Order Number 8825957

Indefinite and maybe information in deductive relational databases

Sunderraman, Rajshekhar, Ph.D.

Iowa State University, 1988

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Indefinite and maybe information in deductive relational databases

by

Rajshekhar Sunderraman

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Members of Committee:

Signature was redacted for privacy.

Iowa State University

Ames Iowa

1988

TABLE OF CONTENTS

1 INTRODUCTION

Incomplete information in relational databases has been studied by many researchers since the introduction of the relational model [9]. The different kinds of incomplete information that have been studied include null values [5], [6], [7], [10], [11], [17], [23], [24], [41], [46], [50], partial values [18,19], indefinite/disjunctive information [16,22,40], and maybe information [29,38]. In this thesis, we focus our attention on the indefinite and maybe kinds of incomplete information.

In [40], the model-theoretic and the proof-theoretic approaches to relational databases have been discussed. The model-theoretic approach views a relational database as a unique model for a first-order theory. On the other hand, the proof-theoretic approach views a relational database as a set of well formed formulas constituting a first-order theory. For example, the usual suppliers-parts relational database in Figure 1.1 represents *definite facts* that correspond to the following logical formulas:

SP

s1	p1
s2	p3
s3	p5

P

p1	blue
p2	green
p3	red
p4	red

Figure 1.1: Suppliers-Parts Database

1. $SP(s1, p1)$, $SP(s2, p3)$, $SP(s3, p5)$, and

2. $P(p1, blue)$, $P(p2, green)$, $P(p3, red)$, $P(p4, red)$.

Suppose we want to add the following *disjunctive facts* to the database:

1. Supplier s4 supplies part p3 or part p4,

2. Supplier s5 supplies part p1 or part p5, and

3. Part p5 or part p6 is red.

The proof-theoretic approach allows us to introduce the disjunctive facts as the following logical formulas:

1. $SP(s4, p3) \vee SP(s4, p4)$,

2. $SP(s5, p1) \vee SP(s5, p5)$, and

3. $P(p5, red) \vee P(p6, red)$

into the database. However, it is difficult to represent disjunctive facts using the model-theoretic approach.

Suppose at a later time, we are interested in adding the definite fact: Supplier s5 supplies part p5. In the proof-theoretic approach, the formula $SP(s5, p5)$ is added to the first-order theory. The fact that $SP(s5, p5)$ subsumes the already present formula $SP(s5, p1) \vee SP(s5, p5)$ removes the disjunctive fact $SP(s5, p1) \vee SP(s5, p5)$ from the database. In the process, the information $SP(s5, p1)$, about the possibility of supplier s5 supplying part p1 is lost. However, it is still useful to keep this kind of *maybe information*. In addition, the user may want to add maybe information of his own, such as part p7 is possibly black.

Now consider the query: Find all the suppliers who supply red parts, in the above described database. Since supplier s2 supplies part p3 which is red, s2 qualifies as a definite answer. Since supplier s4 supplies either part p3 or part p4 and since both the parts are red, s4 also qualifies as a definite answer. Supplier s3 supplies part p5, however, we are not sure about the color of part p5. There is a possibility that it is red. So, s3 qualifies as a maybe answer. Finally, since supplier s5 supplies part p5 and the color of part p5 may be red, s5 qualifies as a maybe answer.

In the above example, three kinds of information were discussed: definite, disjunctive/indefinite, and maybe. This paper addresses the problem of representing and manipulating these kinds of information in a relational database viewed through the model-theoretic approach.

The relational model, as illustrated above, is incapable of handling indefinite and maybe information. All the facts represented in a relational database are definite. A tuple, t, in a relation, r, can be viewed as a definite statement $R(t)$, where R is the predicate symbol associated with the relation r. The relation, in turn, can be viewed as a conjunction of definite statements $R(t_1) \wedge \cdots \wedge R(t_n)$, where $t_1, \ldots, t_n$ are the tuples of r. Finally, a relational database can be viewed as a conjunction of conjunctions, one for each relation in the database, of definite statements. In order to be able to represent indefinite and maybe information, we need to extend the notion of a relation.

In Chapter 3, we define a data structure, called an *I-table*, which is capable of representing definite, indefinite, and maybe information. An I-table, T, consists of three components, one for each of the three kinds of information it represents. The *definite component* consists of definite tuples, the *indefinite component* consists of

indefinite tuple sets, and the *maybe component* consists of maybe tuples. A *definite tuple*, t, can be viewed as a definite statement $R(t)$, where R is the predicate symbol associated with the I-table T. An *indefinite tuple set*, $\{t_1,\ldots,t_k\}$, can be viewed as an indefinite statement $R(t_1) \vee \cdots \vee R(t_k)$. With only the definite and indefinite components under consideration, an I-table can be viewed as a conjunction of definite and indefinite statements and a database, which consists of I-tables, can be viewed as a conjunction of conjunctions, one for each I-table, of definite and indefinite statements. The model-theoretic approach to relational databases now views the database as a set of minimal models [36], instead of a unique model, of the underlying first-order theory.

A *maybe tuple*, t, corresponds to the statement $R(t)$. However, this statement is not necessarily true. Due to the nature of maybe tuples, we treat them differently from the definite and indefinite kinds of information. There are two sources for the maybe tuples. First, the user may want to represent tuples that may belong to the relation. Second, the maybe component may consist of tuples that have appeared in the past in tuple sets, and therefore there is more reason to expect them to be in the relation than tuples that have not been mentioned anywhere.

The information content of an I-table is defined, by a mapping REP, to be a set of definite relations that correspond to the minimal models [36] of the underlying first-order theory and a set of maybe tuples. Redundancy in I-tables is discussed and an operator to remove the redundancy is defined. The database in Figure 1.1 augmented with the disjunctive and maybe information, discussed earlier, is shown as I-tables in Figure 1.2.

SP

s1	p1
s2	p3
s3	p5
s5	p5
s4	p3
s4	p4
s5	p1

P

p1	blue
p3	red
p4	red
p5	red
p6	red
p7	black

Figure 1.2: Supplier-Parts Database as I-tables

We extend the relational algebra to operate on I-tables. However, before we extend the relational algebra, we present the correctness criterion that must be satisfied by the extended relational algebra. The correctness criterion is shown to be satisfied by each of the extended algebraic operators. Queries can be expressed in the extended relational algebra and the user may now expect definite, indefinite, and maybe answers. To maintain a smooth flow throughout the paper, we present the proofs to some of the theorems in the Appendix. Some of the results are presented in [31,32,33].

Deductive databases [13,14,15,16] have developed from the application of ideas from first-order logic and relational databases. The term *deductive* denotes the capability of these systems to deduce new facts from known facts and rules while answering user queries. Deductive databases can be viewed as generalizations of relational databases. They not only contain elementary facts but also general rules defining additional facts. Most of the research in deductive databases has focussed on definite deductive databases in which only Horn clauses are allowed. Recursive Horn clauses have been extensively studied in [3,8,21,35,37,45,48]. Indefinite deductive databases,

which allow for non-Horn clauses to be present, have been studied with respect to negation in [36,49]. Reiter [40] shows that the proof-theoretic approach to relational databases can be very general and can incorporate indefinite information easily.

One of the approaches to realize the deductive component of a definite deductive database is to use the relational algebra to implement the deductive component [25,42]. Imielinski [22] uses the algebraic approach for more general logic databases. The algebraic approach has many advantages as efficient features of existing relational database systems such as search algorithms, file organizations, etc. can be effectively used.

In Chapter 4, we show how the extended relational algebra can be used to realize the deductive component of a subclass of indefinite deductive databases, which consists of non-Horn clauses whose positive literals involve the same predicate symbol.

We consider a subclass of indefinite deductive databases. The non-Horn rules are restricted to have positive literals involving the same predicate symbol. Since the non-Horn rules consist of more than one positive literals, we can no longer use the projection operator to evaluate the rule. We extend the projection operator further to handle this situation. Such an operator will be referred to as *project-union*. The selection operator for I-tables does not satisfy the following property which is true for regular relations:

$$\sigma_{F_1 \vee F_2}(T) = \sigma_{F_1}(T) \cup \sigma_{F_2}(T).$$

To avoid problems stemming from this, we generalize the non-Horn clauses to consist of disjunction of literals instead of just literals on the right hand side. The generalized non-Horn clauses will be referred to as *I-rules*. Recursive I-rules are evaluated by repeated application of the algebraic expressions. Some of the results related to the

application of the extended relational algebra to deductive databases are presented in [30].

In Chapter 5, we generalize the concept of I-tables to represent more general disjunctive information. A general data structure, called M-tables, is defined. M-tables are capable of representing disjunctive information such as $P_1(t_1) \vee \cdots \vee P_n(t_n)$, where P_is could be different predicate symbols. The relational algebra is suitably generalized to operate on M-tables. In addition to the algebraic operators, we define two new operators, *R-projection* and *merge*, which are used in answering queries.

