4 Domain Relational Calculus

We now present two relational "calculi" that we will compare to RA. First, what is the difference between an "algebra" and a "calculus?"

The usual story is that the algebra RA is *operational* – a RA expression describes a step-by-step procedure for computing a result relation. A calculus, on the other hand, is *declarative* – it is essentially a language of predicates which describe whether a tuple value (or a collection of scalar values) is contained in the result. Thus, in principle, a database management system has more freedom implementing a calculus based query language than an algebraic one. Of course, RA is a side-effect-free expression language, which in the programming languages community probably would not be considered operational. So the distinction is slightly dubious . . .

Nevertheless, we present the first of our two calculi, the *Domain Relational Calculus* (DRC). Informally, this is a language of predicates whose variables range over scalar values in the data domain D. A query expression can construct a tuple and ask (with a predicate) whether it is present in a given relation in the database instance. A DRC expression has the following form:

$$\begin{aligned} E \ ::= \ & \{ \ \langle A_1 : x_1, \ldots A_n : x_n \rangle \mid F \ \} \\ & \text{where } FV(F) \ \subseteq \ \{x_1, \ldots, x_n\} \end{aligned}$$

All the work is done in F, which is a first-order formula (formulas are described below). With each formula F we associate a set $FV(F)$ of its *free variables.* The truth value of a formula is determined using values assigned to its free variables. The assignment of values to free variables is given by a *substitution*, formally a function that maps the (countably infinite) set of variable names into the data domain D. Putting this together, the meaning of a formula F is given as

$$[\![F]\!](s, \boldsymbol{r})$$

where $\boldsymbol{r}$ is a database instance (as in the semantics of RA) and s is a substitution (which supplies values for the free variables of F).

We define formulas, free variables, and the semantics of formulas simultaneously as follows:

Atomic Formulas

$$\begin{aligned} F \ ::= \ & \langle B_1 : x_1, \ldots, B_n : x_n \rangle \ \in \ R_i) \\ & \text{where } R_i(B_1 \ldots B_n) \in \boldsymbol{R} \end{aligned}$$

$$FV(F) \;=\; \{x_1, \ldots, x_n\}$$
$$[\![F]\!](s, \boldsymbol{r}) \;=\; (\; (t : \forall i : B_i \mapsto s[x_i]) \;\in\; r_i \;)$$

$$F \;::=\; x_1 \;=\; x_2$$
$$FV(F)(s, \boldsymbol{r}) \;=\; \{x_1, x_2\}$$
$$[\![F]\!](s, \boldsymbol{r}) \;=\; (s[x_1] \;=\; s[x_2])$$

$$F \;::=\; x \;=\; c$$
$$FV(F) \;=\; \{x\}$$
$$[\![F]\!](s, \boldsymbol{r}) \;=\; (s[x] \;=\; v)$$

where $v \in D$ is the value denoted by c.

Compound Formulas

$$F \;::=\; \neg F_1$$
$$FV(F) \;=\; FV(F_1)$$
$$[\![F]\!](s, \boldsymbol{r}) \;=\; \neg[\![F_1]\!](s, \boldsymbol{r})$$

$$F \;::=\; F_1 \;\wedge\; F_2$$
$$FV(F) \;=\; FV(F_1) \cup FV(F_2)$$
$$[\![F]\!](s, \boldsymbol{r}) \;=\; [\![F_1]\!](s, \boldsymbol{r}) \;\wedge\; [\![F_1]\!](s, \boldsymbol{r})$$

$$F \;::=\; \exists x F_1$$
$$FV(F) \;=\; FV(F_1) - \{x\}$$
$$[\![F]\!](s, \boldsymbol{r}) \;=\; \exists v \in D \;(\; [\![F_1]\!](s[x \leftarrow v], \boldsymbol{r})$$

The notation $s[x \leftarrow v]$ denotes a substitution that agrees with s on every argument except x, and returns v on argument x. In the semantics, the value of x ranges over the *infinite* set D; hence x necessarily takes on values that are not in $D_{\boldsymbol{r}}$.

Now we can give the semantics of a DRC expression E:

$$\begin{aligned} &[\![\; \{\langle A_1 : x_1, \ldots A_n : x_n\rangle \mid F\} \;]\!](\boldsymbol{r}) \\ &\quad = \;\{\; t : \; A_1 \mapsto v_1, \ldots, A_n \mapsto v_n \mid \\ &\qquad\qquad [\![F]\!](s_0[x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n, \ldots], \boldsymbol{r}) \;=\; \text{true} \;\} \end{aligned}$$

This is the set of tuples over D for which F is true, given the proper variable bindings. Since the free variables of F are included in $\{x_1, \ldots, x_n\}$, and the initial environment s_0 is updated on all of these variables, this definition is actually independent of s_0. Just as in quantified formulas, the variables range over the *infinite* set D, and so can take on values that are not in $D_{\boldsymbol{r}}$. Thus, the result can contain values that are not in $D_{\boldsymbol{r}}$, and may even be an infinite relation. This is technically important, as will be seen below.

