Logic and Probability, EPFL, Fall 2011
Lecture Notes 6
Val Tannen

14 Rule-based Formalisms

The extension of FOL with fixpoints is very attractive from a computational perspective since these extensions were shown to capture PTIME and PSPACE over ordered structures (in contrast, FOL by itself only defines properties computable in AC0).

However the syntax of FO(FP) is quite cumbersome. Since the 1970's rule-based formalisms, under the name *logic programming*, have been proposed as a relatively practical alternative. A *rule*, a.k.a. *Horn clause*, is a sentence of the form

$$\forall \overline{x}\ \gamma \rightarrow \alpha$$

where γ is a conjunction of atoms and α is a single atom. Rules are usually written in Prolog syntax

$$\alpha :- \gamma \quad \text{or} \quad \alpha \leftarrow \gamma$$

with the universal quantifiers left implicit. Also, the conjunction symbol is usually replaced by a comma.

Prolog programs consist, essentially, of a set of rules over an FO vocabulary and Datalog programs are, essentially, Prolog programs restricted to a relational vocabulary.

The fact that these programs are syntactically sets of FO sentences does not mean that what they compute is expressible in FOL. Several equivalent semantics can be given for Prolog/Datalog: model-theoretic, fixpoint and proof-theoretic. For example, here is how we express the path predicate in Datalog:

$$\begin{aligned} P(u,v) &:- E(u,v) \\ P(u,v) &:- E(u,w), P(w,v) \end{aligned}$$

Notice the close analogy with the inductive definition of the sequence of formulas that we used to translate fixpoints into infinitary disjunctions.

In databases we use rule-based formalisms as query languages. The simplest one uses just single rules, see next.

14.1 Conjunctive Queries

The material in this subsection assumes knowledge of Homework 1.

Conjunctive queries are first-order queries of a particular form: $\{\mathbf{x} \mid \exists \mathbf{y}\, \varphi\}$ where φ is a conjunction of atoms. Example: $\{(x,z) \mid \exists y\, R(x,y) \wedge z = y \wedge S(y,x)\}$. This query is clearly domain-independent, but the following are not: $\{(x,y) \mid R(x)\}$, $\{(x,y) \mid \exists z\, R(x) \wedge y = z\}$ Therefore, we need some additional restrictions to make sure we get only domain-independent queries.

Definition 14.1 A *tableau* T is a set of atoms (relational or equality) that is *range-restricted*, that is, for any variable $x \in var(T)$ either $T \vdash x = c$ for some constant c or $T \vdash x = x'$ for some variable x' that occurs in a relational atom in T.

Here $T \vdash e = e'$ means that $e = e'$ can be derived from the formulas in T via reasoning in FO logic. By completeness, this is the same as $T \models e = e'$. It is clear that the relational atoms in T do not matter and that this kind of consequence is easily decidable, hence so is $T \vdash e = e'$.

Definition 14.2 A *conjunctive query* is a given by a pair $\langle\, \mathbf{u}, T \,\rangle$ where T is a tableau and $\mathbf{u}$ is a tuple of variables or constants (the "output" tuple) [8] such that $var(\mathbf{u}) \subseteq var(T)$. The corresponding FO query is $\{\mathbf{u} \mid \exists \mathbf{y}\, T\}$ where $\mathbf{y} = var(T) \setminus var(\mathbf{u})$ and where we "read" T as the conjunction of its elements.

The range-restriction condition on the tableau part ensures domain-independence (see below).

Here are two examples written in a Prolog-like, or "rule-based", formalism:

$$\begin{array}{lll} ans(x, y) & :- & R(x, z),\ x = c,\ S(x, y, z) \\ ans(c, y) & :- & R(c, z),\ S(c, y, z) \end{array}$$

In the spirit of rule-based/logic programming, the output tuple of conjunctive queries is sometimes called the "head" of the query and the tableau part the "body" of the query.

Notice that the two queries given above are equivalent. It is natural to ask if we can always get rid of equality atoms in a conjunctive query. It turns out that this is the case iff the query is "satisfiable" (see below). Note however that the transformation may introduce constants in the head (as above) or may equate some of the variables in the head, eg.,

$$\begin{array}{lll} ans(x, y) & :- & R(x),\ x = y \\ ans(x, x) & :- & R(x) \end{array}$$

For conjunctive queries without equality atoms we *cannot* assume without loss of generality that the output tuple consists of distinct variables.

We have stated that conjunctive queries are particular cases of FO queries and this defined their semantics. However, it is worthwhile observing that $\mathcal{I}, v \models \exists \mathbf{y} T$ iff $\mathcal{I}, \beta \models T$ for some extension β of v. More precisely:

Definition 14.3 If T is a tableau and $\mathcal{I}$ and instance, a *valuation* for T in $\mathcal{I}$ is a function $\beta : var(T) \longrightarrow \mathbb{D}$. We extend β to map any constant to itself. Moreover, we say that that the valuation β *satisfies* T if

- for any atom $R(\mathbf{e}) \in T$ the relation $R^{\mathcal{I}}$ contains $\beta(\mathbf{e})$, and
- for any atom $e = e' \in T'$ we have $\beta(e) = \beta(e')$.

Proposition 14.1 *For any conjunctive query $q = \langle\, \mathbf{u}, T \,\rangle$ and any instance $\mathcal{I}$*

$$q(\mathcal{I}) = \{\beta(\mathbf{u}) \mid \textit{valuation } \beta \textit{ satisfies } T \textit{ in } \mathcal{I}\}.$$

Here is a quick application of this proposition.

Proposition 14.2 *All conjunctive queries are domain-independent.*

Another application is to characterize and decide satisfiability for conjunctive queries.