2 BACKGROUND MATERIAL

In this chapter, we present some background material. First, we discuss the strong relationship between first-order logic and relational databases. The two logical views of a relational database: the model-theoretic and the proof-theoretic views, are presented. An important generalization of the proof-theoretic view: deductive database, is discussed. The problem of negative information, recursive axioms, and incomplete information are briefly discussed.

2.1 First-Order Logic and Relational Databases

Here, we discuss two logical views of a relational database as described in [40]. We also define definite and indefinite deductive databases, and for each we present an operational definition. We shall use the relational database in Figure 2.1 as an example.

TEACHER	COURSE
A	CS100
B	CS200
C	P200

STUDENT	COURSE
a	CS100
b	CS100
c	CS200
d	P200

Figure 2.1: A Relational Database

2.1.1 Syntax of a first-order language

A *first-order language* is specified by a pair (A, W), where A is an alphabet of

1. zero or more variable symbols,
2. zero or more constant symbols,
3. one or more predicate symbols,
4. punctuation symbols (and), and
5. logical constants $\rightarrow, \neg, \wedge, \vee, \exists$, and $\forall$,

and W is a set of well-formed formulas defined as follows:

1. An atomic formula is a well-formed formula,
2. If W_1 and W_2 are well-formed formulas then so are $W_1 \wedge W_2$, $W_1 \vee W_2$, $W_1 \rightarrow W_2$, and $\neg W_1$,
3. If x is a variable symbol and W is a well-formed formula then so are $(\exists x)(W)$ and $(\forall x)(W)$, and
4. All the well-formed formulas are obtained from 1, 2, and 3,

and an *atomic formula* is of the form $P(x_1, \ldots, x_n)$ where P is a n-ary predicate symbol and $x_1, \ldots, x_n$ are constant or variable symbols. If the arguments of the predicate symbol are all constant symbols then the atomic formula is referred to as a *ground atomic formula.*

A *relational language* is a first-order language (A, W) such that A has the following properties:

1. There are finitely many constants in A (at least one).
2. There are finitely many predicate symbols in A.
3. There is a special predicate symbol, $=$.
4. Among the predicate symbols of A, there is a distinguished subset, possibly empty, of unary predicates, called *simple types.*

2.1.2 Semantics of a first-order language

An *interpretation*, I, for a first-order language $F = (A, W)$ is a triple (D, K, E), where

1. D is a non-empty set, called the *domain* of I,
2. K is a mapping from the constant symbols of A into D, and
3. E is a mapping from the n-ary predicate symbols of A into tuples of elements of D, E(P) $\subseteq D^n$.

An interpretation $I = (D, K, E)$ for a relational language $R = (A, W)$ is a *relational interpretation* if and only if

1. K is a one-one and onto mapping, and
2. $E(=) = \{(d, d) | d \in D\}$.

Example 2.1.1 Let $R = (A, W)$ be a relational language, where A contains the following constant and predicate symbols:

Constants A, B, C, a, b, c, d, CS100, CS200, P200.

Predicates $TEACHER^1$, $COURSE^1$, $STUDENT^1$, $TEACH^2$, $ENROLLED^2$, $=^2$.

Simple Types $TEACHER^1$, $COURSE^1$, $STUDENT^1$.

A relational interpretation for R is (D, K, E), where

D = { A, B, C, a, b, c, d, CS100, CS200, P200 },

K maps the constant symbols into the corresponding domain elements, and

E is shown in Figure 2.2.

TEACHER	COURSE	STUDENT	TEACH		ENROLLED		=	
A	CS100	a	A	CS100	a	CS100	A	A
B	CS200	b	B	CS200	b	CS100	B	B
C	P200	c	C	P200	c	CS200	C	C
		d			d	P200	CS100	CS100
							...	...

Figure 2.2: $E(P)$

Given an interpretation, $I = (D, K, E)$, let ρ, called an *environment*, be a mapping from the variables of A into D. Then, the mapping $||.||_I^\rho$ is defined as follows:

$||c||_I^\rho = K(c)$, for each constant symbol c in A

$||x||_I^\rho = \rho(x)$, for each variable symbol x in A

The truth value of a well-formed formula in an interpretation I and environment ρ is defined as follows:

1. $P(t_1, \ldots, t_n)$ is true in $< I, \rho >$ if and only if $< ||t_1||_I^\rho, \ldots, ||t_n||_I^\rho > \in E(P)$.

2. $W_1 \wedge W_2$ is true if and only if both W_1 and W_2 are true in $< I, \rho >$.

3. $W_1 \vee W_2$ is true if and only if one of W_1 or W_2 is true in $< I, \rho >$.

4. $\neg W_1$ is true in $< I, \rho >$ if and only if W_1 is not true in $< I, \rho >$.

5. $W_1 \rightarrow W_2$ is true in $< I, \rho >$ if and only if $\neg W_1 \vee W_2$ is true in $< I, \rho >$.

6. $(\forall x)(W)$ is true in $< I, \rho >$ if and only if for all $d \in D$, W is true in $< I, \rho' >$, where ρ' is exactly the same as ρ with one exception, ρ' now maps x to d.

7. $(\exists x)(W)$ is true in $< I, \rho >$ if and only if $\neg(\forall x)(\neg W)$ is true in $< I, \rho >$.

Finally, a well-formed formula, W, is true in I if and only if W is true in $< I, \rho >$ for all possible ρs.

2.1.3 Model-theoretic view of a relational database

In the model-theoretic view, a relational database is defined as a triple $DB = (R, I, IC)$, where

1. R is a relational language,

2. I is a relational interpretation, and

3. IC is a set of well-formed formulas, called *integrity constraints.*

For each predicate symbol, P, distinct from $=$, IC must contain

$$(\forall x_1) \cdots (\forall x_n)(P(x_1, \ldots, x_n) \rightarrow T_1(x_1) \wedge \cdots \wedge T_n(x_n))$$

where $T_1, \ldots, T_n$ are simple types and are referred to as the domains of P. The integrity constraints are said to be satisfied if and only if they are true in I.

$E(P)$, for a predicate symbol P other than $=$, corresponds to a relation.

A *query*, Q, for R is of the form

$$\{< x_1,\ldots,x_k > | T_1(x_1) \wedge \cdots \wedge T_n(x_n) \wedge W(x_1,\ldots,x_n)\},$$

where W is a well-formed formula and the only free variables in W are $x_1,\ldots,x_n$ and $T_1,\ldots,T_n$ are simple types.

A tuple $< c_1,\ldots,c_k >$ is an *answer* to a query Q with respect to a database $DB = (R, I, IC)$ if and only if

1. $T_i(c_i)$ is true in I, $1 \leq i \leq k$, and
2. $W(c_1,\ldots,c_k)$ is true in I.

2.1.4 Proof-theoretic view of a relational database

Instead of viewing the relational interpretation I as a set of tables, we can think of it as a set of ground atomic formulas. The proof-theoretic view consists of these ground atomic formulas along with others.

A *relational theory* of a relational language $R = (A, W)$ is a first-order theory $T \subseteq W$ such that T contains the following axioms:

1. *Domain Closure Axiom* $(\forall x)(= (x, c_1) \vee \cdots \vee = (x, c_n))$, where $c_1,\ldots,c_n$ are the constant symbols in A,
2. *Unique Name Axioms* $\neg = (c_i, c_j)$, $1 \leq i \leq n$, $1 \leq j \leq n$, $i \leq j$.
3. *Equality Axioms*:
   - $(\forall x)(= (x, x))$,
   - $(\forall x)(\forall y)(= (x, y) \rightarrow = (y, x))$,
   - $(\forall x)(\forall y)(\forall z)(= (x, y) \wedge = (y, z) \rightarrow = (x, z))$, and

- Principle of substitution:
  $(\forall x_1)\cdots(\forall x_n)\quad (P(x_1,\ldots,x_n)\wedge = (x_1,y_1)\wedge\cdots\wedge = (x_n,y_n)\rightarrow P(y_1,\ldots,y_n))$.

4. *Ground Atomic Formulas*, $\Delta \subseteq W$, such that none of them contains the equality predicate symbol.

   Define $C_P = \{< c_1,\ldots,c_n > | P(c_1,\ldots,c_n) \in \Delta\}$.