5 Tuple Relational Calculus

We now describe our second relational calculus, the Tuple Relational Calculus (TRC). Here variables range over tuple values. A query expression can select attributes from a tuple and can apply predicates. Since variables represent tuples, each variable must be "typed" with the set of attributes over which it is defined, and uses of variables must be consistent with their types. We follow the convention that the type is *part of* the variable name, and write variables as

$$x(X) \qquad \text{or} \qquad x(A_1 \ldots A_k)$$

in the definitions below.

The syntax of a TRC expression is as follows:

$$\begin{aligned} E \ &::= \ \{\ x(X) \mid F\ \} \\ &\qquad \text{where } FV(F) \ = \{x\} \end{aligned}$$

As in the definition of DRC above, all the work is done in the formula F, and we give the semantics of F with respect to a substitution and a database instance:

$$[\![F]\!](s, \boldsymbol{r})$$

In TRC, substitutions map variables to tuple values. We require substitutions to be *well typed*: a substitution applied to variable $x(X)$ yields a tuple value defined on exactly the attributes X. You can verify that this invariant holds of every substitution occurring in the semantics below.

Atomic Formulas

$$\begin{aligned} F \ &::= \ x(X) \ \in \ R_i \\ &\qquad \text{where } R_i(X) \in \boldsymbol{R} \end{aligned}$$

$$FV(F) \;=\; \{x(X)\}$$

$$[\![F]\!](s,\boldsymbol{r}) \;=\; (\; s[x(X)] \;\in\; r_i \;)$$

$$F \;::=\; x_1(X_1).A_1 \;=\; x_2(X_2).A_2$$

where $A_1 \in X_1$ and $A_2 \in X_2$

$$FV(F)(s,\boldsymbol{r}) \;=\; \{x_1(X_1), x_2(X_2)\}$$

$$[\![F]\!](s,\boldsymbol{r}) \;=\; ((s[x_1(X_1)])[A_1] \;=\; (s[x_2(X_2)])[A_2])$$

$$F \;::=\; x_1(X_1).A_1 \;=\; c$$

where $A_1 \in X_1$

$$FV(F)(s,\boldsymbol{r}) \;=\; \{x_1(X_1)\}$$

$$[\![F]\!](s,\boldsymbol{r}) \;=\; (s[x_1(X_1)])[A_1] \;=\; v$$

where $v \in D$ is the value denoted by c.

Compound Formulas

$$F \;::=\; \neg F_1$$

$$FV(F) \;=\; FV(F_1)$$

$$[\![F]\!](s,\boldsymbol{r}) \;=\; \neg[\![F_1]\!](s,\boldsymbol{r})$$

$$F \;::=\; F_1 \;\wedge\; F_2$$

$$FV(F) \;=\; FV(F_1) \cup FV(F_2)$$

$$[\![F]\!](s,\boldsymbol{r}) \;=\; [\![F_1]\!](s,\boldsymbol{r}) \;\wedge\; [\![F_1]\!](s,\boldsymbol{r})$$

$$F \;::=\; \exists x(X) F_1$$

$$FV(F) \;=\; FV(F_1) - \{x(X)\}$$

$$[\![F]\!](s,\boldsymbol{r}) \;=\; (\exists v \in \mathcal{U})((Attr(v) \;=\; X) \;\wedge\; (\; [\![F_1]\!](s[x(X) \leftarrow v], \boldsymbol{r}))$$

Just as in DRC formulas, a quantified variable ranges over an infinite set – all compatibly typed tuples over the infinite data domain D (variable $x(X)$ ranges over all tuples with attributes X). Now we can give the semantics of a TRC expression E:

$$[\![\; \{x(X) \mid F\} \;]\!](\boldsymbol{r})$$

$$= \;\{\; t \in \mathcal{U} \mid (Attr(t) = X) \;\;\wedge$$

$$([\![F]\!](s_0[x(X) \leftarrow t], \boldsymbol{r}) \;=\; \text{true}) \;\}$$

As for DRC, this is the set of correctly typed tuples over D for which F is true. Since the only free variables of F is $x(X)$, this definition is independent of s_0. Just as in quantified formulas, the variable ranges over an *infinite* set, and so the result can contain values that are not in $D_{\boldsymbol{r}}$, and can even be an infinite relation.

6 Equivalence of DRC and TRC

We now show the equivalence of DRC and TRC. In the spirit of the Characterization Theorem for RA that we proved earlier, you might expect a result of the form "for every DRC expression E and every instance $\boldsymbol{r}$, there is a TRC expression $E'(E, \boldsymbol{r})$ that computes the same result:

$$[\![E'(E, \boldsymbol{r})]\!](\boldsymbol{r}) \quad = \quad [\![E]\!](\boldsymbol{r})$$

and *vice versa*. While this is true, we prove a much stronger equivalence: for every DRC expression E there is a TRC expression $E'(E)$ that computes the same function:

$$(\forall \boldsymbol{r})\ ([\![E'(E)]\!](\boldsymbol{r}) \quad = \quad [\![E]\!](\boldsymbol{r}))$$

and *vice versa.*.

