CS632 Notes on Relational Query Languages I

A. Demers

6 Feb 2003

1 Introduction

Here we define relations, and introduce our notational conventions, which are taken almost directly from [AD93].

We begin with a data domain D, an (infinite) set from which all data values will be taken. Thus, while it is conventional to use distinct types for distinct fields of a data record (e.g. the SSNO field is an integer, while the NAME field is a character string), we assume the single domain D is rich enough to represent any values we might be interested in. None of the technical results below is afffected by this decision, and it simplifies some of the arguments.

Assume a countable set $\mathcal{A}$ of *attributes* which we denote by upper case Roman letters near the beginning of the alphabet $(A, B, C, \ldots)$. Upper case Roman letters near the end of the alphabet $(Z, Y, X, \ldots)$ denote finite *sets* of attributes. We use concatenation of these symbols for union of the corresponding sets of names. Thus ABC denotes the set $\{A, B, C\}$, while XA is the set $(X \cup \{A\})$, which may be equal to X (if X contains A).

A relational *tuple* is a function $t : X \to D$, where $X = A_1 \ldots A_k$ is a finite set of attributes which we sometimes call $Attr(t)$. We refer to k (that is, $|Attr(t)|$) as the *arity* of t. We write a tuple by enumeration (it is after all a finite function), either

$$(A_1 \mapsto v_1, \ldots, A_k \mapsto v_k)$$

or, more often,

$$\langle\ A_1 : v_1, \ldots, A_k : v_k\ \rangle$$

by analogy with the usual notation for ordered k-tuples.

A *relation* is a set r of tuples, all over the same set of attributes $\mathit{Attr}(r)$. In the sequel we deal almost exclusively with finite relations, so you should assume a relation is finite unless explicitly told otherwise.

A *relation scheme* is a *relation name* R together with a set X of attributes, typically written $R(X)$. We write $X = \mathit{Attr}(R)$. A relation scheme is an intensional notion: it is not itself a relation; rather, it is a description of the relation values that (a named table in) a database can contain. This is analogous to a variable declaration in a computer program, which is not itself a value, but is a description of the values that can be assigned to the variable.

A *database scheme* is a finite sequence

$$\boldsymbol{R} \quad = \quad (R_1(X_1), \ldots, R_n(X_n))$$

of relation schemes with distinct names.

An *instance* of relation scheme R is a relation r such that $\mathit{Attr}(r) = \mathit{Attr}(R)$.

An *instance* of database scheme $\boldsymbol{R} \quad = \quad (R_1(X_1), \ldots, R_n(X_n))$ is a finite sequence

$$(r) \quad = \quad (r_1, \ldots, r_n)$$

of relations where, for each i, r_i is an instance of R_i.

When we use a name like "R_i" or "R'" for a relation scheme, we will use the corresponding lower case name ("r_i" or "r'") for the relation that instantiates it, often without explicitly mentioning this.

Finally, we use $\mathcal{U}$ to represent the set of all (finite) relations with attributes over $\mathcal{A}$ and data values over D; and we use $\mathcal{U}^+$ to represent the set of all finite sequences of relations in $\mathcal{U}$. Note that every relation r is an element of $\mathcal{U}$, and every database instance $\boldsymbol{r}$ is an element of $\mathcal{U}^+$.

Given a database instance $\boldsymbol{r}$, we define its *active domain* $D_{\boldsymbol{r}}$. to be the set of all values in D that actually occur in (some tuple in some relation in) $\boldsymbol{r}$. In general D is an infinite set; for finite instances $D_{\boldsymbol{r}}$ is finite. This is useful for some of the results below.

A *query* is a function $Q : \mathcal{U}^+ \to \mathcal{U}$, mapping each instance of a database scheme R to a relation. Below we present several query languages, and prove theorems relating their expressive power.

We present a language by giving its syntax, essentially in BNF. We also give its *semantics*, that is a function $[\![\bullet]\!]$ that maps a query expression and a database instance to the result of evaluating the query against the instance.

2 Relational Algebra (RA)

The first query language we consider is relational algebra (RA). RA is a query language based on relation-valued expressions. Thus, in some sense each RA expression is a "recipe" for computing a result relation from a database instance. Below we define (by simultaneous induction) the set of legal RA expressions E, the "type" $\mathit{Attr}(E)$which gives the set of attributes over which the query result is defined, and the semantics – the result of evaluating an RA expression E over a database instance $\boldsymbol{r}$.

