A PRÉCISE DEFINITION OF RRA - A RESHAPED RELATIONAL ALGEBRA WHICH FOLLOWS NATURAL LANGUAGE CONSTRUCTS

by

Yoav Raz

Technical Report #405

March 1986
A PRECISE DEFINITION OF RRA - A RESHAPED RELATIONAL ALGEBRA
WHICH FOLLOW NATURAL LANGUAGE CONSTRUCTS

by

Yoav Raz
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel

ABSTRACT

RRA - the Reshaped Relational Algebra is equivalent to the regular Relational Algebra (RA), but has different operators. Its operators, a modification of the RA's ones, are intended to follow the semantics of queries phrased in Natural English Language (NL). For each operator there is an appropriate elementary syntactical construct of natural language. These elementary constructs can be combined to form complex sentences (queries), having semantics given by algebraic expressions composed of respective operators. As such, the RRA is useful both in defining semantics for Natural Language subsets or languages alike, and in implementing efficiently NL translators for interfaces to relational databases. A precise definition of RRA is given and its connection to Natural Language is explained. It is also shown that RRA is equivalent to RA.
1. INTRODUCTION

The RRA (a Reshaped Relational Algebra - [MR2], [MR3]) was originally defined as a tool for a compact definition of the semantics of ERROL (an Entity Relationship Role Oriented Language - see [MR1], [RM]) which is an English-like query language over the Entity Relationship Model (ERM, see [Che]). In this paper RRA is redefined in order to make it useful on a wider range.

RRA - the Reshaped Relational Algebra is equivalent to the regular Relational Algebra (RA), but has different operators. Its operators, which are a modification of the RA's ones, are intended to follow the semantics of queries phrased in Natural English Language (NL). For each operator there is an appropriate elementary syntactical construct of natural languages. These elementary constructs can be combined to form complex sentences (queries), having semantics given by algebraic expressions composed of the respective operators. As such, the RRA is useful both in defining semantics for Natural Language subsets or languages alike and in implementing efficiently interfaces to relational databases.

The analogies to Natural Language carried by the RRA are exposed when we use distinct relations to describe entity (object) sets and relationship (association) sets. An advantage is taken of the fact that relationships are usually described by a "simple" sentence (see [Bi]). A "closed world" ([Re]) is also assumed.

Part 2 gives a formal definition of the RRA and Part 3 discusses the linguistic aspect of the DB information. In Part 4 the analogy to Natural Language is explained and demonstrated. Part 5 shows the equivalence of RA and RRA, and Part 6 gives some conclusions.
2. THE RRA - A RESHAPED RELATIONAL ALGEBRA

The RRA is equivalent to the RA and has similar operators. Some of its operators are identical to those of RA and some of them are extensions of the RA's (see [Pir]). The main difference lies in an implicit join operation embedded in binary operations. This embedded join is due to references ("the above", "that", "the mentioned") which appear in NL (see below, Section 4.9). A different operator is the NOT (complement) operator which expresses NL's negation. This operator enable RRA to express the the RA's subtraction.

Operators which extend the scope of the usual RA are the AGGREGATE FUNCTION operators which can be viewed as both RRA's and RA's.

2.1 NOTATIONS AND DEFINITIONS

A relation consists of a name, structure and value.

Notation: \( S(R) \), \( V(R) \) are the structure and value, respectively, of a relation with the name \( R \).

Let \( A_1, A_2, \ldots, A_n \) be the attribute names of \( R \) and \( D_{A_1}, D_{A_2}, \ldots, D_{A_n} \) their corresponding domains (a domain is any set of objects of the same type).

Let also

\[
A = \{A_1, A_2, \ldots, A_n\}.
\]

Then, \( R \) is denoted also as \( R(A) \):

\[
S(R) = \{z: D_z | z \in A\} = \{A_i: D_{A_i} | i = 1, \ldots, n\};
\]

\[
V(R) = \{t: A \rightarrow \bigcup_{z \in A} D_z\}
\]

\( t \) (a "tuple") is a total function over \( A \), which maps each \( x \in A \) to an element in \( D_x \).

In the following sections the RRA operations are defined. The result relation of any operation will be denoted \( R' \), \( S' \), \( V' \) for name, structure and value respectively.
2.2 RENAMING

The renaming operator (a unary one) replaces attribute names with others. This operation is needed since some operators take into account the fact that operands may have identical attribute names.

