CS632 Problems I

A. Demers

20 Feb 2003 – due 27 Feb 2003

Here is a set of exercises in stream-of-consciousness style. The topics progress backwards, with more recent material at the beginning and earlier material at the end.

Consult freely, both inanimate and animate resources are fine, but please *reference your sources* and write up your answers individually.

1 Inference Rules for FDs

Recall the set $\{FD_1, FD_2, FD_3\}$ of inference rules for Functional Dependencies:

$$\frac{}{X \rightarrow X}$$

$$\frac{X \rightarrow YZ}{X \rightarrow Y}$$

$$\frac{X \rightarrow YZ \qquad Z \rightarrow W}{X \rightarrow YZW}$$

We showed these rules to be sound and complete. A set $\mathcal{S}$ of inference rules is *independent* if every proper subset $\mathcal{S}' \subset \mathcal{S}$ is strictly weaker than $\mathcal{S}$; that is, if for every S' there exists a set F of constraints and a constraint f such that

$$F \vdash_{\mathcal{S}} f \qquad \text{but not} \qquad F \vdash_{\mathcal{S}'} f$$

The focus here is a bit different from lecture: here we concentrate on the size of the set S of inference rules, rather than on size of the set F of FDs.

Problem 1a: Show that the set $\{FD_1, FD_2, FD_3\}$ is independent.

Problem 1b: Suppose we replace FD_3 by the apparently weaker rule FD_{3a}

$$\frac{X \rightarrow Y \qquad Y \rightarrow Z}{X \rightarrow YZ}$$

Is the revised set of rules complete? Justify your answer.

Problem 1c: Suppose we replace FD_{3a} by the still weaker rule FD_{3b}

$$\frac{X \rightarrow Y \qquad Y \rightarrow Z}{X \rightarrow Z}$$

Is the revised set of rules complete? Justify your answer.

Yes, I realize this part sort of gives away the answer to the previous part.

2 FD Covers

Recall the *length* of a FD f is the sum of the lengths of its left- and right-hand sides; that is,

$$\|X \rightarrow Y\| \quad = \quad |X| \; + \; |Y|$$

The length of a set F of FDs is just the sum of the lengths of all $f \in F$.

Consider a relation scheme $R(A_1 \ldots A_m B_1 \ldots B_n)$. We want to impose the constraint "$A_1 \ldots A_m$ is a superkey;" or, expressed as a FD,

$$A_1 \ldots A_m \;\rightarrow\; A_1 \ldots A_m B_1 \ldots B_n$$

So we seek a *cover*, a set of FDs F such that

$$F^+ \quad = \quad \{\; A_1 \ldots A_m \;\rightarrow\; A_1 \ldots A_m B_1 \ldots B_n \;\}^+$$

Problem 2: What is the minimum-length ("optimum") cover? What is the maximum-length nonredundant cover, and what is its length (asymptotically as a function of m and n)? Explain your answers.

3 TRC Variants

In the notes we defined TRC expressions by

$$E ::= \{ \ x(X) \mid F \ \} \quad \text{where } FV(F) = \{x\}$$

and showed these are equivalent to DRC. As a corollary the domain-independent TRC expressions are equivalent to RA. For now, call this syntax “TRC-0.”

We also defined a more complicated syntax, with a somewhat restrictive form of range declarations, which we compared to SQL. Here is a revised TRC syntax based on that more complicated definition, but without range declarations (yet). For now, call this syntax “TRC”

$$E ::= \{ \ A_1 : x_1[B_1] \ldots A_m : x_m[B_m] \mid y_1(Y_1) \ldots y_n(Y_n) \mid F \ \}$$

We require that the variables x_i and the free variables of F are among the variables y_j, and that each B_i is one of the attributes in Y_j of the corresponding y_j. Informally, a tuple $t(A_1 \ldots A_m)$ is in the result of applying such an expression if there exist tuple values $t_1(Y_1) \ldots t_n(Y_n)$ for which the formula F is *true*, and result t is constructed by selecting the appropriate B_i attributes from the t_j's.

Problem 3a: Argue that TRC as described above is equivalent to TRC-0; that is, given an expression E in TRC-0, show how to construct an equivalent expression E' in TRC, and *vice versa*.
□

In lecture we described a TRC language with range declarations, TRC-RD, obtained by tagging each variable y_j in a TRC expression with a relation name:

$$E ::= \{ \ A_1 : x_1[B_1] \ldots A_m : x_m[B_m] \mid y_1(Y_1) : R_{i_1} \ldots y_n(Y_n) : R_{i_n} \mid F \ \}$$

Informally, variable y_j is allowed to range only over tuples contained in the relation named R_{i_j} in $\boldsymbol{r}$; of course this relation must have attributes Y_j.

