QUERY LANGUAGES FOR RELATIONAL DATA BASE MANAGEMENT SYSTEMS

by

Brian M. Jervis

B.Sc., University of British Columbia, 1972

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the Department
of
COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard.

The University of British Columbia

May 1974

ABSTRACT

A new data base independent query language for relational systems is presented. Queries in this language specify only properties of the data which is to be retrieved. An algorithm for reducing queries to a response relation is described. This reduction algorithm makes use of Micro-Planner to decide which relations in the data base are applicable to the query, and how these relations should be manipulated. A semantic model is used as the basis for this work. This query language is also compared with existing languages.

Table of Contents

ACKNOWLEDGEMENT

I would like to express my appreciation to Dr. Ray Reiter for the valuable guidance, direction, and inspiration he provided throughout this work. I would also like to thank Dr. Doug Seeley for his helpful suggestions.

The financial support of the National Research Council is gratefully acknowledged.

INTRODUCTION

Historically, data base management was performed by individual programmers who attempted to design their files in such a way as to optimize the execution of their programs. If a new application was found, the data would be re-arranged, and a new file containing redundant information would be created.

Since this approach is both costly and impractical, general purpose data base management systems are now in widespread use. These systems permit many users, each with a different application, to concurrently share a dynamic data base. There are now a large number of such systems in existence[6].

From the development of these systems emerged a number of general principles for designing data base management systems[5]. As a result, a number of new proposals have been put forward. One such proposal is the *relational approach* to data base management. In this approach, data bases are viewed as a collection of time varying relations which are operated upon by a number of set theoretic operations.

This thesis is primarily concerned with user languages for relational systems. The relational calculus, which has been proposed by Codd[11] as the basis for all query languages for relational systems, is critically examined. Several inherent difficulties are observed, which lead to the proposal of a new framework.

This framework includes a new query language which requires users to specify only the properties of the data they want retrieved. This language is not data base dependent, and thus queries are expressed in the same fashion regardless of the current organization of the relations. This requires the system to be capable of deciding which elements of the data base are relevant to the user's request, and deciding how these elements should be manipulated in order to produce the correct response. This is a non-trivial problem which adds a completely new dimension to the systems proposed by Codd.

The current system is not only capable of answering queries but will accept real world knowledge which affects the response to the queries, introduce new relations, and accept new information about the current data base which is automatically used to optimize the retrieval process.

I. THE RELATIONAL APPROACH TO DATA BASE MANAGEMENT

¤ 1.1. The Relational Model of Data ¤

"Since set theory provides a wealth of operations for dealing with relations, a set-theoretic data structure appears worth investigation."[1]

In this thesis, we are concerned with the relational model of data presented by Codd [8]. The term relation is used here in its accepted mathematical sense. Given sets S1,S2,...,Sn not necessarily distinct, R is a relation on these n sets if it is a set of n-tuples, each of which has its first element from S1, its second element from S2, etc. More concisely:

$$R(S1\ S2\ \ldots\ Sn) \subset S1 \times S2 \times \ldots \times Sn.$$

We refer to Sj as the jth domain of R. As defined above, R is said to have degree n.

• Example 1.1 •

If S1 is Project#, S2 is Project name, and S3 is Project location, then:

R(Project,Project name,Project location) ⊂
Project# x Project name x Project location

Relations are represented in a table-like format, with the domain names appearing at the top of the table, and the tuples in the relation appearing beneath.

---

[1] Childs, D.L. "Description of a Set-Theoretic Data Structure." Proceedings of the 1968 FJCC, pp.557-564.

• Example 1.2 •

R (PROJECT#	PROJECT-NAME	PROJECT-LOCATION)
1	ROYAL TOWERS	VANCOUVER
2	BURRARD SHIPYARDS	VANCOUVER
3	P.C.C.	MERRITT
4	RAPID TRANSIT	VICTORIA
5	GRANVILLE MALL	VANCOUVER
6	OLYMPIC SEAWAY	VICTORIA

In this framework, the data base consists of a collection of time varying relations of assorted degrees. The relations are not static, but constantly changing. They are subject to insertion, deletion, and modification.

Since relations are only special sets, the relational model makes use of set-theoretic operations in order to perform the data management functions. This set of operations, called a _relational algebra_, forms the only set of operations which may select data from the data base.

• Example 1.3 •

In order to form the relation which lists the projects who are located in Vancouver, one restricts R to those tuples whose value of PROJECT-LOCATION is Vancouver. This produces a new relation R' whose tuples are:

R' (PROJECT#	PROJECT-NAME	PROJECT-LOCATION)
1	ROYAL TOWERS	VANCOUVER
2	BURRARD SHIPYARDS	VANCOUVER
5	GRANVILLE MALL	VANCOUVER

¤ 1.2. Classes of Relations ¤

Relations fall into one of two classes - *simple*, or *compound*. Simple relations have the property that no domain in the relation is itself another relation. Compound relations, on the other hand, have the property that at least one domain in the relation is itself a relation. Compound relations define hierarchies.

• Example 2.1 •

In this example, R is a compound relation.

```
R (PART# PART-DESC Q-O-H (PROJECT# PROJECT-DESC NUMBER-ORDERED))
  ( 1     WIDGETS    47      1        HOUSING           40
                             2       APARTMENTS          3  )
  ( 2     FIDGETS     4      1        HOUSING            2
                             4      TOWN-HOUSES          1  )
```

Codd[9,10] points out that compound relations can be reduced to a number of corresponding simple relations with no information loss. Further, he shows that this is not only possible, but desirable. This process is called *normalization*, and the new relations which are produced are said to be *normalized*.

There are two advantages to using data bases which consist of normalized relations. Firstly, it simplifies the relational algebra, since the operations on relations need only deal with simple relations. Secondly, normalized data bases are consistent, non-redundant, and free of undesirable update dependencies.

• Example 2.2 •

In example 2.1, R can be represented by:

```
P (PART# PART-DESC Q-O-H)
Q (PART# PROJECT# PROJECT-DESC NUMBER-ORDERED)
```

This representation is preferable to that of example 2.1. Notice that with relation R, if a new part is added to the system, it cannot be recorded until at least one project which orders that part is also recorded. With the representation of example 2.2 however, this is no longer true since the relation P contains no information about the projects which use the part. It deals with <u>part information only</u>.

This representation is still inadequate. If a new project is recorded, it cannot be put in the data base until the parts which it will be using are known. Therefore relation Q is split further.

• Example 2.3 •

```
PART    (PART# PART-DESC Q-O-H)
PROJECT (PROJECT# PROJECT-DESCRIPTION)
SUPPLY  (PART# PROJECT# NUMBER-ORDERED)
```

The information dealing with parts, projects, and the supply of parts to projects is now represented by separate relations. The data base contains exactly the same information as before but is now free of the previous update dependencies.

For the purposes of this thesis, we assume that all relations are normalized.

¤ 1.3. Advantages of Relational Systems ¤

There are many properties of relational systems which make them more desirable than traditional fact retrieval systems. These can be summarized as follows:

1. They provide a high degree of independence between the user and the data base.
2. The data bases are consistent, and non-redundant.
3. The data structure is simple, yet extremely powerful.
4. They have a superior query capability.
5. They provide good interactive support for the casual user.

The chief virtue of relational systems is the high degree of user - data independence they provide. This independence occurs at two levels. Firstly, the user is presented with a logical representation of his data which may well differ from its physical representation. This is possible since relations can only be accessed through the operations of the relational algebra. This is a valuable property, since factors such as efficiency and machine configuration can now be taken into account without influencing the users view of his data. While this is a fundamental concept in information retrieval, most current systems violate it[5].

Secondly, the user need not be aware of how his data is arranged into relations. The information which must be stored can be broken into relations in any convenient manner, and information in the data base, even though it may be implicit (i.e. in separate relations), will still be retrievable. No current fact retrieval systems are capable of drawing conclusions from information which is stored in separate files.

Another advantage is that through the process of normalization, redundant and inconsistent information can be removed from the data base. Thus, these systems tend to be free of undesirable update and deletion dependencies. This process also tends to arrange data bases in conceptually clear and concise units.

Thirdly, the data structure is both simple and versatile. Other data structures, such as rings, trees, networks, and graphs, can be represented using relations.

Finally, relational systems lend themselves to a variety of different query languages. This point will be amply demonstrated in the following chapters.

◻ 1.4. Previous Research ◻

Over the past five years, there has been considerable interest in the relational approach to data base management.

The first contributions were by Feldman and Rovner[13]. They introduced an approach whereby data is stored in the form of binary relations. Users can access these relations from Algol programs by means of attribute-object-value triples. A similar approach was taken by Levin and Maron[21]. Several implementations have been based on this scheme, including one by Gammill[14].

The initial contribution towards the goal of a system based on arbitrary relations came from Childs[3,4]. He presents a set theoretic data structure comprised of set operations, datum names, data, and set names. In this system, the set operations are implemented as subroutines which operate on sets in the data base. This approach is still being followed, although the operations in current relational algebras differ from those presented by Childs. The emphasis of this system, however, was still on binary relations, with access paths being pre-defined.

Since Child's paper, most new work has been based upon that of Codd[8]. In this article, he argues in favor of relational data bases, and outlines a set of operations on relations which are applicable to relations of arbitrary degrees. With this framework access paths between relations need not be pre-defined. Supplementing this article, Codd[9,10] deals with normalization of data bases in order to eliminate redundant information and unwanted domain dependencies.

In the area of user languages, the main advances have again come from Codd[7,11,12]. In [11] he defines a relational algebra and proposes the relational calculus, an applied predicate calculus with n-tuple variables. Based upon the relational calculus is DSL-ALPHA[7], an Asap-like query language.

The above papers inspired many implementations. Among these are the systems of Strnad[27], Notley[24], Goldstein and Strnad[15], and Bracchi[2]. Unfortunately, these works show little originality. They are all basically implementations of the relational algebra. One very noticable point about research in this area is that although there are many implementations based on Codd's work, very little has been done to extend it.

Noticable exceptions are Palermo[26], who introduces a new relational calculus with an improved reduction algorithm, and Heath[17], who outlines some unacceptable file operations in a relational data base.

As of yet, there has been no comprehensive study of how relations can best be stored in order to optimize system performance. Since the operations of the relational algebra are well defined, and since the nature of these operations is easy to observe, one would expect some concrete results could be obtained. It is surprising that such little effort has been put into this area, since efficiency is one of the largest problems inhibiting the commercial use of relational systems. Palermo's concept of semi-join is the only advance that has been made.

II. RELATIONAL ALGEBRA

There are certain primitive operations on relations which form the basis of any relational system. This set of operations is referred to as a relational algebra.

These operations are the only ones which can manipulate the relations - all other routines in the system must access the relations through them. This implies that the relational algebra should be chosen so that the selective power it possesses is complete, in the sense that any request which could possibly be formulated in the system can also be formulated as a query in the relational algebra[4].

The operations of the relational algebra take relations as their arguments, and produce a response which is always a new relation. Accordingly, the result is called the response relation.

This section defines the relational algebra which this thesis adopts. It is based upon the algebra proposed by Codd[11].

¤ 2.1. Sample Relations ¤

The following relations will be used as examples throughout this chapter.

```
R1    (       A           B       )
              1           2
              4           7

R2    (   SUPPLIER#    PART#      )
             1           3
             1           2
             2           1
             2           2
             2           3
             3           1

R3    (     PART#    PART-NAME )
             1           A
             2           B
             3           C
```

¤ 2.2. Operations on Relations ¤

The relational algebra includes eight operations on relations. They are:

A. Union
B. Intersection
C. Difference
D. Cross product
E. Projection
F. Join
G. Division
H. Restriction

Of these operations, division and restriction are definable in terms of the first six.

2.2.A Union

The union of two relations R and S is defined to be:

$R \cup S = \{ (r) : r \in R \vee r \in S \}$,

where R and S are each of degree n.

• Example 2.1 •

The union of R1 and R2 is:

```
RP    (      AP          BP      )
              1           2
              1           3
              2           1
              2           2
              2           3
              3           1
              4           7
```

Union is typically used to create a relation which enumerates the values of some domain using a set of other relations, each of which contains this domain.

2.2.B Intersection

The intersection of two relations R and S is defined to be:

$R \text{ INT } S = \{ (r) : r \in R \,\&\, r \in S \}$,

where R and S are each of degree n.

• Example 2.2 •

The intersection of R1 and R2 is:

```
RP    (      AP          BP      )
              1           2
```

Intersection creates a relation that contains the tuples which are common to all the relations being intersected.

2.2.C Difference

The difference of two relations R and S is defined as:

$R - S = \{ (r) : r \in R \ \& \ r \neg\in S \}$

• Example 2.3 •

The difference of R1 and R2 is:

RP (	AP	BP	)
	1	3	
	2	1	
	2	2	
	2	3	
	3	1	

2.2.D Cross Product

The cross product of two relations is defined to be:

$R \times S = \{ (r,s) : r \in R \ \& \ s \in S \}$

• Example 2.4 •

The cross product of R1 and R2 is:

RP (	SUPPLIER#	PART#	A	B	)
	1	3	1	2	
	1	3	4	7	
	2	1	1	2	
	2	1	4	7	
	2	2	1	2	
	2	2	4	7	
	2	3	1	2	
	3	1	4	7	
	1	2	1	2	
	1	2	4	7	
	2	3	4	7	
	3	1	1	2	

2.2.E Projection

Suppose r is a tuple of an n-ary relation R. Then for j=1,2, ... n, r[j] denotes the jth component of tuple r, or the projection of r on domain number j. This notation is extended to a list A = (j1,j2, ... jk), where ji∈(1,2, ... n).

Now, r[A]={r[j1],r[j2],...,r[jk]} .

■ Definition ■

The projection R[A] of R on A is defined by:

R[A] = {r[A] : r∈R} .

Thus, if the relation R (PART# PART-NAME) were projected on its first domain, a relation which contains only part numbers would be formed.

• Example 2.5 •

The projection of R2 on its second domain, R2[2], is:

```
SP    (     PART#   )
              1
              2
              3
```

2.2.F Join

Join is perhaps the most powerful operation of the relational algebra. It is defined as follows. Let θ(r,s) denote an arbitrary predicate whose only variables are of the form r[i], s[j]. Then the θ join of R with S is defined by:

R[θ]S = { (r,s) : r∈R & s∈S & θ(r,s) }

• Example 2.6 •

Suppose s and p are tuples of relations R2 and R3 respectively. Then the join R2(s[2]=p[1])R3 is:

RP (	SUPPLIER#	PART#	PART#	PART-NAME)
	1	3	3	C
	1	2	2	B
	2	1	1	A
	2	2	2	B
	2	3	3	C
	3	1	1	A

The join R2(s[2]>p[1])R3 is:

RP (	SUPPLIER#	PART#	PART#	PART-NAME)
	1	2	3	C
	2	1	3	C
	2	1	2	B
	2	2	3	C
	3	1	3	C
	3	1	2	B

The join R2(s[2]*5=r[1])R1 is:

RP (	SUPPLIER#	PART#	A	B)
	2	1	4	7
	3	1	4	7

2.2.G Restriction

Let θ be an arbitrary predicate whose only variables are of the form r[j]. Then define the restriction $r(\theta)$ of R by θ to be:

$$R(\theta) = \{ (r) : r \in R \ \& \ \theta(r) \} .$$

• Example 2.7 •

The restriction of R2 , R2(r[2]=2) is:

RP (	SUPPLIER#	PART#)
	1	2
	2	2

The above is a typical use of restriction. If there is a relation that indicates the suppliers who supply parts, then restricting that relation to the case where the part number is two will result in a relation showing which suppliers supply part number two.

2.2.H Division

Division is the most counter-intuitive operation in the relational algebra. It is included since it is the algebraic counterpart of the universal quantifier.

Assume R is a binary relation. Then the image set gR(x) of x under R is defined by:

$$gR(x)=\{y : (x,y)\in R\}$$

• Example 2.8 •

When r=(1 3), the image set gR(2)=(1,2) since r[2]=3, and the tuples (1 3) and (2 3) are both in R.

The division of R on A by S on B is defined by:

$$R[A/B]S=\{r[ABAR] : r\in R \;\&\; S[B]\subseteq gR(r[ABAR])\} ,$$

where ABAR is the domain list which is the complement of A, and gR(x) is the image set of x under the relation R. In this definition, we consider that R is a binary relation composed of the two compound domains A and ABAR.

The process of division operates as follows. Consider each tuple in R to consist of two elements, r1 and r2. Then r1 is a tuple in the quotient of R[A/B]S if for each tuple r3 in S[B], there exists a tuple in R with r3=r2 and r1 is always the other "half" of the tuple in which r2 is contained.

• Example 2.9 •

The quotient of R2[2/1]R3[1] is:

```
RP    (      S#      )
             2
```

since supplier 2 is the only supplier who supplies all parts. Notice that r3[1] produces a relaticn which enumerates the part numbers.

◻ 2.3. Choice of Operations in the Relational Algebra ◻

Other authors[3,4,8,17,24] have proposed relational algebras which appear to have the same selective power[4], as the one defined above, yet they contain considerably different operations.

This thesis deals with queries which are expressed in a predicate calculus notation, and must eventually be reduced to a sequence of operations in the relational algebra. Since projection and division form the algebraic counterparts of the existential and universal quantifiers, and since restriction can be used to process restrictions in the query, the choice of this algebra is fitting.

¤ 2.4. Implementation of the Relational Algebra ¤

The operations of the relational algebra have been implemented using the programming language LISP[31]. This section explains how the relations are stored, and shows the syntax of the relational algebra queries.

Each relation has two properties on its property list. The first property is DOMAINS, whose value is a list of all the domain names for the relation. The second is TUPLES, whose value is a list of all the tuple names this relation contains. Then, under the flag DATA on each tuple name is a list which is the actual tuple in the relation.

• Example 4.1 •

```
(GET 'R 'DOMAINS) = (PART# PART-NAME)
(GET 'R 'TUPLES)  = (T1 T2 T3)
(GET 'T1 'DATA)   = (1 A)
```

Notice that in LISP, function calls are written in prefix normal form. The name of the function and its arguments are always enclosed in brackets, and function calls can be nested.

The notation 'R is equivalent to (QUOTE R). (The function QUOTE returns as its value the argument which it was passed without evaluating it.) thus, if one set N equal to 5, the value of N would be 5, whereas the value of (QUOTE N) would be N.

This data structure allows both quick retrieval of the tuples, and ease in processing the relations tuple-wise. This is crucial, since all operations in the relational algebra access the relation by tuple, rather than by domain.

• Example 4.2 •

The following commands illustrate the syntax of each operation in the relational algebra:

```
(RINTERSECT RLIST)
(RUNION RLIST)
(RCROSS RLIST)
(RDIFF 'REL1 'REL2)
(PROJECT 'REL1 'LIST)
(JOIN 'REL1 'REL2 THETA)
(DIVIDE 'REL1 'LIST 'REL2 'LIST)
(RESTRICT 'REL1 THETA) ,
```

THETA is an arbitrary LISP predicate, and DLIST is a list of domain numbers of the preceeding relation, and RLIST is a list of relation names.

In order that specific elements of tuples can be referenced within the predicates, the user can always assume that the variable T1 points to the current tuple in REL1, and T2 points to the current tuple in REL2. In order to express a predicate which says "the second domain of REL1 must be equal to ten times the third domain of REL2", one would write:

```
(EQ (ELEM T1 2) (TIMES (ELEM T2 3) 10)) .
```

Appendix 3 contains a number of sample queries in the relational algebra, and shows the output they produce. For a complete listing of the routines which define these operators, please see Appendix 7.

▫ 2.5. The Relational Algebra as a Query Language ▫

Despite the fact that most current relational systems use the relational algebra as their top level query language, it is clearly unsuitable for general use. The user is required to generate the correct sequence of operations which will retrieve the desired data. Queries are expressed in terms of "how to retrieve the data", rather than in terms of what is wanted. Operations such as division are also counter-intuitive, and the average user would find it difficult, if not impossible, to master their use.

III. RELATIONAL CALCULUS

¤ 3.1. Introduction ¤

In an attempt to provide a more reasonable query language for relational systems, Codd[11] introduced a relational calculus. This query language is not intended to be used directly by users, but to be used as the basis for higher level query languages.

Queries in the relational calculus are expressed using a predicate calculus notation. They tend to be more "property defining" than the queries in the relational algebra.

This chapter presents a relational calculus based on that of Codd[11] and Palermo[26]. It also describes a reduction algorithm which takes a query in the relational calculus and reduces it to a semantically equivalent sequence of operations in the relational algebra. It concludes with a critical evaluation of the usefulness of the relational calculus as a framework for relational systems.

▫ 3.2. The Relational Calculus ▫

3.2.A The Alphabet

The following notation is adopted:

Tuple variables	r1,r2, ...
Range Predicates	P1,P2, ...
Individual Constants	a,b, ...
Index constants	1,2, ...

3.2.B Terms

There are two types of terms in the relational calculus - _range terms_, and _join terms_.

Range terms are used to identify the range of each tuple variable in the query. For each relation Ri, there exists a corresponding monadic predicate Pi which determines whether or not any tuple r in the data base is an element of Ri. [Thus, P can tell if a _tuple_ is in a relation R, whereas R can tell if n _elements_ (where R is of degree n) are in R.]

▪ Definition ▪

A range term is a monadic predicate followed by a tuple variable.

• Example 2.1 •

P3r1 is a range term.

Join terms in the relational calculus are used to determine how relations in the data base are to be joined. They are arbitrary functions, whose purpose is to show how the tuples which are their arguments are to be related.

▪ Definition ▪

An indexed tuple is an expression of the form r[N], where r is a tuple variable, and N is an index constant. Its purpose is to identify the Nth element of r.

▪ Definition ▪

Let a,b be indexed tuples, and c be a constant. Then if θ is a predicate whose only elements are either a and c, or a and b, then θ is a join term.

• Example 2.2 •

R1[1]=r3[2] and (r1[1]*7) = 26

are both join terms, whereas

r1[1]=r3[2]=r5[7] and P1r1

are not.

3.2.C WFFs

The well formed formulae (WFFs) of the relational calculus are defined as follows.

1. Any term is a WFF.
2. If Ψ is a WFF, then so is ¬Ψ.
3. If Ψ1 and Ψ2 are WFFs, so are (Ψ1 & Ψ2) and (Ψ1 v Ψ2) .
4. If Ψ is a WFF in which r occurs as a free variable, then ∃r(Ψ) and ∀r(Ψ) are WFFs.
5. No other formulae are WFFs.

The WFFs of the relational calculus are not suitable as queries in the relational calculus since they allow the formation of meaningless queries. Furthermore, quantified expressions can be written in a more meaningful way than is allowed by the WFFs.

3.2.D Range Formulae

Range formulae attempt to limit the ranges of tuple variables to well defined relations. While doing this, they must still allow the ranges to be specified in some natural manner. This notion was introduced by Palermo[26], and is an improvement over the original formulation by Codd who did not allow join terms in the formulae.

• Definition •

Ψ is a range formula over r if:

1. Ψ is a quantifier free WFF.
2. r is the only tuple variable in Ψ.
3. Ψ is in disjunctive normal form (dnf), and each conjunct contains at least one non-negated range term.
4. The relations defined by each range term in Ψ have the same number of domains.

• Example 2.3 •

P1r1	r1 comes from R1
P1r1 & P2r2	r1 is in R1 and is also in R2
P1r1 & r1[2]=5	r1 comes from R1, and the value of its second domain is 5.

In the definition of range formulae as presented by

Palermo, restriction three above is not present. However, without it, formulae such as:

(P3r3 v r3[2]=1) ,

which clearly do not specify a valid range for r3, are acceptable. Restriction three disallows formulae of this type by specifying that when in dnf, each conjunct must have at least one non-negated range term. Formulae such as:

P3r3 v (P4r3 & r3[2]=1) , and

P3r3 & (P4r3 v r3[2]=1)

which are both valid range formulae are still acceptable using this new definition.

3.2.E Pure Range Formulae

A range formula which consists only of range terms is known as a pure range formula.

3.2.F Range Coupled Quantifiers

▪ Definition ▪

$\exists\Psi$ and $\forall\Psi$ are called range coupled quantifiers, and are defined by the equations:

$\exists\Psi(\Phi) = \exists r(\Psi \,\&\, \Phi)$

$\forall\Psi(\Phi) = \forall r(\neg\Psi \vee \Phi)$

Assume Φ is a WFF having r as a free variable, and Ψ is a range formula over r. Then $\exists\Psi(\Phi)$ and $\forall\Psi(\Phi)$ are also WFFs.

3.2.G Q-formulae

We now define the formulae which can be used in queries in the relational calculus.

▪ Definition ▪

A WFF ϕ in the relational calculus is a Q-formula if it is a conjunction of the form:

ϕ = U1 & U2 & ... & Up & W , where

1. Each U1 is a range formula over ri, i=1,2, ... p.
2. W is either null, or is a WFF in prenex normal form, with free variables v1,v2, ... ,vp, and bound variables Vp+1,Vp+2, ... ,Vp+q.
3. The matrix of W is in disjunctive normal form(dnf).
4. There are no ¬ symbols immediately preceding a join term.
5. Every variable is coupled to a range:
   i. If a variable is free, it belongs to the set of variables whose ranges are specified by u1,u2, ... ,up.
   ii. Every quantifier in W is range coupled. This implies that any bound variable also has its range specified.
6. The matrix of W is devoid of range terms.
7. $p \geq 1$.

These Q-formulae correspond somewhat to the range separable WFFs of Codd[11] and to the C-formulae of Palermo[26]. Examples of Q-formulae can be found in Appendix 4.

3.2.H Q-expressions

We are now in a position to define the queries of the relational calculus. These queries, which will be referred to as Q-expressions, are similar to the Simple Alpha-expressions of Codd[11], and the Gamma-expressions of Palermo[26].

▪ Definition ▪

A Q-expression has the form:

(t1 t2 ... tn) : Q, where:

1. Q is a Q-formula of the relational calculus.
2. The set of tuple variables occurring in t1,t2, ... ,tn is precisely the set of free variables in Q.

▫ 3.3. A Reduction Algorithm ▫

This section shows how a query in the relational calculus can be reduced to a semantically equivalent sequence of operations in the relational algebra. The method used is based upon the reduction algorithm of Palermo[26].

The reduction algorithm does not actually generate a query in the relational algebra. Instead, it works its way through the query, calling upon the operations of the relational algebra to produce new relations which are necessary for the construction of the query's response relation.

The reduction algorithm begins by creating the relations which form the range of each tuple variable. Variations of these relations are then joined, according to the join terms in W. After the appropriate unions and intersections of the remaining relations have been made, the new relation is repeatedly divided or projected in order to take the quantifiers into account. This relation is then projected on the domains of the target list, resulting in the response relation. All operations of the relational algebra are used in this process.

The reduction algorithm operates in such a way as to minimize the amount of necessary core.

3.3.A Global and Local Ranges

In a Q-formula, W has the form:

$$Q(p+1)\ Q(p+2)\ \ldots\ Q(p+q)\ [\theta 1 \lor \theta 2 \ldots \lor \theta k],$$

where Q(i) is a range coupled quantifier, and θi is a conjunction of join terms.

Let $\phi(ik)$ be the subformula of θi consisting of terms whose only variable is r(k), and let:

$\Psi(k)$ = The range formula Uk, if r(k) is a free variable.

= The range formula for the quantifier which binds r(k) otherwise.

■ Definition ■

The local range L(ki) for r(k) in ϕi is defined by the formula:

$$L(ki) = \{ (r) : \Psi(k) \ \& \ \phi(ik) \}.$$

The global range G(k) for r(k) is defined by:

$G(k) = \{ (r) : \Psi(k) \}$.

The local ranges of a variable are simply restrictions of its global range.

When reducing a query, no relation need ever contain a domain which is not explicitly referenced in the query[26]. Thus, we define the _reduced local (global) range_ for a variable to be the projection of its local (global) range on all its referenced domains. It is with these relations that the reduction algorithm deals.

3.3.B The Join Algorithm

In his reduction algorithm, Codd[11] begins by taking the cross product of the global ranges of all the variables used in the query. This cross product is then restricted to the cases defined by the join terms of θ, and the result processed according to the quantifiers. Needless to say, the size of this relation can become unbearably large.

As Palermo[26] observed, the forming of the cross product is unnecessary. The individual local ranges can instead be joined according to the terms of θi, producing the relations Ci. The subset of S defined by θ can now be produced by taking the union of each Ci which was produced. This method results in considerable savings of time and space.