Union

$$
\begin{array}{lcl}
E & ::= & E_1 \cup E_2 \\
 & & \text{where } \mathit{Attr}(E_1) = \mathit{Attr}(E_2) \\
\mathit{Attr}(E) & = & \mathit{Attr}(E_1) \\
[\![E]\!](\boldsymbol{r}) & = & [\![E_1]\!](\boldsymbol{r}) \cup [\![E_2]\!](\boldsymbol{r})
\end{array}
$$

Difference

$$
\begin{array}{lcl}
E & ::= & E_1 - E_2 \\
 & & \text{where } \mathit{Attr}(E_1) = \mathit{Attr}(E_2) \\
\mathit{Attr}(E) & = & \mathit{Attr}(E_1) \\
[\![E]\!](\boldsymbol{r}) & = & [\![E_1]\!](\boldsymbol{r}) - [\![E_2]\!](\boldsymbol{r})
\end{array}
$$

Cartesian Product

$$
\begin{array}{lcl}
E & ::= & E_1 \times E_2 \\
 & & \text{where } \mathit{Attr}(E_1) \cap \mathit{Attr}(E_2) = \emptyset \\
\mathit{Attr}(E) & = & \mathit{Attr}(E_1) \cup \mathit{Attr}(E_2) \\
[\![E]\!](\boldsymbol{r}) & = & \{\ t : (\mathit{Attr}(E_1) \cup \mathit{Attr}(E_2) \to D) \mid \\
 & & (t|_{\mathit{Attr}(E_1)} \in [\![E_1]\!](\boldsymbol{r})) \wedge (t|_{\mathit{Attr}(E_2)} \in [\![E_2]\!](\boldsymbol{r}))\ \}
\end{array}
$$

Renaming

$$
\begin{array}{lcl}
E & ::= & \rho_{A \mapsto B}(E_1) \\
 & & \text{where } (A \in \mathit{Attr}(E_1)) \ \wedge (B \notin \mathit{Attr}(E_1)) \\
\mathit{Attr}(E) & = & \mathit{Attr}(E_1) - \{A\} \cup \{B\} \\
[\![E]\!](\boldsymbol{r}) & = & \{\ t : (\mathit{Attr}(E) \to D) \mid \\
 & & (\exists s \in [\![E_1]\!](\boldsymbol{r}))(\ (t|_{\mathit{Attr}(E)-\{B\}} = s|_{\mathit{Attr}(E)-\{B\}}) \\
 & & \quad \wedge\ (t[B] = s[A])\ \}
\end{array}
$$

Projection

$$
\begin{array}{lcl}
E & ::= & \pi_X(E_1) \\
 & & \text{where } X \subseteq \mathit{Attr}(E_1) \\
\mathit{Attr}(E) & = & X \\
[\![E]\!](\boldsymbol{r}) & = & \{t|_X \mid t \in [\![E_1]\!](\boldsymbol{r})\})
\end{array}
$$

Selection

$$
\begin{array}{lcl}
E & ::= & \sigma_P(E_1) \\
 & & \text{where } \mathit{FV}(P) \subseteq \mathit{Attr}(E_1) \\
\mathit{Attr}(E) & = & \mathit{Attr}(E_1) \\
[\![E]\!](\boldsymbol{r}) & = & \{t \in [\![E_1]\!](\boldsymbol{r}) \mid [\![P]\!](t)\})
\end{array}
$$

This last clause requires the definition of a selection predicate P and its free variables $\mathit{FV}(P)$, as well as the semantics of P applied to a tuple. Informally, selection predicates are just propositional (quantifier-free) Boolean formulas defined over the attributes of a tuple, and their "free variables" are the attributes they mention. Formally:

$$
\begin{array}{lcl}
P & ::= & A = c \\
\mathit{FV}(P) & = & \{A\} \\
[\![P]\!](t) & = & (t[A] \ = \ v)
\end{array}
$$