Let $R = R(A)$

$B = \{B_1, B_2, \ldots, B_n\}$ a set of attribute names.

$M: A \rightarrow B$ such that $M(A_i) = B_i$, $i=1,2,\ldots,n$.

Then:

\[ R' = \text{rename}_M(R); \]
\[ S' = S(R') = \{M(z): D_z | z \in A\} = \{B_i: D_{A_i} | i=1,\ldots,n\} \]
\[ V = V(R') = \{t': M(A) \rightarrow \bigcup_{x \in A} (\exists t \in V(R))(\forall y \in A)(t'(M(x)) = t(x))\}. \]

2.3 PROJECTION

The (unary) projection operator takes the restrictions of tuples in a relation on a common partial domain of attribute names.

Let $R = R(A)$

$A' \subseteq A$.

Then:

\[ R' = R[A'] \text{ or } R' = \text{project}_{A'}(R) \]
\[ S' = S(R') = \{z: D_z | z \in A'\} \]
\[ V = V(R') = \{t': A' \rightarrow \bigcup_{x \in A'} D_x | (\exists t \in V(R))(\forall y \in A')(t'(x) = t(x))\}. \]
2.4 SELECTION

The (unary) selection operator picks from a relation tuples which map a specific attribute name to a specific value.

Let \( R = R(A \cup \{ a \}) \)
\[ a \notin A \]
\[ c = \text{const.} \]

Then
\[ R' = \text{select}_{a \neq c}(R) \]
where \( \vartheta \in \{ =, \neq, <, >, \leq, \geq \} \)
\[ S' = S(R') = \{ x: D_x | x \in A \} \]
\[ V' = V(R') = \{ t': A \to \bigcup_{x \in A} D_x \mid (\exists t \in V(\hat{R})) \{ t' = t[A] \land t(a) = c \} \} \]
where \( t[A] \) is the restriction of \( t \) on the domain \( A \).

2.5 PRODUCT (NATURAL JOIN, CARTESIAN PRODUCT, INTERSECTION)

The (binary) product operator creates a relation with tuples produced each from one tuple of each operand. A resulting tuple has all the attribute names of the two operands and all the values of the two tuples producing it. This implies that the two producing tuples have identical values for identical attribute names (the embedded natural join).

Let \( R_1 = R_1(A \cup B) \), \( R_2 = R_2(B \cup C) \)
where \( A, B, C \) disjoint.

Then
\[ R' = R_1 \ast R_2 \]
\[ S' = S(R') = S(R_1) \cup S(R_2) = \{ x: D_x | x \in A \cup B \cup C \} \]
\[ V' = V(R') = \{ t': \bigcup_{x \in A \cup B \cup C} D_x \mid \} \]
(\exists t_1 \in V(R_1))(\exists t_2 \in V(R_2))(t'[A \cup B] = t_1 \land t'[B \cup C] = t_2)\}

where \( t'[A'] \) is the restriction of \( t' \) on domain \( A' \subset A \cup B \cup C \).

REMARKS:

1) This operation is equivalent to the regular Relational Algebra Natural joint over \( B \).

2) If \( B = \emptyset \) it reduces to the regular Cartesian Product.

3) If \( A = C = \emptyset \) it reduces to the regular Intersection.

2.6 BORDERED UNION or OR

The (binary) bordered union operator extends the union operation for non union-compatible operands. From each tuple of one operand it produces tuples which agree with it (the source tuple) on the common domain and have all possible combinations of values suggested by the other operand on attribute names which appear in the other operand but not in the operand of the source tuple.

Let \( R_1 = R_1(A \cup B), \ R_2 = R_2(B \cup C) \)

where \( A, B, C \) disjoint.

Then

\[ R' = R_1 \text{ or } R_2 \]

\[ S' = S(R') = S(R_1 \cup R_2) = \{ z : D_z | z \in A \cup B \cup C \} \]

\[ V' = V(R') = \{ t' : A \cup B \cup C \to \bigcup_{z \in A \cup B \cup C} D_z | (\exists t_1 \in V(R_1))(\exists t_2 \in V(R_2)) \}

\[ (t'[A \cup B] = t_1 \land t'[C] = t_2[C]) \lor (t'[B \cup C] = t_2 \land t'[A] = t_1[A]) \}

.REMARK: If \( A = C = \emptyset \) it reduces to the regular Union.
2.7 ATTRIBUTE JOIN or \( \vartheta \)-JOIN

The (binary) attribute-join operator produces tuples each of them consists of one tuple from each operand such that a certain comparison relation \( \vartheta \) exists between values of two specific attribute names, one of the first operand and the second of the other operand. These two attribute names do not appear in the resulting relation. In addition the two tuples should agree on the values of common attribute names (the embedded natural join).