5. *Completion Axioms*: Let $C_P = \{< c_1^1,\ldots,c_m^1 >,\cdots,< c_1^r,\ldots,c_m^r >\}$. For each m-ary predicate symbol P,

$$(\forall x_1)\cdots(\forall x_m)(P(x_1,\ldots,x_m)\rightarrow \quad (= (x,c_1^1)\wedge\cdots\wedge = (x_m,c_m^1))\vee\cdots\vee (= (x_1,c_1^r)\wedge\cdots\wedge = (x_m,c_m^r))),$$

Example 2.1.2 For the example relational database of Figure 2.1, T contains:

1. $(\forall x)(= (x,A)\vee\cdots\vee = (x,P200))$.
2. $\neg = (A,B),\ldots$.
3. Equality axioms.
4. $TEACHER(A),\ldots,ENROLLED(d,P200)$.
5. $(\forall x)(TEACHER(x)\rightarrow = (x,A)\vee = (x,B)\vee = (x,C)),\ldots$.

In the proof-theoretic view, a relational database is defined to be a triple $DB = (R,T,IC)$, where R is a relational language, T is a relational theory, and IC is a set of integrity constraints. IC is said to be satisfied in the database DB if and only if $T \models IC$. A *query*, Q, for R is of the form

$$\{< x_1,\ldots,x_k > | T_1(x_1)\wedge\cdots\wedge T_n(x_n)\wedge W(x_1,\ldots,x_n)\},$$

where W is a well-formed formula and the only free variables in W are $x_1,\ldots,x_n$ and $T_1,\ldots,T_n$ are simple types.

A tuple $< c_1,\ldots,c_k >$ is said to be an *answer* to a query Q with respect to $DB = (R,T,IC)$ if and only if

1. $T \models T_i(c_i)$, $1 \leq i \leq k$, and
2. $T \models W(c_1,\ldots,c_k)$.

The following theorem [40] shows that the two views, as defined, are equivalent:

Theorem 2.1.1 *(REITER) Suppose $R = (A,W)$ is a relational language. Then,*

1. *If T is a relational theory for R, then T has a unique model which is a relational interpretation for R.*
2. *If I is a relational interpretation for R, then there is a relational theory, T, such that I is the only model for T.*

The proof-theoretic view can be generalized by adding axioms to it. It is easy to incorporate incomplete information, information about events, hierarchies, and inheritance of properties and aggregations into the proof-theoretic view of a relational database [40].

2.1.5 Deductive databases

A *deductive database* is one of the more important generalizations of the proof-theoretic view in which we add *deductive laws* to the set of axioms that constitute the relational theory. New facts may be derived from facts that were explicitly introduced

and from deductive laws. The general form of clauses that will represent both facts and deductive laws is:

$$P_1, \ldots, P_k \leftarrow Q_1, \ldots, Q_l$$

where P_is and Q_is are atomic formulas. The P_is will be referred to as *left hand side* of the clause and the Q_is will be referred to as *right hand side* of the clause. We shall refer to atomic formulas and their negations as *literals*. The clause is equivalent to

$$P_1 \vee \cdots \vee P_k \vee \neg Q_1 \vee \cdots \vee \neg Q_l.$$

All the variable symbols in the clause are universally quantified and the quantifiers will be omitted for notational convenience. The P_is will be referred to as *positive literals* and the Q_is will be referred to as *negative literals*. If $k = 1$ then the clauses will be referred to as *Horn clauses* and if $k > 1$ then the clauses will be referred to as *non-Horn clauses*. An empty left hand side in a clause is an abbreviation for *false* and an empty right hand side in a clause is an abbreviation for *true*. The different types of clauses and examples are presented below:

Type 1 $k = 1$ and $l = 0$ (*Definite Facts*).

$$TEACH(A, CS100) \leftarrow$$

Type 2 $k = 0$ and $l = 1$ (*Negative Facts*).

$$\leftarrow TEACH(A, P100)$$

Type 3 $k = 0$ and $l > 1$ (*Integrity Constraint*).

$$\leftarrow FATHER(x, y), MOTHER(x, y)$$

Type 4 $k = 1$ and $l \geq 1$ (*Definite Deductive Law/Integrity Constraint*).

$$GRANDMOTHER(x, y) \leftarrow MOTHER(x, z), MOTHER(z, y)$$

Type 5 $k > 1$ and $l = 0$ (*Indefinite Facts*).

$$BG(Tom, A), BG(Tom, B) \leftarrow$$

Type 6 $k > 1$ and $l \geq 1$ (*Indefinite Deductive Law/Integrity Constraint*).

$$BG(x, y), BG(x, z) \leftarrow FATHER(x, u), BG(u, y), MOTHER(x, v), BG(v, z)$$

Definite Deductive Databases (DDDBs): We obtain a *definite deductive database* when we add deductive laws of Type 4 to the set of axioms of the relational theory. The completion axioms are now modified as the following example illustrates:

Example 2.1.3 Let P have the following assertions in T:

1. $P(a, b) \leftarrow$, and
2. $P(c, d) \leftarrow$.

Also let

$$P(x, z) \leftarrow Q(x, y), R(y, z)$$

and

$$P(x, y) \leftarrow S(x, y)$$

be all the clauses in T that imply P. Then the completion axiom for P is:

$$(\forall x)(\forall y)(\forall z)(P(x,y) \rightarrow \begin{array}{l}((=(x,a) \wedge =(x,b)) \vee \\ ((=(x,c) \wedge =(x,d)) \vee \\ (Q(x,y) \wedge R(y,z)) \vee \\ (S(x,y))).\end{array}$$

Operational Definition of DDDB: From an operational point of view, a DDDB consists of elementary definite facts, definite deductive laws, a set of integrity constraints, and a metarule: negation as finite failure to be discussed later. We can avoid the domain closure axioms by restricting to clauses in which all variable symbols in the left hand side are also found somewhere on the right hand side. Such clauses are sometimes referred to as *range-restricted* clauses. The unique-name and completion clauses may be removed if negation is interpreted as finite failure. The equality axioms are no longer needed as we have done away with the domain-closure, unique-name, and completion axioms.

Indefinite Deductive Databases (IDDBs): We obtain an *indefinite deductive database* when we add facts of Type 5 and deductive laws of Type 6 to the set of axioms of a relational theory.

Operational Definition of an IDDB: From an operational point of view, an IDDB consists of elementary definite as well as indefinite facts, definite as well as indefinite deductive laws, a set of definite as well as indefinite integrity constraints, and a metarule: generalized negation as failure, to be discussed later.

Although the proof-theoretic view of relational databases is elegant and expressive, a theorem-prover is needed to perform the deductions. In the case of indefinite deductive databases, such a theorem-prover can prove to be drastically inefficient. As a result, most of the research has concentrated on enhancing the model-theoretic view with the expressiveness of the proof-theoretic view. The deductive components are

realized by the traditional algebraic approaches and other techniques that treat the relational database as a first-order interpretation.

2.2 Negation

Efficient treatment of negative information is an important issue and has been addressed by many researchers. Negative information may overwhelm a system. For example, in a university environment we may know that certain students take a particular course. For the remaining students, presumably large in number, we would be required to list them as not enrolled in that course.

2.2.1 Negation in relational databases

The relational model of data represents positive information only. The assumption here is that the information not explicitly present in the database is not true. A tuple represents the existence of a relationship between its elements. From a failure to find a certain tuple in the relation, the converse of the relationship may be assumed to be true. For example, if no tuple exists to show "supplier s1 supplies part p1" then it is assumed that "supplier s1 does not supply part p1".

2.2.2 Negation in deductive databases

A summary of the relevant results which deals with negation in definite as well as indefinite deductive databases is presented next.

Closed World Assumption (CWA): The closed world assumption [39] states that a negative ground literal $\neg L$ is assumed to be true if we fail to prove L from the

existing set of clauses in the database. The CWA is logically equivalent to adding a new component $DB^{\neg}$ to the database, where

$$DB^{\neg} = \{\neg P(c) | DB \not\models P(c)\}$$

but without having $DB^{\neg}$ stored. When not working under the CWA, we shall say that the open world assumption (OWA) is adopted. The following important theorems have been proven in [39]:

Theorem 2.2.1 *If DB is Horn and consistent then $DB \cup DB^{\neg}$ is also Horn and consistent.*

Theorem 2.2.2 *If $DB \cup DB^{\neg}$ is consistent then the answers to a query under CWA is exactly the same as the answers under OWA.*

The semantic version of the CWA is stated below:

Theorem 2.2.3 *A ground negative atomic formula $\neg P(c)$ can be assumed to be true in a Horn database if and only if $P(c)$ does not belong to the unique minimal model of the Horn database.*

Example 2.2.1 Let $DB = \{P(a), Q(b)\}$. Then the unique minimal model of DB is:

$$\{P(a), Q(b)\}.$$

We may assume $\neg P(b)$ and $\neg Q(a)$.