[8] Without loss of generality we can assume that the output tuple consists of distinct variables.

Proposition 14.3 *A conjunctive query $q = \langle \mathbf{u}, T \rangle$ is unsatisfiable iff $T \vdash c_1 = c_2$ for two distinct constants c_1, c_2. In particular, satisfiability is easily decidable.*

Proof By proposition 14.1 q is satisfiable iff there exists an instance and a valuation that satisfy T. If $T \vdash c_1 = c_2$ then any valuation must equate c_1 and c_2 which is impossible if they are distinct. This gives us one direction of the proposition. For the converse, suppose that $c_1 \equiv c_2$ whenever $T \vdash c_1 = c_2$. Then, for any $x \in var(T)$ there is at most one constant c such that $T \vdash x = c$. Let γ map every variable in $x \in var(T)$ to such a unique c if it exists. If it doesn't, let γ map x to a fixed constant c_0 that we choose apriori. Extend γ to map any constant to itself. Build an instance $\mathcal{C}_0$ as follows:

$$R^{\mathcal{C}_0} \stackrel{\text{def}}{=} \{\gamma(\mathbf{e}) \mid R(\mathbf{e}) \in T\}$$

We claim that γ is a valuation that satisfies T. The relational atoms are satisfied by construction. If $x = c \in T$ then $\gamma(x) = c = \gamma(c)$. If $x_1 = x_2 \in T$ then (1) either $T \vdash x_1 = c$ for some c in which case we also have $T \vdash x_2 = c$ and therefore $\gamma(x_1) = c = \gamma(x_2)$, (2) or $\gamma(x_1) = c_0 = \gamma(x_2)$. □

Note that proposition 14.3 implies that satisfiability depends only on the tableau part of the query. We can therefore talk about (un)satisfiable tableaux. It is easy to show that any equality-free conjunctive query is satisfiable and that for any satisfiable conjunctive query there is an equality-free conjunctive query equivalent to it.

Proposition 14.1 also implies that conjunctive queries are in NP, which is (probably!) better than general FO queries which are PSPACE-complete (all this is for combined complexity):

Theorem 14.4 *The combined complexity of the recognition problem for conjunctive queries is NP-complete.*

Proof Since the queries are domain-independent, it is sufficient for the valuations to take values in $adom(\mathcal{I}) \cup adom(q)$ instead of all of $\mathbb{D}$. Note that the size of such a valuation is polynomial. Then, to test if $\mathbf{a} \in q(\mathcal{I})$ where $q = \langle \mathbf{u}, T \rangle$, we can guess such a poly-size finite valuation β and check in polynomial time that it satisfies the relational and equality atoms of T and that $\mathbf{a} = \beta(\mathbf{u})$. Hence the problem is in NP.

To prove NP-hardness, we reduce CLIQUE to our problem. Given a graph G and a number $n > 1$, consider the relational schema with just one binary relation symbol E and construct the instance $\mathcal{I}_G$ that corresponds to the set of edges (pairs of vertices) of the graph G. Then consider the query $\langle (), T_{clique} \rangle$ where $()$ is the empty (0-ary) tuple, $var(T_{clique}) = \{x_1, \ldots, x_n\}$ and

$$T_{clique} = \bigwedge_{1 \le i \ne j \le n} E(x_i, x_j)$$

Clearly G has a clique of size n iff $() \in \langle (), T_{clique} \rangle(\mathcal{I}_G)$. □

The reduction above from CLIQUE is specifically for combined complexity. Using a reduction from 3-colorability it can be shown that the expression complexity of the recognition problem is also NP-hard. (It is of course in NP, same proof as above.)

The conjunctive queries correspond to a specific fragment of the relational algebra, namely the fragment that uses only the selection, projection, and cartesian product operations. We call this fragment the *SPC algebra*.

Theorem 14.5

1. *There is an effective translation that takes every conjunctive query into an equivalent SPC algebra expression.*

2. *There is an effective translation that takes every SPC algebra expression into an equivalent conjunctive query.*

Proof Sketch For example, the query $ans(c, y) :- R(c, z), S(c, y, z)$ is translated to the expression $\pi_{14}(\sigma_{1c}(\sigma_{3c}(\sigma_{25}(R \times S))))$. This suggests the idea behind the translation in part (1) of the theorem: start with the cartesian product of the occurrences of the relations in the body, continue with selections determined by the equality atoms, by the use of the same variable in several relational atoms and by the occurrence of constants in the relational atoms, and end with a projection corresponding to the positions of the variables that appear in the output. For part (2) the translation is defined by induction on SPC algebraic expression and I am going to skip it. Notice however that conjunctive queries always translate into SPC expressions of a special form: a cartesian product followed by several selections followed by a projection. This is a *normal form* for SPC algebra expressions. By composing the two translations in this theorem we can translate any SPC expression into a normal form. □

Via the translation to the SPC algebra we see that conjunctive queries correspond closely to certain SQL programs. For example, the query $ans(x, y) :- R(x, z), x = c, S(x, y, z)$ corresponds to

```
select r.1, s.2
from   R r, S s
where  r.1=c and s.1=r.1 and r.3=s.2
```

Such SQL programs, in which the "where" clause is a conjunction of equalities arise often in practice. So, although restricted, conjunctive queries are important.

The best evidence for the importance of conjunctive queries comes from the fact that their equivalence is decidable, while equivalence of relational algebra expressions (therefore FO queries) is be shown to be undecidable, just as the equivalence of two FO sentences is undecidable. The decidability of conjunctive query equivalence plays an important role in query *optimization*.