7 RC Simulates RA.

We now show that DRC (and hence TRC) is at least as expressive as RA. That is, given an RA expression E that mentions at most C, there is an equivalent DRC expression E' that mentions at most C.

Just as in the previous section, we must deal with the technical issue that DRC is defined by recursion on formulas rather than on expressions. We show that for every RA expression E there exists a DRC formula F that simulates E in the sense that, when F is embedded in a simple DRC expression E', then E and E' are equivalent.

To simplify the presentation, we assume the set of variables names usable in DRC expressions includes all the attribute names usable in RA expressions.

Lemma: For every RA expression $E(A_1 \ldots A_k)$ there exists a DRC formula F with $FV(F) = \{A_1, \ldots, A_k\}$ and

$$(\forall \boldsymbol{r}) \;\; [\![E]\!](\boldsymbol{r}) \;=\; [\![\{\langle A_1 : A_1, \ldots, A_k : A_k\rangle \mid F\}]\!](\boldsymbol{r})$$

Proof: We use induction on the structure of E.

Case $E = E_1 \cup E_2$: Inductively there exist E_1 and E_2 such that

$$[\![E_1]\!](\boldsymbol{r}) \;=\; [\![\{\langle \ldots A_i : A_i \ldots\rangle \mid F_1\}]\!](\boldsymbol{r})$$
$$[\![E_2]\!](\boldsymbol{r}) \;=\; [\![\{\langle \ldots A_i : A_i \ldots\rangle \mid F_2\}]\!](\boldsymbol{r})$$

Note the attributes on which E_1 and E_2 are defined are the same. Let F be $F_1 \vee F_2$; then

$$[\![E_1 \cup E_2]\!](\boldsymbol{r}) \;=\; [\![\{\langle \ldots A_i : A_i \ldots\rangle \mid F_1 \vee F_2\}]\!](\boldsymbol{r})$$

as desired.

Case $E = E_1 - E_2$: This case is similar to $\cup$, with $F = F_1 \wedge \neg F_2$.

Case $E = E_1(A_1 \ldots A_j) \times E_2(A_{j+1} \ldots A_k)$: Note that the sets $\{A_1 \ldots A_j\}$ and $\{A_{j+1} \ldots A_k\}$ are disjoint (the $\times$ operator is Cartesian product, not join). As usual the formulas F_1 and F_2 exist by induction. Let $F = F_1 \wedge F_2$; then

$$[\![E_1 \times E_2]\!](\boldsymbol{r}) \;=\; [\![\{\langle A_1 : A_1 \ldots A_k : A_k\rangle \mid F_1 \wedge F_2\}]\!](\boldsymbol{r})$$

as required.

Case $E \;=\; \rho_{A_1 \mapsto B} E_1$**:** We may assume without loss of generality that the renamed attribute appears first. Also, note that B does not occur among A_2, $\ldots$, A_k. Inductively there exists F_1 such that

$$[\![E]\!](\boldsymbol{r}) \;=\; [\![\{\langle A_1 : A_1 \ldots A_k : A_k \rangle \mid F_1\}]\!](\boldsymbol{r})$$

We must construct a formula equivalent to F_1 but with the free variable A_1 replaced by B. This can be accomplished with an existential and an equality conjunct. Let

$$F \;=\; (\exists A_1)(\; (A_1 = B) \;\wedge\; F_1 \;)$$

The free variables of F are $\{B, A_2, \ldots, A_k\}$, and

$$[\![E]\!](\boldsymbol{r}) \;=\; [\![\{\langle B, A_2 : A_2 \ldots A_k : A_k \rangle \mid F\}]\!](\boldsymbol{r})$$

as required.

Case $E \;=\; \pi_{A_1 \ldots A_j} E_1(A_1 \ldots A_k)$**:** This case is similar to renaming. Inductively, the usual formula F_1 exists, and we eliminate free variables A_{j+1} through A_k by existential quantification. Let

$$F \;=\; (\exists A_{j+1} \ldots \exists A_k)(F_1)$$

Then

$$[\![E_1 \times E_2]\!](\boldsymbol{r}) \;=\; [\![\{\langle A_1 : A_1 \ldots A_j : A_j \rangle \mid F\}]\!](\boldsymbol{r})$$

as required.