Let \( R_1 = R_1(A \cup B \cup \{a_1\}) \), \( R_2 = R_2(B \cup C \cup \{a_2\}) \)

where \( A, B, C \) disjoint.

\[ a_1, a_2 \not\in A \cup B \cup C \]

\( D_{a_1}, D_{a_2} \) have elements of the same type.

Then

\[ R' = R_1 \text{ join}_{a_1} R_2 \]

\[ \vartheta \in \{=, \neq, <, >, \leq, \geq\} \]

\[ S' = S(R') = \{ z : D_z \mid z \in A \cup B \cup C \} \]

\[ V' = V(R') = \{ t' : A \cup B \cup C \to \bigcup_{z \in A \cup B \cup C} D_z \mid ( \exists t_1 \in V(R_1))( \exists t_2 \in V(R_2)) \}

\[ t'[A \cup B] = t_1[A \cup B] \land t'[B \cup C] = t_2[B \cup C] \land t_1(a_1) \vartheta t_2(a_2) \}

**Remark:** If \( B = \emptyset \) it reduces to the regular \( \delta \)-Join.

2.8 SET-JOIN or GENERALIZED DIVISION

The (binary) set-join operator produces tuples each of them consists of the restrictions ("portions") of two tuples, one for each operand. Two such restrictions of tuples are matched to produce a result tuple if each of them appears in its relation together with a set of restrictions (by the complement set of attribute names) which has a \( \delta \) set-comparison relation with a similar set of restrictions for
the restriction of the other operand's tuple. In addition the two tuples should agree on the values of common attribute names (the-embedded natural join).

Let \( R_1 = R_1(A \cup B \cup D_1) \), \( R_2 = R_2(B \cup C \cup D_2) \)
where \( A, B, C, D_1, D_2 \) are disjoint and there exists a one-one-on mapping \( f: D_1 \rightarrow D_2 \)
such that \( f(x_1) = x_2, x_1 \in D_1, x_2 \in D_2 \) if and only if \( B_{x_1}, D_{x_2} \) have elements of the same type.

Then

\[
R' = R_1 \setminus \text{join } R_2 \text{ where } \delta \in \{ =, \neq, <, >, \leq, \geq \}
\]

\[
S' = S(R') = \{ x: D_2 \mid x \in A \cup B \cup C \}
\]

\[
V' = V(R') = \{ t: A \cup B \cup C \rightarrow \bigcup_{x \in A \cup B \cup C} D_x \}
\]

\[
\left( \frac{t_1 \in V(R_1)}{t_2 \in V(R_2)} \right) \left( t'[A \cup B] = t_1[A \cup B] \land t'[B \cup C] = t_2[B \cup C] \land \right.
\]

\[
\left. \land V((select_{A \cup B=t_1}(R_1))[D_1]) \delta V((select_{B \cup C=t_2}(R_2))[D_2])) \right)
\]

where \( select_{t'=t}(R(A)) \), \( A' \subset A \) stands for repeated
\( select_{t'=t}(z) \) of \( R(A) \) over every \( z \in A' \).

**REMARK:** If \( B = \emptyset \) it reduces to a regular Generalized Division.

### 2.9 COMPLEMENT or NOT

The (unary) complement operator produces all the tuples in the Cartesian Product of a given set of unary relations over attribute names of the operand (considered parameters) which do not appear in the projection of the operand over the attribute names of the above relations.

Let \( R = R(A), R_x = R_x(\{ x \}) \) and \( A' \subset A \).

Then
\[ R' = \text{not}_{[R_a]}(R) \]

\[ S' = S(R') = \{z : D_z \mid z \in A'\} \]

\[ V = V(R') = \{t' : A' \rightarrow \bigcup_{z \in A'} D_z\} \]

\[ (\forall z \in A')(\exists t_z \in V(R))(t'(z) = t_z(x)) \land t' \not\in V(R[A']) = \times_{z \in A'} (V(R_z) - V(R[A'])) \]

where "\( \times \)" stands for the regular Cartesian Product, and "-" stands for the regular Subtraction.