It is the function of the join algorithm to produce the relations Ci defined by their corresponding θi. The algorithm assumes that the local range for each variable in θi has been created.

The join algorithm proceeds as follows.

STEP 1. A list of the reduced local ranges used in θi is created. This list is ordered, with the smallest relation coming first.

STEP 2. The first reduced range is placed in a workspace, called the core, and removed from the list.

STEP 2. A list of all terms in θi which reference a domain in the core is created, since these terms can be used to join the core with some new range from the list.

STEP 4. The range which involves the smallest relation is chosen.

STEP 5. The core is then joined to this range, using the term from θi as the join predicate.

STEP 6. This term is removed from the list, and processing continues with step 2. This process is repeated until either:

i. The range list is not yet empty, indicating more joining is to be done, yet there are no more join terms connected to the core. In this case, save the current core, and go to step 1.

ii. The range list is empty, in which case one can return, providing no cores have been saved. If cores have been saved, then form the cross product of the cores and return.

3.3.C The Reduction Algorithm

The first step in the reduction algorithm is to create the relation defined by θ. Since θ is in dnf, this can be done by taking the union of the relations Ci defined by each θi in θ. Each relation Ci is created using the join algorithm.

STEP 1. Form the reduced global range for each variable in the query. This is done by examining the range formula of the variable.

STEP 2. Form the relations Ci which are defined by θi. In order to do this, first form the reduced local range for each variable used in θi, then utilize the join algorithm with the terms of θi to produce Ci.

STEP 3. Form the union of all Ci, producing the relation Tp+q.

Once the relation Tp+q has been derived, the next step is to take the effect of the quantifiers into account. Quantifiers are processed from right to left - i.e., from Q(p+q) to Q(p). Their effect is:

1. If Q(j) is an existential quantifier, project the relation Tp+j on all domains _except_ those processed by the relation which defines the range of r(j).

2. If Q(j) is a universal quantifier, divide the relation Tp+j by the relation which defines the range of r(j). This results in a relation whose tuples are in some sense "true" for all r(j).

The result of each of the above operations is the relation

Tp+j-1.

STEP 4. The quantifier operations of projection (existential) and division (universal) are applied for each quantifier Q(i) in the prefix of W, with the quantifiers being processed from right to left. This produces the relaticn Tp.

STEP 5. Project the relation Tp on each of the domains specified in the target list. The result is the response relation for the query.

Please refer to Appendix 4 for sample queries in the relational calculus.

¤ 3.4. Implementation of the Reduction Algorithm ¤

The reduction algorithm as described above has been implemented in LISP. Appendix 4, which shows sample reductions, was created using these routines.

The purpose of this section is to show why some restrictions have been placed on the relational calculus for the benefit of the implementation and to show how the queries are represented in LISP.

<u>3.4.A Restrictions on the Relational Calculus</u>

There are two restrictions which have been made in the relational calculus for the benefit of the implementation. The first of these is that in a Q-formula, the matrix of W must be devoid of range terms. This means that when creating the global range for a variable, the matrix of W need not be examined. The result of this restriction is that the range of each free variable must be declared in some Ui rather than in W.

Secondly, the range formulae must be in dnf. Without this restriction, the formulae can become extremely complex, and individual elements of the formula cannot be processed <u>independently</u> of the other elements in the formula. When in dnf, each conjunction will define a valid relation, and thus ranges can be determined by taking the union of the relations defined by each conjunction.

• Example 4.1 •

```
    P7r1 & [P4r1 v (EQ (R1 1) A) v P5r1 v (EQ (R1 2) B)] .
```

Notice that in this example, the second element of the conjunction does not in itself define a valid relation, and thus cannot be processed independently of the first element of the conjunction. When this expression is in dnf however, each conjunct defines a valid relation.

3.4.B Representation of the Queries

Each query is assigned a unique name, and has the following information on its property list:

1. VARS - a list of all the variables in the query, in the reverse order of their appearance.
2. THETA - a list of all θi occurring in the query. For example, (THETA1 THETA2).
3. TARGET - the target list for the query. For example, ((R1 4) (R2 3))

Each element in the list THETA has the flag TERMS on its property list. The value of this flag is a list of the names of the terms occurring in the particular θi.

Each term has its definition as its value. For example, the value of TERM1 might be (EQ (R1 2) 970). On the property list of the term, under the flag VARS, is a list of all the variables in the term. In the above case, this list would be (R1).

Each variable has on its property list the flags:

1. QUANTIFIER - either ALL, if the variable is universally quantified, EXISTS, if the variable is existentially quantified, or NONE.
2. USED-IN-TERMS - a list of all terms in which the variable is used.
3. REFDOMAINS - a list of all domains of the global range of the variable which are referenced anywhere in the query.
4. RANGE - either the name of the reduced global range for the variable, or an expression which defines the global range.

The value of the variable is its current reduced local range.

In writing queries, join terms are expressed as arbitrary LISP predicates, with the list (R1 1) being used to represent the first domain of tuple R1. The above representation completely characterizes the query. The reduction algorithm needs no additional information in order to respond to a query in the relational calculus.

¤ 3.5. Evaluation of the Relational Calculus ¤

It is the position of this thesis that as a top level query language, the relational calculus is inadequate. Further, if the relational calculus is to be used as the target language for a higher level query language, there are a number of problems which must be overcome. The purpose of this section is to evaluate the feasibility of the relational calculus as a framework for relational systems.

Consider the relational calculus as a top-level query language. Since the queries are very much tuple oriented, it is often difficult to express information about domains. In the relational calculus it is possible to say "for all tuples in relation X", something is true, but difficult to say "for all parts" something is true. This can only be done with a single quantifier if the relations which enumerate the parts have exactly the same number of domains. Even then it is possible only if the part occurs in the same position in each relation. If this is not true, several quantifiers will have to be used to quantify one entity. This is indeed undesirable.

Forming queries in the relational calculus tends to be a difficult process.

• Example 5.1 •

Assume the existence of the following three relations:

R1 (SUPPLIER# SNAME SLOCATION)
R2 (PROJECT# PART-NAME)
R3 (SUPPLIER# PART# PROJECT#)

Then the query "Find the numbers of the suppliers, each of whom supplies all parts" is represented as:

R1[1] : P3r1 & ∀P3r2 ∃P3r3 (r1[2]=r3[1] & r2[2]=r3[2])

Despite the fact that this is an applied predicate calculus, it is still not possible for a user to state the _properties_ of the data he wants retrieved.

There are several reasons why the relational calculus is so difficult to use. Firstly, it is very data base oriented. A user must have a thorough knowledge of the current organization of his data base, since in stating the query, the relations which are to be used must be explicitly identified.

This is a very serious flaw, since a prime advantage of relational systems is the independence they present between the system and the current organization of the data. In his original paper, Codd[8] states "users at terminals ... should remain unaffected when the internal representation of the data is changed, and even when some aspects of the external representation are changed". When the relational calculus is used as a top level query language, this basic principle is violated.

Not only does the fact that the queries directly reference the data base make them hard to write, but it means that the same query will have to be expressed differently for a different organization of the data base.

Secondly, not only do the queries tell the system what information to use, they also tell the system what to do with it in order to produce the response relation.

Example:

Assume the existance of the three relations in example 5.1. Then the query "Find the names of the suppliers, each of whom supplies all projects" is represented as:

$R1[2] : P1r1 \ \& \ \forall P2r2 \exists P3r3 \ ((r1[1]=r3[1] \ \& \ (r2[1]=r3[3]) \ )$

Not only does this query tell the system which relations to use, but it also says what should be done with them. Namely, take a tuple r1 in R1. Then for each tuple r2 in R2, there must be some tuple r3 in R3 such that the first domain of r1 is equal to the first domain of r3, and the first domain of r2 equals the third domain of r3. If this is true, save the second domain of r1. Now take the next tuple in R1, and try again.

In order to formulate the above query, the user must know how the data base is arranged, which pieces of it he is interested in, exactly how this information should be related, and how the system can recover his data.

Clearly, any system which forces the user to decide how his data should be selected before he can even formulate his query is unsatisfactory. Such a system is performing only the mechanical part of the retrieval process, while forcing the user to do the real work.

Now consider the relational calculus as the target language for a higher level query language Q. Then Q should possess the following properties:

1. Queries in Q should be expressions of properties of what is to be retrieved.
2. Queries should _never_ reference relations, but instead, reference _domains_. thus, Q will be domain oriented, whereas the relational calculus is tuple oriented.
3. Quantifiers in Q should not be dependent upon the data base, as the relational calculus has been exhibited to be.
4. Queries should not have to contain any information about how the query is to be answered.

In order to overcome the restrictions the relational calculus imposes, the interface between Q and the relational calculus must be capable of taking an arbitrary query expressed in some property defining form, identifying which relations are applicable to the request, determining how they should be joined, and how the retrieval is to be done. No previous system is capable of doing these.

IV. A QUERY LANGUAGE FOR RELATIONAL SYSTEMS

This section presents a new query language for relational systems. The language is an applied predicate calculus which requires users to specify only the properties of the data they want retrieved. Specific relations are never directly referenced in the query. Thus, the language is not data base dependent, and queries are expressed in the same fashion regardless of the current organization of the relations.

The language has been specifically designed as a target language for a natural language system. In fact, a system[16] which compiles queries into a similar representation, but which uses a different problem domain, has already been implemented.

This chapter begins with a formal definition of the query language. This is followed by a somewhat more intuitive explanation of the language, and concludes with a discussion of its use as the target language for a natural language system.

□ 4.1. The Relational Calculus Re-Defined □

Unlike the relational calculus of Codd, queries in the new relational calculus reference domains rather than tuples. This makes the queries data base independent, since specific relations need never be referenced.

4.1.A The Alphabet

The following notation is adopted.

Domain variables	d1,d2, ...
Diadic Predicates	r1,r2, ...
Arbitrary Predicate	F

• Example 1.1 •

PART# and SUPPLIER are both domain variables.

If SUPPLIES is a predicate which says yes or no to "SUPPLIER SUPPLIES PART#" for specific values of SUPPLIER and PART#, then SUPPLIES is a diadic predicate.

4.1.B Terms

There are four types of terms in the relational calculus - simple terms, relational terms, restriction terms, and join terms.

▪ Definition ▪

A _simple term_ is a domain variable.

A _relational term_ is a list of the form:

(d1 r1 d2 <r2 d3 ... <rn dn>>), where <X> denotes an optional occurrence of X.

• Example 1.2 •

PART# and SUPPLIER are simple terms, whereas:

(SUPPLIER SUPPLIES PART#) is a relational term.

• Definition •

A _restriction term_ is a term of the form F(d), where F represents an arbitrary monadic predicate.

A _join term_ is a term of the form F(d1,d2), where F represents an arbitrary diadic predicate.

• Example 1.3 •

(EQ PART# 10) is a restriction term, whereas:

```
(EQ (TIMES PRICE 10)
    (PLUS PART-PRICE 3)) is a join term.
```

• Definition •

Two terms are said to be _compatible_ if they contain a common domain variable.

• Example 1.4 •

(S# SUPPLIES PART#) and

(PROJECT IS-IN PLOC) are not compatible, whereas

(S# SUPPLIES PART#) and

(PART# IS-USED-IN PROJECT#) are.

4.1.C WFFs

The well formed formulae(WFFs) of the relational calculus are defined as follows:

1. Any term is a WFF.
2. If Ψ is a WFF, then so is $\neg\Psi$.
3. If $\Psi1$ and $\Psi2$ are WFFs, so are $(\Psi1 \& \Psi2)$ and $(\Psi1 \vee \Psi2)$.
4. If Ψ is a WFF in which r occurs as a free variable, then $\exists r(\Psi)$ and $\forall r(\Psi)$ are WFFs.

5. No other formulae are WFFs.

4.1.D Q-Expressions

■ Definition ■

A WFF Q in the relational calculus is a Q-expression if:

1. Q contains no quantifiers.

2. Q contains no simple terms which are negated.

3. Q is in dnf, with each conjunction being of the form (ψ & ϕ), where:

   a. ψ contains only simple and relational terms, and ϕ is either null or it contains only restriction and join terms.

   b. If ψ contains more than one term, then each term in ψ is compatible with some other term in ψ.

   c. If a term T in ψ is negated, then ψ contains a non-negated term which is compatible with T.

4. There is a term in each conjunct of Q which is compatible with some other term in a different conjunct of Q.

• Example 1.5 •

The following are all Q-expressions:

```
(SNAME SUPPLIES PART#)

(SNAME SUPPLIES PART#) & (EQ PART# 10)

(((SNAME SUPPLIES PART#) & (EQ PART# 10)) v
 ((PART# USED-IN PROJECT#) & (EQ PROJECT# 4))).

(SNAME & ¬(SNAME IS-IN SLOC) & (EQ SLOC 'VANCOUVER))

(S# SUPPLIED PART#) v (PART# IS-USED-IN PROJECT#)
```

• Example 1.6 •

<u>None</u> of the following are Q-expressions:

```
¬SNAME
(EQ PART# 10)
SNAME & (PART# IS-USED-IN PROJECT#)
¬SNAME & ¬(SNAME SUPPLIES PART#)
(SNAME SUPPLIES PART#) V (EQ PART# 10)
```

<u>4.1.E Range Formulae</u>

▪ Definition ▪

Ψ is a range formula over domain d if:

1. Ψ is a quantifier free Q-expression.
2. Ψ contains at least one relational term which has d as a free variable.
3. Each conjunct in Ψ has d as a simple term.

• Example 1.7 •

(S# SUPPLIES PART#) & (EQ PART# 5)

is a range formula over S#.

<u>4.1.F Range Coupled Quantifiers</u>

▪ Definition ▪

Let Φ be a WFF having d as a free variable, and Ψ be a range formula over d. Then $\exists\Psi$ and $\forall\Psi$ are called range coupled quantifiers over d, and are defined by the equations:

$\exists\Psi(\Phi) = \exists d(\Psi \;\&\; \Phi)$

$\forall\psi(\phi) = \forall d(\neg\psi \vee \phi)$.

$\exists\psi(\phi)$ and $\forall\psi(\phi)$ are also WFF.

• Example 1.8 •

∀(PART#), and

∀(PART# & (PART# IS-SUPPLIED-BY S#) & (GREATERP S# 10)) are both range coupled quantifiers.

4.1.G Target List

▪ Definition ▪

A target list T is a sequence T=t1,t2, ... ,tk cf domain variables.

4.1.H Queries

We are now in a position to define the queries of the new relational calculus.

▪ Definition ▪

A WFF in the relational calculus is a query if it is a WFF of the form:

T : W , where

1. T is a target list.
2. W is a WFF in prenex normal form.
3. All quantifiers in W are range coupled.
4. The matrix of W is a Q-expression.
5. There are no range coupled quantifiers over any element ti of T.
6. Each domain variable in T is also in each disjunct of W.

• Example 1.9 •

The following are sample queries in the relational calculus.

1. List the names of the parts that supplier number 1 supplies.

```
PART-NAME : (S# SUPPLIES PART-NAME) & (EQ S# 1)
```

2. List the projects that supplier number 1 supplies.

```
PROJECT-NAME : (S# SUPPLIES-TO PROJECT-NAME) & (EQ S# 1)
```

3. Which projects use part 5?

```
PROJECT-NAME : (PROJECT-NAME USES PART#) & (EQ PART# 5)
```

4. Which suppliers supply all suppliers?

```
SNAME : (∀ PROJECT-NAME) (SNAME SUPPLIES PROJECT-NAME)
```

5. Which suppliers have more than 10 units of part 12?

```
SNAME : (SNAME HAS QOH OF-TYPE PART#) & (GREATERP QOH 9)
        & (EQ PART# 12)
```

6. Which suppliers supply all parts that cost more than 5 dollars?

```
SNAME : (∀ PART# & (PART# COSTS PRICE) & (GREATERP PRICE 5))
        & (SNAME SUPPLIES PART#)
```

A more complete set of sample queries and their responses can be found in Appendix 4.

¤ 4.2. Explanation of the Queries ¤

One tends to think of the data base in terms of the domains involved (eg. SUPPLIERS and PARTS). Queries in the new relational calculus allow queries to be formulated in terms of these domains, rather than in terms of the relations.

The terms of the language reference domains. Simple terms such as PART# are domain names. Relational terms such as (S# SUPPLIES PART#) exhibit relationships among domains. Restriction terms, such as (EQ PART# 10) restrict the values of domains, and join terms are used to indicate when terms with different names are really the same. For example, (EQ PARTA PARTB) is a join term. The terms in the relational calculus can be combined to produce meaningful queries.

Using these terms, one constructs the WFFs of the relational calculus. The WFFs, however, are far too powerful to be of use since they can be used to define meaningless queries. For example, the formulae of example 1.6 are WFFs, yet they do not define meaningful queries. The query language must therefore be a restricted subset of the WFFs.

The restriction process involves three stages. Firstly, the Q-expressions are defined. These expressions will eventually form the body of the query. They define a class of WFFs which express a valid query (and thus define a valid relation) and are in a form which the new reduction algorithm can process. Q-expressions may not contain quantifiers.

Secondly, the range formulae are defined. This is a set of WFFs which are only capable of defining a subset of the values of a particular domain. Since the query language deals with quantifiers which express domains or specific subsets thereof, this set of WFFs had to be isolated.

A range coupled quantifier is now defined to be a WFF whose range is a range formula.

Finally, users must be able to specify the domains which they want retrieved. This they do through the target list.

Having defined the valid quantifiers Q, the valid query bodies B and target lists T, the queries can be defined to be a list of the form:

T : W

where all the quantifiers in W are of the form Q, the matrix of W is a valid body B, and each element of T is in W.

4.2.A Comments on the Relational Calculus

If the relational calculus were to be used as a top level query language, then two changes would be desirable.

Firstly, it would be nice to say:

(SNAME SUPPLIES 10) , rather than

(SNAME SUPPLIES PART#) & (EQ PART# 10)

This would require the system to determine that 10 was a PART# and not, say, a PROJECT#. In cases such as:

```
    (WIDGETS ARE-USED-IN VANCOUVER) ,
```

the system must know that WIDGETS are part names, and that VANCOUVER is being used as the location of a project and not as the location of a supplier. The semantic model to be proposed in Chapter V would be of use for disambiguating information of this type.

Secondly, one would like the restriction that each variable in T must also be in W to be removed. Thus, queries such as:

```
    (SNAME SLOC : (SNAME SUPPLIES PART#))
```

would be valid. Currently, this query is expressed as:

```
    (SNAME SLOC : (SNAME SUPPLIES PART#) & (SNAME IS-IN SLOC))
```

The problem with handling queries of this type can not be discussed until the mechanism by which queries are handled is understood. A solution to this problem is presented in Chapter V.

□ <u>4.3. Use with Natural Language</u> □

The overriding goal in the design of this system is to produce a framework which is suitable for direct use in a natural language environment. In particular, the query language must be structured so that the semantics of the natural language system can produce queries in this language. Since the relational calculus of Codd contains direct reference to the data base, and since the queries contain so much information about how the retrieval is to be done, the relational calculus is not suitable for use in this context. A data base independent language and a system which can decide how the retrieval precess is to be done are necessary.

WALT[16], a system for handling natural language interrogations, has been implemented at the University of British Columbia. This system is modelled after the LSNLIS system by Woods[34]. The program attempts to retrieve information about LISP programs as a result of natural language queries.

WALT makes use of an Augmented Transition Network grammar[35] in order to parse the sentence and produce a linguistic deep structure. The semantic component then uses the parse tree to build an interpretation of the sentence. This type of semantics is based on the procedural semantics of Woods[33].

The semantic construct produced is a FOR statement whose form is:

```
    (FOR QUANT X CLASS R(X) P(X))
```

where QUANT is a quantifier, X is the variable being quantified, CLASS is the name of a set over which the quantification is to range, R(X) is a restriction on the range of quantification, and P(X) is the proposition which represents an action to be taken.

• Example 3.1 •

In WALT, the query "List the functions which the routine MATCH calls." would have the interpretation:

```
(FOR EACH X (FUNCTION) (CALLS 'MATCH X) (PRINT X)) .
```

The query language proposed in this chapter contains the same information as does the FOR statement of WALT. Only the syntax used to express the information is different.

• Example 3.2 •

The relational calculus query:

```
SNAME : (SNAME SUPPLIES PART#) & (EQ PART# 10)
```

is equivalent to:

```
(FOR EACH X (SNAME) TRUE
   (FOR THE Y (PART#) (EQ PART# 10)
      (AND (SUPPLIES X Y)
           (PRINT X)) ))
```

It is hoped that the approach presented in this thesis will soon be extended to encompass natural language queries.

V. A NEW FRAMEWORK FOR RELATIONAL SYSTEMS

This chapter presents a new framework for interacting with relational systems. Using this framework, it is possible to reduce queries in a new relational calculus to a sequence of semantically equivalent operations in the relational algebra. This involves determining which relations are relevant to the query, and how the retrieval should be done. Previously, these tasks were assigned to the user. The relational calculus of Codd is <u>not</u> used as an intermediate language.

The chapter begins by describing the basic methodology which is employed to respond to a query. Following this is a description of the main components of the new framework - the semantic model, and the theorem prover. These are discussed in detail, and a general result which shows when it is possible to prove that two or three domains are related by some arbitrary relation is presented.

The chapter concludes with a description of a reduction algorithm which makes use of the semantic model and the theorem prover to respond to queries in the relational calculus.

The approach which has been taken is somewhat interesting in itself. It makes use of standard Artificial Intelligence(AI) techniques in order to solve a problem in fact retrieval. Instead of representing information using tables or graphs, a general semantic model is used. This concept, which is basic to most work in AI, allows the most pertinent information about the system to be explicitly represented. The meaning of the relations are represented in this way. From this basic information, other information is deduced by a theorem prover. In fact, queries in the system are formulated as a series of theorems to be proven true. The steps of these proofs show how the retrieval can be done.

This framework is both incremental and flexible. New information can be constantly added to the semantic model, and its effects will be automatically taken into account. The addition of new relations is also a trivial task.

¤ 5.1. Reduction of Queries - An Overview ¤

This section outlines the basic approach to handling queries, and gives a brief description of Micro-Planner, a language which is used in the reduction process.

The basic approach to answering queries is as follows. Firstly, the user enters a query in the language described in Chapter IV.

• Example 1.1 •

If one wanted a list of all suppliers who supply all parts, one would say:

```
SNAME : (∀ PART#) (SNAME SUPPLIES PART#) .
```

This query is then formulated as a series of theorems to be proved. In Example 1.1, an attempt would be made to prove:

(SNAME SUPPLIES PART#) .

If this is possible, then the system knows that there is indeed information in the data base which allows us to conclude that some suppliers supply part numbers. As a side effect, the programs which perform this proof create instructions which show how the relation which contains this information can be created. In the above example, the relation:

SUPPLIES (SNAME PART#)

would be created. Quantifiers and simple terms in the query are treated in a similar manner. These relations are then restricted and joined according to the restriction and join terms in the query, and a response relation is derived.

5.1.A The Use of Micro-Planner

The reduction of a query is based upon the ability to prove that the simple and relational terms of the query are true, using the elements of the semantic model as the axioms of the system. Section 5 outlines this procedure in detail.

In order to accomplish this task, the language *Micro-Planner*[18] is used. Micro-Planner is a language which is oriented towards the accomplishment of goals, which in turn are broken into a series of sub-goals. It provides a back-up mechanism, so that if one possible way of accomplishing the goal is tried and fails, then another possibility will be tried etc...

The following traditional example of deduction will illustrate some elementary features of Micro-Planner.

• Example 1.2 •

If we know that Turing is a human, and all humans are fallible, then Turing is fallible.

In Micro-Planner, this is expressed by saying:

```
(THASSERT (HUMAN TURING))

(THCONSE (X) (FALLIBLE $?X)
     (THGOAL (HUMAN $?X)) )
```

THASSERT and THGOAL can be abbreviated to $A and $G respectively. The proof would be generated by evaluating the goal:

```
(THGOAL (FALLIBLE TURING) $T)
```

From this example, several points should be observed. First, information is stored in Micro-Planner in one of two ways - as ASSERTIONS, or as THEOREMS. Simple facts, such as Turing is a human, are represented by assertions, whereas more complicated facts which may involve quantification and logical connectives are expressed as theorems. In the above example, a THCONSE (consequence) theorem is shown. This theorem states that a consequence of X being a human is that X is fallible.

Micro-Planner, being a programming language, provides a set of functions which can be used to define theorems and goals. For example:

```
(THFIND ALL $?X (X) ($G (FALLIBLE $?X) $T)) )
```

would return a list of all items which are fallible. These items need not explicitly be stated as being fallible, but can be items which are provably fallible using the current set of theorems and assertions.

As well as THCONSE theorems, Micro-Planner provides THANTE (antecedent) theorems.

• Example 1.3 •

```
(THANTE T (X Y) (LIKES $?X $?Y)
    (THASSERT (HUMAN $?X)) )
```

In this example, the theorem T says that if an assertion is made about X liking Y, then we should immediately assert that X is human. These theorems decrease the amount of explicit information which is necessary at any one time.

The following chapters will reveal how the facilities of Micro-Planner are used to form a framework for relational systems.

◻ 5.2. The Semantic Model ◻

Traditional relational systems have claimed a high degree of user-data independence. This is true only in that the users need not be aware of how their relations are _physically_ stored on a storage device. Unforunately, they must still be aware of how the data is organized. In particular, they must know which relations exist, and what information each contains. In order to overcome this, the concept of a semantic model is introduced. The semantic model is the most vital component of the system.

The semantic model serves as an interface between the system and the data base, and is used to determine the relations which are relevant to a request. The process which reduces queries refers only to this model, and never to the data base itself. Just as the use of the relational algebra allows the user to be independent of the physical representation of his relations, the semantic model allows him to be independent of the organization of the relations.

The basic function of the semantic model is to describe the information conveyed by each relation in the data base. That is, it describes the meanings of the relations. Without this information, it would be impossible to tell which relations contain information about an event X, and which do not. The model also contains real world knowledge which describes the current state of the environment which the data base represents. Information describing various properties which the relations may or may not possess and information which is useful for optimizing the retrieval is also contained in this model.

5.2.A The Meaning of Relations

Consider the relation:

R1 (S# PART# PROJECT#)

where a tuple (X Y Z) is in R if supplier X supplies part Y to project Z. Exactly what information does this relation convey? It tells us that:

1. S# supplies PART#
2. PART# is used in PROJECT#
3. PROJECT# uses PART#
4. S# supplies PROJECT#
5. PART# is supplied by S#
6. PROJECT# is supplied by supplier S#, and

7. S# supplies PART# to PROJECT#

If queries are to be expressed in a language which does not directly reference the relations, then this information will be necessary to identify the relations which are applicable to a request. Therefore, information of this type must be included in the semantic model. The entries in the semantic model for the above relation would be:

```
($A (S# SUPPLIES PART# R1))
($A (PART# IS-USED-IN PROJECT# R1))
($A (PROJECT# USES PART# R1))
($A (S# SUPPLIES-TO PROJECT# R1))
($A (PART# IS-SUPPLIED-BY S# R1))
($A (PROJECT# IS-SUPPLIED-BY-S S# R1))
($A (S# SUPPLIES PART# SUPPLIES-TO PROJECT# R1))
```

It should be noted at this point that no special information <u>about</u> the semantic model need be expressed. For example, Micro-Planner does not need to know what (S# SUPPLIES PART#) means, but will accept it as a primitive fact. Thus, it is as easy to represent information about employees and wages as about suppliers and parts.

Information describing the overall topic with which the relation deals can also be expressed. For example, if R2 (PART# PRICE) were present, we might:

```
($A (R2 CONCERNS CURRENT PARTS))
```

This is especially useful if several relations in the data base have identical domains, but different meaning. For example, if R3 (PART# PRICE) were also included, then we would:

```
($A (R3 CONCERNS OBSOLETE PARTS))
```

5.2.B Properties of Relations

In the process of determining which relations in the data base are relevant to a given request, it is often necessary to check for specific properties which a relation may cr may not possess. Therefore, the semantic model also contains information showing the properties each relation possesses. In this framework, the properties of importance are T-TRANS and DETERMINES (see section 3.B).