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

$$
\begin{array}{lcl}
P & ::= & A = B \\
\mathit{FV}(P) & = & \{A, B\}
\end{array}
$$

$$[\![P]\!](t) \;=\; (t[A] \;=\; t[B])$$

$$\begin{aligned} P \;&::= \neg P_1 \\ FV(P) \;&=\; FV(P_1) \\ [\![P]\!](t) \;&= \neg [\![P_1]\!](t) \end{aligned}$$

$$\begin{aligned} P \;&::= P_1 \wedge P_2 \\ FV(P) \;&=\; FV(P_1) \;\cup\; FV(P_2) \\ [\![P]\!](t) \;&= ([\![P_1]\!](t)) \;\wedge\; ([\![P_2]\!](t)) \end{aligned}$$

Other propositional connectives (e.g. $\vee$, $\Rightarrow$) can be defined as usual in terms of the ones given above.

3 Characterization Theorem

In this section we give a theorem that characterizes the relations that can be produced as the result of RA queries.

Intuitively, we show that the result of evaluating a RA query expression on a database instance contains only information already present in the instance. More formally, we show that a relation s can be the result of evaluating some RA query on a database instance $\boldsymbol{r}$ if and only if s is invariant under all *automorphisms* of $\boldsymbol{r}$. The automorphisms of $\boldsymbol{r}$ are a class of functions claimed to characterize the distinctions among data values that can be justified using information in $\boldsymbol{r}$. This will be explained in more detail below.

First, you should note the strange form of this characterization. We fix on a database instance $\boldsymbol{r}$, then look at all the (infinitely many) RA query expressions and ask what the possible results are. This says nothing about the expressiveness of RA in terms of the *functions* it is able to compute. As a preview of something we'll return to later, consider the transitive closure r^+ of a binary relation r. Both the following are true:

$$(\forall r)\ (\exists E)\ [\![E]\!](r) \;=\; r^+$$

and

$$\neg(\exists E)\ (\forall r)\ [\![E]\!](r) \;=\; r^+$$

That is, for every relation r there is a RA query expression E_r that produces its transitive closure. But E_r depends critically on r. There is no single RA query that computes the transitive closure of every relation.

Before attacking the Characterization Theorem, we give an important technical result: informally, that the result of an RA query applied to a database instance cannot contain any scalar values that are not already present in the database instance. Formally, we have

Let E be an RA expression. Then

$$D_{[\![E]\!](\boldsymbol{r})} \quad \subseteq \quad D_{\boldsymbol{r}}$$

for any database instance $\boldsymbol{r}$.

The proof is a straightforward induction on the structure of E, and is left as an exercise. Note this result holds even if some selection predicates in E mention constants whose values are not in $D_{\boldsymbol{r}}$.
□

By this theorem, the result of a RA query cannot contain any values that are not already present in the database instance to which the query is applied. In fact our characterization result is somewhat stronger than this. It says that a RA query result cannot make distinctions (between values) that are not already present, either in the database instance or in the query itself. We now proceed to formalize the notion of "distinctions between values."

Consider a function $h : D \to D$. We can extend h in a straightforward way to tuples, relations and relation instances:

$$\begin{aligned}
h(t) &= t' : A \mapsto h(t[a]) \qquad \text{for any tuple } t \\
h(r) &= \{\ h(t) \mid t \in r\} \qquad \text{for any relation } r \\
h((r)) &= (\ R_1 : h(r_1), \ldots, R_n : h(r_n)\ ) \\
&\qquad \text{where } (r) = (\ R_1 : r_1, \ldots, R_n : r_n\ )
\end{aligned}$$

Now, we say h is an *automorphism* of $\boldsymbol{r}$ if $h((r)) = \boldsymbol{r}$. That is, r is *invariant under* h. We denote by $\mathcal{H}(\boldsymbol{r})$ the set of all automorphisms of $\boldsymbol{r}$.

Since $D_{\boldsymbol{r}}$ comprises *exactly* those values that occur in $\boldsymbol{r}$, it follows that any automorphism of $\boldsymbol{r}$ must be a permutation (bijection) of $D_{\boldsymbol{r}}$.