Case $E \;=\; \sigma_p E_1$**:** Inductively, the usual formula F_1 exists such that

$$[\![E_1]\!](\boldsymbol{r}) \;=\; [\![\{\langle \ldots A_i : A_i \ldots \rangle \mid F_1\}]\!](\boldsymbol{r})$$

The select condition p is a propositional formula with free variables among A_1 through A_k, and so can be interpreted as a DRC formula in the same free variable environment used for F_1. Let

$$F \;=\; (F_1 \;\wedge\; p)$$

Then

$$[\![\sigma_p E_1]\!](\boldsymbol{r}) \;=\; [\![\{\langle A_1 : A_1 \ldots A_j : A_j\rangle \mid (F_1 \;\wedge\; p)\}]\!](\boldsymbol{r})$$

as required.
□

This demonstrates that DRC (or TRC) is at least as expressive as RA. The possibility remains that DRC/TRC is strictly more expressive. We deal with this next.

8 Domain Independence (Safety)

Our ultimate goal is to show that RA and RC are in some sense equally expressive. There is an easy technical issue we must dispose of first.

Let $\boldsymbol{R}$ be the database scheme $\{R_1(A)\}$, a single relation with a single column. Consider the following simple example query in DRC.

$$[\![\{\langle B : x\rangle \mid x = c\}]\!](\boldsymbol{r}) \quad = \quad \{\langle B : c\rangle\}$$

This holds whether or not the value c occurs anywhere in the instance $\boldsymbol{r}$, because the quantified variable x ranges over all values in D. Thus, the theorem

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

which we proved earlier for RA expressions, fails for RC expressions.

Next consider the two examples

$$[\![\{\langle B : x\rangle \mid x = x\}]\!](\boldsymbol{r}) \quad = \quad \{\langle B : v\rangle \mid v \in D\}$$

and

$$[\![\{\langle B : x\rangle \mid \neg(\langle A : x\rangle \in R_1)\}]\!](\boldsymbol{r}) \quad = \quad \{\langle B : v\rangle \mid v \in (D - D_{\boldsymbol{r}})\}$$

The result in the first example is a complete copy of D; in the second case it is the complement of relation r_1 in D. Thus, both finiteness and domain-independence, which we proved earlier for RA expressions, fail for RC expressions. A domain-dependent expression like the examples above is sometimes called *unsafe*.

Relation Constants For the problem illustrated by the first example above there is a simple technical fix. We augment RA expressions with *relation constants* as follows:

$$\begin{aligned} E &::= \{\langle A_1 : c_1, \ldots, A_k : c_k \rangle\} \\ \mathit{Attr}(E) &= A_1 \ldots A_k \\ [\![E]\!](\boldsymbol{r}) &= \{\langle A_1 : c_1, \ldots, A_k : c_k \rangle\} \end{aligned}$$

Since RA includes a union operator, expressions for relations with only a single tuple are sufficient. We then augment the definition of "E mentions at most C" in a straightforward way to include constants appearing in relation-valued expressions as well as those appearing in select conditions. It is easy to see that the revised theorem

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

holds, where E is a RA expression that mentions at most C.

Domain Independence Having added relation constants to the RA language, we shall now argue that the *only* remaining issue is domain dependence, illustrated by the second and third examples above.

We showed above that for every DRC expresison $E^{(D)}$ there is an equivalent TRC expression $E^{(T)}$, and *vice versa*. Since domain-independence is a semantic notion, $E^{(T)}$ is domain-independent if and only if $E^{(D)}$ is. Let DRC-DI be the language consisting of just the domain-independent DRC expressions; and let TRC-DI be the domain-independent TRC expressions. Ignore for the moment the question of how to decide whether a particular expression is domain-independent or not. Clearly DRC-DI and TRC-DI are equivalent. Moreover, both languages are at least as expressive as RA: we showed above that for every RA expression $E^{(A)}$ there is an equivalent TRC expression $E^{(T)}$ and an equivalent DRC expression $E^{(D)}$; since $E^{(A)}$ is domain-independent, $E^{(T)}$ and $E^{(D)}$ must be domain-independent also.