• Example 2.1 •

In the relation R4 (S# SNAME), it is likely that the supplier number uniquely determines the supplier name and vice versa. This would be represented by:

```
($A (S# DETERMINES SNAME R4))
($A (SNAME DETERMINES S# R4))
```

It also happens that S# and SNAME are T-TRANS in R4. This would be represented by either:

```
($A (S# AND SNAME ARE T-TRANS IN R4)) or
($A (R4 IS T-TRANS))
```

5.2.C Optimizing Information

The semantic model may also contain information which can be used by the system in order to lessen the time required to retrieve the data for a request.

The system which is implemented shows one possible use of this facility. Consider a query which requires the system to create a relation which enumerates the elements of a domain d. To do this, one would project _each_ relation in the data base on d, and then take the union of these results. This produces a relation which is guaranteed to contain all the values of a domain which are present in the data base.

If, however, a single relation enumerates the values of d, then this process is unnecessary. All that is needed is a projection of this one relation. Even if two relations together enumerate d, then time can be saved if the system is made aware of this fact. Therefore, one can add information of the form:

```
    ($A (R1 ENUMERATES S#))
    ($A (R1 PARTIALLY-ENUMERATES S#))
```

This is especially useful when quantifiers are being processed, since quantification is always over a domain or a restricted subset of that domain.

5.2.D Active and Inactive Data

The semantic model also contains information that reflects the current state of the environment which the data base describes. This we refer to as "real world" knowledge.

• Example 2.2 •

(VANCOUVER SUPPLIERS ARE ON STRIKE)

is a typical example.

In order to handle real world information, the concepts of the currently _active_ and _inactive_ portions of the data base are introduced. The active information is that which can be used in responding to a query. The inactive data is data which is present in the data base, but should temporarily be ignored.

The system will never make use of inactive data when responding to a query.

The effect of real world knowledge is to temporarily alter the part of the data base which is currently considered active. There are two cases which must be dealt with. Firstly, whole relations can be inactive. Alternatively, the tuples in certain relations which satisfy some criterion can be considered inactive.

In order to indicate that a relation is currently inactive, one simply adds an assertion of the form:

```
    ($A ($?REL IS INACTIVE))
```

to the semantic model, where $?REL is a variable whose value is the name of the relation which is inactive.

The inactivation of tuples within a relation is accomplished in a somewhat more complicated manner. It is most suitable to restrict tuples by showing the properties that each domain in the tuple must satisfy. Therefore, one can add information of the type:

```
    ($A (REST RESTRICTS DOMAIN TO LPRED))
```

where REST is the name of the restriction, DOMAIN is the domain to be restricted, and LPRED is an arbitrary LISP predicate whose only unbound variable is DOMAIN.

• Example 2.3 •

The semantic model would contain the assertion:

($A (REST#1 RESTRICTS SLOC TO (NOT (EQ SLOC 'VANCOUVER))))

if the tuples in some relation are to be restricted to the case where the location of the supplier is not VANCOUVER.

This allows restrictions to be specified _independently_ of the relations which are restricted.

In order to make use of this information, the semantic model can also contain assertions of the form:

($A (PRED IN RELATION IS-RESTRICTED-TO REST#))

where PRED is the name of a diadic predicate in the query language, RELATION is the name of a relation, and REST# is the name of the restriction. For example, we might have:

(SUPPLIES IN R3 IS-RESTRICTED-TO REST#1).

This says that if the predicate SUPPLIES is seen in a relational term of a query and if R3 is being used in the reduction of the query, then only those tuples of R3 which satisfy REST#1 should be used. Notice that this is more general than restricting whole relations to specific cases.

The way that these assertions are used to represent real world knowledge is discussed in section 5.6.

5.2.E Generality of the Semantic Model

It is felt that a semantic model is necessary to process queries in any language which does not make direct reference to the data base. This work shows the type of information which a typical semantic model might contain. The bulk of this information describes the _meaning_ of the relations. For example,

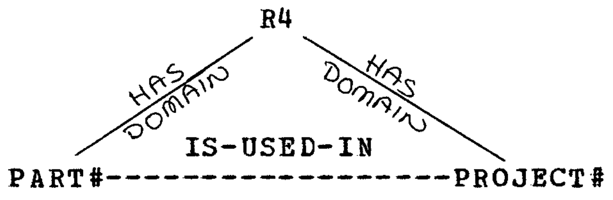
($A (PART# IS-USED-IN PROJECT# R5)).

The semantic model is represented by a set of Micro-Planner assertions. However, the representation is _not_ the important feature; the model is proposed as a _general_ concept, and the manner in which it is represented can be determined by the application in which it is being used.

In this thesis, the framework is formulated in such a way that it is useful in the context of a natural language system. For this application, using Micro-Planner to describe the semantic model is ideal. If one were attempting to design a commercially usable retrieval system, however, it is unlikely that LISP and Micro-Planner would be used. They make implementing the system easy, but are not exceptionally efficient. In this application, the semantic model could be represented using a more "conventional" data structure such as a network. The nodes of the network could be the domains in the data base, with the arcs being labelled with the predicate that relates the nodes. For example, the assertion:

(PART# IS-USED-IN PROJECT# R5)

would be represented by:



The results which are presented in the following chapters are applicable regardless of how the semantic model is represented, and are not dependent upon the use of Micro-Planner.

¤ 5.3. Proving Relational Terms ¤

Queries in the new relational calculus contain nc direct reference to the data base. Therefore, the system which handles them must be capable of deciding which relations are relevant to the request, and how the relations should be manipulated in order to produce the correct response. This section discusses how this is accomplished.

In any query, the relational terms are of primary importance. The users view these as the mechanism through which they can express the properties of the data they desire. To the system, however, they represent new relations which must be created.

• Example 3.1 •

Consider the query:

(SNAME : (SNAME SUPPLIES PROJECT#) & (EQ PROJECT# 17))

which asks for all suppliers who supply project 17. In order to answer this, the relation:

SUPPLIES (SNAME PROJECT#) must first be created.

When processing a query, the relation which corresponds to each relational term and each simple term must be created. (Restriction and join terms do not require new relations to be created.) The queries, however, contain no information as to how this should be done. It is up to the system to determine how the information can be extracted from the data base.

In order to create the relation corresponding to some relational term T, the following approach is taken. First, T is formulated as a theorem, and an attempt is made to prove it using the semantics of the data base as the axioms. If the theorem can be proven true, then the proof will show how the relation can be created. If it is false, then we can conclude that the data base does not contain sufficient information to answer the query.

The remainder of this section is devoted to discussing when, in general, a term can be proven true. The next section discusses the implementation.

<u>5.3.A Explanation of the Problem</u>

In the following sections, we represent relational terms of the form (X R Y) by the standard set-theoretic notation xRy.

Consider relational terms of the form xRy. If a single relation in the data base contains the information requested in the relational term, then the axiom xRy will appear in the semantic model, and the proof will be trivial. It will not always be the case that a single relation contains all the desired information. Several relations will often be needed.

• Example 3.2 •

To form the relation defined by the term in example 3.1 above, relations R3 and R5 must be used.

Consider the problem of proving xRy, when two relations are necessary. If one relation says xRz, and another says that zR'y, then can we conclude that xRy, or xR'y, or even xR"y? Consider the following examples which all attempt to prove a term of the form xRz by showing xRy and yR'z.

3.2.1. If we know (SNAME IS-NUMBERED S#) and (S# SUPPLIES PROJECT), then we can conclude that (SNAME SUPPLIES PROJECT).

3.2.2. If we know (S# SUPPLIES PART#), and (PART# IS-USED-IN PROJECT#), then we can <u>not</u> conclude that (S# SUPPLIES PROJECT#). We can only conclude that (S# R PROJECT#), where R is some relation whose name happens to be "maybe supplies". If, however, we know that PART# uniquely determines PROJECT#, then we <u>can</u> conclude that

(S# SUPPLIES PROJECT#).

3.2.3. If we know that (A IS-THE-SQUARE-OF B), and (B DOUBLED-IS C), then we cannot conclude that (A IS-THE-SQUARE-OF C) _in any case_, regardless of the fact that B uniquely determines A.

3.2.4. If we know that (SUB-PART# IS-PART-OF PART#), and (PART# HAS-NAME PART-NAME), then we can always conclude that (SUB-PART# IS-PART-OF PART-NAME).

These examples illustrate that the proof of xRy is going to depend upon the _properties_ of the relations involved, not just upon their content.

5.3.B Properties of Relations

There are two properties which relations may possess that influence the way they can be used in creating new relations. These properties are called T-TRANS and UNIQUELY-DETERMINES.

▪ Definition ▪

Two domains a and y in relation R' are said to be T-TRANS (Through Transitive) if, for each xRa,

xRa & aR'y => xRy.

A relation R' is said to be T-TRANS if all domains in R' are pairwise T-TRANS.

• Example 3.3 •

If (X R S#) and (S# R' SNAME) => (X R SNAME), then SNAME and S# are T-TRANS in R'.

• Example 3.4 •

The relation R(S# SNAME PROJECT# PROJECT-NAME) is not T-TRANS, whereas R'(S# SNAME) is.

This property will be used to show that the conclusions in examples 3.2.1 and 3.2.4 are valid, whereas no conclusion can be drawn from 3.2.3, which is _syntactically_ similar to both 3.2.1 and 3.2.4.

▪ Definition ▪

A domain a in R _uniquely determines_ domain b in R if, for any value of a in R, there exists only one value of b.

• Example 3.5 •

R (	S#	PART#)
	1	4
	1	7
	2	8
	3	9

Here, PART# uniquely determines S#. Notice that S# does not uniquely determine PART#. This property will be necessary in the proof of 3.2 above.

5.3.C Proving xRy

If the axiom xRy is not present in the semantic model, then more than one relation will be involved in the proof. Thus, the proof is broken up into two stages. Firstly, an attempt is made to find a domain a such that xR'a. If this succeeds then the proof of aR"y is attempted. If this also succeeds, then we can conclude that a relation R exists between x and y. The validity of this conclusion, however, depends upon certain relationships which may or may not exist between a, x, and y. Example 3.2 illustrated this point. Further, it will most often be the case that the relation R which is desired will be specified, rather than arbitrary. Thus, the property T-TRANS will also need to be taken into account.

Consider first the problem of trying to prove xRy, where R is an arbitrary relation.

Heath[17], in attempting to show that any relation can be reduced to a natural join of relations in third normal form[10], proves the following two results:

1. The relation R(A,B,C), where A determines B, is the join of R'(A,B) and R"(A,C).
2. The relation R(A,B,C), where A determines B and B determines C, is the join of R'(A,B) and R"(B,C).

Using these results, we define the concept of a _valid_ result.

• Definition •

If A determines B in R'(A,B), and neither A nor C determine each other in R"(A,C), then the projection R(B,C) of the relation R(A,B,C), formed by joining R'(A,B) and R"(A,C) is a valid relation.

Similarly, if A determines B in R'(A,B) and B determines C in R"(B,C), then the projection R(A,C) of the relation R(A,B,C) formed by joining R'(A,B) and R"(B,C) is also a valid relation.

We speak of the steps of the proof of a relational term as the path of the proof. Any path which requires the formation of valid relations and valid relations only is said to be a valid path.

• Example 3.6 •

> If we know that A:(PART# IS-SUPPLIED-BY S#), and that B:(PART# IS-USED-IN PROJECT#), then if PART# uniquely determines S# in the relation defined by the relational term A, we can form the valid relation R(S#,PROJECT#), where the name of R is actually "supplies-to".

In any proof, an attempt is made to find a valid path before a non-valid path. Note that a non-valid path is not always undesirable, since even if PART# does not determine S# in example 3.6, the join of the relations defined by the terms A and B produces a new relation whose name is "maybe supplies", rather than "supplies". This concept is extremely valuable, as it guides the proof in a reasonable direction. This point will be illustrated later.

Now consider the problem of trying to prove xRy where R is specified. Given relations R'(X,A) or R'(A,X), and R"(A,Y) or R"(Y,A), then the following table shows the conclusions which can be drawn <u>assuming</u> the new relations are valid. The conclusions marked with asterisks are the ones of interest, since they show when one can prove xRy.

	R' T-TRANS	R" T-TRANS
R'(A,X) R"(A,Y)	R"(X,Y)*	R'(Y,X)
R'(X,A) R"(A,Y)	R"(X,Y)*	R'(X,Y)*
R'(A,X) R"(Y,A)	R"(Y,X)	R'(Y,X)
R'(X,A) R"(Y,A)	R"(Y,X)	R'(X,Y)*

We are now in a position to define the conditions under which xRy can be proven true. xRy is true if either:

1. xRy is an axiom of the system.

2. i. xRa and either aR'y or yR'a.

   ii. y and a are T-TRANS in R'.

   iii. Either a determines x in R or a determines y in R'.

3. i. aRy and either aR'x or xR'a.

   ii. x and a are T-TRANS in R'.

   iii. Either a determines y in R or a determines x in R'.

These are the only conditions which allow us to show that xRy.

• Example 3.7 •

Say one is attempting to prove xRy. If it is known that xRa, then one must simply show that aR'y where y and a are T-TRANS in R', and either a determines x in R or a determines y in R'. This in turn could be done by showing that aR"b and bR"'y, where b and y are T-TRANS in R"', and where b determines a in R" or b determines y in R"'.

Thus, this method allows for proofs of arbitrary length, not just proofs with two steps.

• Example 3.8 •

The term (S# SUPPLIES-TO PROJECT-NAME) can be proven true, since:

1. (S# SUPPLIES-TO PROJECT#)
2. (PROJECT# IS-NAMED PROJECT-NAME)
3. The relation defined by 2 is T-TRANS, and
4. In the relation defined by 2, PROJECT# determines PROJECT-NAME.

5.3.D Proving xRyR'z

There are six conditions under which xRyR'z can be proven true. xRyR'z is true if either:

1. xRyR'z is an axiom of the system.
2. i. aRyR'z and either aR"x or xR"a.
   ii. x and a are T-TRANS in R".
   iii. Either a determines x in R" or a determines (y z) in the relation defined by the term.

3. i. xRaR'z and either aR"y or yR"a.
   ii. y and a are T-TRANS in R".
   iii. Either a determines y in R" or a determines (x z) in the relation defined by the term.
4. i. xRyR'a and either aR"z or zR"a.
   ii. z and a are T-TRANS in R".
   iii. Either a determines z in R" or a determines (x y) in the relation defined by the term.
5. i. xRy and xR'z
   ii. x determines y in the relation defined by (x R y)
6. i. xRy and yR'z
   ii. x determines y in the relation defined by (x R y)
   iii. y determines z in the relation defined by (x R' z)

Notice that in case 5, x is related to y and z, while in case 6 x is related to y and y is related to z. This represents the two interpretations of ternary semantic information. In this framework, it is possible to prove a term of the form xRyR'z, even if the semantics contain only binary information.

• Example 3.9 •

We can prove (SNAME SUPPLIED PART# SUPPLIED-TO PROJECT#), since:

1. (S# SUPPLIED PART# SUPPLIED-TO PROJECT#)
2. (S# IS-CALLED SNAME)
3. S# and SNAME are T-TRANS in the relation defined by 2
4. S# determines SNAME in the relation defined by 2.

5.3.E Usefulness of the Valid Path

The concept of a valid path is crucial to the proof mechanism since it tends to eliminate "garbage" paths. Consider, for example, an attempt to prove that (S# SUPPLIES-TO PROJECT-NAME). The desired proof is:

1. (S# SUPPLIES-TO PROJECT#), and
2. (PROJECT# IS-NAMED PROJECT-NAME).

At first glance there seem to be several other correct but undesirable proofs. For example:

A. (S# SUPPLIES PART#), and

B. (PART# IS-USED-IN PROJECT#), and

C. (PROJECT# IS-NAMED PROJECT-NAME)

would also appear to be correct. We would hope that if the system found this version of the proof before the first version, it would be rejected. This is, in fact, the case. Unless PART# uniquely determines S# in A, steps A and B do not define a valid relation, and therefore, the system will abandon this path and the correct proof will eventually be found. If PART# does determine S# in A, then this proof produces the same response relation as does the first, and is therefore acceptable.

5.3.F Summary

This section has defined the conditions under which the relational terms xRy and xRyR'z can be proven. Above all, it has demonstrated that in order to prove a term such as xRy, it is not sufficient to show that:

xRa & aR'b & bR"c & cR'"y.

The properties of the relations _must_ be taken into account.

□ 5.4. The Processing of Relational Terms □

This section discusses the implementation of the procedures which prove the validity of the relational terms. In the previous section, the conditions under which this could be done were outlined.

Relational terms can be proven true only if the data base contains enough information to create the relation which the term defines. Therefore, the proof is constructed by showing that it is, in fact, possible to create such a relation. The steps of the proof are "remembered", and from these, we can determine how the relation can be constructed.

This section begins by describing the format in which Micro-Planner saves intermediate steps. It also describes the routines which prove that x, xRy, or xRyR'z are true. Included are descriptions of how real world knowledge is handled, and how Micro-Planner saves the relevant parts of the proof.

5.4.A The List Returned by Micro-Planner

In proving a relational term, it is important to know not only that the corresponding relation can be created, but how it can be created. A routine which just says "yes, you can create the relation" is of little use in itself. Therefore, the Micro-Planner procedures keep track of the information they use in the proof, and organize it in a fashion which makes it easy to see how the relation can be created. This information is stored in a list, and given the name #RESULT# since it is the result of a successful proof. The list shows which relations are needed in order to create the relation defined by the relational term, and shows how they should be used to form it.

If the relational term T has n domain variables, then the first n elements of #RESULT# are lists of the form:

(DOMAIN RELS),

where DOMAIN is the name of a domain in T, and RELS is the list of relations which together enumerate the elements of DOMAIN. For example, we might have:

(PART# (R7 R5))

if PART# was to come from relations R5 and R7. The list RELS is not actually used in the current scheme. The last element of #RESULT# is a list which shows how to create the relation. It has the form:

(REL1 D1 REL2 <D2 REL3 ... <DN RELN>>), where RELi is a list of relations, and Di is a domain name.

Using the list #RESULT# the relation defined by the term is created as follows:

1. Create a list T of each domain in the term, and delete the first n elements from #RESULT#.

2. Take the first list of relations from #RESULT#. Call this list R. If R is a list of one element, then go to step 3. Otherwise, find the list L of domains which are common to all relations in R, and project them on these domains. Take the union of the results, and go to step 3.

3. Now do the same for the list of relations which is the third element in #RESULT#.

4. Join the relations created in steps 2 and 3 on the condition that the values of the domain which is specified by the second element of #RESULT# are equal.

5. Replace the first three elements of #RESULT# with the name of the relation from step 4. If #RESULT# contains more than one element, go to step 2. Otherwise, continue with step 6.

6. Project the relation which is the first (and only) element of #RESULT# upon the domains D1,D2 ... Dn which are in the list T. This is the relation defined by the relational term.

• Example 4.1 •

The proof of the relational term (SNAME SUPPLIES PART#) will return the list:

```
((SNAME (R3)) (PART# (R5 R7)) ((R3) S# (R5 R7))).
```

To form this relation, take the union U of the projections of R5 and R7 on (S#,PART#). Now join R3 and U on the common domain S#.

5.4.B Proving X

A simple term is provably true if its domain variable occurs as the domain of any relation in the data base. Otherwise the proof will fail, and the query cannot be answered.

As a side effect, any successful proof will create a list which shows how to obtain the relation which _enumerates_ the values of this domain. Proving a simple term X is accomplished by issuing the goal:

```
($G (ENUMERATE $?X) $T).
```

• Example 4.2 •

($G (ENUMERATE PART#) $T) results in #RESULT# being bound to the list:

```
((PART# (R1 R8)) (R1 R8)),
```

since taking the union of R1 and R8, and projecting this relation on the domain PART# results in a relation which enumerates the parts currently mentioned in the system.

The list showing which relations enumerate the term will be derived in one of two ways. The first possibility is that the semantic model contains an assertion which states that a certain relation enumerates the term. If this is true, the list is simply a list of one element, namely this relation. Secondly, the semantic model could contain several assertions which state that certain relations partially enumerate the term. In this case, the list is a list of all such relations. If neither of these are true, then the semantic model must be examined to see which relations, if any, deal with this domain. If one or more are found, then a list of all these relations is built. Otherwise, the proof fails.

In this last case, only semantic information of the form (D P D) or (D) need be examined, since each domain in each relation will be included in a semantic term of this type.

5.4.C Proving xRy

Section 3 outlined the conditions under which it is valid to conclude that xRy. The implementation of this proof procedure must do more than just check for these conditions. It must also check that each relation it attempts to use is active, check for restrictions upon the tuples it selects, and process any restrictions which are found. It must not only show that the proof can be done, but remember which relaticns were involved, and how they were used. Further, since Micrc-Planner performs its proofs in a depth first manner, simple proofs must always be attempted before complex ones. Thus, the conditions under which a proof is possible cannot simply be stated, but must be expressed so that Micro-Planner will construct the proof with a _minimum_ of wasted effort.

5.4.C.1 Representation of the Goals

Proving a relational term xRy is accomplished by issuing the goal:

```
($G ($?X $?R $?Y $?REL) $?T),
```

where X and Y are domains, R is a diadic predicate, and REL is the name of a virtual relation which, if it were created, would show that xRy.

• Example 4.3 •

A typical goal is:

```
($G (PART# IS-USED-IN PROJECT-NAME $?REL) $T).
```

None of the variables in the goal need be specified. Therefore, the goal:

```
($G (PART# $?R PROJECT-NAME $?REL) $T)
```

is perfectly valid. If this proof is successful, then $?R will contain the name of the relation between PART# and PROJECT-NAME. The possible uses of this feature are discussed in section 6.

5.4.C.2 Basic Proof Strategy

Omitting the conditions that specify when X, Y, and A must be T-TRANS and DETERMINE one another, it is possible to prove (X R Y REL) if we can prove either:

1. (X R A REL1) and either (Y R' A REL2) or (A R' Y REL2)
2. (A R Y REL1) and either (X R' A REL2) or (A R' X REL2).

The program begins by checking to see if (X R A REL1) is true in the semantic model. If it is, then the system looks to see if either (A R' Y REL2) or (Y R' A REL2) are in the semantic model, where R' is now an arbitrary relation. If one of these succeeds, then the proof succeeds. If both fail, then an attempt is made to prove that (A R' Y REL2) (for some arbitrary R') in the same manner that the original proof of (X R A REL1) was attempted. If this fails, then we attempt to prove

(Y R' A REL2).

If this also fails, then the system looks to see if (A R Y REL1) is in the semantic model. If it is, then the system looks in the semantic model for either (X R' A REL2) or (A R' X REL2). Should either of these succeed, then the proof also succeeds. If they both fail, then an attempt is made to prove that one of them is true.

By checking in this manner, the system tends to come up with trivial proofs quickly. If it attempted to prove (X R A REL1) in general _before_ checking to see if (A R Y REL1) is in the data base, proofs could often take an excessive amount of time. Checking the data base first saves a great deal of wasted effort.

If the proof succeeds this far, then the properties of T-TRANS and DETERMINES must be checked. The conditions which must apply are stated in section 3.C of this chapter.

If these properties are both present, then the proof will succeed. Since we now know that it is possible to create a relation which shows that xRy, we add this information to the semantic model. In fact, we create a _virtual relation_ - that is, a relation which is fully semanticized, but not physically present. The relation is given some arbitrary name N, and the assertion:

(X R Y N)

is placed in the semantic model. Further, the _properties_ of N are also added. For example, if A determines X in R, and A determines Y in R', then X determines Y and Y determines X in N. The variable $?REL in the original goal is now bound to N, and the proof procedure "returns".

The reason N is not explicitly created at this point is that the correct proof is seldom generated immediately. Incorrect paths are often taken. Thus, immediate generation of relations would mean that unnecessary relations would often be created. It is considerably more efficient to create the relation only when it is known for sure that it <u>must</u> be done.

5.4.C.3 Limiting Unnecessary Work

There are several facts that the proof mechanism makes use of in order to restrict the amount of unnecessary work which is done. If we are attempting to prove ($?X $?R $?Y $?REL) where $?Y is unbound, then unless the proof succeeds using an axiom in the data base, there is no point trying to prove it via the xRa method, since this will always fail. Likewise, if $?X is unbound, there is no point trying the aRy method.

Secondly, once an assertion in the semantic model has been used as a step in the proof, the same assertion should <u>never</u> be used again. Therefore, it is removed from the semantic model for the duration of the proof. This saves a considerable amount of extra effort, especially if the proof is doomed to fail.

5.4.C.4 Remembering the Proof

In order to create the relation corresponding to the relational term, Micro-Planner keeps track of the relevant part of the proof. This information is kept in the lists #RESULT#, #RESTRICT#, and #EXTRA#. The facts which are relevant are the names of the relations which are used, and the names of the domains not present in the relational term which are used in the proof. In fact, any time a piece of semantic information is used, the name of the relation which contains this information is saved. Further, if the information involves a domain not referenced in the original term, the name of the domain is also saved.

Consider, as an example, a proof of xRy which goes as follows:

```
    (X R A REL1)
    (A R' Y REL2)
```

Then Micro-Planner will save REL1, A, and REL2 in #RESULT#, since the relation which defines xRy can be created by joining REL1 to REL2 on the common domain A. The value of #RESULT# will be:

```
    ((X REL1) (Y REL2) (REL1 A REL2)).
```

In general, #RESULT# is constructed as follows. Any time that (X R A RELN) succeeds, the list (RELN A) is added to #RESULT#. When (A R' Y RELX) or (Y R' A RELX) succeeds, RELX is also added to the end. If both (A R' Y RELX) and (Y R' A RELX) fail, then Micro-Planner backup automatically erases the list

(RELN A) from #RESULT#, and a new choice of A is tried.

The same basic method is applied when (A R Y RELN) succeeds. Care is always taken so that when the list #RESULT# is processed from left to right, all the domains necessary for the joins will be present.

Any time a proof of a term (X R Y RELN) is made by finding the axiom xRy in the semantic model, then all relations which show that xRy are found. This is easily done with a THFIND ALL statement. The reason for doing this is that it is not sufficient to examine only one relation which says that xRy - <u>all</u> relations which contain this information must be examined. For example, if we attempt to prove that:

(S# SUPPLIES PART# $?REL),

then since both R5 and R7 contain this information, they must both be used in constructing the relation defined by the term. This is why #RESULT# is composed of lists of relation names, rather than single relation names.

• Example 4.4 •

Consider the term (SNAME SUPPLIES PART#). In order to prove this term, we would first check to see if (SNAME SUPPLIES $?A $?REL) is an axiom. This goal will succeed, with $?A being bound to S#, and $?REL being bound to R5. A search for all relations containing the information (S# SUPPLIES PART#) produces the list (R5 R7). #RESULT# is now set to ((R5 R7) S#). The next goal is to see if (SNAME $?R' S# $?REL2) is an axiom. This it is, and $?R' is bound to IS-NUMBERED, and $?REL2 is bound to R3. Since this is the only relation which relates SNAME and S#, #RESULT# is set to ((R5 R7) S# (R3)), which shows how to create the relation defined by the original term. Since SNAME determines S#, and since SNAME and S# are T-TRANS in R3, the proof succeeds.

<u>5.4.C.5 The Effect of Real World Knowledge</u>

In the description of the basic proof strategy, the influence of real world restrictions was ignored. Real world restrictions can take on one of two forms. They can either restrict whole relations, or specific tuples within relations. Each time a new relation is used in the proof, a check is made to see if it is currently active. If so, then the relation can be used. If not, then the goal which just succeeded is forced to fail.

• Example 4.5 •

If we attempt to prove (S# SUPPLIES PART# $?REL), and $?REL is returned bound to R5, then R5 must be an active relation. If not, we try for a new relation. This also implies that when the list of relations which show (S# SUPPLIES PART#) is formed, only active relations should be included.

Each time that Micro-Planner attempts to prove a term of the form (X R Y RELN), a check is made to see if R is restricted in RELN. For example, in (S# SUPPLIES PART# R5), we would check to see if SUPPLIES is restricted in R5. If a supplier were on strike, then then this would be true.

If a restriction is found, then the restriction number is noted. Using this number, it is possible to find out which domain is being restricted, and what the restriction is. It is possible, however, that the domain being restricted is in a relation which is not being used in the construction of the relation defined by the relational term. For example, if the term T were (S# SUPPLIES PART#), then the restriction could be on SLOC which appears only in relation R3. Nowhere is R3 used in the construction of the relation defined by T. For this reason, if the domain D being restricted is neither X nor Y (the domains in the term), the system attempts to relate either X to D, or Y to D. This results in a method whereby the relation being created can be joined to a relation which contains the domain being restricted.