Intuitively, if there is an automorphism of $\boldsymbol{r}$ that maps $u \mapsto u'$ then u and u' are in some sense indistinguishable in $\boldsymbol{r}$. You should be careful not to rely too heavily on this intuition, however. It may be that the every automorphism mapping $u \mapsto u'$ must also map $v \mapsto v'$. For example suppose $\boldsymbol{r}$ contains a relation consisting of the two tuples $(A : u, B : v)$ and $(A : u', B : v')$. The relation contains $(A : u, B : v)$ but not $(A : u', B : v)$, which is somewhat at odds with the intuition that u and u' are indistinguishable. Anyway ...

There is one final technical consideration before we can state and prove the characterization theorem. In the example above, we gave a relation that was invariant under the function h that exchanges u with u' and v with v'. But suppose we apply the selection $\sigma_{A=u}$ to that relation. The result consists of a single tuple $(A : u, B : v)$, and is obviously *not* invariant under h. After a little thought we should not find this surprising: a query expression that explicitly mentions u in a selection predicate is *of course* able to distinguish u from other values. Thus, we need to enhance our definition of automorphism to deal explicitly with constants that might be mentioned in selection predicates.

Let C be a collection of constants in $D_{\boldsymbol{r}}$. We say RA expression E *mentions (at most)* C if every constant occurring (in a selection predicate) in E is an element of C.

> I am actually being a bit sloppy here. As in the semantics above, I ought to talk about a set of *constants*, which are symbols that appear in query expressions, and the set of *values* in D that are the denotations of those constants. But that would be just *too* pedantic, so I assume that any value in D can be used as a constant in a query selection predicate. It is also worth pointing out that selection predicates are the *only* place in an RA expression where constants can be used.

An automorphism h of $\boldsymbol{r}$ is C-*fixed* if $h(v) = v$ for every $v \in C$. We let $\mathcal{H}(C, \boldsymbol{r})$ denote the set of all C-*fixed* automorphisms of $\boldsymbol{r}$.

We are now in a position to state the Characterization Theorem precisely.

Theorem: Let $\boldsymbol{r}$ be any database instance. A relation s, where $D_s \subseteq D_{\boldsymbol{r}}$, can be obtained as a result of a RA expression E mentioning C if and only if s is invariant under every C-fixed automorphism of $\boldsymbol{r}$.

Before giving the proof, we consider a few limiting cases.

Suppose $C = \emptyset$. Then the theorem says s must be invariant under every automorphism of $\boldsymbol{r}$. If we think of automorphisms as describing the ability to make distinctions among values, this says s can make no distinctions not made by $\boldsymbol{r}$.

Suppose $C = D_{\boldsymbol{r}}$. Then every C-fixed automorphism is the identity on all of $D_{\boldsymbol{r}}$, so the invariance requirement of the theorem becomes trivial, and *any* relation over $D_{\boldsymbol{r}}$ is a possible query result.

Note that any values in $C - D_{\boldsymbol{r}}$ are unimportant. You should convince yourself that, since s contains only values in $D_{\boldsymbol{r}}$, invariance of s with respect to C-fixed automorphisms is the same as invariance with respect to$(C \cup D_{\boldsymbol{r}})$-fixed automorphisms.

Proof ($\Rightarrow$): Given E, which mentions C, we need to show

$$h(\ [\![E]\!](\boldsymbol{r})\ ) \quad = \quad [\![E]\!](\boldsymbol{r}) \qquad (\forall h \in \mathcal{H}(C, \boldsymbol{r}))$$

This is done by induction on the structure of E. We present the interesting cases:

Case $E = R_i$: Here E is the name of one of the relations in the database; the desired result

$$h([\![E]\!](\boldsymbol{r})) \quad = \quad h(\ r_i\ )$$

is immediate since h is an automorphism of $\boldsymbol{r}$, which must leave each $r_i \in \boldsymbol{r}$ unchanged. This case is in some sense the "basis" of the induction.

Case $E = E_1 \cup E_2$: This case is proved by symbol-pushing, using the fact that application of h distributes over set union:

$$\begin{aligned}
&h([\![E_1 \cup E_2]\!](\boldsymbol{r})) \\
&\quad = \ h(\ [\![E_1]\!](\boldsymbol{r}) \ \cup \ [\![E_2]\!](\boldsymbol{r})\ ) \\
&\quad = \ h(\ [\![E_1]\!](\boldsymbol{r})\ ) \ \cup \ h(\ [\![E_2]\!](\boldsymbol{r})\ ) \\
&\quad = \ [\![E_1]\!](\boldsymbol{r}) \ \cup \ [\![E_2]\!](\boldsymbol{r}) \\
&\qquad\qquad \text{(by induction hypothesis used twice)} \\
&\quad = \ [\![E_1 \ \cup \ E_2]\!](\boldsymbol{r})
\end{aligned}$$

as desired.