Recall we proved the Characterization Theorem by induction on the structure of a RA expression. We cannot quite achieve this for relational calculus, because DRC and TRC expressions are not defined recursively. A RC expression E is just some bound variables and a formula F. It is the formula that is defined recursively, *not* the expression. To prove equivalence of DRC and TRC, we first define a sense in which a TRC formula F and a DRC formula F' can be considered equivalent, and then prove the equivalence of the expression languages by induction on the structure of formulas.

Roughly, we shall say F (a TRC formula) and F' (a DRC formula) are equivalent if the expressions

$$E \quad = \quad \{\ x(\ldots A_i \ldots) \mid F\ \} \qquad \text{and} \qquad E' \quad = \quad \{\ \langle \ldots A_i : x_i \ldots \rangle \mid F'\ \}$$

are equivalent (i.e. yield the same result when applied to any database instance).

In more detail, suppose the free variables of F are

$$FV(F) \ = \ \{\ldots, x_i(A_{i,1} \ldots A_{i,m}, \ldots\}.$$

and the free variables of F' are

$$FV(F') \ = \ \{\ldots, y_{i,1} \ldots y_{i,m}, \ldots\}$$

with the obvious correspondence between $y_{i,j}$ and $x_i[A_{i,j}]$. A TRC substitution s and a DRC substitution s' are *similar* (*for* F *and* F') if

$$(\forall i, j)(\ (s(x_i))[A_{i,j}] \ = \ s'(y_{i,j})$$

That is, s and s' represent the same values, reorganized to reflect the difference between the tuple-valued variables that occur free in F and the simple variables that occur free in F'.

Now, the sense in which F and F' are equivalent is this:

$$[\![F]\!](s, \boldsymbol{r}) \quad = \quad [\![F']\!](s', \boldsymbol{r})$$
$$\text{(for all } \boldsymbol{r}\text{and all substitutions } (s, s') \text{ similar for } (F, F'))$$

It should be clear that a TRC expression and a DRC expression are equivalent exactly when their top-level formulas are equivalent by the above definition. So it suffices to show the following:

Lemma: For every TRC formula F there is an equivalent DRC formula F'.

Proof: We show how to construct F' by induction on the structure of F.

Case $F \ = \ x_i(A_{i,1} \ldots A_{i,m}) \ \in \ R$:

Let

$$F' \quad = \quad \langle A_{i,1} : y_{i,1}, \ldots, A_{i,m} : y_{i,m} \rangle \ \in \ R$$

and the equivalence follows from the definition of similar substitutions.

Case $F \ = \ x_i(A_{i,j}) \ = \ x_k(A_{k,l})$:

Let

$$F' \quad = \quad y_{i,j} \ = \ y_{k,l}$$

and again the equivalence follows from the definition of similar substitutions.

Case $F \ = \ x_i(A_{i,j}) \ = \ c$:

This is similar to the previous case.

Case $F \;=\; \neg F_1$:

Let

$$F' \;=\; \neg F_1'$$

The equivalence of F and F' follows from the equivalence of F_1 and F_1', together with the observation that all substitution pairs (s, s') that are equivalent for F and F' are equivalent for F_1 and F_1' (since $FV(F)$ and $FV(F_1)$ are identical).

Case $F \;=\; F_1 \;\wedge\; F_2$:

Let

$$F' \;=\; F_1' \;\wedge\; F_2'$$

This is similar to the previous case. Since

$$FV(F) \;=\; FV(F_1) \;\cup\; FV(F_2)$$

it again follows that all substitution pairs (s, s') that are equivalent for F and F' are equivalent for F_1 and F_1', and for F_2 and F_2'.

Case $F \;=\; \exists x_i(X_i)F_1$:

Let

$$\begin{array}{lcl} F' & = & \exists x_{i,1}, \ldots, \exists x_{i,m} F_1' \\ & & \quad \text{where } X_i \;=\; A_{i,1} \ldots A_{i,m} \end{array}$$

The desired equivalence is

$$\begin{array}{l} (\exists v \in \mathcal{U})((\mathit{Attr}(v) = X_i) \;\wedge\; (\; [\![F_1]\!](s[x_i(X_i) \leftarrow v], \boldsymbol{r}\;)\;) \\ \qquad\qquad \text{iff} \quad (\exists v_{i,1}, \ldots, v_{i.m} \in D)\; (\; [\![F_1']\!](s'[y_{i,j} \leftarrow v_{i.j}], \boldsymbol{r})\;) \end{array}$$

Since

$$FV(F_1) \;\;\subseteq FV(F) \;\cup\; \{x_i(A_{i,1}, \ldots, A_{i.m})\}$$

it follows that all substitution pairs (s, s') that are equivalent for F and F' generate substitution pairs that are equivalent for F_1 and F_1', as required.
□

A similar construction works for the other direction:

Lemma: for every DRC formula F there is an equivalent TRC formula F'.

This proof is quite similar to the previous one, and is left as an exercise.
□
Together, these two lemmas yield the promised equivalence theorem:

Theorem: TRC and DRC are equivalent in expressive power.
□