We shall complete the argument to show that DRC-DI (TRC-DI) and RA have exactly the same expressive power, by showing that for every DRC-DI expression there is an equivalent RA expression. Formally, we show:

Theorem: For any DRC formula F on free variables $\{X_1, \ldots, X_k\}$ mentioning at most C, there exists an RA expression E with attributes $\{X_1, \ldots, X_k\}$ and mentioning at most C such that

$$[\![E]\!](\boldsymbol{r}) = [\![\{\langle X_1 : X_1 \ldots X_k : X_k \rangle\} \mid F]\!]_{D_{\boldsymbol{r}} \cup C}(\boldsymbol{r})$$

for all instances $\boldsymbol{r}$. Note that F and E mention *at most* C; the case that C includes some constants not mentioned in E or F is explicitly allowed. In fact, it is *necessary* to make the inductive hypothesis apply to subformulas in the proof below.

Proof: As usual, we assume all DRC variable names can be used as attribute names, and proceed by induction on F.

First we make an observation. Given the set of constants C and a any attribute name A, we can construct an RA expression

$$ED_{C,A} \quad = \quad \{\langle A : c_1 \rangle\} \cup \ldots \cup \{\langle A : c_n \rangle\} \cup (\bigcup_{R \in \boldsymbol{R}} \bigcup_{B \in U} \rho_{B \mapsto A}(\pi_B(R)))$$

Then

$$[\![ED_{C,A}]\!](\boldsymbol{r}) \quad = \quad \{ \ \langle A : v \rangle \mid v \in (D_{\boldsymbol{r}} \cup C) \ \}$$

That is, the value of $ED_{C,A}$ is just the active domain of $\boldsymbol{r}$ together with the constants in C. This is the domain appearing in the statement of the theorem. It is the smallest domain over which the DRC formula F can be evaluated.

Now we show by induction on F how to construct the equivalent RA expression E:

Case $F = (\langle \ldots B_i : X_i \ldots \rangle \in R)$: This "basis" case requires a bit of thought. Relation R is defined over attributes $B_1, \ldots, B_k$, but the free variables of F are actually $X_1, \ldots, X_k$, and the attributes of the required result expression must be the free variables of F. Thus, let

$$E \quad = \quad \rho_{\ldots B_i \mapsto X_i \ldots}(R)$$

and the desired result follows.

Case $F = (X_1 = a)$: Let

$$E \quad = \quad \sigma_{X_1 = a}(E_{C,X_1})$$

and the desired result follows. Of course there are simpler choices for E, but this choice is correct and establishes a pattern that continues in the remaining cases.

Case $F = (X_1 = X_2)$**:** Let

$$E = \sigma_{X_1=X_2}(E_{C,X_1} \times E_{C,X_2})$$

and the desired result follows.

Case $F = \neg F_1$**:** Note the free variables of F and F_1 are the same. Inductively, there exists an RA expression E_1 equivalent to F_1. Let

$$E = (E_{C,X_1} \times \ldots \times E_{C,X_k}) - E_1$$

and the desired result follows.

Case $F = (F_1 \wedge F_2)$**:** This case also requires some thought, because there is no reason to expect the free variables of F_1 and F_2 to be the same. Without loss of generality, choose $1 \leq m \leq n \leq k$ and let the free variables be

$$\begin{aligned} FV(F_1) &= \{ X_1, \ldots, X_m, X_{n+1}, \ldots, X_k \} \\ FV(F_2) &= \{ X_{m+1}, \ldots, X_k \} \end{aligned}$$

Inductively, there exist RA expressions E_1 and E_2 over these sets of attributes that are equivalent to F_1 and F_2 respectively. We can easily extend both expressions to the free variables of F:

$$\begin{aligned} E_1' &= (E_{C,X_{m+1}} \times \ldots \times E_{C,X_n}) \times E_1 \\ E_2' &= (E_{C,X_1} \times \ldots \times E_{C,X_m}) \times E_2 \end{aligned}$$

We can also easily define the "universe" over these free variables:

$$E^* = E_{C,X_1} \times \ldots \times E_{C,X_k}$$

Now the desired expression is simply the intersection of E_1' and E_2', which can be expressed by

$$E = E^* - ((E^* - E_1') \cup (E^* - E_2'))$$

as desired.