2.10 AGGREGATE FUNCTION (AF)

Every (unary) aggregate function operator has as a parameter an attribute name \((a)\) of the operand. For each restriction, by all attribute names except the operand’s tuple, there is a set of the parameter’s values which appear in tuples with the same restriction. Applying the appropriate AF to this set (provided the parameter’s type matches the AF) we get a value \((v)\) for a new attribute name \((AF.A)\). The above restriction together with the pair \((AF.A.v)\) are a tuple in the resulting relation (remember that a tuple is a function). AF can be SUM which gives the sum of set elements; COUNT – for the number of elements in the set; MAX – for the maximal element in the set, etc.

Let \( R = R(A \cup \{a\}) \), \( a \not\in A \)

\[
AF.a : 2^{D_a} \rightarrow D_{AF.a}
\]

where \( AF \in \{\text{SUM, COUNT, MAX, MIN...}\} \)

Then

\[ R' = AF.a(R) \]

\[ S' = S(R') = \{z : D_z \mid z \in A' = A \cup \{AF.a\}\} \]

\[ V' = V(R') = \{t' : A' \rightarrow \bigcup_{z \in A'} D_z\} = t'\{A'\} \land \]
\[ t'(AF_\alpha) = AF(Y(\text{select}_{\alpha=A}(R))[\alpha])) \]

**REMARK:** When \( AF = \text{SUM} \), the "select" and "project" should be modified; its results are multisets rather than sets.

### 3. THE LINGUISTIC ASPECT OF THE INFORMATION IN THE DATABASE

It is common among DB designers to describe databases on its conceptual level using Entity (Object) - Relationship (association) Diagrams (ERDs, [Che]) which are a kind of semantic networks ([BF]) used for knowledge representation. Informally, an ERD is a graph with three types of nodes:

1. **Entity sets.**
2. **Relationship sets.**
3. **Attributes.**

The above sets consist of elements of the same type (with the same attributes). The arcs connect entity sets with appropriate attributes and relationship sets, and also relationship sets with its attributes. An example of an ERD appears in Fig. 1. The notion of ERD can be extended beyond what demonstrated here to capture more of the real world's semantics. However, even with its simple version the basic ideas can be presented.

A relationship between entities can usually be described by a simple sentence with an entity as the subject part, the other entities participating in the relationship as objects and a predicate part which induces the relationships' name.

For example, a member in the relationship set SUPPLY can be described by the sentence

- **SUPPLIER** \( \text{SUPPLIES} \) **ITEM** to **DEPARTMENT.**
- **DEPARTMENT** is **SUPPLIED** by **SUPPLIER** with **ITEM.**

If the relationship has attributes, they may be combined in the sentence:

- **SUPPLIER** STOCK QUANTITY of **ITEM.**
The association between an entity and its attribute may be expressed as

SUPPLIER HAS NAME... or

NAME OF SUPPLIER...

The RRA has been designed to take advantage of the ERD's linguistic aspect by calculating relations which contain tuples satisfying predicates expressed in natural language sentences based on the sentences derived from the ERD (i.e., the attributes in the result are renamed attributes of those which appear in the NL sentence).

Fig. 1

The rectangular boxes represent entity sets; diamonds - relationship sets; round boxes - attributes.
DEPARTMENT = DEPARTMENT(No, NAME, FLOOR); key = No.
ITEM = ITEM(No; NAME, COLOR, TYPE); key = No.
SUPPLIER = SUPPLIER(No, NAME, LOCALITY); key = No.
REQUEST = REQUEST(DEPARTMENT-key, ITEM-key, QUANTITY)
STOCK = STOCK(ITEM-key, SUPPLIER-key, QUANTITY)
SUPPLY = SUPPLY(ITEM-key, SUPPLIER-key, DEPARTMENT-key, QUANTITY, PRICE)

Fig. 2

The relations representing the ERD in Fig. 1.