Case $E = E_1 - E_2,\ E = E_1 \times E_2$: These cases are similar to the preceding one.

Case $E = \rho_F(E_1),\ E = \pi_X(E_1)$: These cases are straightforward: renaming and projection do not interact with application of the automorphism.

Case $E = \sigma_{A=B}(E_1)$: This case is a slightly subtle computation:

$$\begin{aligned}
&h([\![\sigma_{A=B}(E_1)]\!](\boldsymbol{r})) \\
&\quad = \ h(\ \{\ t \mid (t \in [\![E_1]\!](\boldsymbol{r})) \ \wedge \ (t[A] = t[B])\ \}\ ) \\
&\quad = \ \{\ h(t) \mid (t \in [\![E_1]\!](\boldsymbol{r})) \ \wedge \ (t[A] = t[B])\ \}
\end{aligned}$$

$$= \{\ h(t) \mid (t \in [\![E_1]\!](\boldsymbol{r}))\ \wedge\ (h(t)[A] = h(t)[B])\ \}$$

(This step follows because h is a bijection on $D_{\boldsymbol{r}}$, so $x = y$ if and only if $h(x) = h(y)$).

$$= \{\ t' \mid (t' \in h([\![E_1]\!](\boldsymbol{r})))\ \wedge\ (t'[A] = t'[B])\ \}$$

(This step is a renaming of $h(t)$ to t'; think carefully about it).

$$= \{\ t' \mid (t' \in [\![E_1]\!](\boldsymbol{r}))\ \wedge\ (t'[A] = t'[B])\ \}$$

(This step uses the induction hypothesis, at last).

$$= [\![\sigma_{A=B}(E_1)]\!](\boldsymbol{r})$$

as required.

Case $E = \sigma_{A=c}(E_1)$**:** We proceed as in the previous case:

$$\begin{aligned} & h([\![\sigma_{A=c}(E_1)]\!](\boldsymbol{r})) \\ & \quad = h(\ \{\ t \mid (t \in [\![E_1]\!](\boldsymbol{r}))\ \wedge\ (t[A] = c)\ \}\ ) \\ & \quad = \{\ h(t) \mid (t \in [\![E_1]\!](\boldsymbol{r}))\ \wedge\ (t[A] = c)\ \} \\ & \quad = \{\ h(t) \mid (t \in [\![E_1]\!](\boldsymbol{r}))\ \wedge\ (h(t)[A] = c)\ \} \end{aligned}$$

(In the previous case this step followed because h is a bijection on $D_{\boldsymbol{r}}$, and thus treats the two sides of the equation identically. In this case, it follows because h is a C-fixed automorphism and $c \in C$. Since $h(c) = c$ it follows that $h(x) = c$ if and only if $x = c$).

$$= \{\ t' \mid (t' \in h([\![E_1]\!](\boldsymbol{r})))\ \wedge\ (t'[A] = c)\ \}$$

(This step is the same renaming of $h(t)$ to t'used in the previous case).

$$\begin{aligned} & = \{\ t' \mid (t' \in [\![E_1]\!](\boldsymbol{r}))\ \wedge\ (t'[A] = c\ \} \\ & = [\![\sigma_{A=B}(E_1)]\!](\boldsymbol{r}) \end{aligned}$$

as desired.

Note the last case is the only one in which we use the fact that h is C-fixed.
□

Proof ($\Leftarrow$): The idea of the proof in this direction is fairly simple; the details are not. So we sketch the idea, and leave filling in the details to the motivated reader.

Suppose we are given a relation $s(A_1 \ldots A_k)$ over $D_{\boldsymbol{r}}$ that is invariant under every C-fixed automorphism $h \in \mathcal{H}(C, \boldsymbol{r})$. We need to construct a RA expression E_s whose result, when applied to $\boldsymbol{r}$, is s.