Case $F = (\exists Y\ F_1)$**:** Without loss of generality, the free variables of F_1 are $X_1, \ldots, X_k, Y$; and by induction there exists an RA expression E_1 such that

$$[\![E_1]\!](\boldsymbol{r}) = [\![\{\langle X_1 : X_1, \ldots X_k : X_k, Y : Y\rangle \mid F_1\}]\!]_{D_{\boldsymbol{r}} \cup C}(\boldsymbol{r})$$

Now let

$$E = \pi_{X_1,\ldots,X_k} E_1$$

and the result follows.
□

Range Declarations

We can augment DRC expressions with "Range Declarations" to get a language DRC-RD that is equivalent to RA.

The syntax of a *Range Declaration* is just

$$RD ::= (\ c_1, \ldots, c_m, R_{i_1}(A_{i_1}), \ldots, R_{i_n}(A_{i_n})\ )$$

that is, a finite list of constants and relation columns.

A range declaration describes an obvious set of values: the explicitly enumerated constants, plus all the values that occur in the specified columns of the database instance. If constants appearing in a range declaration are considered to be "mentioned," it should be clear that a range declaration describes a subset of $D_{\boldsymbol{r}} \cup C$.

Now, to define DRC-RD, we simply add a range declaration wherever a variable is introduced in an expression.

$$E ::= \{\ \langle A_1 : x_1 : RD_1, \ldots A_n : x_n : RD_n\rangle \mid F\ \}$$
$$F ::= (\exists x : RD)\ F_1$$

Variables are allowed to range only over values defined by the associated range declarations.

By arguments very similar to those used in our discussion of DRC-DI, we can show that every DRC-RD expression can be simulated by an RA expression, and *vice versa*.

The situation is not so nice with TRC. Here the "obvious" way to define TRC-RD would allow each tuple-valued variable to range over a relation in the database schema. This would be inadequate for a number of reasons. For example, the result of anyTRC-RD query would necessarily have the same set of attributes as one of the relations in the schema.

Here is a syntax for TRC-RD that is stronger than simply allowing variables to range over tables, though it still lacks the full power of TRC-DI. This syntax is important because of its relation to the SQL query language used in commercial database systems, as we discuss below. In this syntax, each tuple variable ranges over a single table of the database instance, but there is special mechanism for constructing a result relation with attributes that do not match any of the tables in the database schema.

$$
\begin{aligned}
E \ ::= \ & \{ \ A_1 : x_1[B_1] \ldots A_m : x_m[B_m] \mid \\
& \qquad y_1(Y_1) : R_{i_1} \ldots y_n(Y_n) : R_{i_n} \mid F \ \} \\
F \ ::= \ & (\exists z(Z) : R_i) \ F_1
\end{aligned}
$$

Variable y_j is allowed to range only over tuples contained in the relation named R_{i_j} in $\boldsymbol{r}$; which must have attributes Y_j. Similarly, the variable $z(Z)$ introduced in a quantified formula ranges over the specified relation R_i, which must have attributes Z. The result tuples have attributes $A_1 \ldots A_m$ and are constructed from y_j values for which formula F is satisfied.

With this TRC-RD syntax, tuple-valued variables are constrained to range over individual relations in the instance, but it is possible for a query result to be a relation over an arbitrary set of attributes. However, the language is still strictly weaker than TRC-DI or RA. Consider the RA example:

$$
\begin{aligned}
\boldsymbol{R} \ &= \ (R_1(A), R_2(A)) \\
E \ &= \ (R_1 \ \cup \ R_2)
\end{aligned}
$$

which simply takes the union of two relations. It is clear that a TRC-RD expression has the property that every column of its result relation is entirely contained in some column of some relation of the input database instance. Since union does not have this property, no TRC-RD expression can be equivalent to the above RA expression.

Within formulas, the inability to have a variable range over the union of several tables is not a fundamental limitation. It is easy to see that the equivalence

$$
((\exists z(Z) : (R_i \cup R_j)) \ F) \ \Leftrightarrow \ ((\exists z(Z) : R_i) \ F) \ \vee \ ((\exists z(Z) : R_j) \ F)
$$

is always valid, so a union in a range declaration of a TRC-RD formula could always be transformed away, at the cost of increasing the size of the formula. But there is no way to simulate this transformation "at top level," in constructing the result relation. One can show the following:

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

The proof is left as an exercise.
□

It turns out union is the *only* weakness of the TRC-RD language. One can show that the language of expressions defined by

$$E \ ::= \ E_1 \ \cup \ \ldots \ \cup \ E_k$$
where E_i is a TRC-RD expression

is equivalent to RA. This result is of some "practical" significance, since some early versions of SQL (without aggregation operators or very general subquery mechanisms) were *exactly* equivalent to TRC-RD – they were, in fact, just slight syntactic variants on it. So this result explains why it was necessary to augment SQL with set-theoretic operators at top level: it made the language relationally complete.