The CWA as defined for definite deductive databases is not applicable to indefinite deductive databases as the following example illustrates:

Example 2.2.2 Consider a database that consists of the following clauses:

$$CAT(felix) \leftarrow$$

$$BLACK(x), WHITE(x) \leftarrow CAT(x)$$

Since $BLACK(felix)$ cannot be proved, CWA allows us to assume $\neg BLACK(felix)$. Similarly, we can assume $\neg WHITE(felix)$. This results in the following inconsistent database:

$$\neg BLACK(felix) \leftarrow$$

$$\neg WHITE(felix) \leftarrow$$

$$CAT(felix) \leftarrow$$

$$BLACK(x), WHITE(x) \leftarrow CAT(x)$$

Minker [36] extends the CWA to solve the above mentioned problem. Let E be the set of all purely positive (possibly empty) clauses not provable. The generalized closed world assumption (GCWA) states that we can assume $\neg P(x)$ if and only if $P(x) \vee C$ is not provable for any C in E. The semantic version of the GCWA is stated below:

Theorem 2.2.4 *A ground atomic formula $P(c)$ can be assumed to be true in a non-Horn database if $P(c)$ is not present in any minimal model of the non-Horn database.*

Example 2.2.3 Let $DB = \{P(a) \vee P(b), Q(b)\}$. The minimal models of DB are $\{Q(b), P(a)\}$ and $\{Q(b), P(b)\}$. Since $Q(a)$ is not in any minimal model, we can assume $\neg Q(a)$ to be true.

2.3 Recursive Axioms in Definite Deductive Databases

The view mechanism offered by most relational systems is actually a special case of the deductive laws where the views are restricted to be non-recursive. In this

section, we present some discussion on the recursion problem in definite deductive databases.

A Horn clause is *recursive* if it is of the form

$$P_1 \leftarrow \ldots, P_2, \ldots,$$

where P_1 and P_2 both use the same predicate symbol. For example the Horn clause

$$ANCESTOR(x, y) \leftarrow ANCESTOR(x, z), ANCESTOR(z, y)$$

is recursive. A *linear recursive* Horn clause is one in which the recursive literal appears exactly once on the right hand side of the rule.

Recursion can be classified into the following two types:

1. Recursion whose bound does not depend on the database state. The recursive clauses which correspond to this type are referred to as *singular rules.* This kind of recursion is easily solved syntactically.

2. Recursion whose bound depends on the database state. The recursive clauses which correspond to this type are referred to as *non-singular rules.* Examples of this type of recursion is the classical transitive closure of a relation.

2.3.1 Singular rules

Minker and Nicolas [37] define singular rules as follows:

Definition 2.3.1 A recursive rule of the form

$$P \leftarrow P_1, \ldots, P_n, F$$

where $P_1, \ldots, P_n$ are literals that use the same predicate symbol as P and F is a conjunction of literals using non-recursive predicates, is a *singular rule* if and only if

1. Each variable symbol that occurs in a literal P_i and does not occur in P only occurs in P_i, and

2. Each variable in P occurs in the same argument position in any literal P_i where it appears, except in at most one literal P_i that contains all of the variables in P.

In the above definition, the first condition rules out explicit transitivity while the second condition rules out any underlying transitivity relationship.

Example 2.3.1

1. $R(x,y,z) \leftarrow R(x,y,z_1), R(x,y_1,z)$ is singular.

2. $R(x,y,z) \leftarrow R(y_1,x,z), R(x,y,z_1)$ is not singular.

3. $R(x,y,z) \leftarrow R(z,x,y), R(x,y_1,z), Q(x,y,z_1)$ is singular.

Some Useful Definitions: The variables whose values are required in the answer are termed *output variables* and are superscripted with an asterisk. A *substitution* is a set of pairs of variables, $\rho = \{x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n\}$, where x_is are termed *old variables* and y_is are termed *new variables.* The *application* of ρ to an expression E consists of replacing the variables in E which occur as old variables in ρ by the corresponding new variables. The expression so obtained is denoted by $E(\rho)$. A substitution ρ is *safe* if and only if

1. none of the old variables in ρ is an output variable, and

2. all new variables in ρ are different and none of them occur in E.

Given two expressions, E_1 and E_2, which are conjunction of literals, E_1 *subsumes* E_2 if and only if there exists a substitution ρ_1 safe with respect to E_1 and a substitution ρ_2 safe with respect to E_2 such that each literal in $E_1(\rho_1)$ is identical to some literal in $E_2(\rho_2)$.

Halting Condition: A derivation can be stopped while preserving answer completeness immediately after a generated expression that is subsumed by one of its ancestor expressions.

Example 2.3.2 Consider the singular rule:

$$P(z, y) \leftarrow P(y, z), Q(x, y),$$

and the query: $\leftarrow P(u^*, v^*)$. We obtain the following derivation path by repeated backward chaining:

E1: $\leftarrow P(u^*, v^*)$

E2: $\leftarrow P(v^*, u^*), Q(x, v^*)$

E3: $\leftarrow P(u^*, v^*), Q(x, u^*), Q(x, v^*)$

Note that E3 is subsumed by E1. So the Halting Condition allows us to stop the derivation just before generating E3 while still preserving answer completeness. The following useful theorem has been proved in [37]:

Theorem 2.3.1 *Any potentially infinite derivation path induced by a singular rule can be stopped by means of the Halting Condition.*

2.3.2 Non-singular rules

The second type of recursive rules, the non-singular rules, are more interesting as no syntactic solution exists. We discuss a solution to evaluate non-singular rules which forms the core of most of the solutions proposed.

Naive Evaluation: We shall present Naive Evaluation through an example. Consider the follow Horn clauses:

$$ANCESTOR(x, y) \leftarrow PARENT(x, y)$$
$$ANCESTOR(x, y) \leftarrow PARENT(x, z), ANCESTOR(z, y)$$
$$QUERY(x) \leftarrow ANCESTOR(x, d)$$

and the relation PARENT in Figure 2.3. The method consists of compiling into an

PARENT

a	b
a	c
b	d
b	e
c	f
c	g

Figure 2.3: Relation *PARENT*

iterative program the rules that derive $QUERY(x)$. The object program for this example is shown below.

```
begin
    i := 0;
```

$ANCESTOR^0 := PARENT$;

$ANCESTOR^* := PARENT$;

repeat

$ANCESTOR^{i+1} := \Pi_{1,4}(\sigma_{2=3}(PARENT \times ANCESTOR^*))$;

$ANCESTOR^* := ANCESTOR^* \cup ANCESTOR^{i+1}$;

i := i + 1

until (there are no changes to $ANCESTOR^*$);

$ANCESTOR := ANCESTOR^*$;

$QUERY := \Pi_1(\sigma_{2='d'}(ANCESTOR))$

end

The value of $ANCESTOR$ relation after each iteration and the value of $QUERY$ are shown in Figure 2.4.

$ANCESTOR^0$

a	b
a	c
b	d
b	e
c	f
c	g

$ANCESTOR^1$

a	d
a	e
a	f
a	g

$ANCESTOR^2 = \emptyset$

$QUERY$

b
a

Figure 2.4: Relations $ANCESTOR^1$, $ANCESTOR^2$, and $QUERY$

Naive evaluation is the most widely described method in the literature. It has been presented in many papers under different forms. The inference engine of SNIP presented in [35] is in fact an interpreted version of the naive evaluation. The method

presented in [8] is a compiled version of the naive evaluation which works for only linear recursive rules.

2.4 Incomplete Information in Databases

The notion of incompleteness is inherent in the domain of databases. Many attempts have been made to characterize the different kinds of incompleteness. Null values were treated in [10], where a three-valued logic was introduced and a maybe-algebra was defined. Grant [17] improved on Codd's approach. Lipski [29] characterizes two interpretations of a query in the context of an incomplete database: the *external interpretation* in which the query is referred to the real world modeled, in an incomplete way, by the system, and the *internal interpretation* in which the query is referred to the system's knowledge of the real world. The external interpretation of a query has two bounds:

1. the lower bound, which includes all those objects for which we can positively conclude, from the information available in the system, that they are in the external interpretation of the query, and

2. the upper bound, which includes all those objects for which we cannot rule out the possibility of belonging to the external interpretation of the query.

Levesque [27] defines a query language which is capable of obtaining the internal interpretation of a query. Most of the research in incomplete databases, however, has concentrated on the external interpretation of a query.

3 EXTENDED RELATIONAL MODEL

In this chapter, we extend the relational model to represent indefinite and maybe information. A data structure, called *I-tables* is introduced. The information content of an I-table is precisely defined. Redundancy in I-tables is characterized and an operator to remove the redundancy is defined. The relational algebraic operators are extended, in a semantically correct manner, to operate on I-tables. Then, we show how queries can be answered in the extended relational model. The answers to queries may now contain indefinite and maybe tuples. Finally, we give the syntax for simple update operators like insert, delete, and modify.

3.1 Indefinite/Maybe Information

In this section, we introduce I-tables, which are capable of representing definite, indefinite, and maybe information. The I-table is merely an extension of the table representing a relation in the relational model. We use a mapping REP to characterize the information content of an I-table in terms of the various definite relations it represents. We also define the notion of redundancy in I-tables and define an operator, called $REDUCE$, to remove these redundancies. Then, we present some properties of REP and $REDUCE$. Finally, we present an approximate time complexity analysis of the $REDUCE$ operator.