We claimed (with a handwaving proof) the following, which is Theorem 2.7 in [AD93]:

Theorem 2.7: TRC-RD is equivalent to an algebraic language that has all the RA operators except union and does not include relation-valued constants.

You may assume this result below.

Problem 3b: Show that adding union to TRC-RD (to get TRCU-RD) makes it equivalent to RA (with union, but without relation-valued constants). Unions are added "at the top level" only; so the syntax is

$$E ::= E_1 \cup \ \dots \ \cup E_k$$
$$\text{where } E_i \text{ is a TRC-RD expression}$$

Note that unions can appear "inside" a general RA expression, as in

$$\sigma_p(E_1 - (E_2 \cup E_3))$$

while they can appear only at the top level of a TRCU-RD expression. This is the only reason the result is not immediate.
□

Instead of requiring range declarations, we can define a TRC language with implicit ranges, TRC-IR. Syntactically, a TRC-IR expression is identical to a TRC expression. The difference lies in the semantics. The result of a TRC-IR expression E is

$$[\![E]\!]_{(D_{\boldsymbol{r}} \cup C(E))}(\boldsymbol{r})$$

where $C(E)$ is the set of constants that are mentioned in E, and the subscript $(D_{\boldsymbol{r}} \cup C(E))$ specifies the data domain over which the values of quantified variables may range. (This subscript convention is used in the notes in the discussion of Domain-Independence.) Informally, we compute the result of a TRC-IR expression E applied to instance $\boldsymbol{r}$, by restricting ourselves to data values that are explicitly mentioned in E or occur in $\boldsymbol{r}$.

It is (I hope) self-evident that TRC-IR is domain-independent, and that its semantics coincides with TRC if the data domain is $(D_{\boldsymbol{r}} \cup C(E))$. Thus, TRC-IR is no more expressive than TRC-DI (which is equivalent to RA, DRC-RD, and TRCU-RD).

Problem 3c: Show that TRC-IR is equivalent to RA, by showing that TRC-IR can simulate TRCU-RD, which is equivalent to RA.
□

4 Quantifying Over Relations

We have given calculi with variables that range over scalars and tuples. You might find it natural to move one step "up" and allow relation-valued variables.

Here we explore this a bit, describing languages RRC and RRC-IR by analogy with what we've done above.

The syntax for RRC is a simple extension of that for TRC: we introduce an infinite set of typed *relation-valued variables* of the form $u(X)$, completely analogous to (and disjoint from) the tuple-valued variables $x(X)$. We do not allow relation-valued variables to appear at "top level", so the syntax remains

$$E \ ::= \ \{ \ A_1 : x_1[B_1] \ldots A_m : x_m[B_m] \mid \\ y_1(Y_1) \ldots y_n(Y_n) \mid F \ \}$$

However, we add two clauses to the syntax of formulas F. A formula can test whether a tuple-valued variable is in a relation-valued variable:

$$F \ ::= \ (x(X) \ \in \ u(X))$$

and a formula can quantify over relations with a given set of attributes:

$$F \ ::= \ (\exists u(X))F_1$$

There are two versions of the semantics. With unbounded quantification (RRC) variables (both tuple-valued and relation-valued) range over the entire data domain. With bounded quantification (RRC-IR) variables (of both kinds) range over $(D_{\boldsymbol{r}} \cup C(E))$.

As you might expect, quantifying over relations adds considerable power.

Problem 4a: Show how to express transitive closure in RRC-IR. That is, given the database schema $\boldsymbol{R} = (R_1(AB))\}$, construct an RRC-IR expression E such that for any instance $\boldsymbol{r} = (r_1)$,

$$[\![E]\!]_{(D_{\boldsymbol{r}} \cup C(E))}(\boldsymbol{r}) \quad = \quad (r_1)^+$$

that is, the result is the transitive closure of r_1. Explain your answer.
□
Unfortunately quantifying over relations adds *too much* power to be practical – the cost of evaluating a RRC or RRC-IR query expression is too high.

Problem 4b: (Optional) Prove zero to two of the following:

Show that given a RRC expression E, a database instance $\boldsymbol{r}$, and a tuple value t, it is undecidable whether t is in $[\![E]\!](\boldsymbol{r})$.

Show that given a RRC-IR expression E, a database instance $\boldsymbol{r}$, and a tuple value t, it is NP-complete to determine whether t is in $[\![E]\!](\boldsymbol{r})$.

The proofs I have in mind are nearly identical.
□