Any time a restriction is found, the following four pieces of information are placed in the front of #RESTRICT#:

1. The restriction.
2. The domain being restricted.
3. The relation R which contains this domain.
4. The domain JD which can be used to join R to the original relation.

When processing of the relational term is complete, each domain which is restricted is compared to #RESULT#. If #RESULT# contains the domain being restricted, then the restriction on the domain is simply added to the end of the query. It can now be treated as if the user had specified it. If the domain is not used in the construction of the relation, then we must join the relation defined by the term with the relation R on the common domain JD. Thus, the list (JD R) is added to the end of #RESULT#.

This approach presents a general mechanism for relating domains to the query when the user has not specified how the domain is to be related.

Examples of this can be found in Appendix 5 and Appendix 6.

5.4.D Proving xRyR'z

A procedure for proving relational terms of the form xRyR'z has also been implemented. It utilizes the result given in section 3.D, which shows when it is possible to prove relational terms of this type. This procedure operates in a manner which is very similar to that described for its binary counterpart.

For a complete listing of the routines described in this section, please see Appendix 9.

□ 5.5. The Reduction Algorithm □

In Chapter IV, a new query language which possesses many desirable properties was presented. The preceeding sections of this chapter proposed a semantic model that is used to determine which relations are relevant to a request. This model describes the meaning of the relations as well as the properties they possess, and contains certain other information which affects the applicability of the relations to the queries. This section also presented a method whereby Micro-Planner could be used to construct a list showing how to create the relations which corresponds to a relational term. Further, it was shown how real world information could be used in this process.

This section utilizes the results presented in the previous sections to define a process which _reduces_ a query in the relational calculus to a semantically equivalent sequence of operations in the relational algebra. This process is referred to as the _reduction algorithm._

5.5.A The Reduction Algorithm - An Overview

This section presents a reduction algorithm for reducing queries in the new relational calculus. The basic approach is to create the relation defined by each conjunct of the dnf query, and take the union of the common domains of these relations. This result is projected upon the domains of the target list. The relation defined by a conjunct is created by taking the intersection of the relations defined by the relational terms in the conjunct.

Notice that the relational terms can be processed independently of each other. Further, the quantifiers can be processed independently of the rest of the query. The algorithm proceeds as follows:

STEP 1. This step is referred to as the DISJUNCT processor. Since the query is in dnf, pass each element C of the disjunction to STEP 2, the CONJUNCT processor. Each time this is done, a relation will be returned. When all such relations have been returned, find the domains D which are common to them all, and project each relation on D. Take the union of these relations, and call the result R. Go to step 7.

STEP 2. This step is referred to as the CONJUNCT processor. Go through C, and attempt to prove each relational term therein. If the term can be proven true, then create the relation defined by the term. Otherwise, the query fails. For each relation, remember if the relational term which defined it is negated. Call the set of relations defined by the relational terms of C RC.

STEP 3. Go through each restriction term in C, and apply the restrictions to each relation in RC which contains a domain mentioned in the restriction.

STEP 4. Pass through the join terms J of C, and join all relations in RC using J as the predicate of the join. Call the set of relations which were not used in this process and the relation which this process created RCJ.

STEP 5. Create a list L of all the relations in RCJ whose defining relational terms were not negated. Take the first relation from L, and remove it from the list. Join this relation with some joinable relation R in L under the criterion that the common domains are equal. Remove R from the list, and start again at the front of L. This produces a relation RP.

<u>STEP 6.</u> Find a relation RN whose defining term was negated. Create a list L of the domains which are common to RN and RP. Form a new relation R which is the difference between RP projected on L and RN projected on L. Set RP to be the join of RP with R. This ,in effect, takes the difference between RP and RN. Continue this process for each relation RN whose defining term was negated. This results in a relation RC, the relation defined by the conjunction. Return to step 1.

<u>STEP 7.</u> This step takes the effect of the quantifiers into account. Determine the relation RQ defined by the right most quantifier. This is done by passing the quantifier to the DISJUNCTION processor. If the quantifier is $\forall$, then create the relation defined by the range of the quantifier. Then divide R by RQ. If the quantifier is $\exists$, then join R with RQ and project the result on all domains <u>except</u> the domain which RQ contains. Do the same for each quantifier in the query. The result is a relation R.

<u>STEP 8.</u> This step creates the response relation. This is done by projecting the relation R on the domains given in the target list. Print this relation.

Although queries are assumed to be in dnf, most queries will consist of a single disjunct. Therefore, the reduction algorithm checks the second element of the query (not including the target list or quantifiers). If it is not an V symbol, then the query is assumed to consist of a set of conjoined terms.

5.5.B Justification of the Reduction Algorithm

5.5.B.1 Treatment of Conjunctions and Disjunctions

Begin by considering the treatment of relational terms in conjuncts. If two terms A and B have a common domain variable D, then the reduction algorithm states that the relation defined is the join of the relation defined by A with the relation defined by B under the criterion that the values of D are equal. For example, if a conjunct contains terms:

(PROJECT# USED PART#) & (S# SUPPLIED PART#),

then the relation desired is one that shows the parts which were used by the project _and_ supplied by the supplier. Joining the relations defined by these two terms produces this relation, since any part number in the first relation which is not in the second will not appear in the join and vice versa.

In processing a disjunction, however, the relation defined is the union of certain projections of the relation defined by the elements in the disjunction. For example, if the query contains:

(SUPPLIER SUPPLIES PART#) v (PROJECT# USES PART#)

taking the union of the projection of the first relation on PART# with the projection of the second relation on PART#

produces the corresponding relation.

Notice that the query language has been defined so that the relational terms are compatible, and the join and unicn process can _always_ be carried out.

5.5.B.2 Treatment of Negated Terms

In processing a conjunct, the relations defined by the non-negated terms are each created. These relations are then joined, resulting in a single relation RP. The effect of the negated terms must now be taken into account. Since each negated term defines a set of values for some domain D which must _not_ be contained in the response relation, we attempt to eliminate any tuple in R whose value of D is equal to a value of D in RN. The easiest way to do this is to take the difference between the projection of RP on D and the projection of RN on D, (which results in a list of acceptable values for the domain), then join this relation with RP.

• Example 5.1 •

If the query is:

(S# : (S# SUPPLIES PART#) & ¬(PART# IS-USED-IN PROJECT#))

then we create the relations R1 (S# PART#), and R2 (PART# PROJECT#). Taking the difference between R1[PART#] and R2[PART#] produces a relation R3 which enumerates all allowable parts. Joining R3 with R1 using the predicate (EQ (R1 2) (R2 1)) results in a relation R4 (S# PART#), where the supplier supplies a part which is not used in the project.

5.5.B.3 Treatment of Quantifiers

The query language defines range coupled quantifiers in such a way that quantification can only be over a single domain , although the allowable values for the domain can easily be specified. Therefore, each quantifier defines a relation which has one domain.

Since quantifiers are in dnf, the relation they define can be created by calling the disjunction processor. This processor returns the name of the relation over which the quantification is to range.

• Example 5.2 •

Calling the disjunction processor with the quantifier:

(PART# & (PART# IS-USED-IN PROJECT#) & (EQ PROJECT# 7))

results in a relation whose tuples are the parts which are used in project 7.

If the quantifier is universal, then the relation R defined by the query must be divided by the relation RQ defined by the quantifier, since only the tuples in R which are "true" for each value of RQ are desired.

• Example 5.3 •

If the relation R defined by the query is:

```
R    (      S#        PART#    )
            1          1
            1          2
            3          1
```

and the relation RQ defined by the quantifer (∀ PART#) is:

RQ	(	PART#	)
		1	
		2	

Then the quotient R(PART#/PART#)RQ is:

RP	(	S#	)
		1	

If the quantifier is existential, then the relation R defined by the query must be joined with RQ, and the result projected on all domains but the domain D of RQ. This is done so that the only tuples of R are those for which there exists a value of D in R which also exists in a tuple of RQ.

• Example 5.4 •

If the relation R defined by the matrix of the query is:

R	(	S#	PART#	)
		1	7	
		2	6	
		3	8	
		4	4	

and the relation RQ defined by the quantifier:

∃(PART# & (S# SUPPLIES PART#)) is:

RQ	(	PART#	)
		7	
		8	

then the join R with RQ yields:

RP	(	S#	PART#	PART#	)
		1	7	7	
		3	8	8	

Projecting this relation on S# produces:

RR	(	S#	)
		1	
		3	

This relation lists all suppliers for which there exists a part with the desired condition.

5.5.C The Allowable Variables in Target Lists

The current version of the query language specifies that any variable occurring in a target list must also appear in each disjunct of the query body. Thus, queries such as:

```
    (SNAME SLOC : (SNAME SUPPLIES PART#))
```

are not acceptable. The reason for this restriction is that the domain SLOC may never be used in the construction of the relation defined by the query. Therefore, the system, after creating the initial response relation, would have to try to relate SNAME to SLOC, and thus create a new response relation. This could be done in a manner similar to the way that real world restrictions which reference domains not used in the query are handled. However, with queries such as:

```
    (SNAME SLOC : (PART# IS-USED-IN PROJECT#))
```

it is still unclear what should be done. Should SNAME be related to PART#, or PROJECT#? Forcing the user to make such relations _explicit_ eliminates the need to make these decisions.

5.5.D Why the Relational Calculus of Codd is Bypassed

One way of reducing queries in this language would be to use Micro-Planner to identify the relations which are relevant to the request, then translate the query into an expression in the relational calculus proposed by Codd. Since it is already known how queries in this language can be reduced, and since this reduction algorithm has already been implemented, it would appear to be a logical approach.

However, the process of translating queries in the new relational calculus to the old relational calculus is extremely difficult. Firstly, one is translating from a representation which uses domain variables to one which uses tuple variables. As Chapter III showed, queries using tuple variables can be extremely awkward to express, especially when the data base becomes complicated. Queries using domain variables, however, are unaffected by the complexity of the data base.

Micro-Planner also provides complete information as to how the retrieval can be accomplished. Using only this information, it is a trivial task to create the relation. Re-expressing this query as a query in the old relational calculus while possible, is simply not practical. Codd's relational calculus is just too awkward to use.

Further, the new reduction algorithm is often more efficient than Codd's. Consider, for example, the quantified expression:

$\forall$(P2r2 & (r1[1]=10 v r1[1]=12))

In the new relational calculus, this query is represented as:

$\forall$(PART# & (OR (EQ PART# 10) (EQ PART# 12)))

using Codd's reduction algorithm, the two relations defined by:

(P2r2 & r2[1]=10), and

(P2r2 & r2[1]=12)

are created, and their union is taken. This process requires four passes through relations: two for creating the new relations defined above, and two for constructing the union of these relations. With the new reduction algorithm, only one pass of one relation is made, since the tuples of the relation which enumerate PART# are simply subjected to the restriction which is given.

□ 5.6. The Treatment of Real World Knowledge □

In the current system, input from a user which cannot be interpreted as a query is added to the semantic model. This allows users to give the system any information which they think is relevant, and the system in turn uses this information when processing a query. This gives the user the impression that the system has "understood" what he has said. One such class of information is *real world information*.

5.6.A Representation of Real World Knowledge

When a user enters a piece of real world information, it is immediately placed in the semantic model. This in itself does little good, since there is no indication of *how* this information affects the data base. Therefore, corresponding to each type of real world knowledge which might be entered is a Micro-Planner ANTECEDENT theorem, whose purpose is to notify the system which part of the data base is affected by this type of information. The system must provide facilities to allow these theorems to be expressed. Of particular concern is that in writing theorems the user need never explicitly reference a relation in the data base. Instead, he should be able to say, "This information affects all relations which satisfy this criterion."

This is in keeping with the general principle that if the organization of the data is changed and the semantic model is changed accordingly, the user and his programs should be unaffected by the change.

The effect of any real world information will be to alter the active portion of the data base. Therefore, the theorems discussed above must make use of the assertions which state that relations are inactive and that the usable tuples of certain relations are to be restricted.

The current system demonstrates how real world information is handled. For example, we can say:

```
    (CURRENT PARTS ARE NOT AVAILABLE) or
    (OBSOLETE PARTS ARE NOT AVAILABLE).
```

The corresponding Micro-Planner theorem is:

```
(THANTE NOT-AVAILABLE (X RELS) ($?X ARE NOT AVAILABLE)
 (THSETQ $?RELS
      (THFIND ALL ($?REL) (REL) ($G ($?REL CONCERNS $?X)) ))
 (THMAPC 'ASSERTFUN $?RELS) )

(THCONSE ASSERTFUN (X) ($?X) ($A ($?X IS INACTIVE)))
```

This theorem states that the result of asserting that X is not available is to assert for each relation REL which CONCERNS X that REL is inactive. Thus if we:

($A (OBSOLETE PARTS ARE NOT AVAILABLE) $T),

the assertion:

($A (R8 IS NOT AVAILABLE))

will automatically be added to the semantic model.

The system also handles real world information dealing with suppliers who go on strike. Since strikes affect the supply of parts, when someone asks for the suppliers who can supply a certain part, clearly any supplier who is on strike should not be included. Therefore, any semantic information which is concerned with the supplying of a part should be "notified" that there is a restriction on the suppliers who can currently supply the parts. On the other hand, if we ask for a list of all suppliers, we really want all suppliers regardless of whether or not they are on strike. This is the reason why restrictions are stated as:

(SUPPLIES IN R3 IS-RESTRICTED-TO REST#1) rather than

(R3 IS-RESTRICTED-TO REST#1).

As an example, consider the statement:

(VANCOUVER SUPPLIERS ARE ON STRIKE).

If this were asserted, the antecedent thereom ON-STRIKE would immediately be invoked. This general theorem says that if we know that X suppliers are on strike, then:

1. Assert that (REST#1 RESTRICTS SLOC TO (NOT (EQ SLOC X)))
2. Find all relations R which say that either:
    a. A supplies B
    b. A supplies# B, where supplies# is "supplies number of

parts"

c. A is-supplied-by B,

and then assert that

a. (SUPPLIES IN R IS-RESTRICTED-TO REST#N), or

b. (SUPPLIES# IN R IS-RESTRICTED-TO REST#N), or

c. (IS-SUPPLIED-BY IN R IS-RESTRICTED-TO REST#N)

In other words, notify all relevant predicates that the suppliers they have as arguments must satisfy certain criterion before they can be used.

The use of this type of system means that if a strike occurs, it is not necessary to keep two copies of each relation - one which takes the affects of the strike into account, and one which does not. Instead, the data base remains constant, and the system makes sure that it uses only active tuples and relations.

¤ 5.7. Multiple Path Problems ¤

One of the serious problems which arises when trying to resolve a query in a language which is relation independent is that of multiple paths. There are often several ways to answer the same query, and thus there are often several "correct" proofs. Sometimes any one of these proofs is sufficient; sometimes all are required. Three cases are examined - one where multiple paths occur due to several relations containing the same information, one where queries are ambiguous and several paths are found, each of which has a different meaning, and finally one where several paths exist but each has the same meaning.

Consider a semantic model which contains the following information:

a. (S# IS-NAMED SNAME R3)

b. (S# SUPPLIES PART# R5)

c. (S# SUPPLIES PART# R7)

In this case, there are two ways in which SNAME and PART# can be related, and thus the proof of (SNAME SUPPLIES PART# $?REL) could be accomplished using either a and b or a and c. The proof which is selected is of no importance, since before the final result is computed, the system automatically searches for all the relations which contain any information used in the proof. The union of these relations is taken, and the result used in further processing. Multiple paths due to several relations containing the same information are all dealt with in this manner.

Consider now a semantic model which contains the following information:

a. (EMPLOYEE IS-PAID SALARY WORKS-SHIFT SHIFT# R1)

b. (SHIFT# IS-SUPERVISED-BY FOREMAN R2)

c. (FOREMAN IS-PAID SALARY R3)

The request (EMPLOYEE $?R FOREMAN $?REL) to relate employees to foremen by some arbitrary relation is ambiguous; the system does not know whether the user wants the employees and foremen who are working the same shift, or who receive the same salary. In any case where the query contains a predicate which was left unspecified, the query may be ambiguous and there will be several "correct" proofs. The current system simply chooses one of them.

If one is working in a natural language environment, if a query is encountered which compiles into a relational calculus query which has an unspecified predicate, the user should either be asked to clarify his request, or be told which meaning the system has assumed.

Finally, consider a semantic model which contains the following information:

a. (S# SUPPLIES PART# R1)

b. (PART# IS-CALLED PART-NAME R2)

c. (S# SUPPLIES PROJECT R3)

d. (PROJECT USES PART-NAME R4)

If a relational term (S# SUPPLIES PART-NAME $?REL) were to be proven, then assuming the proper T-TRANS and DETERMINES conditions hold, the following two proofs are possible:

1. (S# SUPPLIES PART# R1) and
   (PART# IS-CALLED PART-NAME R2), or

2. (S# SUPPLIES PROJECT R3) and
   (PROJECT USES PART-NAME R4).

This example illustrates the case where two paths exist each of which have the same meaning. Providing the data base is consistent it does not matter which proof is chosen since the result of each proof is the same. However, if the data base is not consistent (ie. Some information is missing from one of the files) then one _must_ perform both proofs and take the union of the results.

In general it is necessary to find all valid proofs. The current system does _not_ do this; it finds only one such proof. Attempting to find all possible proofs is combinatorially explosive since there is no a priori bound on the maximum length of a valid path. In effect, one must do a breadth first search with truncation on cycles. The current system essentially does theorem proving in the propositional calculus, and therefore this problem is decidable. However, the solution is impractical.

It follows that the system presented in _incomplete_ when data bases are not consistent. That is, there exist queries for which a partial solution is returned. The system will sometimes return certain values which are correct, but not necessarily _all_ the values. This problem only arises when the information in the data base is incomplete. This case must be considered since even consistent data bases are often inconsistent during periods of online update.

VI. SUMMARY

Current research in relational data base management systems has resulted in the proposal of several different query languages. Each of these require the user to determine how the retrieval is to be done, and to specify the relations which are relevant to the request. This type of language is unsuitable for many applications, and is undesirable since the user is not independent of the organization of the data base.

A need was observed for a new query language in which queries specify only the properties of the data to be retrieved. This necessitates the development of a system which is capable of determining the relations which are relevant to a query, and deciding how the retrieval can be done using these relations.

This thesis has presented a new query language for relational systems, and a framework which is capable of processing these queries. The language is data base independent and queries never reference the relations directly. In order to retrieve information, each query is formulated as a series of theorems to be proven. The steps of the proof show how the retrieval can be done.

The reduction of queries is based upon information contained in a semantic model. This model forms the basis for the entire system, since the information it contains completely characterizes the data base. In this work, the semantic model is represented as a set of Micro-Planner assertions, although it is shown that it could easily be represented using a more conventional data structure. It is felt that a semantic model will be necessary in any system which attempts to handle queries in a data base independent language.

The new query language can be used as the target language of a natural language system. The semantic components of these systems are capable of compiling a natural language query into a representation which specifies the properties of the request. They are not capable of determining how the retrieval is to be accomplished, nor should they be. This means that the query languages of existing systems are not suitable as target languages, since the systems can not determine how the retrieval should be done.

Future relational systems which hope to make use of natural language input, or at least provide users with a query language which does not require them to know how their request will be processed, will be forced to address the problems discussed in this thesis.

BIBLIOGRAPHY

1. Armenti, A. et al. "LISTAR - Lincoln Information Storage and Associative Retrieval System." Proceedings of the 1970 SJCC, pp.313-322.

2. Bracchi, G. et al. "A Relational Data Base Management System." Proceedings of the 1972 Annual Conference of the ACM, pp.1080-1089.

3. Childs, David L. "Description of a Set-Theoretic Data Structure." Proceedings of the 1968 FJCC, pp.557-564.

4. Childs, David L. "Feasibility of a Set-Theoretic Data Structure." Proceedings of the 1968 IFIP Congress, pp.420-430.

5. Codasyl Systems Committee. Feature Analysis of Generalized Data Base Management Systems, May 1971.

6. Codasyl Systems Committee. A Survey of Generalized Data Base Management Systems, May 1969.

7. Codd, E.F. "A Data Base Sublanguage Founded on the Relational Calculus." Proceedings of the 1971 ACM SIGFIDET Workshop, pp.35-67.

8. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM 6(1970), pp.337-387.

9. Codd, E.F. Further Normalization of the Data Base Relational Model. I.B.M. Research Report RJ909, Yorktown Heights, N.Y., 1971.

10. Codd, E.F. Normalized Data Base Structure: A Brief Tutorial. I.B.M. Research Report RJ935, Yorktown Heights, N.Y., 1971.

11. Codd, E.F. Relational Completeness of Data Base Sublanguages. I.B.M. Research Report RJ987, Yorktown Heights, N.Y., 1972.

12. Feldman, J. and P. Rovner. "An Algol-Based Associative Language." Communications of the ACM 8(1969), pp.439-458.

13. Codd, E.F. Seven Steps to Rendezvous With the Casual User. I.B.M. Research Report RJ1333, Yorktown Heights, N.Y., 1974.

14. Gammill, R.C. "Applications of Relational Data Structure Models in Man-Machine Systems." Proceedings of the 1972 Annual Conference of the ACM, pp.460-468.

15. Goldstein, Robert C. and Alois Strnad. "The MacAIMS Data Management System." Proceedings of the 1970 ACM SIGFIDET Workshop, pp.200-228.

16. Hall. W., B. Jervis, and J. Jervis. WALT. Department of Computer Science, University of British Columbia, 1973.

17. Heath, I.J. "Unacceptable File Operations in a Relational Data Base." Proceedings of the 1971 ACM SIGFIDET Workshop, pp.19-33.

18. Hewitt, Carl. Description and Theoretical Analysis (Using Schema) of Planner: A Language for Proving Theorems and Manipulating Models in a Robot. Cambridge, Mass: Massachusetts Instutute of Technology, Revised Ph.D. Dissertation, AI Technical Report no 258, 1972.

19. Jervis, B. And J.L. Parker. "An Approach for a Working Relational Data System." Proceedings of the 1972 ACM SIGFIDET Workshop, pp.125-145.

20. Kuhns, J.L. "Quantification in Query Systems." Proceedings of the 1971 ACM Symposium on Information Storage and Retrieval, pp.81-92.

21. Levien, R.E. and M.E. Maron. "A Computer System for Inference Execution and Data Retrieval." Communications of the ACM 11(1967), pp.715-721.

22. Mealy, G.H. "Another Look at Data." Proceedings of the 1967 FJCC, pp.525-534.

23. Minker, J. and S. Rosenfeld. "Introduction and Perspectives for the 1971 ACM Information Storage and Retrieval Symposium." Proceedings of the 1971 ACM Symposium on Information Storage and Retrieval, pp.1-3.

24. Notley, M.G. The PETERLEE IS/1 System. I.B.M. UK Scientific Report, Peterlee, U.K., 1972.

25. Pacak, Milos and A. Pratt. "The Function of Semantics in Automated Language Processing." Proceedings of the 1971 ACM Symposium on Information Storage and Retrieval, pp.5-17.

26. Palermo, Frank P. A Data Base Search Problem. I.B.M. Research Report RJ1072, Yorktown Heights, N.Y., 1972.

27. Strnad, A.J. "The Relational Approach to the Management of Data Bases." Proceedings of the 1971 IFIP Congress, August 1971. Pp.91-95.

28. Symonds, A.J. and R. Lorie. "A Schema for Describing a Relational Data Base." Proceedings of the 1970 ACM SIGFIDET Workshop, pp.230-244.

29. Thompson, F. et al. "REL - A Rapidly Extensible Language System." Proceedings of the 24th ACM National Conference, 1969. pp.399-417.

30. Tsichritzis, D. et al. "The Case for Relational Data Bases." CIPS Computer Magazine 2(1974), pp.21-28.

31. Wilcox, B. and C. Hafner. LISP/MTS User's Guide. Mental Health Research Institute, Ann Arbor, Michigan, 1973.

32. Winograd, Terry. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Cambridge, Mass.: Massachusetts Institute of Technology, Revised Ph.D. Dissertation, MAC TR-84,1971.

33. Woods, W. "Procedural Semantics for a Question Answering Machine." Proceedings of the 1968 FJCC, pp.457-471.

34. Woods, W. The Lunar Sciences Natural Language and Information System: Final Report. Bolt, Beranek, and Newman Inc., BBN Report No. 2378, 1972.

35. Woods, W. "Transition Network Grammars for Natural Language Analysis." Communications of the ACM 10 (1970), pp.591-606.

APPENDIX 1.

The Data Base

R2 (

PROJECT#	PROJECT-NAME	PLOC)
1	ROYAL TOWERS	VANCOUVER
2	BURRARD SHIPYARDS	VANCOUVER
3	P.C.C.	MERRIT
4	RAPID TRANSIT	VICTORIA
5	GRANVILLE MALL	VANCOUVER
6	OLYMPIC SEAWAY	VICTORIA

R3 (

SNAME	S#	SLOC)
APPOLLO SHEET METAL	1	MONTREAL
VALLEY STEEL	2	TORONTO
PEARSON IRON WORKS	3	MONTREAL
COAST STEEL	4	VANCOUVER

R4 (

S#	PART#	#PARTS	PROJECT#	DATE-RECEIVED)
1	7	4	2	SEPT 15
1	3	1000	3	SEPT 19
1	7	25	3	SEPT 19
1	2	40	6	OCT 1
2	1	400	4	AUG 21
2	1	800	5	AUG 24
2	5	50	5	AUG 24
2	3	1000	1	SEPT 7
2	7	12	3	SEPT 15
2	7	15	2	SEPT 17
2	7	26	6	SEPT 19
3	4	80	2	MAR 14
3	4	80	2	MAR 27
3	2	200	2	APRIL 17
4	1	400	1	SEPT 6
4	2	20	1	SEPT 6
4	3	1000	2	SEPT 9
4	4	40	1	SEPT 6
4	5	25	4	OCT 12
4	6	1	5	OCT 17
4	7	23	5	OCT 17
4	8	500	3	NOV 3
4	9	10	3	NOV 3

R1 (	PART#	PRICE	PURCHASE-UNITS	)
	1	20	400	
	8	25	500	
	3	2	1000	
	9	40	10	
	5	45	25	
	6	98	1	
	7	74	1	

R5 (	PROJECT#	PART#	#PARTS	S#	)
	1	1	400	4	
	1	7	14	4	
	1	1	800	1	
	1	3	1000	2	
	1	8	25	2	
	1	1	400	2	
	1	7	15	2	
	1	1	400	3	
	2	4	40	1	
	2	3	1000	4	
	3	3	1000	1	
	3	4	80	2	
	3	9	10	4	
	3	6	1	4	
	4	6	4	3	
	4	9	100	3	
	4	5	50	3	
	4	6	3	1	
	4	4	40	4	
	4	5	25	4	
	4	8	500	4	
	5	3	1000	1	
	5	9	10	4	
	6	5	50	1	
	6	7	2	4	

R10 (	ORDER#	SALESMAN#	)
	1	17	
	2	27	
	3	32	
	4	4	
	5	27	

R6 (

PROJECT#	PART#	QOH
1	1	14
1	7	6
1	3	754
2	3	201
2	4	1
2	1	207
2	5	13
3	4	1
4	6	2
4	5	140
5	9	47
5	3	28
5	1	101
6	5	24

)

R7 (

S#	PART#	QOH
1	1	1600
1	4	40
1	3	4000
2	2	20
2	4	41
3	1	1200
3	6	40
3	9	3
4	1	800
4	2	3
4	3	2000
4	8	500
4	5	50
4	6	3
4	9	20

)

R8 (

PART#	PRICE	PURCHASE-UNITS
2	36	10
4	48	40

)

R9 (

ORDER#	SOURCE
1	MONTREAL
2	MONTREAL
3	TORONTO
4	VANCOUVER

)

APPENDIX 2.