3.1.1 I-tables and their information content

A *domain* is a finite set of values, usually non-empty. The *cartesian product* of domains $D_1,\ldots,D_n$ is denoted by $D_1 \times \cdots \times D_n$ and is the set of all tuples $< a_1,\ldots,a_n >$ such that for any $i \in \{1,\ldots,n\}$, $a_i \in D_i$. A *I-table scheme* is an ordered list of attribute names, $R =< A_1,\ldots,A_n >$. Associated with each attribute name, A_i, is a domain D_i. Then, $T =< T_D, T_I, T_M >$ is an I-table over the scheme R, where

$$
\begin{aligned}
T_D &\subseteq D_1 \times \cdots \times D_n, \\
T_I &\subseteq 2^{D_1 \times \cdots \times D_n} - (\{\emptyset\} \cup \{\{t\} | t \in D_1 \times \cdots \times D_n\}), \text{ and} \\
T_M &\subseteq D_1 \times \cdots \times D_n.
\end{aligned}
$$

Note: We shall use the symbol $\subseteq$ for improper subset and the symbol $\subset$ for proper subset.

T_D is the *definite component* of the I-table and consists of tuples which we will refer to as *definite tuples.* T_I is the *indefinite component* of the I-table and consists of sets of tuples which we will refer to as *indefinite tuple sets.* The indefinite tuple sets correspond to inclusive disjunctions, i.e., it is possible for more than one tuple within a tuple set to be the real world truth. T_M is the *maybe component* of the I-table and consists of tuples which we will refer to as *maybe tuples.*

NOTATION: We shall use the symbols $T, T_1, \ldots$ for I-tables, $t, t_1, \ldots$ for tuples, $w, w_1, \ldots$ for tuple sets, $r, r_1, \ldots$ for relations, $a, b, c, \ldots$ for domain values, and $< U, v >, < U_1, v_1 >, \ldots$ for elements of Σ_R (to be defined later). Also, we shall assume that $T_i =< T_D^i, T_I^i, T_M^i >$.

An I-table can be viewed as consisting of two kinds of information: sure and

maybe. The definite and indefinite components of an I-table represent sure information and the maybe component represents maybe information. The sure components of an I-table represent various definite relations, at least one of which is the real world truth. These definite relations correspond to the various models of the underlying first-order theory [36,40]. Some of these definite relations may correspond to non-minimal models, in the sense that they are subsumed by other definite relations. The information content of an I-table consists of two components: the sure component, which consists of definite relations that correspond to the minimal models of the underlying first-order theory, and the maybe component, which consists of all the maybe tuples obtained from the I-table. Given a scheme R, we define Γ_R and Σ_R as follows:

$\Gamma_R = \{T | T$: I-table over $R\}$, and

$\Sigma_R = \{< U, v > | U$: set of relations over R , v : relation over $R\}$.

Now, we are ready to present the formal definition of the information content of an I-table. The information content of an I-table is defined as a mapping, REP, which is the composition of two other mappings, $REDUCEREP$ and $< MM, M >$, defined as follows:

Definition 3.1.1 $< MM, M >: \Gamma_R \rightarrow \Sigma_R$, is a mapping, where

$< MM, M > (T) =< MM(T), M(T) >$,

$T =< T_D, T_I, T_M >$,

$T_I = \{w_1, \ldots, w_n\}$,

$MM(T) = \{T_D \cup \{t_1, \ldots, t_n\} | (\forall i)(1 \leq i \leq n \rightarrow t_i \in w_i)\}$, *and*

$M(T) = T_M$.

$MM(T)$ consists of all the definite relations represented by the sure components of the I-table and $M(T)$ is simply T_M. Note that $MM(T) = \{\emptyset\}$ when $T =< \emptyset, \emptyset, \emptyset >$.

An example of the mapping $< MM, M >$ is given in Figure 3.1.

Definition 3.1.2 $REDUCEREP : \Sigma_R \rightarrow \Sigma_R$, is a mapping, where

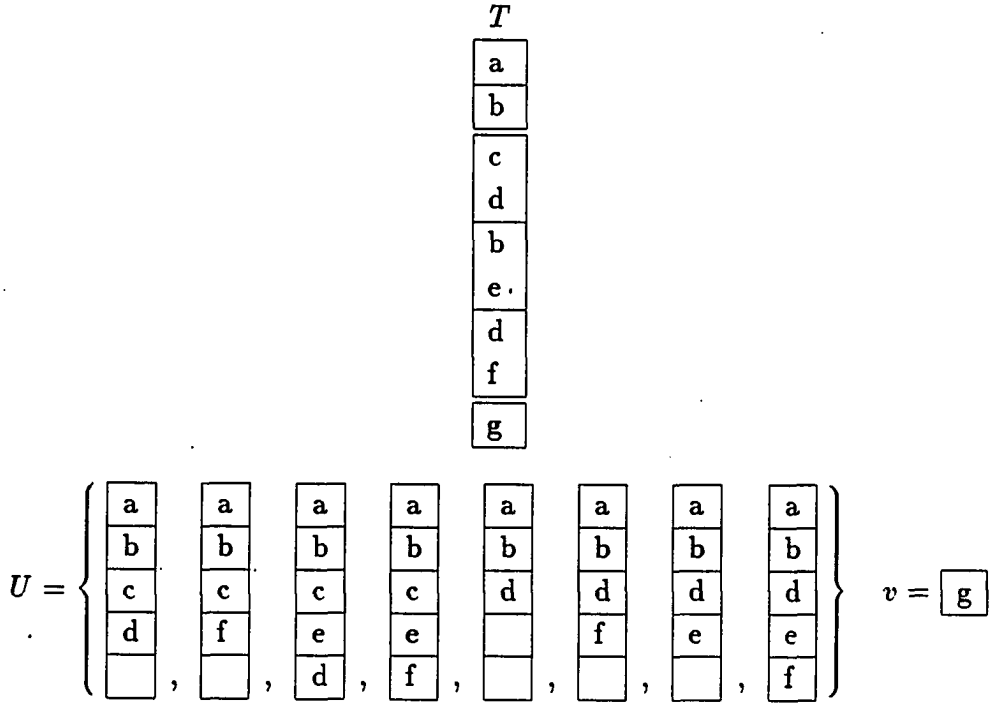


Figure 3.1: $< MM, M > (T) =< U, v >$

$REDUCEREP(< U, v >) =< U^0, v^0 >$,

$U^0 = \{r \mid (r \in U \wedge \neg(\exists r_1)(r_1 \in U \wedge r_1 \subset r)\}$, *and*

$v^0 = \{t \mid (t \in v \vee (\exists r_1)(\exists r_2)(r_1 \in U \wedge r_2 \in U \wedge r_1 \subset r_2 \wedge t \in r_2 - r_1)) \wedge$
$\neg(\exists r)(r \in U^0 \wedge t \in r)\}$.

U^0 is U with all the definite relations that correspond to non-minimal models of the underlying first-order theory removed, and v^0 is v along with some tuples from the definite relations removed from U. Applying $REDUCEREP$ to $< MM, M > (T)$ of

Figure 3.1, we obtain Figure 3.2.

$$U^0 = \left\{ \begin{array}{|c|}\hline a \\ \hline b \\ \hline d \\ \hline \\ \hline \end{array} \; , \; \begin{array}{|c|}\hline a \\ \hline b \\ \hline c \\ \hline f \\ \hline \end{array} \right\} \qquad v^0 = \begin{array}{|c|}\hline e \\ \hline g \\ \hline \end{array}$$

Figure 3.2: $REDUCEREP(< MM, M > (T)) =< U^0, v^0 >$

The following theorem states that $REDUCEREP$ is idempotent:

Theorem 3.1.1 *For any* $< U, v > \in \Sigma_R$,

$$REDUCEREP(REDUCEREP(< U, v >)) = REDUCEREP(< U, v >).$$

Proof: Follows from definition.

The following lemma can easily be observed:

Lemma 3.1.1 *Let* $< U, v > \in \Sigma_R$ *and* $REDUCEREP(< U, v >) =< U_1, v_1 >$. *Then,*

$$\bigcup_{r \in U} (r) \cup v = \bigcup_{r \in U_1} (r) \cup v_1.$$

Finally, we define the information content of an I-table as follows:

Definition 3.1.3 $REP : \Gamma_R \rightarrow \Sigma_R$, is a mapping, where

$REP(T) = REDUCEREP(< MM, M > (T))$.

$REP(T)$ for the I-table T of Figure 3.1 is shown in Figure 3.2.