9 Undecidability of Domain Independence

We now know that the domain-independent TRC (DRC) expressions are equivalent to RA in expressiveness. Unfortunately, however, there is no algorithm to decide which TRC (DRC) expressions are domain-independent. Here we sketch the proof of this fact for TRC.

We shall actually show that *satisfiability* of TRC formulas is undecidable, that is, given a TRC formula F (even one with range declarations) it is undecidable whether there exists a database instance $\boldsymbol{r}$ such that $[\![E]\!](\boldsymbol{r})$ is true. This is sufficient for our purposes, since, if F is a (closed) TRC formula, the expression

$$E \ = \ \{ \ x(A) \mid F \ \}$$

is domain-independent iff it is the empty relation – that is, iff F is false for every instance $\boldsymbol{r}$. Thus, satisfiability of TRC formulas is reducible to domain-independence of TRC expressions; so if we show satisfiability is undecidable it follows that domain-independence must be undecidable as well.

Here is a sketch of an elementary proof that satisfiability is undecidable. We start from the undecidable problem: given M, a 2-counter machine, does M accept on blank input? A configuration of M can be represented as a triple $\langle q, c_1, c_2 \rangle$ consisting of the state and the two counter values; and we may assume without loss of generality that to accept its input the machine loops in a unique accepting state. We show how to construct a TRC formula F_M that is satisfiable exactly if M accepts the empty input.

If the data domain were the natural numbers the construction would be easy. We would use the schema

$$\boldsymbol{R} \quad = \quad (\ R_1(\text{step, state, cntr1, cntr2})\ )$$

and express each of the following as TRC formulas:

step is a superkey for r_1

$\langle 0, q_0, 0, 0 \rangle \ \in \ r_1$

$(\exists \langle n, q, c_1, c_2 \rangle \ \in \ r_1)$ s.t. q accepts

$(\forall \langle i, q, c_1, c_2 \rangle \ \in \ r_1)\ (\exists \langle i', q', c_1', c_2' \rangle \ \in \ r_1)$
where $(q, c_1, c_2) \ \vdash_M \ (q', c_1', c_2')$

For the last of these formulas, we can assume without loss of generality that

$$c_1' = c \text{ or } c_1' = (c_1 + 1) \text{ or } (c_1' + 1) = c_1$$

and similarly for c_2; and we can encode the next-state function of M in a query whose size depends only on M. Also, we use the standard trick of encoding

$$(\forall x)F \quad \mapsto \quad \neg((\exists x)(\neg F))$$

to eliminate universal quantifiers.

It should be clear that an input instance satisfies the conjunction of these formulas if and only if it represents an accepting computation of M. So, if the data domain were the natural numbers, we would be done.

We can avoid reliance on the natural numbers by adding a second relation to the schema, and instantiating it with a relation value that is "close enough" to the successor function for our needs. Let

$$\boldsymbol{R} \quad = \quad (\ R_1(\text{step, state, cntr1, cntr2}),\ R_2(\text{num, suc})\ )$$

We express as TRC formulas the following:

num is a superkey for r_2

suc is a superkey for r_2

We now introduce a new constant k_0 and change the above formulas constraining r_1 as follows:

$0 \mapsto k_0$

$((\exists i)F) \mapsto ((\exists t)(t \in r_2 \wedge F')$ where F' is F with $t[\text{num}]$ replacing i

$y = (x+1) \mapsto ((\exists t)(t \in r_2 \wedge t[\text{num}] = x \wedge t[\text{suc}] = y)$

We claim the new formula is satisfiable if and only if M halts. One direction is obvious: if M halts within n steps, we can simply encode the successor relation on the natural numbers up through n in r_2 (using an arbitrary set of $n+1$ distinct data values) and then encode an accepting computation of M in r_1 consistently.

For the other direction, observe that the constraints on r_2 guarantee that its two columns are a permutation. If the formula is satisfied, then confining our attention to the cycle of the permutation that contains k_0 we can identify an accepting computation of M.

This completes the proof sketch, and we are done.
□