4: THE RRA OPERATORS AS MEANING OF NATURAL LANGUAGE CONSTRUCTS

The analogy between Natural Language and RRA is exposed when Entity sets and relationship sets in an ERD are expressed as relations in the most straightforward way:

For each entity set there is a relation which attributes are those of the entities in the set;

For each relationship there is a relation which attributes are those of the relationships in the set, augmented with the key attributes in all the entity sets associated by the relationship (key attributes are such that given its values, an entity is uniquely specified. Since a key in a relationship is used as an identifier only, we shall view all the key attributes of an entity appearing in a relationship as a simple attribute with values consisting of concatenating key attribute values of entities in a certain defined order).

The relational representation of the ERD of Fig. 1 appears in Fig. 2. If the DB relations are not such, it can sometimes (intuitively, if inherently the DB bears the information about the entities and the relationships. See [MMR]) be converted into this form using RRA operations. We shall not go into the conversion problem,
and assume that the DB relations are in the desired form.

In what follows, the Natural Language analogies of the RRA operations are given (For the NL combinations see [Win]).

4.1 RELATIVIZATION AND THE NATURAL JOIN (PRODUCT) OPERATION

Relativization is connecting a sequence of sentences such that any two neighboring sentences are chained on an object part of the first one. One possibility is that the last object part of the first sentence is the subject of the second. For example:

SUPPLIER STOCKS ITEM REQUESTED by DEPARTMENT.

is a combination of SUPPLIER STOCKS ITEM and ITEM IS REQUESTED by DEPARTMENT.

The meaning is given by the RRA expression

STOCK*REQUEST,

where

STOCK=STOCK(SUPPLIER_key, ITEM_key, ...) REQUEST=REQUEST(ITEM_key, DEPARTMENT_key, ...)

Each tuple in the resulting relation includes keys of a supplier, an item and a department such that the supplier really stock this item, and this item is really requested by this department. The embedded join in the product operation takes care that each tuple is constructed of tuples in STOCK and REQUEST which match on the ITEM's key.

Suppose that we want to make the following query:

Get the SUPPLIERS who STOCK (any) ITEM REQUESTED by (any) DEPARTMENT.

The desired suppliers are received by performing the PROJECT operation by the
SUPPLIER's key on the above expression.

Now suppose that we change a little bit the query and ask:

Get the NAMES of SUPPLIERS who STOCK ITEM REQUESTED by DEPARTMENT.

Since NAME is an attribute of SUPPLIER and not of the relationship STOCK the projection by NAME should be performed on SUPPLIER*STOCK*REQUEST

The first product operation has noting to do with the relativization, but it needed here to connect the supplier keys, which appear in STOCK with appropriate attribute names which appear in SUPPLIER. This kind of "gluing" an entity-relation with a relationship-relation using a natural-join appears also for other NL constructs, whenever an entity's attribute appears explicitly in the construct.

4.2 RESTRICTION BY A CONSTANT AND THE SELECT OPERATION

A restriction is a construct where objects are characterized by a specific value or values of one of their attributes. A specific value may be defined as a constant.

For example

The DEPARTMENT HAS the NAME (EQUALS to) "engineering". or

The ITEM HAS TYPE GREATEST than (> 3).

The SUPPLIER STOCKS QUANTITY SMALLER than 7 of ITEM

The meaning is given by the following three expressions respectively:

select NAME = ENGINEERING (DEPARTMENT).

select TYPE > (ITEM).

select QUANTITY < (STOCK).
4.3 RESTRICTION BY A VARIABLE AND THE $\delta$-JOIN OPERATION

Restriction can appear also with a comparison of an attribute to an attribute (the same, or different) of another entity.

For example

The EMPLOYEE HAS NAME (which is) EQUAL to NAME of SUPPLIER. or
The ITEM HAS TYPE GREATER than FLOOR OF DEPARTMENT.

The meaning is given by the following expressions respectively:

\[
\text{EMPLOYEE} \stackrel{\text{EMPLOYEE.NAME = SUPPLIER.NAME}}{\text{join}} \text{SUPPLIER.}
\]

\[
\text{ITEM} \stackrel{\text{TYPE > FLOOR}}{\text{join}} \text{DEPARTMENT.}
\]

REMARK: In the first example the attributes NAME should be renamed, because the operation is defined only for different such (see Sect. 2.7).

4.4 COORDINATION

Coordination means connecting sentences by the logical connectives "and", "or".