The Semantic Model

```
($A (PART#  COSTS  PRICE  R1))
($A (PART#  COMES-IN  PURCHASE-UNITS  R1))
($A (R1 P-ENUMERATES PART#))
($A (R1 CONCERNS CURRENT-PARTS))


($A (PROJECT#  IS-CALLED  PROJECT-NAME  R2))
($A (PROJECT-NAME  IS-NUMBERED  PROJECT#  R2))
($A (PROJECT#  IS-IN  PLOC  R2))
($A (PROJECT# DETERMINES PLOC R2))
($A (PROJECT# DETERMINES PROJECT-NAME R2))
($A (PROJECT-NAME DETERMINES PROJECT# R2))
($A (PROJECT-NAME DETERMINES PLOC R2))
($A (R2 ENUMERATES PROJECT#))
($A (R2 ENUMERATES PLOC))
($A (R2 ENUMERATES PROJECT-NAME))
($A (R2 IS T-TRANS))


($A (S#  IS-CALLED  SNAME  R3))
($A (SNAME  IS-NUMBERED  S#  R3))
($A (S#  IS-IN  SLOC  R3))
($A (S# DETERMINES SLOC R3))
($A (S# DETERMINES SNAME R3))
($A (SNAME DETERMINES S# R3))
($A (SNAME DETERMINES SLOC R3))
($A (R3 ENUMERATES S#))
($A (R3 ENUMERATES SNAME))
($A (R3 ENUMERATES SLOC))
($A (R3 IS T-TRANS))


($A (S#  SUPPLIED  PART#  R4))
($A (S#  SUPPLIED-TO  PROJECT#  R4))
($A (PROJECT#  ORDERED-FROM  S#  R4))
($A (S#  SUPPLIED-#  #PARTS  R4))
($A (PROJECT#  USES  PART#  R4))
($A (PART#  IS-USED-IN  PROJECT#  R4))
($A (PROJECT#  ORDERED  PART#  R4))
($A (PROJECT#  RECEIVED-ON  DATE-RECEIVED  R4))
($A (PROJECT#  RECEIVED-#  #PARTS  RECEIVED-ON DATE-RECEIVED R4))
($A (PROJECT# RECEIVED PART# RECEIVED-ON DATE-RECEIVED R4))
($A (S#  SUPPLIED-#  #PARTS  SUPPLIED-TO  PROJECT#  R4))
($A (S#  SUPPLIED  PART#  SUPPLIED-TO  PROJECT#  R4))
```

```
($A (S#  SUPPLIES-TO  PROJECT#   R5))
($A (S#  SUPPLIES  PART#   R5))
($A (S#  SUPPLIES-#  #PARTS   R5))
($A (PROJECT#  ORDERED-FROM  S#   R5))
($A (PROJECT#  ORDERED-#  #PARTS   R5))
($A (PROJECT#  ORDERED  PART#   R5))
($A (PROJECT#  USES  PART#   R5))
($A (PART#  IS-USED-IN  PROJECT#   R5))
($A (PART# IS-SUPPLIED-BY S# R5))
($A (S# SUPPLIES PART# SUPPLIES-TO PROJECT# R5))
($A (S# SUPPLIES-# #PARTS SUPPLIES-TO PROJECT# R5))


($A (PROJECT#  USES  PART#   R6))
($A (PROJECT#  HAS  QOH  OF-TYPE  PART#   R6))
($A (PART#  IS-USED-IN  PROJECT#   R6))


($A (S#  SUPPLIES  PART#   R7))
($A (S#  HAS  QOH  OF-TYPE  PART#   R7))


($A (PART#  COSTS  PRICE   R8))
($A (PART#  COMES-IN  PURCHASE-UNITS   R8))
($A (R8 P-ENUMERATES PART#))
($A (R8 CONCERNS OBSOLETE-PARTS))


($A (ORDER# COMES-FROM SOURCE R9))
($A (R9 P-ENUMERATES ORDER#))


($A (ORDER# WAS-SOLD-BY SALESMAN# R10))
($A (SALESMAN# SOLD ORDER# R10))
($A (R10 ENUMERATES SALESMAN#))
($A (R10 P-ENUMERATES ORDER#))
```

```
($A (EXTRA DETERMINES))
($A (EXTRA ENUMERATES))
($A (EXTRA IS))
($A (EXTRA CONCERNS))
($A (EXTRA P-ENUMERATES))
($A (EXTRA RESTRICTS))
($A (EXTRA TO))
($A (EXTRA IN))
($A (EXTRA IS-RESTRICTED-TO))
```

APPENDIX 3.

Sample Queries in the Relational Algebra

This appendix shows some sample queries in the relational algebra. There are five sets of queries, each of which produces a relation that contains the information which answers a certain request. The requests are shown at the start of each set of queries, and intermediate relations which are formed are usually printed.

1. List all the part numbers.

```
(RUNION '(R1 R8))
   REL1
(PRINTREL 'REL1)
   REL1 (  PART#  PRICE  PURCHASE-UNITS    )
             7     74           1
             6     98           1
             5     45          25
             9     40          10
             3      2        1000
             8     25         500
             1     20         400
             4     48         40
             2     36         10
   REL1

(PROJECT 'REL1 '(1))
   REL2
(PRINTREL 'REL2)
   REL2 (  PART#   )
             7
             6
             5
             9
             3
             8
             1
             4
             2
   REL2
```

2. Find the parts which no supplier has in stock.

```
(RDIFF 'REL2 (PROJECT 'R7 '(2)))
   REL4
(PRINTREL 'REL4)
   REL4 (  PART#   )
             7
   REL4
```

3. List the names of the projects which are in Vancouver.

```
(PROJECT (RESTRICT 'R2 '(EQ (ELEM T1 3) 'VANCOUVER)) '(2))
   REL13
(PRINTREL 'REL13)
   REL13 (   PROJECT-NAME   )
             ROYAL TOWERS
           BURRARD SHIPYARDS
            GRANVILLE MALL
   REL13
```

4. Find the suppliers who supplied EACH project.

```
(PROJECT 'R5 '(1 4))
   REL5
(PROJECT 'R2 '(1))
   REL6
(RDIVIDE 'REL5 'REL6 '(1) '(1))
   REL8
(PRINTREL 'REL8)
   REL8 (  S#   )
           4
           1
   REL8
```

5. Find the names of the suppliers who supplied more
   than 25 units of part 4 to some project.

```
(RESTRICT 'R4 '(AND (EQ (ELEM T1 2) 4)
                    (GREATERP (ELEM T1 3) 25)))
   REL10
(PRINTREL 'REL10)
   REL10 (  S#  PART#  #PARTS  PROJECT#  DATE-RECEIVED   )
            4     4      40        1         SEPT 6
            3     4      80        2         MAR 27
            3     4      80        2         MAR 14
   REL10
(JOIN 'REL10 'R3 '(EQ (ELEM T1 1) (ELEM T2 2)))
   REL11
(PROJECT 'REL11 '(6))
   REL12
(PRINTREL 'REL12)
   REL12 (        SNAME   )
               COAST STEEL
            PEARSON IRON WORKS
   REL12
```

APPENDIX 4.

Sample Queries in Codd's Relational Calculus

In this appendix, the following relations are used:

R1 (	S#	SLOC	SNAME	)
	211	NY	AA	
	325	SF	XX	
	237	LA	YY	

R2 (	J#	JLOC	JNAME	)
	970	POK	A	
	971	SJ	X	
	972	SJ	Y	

R3 (	P#	PTYPE	)
	31	A	
	32	A	
	33	B	

R4 (	S#	P#	J#	DR	)
	211	31	971	11	
	325	32	971	31	
	211	33	970	55	
	211	31	972	66	
	237	31	970	75	
	237	32	970	91	
	237	33	970	101	
	237	32	971	121	
	237	31	971	121	
	237	31	972	125	

```
1. r1[3],r1[2] : P1r1 & ∃(P2r2 & r2[2]=SJ)
                        ∀(P3r3 & r3[2]=A)
                        ∃(P4r4)
                        ((r4[1]=r1[1]) &
                         (r4[3]=r2[1]) &
                         (r4[2]=r3[1]))
```

```
(REDUCE 'Q1)
    THE REDUCED GLOBAL RANGE FOR R4 IS:
    REL1 (  S#   P#  J#   )
            237  31  972
            237  31  971
            237  32  971
            237  33  970
            237  32  970
            237  31  970
            211  31  972
            211  33  970
            325  32  971
            211  31  971
    THE REDUCED GLOBAL RANGE FOR R3 IS:
    REL3 (  P#   )
            31
            32
    THE REDUCED GLOBAL RANGE FOR R2 IS:
    REL5 (  J#   )
            971
            972
    THE REDUCED GLOBAL RANGE FOR R1 IS:
    REL6 (  S#   SLOC  SNAME   )
            237   LA    YY
            325   SF    XX
            211   NY    AA
    THE REDUCED LOCAL RANGE FOR R4 IN THETA1 IS
    REL1 (  S#   P#  J#   )
            237  31  972
            237  31  971
            237  32  971
            237  33  970
            237  32  970
            237  31  970
            211  31  972
            211  33  970
            325  32  971
            211  31  971
    THE REDUCED LOCAL RANGE FOR R3 IN THETA1 IS
    REL3 (  P#   )
            31
            32
    THE REDUCED LOCAL RANGE FOR R2 IN THETA1 IS
    REL5 (  J#   )
            971
            972
    THE REDUCED LOCAL RANGE FOR R1 IN THETA1 IS
    REL6 (  S#   SLOC  SNAME   )
            237   LA    YY
            325   SF    XX
            211   NY    AA
```

```
THE FIRST ELEMENT IN THE CORE OF THETA1 IS:
REL3 (  P#    )
        31
        32
JOINING CORE WITH R4 YIELDS:
REL7 (  P#   S#   P#   J#    )
        32   325  32   971
        32   237  32   970
        32   237  32   971
        31   211  31   971
        31   211  31   972
        31   237  31   970
        31   237  31   971
        31   237  31   972
JOINING CORE WITH R2 YIELDS:
REL8 (  P#   S#   P#   J#   J#    )
        31   237  31   972  972
        31   237  31   971  971
        31   211  31   972  972
        31   211  31   971  971
        32   237  32   971  971
        32   325  32   971  971
JOINING CORE WITH R1 YIELDS:
REL9 (  P#   S#   P#   J#   J#   S#   SLOC  SNAME   )
        32   325  32   971  971  325   SF     XX
        32   237  32   971  971  237   LA     YY
        31   211  31   971  971  211   NY     AA
        31   211  31   972  972  211   NY     AA
        31   237  31   971  971  237   LA     YY
        31   237  31   972  972  237   LA     YY
PROJECTING OFF R4 YIELDS:
REL10 (  P#   J#   S#   SLOC  SNAME   )
         31   972  237   LA     YY
         31   971  237   LA     YY
         31   972  211   NY     AA
         31   971  211   NY     AA
         32   971  237   LA     YY
         32   971  325   SF     XX
DIVISION BY R3 YIELDS:
REL12 (  J#   S#   SLOC  SNAME   )
         971  237   LA     YY
PROJECTING OFF R2 YIELDS:
REL13 (  S#   SLOC  SNAME   )
         237   LA     YY
THE RESPONSE RELATION IS:
REL14 (  SNAME  SLOC   )
          YY     LA
REL14
```

```
2. r1[3]: P1r1 & P4r2 &
                  ((r2[2]=32 & r1[1]=r2[1]) V
                   (r2[3]=970 & r1[1]=r2[1] & ¬(r1[1]=237)))

   (REDUCE 'Q1)
       THE REDUCED GLOBAL RANGE FOR R2 IS:
       REL15 (  S#   P#  J#   )
                237  31  972
                237  31  971
                237  32  971
                237  33  970
                237  32  970
                237  31  970
                211  31  972
                211  33  970
                325  32  971
                211  31  971
       THE REDUCED GLOBAL RANGE FOR R1 IS:
       REL16 (  S#   SLOC  SNAME   )
                237   LA    YY
                325   SF    XX
                211   NY    AA
       THE REDUCED LOCAL RANGE FOR R2 IN THETA1 IS
       REL17 (  S#   P#  J#   )
                325  32  971
                237  32  970
                237  32  971
       THE REDUCED LOCAL RANGE FOR R1 IN THETA1 IS
       REL16 (  S#   SLOC  SNAME   )
                237   LA    YY
                325   SF    XX
                211   NY    AA
       THE FIRST ELEMENT IN THE CORE OF THETA1 IS:
       REL16 (  S#   SLOC  SNAME   )
                237   LA    YY
                325   SF    XX
                211   NY    AA
       JOINING CORE WITH R2 YIELDS:
       REL18 (  S#   SLOC  SNAME  S#   P#  J#   )
                325   SF    XX    325  32  971
                237   LA    YY    237  32  971
                237   LA    YY    237  32  970
       THE REDUCED LOCAL RANGE FOR R2 IN THETA2 IS
       REL19 (  S#   P#  J#   )
                211  33  970
                237  31  970
                237  32  970
                237  33  970
```

```
THE REDUCED LOCAL RANGE FOR R1 IN THETA2 IS
REL20 (  S#    SLOC  SNAME    )
         211    NY    AA
         325    SF    XX
THE FIRST ELEMENT IN THE CORE OF THETA2 IS:
REL20 (  S#    SLOC  SNAME    )
         211    NY    AA
         325    SF    XX
JOINING CORE WITH R2 YIELDS:
REL21 (  S#    SLOC  SNAME  S#    P#   J#    )
         211    NY    AA    211   33   970
PROJECTING OFF R2 YIELDS:
REL23 (  S#    SLOC  SNAME    )
         325    SF    XX
         237    LA    YY
         211    NY    AA
THE RESPONSE RELATION IS:
REL24 (  SNAME    )
          AA
          YY
          XX
REL24
```

APPENDIX 5.

Sample Micro-Planner Runs

```
($G (PROVE* SNAME SUPPLIES PART#) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES PART#
   (R7 SAYS S# SUPPLIES PART#)
   (TRY TO RELATE SNAME AND S#)
   (R3 SAYS S# IS-CALLED SNAME)
   DONE.
#RESULT#
   ((SNAME (R3)) (PART# (R5 R7)) ((R3) S# (R5 R7)))
#RESTRICT#
   NIL

($G (PROVE* SNAME SUPPLIES-TO PROJECT-NAME) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES-TO PROJECT-NAME
   (R5 SAYS S# SUPPLIES-TO PROJECT#)
   (TRY TO RELATE SNAME AND S#)
   (R3 SAYS S# IS-CALLED SNAME)
   (G2 SAYS SNAME SUPPLIES-TO PROJECT#)
   (TRY TO RELATE PROJECT# AND PROJECT-NAME)
   (R2 SAYS PROJECT# IS-CALLED PROJECT-NAME)
   DONE.
#RESULT#
   ((SNAME (R3)) (PROJECT-NAME (R2))
                 ((R3) S# (R5) PROJECT# (R2)))

($G (PROVE* PROJECT-NAME USES PART#) $T)
   ATTEMPT TO PROVE PROJECT-NAME USES PART#
   (R6 SAYS PROJECT# USES PART#)
   (TRY TO RELATE PROJECT-NAME AND PROJECT#)
   (R2 SAYS PROJECT# IS-CALLED PROJECT-NAME)
   DONE.
#RESULT#
   ((PROJECT-NAME (R2)) (PART# (R4 R5 R6))
                        ((R2) PROJECT# (R4 R5 R6)))

($G (PROVE* PROJECT-NAME ORDERED PART#) $T)
   ATTEMPT TO PROVE PROJECT-NAME ORDERED PART#
   (R5 SAYS PROJECT# ORDERED PART#)
   (TRY TO RELATE PROJECT-NAME AND PROJECT#)
   (R2 SAYS PROJECT# IS-CALLED PROJECT-NAME)
   DONE.
#RESULT#
   ((PROJECT-NAME (R2)) (PART# (R4 R5)) ((R2) PROJECT# (R4 R5)))
```

```
($G (PROVE* PROJECT-NAME ORDERED-FROM S#) $T)
   ATTEMPT TO PROVE PROJECT-NAME ORDERED-FROM S#
   (R5 SAYS PROJECT# ORDERED-FROM S#)
   (TRY TO RELATE PROJECT-NAME AND PROJECT#)
   (R2 SAYS PROJECT# IS-CALLED PROJECT-NAME)
   DONE.
#RESULT#
   ((PROJECT-NAME (R2)) (S# (R4 R5)) ((R2) PROJECT# (R4 R5)))

($G (PROVE* S# SUPPLIES-TO PROJECT-NAME) $T)
   ATTEMPT TO PROVE S# SUPPLIES-TO PROJECT-NAME
   (R5 SAYS S# SUPPLIES-TO PROJECT#)
   (TRY TO RELATE PROJECT# AND PROJECT-NAME)
   (R2 SAYS PROJECT# IS-CALLED PROJECT-NAME)
   DONE.
#RESULT#
   ((S# (R5)) (PROJECT-NAME (R2)) ((R5) PROJECT# (R2)))

($G (PROVE* PART# COMES-IN PURCHASE-UNITS) $T)
   ATTEMPT TO PROVE PART# COMES-IN PURCHASE-UNITS
   DONE.
#RESULT#
   ((PART# (R1 R8)) (PURCHASE-UNITS (R1 R8)) ((R1 R8)))

($G (PROVE* SNAME SUPPLIED-TO PLOC) $T)
   ATTEMPT TO PROVE SNAME SUPPLIED-TO PLOC
   (R4 SAYS S# SUPPLIED-TO PROJECT#)
   (TRY TO RELATE SNAME AND S#)
   (R3 SAYS S# IS-CALLED SNAME)
   (G8 SAYS SNAME SUPPLIED-TO PROJECT#)
   (TRY TO RELATE PROJECT# AND PLOC)
   (R2 SAYS PROJECT# IS-IN PLOC)
   DONE.
#RESULT#
   ((SNAME (R3)) (PLOC (R2)) ((R3) S# (R4) PROJECT# (R2)))

($G (PROVE* S# SUPPLIES PART# SUPPLIES-TO PROJECT-NAME) $T)
   ATTEMPT TO PROVE S# SUPPLIES PART# SUPPLIES-TO PROJECT-NAME
   DONE.
#RESULT#
   ((S# (R5)) (PART# (R5)) (PROJECT-NAME (R2))
                           ((R5) PROJECT# (R2)))

($G (PROVE* SNAME SUPPLIES PART# SUPPLIES-TO PROJECT-NAME) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES PART#
    SUPPLIES-TO PROJECT-NAME
   DONE.
#RESULT#
   ((SNAME (R3)) (PART# (R5)) (PROJECT-NAME (R2))
                              ((R5) PROJECT# (R2) S# (R3)))
```

```
($G (ENUMERATE S#) $T)
   (ENUMERATE S#)
#RESULT#
   ((S# (R3)) ((R3)))

($G (ENUMERATE PART#) $T)
   (ENUMERATE PART#)
#RESULT#
   ((PART# (R1 R8)) ((R1 R8)))

($G (ENUMERATE PLOC) $T)
   (ENUMERATE PLOC)
#RESULT#
   ((PLOC (R2)) ((R2)))

($G (ENUMERATE PURCHASE-UNITS) $T)
   (ENUMERATE PURCHASE-UNITS)
#RESULT#
   ((PURCHASE-UNITS (R8 R1)) ((R8 R1)))

($A (OBSOLETE-PARTS ARE NOT AVAILABLE) $T)
   ((OBSOLETE-PARTS ARE NOT AVAILABLE))

($G (ENUMERATE PART#) $T)
   (ENUMERATE PART#)
#RESULT#
   ((PART# (R1)) ((R1)))

($A (R7 IS INACTIVE))
   ((R7 IS INACTIVE))

($G (PROVE* SNAME SUPPLIES PART#) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES PART#
   (R5 SAYS S# SUPPLIES PART#)
   (TRY TO RELATE SNAME AND S#)
   (R3 SAYS S# IS-CALLED SNAME)
   DONE.
#RESULT#
   ((SNAME (R3)) (PART# (R5)) ((R3) S# (R5)))

(THERASE (R7 IS INACTIVE))
   ((R7 IS INACTIVE))
```

```
($A (VANCOUVER SUPPLIERS ARE ON STRIKE) $T)
   ((VANCOUVER SUPPLIERS ARE ON STRIKE))

($G (PROVE* SNAME SUPPLIES PART#) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES PART#
   MAKING NOTE OF RESTRICTION ON SUPPLIES IN R5
   (R5 SAYS S# SUPPLIES PART#)
   (TRY TO RELATE SNAME AND S#)
   (R3 SAYS S# IS-CALLED SNAME)
   DONE.
#RESULT#
   ((SLOC) (SNAME (R3)) (PART# (R7 R5)) ((R3) S# (R7 R5)))
#RESTRICT#
   ((NOT (EQ SLOC (QUOTE VANCOUVER))))

($G (PROVE* S# SUPPLIES PART#) $T)
   ATTEMPT TO PROVE S# SUPPLIES PART#
   MAKING NOTE OF RESTRICTION ON SUPPLIES IN R7
   DONE.
#RESULT#
   ((SLOC) (S# (R5 R7)) (PART# (R3)) ((R5 R7) S# (R3)))
#RESTRICT#
   ((NOT (EQ SLOC (QUOTE VANCOUVER))))

($G (PROVE* SNAME SUPPLIES PART# SUPPLIES-TO PROJECT-NAME) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES PART#
    SUPPLIES-TO PROJECT-NAME
   MAKING NOTE OF RESTRICTION ON SUPPLIES IN R5
   DONE.
#RESULT#
   ((SNAME (R3)) (PART# (R5)) (PROJECT-NAME (R2))
                              ((R5) PROJECT# (R2) S# (R3)))
#RESTRICT#
   ((NOT (EQ SLOC (QUOTE VANCOUVER))))

($A (MERRIT SUPPLIERS ARE ON STRIKE) $T)
   ((MERRIT SUPPLIERS ARE ON STRIKE))

($G (PROVE* SNAME SUPPLIES PART#) $T)
   ATTEMPT TO PROVE SNAME SUPPLIES PART#
   MAKING NOTE OF RESTRICTION ON SUPPLIES IN R5
   (R5 SAYS S# SUPPLIES PART#)
   (TRY TO RELATE SNAME AND S#)
   (R3 SAYS S# IS-CALLED SNAME)
   DONE.
#RESULT#
   ((SLOC) (SNAME (R3)) (PART# (R7 R5)) ((R3) S# (R7 R5)))
#RESTRICT#
   ((NOT (EQ SLOC (QUOTE VANCOUVER)))
    (NOT (EQ SLOC (QUOTE MERRIT))) )
```

```
(THFIND 1 ($?X $?R) (X R)
          ($G (PROVE SALESMAN# $?R DESTINATION $?X) $T)))
   (R10 SAYS SALESMAN# SOLD ORDER#)
   (TRY TO RELATE ORDER# AND DESTINATION)
   (R9 SAYS ORDER# COMES-FROM DESTINATION)
   (R9 SAYS ORDER# COMES-FROM DESTINATION)
   (TRY TO RELATE SALESMAN# AND ORDER#)
   (R10 SAYS ORDER# WAS-SOLD-BY SALESMAN#)
   ABOUT TO TRY FOR A NON-VALID PATH
   (R10 SAYS SALESMAN# SOLD ORDER#)
   (TRY TO RELATE ORDER# AND DESTINATION)
   (R9 SAYS ORDER# COMES-FROM DESTINATION)
   DONE.
#RESULT#
   ((SALESMAN# (R10)) (DESTINATION (R9)) ((R10) ORDER# (R9)))
#RESTRICT#
   NIL
```

APPENDIX 6.

Sample Queries in the New Relational Calculus

1. List the suppliers who supply any part.
```
(SNAME : (SNAME SUPPLIES PART#))
   REL6 (         SNAME    )
              COAST STEEL
          PEARSON IRON WORKS
             VALLEY STEEL
          APPOLLO SHEET METAL
```

2. List the suppliers who supply a part whose number is greater than 20.
```
(SNAME : (SNAME SUPPLIES PART#) & (GREATERP PART# 20))
   NONE.
```

3. List the suppliers who supply a part whose number is greater than 8.
```
(SNAME : (SNAME SUPPLIES PART#) & (GREATERP PART# 8))
   REL33 (         SNAME    )
           PEARSON IRON WORKS
               COAST STEEL
```

4. List the projects who are supplied by supplier number 1.
```
(PROJECT-NAME : (S# SUPPLIES-TO PROJECT-NAME) & (EQ S# 1))
   REL37 (    PROJECT-NAME    )
              ROYAL TOWERS
           BURRARD SHIPYARDS
                 P.C.C.
              RAPID TRANSIT
             GRANVILLE MALL
             OLYMPIC SEAWAY
```

5. List the suppliers who supply each project.
```
(SNAME : (V PROJECT#) (SNAME SUPPLIES-TO PROJECT#))
   REL44 (         SNAME    )
           APPOLLO SHEET METAL
               COAST STEEL
```

6. List the suppliers who supply each part.
```
(SNAME : (V PART#) (SNAME SUPPLIES PART#))
   REL65 (     SNAME    )
           COAST STEEL
```

7. List the parts which are used by project 2.
```
(PART# : (PROJECT# USES PART#) & (EQ PROJECT# 2))
   REL8 (  PART#   )
              5
              1
              7
              4
              2
              3
```

8. List the parts which are either used by project 2 or are supplied by a Vancouver supplier.
```
(PART# : ((PROJECT# USES PART#) & (EQ PROJECT# 2)) V
         ((SLOC SUPPLIES PART#) & (EQ SLOC 'VANCOUVER)))
   REL25 (  PART#   )
               1
               3
               6
               4
               5
               8
               9
               7
               2
```

9. List the suppliers who supply each part which is used by project 2.
```
(SNAME : (V PART# & (PROJECT# USES PART#) & (EQ PROJECT# 2))
         (SNAME SUPPLIES PART#))
   REL54 (     SNAME    )
            COAST STEEL
```

10. List the Vancouver suppliers.
```
(SNAME : (SNAME IS-IN SLOC) & (EQ SLOC 'VANCOUVER))
   REL58 (     SNAME    )
            COAST STEEL
```

11. List the suppliers who are not located in Vancouver.
```
(SNAME : SNAME & ¬(SNAME IS-IN SLOC) & (EQ SLOC 'VANCOUVER))
   REL67 (         SNAME    )
            PEARSON IRON WORKS
               VALLEY STEEL
            APPOLLO SHEET METAL
```

12. List all the suppliers.
```
(SNAME : SNAME)
   REL69 (         SNAME    )
            APPOLLO SHEET METAL
               VALLEY STEEL
            PEARSON IRON WORKS
                COAST STEEL
```

13. List the suppliers who supply part 3 to a Vancouver project.

```
(SNAME : (SNAME SUPPLIES PART# SUPPLIES-TO PLOC)
        & (EQ PART# 3) & (EQ PLOC 'VANCOUVER))
   REL6 (          SNAME    )
               VALLEY STEEL
                COAST STEEL
            APPOLLO SHEET METAL
```

14. List the parts that are supplied to each project by a Vancouver supplier.

```
(PART# : (SLOC SUPPLIED PART# SUPPLIED-TO PROJECT#)
        & (EQ SLOC 'VANCOUVER))
  NONE.
```

15. List the suppliers who have more than 10 units of part 8 in stock.

```
(SNAME : (SNAME HAS QOH OF-TYPE PART#) & (GREATERP QOH 10)
        & (EQ PART# 8))
   REL90 (      SNAME    )
              COAST STEEL
```

16. List the suppliers and the projects where the supplier supplies a part which is used in the project.

```
(SNAME PROJECT# : (SNAME SUPPLIES PART#) &
                  (PROJECT# USES PART#))
   REL82 (             SNAME          PROJECT#    )
              APPOLLO SHEET METAL        4
              APPOLLO SHEET METAL        1
              APPOLLO SHEET METAL        2
              APPOLLO SHEET METAL        5
              APPOLLO SHEET METAL        3
              APPOLLO SHEET METAL        6
                 VALLEY STEEL            1
                 VALLEY STEEL            2
                 VALLEY STEEL            5
                 VALLEY STEEL            3
                 VALLEY STEEL            4
                 VALLEY STEEL            6
              PEARSON IRON WORKS         4
              PEARSON IRON WORKS         1
              PEARSON IRON WORKS         2
              PEARSON IRON WORKS         5
              PEARSON IRON WORKS         3
              PEARSON IRON WORKS         6
                  COAST STEEL            4
                  COAST STEEL            1
                  COAST STEEL            2
                  COAST STEEL            5
                  COAST STEEL            3
                  COAST STEEL            6
```