Since we are dealing with disjunctive information that correspond to the inclusive or, we need the following definition:

Definition 3.1.4 Let U be a set of relations over the scheme R. Then,

$$POSS(U) = \{r \mid (\exists k)(1 \leq k \leq |U| \wedge (\exists r_1)\cdots(\exists r_k)(r_1 \in U \wedge \cdots \wedge r_k \in U \wedge r = r_1 \cup \cdots \cup r_k))\}.$$

Given $REP(T) =< U, v >$, $POSS(U)$ represents all the different real world possibilities represented by the I-table T, including those that correspond to the possibility of more than one relation in U being the real world truth.

3.1.2 Redundancy in I-tables

In this section, we first characterize the different kinds of redundant information that could be found in an I-table. Then, we introduce an operator, called $REDUCE$, which removes these redundancies.

We have identified the following four kinds of redundant information that could be present across the components of an I-table, $T =< T_D, T_I, T_M >$:

1. $t \in T_D$ and $w \in T_I$ and $t \in w$. Here, a definite statement is part of an indefinite statement. This redundancy is removed by deleting w from T_I and including in T_M all the tuples in $w - \{t\}$.

2. $w_1 \in T_I$ and $w_2 \in T_I$ and $w_1 \subset w_2$. Here, an indefinite statement is part of another indefinite statement. This redundancy is removed by deleting w_2 from T_I and including in T_M all the tuples in $w_2 - w_1$.

3. $t \in T_M$ and $t \in T_D$. Here, a maybe statement is also a definite statement. This redundancy is removed by simply deleting t from T_M.

4. $t \in T_M$ and $t \in w$ and $w \in T_I$. Here, a maybe statement is part of an indefinite statement. This redundancy is removed by simply deleting t from T_M.

Note that the first two kinds of redundancies correspond to the subsumption of an indefinite fact by either a definite or another indefinite fact. The last two kinds of redundancies correspond to the appearance of a maybe fact as a definite fact or in an indefinite fact. We now define an operator, called $REDUCE$, which takes in as input an I-table and returns the I-table with all the redundancies removed. $REDUCE$ is defined as a mapping $REDUCE : \Gamma_R \rightarrow \Gamma_R$ as follows:

Definition 3.1.5 Let T be an I-table. Then, $REDUCE(T) = T^0$, where T^0 is defined as follows:

$$T_D^0 = \{t \mid t \in T_D\},$$
$$T_I^0 = \{w \mid (w \in T_I \wedge \neg(\exists t)(t \in T_D \wedge t \in w) \wedge \neg(\exists w_1)(w_1 \in T_I \wedge w_1 \subset w)\}, \text{ and}$$
$$T_M^0 = \{t \mid (t \in A) \wedge (t \notin T_D^0) \wedge \neg(\exists w)(w \in T_I^0 \wedge t \in w)\},$$

where A is defined as follows:

$$A = \{t \mid (t \in T_M) \vee$$
$$(\exists t_1)(\exists w)(t_1 \in T_D \wedge w \in T_I \wedge t_1 \in w \wedge t \in w - \{t_1\}) \vee$$
$$(\exists w_1)(\exists w_2)(w_1 \in T_I \wedge w_2 \in T_I \wedge w_1 \subset w_2 \wedge t \in w_2 - w_1)\}.$$

An example of the $REDUCE$ operator is shown in Figure 3.3.

We shall refer to a non-redundant I-table as a *reduced I-table*. The following lemma can easily be deduced from the definition of $REDUCE$:

Lemma 3.1.2 *Let T be an I-table and let $T^0 = REDUCE(T)$. Then,*

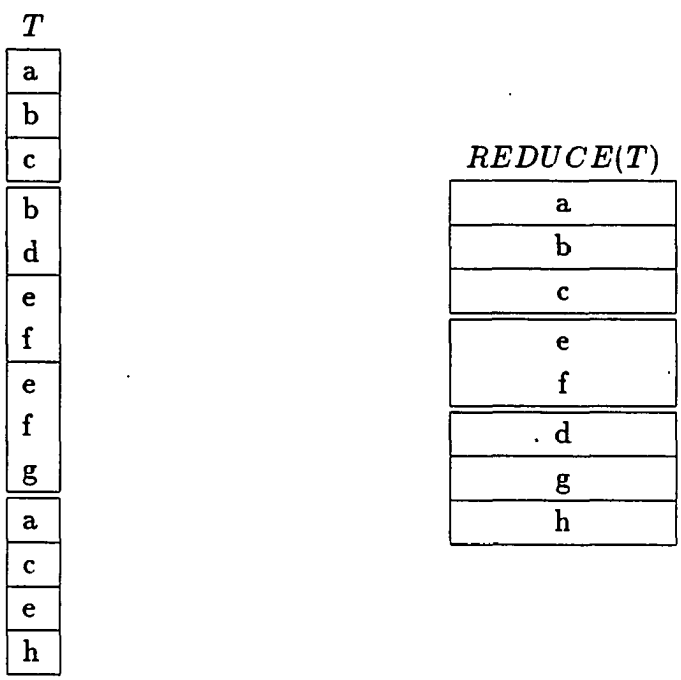


Figure 3.3: $REDUCE(T)$

$$T_D \cup \bigcup_{w \in T_I} (w) \cup T_M = T_D^0 \cup \bigcup_{w \in T_I^0} (w) \cup T_M^0.$$

3.1.3 Some properties

In this section, we present some properties of the $REDUCE$ operator and the mapping REP.

The following theorem states that $REDUCE$ is idempotent:

Theorem 3.1.2 *For any I-table* $T \in \Gamma_R$,

$$REDUCE(REDUCE(T)) = REDUCE(T).$$

Proof: Follows from definition.

The next theorem establishes the fact that $REDUCE$ neither creates nor destroys any information.

Theorem 3.1.3 *For any I-table* $T \in \Gamma_R$,

$$REP(REDUCE(T)) = REP(T).$$

Theorem 3.1.3 is illustrated in Figure 3.4.

Figure 3.5 shows $REP(T)$ and $REP(REDUCE(T))$ for the I-table T of Figure 3.3. However, $REDUCEREP(< MM, M > (T)) \neq < MM, M > (REDUCE(T))$, as Figure 3.6 illustrates.

The mappings REP and $REDUCE$ induce the following equivalence relations over Γ_R:

Definition 3.1.6 For any two I-tables T_1 and T_2 in Γ_R,

$$T_1 \equiv^{REP} T_2 \text{ if and only if } REP(T_1) = REP(T_2).$$

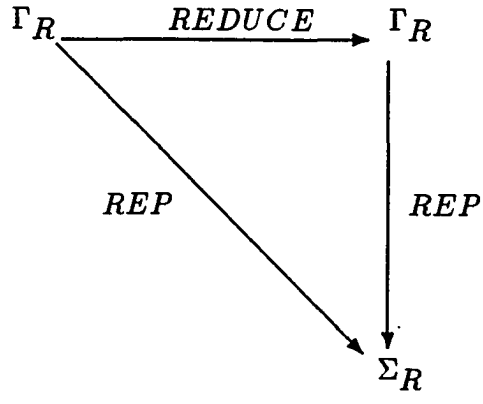
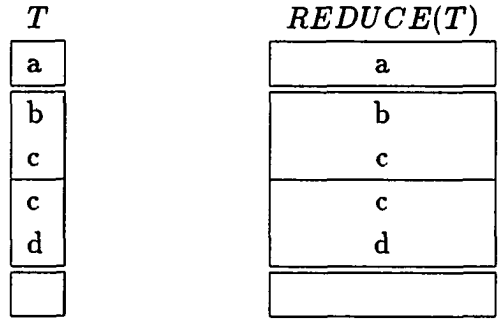


Figure 3.4: $REP(T) = REP(REDUCE(T))$

$$U = \left\{ \begin{array}{|c|}\hline a \\ \hline b \\ \hline c \\ \hline e \\ \hline \end{array} \;,\; \begin{array}{|c|}\hline a \\ \hline b \\ \hline c \\ \hline f \\ \hline \end{array} \right\} \qquad v = \begin{array}{|c|}\hline d \\ \hline g \\ \hline h \\ \hline \end{array}$$

Figure 3.5: $REP(T) = REP(REDUCE(T)) =< U, v >$



$$< MM, M > (T) =< MM, M > (REDUCE(T)) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|}\hline a \\ \hline b \\ \hline c \\ \hline \end{array} , \begin{array}{|c|}\hline a \\ \hline b \\ \hline d \\ \hline \end{array} , \begin{array}{|c|}\hline a \\ \hline c \\ \hline \quad \\ \hline \end{array} , \begin{array}{|c|}\hline a \\ \hline c \\ \hline d \\ \hline \end{array} \right\} \quad v_1 = \emptyset$$

$$REDUCEREP(< MM, M > (T)) =< U_2, v_2 >$$

$$U_2 = \left\{ \begin{array}{|c|}\hline a \\ \hline c \\ \hline \quad \\ \hline \end{array} , \begin{array}{|c|}\hline a \\ \hline b \\ \hline d \\ \hline \end{array} \right\} \quad v_2 = \emptyset$$

Figure 3.6: $REDUCEREP \circ < MM, M >, < MM, M > \circ REDUCE$

Definition 3.1.7 For any two I-tables T_1 and T_2 in Γ_R,

$$T_1 \equiv^{REDUCE} T_2 \text{ if and only if } REDUCE(T_1) = REDUCE(T_2).$$

The following theorem establishes the relationship between the two mappings REP and $REDUCE$:

Theorem 3.1.4 *For any two I-tables T_1 and T_2 from Γ_R.*

$$REP(T_1) = REP(T_2) \text{ if and only if } REDUCE(T_1) = REDUCE(T_2).$$

Corollary 3.1.1 $\equiv^{REP} = \equiv^{REDUCE}$.