4.4.1 OR COORDINATION AND THE BORDERED UNION

For example

The ITEM IS STOCKED by SUPPLIER OR REQUESTED by DEPARTMENT.

which its meaning is given by

\[
\text{STOCK or REQUEST}
\]

The resulting relation includes tuples with keys of SUPPLIER, ITEM, DEPARTMENT such that either the item is stocked by the supplier or the item is requested by the department. If ITEM is stocked by SUPPLIER the combination ITEM, SUPPLIER appears with all the values of DEPARTMENT in REQUEST. Similarly, if ITEM is
requested by DEPARTMENT, the combination ITEM, DEPARTMENT appears with all the values of SUPPLIER in STOCK.

4.4.2 AND COORDINATION AND THE BORDERED INTERSECTION (PRODUCT)

In the case of AND coordination the PRODUCT operator which reduces to BORDERED INTERSECTION gives the right meaning. For example, if in the example of section 4.4.1 we replace the OR by NOT, the appropriate RRA expression is

\[ \text{STOCK} \cdot \text{REQUEST} \]

which gives all tuples with ITEM, SUPPLIER, DEPARTMENT such that ITEM is both stocked by SUPPLIER and requested by DEPARTMENT.

4.5 NEGATION AND THE NOT (COMPLEMENT) OPERATION

When the word "NOT" appears in a sentence the meaning of the sentence turns to its logical complement. Assuming that the DB includes true facts and what is not included is not true (The closed world assumption [Rei]), the complement operator enables us to compute the opposite meaning of a sentence.

As we have seen, the meaning of

\[ \text{SUPPLIER STOCKS ITEM} \]

is given by the relation STOCK. In STOCK all the facts about "who" stocks "what" are kept. The meaning of the opposite sentence

\[ \text{The SUPPLIER DOES NOT STOCK ITEM.} \]

is all the relevant pairs of "who"-and "what" which do not appear in STOCK. This is accepted by the following expression:

\[ \text{not}([\text{SUPPLIER}][\text{SUPPLIER-key}], [\text{ITEM}][\text{ITEM-key}]) (\text{STOCK}). \]

REMARK: When a NL expression is a combination of negation and coordination it can be converted using the De-Morgan formulas. This property is induced on the
RRA for the NOT, BORDERED UNION and BORDERED INTERSECTION.

4.6 SET COMPARISON, UNIVERSAL QUANTIFIERS AND THE SET JOIN (GENERALIZED DIVISION)

Set comparison is not "natural" in NL, but is equivalent to universal quantifiers like "all", "at least", "more than" etc., which appear frequently. Suppose we want to check which supplier stocks all the items in the database. Such supplier can be characterized as follows:

The SUPPLIER STOCKS a SET of ITEMS which CONTAINS the SET of (all) ITEMS

The operation which gives all the suppliers satisfying this sentence is the SET-JOIN:

\[
\text{STOCK} \quad \text{set-join} \quad \text{STOCK.ITEM-key} \quad \text{ITEM.\_key} \quad \text{STOCK.ITEM-key} \quad \text{ITEM.\_key}
\]

Suppose now that we want to check which supplier stocks all the items that any other supplier supplies. Such supplier can be characterized as follows:

The SUPPLIER STOCKS a SET of ITEMS which CONTAINS the SET of ITEMS SUPPLIED by (any) SUPPLIER

The operation which gives all the suppliers satisfying this sentence is again the SET-JOIN:

\[
\text{STOCK} \quad \text{set-join} \quad \text{SUPPLY.ITEM-key} \quad \text{SUPPLY.ITEM-key}
\]

A special case of this construct is when the supplier stocks all the items that he supplies. Such supplier is characterized by the same RRA expression with renamed attributes in a certain way. As a consequence the division is automatically followed by a selection of tuples such that the first and the last supplier are the same. This issue called referencing is taken care of in section 4.9.
4.7 AGGREGATE FUNCTIONS AND THE AF OPERATORS

 Aggregate functions are related to operations performed on sets. In NL, their names appear as common keywords like SUM, NUMBER OF (COUNT), MINIMUM, MAXIMUM, AVERAGE, etc., with a well-defined meaning.

 For example:

 The SUPPLIER SUPPLYING NUMBER (COUNT) of ITEMS...
 (The three dots mean that usually there is a continuation of such construct, e.g., the SUPPLIER SUPPLYING NUMBER of ITEMS GREATER than 100.)

 or

 The MINIMAL QUANTITY for (any) ITEM STOCKED BY (any) SUPPLIER...

 or

 The DEPARTMENT REQUESTS SUM of QUANTITIES of ITEMS...