17. List the suppliers and the projects where the supplier supplies a part to the project.

```
   (SNAME PROJECT# : (SNAME SUPPLIES PART# SUPPLIES-TO PROJECT#))
      REL85 (          SNAME          PROJECT#    )
                    COAST STEEL          6
                APPOLLO SHEET METAL      6
                    COAST STEEL          5
                APPOLLO SHEET METAL      5
                    COAST STEEL          4
                APPOLLO SHEET METAL      4
                PEARSON IRON WORKS       4
                    COAST STEEL          3
                   VALLEY STEEL          3
                APPOLLO SHEET METAL      3
                    COAST STEEL          2
                APPOLLO SHEET METAL      2
                PEARSON IRON WORKS       1
                   VALLEY STEEL          1
                APPOLLO SHEET METAL      1
                    COAST STEEL          1
```

18. List the suppliers who do not supply part 3.

```
   (SNAME : SNAME & ¬(SNAME SUPPLIES PART#) & (EQ PART# 3))
      REL102 (          SNAME    )
                 PEARSON IRON WORKS
```

19. Inform the system that all Vancouver suppliers are on strike.

```
   (VANCOUVER SUPPLIERS ARE ON STRIKE)
      O.K.
```

20. List the suppliers who are currently supplying parts.

```
   (SNAME : (SNAME SUPPLIES PART#))
      REL7 (          SNAME    )
              APPOLLO SHEET METAL
                 VALLEY STEEL
              PEARSON IRON WORKS
```

21. List the suppliers who do not supply part 3.

```
   (SNAME : SNAME & ¬(SNAME SUPPLIES PART#) & (EQ PART# 3))
      REL20 (          SNAME    )
                    COAST STEEL
                PEARSON IRON WORKS
```

APPENDIX 7 - RELATIONAL ALGEBRA

```
 ; PROJECT
 ; RETURNS THE PROJECTION OF THE RELATION ON THE DOMAINS WHOSE
 ; POSITIONS ARE GIVEN IN THE LIST A.


(DEFUN PROJECT (RELATION A)
   (PROG (NEWREL DOMAINS NEWDOMAINS TNAME NEWDATA PROJECTION)
      (SETQ NEWREL (GENSYM1 'REL))
      (PUT NEWREL '#TUPLES 0)
      (SETQ DOMAINS (GET RELATION 'DOMAINS))
      (PUT NEWREL 'DOMAINS (CHOOSE DOMAINS A))
      (MAPC '(LAMBDA (TUPLE)
               (SETQ TUPLE (GET TUPLE 'DATA))
               (SETQ NEWDATA (CHOOSE TUPLE A))
               (COND ( (NOT (MEMBER NEWDATA PROJECTION))
                        (ADDTUPLE NEWDATA NEWREL)
                        (SETQ PROJECTION (CONS NEWDATA PROJECTION))
                      ))
             )
            (GET RELATION 'TUPLES) )
   (RETURN NEWREL)
   )
)



 ; JOIN
 ; RETURNS THE THETA-JOIN OF R1 WITH R2.  THETA IS EXPRESSED
 ; BY PREDICATE, AN **ARBITRARY** LISP EXPRESSION WHICH MUST
 ; BE SATISFIED BEFORE TWO TUPLES CAN BECOME PART OF THE JOIN.


(DEFUN JOIN (R1 R2 PREDICATE)
   (PROG (NEWREL TNAME)
      (SETQ NEWREL (GENSYM1 'REL))
      (PUT NEWREL '#TUPLES 0)
      (PUT NEWREL 'DOMAINS (APPEND (GET R1 'DOMAINS)
                                   (GET R2 'DOMAINS)))
      (MAPC '(LAMBDA (T1)
         (SETQ T1 (GET T1 'DATA))
         (MAPC '(LAMBDA (T2)
            (SETQ T2 (GET T2 'DATA))
            (COND ( (EVAL PREDICATE)
                     (ADDTUPLE (APPEND T1 T2) NEWREL) ))
         ) (GET R2 'TUPLES))
     ) (GET R1 'TUPLES))
   (RETURN NEWREL)
   )
)



 ; RESTRICT
 ; RESTRICT RETURNS A NEW RELATION WHICH CONSISTS OF THOSE
 ; TUPLES IN "RELATION" WHICH SATISFY THE PREDICATE.


(DEFUN RESTRICT (RELATION PREDICATE)
   (PROG (NEWREL T1)
      (SETQ NEWREL (GENSYM1 'REL))
      (PUT NEWREL '#TUPLES 0)
      (PUT NEWREL 'DOMAINS (GET RELATION 'DOMAINS))
      (MAPC '(LAMBDA (TNAME)
               (SETQ T1 (GET TNAME 'DATA))
               (COND ( (EVAL PREDICATE)
                        (PUT NEWREL 'TUPLES (CONS TNAME
                                                  (GET NEWREL 'TUPLES)))
                     ))
             )
            (GET RELATION 'TUPLES))
      (PUT NEWREL '#TUPLES (LENGTH (GET NEWREL 'TUPLES)))
   (RETURN NEWREL)
   )
)



 ; DIVIDE
 ; A IS A LIST OF THE DOMAIN NUMBERS OF RELATION R1 TO BE
 ; DIVIDED BY DOMAIN NUMBERS B OF R2.
 ;
 ; DIVIDE WORKS AS FOLLOWS.  FOR EACH ROW R IN R1, IF EACH
 ; TUPLE IN THE PROJECTION OF R2 ON B IS A MEMBER OF THE
 ; IMAGE SET OF THE PROJECTION OF ABAR ON R, UNDER R1, THEN
 ; KEEP THE PROJECTION OF ABAR ON R.


(DEFUN RDIVIDE (R1 R2 A B)
   (PROG (REL SONB ISET ABAR ALL USED ABARDATA)
      (SETQ SONB (PROJECT R2 B))
      (SETQ ABAR (SETDIF (GENLIST (GET R1 'DOMAINS)) A))
      (SETQ REL (GENSYM1 'REL))
      (PUT REL '#TUPLES 0)
      (PUT REL 'DOMAINS (CHOOSE (GET R1 'DOMAINS) ABAR))
      (MAPC '(LAMBDA (T1)
         (SETQ T1 (GET T1 'DATA))
         (SETQ ABARDATA (CHOOSE T1 ABAR))
         (COND ( (NOT (MEMBER ABARDATA USED))
         (SETQ ISET (IMAGESET (CHOOSE T1 ABAR)
                                R1
```

```
                                         ABAR
                                         A))
                (SETQ ALL T)
                (SETQ USED (CONS ABARDATA USED))
                (MAPC '(LAMBDA (T2)
                   (SETQ T2 (GET T2 'DATA))
                   (COND ( (NOT (MEMBER T2 ISET))
                           (SETQ ALL NIL)
                           (UNEVAL 'MAPC NIL)) )
                      )
                     (GET SCNB 'TUPLES))
                (AND ALL (ADDTUPLE ABARDATA REL))
                ))
                    )
                   (GET R1 'TUPLES))
         (RETURN REL)
         )
   )



   (DEFUN RUNICN (L)
     : THIS PROCEDURE WILL RETURN A RELATION WHICH IS THE UNION
     : OF ALL THE RELATIONS IN L.
     (PRCG (REL)
       (SETQ REL (CAR L))
       (MAPC '(LAMBDA (R)
              (COND ( (NULL R) (RETURN REL))
                    ( T (SETQ REL (RUNION* REL R))) )
             )
             (CDR L))
      (RETURN REL))
   )



   (DEFUN RUNION* (R1 R2)
     : THIS PROCEDURE WILL RETURN A RELATION WHICH IS THE UNION
     : OF R1 AND R2.  THE NEW DOMAIN NAMES ARE THOSE OF R1.
      (PROG (REL UNION TUPLE)
        (CCND ( (NOT (EQ (LENGTH (GET R1 'DOMAINS))
                         (LENGTH (GET R2 'DOMAINS)) ))
                (RETURN NIL)))
        (SETQ REL (GENSYM1 'REL))
        (PUT REL 'DOMAINS (GET R1 'DOMAINS))
        (PUT REL '#TUPLES 0)
        (MAPC '(LAMBDA (X)
           (SETQ TUPLE (GET X 'DATA))
           (COND ( (NOT (MEMBER TUPLE UNION))
                   (ADDTUPLE TUPLE REL)
                   (SETQ UNION (CONS TUPLE UNION))) )
              )
              (APPEND (GET R1 'TUPLES) (GET R2 'TUPLES)) )
        (RETURN REL) )
   )



   (DEFUN RCROSS (L)
     : THIS PROCEDURE WILL RETURN A NEW RELATION WHICH IS THE
     : CROSS PRODUCT OF ALL THE RELATIONS IN L.
     (PROG (REL)
       (SETQ REL (CAR L))
       (MAPC '(LAMBDA (R)
              (COND ( (NULL R) (RETURN REL))
                    ( T (SETQ REL (RCROSS* REL R))) )
            )
            (CDR L))
     (RETURN REL)
      )
   )



   (DEFUN RCROSS* (R1 R2)
     : THIS PROCEDURE WILL RETURN A NEW RELATION WHICH IS THE
     : CROSS PRODUCT OF R1 AND R2.
     (PROG (REL)
        (SETQ REL (GENSYM1 'REL))
        (PUT REL 'DOMAINS (APPEND (GET R1 'DOMAINS)
                                  (GET R2 'DOMAINS)))
        (PUT REL '#TUPLES 0)
        (MAPC '(LAMBDA (T1)
          (MAPC '(LAMBDA (T2)
                (ADDTUPLE (APPEND (GET T1 'DATA)
                                  (GET T2 'DATA)) REL)
              )
             (GET R2 'TUPLES))
            )
           (GET R1 'TUPLES))
     (RETURN REL) )
   )


   (DEFUN RINTERSECT (L)
     : THIS ROUTINE WILL RETURN THE INTERSECTION OF ALL RELATIONS IN L
     (PRCG (REL)
       (SETQ REL (CAR L))
       (MAPC '(LAMBDA (R)
              (SETQ REL (PROJECT (JOIN REL R '(ECUAL T1 T2))
                                         (GENLIST (GET REL 'DOMAINS))))
              )
           (CDR L))
    (RETURN REL))
  )



 (DEFUN RDIFF (R1 R2)
   ; THIS PROCEDURE RETURNS R1 - R2.
   (SETQ REL (GENSYM1 'REL))
   (PUT REL 'DOMAINS (GET R1 'DOMAINS))
   (PUT REL '#TUPLES 0)
   (MAPC '(LAMBDA (T1)
            (OR (IN T1 R2)
                (ADDTUPLE (GET T1 'DATA) REL))
          )
         (GET R1 'TUPLES))
   REL
  )



 (DEFUN IN (TUPLE R)
   ; THIS PROCEDURE RETURNS T IF TUPLE IS IN THE RELATION R.
   (PROG ( )
     (MAPC '(LAMBDA (T2)
              (COND ( (EQUAL (GET T2 'DATA)
                             (GET TUPLE 'DATA))
                        (RETURN T)) )
            )
           (GET R 'TUPLES))
    )
  )



 ; IMAGESET
 ; PROVIDES THE IMAGE SET OF X UNDER THE RELATION R, WHERE A
 ; IS A LIST OF DOMAIN NUMBERS CORRESPONDING TO X, AND ABAR
 ; IS THE COMPLEMENT OF A.


 (DEFUN IMAGESET (X R A ABAR)
    (PROG (ISET)
       (MAPC '(LAMBDA (TUPLE)
           (SETQ TUPLE (GET TUPLE 'DATA))
           (COND ( (EQUAL (CHOOSE TUPLE A) X)
                 (SETQ ISET (CONS (CHOOSE TUPLE ABAR)
                                  ISET)) ))
             )
            (GET R 'TUPLES))
    (RETURN ISET)
    )
 )



 ; SETDIF
 ; RETURNS SET1-SET2.


 (DEFUN SETDIF (SET1 SET2)
    ; (REVERSE (EXCLUDE SET2 SET1))
    (PROG (DIFF)
     (MAPC '(LAMBDA (E)
        (OR (MEMBER E SET2) (SETQ DIFF (CONS E DIFF)))
            )
           SET1)
    (RETURN (REVERSE DIFF))
 )
 )






 ; ADDTUPLE
 ; ADDS THE TUPLE TO THE RELATION, AND RETURNS THE NAME OF THE TUPLE


 (DEFUN ADDTUPLE (TUPLE RELATION)
    (PROG (TNAME)
       (PUT RELATION 'TUPLES (CONS (SETQ TNAME (GENSYM1 'T))
                                   (GET RELATION 'TUPLES)))
       (PUT RELATION '#TUPLES (ADD1 (GET RELATION '#TUPLES)))
       (PUT TNAME 'DATA TUPLE)
    (RETURN TNAME)
     )
 )



 ; CHOOSE
 ; RETURNS THE PROJECTION OF THE TUPLE ON THE DOMAINS WHOSE
 ; POSITIONS ARE GIVEN BY THE LIST A.


(DEFUN CHOOSE (TUPLE A)
   (MAPCAR '(LAMBDA (DOMAIN) (ELEM TUPLE DOMAIN))
           A)
)



 ; ELEM
 ; RETURNS THE NTH ELEMENT OF A LIST.
 ; IF POSITION IS A LIST OF DOMAIN NUMBERS, ALL CORRESPONDING ELEMENTS
 ; WILL BE RETURNED.


(DEFUN ELEM  (LIST POSITION)
   (COND ( (NUMBERP POSITION)
           (CAR (NTH LIST POSITION)) )
         ( T (MAPCAR '(LAMBDA (N)
                          (CAR (NTH LIST N))
                       )
                     POSITION) ))
)



 ; GENLIST
 ; GENERATES A LIST OF NUMBERS FROM 1 TO THE NUMBER OF DOMAINS


(DEFUN GENLIST (DOMAINS)
   (PROG (INC)
      (SETQ INC 0)
      (MAPCAR '(LAMBDA (X) (SETQ INC (ADD1 INC))) DOMAINS)
   )
)


    (OPEN (IMPLODEBUFFER 100))
END OF FILE
```

```
(DEFUN REDUCE (QUERY)
  ; THIS ROUTINE WILL TAKE A QUERY IN THE RELATIONAL CALCULUS
  ; AND REDUCE IT TO A SEQUENCE OF OPERATIONS IN THE RELATIONAL
  ; ALGEBRA.  IT IS MODELLED ON THE IMPROVED REDUCTION ALGORITHM
  ; OF PALERMO.
  (PROG (RESULT GLOBAL_RANGE)
  ; FORM THE GLOBAL RANGE FOR EACH VARIABLE IN THE QUERY.
   (MAPC '(LAMBDA (VAR)
             (PUT VAR 'RANGE (SETQ GLOBAL_RANGE (RANGE VAR)))
             (COND ( (NULL (GET GLOBAL_RANGE 'TUPLES))
                       (PRIN1 '"THE GLOBAL RANGE FOR") (PRIN1 VAR)
                       (PRIN1 '"IS EMPTY") (TERPRI) (RETURN NIL)) )
             (AND TRACE (PRIN1 '"THE REDUCED GLOBAL RANGE FOR")
                          (PRIN1 VAR) (PRIN1 'IS:)
                          (PRINTREL GLOBAL_RANGE) )
           )
          (GET QUERY 'VARS))
  ; CONSTRUCT THE COMPONENTS C(I), EACH DEFINED BY THETA(I)
   (MAPC '(LAMBDA (THETAI)
             (SETQ RESULT (RUNION (LIST (FORM_CI THETAI) RESULT)))
           )
          (GET QUERY 'THETA))
  ; APPLY THE OPERATIONS OF DIVISION AND PROJECTION TO THE RESULT
  ; TO OBTAIN THE RELATION TP.
   (MAPC '(LAMBDA (VAR)
             (SET VAR (GET VAR 'RANGE))
             (COND ( (GET VAR 'QUANTIFIER)
                        (SETQ RESULT (PORD RESULT VAR))
                        (AND TRACE (PRINTREL RESULT))
                        (MAPC '(LAMBDA (V)
                          (COND ( (GREATERP (GET V 'STARTS)
                                              (GET VAR 'STARTS))
                                 (PUT V 'STARTS (SUB (GET V 'STARTS)
                                   (LENGTH (GET (EVAL VAR) 'DOMAINS)) )) ) )
                              )
                            (GET QUERY 'VARS)) )
                    (T (UNEVAL 'MAPC NIL)) )
           )
          (GET QUERY 'VARS))
  ; PROJECT TP ONTO THE DESIRED OUTPUT DOMAINS
   (PRIN1 '"THE RESPONSE RELATION IS:")
   (RETURN (PRINTREL (PROJECT RESULT (GENTARGET (GET QUERY 'TARGET)))))
))



(DEFUN FORM_CI (THETAI)
  ; THIS ROUTINE WILL FORM THE SUBSET CI DEFINED BY THETAI
  ; IT USES THE GOLBAL VARIABLE RANGE_LIST
  (PROG (L_OF_CORES POSS_TERMS USEDVARS CORE JOIN_TERM
          POSITION CORE_RANGE STARTS RANGE_LIST V INO)
   (SETQ STARTS 1)
  ; FORM THE REDUCED LOCAL RANGE FROM THE REDUCED GLOBAL RANGE.
   (MAPC '(LAMBDA (VAR)
     (SET VAR (GET VAR 'RANGE))
     (FORM_RR VAR THETAI)
     (AND TRACE
          (PRIN1 '"THE REDUCED LOCAL RANGE FOR") (PRIN1 VAR)
          (PRIN1 'IN) (PRIN1 THETAI) (PRIN1 'IS) (PRINTREL (EVAL VAR)) )
          )
         (GET QUERY 'VARS))
  ; CREATE A LIST OF THE RANGES USED IN THETAI, WITH THOSE HAVING
  ; THE SMALLEST NUMBER OF TUPLES COMING FIRST.
   (SETQ RANGE_LIST (RANGES THETAI))
   LOOP (SETQ POSS_TERMS NIL)
        (SETQ CORE (EVAL (CAR RANGE_LIST)))
        (AND TRACE
             (PRIN1 '"THE FIRST ELEMENT IN THE CORE OF")
             (PRIN1 THETAI) (PRIN1 'IS:) (PRINTREL CORE) )
        (PUT (CAR RANGE_LIST) 'STARTS STARTS)
        (SETQ STARTS (ADD STARTS
                          (LENGTH (GET (EVAL (CAR RANGE_LIST)) 'DOMAINS))))
        (SETQ USEDVARS (CONS (UNCONS RANGE_LIST RANGE_LIST) NIL))
  ; CONSTRUCT A LIST OF TERMS WHICH CAN BE USED TO JOIN THE
  ; CORE WITH A NEW RANGE.
   LOOP2 (SETQ POSS_TERMS (UNION (INTERSECT
                                 (GET (CAR USEDVARS) 'USED_IN_TERMS)
                                 (GET THETAI 'TERMS))
                               POSS_TERMS))
  ; IF THERE ARE NO MORE JOIN TERMS CONNECTED TO THE CORE,
  ; START AGAIN.
        (COND ( (NULL POSS_TERMS)
                 (SETQ L_OF_CORES (CONS CORE L_OF_CORES))
                 (GO LOOP)) )
  ; NOW THAT WE KNOW WHICH JOIN TERMS CAN BE USED, CHOOSE THE ONE WHICH
  ; INVOLVES THE SMALLEST RELATION
        (MAPC '(LAMBDA (VAR)
            (COND ( (SETQ JOIN_TERM (INTERSECT POSS_TERMS
                                              (GET VAR 'USED_IN_TERMS)))
                  (SETQ JOIN_TERM (CAR JOIN_TERM))
                  (SETQ RANGE_LIST (DELETE VAR RANGE_LIST))
                  (SETQ POSS_TERMS (DELETE JOIN_TERM POSS_TERMS))
                  (SETQ NEW_RANGE VAR)
  ; FIND ALL OTHER TERMS WHICH INVOLVE ONLY THESE TWO RELATIONS.
                  (MAPC '(LAMBDA (TERM)
                    (COND ( (EQUAL (GET TERM 'VARS) (GET JOIN_TERM 'VARS))
                             (SETQ POSS_TERMS (DELETE TERM POSS_TERMS))
                             (SETQ JOIN_TERM (LIST 'AND JOIN_TERM TERM)) ))
                          )
                        POSS_TERMS)
```

```
                (COND ( (EQ (CAR (GET JOIN_TERM 'VARS)) NEW_RANGE)
                       (SETQ CORE_RANGE (CADR (GET JOIN_TERM 'VARS))) )
                     ( T (SETQ CORE_RANGE (CAR (GET JOIN_TERM 'VARS)))) )
                (PUT VAR 'STARTS STARTS)
                (SETQ STARTS (ADD STARTS (LENGTH (GET (EVAL VAR) 'DOMAINS))))
                (SETQ USEDVARS (CONS VAR USEDVARS))
                (UNEVAL 'MAPC NIL)) )
               )
              RANGE_LIST)
  : JOIN THE CORE TO THE NEW RANGE
  (SETQ CORE (JOIN CORE (EVAL NEW_RANGE)
             (FIX* (EVAL JOIN_TERM) CORE_RANGE 'T1 NEW_RANGE 'T2)))
            (AND TRACE (PRIN1 '"JOINING CORE WITH")
                (PRIN1 NEW_RANGE) (PRIN1 'YIELDS:)
                (PRINTREL CORE) )
  : SEE IF WE ARE DONE
  (COND ( RANGE_LIST (GO LOOP2))
       ( L_OF_CORES (RETURN (RCROSS (REVERSE L_OF_CORES))))
       ( T (RETURN CORE)))
))



(DEFUN FIX* (JOIN_TERM CORE_RANGE T1 OTHER_RANGE T2)
  : THIS ROUTINE WILL CHANGE ALL OCCURRENCES OF (OTHER_RANGE N)
  : IN JOIN_TERM TO (ELEM T2 N), AND ALL OCCURRENCES OF
  : (CORE_RANGE N) TO (ELEM T1 CORE-N).
  (COND ( (NULL JOIN_TERM) NIL)
       ( (ATOM (CAR JOIN_TERM))
         (FIX* (CDR JOIN_TERM) CORE_RANGE T1 OTHER_RANGE T2))
       ( (EQ (CAAR JOIN_TERM) OTHER_RANGE)
         (RPLACA JOIN_TERM (LIST 'ELEM
                              T2
                             (NEWDOMAIN OTHER_RANGE (CADAR JOIN_TERM))))
         (FIX* (CDR JOIN_TERM) CORE_RANGE T1 OTHER_RANGE T2))
       ( (EQ (CAAR JOIN_TERM) CORE_RANGE)
         (RPLACA JOIN_TERM (LIST 'ELEM  T1
                             (SUB1 (ADD (GET CORE_RANGE 'STARTS)
                         (NEWDOMAIN CORE_RANGE (CADAR JOIN_TERM)) ))))
         (FIX* (CDR JOIN_TERM) CORE_RANGE T1 OTHER_RANGE T2))
       ( T (FIX* (CAR JOIN_TERM) CORE_RANGE T1 OTHER_RANGE T2)
          (FIX* (CDR JOIN_TERM) CORE_RANGE T1 OTHER_RANGE T2)
       ))
  JOIN_TERM
)



(DEFUN PORD (REL VAR)
  : THIS ROUTINE WILL EITHER PROJECT REL ON ALL ITS DOMAINS,
  : EXCEPT THE ONES CONTAINED IN THE RELATION SPECIFIED BY VAR,
  : OR IT WILL DIVIDE REL BY THE RELATION SPECIFIED BY VAR.
  (PROG (DOM#S PDOM#S DOMAIN#S)
  (SETQ DOMAIN#S (GENLIST (GET REL 'DOMAINS)))
  (SETQ DOM#S (MAPCAR '(LAMBDA (D#)
               (ADD (SUB1 D#) (GET VAR 'STARTS))
                    )
                   (GENLIST (GET (EVAL VAR) 'DOMAINS)) )).
  (COND ( (EQ (GET VAR 'QUANTIFIER) 'EXISTS)
         (MAPC '(LAMBDA (X)
           (OR (MEMQ X DOM#S)
              (SETQ PDOM#S (CONS X PDOM#S)))
               )
              DOMAIN#S)
         (AND TRACE (PRIN1 '"PROJECTING OFF") (PRIN1 VAR)
                  (PRIN1 'YIELDS:))
         (RETURN (PROJECT REL (REVERSE PDOM#S))) )
       ( T (AND TRACE (PRIN1 '"DIVISION BY") (PRIN1 VAR)
               (PRIN1 'YIELDS:))
          (RETURN (RDIVIDE REL
                 (EVAL VAR)
                 DOM#S
                 (GENLIST (GET (EVAL VAR) 'DOMAINS)) ))) )
  )
)



(DEFUN FORM_RR (VAR THETAI)
  : THIS ROUTINE FORMS THE REDUCED LOCAL RANGE FOR VAR IN
  : THETAI.  THIS AMOUNTS TO RESTRICTING VAR TO THOSE CASES
  : DEFINED BY TERMS IN THETAI WHICH CONTAIN ONLY 1 TUPLE
  : VARIABLE.
  (PROG (V)
   (MAPC '(LAMBDA (TERM)
     (COND ( (AND (EQ (LENGTH (GET TERM 'VARS)) 1)
                 (EQ (CAR (GET TERM 'VARS)) VAR))
            (SETQ V (CAR (GET TERM 'VARS)))
            (SET V (RESTRICT (EVAL V)
                           (FIX* (EVAL TERM) NIL NIL V 'T1)))
            (PUT THETAI 'TERMS (DELETE TERM (GET THETAI 'TERMS)))
          ))
          )
         (GET THETAI 'TERMS))
  )
)
```

```


 (DEFUN GENTARGET (TARGET_LIST)
   ; THIS PROCEDURE RETURNS A LIST OF THE DOMAIN NUMBERS
   ; UPON WHICH THE RESPONSE RELATION SHOULD BE PROJECTED.
   (MAPCAN '(LAMBDA (ELEMENT)
              (LIST (ADD (NEWDOMAIN (CAR ELEMENT) (CADR ELEMENT))
                         (SUB1 (GET (CAR ELEMENT) 'STARTS))) )
            )
           TARGET_LIST)
 )



 (DEFUN ORDER (VARS)
   ; THIS ROUTINE TAKES A LIST OF VARIABLES, AND ORDERS THEM
   ; WITH THE ONES WHICH HAVE THE SMALLEST RANGES COMING FIRST.
   (PROG (ANS LEN L)
    (SETQ ANS (LIST (CAR VARS)))
    (MAPC '(LAMBDA (V)
       (SETQ LEN (GET (EVAL V) '#TUPLES))
       (COND ( (NOT (GREATERP LEN (GET (EVAL (CAR ANS)) '#TUPLES) ))
               (SETQ ANS (CONS V ANS)) )
             ( (NOT (LESSP LEN (GET (EVAL (CAR (LAST ANS))) '#TUPLES) ))
               (SETQ ANS (APPEND ANS (LIST V))) )
             ( T (SETQ L (CONS NIL ANS))
                 (MAPC '(LAMBDA (ELT)
                    (COND ( (LESSP LEN (GET (EVAL ELT) '#TUPLES))
                            (RPLACD L (CONS V (CDR L)))
                            (UNEVAL 'MAPC NIL))
                          ( T (SETQ L (CDR L))) )
                          )
                       ANS) ))
        )
       (CDR VARS))
   (RETURN ANS))
 )



 (DEFUN NEWDOMAIN (VAR OLD#)
   ; GIVEN A VARIABLE, AND A DOMAIN NUMBER IN THE GLOBAL RANGE
   ; FOR THAT VARIABLE, THIS ROUTINE WILL RETURN THE DOMAIN NUMBER
   ; OF THE SAME DOMAIN IN THE GLOBAL OR LOCAL REDUCED RANGE.
   (COND ( (NUMBERP OLD#)
           (SUB (LENGTH (GET VAR 'REFDOMAINS))
                (LENGTH (CDR (MEMQ OLD# (GET VAR 'REFDOMAINS))))))
         ( T (MAPCAR '(LAMBDA (N)
                (SUB (LENGTH (GET VAR 'REFDOMAINS))
                     (LENGTH (CDR (MEMQ N (GET VAR 'REFDOMAINS)))))
                     )
                     OLD#)) )
 )



 (DEFUN RANGES (THETAI)
   ; THIS ROUTINE WILL RETURN A LIST OF THOSE RANGES USED IN
   ; THETAI, WITH THOSE HAVING THE SMALLEST NUMBER OF TUPLES
   ; APPEARING FIRST.
   (PROG (RANGE_LIST)
     (MAPC '(LAMBDA (T)
               (SETQ RANGE_LIST (UNION (GET T 'VARS) RANGE_LIST))
             )
           (GET THETAI 'TERMS))
     (RETURN (ORDER RANGE_LIST))
   )
 )



 (DEFUN RANGE (VAR)
   ; THIS ROUTINE WILL CREATE THE REDUCED GLOBAL RANGE FOR VAR.
   (PROG (FORM UN)
     (SETQ FORM (GET VAR 'RANGE))
     (COND ( (ATOM FORM) (SETQ UN FORM))
           ( (EQ (CAR FORM) 'AND)
             (SETQ UN (EVAL (PROCESS_AND (CDR FORM) VAR))) )
           ( (EQ (CAR FORM) 'OR)
             (MAPC '(LAMBDA (CONJ)
                       (COND ((ATOM CONJ) (SETQ CONJ (LIST CONJ)))
                             ( T (SETQ CONJ (CDR CONJ))) )
                       (SETQ UN (RUNION (LIST (EVAL (PROCESS_AND CONJ VAR))
                                              UN)))
                    )
                   (CDR FORM)) )
           ( T (PRINT '"ILLEGAL RANGE FORMULA")) )
     (RETURN (PROJECT UN (GET VAR 'REFDOMAINS))) )
 )