Given a scheme R, we can compare I-tables over R with respect to the information contained in them. We present the syntactic and the semantic versions of *weaker I-tables* in the following two definitions:

Definition 3.1.8 Let T_1 and T_2 be two I-tables defined over the scheme R and let $T_3 = REDUCE(T_1)$ and $T_4 = REDUCE(T_2)$. Then, T_1 is *weaker* than T_2, written $T_1 \leq T_2$, if and only if

1. $T_D^3 \subseteq T_D^4$,
2. $(\forall w)(w \in T_I^3 \rightarrow ((\exists t)(t \in T_D^4 \wedge t \in w) \vee (\exists w_1)(w_1 \in T_I^4 \wedge w_1 \subseteq w)))$, and
3. $(\forall t)(t \in T_M^3 \rightarrow (t \in T_D^4) \vee (\exists w)(w \in T_I^4 \wedge t \in w) \vee (t \in T_M^4)))$.

Definition 3.1.9 Let T_1 and T_2 be two I-tables defined over the scheme R and let $REP(T_1) = < U_1, v_1 >$ and $REP(T_2) = < U_2, v_2 >$. Then, T_1 is *weaker* than T_2, written $T_1 \leq T_2$, if and only if

1. $(\forall r_2)(r_2 \in U_2 \rightarrow (\exists r_1)(r_1 \in U_1 \wedge r_1 \subseteq r_2))$, and

2. $v_1 \subseteq (\bigcup_{r \in U_2} (r) \cup v_2)$.

It can easily be shown that the above two definitions are equivalent. Informally, $T_1 \leq T_2$ means that all the information present in T_1 can also be deduced from T_2.

Example 3.1.1 In Figure 3.7, $T_1 \leq T_2$. Note that the empty I-table $< \emptyset, \emptyset, \emptyset >$ is weaker than all I-tables.

Definition 3.1.10 Let T_1 and T_2 be two I-tables defined over the scheme R. Then, $T_1 \equiv T_2$ if and only if $T_1 \leq T_2$ and $T_2 \leq T_1$.

The following theorem can easily be observed:

Theorem 3.1.5 $\equiv = \equiv^{REP}$.

3.1.4 Time complexity of the *REDUCE* operator

In this section, we present an approximate analysis of the time complexity of the *REDUCE* operator. Let T be an I-table and let n_D be the number of tuples in T_D, n_I the number of tuple sets in T_I, and n_M the number of tuples in T_M. We shall assume that the size of the largest tuple set is k, usually a small integer, a constant. For convenience, we shall assume that T_D consists of singleton sets of tuples instead of tuples and shall refer to $T_D \cup T_I$ as T_{sure}. Note that the maximum number of tuples in T_{sure} is $n_D + k\, n_I$. We now present an algorithm for *REDUCE*.

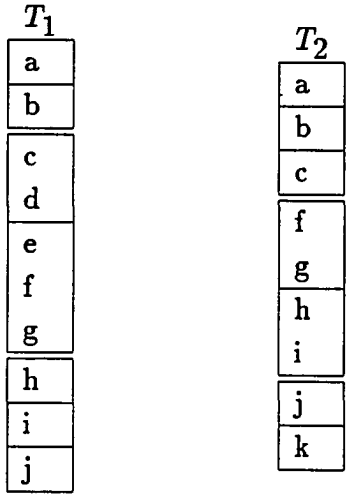
Algorithm 2.1 REDUCE

Input: An I-table T_1

Output: $T = REDUCE(T_1)$

Method:

Step 1: Sort T^1_{sure} as follows:



$$REP(T_1) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{cccccc} a & a & a & a & a & a \\ b & b & b & b & b & b \\ c & c & c & d & d & d \\ e & f & g & e & f & g \end{array} \right\} \quad v_1 = \begin{array}{|c|}\hline h \\ \hline i \\ \hline j \\ \hline \end{array}$$

$$REP(T_2) =< U_2, v_2 >$$

$$U_2 = \left\{ \begin{array}{cccc} a & a & a & a \\ b & b & b & b \\ c & c & c & c \\ f & f & g & g \\ h & i & h & i \end{array} \right\} \quad v_2 = \begin{array}{|c|}\hline j \\ \hline k \\ \hline \end{array}$$

Figure 3.7: I-tables T_1, T_2 and their $REPs$

Step 1.1: First, sort the tuple sets in the increasing order of their sizes (number of tuples) to obtain k groups of tuple sets, where k is the size of the largest tuple set.

Step 1.2: Next, sort the tuples within the tuple sets.

Step 2: Traverse T^1_{sure} using k pointers, one for each group of tuple sets, and in the process collect, in T_{sure}, tuple sets that are not proper subsets of other tuple sets. Also collect, in A, any tuples that are present in tuple set $u \in T^1_{sure}$ and not in tuple set $v \in T^1_{sure}$ such that $v \subset u$.

Step 3: Sort $A \cup T^1_M$ and then delete any tuple in $A \cup T^1_M$ that is also present anywhere in T_{sure}. This will result in T_M.

Let us assume that we employ an $O(nlogn)$ sorting algorithm. The time taken to sort the tuple sets in the increasing order of their sizes (Step 1.1) is of the order of $(n_D + n_I)log(n_D + n_I)$ and the time taken to sort the tuples within the tuple sets (Step 1.2) is of the order of n_I, assuming that it takes constant time to sort the tuples within each tuple set. The time taken to obtain T_{sure} and A in Step 2 is proportional to $n_D + n_I$. Finally, the time taken to sort $A \cup T^1_M$ in Step 3 is of the order of $(n_I + n_M)log(n_I + n_M)$ and to delete tuples from $A \cup T^1_M$ that are not present anywhere in T_{sure} is of the order $(n_D + n_I + n_M)$.

Using the above estimates, we conclude that the time complexity of $REDUCE$ is

$$O((n_D + n_I)log(n_D + n_I) + (n_I + n_M)log(n_I + n_M)).$$

3.2 Extended Relational Algebra

In this section, we first discuss the notion of correctness of extended relational algebraic operations on I-tables. Then, for each algebraic operator, we first present the definition on Σ_R and then the definition on Γ_R that satisfies the correctness criterion. We shall use the same symbol to represent the regular relational operator, the operator on Σ_R, and the operator on Γ_R. The operator in question will be determined by its operands.

3.2.1 Notion of correctness of extended relational algebra

As has been defined earlier, the mapping REP maps an I-table, T, over scheme R, to elements of Σ_R. $REP(T)$ consists of two components: U, a set of definite relations at least one of which represents the real world truth, and v, a set of maybe tuples. Now, consider a relational algebraic operator, f. In order to extend f to operate on I-tables, we must ensure that the extended operator captures the effect of the corresponding regular operator on the various definite relations represented by the I-tables. This notion of correctness is captured in Figure 3.8. For each operator, we first need to define f_Σ on Σ_R and then define f_Γ on Γ_R, that satisfies the following correctness criterion illustrated by Figure 3.8:

1. $REP(f_\Gamma(T)) = f_\Sigma(REP(T))$, for unary f, and

2. $REP(f_\Gamma(T_1, T_2)) = f_\Sigma(REP(T_1), REP(T_2))$, for binary f.

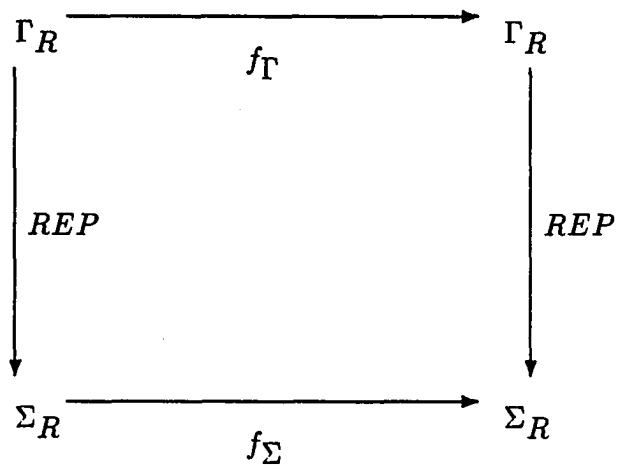


Figure 3.8: Commutativity of REP and f

3.2.2 Selection

First we define selection on elements of Σ_R, as a mapping, $\sigma_F : \Sigma_R \rightarrow \Sigma_R$.