Define $\mathscr{ar}_C$, the *C-fixed automorphism relation for* $\boldsymbol{r}$, as follows. The columns are labeled by attributed names

$$B_1, B_2, \ldots, B_m \qquad \text{where } D_{\boldsymbol{r}} = \{v_1, \ldots, v_m\}$$

That is, there is an attribute for each distinct value in $D_{\boldsymbol{r}}$. There is a tuple (row) for each $h \in \mathcal{H}(C, \boldsymbol{r})$. For the tuple t corresponding to automorphism h, we have

$$(\forall 1 \leq i \leq m) \;\; t[B_i] \;=\; h(v_i)$$

That is, tuple t is just an enumeration of th applied to each value in $D_{\boldsymbol{r}}$. Assume for now we have a RA expresssion $E_{\mathscr{ar}_C}$ whose result is $\mathscr{ar}_C$. This assumption will be discharged below.

Recall the attributes of s are $A_1 \ldots A_k$, so the arity of s is k. We extend $\mathscr{ar}_C$ to $\mathscr{ar}_{C,k}$, which is similar except for having k distinct copies of each value. Formally, the columns are labeled by attributed names

$$B_1^{(1)}, \ldots, B_1^{(k)}, B_2^{(1)}, \ldots, B_2^{(k)}, \ldots, B_m^{(1)}, \ldots, B_m^{(k)}$$
$$\text{where } D_{\boldsymbol{r}} = \{v_1, \ldots, v_m\}$$

and there is a tuple for each C-fixed automorphism of $\boldsymbol{r}$, containing k copies of each value. For the tuple t corresponding to automorphism h, we have

$$(\forall 1 \leq i \leq m)(\forall 1 \leq j \leq k) \;\; t[B_i^{(j)}] \;=\; h(v_i)$$

As above, tuple t is just an enumerated representation of automorphism function h, but with k copies of each value.

Given $E_{\mathscr{ar}_C}$, we can easily construct a RA expression $E_{\mathscr{ar}_{C,k}}$ for $\mathscr{ar}_{C,k}$. This involves $\Theta(mk)$ renamings, a k-fold Cartesian product, and $\Theta(mk)$ selections to enforce the property

$$(\forall 1 \leq i \leq m)(\forall 1 \leq j < k) \;\; t[B_i^{(j)}] \;=\; t[B_i^{(j+1)}]$$

That is, the k "copies" of $h(v_i)$ in each tuple are all equal.

With $E_{\mathscr{ar}_{C,k}}$ at hand, we can now construct E_s, which generates s as desired. Consider any tuple

$$t \;=\; (\; A_1 : v_{i_1}, \ldots, A_k : v_{i_k} \;)$$

in s. Construct an RA expression E_t that projects $\mathscr{ar}_{C,k}$ onto the attributes

$$B_{i_1}^{(1)},\; B_{i_2}^{(2)},\; \ldots,\; B_{i_k}^{(k)}$$

and then renames according to

$$(\forall i, j)\; B_i^j \;\mapsto\; A_i$$

Since the rows of $\mathscr{ar}_{C,k}$ corrsepond to the C-fixed automorphisms of $\boldsymbol{r}$, it follows that

$$[\![E_t]\!](\boldsymbol{r}) \;=\; \{\; h(t) \mid h \in \mathcal{H}(C, \boldsymbol{r}) \;\}$$

Consider an enumeration of all the (finitely many) tuples in s:

$$s \;=\; \{\; t_1,\; t_2,\; \ldots,\; t_p \;\}$$

Construct the RA expression

$$E_s \;=\; E_{t_1} \;\cup\; E_{t_2} \;\cup\; \ldots \;\cup\; E_{t_p}$$

Now, for every $h \;\in\; \mathcal{H}(C, \boldsymbol{r})$

$$\begin{aligned}
&[\![E_s]\!](\boldsymbol{r}) \\
&\qquad = \;\; ([\![E_{t_1}]\!](\boldsymbol{r})) \;\cup\; \ldots \;\cup\; ([\![E_{t_p}]\!](\boldsymbol{r})) \\
&\qquad = \;\; h([\![E_{t_1}]\!](\boldsymbol{r})) \;\cup\; \ldots \;\cup\; h([\![E_{t_p}]\!](\boldsymbol{r})) \\
&\qquad = \;\; h(([\![E_{t_1}]\!](\boldsymbol{r})) \;\cup\; \ldots \;\cup\; ([\![E_{t_p}]\!](\boldsymbol{r}))) \\
&\qquad = \;\; h([\![E_s]\!](\boldsymbol{r}))
\end{aligned}$$

which is the desired result.