```

```
          (DEFUN PROCESS_AND (ARGS VAR)
             ; THIS ROUTINE WILL PROCESS THE ELEMENTS OF A CONJUNCTION IN
             ; THE RANGE FORMULA OF VAR.
            (PROG (FORM INT_TERMS DIFF_TERMS REST_TERMS SAVE)
              (MAPC '(LAMBDA (ELT)
                 (COND ( (ATOM ELT)
                           (SETQ INT_TERMS (CONS ELT INT_TERMS)) )
                         ( (AND (EQ (CAR ELT) 'NOT)
                                (ATOM (CADR ELT)))
                           (SETQ DIFF_TERMS (CONS (CADR ELT) DIFF_TERMS)) )
                         ( T (SETQ REST_TERMS (CONS ELT REST_TERMS))) )
                        )
                       ARGS)
              (SETQ SAVE (GET VAR 'REFDOMAINS))
              (PUT VAR 'REFDOMAINS (GENLIST (GET (CAR INT_TERMS) 'DOMAINS)))
              (COND ( (GREATERP (LENGTH INT_TERMS) 1)
                        (SETQ FORM (LIST 'RINTERSECT (LIST 'QUOTE INT_TERMS))) )
                       ( T (SETQ FORM (LIST 'QUOTE (CAR INT_TERMS)))) )
              (MAPC '(LAMBDA (T)
                         (SETQ FORM (LIST 'RDIFF FORM (LIST 'QUOTE T)))
                        )
                       DIFF_TERMS)
              (COND ( (GREATERP (LENGTH REST_TERMS) 0)
                        (SETQ FORM (LIST 'RESTRICT FORM (LIST 'QUOTE (FIX* (CONS 'AND REST_TERMS)
                                                                NIL NIL
                                                                VAR 'T1)))))
                        (PUT VAR 'REFDOMAINS SAVE) ))
            (RETURN FORM))
          )
END OF FILE
```

```
(THCONSE LISP2 (X R Y) (PROVE* $?X $?R $?Y)
  (THDO (THAND TRACE
               (PRIN1* '"ATTEMPT TO PROVE")
               (THCOND ( (THASVAL $?X) (PRIN1* $?X)) (T (PRIN1* '$?X)) )
               (THCOND ( (THASVAL $?R) (PRIN1* $?R)) (T (PRIN1* '$?R)) )
               (THCOND ( (THASVAL $?Y) (PRIN1* $?Y)) (T (PRIN1* '$?Y)) )
               (TERPRI)) )
  (THDO (THFIND 1 $?REL (REL) ($G (PROVE $?X $?R $?Y $?REL) (THUSE PROVE-X-R-Y)))
)).


(THCONSE PROVE-X-R-Y (X R Y REL VALID RESULT USED) (PROVE $?X $?R $?Y $?REL)
   (THSETQ $?USED (LIST NIL))
   (THDO (SETQ #RESTRICT# NIL #EXTRA# NIL))
   (THCOND ( (THAND ($G ($?X $?R $?Y $?REL)) ($G (ACTIVE $?REL) $T))
                         ($G (UPDATE-REST $?X $?R $?Y $?REL) $T)
              (THSETQ $?RESULT ($G (F-ALL $?X $?R $?Y) (THUSE FIND-RELS)))
              (SETQ #RESULT# $?RESULT) )
           ( ($G ($?X $?R $?Y $?REL) $T)
              (SETQ #RESULT# THVALUE) )
           ( (THAND (THSETQ $?VALID T)
                    (PRINT* '"ABOUT TO TRY FOR A NON-VALID PATH")
                    (THSETQ $?USED (LIST NIL))
                    (THDO (SETQ #RESTRICT# NIL))
                    ($G ($?X $?R $?Y $?REL) $T)
                    (SETQ #RESULT# THVALUE) ) ))
  (UPDATE-RESULT)
  (SETQ #RESULT# (LIST (LIST $?X (CAR #RESULT#))
                         (LIST $?Y (CAR (LAST #RESULT#)) )
                         #RESULT#))
  (THCOND ( #EXTRA# (SETQ #RESULT# (APPEND #EXTRA# #RESULT#)))
  ( (THSUCCEED)) )
  (THFAIL THEOREM)
 )


(THCONSE RELATE-X-Y (X Y R R2 RP A REL REL2 REL3 VAR A2 RELATION RES)
                ($?X $?R $?Y $?RELATION)
  (THPROG (LHS RHS)
  (THCOND ( (THAND (THASVAL $?X) (THASVAL $?Y))
            (THCOND ( (MEMBER (LIST $?X $?Y) $?USED) (THFAIL THEOREM))
                    ( (THVSETQ $?USED (CONS (LIST $?X $?Y) $?USED)) ) ) )
          ( (THSUCCEED)) )
  (THCOND ( (THNOT (THASVAL $?R))
            (THSETQ $?VAR T))
          ( (THSETQ $?R2 $?R)) )
  (THOR (THAND (THASVAL $?Y)
               (THOR (THAND ($G ($?X $?R $?A $?REL)) ($G (OK $?R) $T)
                            ($G (ACTIVE $?REL) $T)
                            ($G (UPDATE-REST $?X $?R $?A $?REL) $T)
                            ($G (F-ALL $?X $?R $?A) (THUSE FIND-RELS))
                            (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#))
                     (THAND (THNOT (THAND ($G ($?A2 $?R2 $?Y $?REL3)) ($G (OK $?R2) $T) ))
                            ($G ($?X $?R $?A $?REL) $T)
                            (SETQ #RESULT# THVALUE)
                            (THSETQ $?LHS #RESULT#)
                            ($G (ACTIVE $?REL) $T)
                            ($G (OK $?R) $T) ) )
               (THSETQ $?LHS (APPEND $?LHS (LIST $?A)))
               (PRINT* (LIST $?REL 'SAYS $?X $?R $?A))
               (THCOND ( (THASVAL $?Y) (NOT (EQ $?Y $?A))
                         (PRINT* (LIST '"TRY TO RELATE" $?A 'AND $?Y)) ))
               (THCOND ( (THAND ($G ($?A $?RP $?Y $?REL2)) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                         ($G (UPDATE-REST $?A $?RP $?Y $?REL2) $T)
                         ($G (F-ALL $?A $?RP $?Y) (THUSE FIND-RELS))
                         (SETQ #RESULT# THVALUE) (THSETQ $?RHS #RESULT#)
                         (PRINT* (LIST $?REL2 'SAYS $?A $?RP $?Y)) )
                       ( (THAND ($G ($?Y $?RP $?A $?REL2)) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                         ($G (UPDATE-REST $?Y $?RP $?A $?REL2) $T)
                         ($G (F-ALL $?Y $?RP $?A) (THUSE FIND-RELS))
                         (SETQ #RESULT# THVALUE) (THSETQ $?RHS #RESULT#)
                         (PRINT* (LIST $?REL2 'SAYS $?Y $?RP $?A)) )
                       ( (THAND ($G ($?A $?RP $?Y $?REL2) $T) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                         (SETQ #RESULT# THVALUE)
                         (THSETQ $?RHS #RESULT#)
                         (PRINT* (LIST $?REL2 'SAYS $?A $?RP $?Y)) )
                       ( (THAND ($G ($?Y $?RP $?A $?REL2) $T) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                         (SETQ #RESULT# THVALUE)
                         (THSETQ $?RHS #RESULT#)
                         (PRINT* (LIST $?REL2 'SAYS $?Y $?RP $?A)) ))
               (THOR ($G ($?A AND $?Y ARE T-TRANS IN $?REL2))
                     ($G ($?REL2 IS T-TRANS))
                     (THAND (THASVAL $?VAR) (THSETQ $?R (GENSYM 'ISR))) )
               (THOR ($G ($?A DETERMINES $?X $?REL))
                     ($G ($?A DETERMINES $?Y $?REL2))
                     (THASVAL $?VALID) )
               (THSETQ $?VAR (GENSYM))
               ($A ($?X $?R $?Y $?VAR))
               (THCOND ( (THAND ($G ($?A DETERMINES $?X $?REL))
                                ($G ($?A DETERMINES $?Y $?REL2)) )
                         ($A ($?X DETERMINES $?Y $?VAR))
                         ($A ($?Y DETERMINES $?X $?VAR)) )
                       ( (THSUCCEED)) )
               (THSETQ $?RELATION $?VAR)
        )
        (THAND (THASVAL $?X)
               (THCOND ( (THAND ($G ($?A $?R $?Y $?REL)) ($G (ACTIVE $?REL) $T))
                         ($G (UPDATE-REST $?A $?R $?Y $?REL) $T)
                         ($G (F-ALL $?A $?R $?Y) (THUSE FIND-RELS))
                         (SETQ #RESULT# THVALUE) (THSETQ $?RHS #RESULT#))
                         ( (THAND ($G ($?A $?R $?Y $?REL) $T) ($G (ACTIVE $?REL) $T))
                            (SETQ #RESULT# THVALUE)
                            (THSETQ $?RHS #RESULT#) ))
                 (THSETQ $?RHS (CONS $?A $?RHS))
                 ($G (OK $?R) $T)
                 (PRINT* (LIST $?REL 'SAYS $?A $?R $?Y))
                 (THCOND ( (THASVAL $?X) (NOT (EQ $?X $?A))
                            (PRINT* (LIST '"TRY TO RELATE" $?X 'AND $?A)) ))
                 (THCOND ( (THAND ($G ($?A $?RP $?X $?REL2)) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                            ($G (UPDATE-REST $?A $?RP $?X $?REL2) $T)
                            ($G (F-ALL $?A $?RP $?X) (THUSE FIND-RELS))
                            (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                            (PRINT* (LIST $?REL2 'SAYS $?A $?RP $?X)) )
                          ( (THAND ($G ($?X $?RP $?A $?REL2)) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                            ($G (UPDATE-REST $?X $?RP $?A $?REL2) $T)
                            ($G (F-ALL $?X $?RP $?A) (THUSE FIND-RELS))
                            (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                            (PRINT* (LIST $?REL2 'SAYS $?X $?RP $?A)) )
                          ( (THAND ($G ($?A $?RP $?X $?REL2) $T) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                            (SETQ #RESULT# THVALUE)
                            (THSETQ $?LHS #RESULT#)
                            (PRINT* (LIST $?REL2 'SAYS $?A $?RP $?X)) )
                          ( (THAND ($G ($?X $?RP $?A $?REL2) $T) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
                            (SETQ #RESULT# THVALUE)
                            (THSETQ $?LHS #RESULT#)
                            (PRINT* (LIST $?REL2 'SAYS $?X $?RP $?A))) )
                (THOR ($G ($?A AND $?X ARE T-TRANS IN $?REL2))
                      ($G ($?REL2 IS T-TRANS))
                      (THAND (THASVAL $?VAR) (THSETQ $?R (GENSYM 'ISR))) )
                (THOR ($G ($?A DETERMINES $?Y $?REL))
                    ($G ($?A DETERMINES $?X $?REL2))
                    (THASVAL $?VALID) )
                (THSETQ $?VAR (GENSYM))
                ($A ($?X $?R $?Y $?VAR))
                (THCOND ( (THAND ($G ($?A DETERMINES $?Y $?REL))
                                 ($G ($?A DETERMINES $?X $?REL2)) )
                          ($A ($?X DETERMINES $?Y $?VAR))
                          ($A ($?Y DETERMINES $?X $?VAR)) )
                          ( (THSUCCEED)) )
                (THSETQ $?R $?R2)
                (THSETQ $?RELATION $?VAR) ) )
  (THSETQ $?RES (APPEND $?LHS $?RHS)) )
  (THRETURN $?RES)
)



(THCONSE LISP3 (X R1 Y R2 Z) (PROVE* $?X $?R1 $?Y $?R2 $?Z)
  (THDO (THAND TRACE
                (PRIN1* '"ATTEMPT TO PROVE")
                (THCOND ( (THASVAL $?X) (PRIN1* $?X)) ( T (PRIN1* '$?X)) )
                (THCOND ( (THASVAL $?R1) (PRIN1* $?R1)) ( T (PRIN1* '$?R1)) )
                (THCOND ( (THASVAL $?Y) (PRIN1* $?Y)) ( T (PRIN1* '$?Y)) )
                (THCOND ( (THASVAL $?R2) (PRIN1* $?R2)) ( T (PRIN1* '$?R2)) )
                (THCOND ( (THASVAL $?Z) (PRIN1* $?Z)) ( T (PRIN1* '$?Z)) )
                (TERPRI) ))
  (THDO (THFIND 1 $?R (R) ($G (PROVE $?X $?R1 $?Y $?R2 $?Z $?R) (THUSE PROVE-X-R-Y-R2-Z)))
))



(THCONSE PROVE-X-R-Y-R2-Z (X R1 Y R2 Z REL XVAR YVAR ZVAR VALID RESULT USED)
    (PROVE $?X $?R1 $?Y $?R2 $?Z $?REL)
  (THDO (SETQ #RESTRICT# NIL #EXTRA# NIL))
  (THOR (THNOT (THASVAL $?X)) (THSETQ $?XVAR $?X))
  (THOR (THNOT (THASVAL $?Y)) (THSETQ $?YVAR $?Y))
  (THOR (THNOT (THASVAL $?Z)) (THSETQ $?ZVAR $?Z))
  (THCOND ( (THAND ($G ($?X $?R1 $?Y $?R2 $?Z $?REL)) ($G (ACTIVE $?REL) $T))
            (THSETQ $?RESULT ($G (F-ALL3 $?X $?R1 $?Y $?R2 $?Z) (THUSE F3RELS)))
            (THSETQ $?YVAR (CAR $?RESULT))
            (SETQ #RESULT# $?RESULT) )
          ( ($G ($?X $?R1 $?Y $?R2 $?Z $?REL) (THUSE RELATE-X-Y-Z))
            (PRINT* (SETQ #RESULT# THVALUE))) )
  (SETQ #RESULT# (REVERSE #RESULT#))
  (UPDATE-RESULT)
  (SETQ #RESULT# (LIST (LIST $?X $?XVAR)
                       (LIST $?Y $?YVAR)
                       (LIST $?Z $?ZVAR)
                       #RESULT#))
  (THFAIL THEOREM)
)



(THCONSE RELATE-X-Y-Z (X Y Z R1 R2 REL REL2 A LHS RHS RELATION ITEM VAR1 VAR2 RP)
    ($?X $?R1 $?Y $?R2 $?Z $?RELATION)
  (THOR (THNOT (THASVAL $?R1)) (THSETQ $?VAR1 T))
  (THOR (THNOT (THASVAL $?R2)) (THSETQ $?VAR2 T))
  (THOR (THAND ($G ($?A $?R1 $?Y $?R2 $?Z $?REL))
             ($G (ACTIVE $?REL) $T)
             ($G (RELATE-X-A $?X $?A) (THUSE R-X-A))
             (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
             ($G (UPDATE-REST3 $?A $?R1 $?Y $?R2 $?Z $?REL) $T)
             (THSETQ $?ITEM 1)
             (THSETQ $?RHS ($G (F-ALL3 $?A $?R1 $?Y $?R2 $?Z) (THUSE F3RELS)))
             (THOR (THNOT (EQ $?XVAR $?X)) (THSETQ $?XVAR (CAR $?LHS)))
             (THOR (THNOT (EQ $?ZVAR $?Z)) (THSETQ $?ZVAR (CAR $?RHS)))
             (THOR (THNOT (EQ $?YVAR $?Y)) (THSETQ $?YVAR (CAR $?RHS))) )
       (THAND ($G ($?X $?R1 $?A $?R2 $?Z $?REL))
             ($G (ACTIVE $?REL) $T)
```

```
                   ($G (RELATE-X-A $?Y $?A) (THUSE R-X-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                   (THOR (THNOT (EQ $?YVAR $?Y)) (THSETQ $?YVAR #RESULT#))
                   ($G (UPDATE-REST3 $?X $?R1 $?A $?R2 $?Z $?REL) $T)
                   (THSETQ $?ITEM 2)
                   (THSETQ $?RHS ($G (F-ALL3 $?X $?R1 $?A $?R2 $?Z) (THUSE F3RELS)))
                   (THOR (THNOT (EQ $?XVAR $?X)) (THSETQ $?XVAR (CAR $?RHS)))
                   (THOR (THNOT (EQ $?ZVAR $?Z)) (THSETQ $?ZVAR (CAR $?RHS))) )
            (THAND ($G ($?X $?R1 $?Y $?R2 $?A $?REL))
                   ($G (ACTIVE $?REL) $T)
                   ($G (RELATE-X-A $?Z $?A) (THUSE R-X-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                   ($G (UPDATE-REST3 $?X $?R1 $?Y $?R2 $?A $?REL) $T)
                   (THSETQ $?ITEM 3)
                   (THSETQ $?RHS ($G (F-ALL3 $?X $?R1 $?Y $?R2 $?A) (THUSE F3RELS)))
                   (THOR (THNOT (EQ $?XVAR $?X)) (THSETQ $?XVAR (CAR $?RHS)))
                   (THOR (THNOT (EQ $?ZVAR $?Z)) (THSETQ $?ZVAR (CAR $?LHS)))
                   (THOR (THNOT (EQ $?YVAR $?Y)) (THSETQ $?YVAR (CAR $?RHS))) )
            (THAND ($G (RELATE-X-A $?X $?A) (THUSE R-X-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                   (THOR (THNOT (EQ $?XVAR $?X)) (THSETQ $?XVAR (CAR $?LHS)))
                   ($G (RELATE-X-Y-A $?A $?R1 $?Y $?R2 $?Z) (THUSE R-X-Y-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?RHS #RESULT#)
                   (THSETQ $?ITEM 1) )
            (THAND ($G (RELATE-X-A $?Y $?A) (THUSE R-X-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                   (THOR (THNOT (EQ $?YVAR $?Y)) (THSETQ $?YVAR (CAR $?LHS)))
                   ($G (RELATE-X-Y-A $?X $?R1 $?A $?R2 $?Z) (THUSE R-X-Y-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?RHS #RESULT#)
                   (THSETQ $?ITEM 2) )
            (THAND ($G (RELATE-X-A $?Z $?A) (THUSE R-X-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?LHS #RESULT#)
                   (THOR (THNOT (EQ $?ZVAR $?Z)) (THSETQ $?ZVAR (CAR $?LHS)))
                   ($G (RELATE-X-Y-A $?X $?R1 $?Y $?R2 $?A) (THUSE R-X-Y-A))
                   (SETQ #RESULT# THVALUE) (THSETQ $?RHS #RESULT#)
                   (THSETQ $?ITEM 3) )
            (THAND ($G ($?X $?R1 $?Y $?REL) $T)
                   (SETQ #RESULT# THVALUE)
                   (THSETQ $?LHS #RESULT#)
                   (THSETQ $?USED (LIST NIL))
                   ($G (UPDATE-REST $?X $?R1 $?Y $?REL) $T)
                   ($G (ACTIVE $?REL) $T)
                   ($G ($?X $?R2 $?Z $?REL2) $T)
                   (THSETQ $?USED (LIST NIL))
                   (SETQ #RESULT# THVALUE)
                   ($G (UPDATE-REST $?X $?R2 $?Z $?REL2) $T)
                   ($G (ACTIVE $?REL2) $T)
                   (THSETQ $?RHS (LIST $?X #RESULT#))
                   (THOR ($G ($?X DETERMINES $?Y $?REL))
                         ($G ($?X DETERMINES $?Z $?REL2))
                         (THASVAL $?VALID) ))
            (THAND ($G ($?X $?R1 $?Y $?REL) $T)
                   (THSETQ $?USED (LIST NIL))
                   (SETQ #RESULT# THVALUE)
                   ($G (UPDATE-REST $?X $?R1 $?Y $?REL) $T)
                   ($G (ACTIVE $?REL) $T)
                   (THSETQ $?LHS #RESULT#)
                   (THOR ($G ($?X DETERMINES $?Y $?REL)) (THASVAL $?VALID))
                   ($G ($?Y $?R2 $?Z $?REL2) $T)
                   (THSETQ $?USED (LIST NIL))
                   (SETQ #RESULT# THVALUE)
                   ($G (UPDATE-REST $?Y $?R2 $?Z $?REL2) $T)
                   ($G (ACTIVE $?REL2) $T)
                   (THSETQ $?RHS (LIST $?Y #RESULT#))
                   (THOR ($G ($?Y DETERMINES $?Z $?REL2)) (THASVAL $?VALID))
                   (THSETQ $?YVAR (APPEND (CAR $?LHS) (CDR (LAST $?RHS)))) )
    )
    (THCOND ( (THNOT (THASVAL $?ITEM)) (THSUCCEED))
            ( (EQ $?ITEM 1)
              (THOR ($G ($?A DETERMINES $?X $?REL2))
                    (THAND ($G ($?A DETERMINES $?Y $?REL))
                           ($G ($?A DETERMINES $?Z $?REL)) ) )
              (THOR ($G ($?REL2 IS T-TRANS))
                    (THAND (THASVAL $?VAR1) (THSETQ $?R1 (GENSYM 'ISR))
                           ($G ($?A DETERMINES $?Y $?REL))
                           (THASVAL $?VAR2)
                           (THSETQ $?R2 (GENSYM 'ISR)) )))
            ( (EQ $?ITEM 2)
              (THOR ($G ($?A DETERMINES $?Y $?REL2))
                    (THAND ($G ($?A DETERMINES $?X $?REL))
                           ($G ($?A DETERMINES $?Z $?REL)) ) )
              (THOR ($G ($?REL2 IS T-TRANS))
                    (THAND (THASVAL $?VAR1) (THSETQ $?R1 (GENSYM 'ISR))
                           ($G ($?A DETERMINES $?Z $?REL))
                           (THASVAL $?VAR2)
                           (THSETQ $?R2 (GENSYM 'ISR)) )) )
            ( (EQ $?ITEM 3)
              (THOR ($G ($?A DETERMINES $?Z $?REL2))
                    (THAND ($G ($?A DETERMINES $?X $?REL))
                           ($G ($?A DETERMINES $?Y $?REL)) ) )
              (THOR ($G ($?REL2 IS T-TRANS))
                    (THAND (THASVAL $?VAR2)
                           (THSETQ $?REL2 (GENSYM 'ISR)) )) )
    )
    (THSETQ $?A (GENSYM))
    ($A ($?X $?R1 $?Y $?R2 $?Z $?A))
    (THSETQ $?RELATION $?A)
    (THSUCCEED THEOREM (APPEND $?LHS $?RHS) )
   )
```

```



(THCONSE ENUMERATE-DOMAIN (X RESULT R) (ENUMERATE $?X)
  (THCOND ( ($G ($?R ENUMERATES $?X))
            ($G (ACTIVE $?R) $T)
            (THSETQ $?RESULT (LIST $?R)))
          ( ($G ($?R P-ENUMERATES $?X))
            (THSETQ $?RESULT
            (THFIND ALL $?RELS (RELS) (THAND ($G ($?RELS P-ENUMERATES $?X))
                                             ($G (ACTIVE $?RELS) $T))) ))
          ( (THSETQ $?RESULT
            (THFIND ALL $?RELS (RELS A) (THAND
                                 (THOR ($G ($?X $?R $?A $?RELS))
                                       ($G ($?A $?R $?X $?RELS)))
                                 ($G (ACTIVE $?RELS) $T))) ))
  )
  (SETQ #RESULT# (LIST (LIST $?X $?RESULT) (LIST $?RESULT)))
)



(THCONSE VALID-PREDICATE (R) (OK $?R)
  (THNOT ($G (EXTRA $?R)))
)


(THCONSE ACTIVE-RELATION (R) (ACTIVE $?R)
  (THNOT ($G ($?R IS INACTIVE)))
)


(THCONSE CHECK-RESTRICTIONS (X Y R REL NOS)
   (UPDATE-REST $?X $?R $?Y $?REL)
  (THCOND ( (THSETQ $?NOS (THFIND ALL ($?X $?Y $?NO) (NO) ($G ($?R IN $?REL IS_RESTRICTED_TO $?NO))) ))
          ( (THSUCCEED THEOREM)) )
  (THDO (THAND TRACE (PRIN1* '"MAKING NOTE OF RESTRICTION ON")
                (PRIN1* $?R) (PRIN1* 'IN) (PRIN1* $?REL) (TERPRI)) )
  (THMAPC 'UPDATE $?NOS)
)



(THCONSE UPDATE (X Y NO R2 REL2 DOMAIN A REST) ($?X $?Y $?NO)
  ($G ($?NO RESTRICTS $?DOMAIN TO $?REST))
  (THCOND ( ($G ($?X $?R2 $?DOMAIN $?REL2))
            ($G (OK $?R2) $T)
            (THSETQ $?A $?X) (THSETQ $?REL2 (LIST $?REL2)) )
          ( ($G ($?Y $?R2 $?DOMAIN $?REL2))
            ($G (OK $?R2) $T)
            (THSETQ $?A $?Y) (THSETQ $?REL2 (LIST $?REL2)) )
          ( ($G ($?X $?R2 $?DOMAIN $?REL2) $T)
            (SETQ #RESULT# THVALUE) (THSETQ $?REL2 #RESULT#)
            (THSETQ $?A $?X))
          ( ($G ($?Y $?R2 $?DOMAIN $?REL2) $T)
            (SETQ #RESULT# THVALUE) (THSETQ $?REL2 #RESULT#)
            (THSETQ $?A $?Y)) )
  (SETQ #RESTRICT# (APPEND (LIST $?REST $?A $?REL2 $?DOMAIN)
                           #RESTRICT#))
)



(THCONSE CHECK-3RESTRICTIONS (X R1 Y R2 Z REL)
    (UPDATE-REST3 $?X $?R1 $?Y $?R2 $?Z $?REL)
  ($G (UPDATE-REST $?X $?R1 $?Y $?REL) $T)
  (THOR (THAND ($G ($?X $?R2 $?Z $?REL))
               ($G (UPDATE-REST $?X $?R2 $?Z $?REL) $T))
        ($G (UPDATE-REST $?Y $?R2 $?Z $?REL)$T ) )
)



(THCONSE FIND-RELS (X R Y RES) (F-ALL $?X $?R $?Y)
  (THSETQ $?RES
    (THFIND ALL $?RELS (RELS) (THAND ($G ($?X $?R $?Y $?RELS))
                                     ($G (ACTIVE $?RELS) $T)
                                     (THERASE ($?X $?R $?Y $?RELS)) ))
  )
  (THSUCCEED THEOREM (LIST $?RES))
)



(THCONSE F3RELS (X R1 Y R2 Z RES) (F-ALL3 $?X $?R1 $?Y $?R2 $?Z)
  (THSETQ $?RES
    (THFIND ALL $?RELS (RELS) (THAND ($G ($?X $?R1 $?Y $?R2 $?Z $?RELS))
                                     ($G (ACTIVE $?RELS) $T)) )
  )
  (THSUCCEED THEOREM (LIST $?RES))
)