 The respective AF operations are

 \[ \text{COUNT}_{\text{ITEM-key}}(\text{SUPPLY}) \]
 \[ \text{MINQUANTITY}(\text{STOCK}) \]
 \[ \text{SUMQUANTITY}(\text{REQUEST}) \]
### 4.8 SUMMARY OF BASIC CONSTRUCTS

The meanings of NL constructs expressed by a respective RRA operations are summarized in the following table.

<table>
<thead>
<tr>
<th>Examples</th>
<th>NATURAL LANGUAGE (Syntax)</th>
<th>RRA (Semantics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...ITEM HAVING COLOR = &quot;RED&quot;</td>
<td>Restriction by a constant</td>
<td>Select</td>
</tr>
<tr>
<td>...ITEM HAVING COLOR = COLOR OF ITEM...</td>
<td>Restriction by a variable</td>
<td>( \delta )-join (parameters: ( =, \neq, &gt;, &lt;, \geq, \leq ))</td>
</tr>
<tr>
<td>...ITEM REQUESTED BY DEPARTMENT MANAGED BY EMPLOYEE...</td>
<td>Relativization</td>
<td>Product (reduces to Natural-join)</td>
</tr>
<tr>
<td>...DEPARTMENT REQUESTING ITEM... AND MANAGED BY EMPLOYEE...</td>
<td>AND coordination</td>
<td>Product (reduces to intersection)</td>
</tr>
<tr>
<td>...DEPARTMENT EMPLOYING EMPLOYEE... OR HAVING LOCALITY...</td>
<td>OR coordination</td>
<td>Bordered union</td>
</tr>
<tr>
<td>...DEPARTMENT NOT REQUESTING...</td>
<td>Negation</td>
<td>Complement</td>
</tr>
<tr>
<td>...SET ITEM CONTAINS SET ITEM...</td>
<td>Set comparison (Universal Quantifiers)</td>
<td>Set-join (Generalized division, parameters: ( =, \neq, &gt;, &lt;, \geq, \leq ))</td>
</tr>
<tr>
<td>...SUM QUANTITY.</td>
<td>Aggregate function</td>
<td>AF: sum, min, max, count</td>
</tr>
</tbody>
</table>

The RRA RENAME and PROJECT do not have matching NL constructs but they are necessary: the first one to correctly construct RRA expressions (i.e., expressions that give the correct meaning of given NL constructs; the second to drop unnecessary attributes from relations taking part in an RRA expression.)
4.9 THE REFERENCE AND THE EMBEDDED JOIN

Referencing is common in NL. It is used in NL via expressions like "the same", "the above", "that", etc.

For example,

"Department requests item which is supplied to IT."

"to IT" means referring to the same department mentioned at the beginning of the sentence.

Computing tuples which satisfy such sentence as a predicate, is equivalent to computing tuples which satisfy the following predicate:

"Department requests item which is supplied to department."

and then selecting only the tuples where the two departments' values are the same.

The RRA was designed to take care of references automatically. It is done by a natural join operation embedded in the RRA's operators (see definitions in Part 2). All that should be done is to rename the entities involved in referencing with the same name. Since the embedded join is automatically performed on attributes with the same names, the desired result is achieved.

The RRA expression for the example above is the following:

\[
\text{REQUEST} \cdot \text{SUPPLY}
\]

where

\[
\text{REQUEST} = \text{REQUEST}(\text{DEPARTMENT-key,ITEM-key,...})
\]

\[
\text{SUPPLY} = \text{SUPPLY}(\text{ITEM-key,...,DEPARTMENT-key,...})
\]

and the product operation reduces to a natural join over \text{DEPARTMENT-key} and \text{ITEM-key}. 

Technion - Computer Science Department - Technical Report CS0405 - 1986
4.10. CONSTRUCTING RRA EXPRESSIONS FOR COMPLEX SENTENCES

The basic constructs presented in Sections 4.1 to 4.8 can be combined to form complex sentences. Such combination induces an RRA expression which consists of respective RRA operators and gives the correct meaning of the complex sentence.

For example

"Department requests item which is not stocked by (any) supplier or has a red color."