We now show how to construct $E_{\mathscr{ar}_C}$ as promised. For simplicity, assume there is only a single relation scheme in $\boldsymbol{R}$, named $R_1(A_1 \ldots A_k)$; corresponding instance $\boldsymbol{r}$ contains a single relation r_1 (defined over attributes $A_1 \ldots A_k$) containing p distinct tuples. The generalization to more relations is tedious but not conceptually difficult. We proceed as follows.

We construct relation $s(\ldots A_{i,j} \ldots)$ where $1 \leq i \leq k$ and $1 \leq j \leq p$, by taking the p-fold Cartesian product of r_1 with itself and renaming suitably. Intuitively i ranges over the attributes of r_1 and j ranges over tuples of r_1, and many (but not all) of the tuples in s contain all of the tuples of r_1 in some permutation.

Fix on one tuple t in s that contains all of r_1 in this way, and call it the "identity permutation."

Now select only those rows of s that "look like" t up to permuting the value space. That is, construct a (huge) select predicate of conjunctions of the form

$$\begin{array}{ll} (A_{i,j} = A_{k,l}) & \text{if } t[A_{i.j}] \;=\; t[A_{k,l}] \\ \neg(A_{i,j} = A_{k,l}) & \text{if } t[A_{i.j}] \;\neq\; t[A_{k,l}] \end{array}$$

Now choose a set of attributes $C_1, \ldots, C_n$ to select all the values of $D_{\boldsymbol{r}}$ from t; that is,

$$D_{\boldsymbol{r}} \;=\; \{\; t[C_1],\; t[C_2],\; \ldots,\; t[C_n] \;\}$$

and project onto this set of attributes. Interpret the resulting relation as representing functions over $D_{boldsymbolr}$, where a tuple t' represents a function

$$h \;:\; t[C_i] \;\mapsto\; t'[C_i] \qquad (\forall 1 \leq i \leq n)$$

The tuple t represents the identity function. You should convince yourself that this relation contains exactly the automorphisms of $\boldsymbol{r}$. To restrict to C-fixed automorphisms, we select those tuples t' such that $t'[C_i] \;=\; t[C_i]$ for all i such that $t[C_i] \;\in\; C$.

This is the desired construction of $E_{\boldsymbol{r}C}$, and completes the proof of the Characterization Theorem.

□

As we mentioned above, the Characterization Theorem tells us what relations can be the result of RA queries applied to a fixed database instance $\boldsymbol{r}$. It says little about what functions are representable by RA expressions. Here are a few properties of the functions representable by RA expressions E. The proofs are straightforward exercises.

Strict:

$$[\![E]\!](\emptyset) \;=\; \emptyset$$

The result of a query applied to an empty relation instance is empty.

Monotonic:

$$\boldsymbol{r}' \supseteq \boldsymbol{r} \quad \Rightarrow \quad [\![E]\!](\boldsymbol{r}') \ \supseteq \ [\![E]\!](\boldsymbol{r})$$

provided E does not use the '-' operator (set difference).

Active Domain:

$$D_{[\![E]\!](\boldsymbol{r})} \ \subseteq \ D_{\boldsymbol{r}}$$

Finite:

$[\![E]\!](\boldsymbol{r})$ is finite

Domain Independence:

$$(\forall D_1, D_2 \ \supseteq \ D_{\boldsymbol{r}})[\![E]\!]_{D_1}(\boldsymbol{r}) \ = \ [\![E]\!]_{D_2}(\boldsymbol{r})$$

This last one requires a bit of explanation. In our semantics for RA expressions, there is essentially no reference to the data domain D. Domain Independence just states this fact explicitly – the result of evaluating a RA query expression E on an instance $\boldsymbol{r}$ is the same in *any* data domain (D_1 or D_2 above) as long as the domain is "big enough" to contain $\boldsymbol{r}$. This obvious (and important) property of RA expressions fails for the relational calculi, which we discuss next.