```

```


(THCONSE R-X-A (X A RP REL2 LHS) (RELATE-X-A $?X $?A)
  (THCOND ( (THAND ($G ($?X $?RP $?A $?REL2)) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
            ($G (UPDATE-REST $?X $?RP $?A $?REL2) $T)
            (THSETQ $?LHS ($G (F-ALL $?X $?RP $?A)  (THUSE FIND-RELS))) )
          ( (THAND ($G ($?A $?RP $?X $?REL2)) ($G (OK $?RP) $T) ($G (ACTIVE $?REL2) $T))
            ($G (UPDATE-REST $?A $?RP $?X $?REL2) $T)
            (THSETQ $?LHS ($G (F-ALL $?A $?RP $?X)  (THUSE FIND-RELS))) ) )
  (THSUCCEED THEOREM (APPEND $?LHS (LIST $?A)) )
)


(THCONSE R-X-Y-A (X Y Z R1 R2 RHS) (RELATE-X-Y-A $?X $?R1 $?Y $?R2 $?Z)
  (THCOND ( (THAND ($G ($?X $?R1 $?Y $?R2 $?Z $?REL)) ($G (ACTIVE $?REL) $T))
            ($G (UPDATE-REST3 $?X $?R1 $?Y $?R2 $?Z $?REL) $T)
            (THSETQ $?RHS ($G (F-ALL3 $?X $?R1 $?Y $?R2 $?Z) (THUSE F3RELS)))
            (THOR (NOT (EQ $?X $?XVAR)) (THSETQ $?XVAR (CAR $?RHS)))
            (THOR (NOT (EQ $?Z $?ZVAR)) (THSETQ $?ZVAR (CAR $?RHS)))
            (THOR (NOT (EQ $?Y $?YVAR)) (THSETQ $?YVAR (CAR $?RHS))) )
          ( (THAND ($G ($?X $?R1 $?Y $?R2 $?Z $?REL) (THUSE RELATE-X-Y-Z)) )
            (SETQ #RESULT# THVALUE)
            ($G (ACTIVE $?REL) $T)
            (THSETQ $?RHS #RESULT#)) )
  (THSUCCEED THEOREM $?RHS)
)



; START OF ANTECEDENT THEOREMS


(THANTE NOT-AVAILABLE (X RELS) ($?X ARE NOT AVAILABLE)
  (THSETQ $?RELS (THFIND ALL ($?REL) (REL) ($G ($?REL CONCERNS $?X)) ))
  (THMAPC 'ASSERTFUN $?RELS)
)



(THCONSE ASSERTFUN (X) ($?X) ($A ($?X IS INACTIVE)))



(THANTE ON-STRIKE (X REST# RELS LIST) ($?X SUPPLIERS ARE ON STRIKE)
  (THSETQ $?REST# (GENSYM 'REST#))
  (THSETQ $?LIST (LIST 'NOT (LIST 'EQ 'SLOC (LIST 'QUOTE $?X))))
  ($A ($?REST# RESTRICTS SLOC TO $?LIST))
  (THSETQ $?RELS (THFIND ALL (SUPPLIES $?R $?REST#) (R A B)
        ($G ($?A SUPPLIES $?B $?R)) ))
  (THMAPC 'ASSERT-RESTRICTION $?RELS)
  (THSETQ $?RELS (THFIND ALL (SUPPLIES-# $?R $?REST#) (R A B)
        ($G ($?A SUPPLIES-# $?B $?R)) ))
  (THMAPC 'ASSERT-RESTRICTION $?RELS)
  (THSETQ $?RELS (THFIND ALL (IS-SUPPLIED-BY $?R $?REST#) (R A B)
        ($G ($?A IS-SUPPLIED-BY $?B $?R)) ))
  (THMAPC 'ASSERT-RESTRICTION $?RELS)
)



(THCONSE ASSERT-RESTRICTION (RELSHIP REL# REST#) ($?RELSHIP $?REL# $?REST#)
  ($A ($?RELSHIP IN $?REL# IS_RESTRICTED_TO $?REST#))
)



(DEFUN UPDATE-RESULT ( )
  (PROG (REST TEMP)
   (COND ( (NULL (SETQ REST #RESTRICT#)) (RETURN T)) )
   LOOP (SETQ TEMP (CONS (CAR REST) TEMP))
         (OR (ASSO (CADDDR REST) #RESULT#)
              (SETQ #EXTRA# (APPEND (LIST (LIST (CADDDR REST)))
                                              #EXTRA#)) )
         (COND ( (NOT (MEMBER (CADDR REST) #RESULT#))
                   (SETQ #RESULT# (APPEND #RESULT#
                                             (LIST (CADR REST) (CADDR REST))) ) ))
         (COND ( (NULL (SETQ REST (CDDDDR REST)))
                   (SETQ #RESTRICT# TEMP)
                   (RETURN T))
                 ( T (GO LOOP)) )
  )
 )
END OF FILE
```

APPENDIX 10 - NEW REDUCTION ALGORITHM

```
(DEFUN REDUCE (QUERY)
  ; THIS ROUTINE WILL REDUCE A QUERY IN THE RELATIONAL CALCULUS TO
  ; A RESPONSE RELATION.  IT DOES SO BY USING THE RELATIONAL ALGEBRA.
  (PROG (TARGET_LIST RESPONSE QUANTS #RESULT# #EXTRA# #RESTRICT#)
   (AND TRACE (PRINT '"ABOUT TO REDUCE THE QUERY"))
   (SETQ TARGET_LIST (CREATE_TARGET QUERY))
   (SETQ QUERY (CDR (MEMQ ': QUERY)))
   (MAPC '(LAMBDA (QUANT)
            (COND ( (MEMQ (CAR QUANT) '(E V))
                    (SETQ QUERY (CDR QUERY))
                    (SETQ QUANTS (CONS (GENSYM1 'QUANT) QUANTS))
                    (SET (CAR QUANTS) (CADR QUANT))
                    (PUT (CAR QUANTS) 'REL (CREATE_QUANT (CDR QUANT))) )
                  ( (UNEVAL 'MAPC NIL)) )
          )
         QUERY)
   (SETQ RESPONSE (DIVIDE_OR_PROJECT (DISJUNCT QUERY) QUANTS))
   (RETURN (PROJECT RESPONSE (DOMAIN#S RESPONSE TARGET_LIST)))
 ))



(DEFUN CREATE_TARGET (QUERY)
  ; THIS ROUTINE WILL CREATE A LIST OF THOSE DOMAINS WHICH ARE TO
  ; BE PRESENT IN THE RESPONSE RELATION.
  (PROG (RESULT)
   (MAPC '(LAMBDA (ELT)
           (COND ( (EQ ELT ':) (UNEVAL 'MAPC NIL))
                 ( T (SETQ RESULT (CONS ELT RESULT))) )
          )
         QUERY)
   (RETURN (REVERSE RESULT))
 ))



(DEFUN CREATE_QUANT (QUANT)
  ; THIS ROUTINE TAKES A QUANTIFIER FROM THE QUERY, AND CREATES THE
  ; CORRESPONDING RELATION.
  (PROG (RESULT)
   (AND TRACE (PRIN1 '"DETERMINE THE RELATION WHICH QUANTIFIES")
              (PRIN1 (CAR QUANT)) (TERPRI))
   (SETQ RESULT (DISJUNCT QUANT))
   (PROJECT RESULT (DOMAIN#S RESULT (CAR QUANT)))
 ))



(DEFUN CREATE_RELATION (PLIST)
  ; THIS ROUTINE TAKES THE LIST THAT PLANNER RETURNS, AND CREATES
  ; THE RELATION WHICH IT DEFINES.
  (PROG (METHOD RESULT JTERM #L1# #L2#)
   (SETQ METHOD (CAR (LAST PLIST)))
   (SETQ PLIST (DELETE METHOD PLIST))
   (SETQ RESULT (REL_DEF_BY (UNCONS METHOD METHOD)))
   (MAPC '(LAMBDA (REL)
           (COND ( (ATOM REL) (SETQ JTERM REL))
                 ( T (SETQ REL (REL_DEF_BY REL))
                     (SETQ #L1# (DOMAIN#S RESULT JTERM)
                           #L2# (DOMAIN#S REL JTERM))
                     (SETQ RESULT (JOIN RESULT
                                        REL
                                        '(EQUAL (ELEM T1 #L1#)
                                                (ELEM T2 #L2#)) )) ))
          )
         METHOD)
   (SETQ DOMAINS (MAPCAR 'CAR PLIST))
  (RETURN (PROJECT RESULT (DOMAIN#S RESULT DOMAINS)))
 ))



(DEFUN REL_DEF_BY (RELS)
  ; THIS ROUTINE TAKES A LIST OF RELATION NAMES, AND RETURNS THE RELATION
  ; WHOSE DOMAINS ARE THOSE COMMON TO ALL RELATIONS IN RELS.
  (PROG (DOMAINS RESULT)
   (COND ( (EQ (LENGTH RELS) 1) (RETURN (CAR RELS)) ))
  ; CREATE A LIST OF THE DOMAINS WHICH ALL THE RELATIONS HAVE IN COMMON.
   (SETQ DOMAINS (GET (CAR RELS) 'DOMAINS))
   (MAPC '(LAMBDA (REL)
            (SETQ DOMAINS (INTERSECT DOMAINS (GET REL 'DOMAINS)))
          )
         (CDR RELS))
   (SETQ RESULT (PROJECT (CAR RELS) (DOMAIN#S (CAR RELS) DOMAINS)))
   (MAPC '(LAMBDA (REL)
            (SETQ RESULT (RUNION* RESULT (PROJECT REL (DOMAIN#S REL DOMAINS))) )
          )
         (CDR RELS))
  (RETURN RESULT)
 ))





(DEFUN DISJUNCT (QUERY)
  ; THIS ROUTINE WILL PROCESS A QUERY WHICH IS IN D.N.F.  IT DOES THIS
  ; BY DETERMINING THE RELATION DEFINED BY EACH CONJUNCT, AND THEN TAKING
  ; THE UNION OF THESE RESULTS ON THE COMMON DOMAINS.
```

```
   (PROG (RELS)
    (COND ( (OR (NULL (CDR QUERY)) (EQ (CADR QUERY) '&))
           (RETURN (CONJUNCT QUERY)) ))
    (MAPC '(LAMBDA (C)
           (OR (EQ C 'V)
               (SETQ RELS (CONS (CONJUNCT C) RELS)) )
         )
        QUERY)
    (REL_DEF_BY RELS)
 ))



 (DEFUN CONJUNCT (QUERY)
   ; THIS ROUTINE WILL RETURN THE RELATION DEFINED BY A CONJUNCTION
   ; IN THE QUERY.  ALL RELATIONAL TERMS ARE PASSED TO PLANNER AS
   ; THGOALS, AND THE CORRESPONDING RELATIONS ARE CREATED.
   ; THESE ARE THEN SUBJECTED TO THE RESTRICTIONS IN THE CONJUNCTION,
   ; WITH THE FINAL RESULTS BEING JOINED TOGETHER ON EITHER COMMON
   ; DOMAINS, OR AS SPECIFIED BY THE JOIN TERMS.
   (PROG (FNR RELS JTERMS R FLAG RELSP RELSN DOMAINS)
    (MAPC '(LAMBDA (RELTERM)
      (COND ( (EQ RELTERM '&) NIL)
            ( (EQ RELTERM '¬) (SETQ FLAG T))
            ( (ATOM RELTERM) (THVAL (LIST '$G (LIST 'ENUMERATE RELTERM) '$T)
                                    (LIST NIL NIL)))
            ( (FUNCTIONP (CAR RELTERM))
              (UNEVAL 'MAPC NIL))
            ( T (THVAL (LIST '$G (CONS 'PROVE* RELTERM) '$T)
                       (LIST NIL NIL)) ))
      (OR (NULL #RESTRICT#) (SETQ QUERY (APPEND QUERY #RESTRICT#)))
      (SETQ QUERY (CDR QUERY))
      (OR (EQ RELTERM '&) (EQ RELTERM '¬)
          (SETQ RELS (CONS (CREATE_RELATION #RESULT#) RELS)) )
      (COND ( (AND FLAG (NOT (EQ RELTERM '¬))) (PUT (CAR RELS) 'NEGATED T)
                  (SETQ FLAG NIL)) )
      (SETQ #RESTRICT# NIL #EXTRA# NIL)
        )
        QUERY)
   ; GO THROUGH ALL THE RESTRICTION AND JOIN TERMS, APPLYING THE RESTRICTIONS
   ; TO THE APPROPRIATE RELATIONS, AND SAVING THE JOIN TERMS, ALONG WITH
   ; THE NAMES OF THE DOMAINS THEY CONTAIN.
    (MAPC '(LAMBDA (TERM)
      (COND ( (EQ TERM '&) NIL)
            ( (ATOM (SETQ DOMAINS (DOMAINS_IN TERM)))
              (MAPC '(LAMBDA (REL)
                (AND (MEMQ DOMAINS (GET REL 'DOMAINS))
                     (SETQ R (RESTRICT REL (FIXPRED DOMAINS TERM REL 'T1)))
                     (PUT REL 'TUPLES (GET R 'TUPLES))
                     (PUT REL '#TUPLES (GET R '#TUPLES))
                     (AND TRACE (PRIN1 'RESTRICT) (PRIN1 REL) (PRIN1 'TO)
                          (PRIN1 TERM) (TERPRI)) )
                   )
                   RELS) )
            ( T (SETQ JTERMS (CONS (LIST TERM DOMAINS) JTERMS)) ))
          )
        QUERY)
   ; GLUE ALL THE RESULTS TOGETHER - JOIN ALL RELATIONS WHICH MUST BE
   ; JOINED USING JOIN TERMS.
    (SETQ RELS (JOIN_WJT RELS JTERMS))
   ; TAKE THE UNION OF ALL NON-NEGATED RELATIONS
    (MAPC '(LAMBDA (R)
           (COND ( (GET R 'NEGATED) (SETQ RELSN (CONS R RELSN)) )
                 ( T (SETQ RELSP (CONS R RELSP)) ))
         )
        RELS)
    (AND TRACE (PRIN1 '"TAKE THE UNION OF RELATIONS") (PRIN1 RELSP) (TERPRI))
    (SETQ RELSP (JOIN_RELS RELSP))
   ; TAKE THE DIFFERENCE BETWEEN THIS RELATION AND EACH OF THE NEGATED
   ; RELATIONS.
    (MAPC '(LAMBDA (RN)
      (AND TRACE (PRIN1 '"THE DIFFERENCE BETWEEN") (PRIN1 RELSP)
                 (PRIN1 'AND) (PRIN1 RN) (TERPRI))
      (SETQ DOMAINS (INTERSECT (GET RELSP 'DOMAINS) (GET RN 'DOMAINS)))
      (SETQ RELS (RDIFF (PROJECT RELSP (DOMAIN#S RELSP DOMAINS))
                        (PROJECT RN (DOMAIN#S RN DOMAINS)) ))
      (SETQ RELSP (JOIN_RELS (LIST RELSP RELS)))
         )
        RELSN)
   (RETURN RELSP)
 ))



 (DEFUN DIVIDE_OR_PROJECT (REL QUANTS)
   ; THIS ROUTINE WILL PROCESS THE QUANTIFIERS FROM RIGHT TO LEFT,
   ; DIVIDING OR PROJECTING THE CURRENT RELATION BY THE QUANTIFIED
   ; VARIABLE.
   (PROG (VAR DOMAINS)
    (MAPC '(LAMBDA (Q)
      (SETQ VAR (GET Q 'REL))
      (SETQ DOMAINS (GET VAR 'DOMAINS))
      (COND ( (EQ Q 'E)
              (COND ( TRACE (PRIN1 '"PROJECT THE RESPONSE RELATION ON ALL FIELDS EXCEPT")
                            (PRIN1 (EVAL Q)) (TERPRI))
                    ( T T))
              (SETQ REL (PROJECT REL (DOMAIN#S REL (SETDIF (GET REL 'DOMAINS)
                                                           (GET VAR 'DOMAINS))) )) )
            ( T (SETQ REL (RDIVIDE REL VAR (DOMAIN#S REL DOMAINS)
                                           (GENLIST DOMAINS)))
                (AND TRACE (PRIN1 '"DIVIDE THE RESPONSE RELATION BY")
                           (PRIN1 (EVAL G)) (TERPRI)) ))
           )
          QUANTS)
   (RETURN REL)
 ))



(DEFUN JOIN_WJT (RELS JTERMS)
  ; THIS ROUTINE WILL JOIN ALL RELATIONS IN RELS WHICH CAN BE JOINED
  ; UNDER CRITERION SPECIFIED BY JTERMS.
  ; IT RETURNS A NEW LIST OF THE CURRENT RELATIONS
 (PROG (R1 R2 R)
  (MAPC '(LAMBDA (JTERM)
     (MAPC '(LAMBDA (R)
       (COND ( (GET R 'NEGATED) (UNEVAL 'MAPC NIL))
             ( (MEMQ (CAADR JTERM) (GET R 'DOMAINS))
               (SETQ R1 R))
             ( (MEMQ (CADADR JTERM) (GET R 'DOMAINS))
               (SETQ R2 R)) )
            )
           RELS)
    (SETQ JTERM (FIXPRED (CAR DOMAINS) (FIXPRED (CADR DOMAINS) JTERM R2 'T2) R1 'T1))
    (SETQ R (JOIN R1 R2 JTERM))
    (SETQ RELS (DELETE R1 (DELETE R2 RELS))
          RELS (CONS R RELS))
          )
        JTERMS)
  (RETURN RELS)
))



(DEFUN JOIN_RELS (RELS)
  ; THIS ROUTINE WILL TAKE A LIST OF RELATIONS AND JOIN THEM
  ; ON COMMON DOMAINS.
 (PROG (REL DOMAINS #L1# #L2#)
  (SETQ REL (UNCONS RELS RELS))
  (AND (NULL RELS) (RETURN REL))
  (MAPC '(LAMBDA (R)
    (AND (SETQ DOMAINS (INTERSECT (GET R 'DOMAINS) (GET REL 'DOMAINS)))
         (SETQ #L1# (DOMAIN#S REL DOMAINS))
         (SETQ #L2# (DOMAIN#S R DOMAINS))
         (SETQ REL (JOIN REL R '(EQUAL (ELEM T1 #L1#) (ELEM T2 #L2#))))
         (NULL (SETQ RELS (DELETE R RELS))) (RETURN REL)
         ))
       RELS)
  (JOIN_RELS (CONS REL RELS))
))



(DEFUN DOMAIN#S (RELATION L_OF_DOMAINS)
  ; THIS ROUTINE WILL RETURN THE POSITIONS EACH OF THE DOMAINS IN L_OF_DOMAINS
  ; OCCUPIES IN RELATION.
 (PROG (DOMAINS)
  (AND (ATOM L_OF_DOMAINS) (SETQ L_OF_DOMAINS (LIST L_OF_DOMAINS)))
  (SETQ DOMAINS (GET RELATION 'DOMAINS))
  (MAPCAR '(LAMBDA (DOMAIN)
             (ADD1 (SUB (LENGTH DOMAINS)
                        (LENGTH (MEMQ DOMAIN DOMAINS)) ))
           )
          L_OF_DOMAINS)
))



(DEFUN INTERSECT (L1 L2)
  ; THIS PROCEDURE RETURNS THE INTERSECTION OF LISTS L1 AND L2.
 (PROG (RESULT)
  (MAPC '(LAMBDA (E1)
           (AND (MEMBER E1 L2) (NOT (MEMBER E1 RESULT))
                (SETQ RESULT (CONS E1 RESULT))) )
         )
        L1)
  (RETURN RESULT)
))



(DEFUN FIXPRED (DOMAIN JTERM REL TX)
  ; THIS ROUTINE WILL CHANGE ALL OCCURRENCES OF DOMAIN IN JTERM
  ; TO (ELEM TX N) , WHERE DOMAIN IS IN THE NTH POSITION OF REL.
  (COPY JTERM DOMAIN (LIST 'ELEM TX (CAR (DOMAIN#S REL DOMAIN))))
)



(DEFUN DOMAINS_IN (FUN)
  ; THIS ROUTINE WILL RETURN EITHER A LIST OF ALL DOMAIN VARIABLES
  ; IN THE FUNCTION, OR THE NAME OF THE VARIABLE IN THE FUNCTION
  ; IF THERE IS ONLY ONE.
 (PROG (ANS)
  (DOMAINS_IN* FUN)
  (COND ( (EQ (LENGTH ANS) 1) (RETURN (CAR ANS)))
        ( T (RETURN ANS)) )
))
```

```


(DEFUN DOMAINS_IN* (FUN)
  (PROG ( )
  (COND ( (OR (NULL FUN) (EQ (CAR FUN) 'QUOTE))
           (RETURN NIL))
        ( (ATOM (CAR FUN))
          (OR (NUMBERP (CAR FUN)) (FUNCTIONP (CAR FUN))
               (SETQ ANS (CONS (CAR FUN) ANS)) ))
        ( T (DOMAINS_IN* (CAR FUN))) )
  (DOMAINS_IN* (CDR FUN))
 ))



(DEFUN FUNCTIONP (FUN)
  ; THIS ROUTINE RETURNS T IF FUN IS A LISP FUNCTION.
  (OR (GET FUN 'EXPR)
      (GET FUN 'SUBR)
      (GET FUN 'FSUBR))
 )
END OF FILE
```

```
     : QUERY
     : THIS ROUTINE WILL ACCEPT INPUT FROM THE USER.  THE INPUT CAN EITHER
     : BE A CUERY, OR A PIECE OF INFORMATION TO BE ADDED TO THE SEMANTIC MODEL.

     (DEFUN CUERY ( )
       ; READ - REPLY LOOP
       (PRCG (RR TRACE)
       (PRINT '"TRACE ON, OR OFF?")
       (OR (EQUAL (READ) 'OFF) (SETQ TRACE T))
        RD (SETQ RR (READ))
           (COND ( (EQ RR '*END) (RETURN 'THANKS))
                 ( (NOT (MEMQ ': RR))
                   (THVAL (LIST 'SA RR 'ST) (LIST NIL NIL))
                   (PRINT '"O.K.") (GO RD))
                 ( T (SETQ RR (REDUCE RR))) )
           (COND ( (ZEROP (GET RR '#TUPLES)) (PRINT '"NONE."))
                 ( T (PRINTREL RR)) )
           (GO RD)
     ))



     : DEFRELS (DEFINE RELATIONS)
     : DEFRELS READS IN RELATIONS FROM THE FILE "FILE", AND STORES THEM
     : IN INTERNAL FORMAT:
     : THE RELATION NAME HAS 3 PROPERTIES ON ITS P-LIST:
     :    DOMAINS- A LIST OF THE DOMAIN NAMES UPON WHICH THE RELATION
     :             IS DEFINED.
     :    TUPLES - A LIST OF THE TUPLE NAMES WHICH OCCUR IN THE RELATION.
     :    #TUPLES - THE NUMBER OF TUPLES CURRENTLY IN THE RELATION.
     : ON THE P-LIST OF A TUPLE NAME, UNDER THE FLAG "DATA", IS THE
     : ACTUAL TUPLE.
    (DEFUN CEFRELS FEXPR (FILE)
       (APPLY1 'OPEN (LIST 'RELINPUT 255 (CAR FILE)))
       (OPEN (BUFFER 255))
       (PRCG (RELATION RNAME TNAME)
          RD (COND ( (EQ (SETQ RELATION (READ RELINPUT)) '*END)
                      (RETURN NIL)) )
          (SETQ RNAME (UNCONS RELATICN RELATION))
          (PUT RNAME '#TUPLES 0)
          (PUT RNAME 'DOMAINS (CAR RELATION))
          (MAPC '(LAMBDA (TUPLE)
             (ADDTUPLE TUPLE RNAME) )
                (CDR RELATION))
          (GO RD)
       )
    )



     : PRINTREL
     : PRINTS A GIVEN RELATION


    (DEFUN PRINTREL (RELATION)
       (PROG (SPACES CENTRES)
          (TERPRI)
          (COND ( (NULL (SETQ CENTRES (GET RELATION 'CENTRES)))
                   (SETQ CENTRES (FIND_CENTRES RELATICN)) ))
          (PRIN1 RELATION)
          (PRIN1 '"(") (PRINT_TUPLE (GET RELATION 'DOMAINS) CENTRES)
          (PRIN1 '"  )") (TERPRI)
          (MAPC '(LAMBDA (TUPLE)
             (PRINT_TUPLE (GET TUPLE 'DATA) CENTRES)
             (TERPRI))
                (REVERSE (GET RELATION 'TUPLES)))
     (RETURN RELATION))
    )



    (DEFUN PRINT_TUPLE (TUPLE CENTRES)
      : THIS ROUTINE WILL PRINT A TUPLE, WITH EACH ELEMENT BEING
      : CENTRED ABOUT THE POSITION GIVEN IN THE LIST CENTRES
      (PRCG (L LEN)
       (MAPC '(LAMBDA (NAME POS)
                (COND ( (NUMBERP NAME) (SETQ LEN (NLEN NAME)))
                      ( (SETQ LEN (PLEN NAME))) )
                (TAB (SUB POS (FIX (DIVIDE LEN 2))))
                (PRIN1 NAME)
              )
             TUPLE CENTRES)
    ))
```

```

 (DEFUN FIND_CENTRES (RELATION)
   ; THIS ROUTINE WILL CREATE A LIST WHICH CONTAINS THE PRINT POSITION
   ; WHERE EACH DOMAIN IN THE RELATION SHOULD BE CENTRED.
   (PROG (TEMP LEN CENTRE)
     (COND ( (NULL (GET RELATION 'CENTRES))
             (PUT RELATION 'CENTRES (MAPCAR '(LAMBDA (X) 0) (GET RELATION 'DOMAINS))) ))
    (SETQ CENTRE (GET RELATION 'CENTRES))
    (MAPC '(LAMBDA (TUPLE)
      (OR (LISTP TUPLE) (SETQ TUPLE (GET TUPLE 'DATA)))
      (SETQ TEMP CENTRE)
      (MAPC '(LAMBDA (ELT)
              (COND ( (NUMBERP ELT) (SETQ LEN (NLEN ELT)))
                    ( (SETQ LEN (PLEN ELT))) )
             (AND (GREATERP LEN (CAR TEMP)) (RPLACA TEMP LEN))
             (SETQ TEMP (CDR TEMP))
           )
          TUPLE)
          )
        (APPEND (LIST  (GET RELATION 'DOMAINS)) (GET RELATION 'TUPLES)) )
   (PUT RELATION 'CENTRES (UPDATE_CENTRES RELATION CENTRE))
 ))



 (DEFUN UPDATE_CENTRES (RELATION CENTRES)
   ; THIS ROUTINE WILL CHANGE THE LIST OF MAXIMUM DOMAIN SIZES
   ; TO A LIST OF CENTRES FOR EACH DOMAIN
   (PROG (SUM LIST)
    (SETQ SUM (ADD 4 (PLEN RELATION)))
    (MAPC '(LAMBDA (C)
           (SETQ LIST (CONS (ADD (FIX (DIVIDE C 2)) 1 SUM) LIST))
           (SETQ SUM (ADD SUM C 2))
          )
        CENTRES)
    (RETURN (REVERSE LIST))
 ))



 (DEFUN NLEN (NO)
   ; THIS ROUTINE RETURNS THE PRINT LENGTH OF A NUMBER
   (TAB 1 BUFFER)
   (PRIN1 NO BUFFER 2)
   (PLEN BUFFER)
 )



 ; GENSYM1
 ; LIKE GENSYM, EXCEPT THAT NUMBERING STARTS AT 1 FOR EACH DIFFERENT
 ; CHARACTER STRING C.


 (DEFUN GENSYM1 (C)
   (COND ( (NULL (GET C 'GS#)) (PUT C 'GS# 0)) )
   (IMPLODE1 C (PUT C 'GS# (ADD1 (GET C 'GS#))) )
 )



 ; IMPLODE1
 ; GIVEN TWO ATOMS A AND B, RETURNS THE ATOM AB.

 (DEFUN IMPLODE1 (A B)
   (TAB 1 IMPLODEBUFFER)
   (PRIN1 A IMPLODEBUFFER 2)
   (PRIN1 B IMPLODEBUFFER 2)
   (READ IMPLODEBUFFER)
 )





 ; PRINT*
 ; THIS ROUTINE WILL PRINT THE ARGUMENT *ONLY* IF TRACE IS ON.

 (DEFUN PRINT* (EXP)
   (OR (NOT TRACE) (PRINT EXP))
 )



 ; PRIN1*
 ; THIS ROUTINE WILL CALL PRIN1 ONLY IF TRACE IS NIL.

 (DEFUN PRIN1* (EXP)
   (OR (NOT TRACE) (PRIN1 EXP))
 )
END OF FILE
```