This sentence consists of the following basic constructs:
1) "Department requests item..." (relativization)
2) "Item not stocked by (any) supplier" (negation)
3) "Item has a red color". (restriction)
4) "Item not... or has..." (coordinatior)

The matching operations are
1) REQUEST • ...
2) not[ITEM[KEY], SUPPLIER[KEY]](STOCK)
3) selectCOLOR=RED(ITEM)
4) (not...) or (select..)

Combining it altogether we get

REQUEST* ((not[ITEM[KEY], SUPPLIER[KEY]](STOCK)) or (selectCOLOR=RED(ITEM))).

5. THE EQUIVALENCE OF RA AND RRA

The equivalence is implied by expressing each RA operator using RRA and vice versa.

5.1 The RA Operators Expressed by the RRA's

A complete set (not minimal) of RA operators is the following ([U1]):

1. renaming
2. selection
3. projection
4. Cartesian product (×)
5. union (∪)
6. intersection (∩)
7. difference (−)

For a precise definition of RA’s operators see [Pir]. All the operators except the difference can be easily expressed by the RRA’s ones.

The definition of the difference is the following:

Let \( R_1 = R_1(A) \) and \( R_2 = R_2(A) \) be two (union-compatible) relations and let \( R' \) be their difference:

\[ R' = R_1 - R_2 \]

then

\[ S' = S(R') = \{ x : D_2 | x \in A \} \]

\[ V' = \{ t' : A \rightarrow \bigcup D_2 | (t \in V(R_1))(t = t' \wedge t \notin V(R_2)) \} \]

Expressing it using RRA we get:

**Lemma 1:**

\[ R_1 - R_2 = \text{not}_{[R_1[x]]} \{ x \in A | (R_1 \circ R_2) \text{ or } \text{not}_{[R_2[x]]} \} \]

5.2 Expressing the RRA Operators by the RA’s

The RRA renaming, projection, selection, product and ⋋-join are identical or very similar to their matching RA’s. The expressions for the bordered union, the generalized division and the not operator appear in the following lemmas:

**Lemma 2:**

Let \( R_1 = R_1(A \cup B) \), \( R_2 = R_2(B \cup C) \)
Lemma 3:
Let \( R_1 = R_1(A \cup B \cup D_1) \) and \( R_2 = R_2(B \cup C \cup D_2) \)

then

\[
R_1 \cup R_2 = R_1 \times R_2([C] \cup R_1[A] \times R_2)
\]

Lemma 3:
Let \( R_1 = R_1(A \cup B \cup D_1) \), \( R_2 = R_2(B \cup C \cup D_2) \)

and

\[
R'_2 = \text{rename}_{B \leftrightarrow B'}(R_2) = R'_2(B' \cup C \cup D_2)
\]

then

1. \( R_1 \text{ set-join} R_2 = (\text{select}_{B=B'}(R_1[A \cup B]) \times R_2'[B' \cup C]) - \\
   - (R_1[A \cup B] \times R_2 - R_1 \times R_2'[A \cup B \cup B' \cup C]))[A \cup B \cup C]
\]

(See [Pir] for generalized division without embedded natural join.)

2. \( R_1 \text{ set-join} R_2 = R_2 \text{ set-join} R_1 \)

3. \( R_1 \text{ set-join} R_2 = (R_1 \text{ set-join} R_2) \cap (R_1 \text{ set-join} R_2) \)

4. \( R_1 \text{ set-join} R_2 = \\
   R_1 \text{ set-join} R_2 \)

Lemma 4:
Let \( R = R(A) \), \( R_x = R_x(x) \), \( x \in A' \subset A \).

Then

\[
\text{not}_{[R_x]_{x \in A'}}(R) = \bigcap_{x \in A'} R_x - R[A']
\]

REMARK: The above lemmas' proofs follow directly from the definitions, though some of them are quite long.
6. CONCLUSION

The RRA was presented and redefined, and its strong connection to basic NL constructs was demonstrated. As such, the RRA is a convenient tool for defining the semantics of NL subsets or NL-like languages being used as query languages over relational databases. This kind of semantics enables a direct and relatively simple implementation of such query languages. Since RA and RRA are equivalent as was shown, these languages can be Codd-complete with the aggregate functions' additional power.

The ideas presented has been developed during the ERROL project. ERROL (an Entity Relationship Role Oriented Language) is an English like language. The project has been concentrated on user interface to relational databases based on ERROL ([Alp],[Coh],[RCI],[IR]).
REFERENCES


