, \dots, x_k)$ is called the *head* of the query, and $R_1(\vec{y_1}), \dots, R_n(\vec{y_n})$ is called the *body* of the query. Each expression $R_i(\vec{y_i})$ is called an *atom*: notice that the atom is different from a relation, since many atoms can correspond to the same relation! The set of variables in the query is denoted $\mathbf{var}(q)$.

Example 1.1. *Below are some examples of conjunctive queries:*

$$\begin{aligned}
q_1(x,y) &: -R(x,y), S(y,z) \\
q_2() &: -R(x,y), S(y,a), T(x) \\
q_3(x,y,z) &: -R(x,y), R(y,z), R(z,x) \\
q_4(x,b) &: -R(x,x), S(x,y)
\end{aligned}$$

When writing CQs in Datalog form, we can also choose to use equality: for example, the query $q(x) : -R(x,y), y = a$ is a valid CQ. However, we are not allowed to use any other predicate symbols, such as $<, \leq, >, \geq, \neq$.

1.1.1 Semantics

So far we looked at the syntactic definition of a conjunctive query. We now turn our attention to the semantics of CQs. The intuition here is that we will try to match to each variable of the body a value from the domain such that the body is true, and then we can infer a new fact from the head of the query.

Formally, we define a *valuation* v over a set of variables V as a total function[1] from V to the domain **dom**. For example, for query q_1, the function v, where $v(x) = a, v(y) = b, v(z) = c$ is a valuation. We extend the valuation to be the identity from **dom** to **dom**, and then extend it naturally to map free tuples to tuples over **dom**. For example, $v(\langle x,y\rangle) = \langle v(x), v(y)\rangle = \langle a,b\rangle$, and $v(\langle x,y,c\rangle) = \langle v(x), v(y), v(c)\rangle = \langle a,b,c\rangle$.

We can now formally define the semantics for CQs. Let I be a database instance over the schema **R**. Then, for the conjunctive query q, as given in (1.1):

$$q(I) = \{v(\vec{x}) \mid v \text{ is a valuation over } \mathbf{var}(q) \text{ and } \forall i = 1, \ldots, n : v(\vec{y}_i) \in R_i^I\}$$

Observe that the query q returns a new relational instance over a new schema defined by the head of the query.

Exercise 1.2. *Evaluate the queries q_1, q_2, q_3 over the database instance*

$$I = \{R(a,a), R(a,b), R(b,c), R(c,a), S(b,c), S(b,b), T(a)\}.$$

In Datalog terminology, the expressions $R_i(\vec{y}_i)$ are also called *subgoals*. The relations $R_1, \ldots, R_n$ are called *extensional relations*, since they are provided as input to the query. The relation q is called *intensional relation*, since its content is only given by "intension" or "definition" through the query.

[1] A total function is a function defined for all possible input values.

1.1.2 Equivalent Formalisms

In the formalism of relational calculus, the query q can be written as follows:

$$\{x_1, \ldots, x_k \mid \exists z_1, \ldots, z_m (R_1(\vec{y_1}) \wedge \cdots \wedge R_n(\vec{y_n}))\}$$

where $z_1, \ldots, z_m$ are the variables that appear in the body, but not in the head of the query q. Notice that this is a first-order logical formula that consists of only existential quantification, followed by a conjunction of atoms: this is the reason why this class of queries is called conjunctive queries. As an example, q_1 would be expressed in relational calculus as follows:

$$\{x, y \mid \exists z (R(x, y) \wedge S(y, z))\}$$

The above formalism in relational calculus is equivalent to the Datalog definition of CQs. The other formalism that is equivalent is the class of SPJ queries in relational algebra: these are queries that contain only selections (S), projections (P) and joins (J). Notice that the relational algebra formalism is procedural, in contrast to the other formalisms that are *declarative*; in other words, they specify what the result of a query is instead of specifying how to compute it.

Exercise 1.3. *Express the queries $q_1, \ldots, q_4$ in relational algebra and relational calculus.*

In SQL, conjunctive queries correspond to SELECT FROM WHERE queries, where the WHERE conditions contain only equalities.

1.1.3 More Definitions

Definition 1.4 (Monotonicity)**.** *We say that a query q is* monotone *if for instances $I \subseteq J$, it holds that $q(I) \subseteq q(J)$.*

Lemma 1.5. *Every conjunctive query is monotone.*

Proof. Consider some tuple $t \in q(I)$. Then, there exists a valuation v over $\mathbf{var}(q)$ such that $t = v(\vec{x})$, and for every R_i, we have $v(\vec{y_i}) \in R_i^I$. Since $I \subseteq J$, we have that $v(\vec{y_i}) \in R_i^J$, and thus $t \in q(J)$ as well. $\square$

The above lemma immediately tells us that there are queries that cannot be written as a conjunctive query (every non-monotone query).

When the head of a conjunctive query is of the form $q()$, then we say that q is a *boolean* CQ. For example, the query q_2 is boolean. The answer to a boolean CQ is essentially a yes or no, depending on whether the answer is the set containing a single tuple with no attributes $\{\langle\rangle\}$, or it is the empty set $\{\}$ respectively.

1.2 Beyond Conjunctive Queries

If we add inequality ($\neq$) to conjunctive queries we obtain the class $CQ^{\neq}$. For example, we can now express the following query: *"Return the endpoints for paths of length 3 that start and end in different vertices."*

$$q(x,w) : -R(x,y), R(y,z), R(z,w), x \neq w.$$

The above query cannot be expressed as a standard CQ! If we add $<, \leq, >, \geq$, we obtain the class $CQ^{<}$. In this case, we also have to assume a total order on the values of the domain **dom**. For example, we can express the following query: *"Return the endpoints for paths of length 3 with strictly increasing value of vertices."*

$$q(x,w) : -R(x,y), R(y,z), R(z,w), x < y, y < z, z < w.$$

The other class of queries of interest to us is the Union of Conjunctive Queries (UCQ) class. A UCQ is a query of the form $q_1 \cup q_2 \cup \ldots q_m$, where each q_i is a conjunctive query. For example, the query $q = q_1 \cup q_2$, where $q_1(x,y) = R(x,z), R(z,y)$, and $q_2(x,y) = R(x,z), R(z,w), R(w,y)$ is a UCQ that returns the endpoints of paths of length 2 or 3. The class of UCQs corresponds to the fragment of relational algebra that uses Selection, Projection, Joins and Union (SPJU), and to relational calculus queries that uses $\exists, \vee, \wedge$ (so no universal quantifier $\forall$ or negation).

The query languages $CQ^{\neq}, CQ^{<}, UCQ$ are all monotone (prove as an exercise why).