Definition 3.2.1 Let $< U_1, v_1 > \in \Sigma_R$. Then,

$\sigma_F(< U_1, v_1 >) = REDUCEREP(\sigma_F^0(< U_1, v_1 >))$, where

$\sigma_F^0(< U_1, v_1 >) =< U, v >$,

$U = \{r|(\exists r_1)(r_1 \in POSS(U_1) \wedge r = \sigma_F(r_1))\}$, *and*

$v = \sigma_F(v_1)$.

The property: $r_1 \subseteq r_2$ implies $\sigma_F(r_1) \subseteq \sigma_F(r_2)$ and the definition of $REDUCEREP$ allows us to simplify the above definition into the following equivalent definition:

Definition 3.2.2 Let $< U_1, v_1 > \in \sigma_R$. Then,

$\sigma_F(< U_1, v_1 >) = REDUCEREP(\sigma_F^0(< U_1, v_1 >))$, where

$\sigma_F^0(< U_1, v_1 >) =< U, v >$,

$U = \{r|(\exists r_1)(r_1 \in U_1 \wedge r = \sigma_F(r_1))\}$ *and*

$v = \sigma_F(v_1)$.

The following theorem shows that the selection on Σ_R commutes with $REDUCEREP$:

Theorem 3.2.1 *For any* $< U, v > \in \Sigma_R$,

$$\sigma_F(< U, v >) = \sigma_F(REDUCEREP(< U, v >)).$$

Next, we define selection of I-tables. The definite and maybe tuples that satisfy the selection condition are included in the respective components of the selection. If all tuples within a tuple set satisfy the selection condition then the tuple set is included in the selection. Otherwise, only those tuples within the tuple set which satisfy the selection condition are included in the maybe component of the selection. Redundancies introduced are removed with the $REDUCE$ operator. Formally, selection of I-tables is defined as a mapping, $\sigma_F : \Gamma_R \rightarrow \Gamma_R$, as follows:

Definition 3.2.3 Let T_1 be an I-table and F be a formula involving operands that are constants or attribute numbers, arithmetic comparison operators: $<, =, >, \leq, \geq, \neq$, and logical operators $\wedge, \vee$, and $\neg$. Then, $\sigma_F(T_1) = REDUCE(T)$, where

$$T_D = \{t \mid t \in T_D^1 \wedge F(t)\},$$

$$T_I = \{w \mid w \in T_I^1 \wedge (\forall t)(t \in w \rightarrow F(t))\},$$

$$T_M = \{t \mid (t \in T_M^1 \wedge F(t)) \vee (\exists w)(w \in T_I^1 \wedge t \in w \wedge F(t))\},$$

and $F(t)$ is F with attribute number i replaced by $t[i]$.

Remark: Consider the bloodgroup I-table in Figure 3.9 and the query: Find all the persons with bloodgroup "A" or "O". The query expressed in the extended relational algebra is:

$$\Pi_1(\sigma_{(2="A")\vee(2="O")}(BG))$$

BG

Tom	A
Gary	O
John	A
John	O
Tim	A

Figure 3.9: Bloodgroup I-table *BG*

and the answer to the query includes "Tom", "Gary", and "John" as definite answers and "Tim" as a maybe answer. However, if we express the query as:

$$\Pi_1(\sigma_{(2="A")}(BG)) \cup \Pi_1(\sigma_{(2="O")}(BG)),$$

the answer will include "Tom" and "Gary" as definite answers and "Tim" and "John" as maybe answers. Note that "John" qualifies as a definite answer in the first case and as a maybe answer in the second case. The reason for this discrepancy is that the evaluation of one of the sub-conditions, (2="A") or (2="O"), in the second case ignores the effect of the other sub-condition if the two were to be evaluated together. This observation has been noted by Lipski in [29]. According to [29], a query is interpreted in two ways: the external interpretation where the query is referred directly to the real world modeled by the system, and the internal interpretation where the query is referred to the system's information about the real world. The external interpretation has two bounds: $||Q||_*$ which corresponds to the sure answers and $||Q||^*$ which corresponds to the answers that cannot be ruled out. It has been noted that

$$||C_1 \vee C_2||_* \neq ||C_1||_* \cup ||C_2||_*.$$

The following theorem shows that the selection of I-tables commutes with $REDUCE$:

Theorem 3.2.2 *For any I-table T,*

$$\sigma_F(T) = \sigma_F(REDUCE(T)).$$

The correctness of the selection operator is established in the following theorem:

Theorem 3.2.3 *For any reduced I-table T and formula F,*

$$REP(\sigma_F(T)) = \sigma_F(REP(T)).$$

Corollary 3.2.1 *For any I-table T and formula F,*

$$REP(\sigma_F(T)) = \sigma_F(REP(T)).$$

Theorem 3.2.3 is illustrated in Figure 3.10.

3.2.3 Projection

We first define projection on Σ_R, as a mapping, $\Pi_A : \Sigma_R \rightarrow \Sigma_A$.

Definition 3.2.4 Let $< U_1, v_1 > \in \Sigma_R$. Then,

$\Pi_A(< U_1, v_1 >) = REDUCEREP(\Pi^0_A(< U_1, v_1 >))$, where

$\Pi^0_A(< U_1, v_1 >) =< U, v >$,

$U = \{r|(\exists r_1)(r_1 \in POSS(U_1) \wedge r = \Pi_A(r_1))\}$, *and*

$v = \Pi_A(v_1)$.

The property: $r_1 \subseteq r_2$ implies $\Pi_A(r_1) \subseteq \Pi_A(r_2)$ and the definition of $REDUCEREP$ allows us to simplify the above definition into the following equivalent definition:

Definition 3.2.5 Let $< U_1, v_1 > \in \Sigma_R$. Then,

T

a1	b1
a2	b1
a2	b2
a3	b2
a3	b3
a4	b2
a5	b4

$\sigma_{2="b1"\vee 2="b2"}(T)$

a1	b1
a2	b1
a2	b2
a3	b2
a4	b2

$$REP(T) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a2} & \text{b1} \\ \hline \text{a3} & \text{b2} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a2} & \text{b1} \\ \hline \text{a3} & \text{b3} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a2} & \text{b2} \\ \hline \text{a3} & \text{b2} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a2} & \text{b2} \\ \hline \text{a3} & \text{b3} \\ \hline \end{array} \right\} \quad v_1 = \begin{array}{|c|c|}\hline \text{a4} & \text{b2} \\ \hline \text{a5} & \text{b4} \\ \hline \end{array}$$

$$REP(\sigma_{2="b1"\vee 2="b2"}(T)) = \sigma_{2="b1"\vee 2="b2"}(REP(T)) =< U, v >$$

$$U = \left\{ \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a2} & \text{b1} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a2} & \text{b2} \\ \hline \end{array} \right\} \quad v = \begin{array}{|c|c|}\hline \text{a3} & \text{b2} \\ \hline \text{a4} & \text{b2} \\ \hline \end{array}$$

Figure 3.10: Selection

$\Pi_A(< U_1, v_1 >) = REDUCEREP(\Pi^0_A(< U_1, v_1 >))$, where
$\Pi^0_A(< U_1, v_1 >) =< U, v >$,

$U = \{r|(\exists r_1)(r_1 \in U_1 \wedge r = \Pi_A(r_1))\}$, *and*

$v = \Pi_A(v_1)$.

The next theorem shows that the projection on Σ_R commutes with $REDUCEREP$:

Theorem 3.2.4 *For any* $< U, v > \in \Sigma_R$,

$$\Pi_A(< U, v >) = \Pi_A(REDUCEREP(< U, v >)).$$

Next, we define projection of I-tables. The projection of I-tables is quite similar to the regular projection. Some tuple sets may become singletons on projection, in which case they are moved over to the definite component of the projection. Formally, projection is defined as a mapping, $\Pi_A : \Gamma_R \rightarrow \Gamma_A$, as follows:

Definition 3.2.6 Let T_1 be an I-table and let A be a list of attribute numbers. Then, $\Pi_A(T_1) = REDUCE(T)$, where

$$\begin{aligned} T_D = \{t \mid\ & (\exists t_1)(t_1 \in T^1_D \wedge t[A] = t_1[A]) \vee \\ & (\exists w)(w \in T^1_I \wedge (\forall t_1)(t_1 \in w \rightarrow t[A] = t_1[A]))\}, \\ T_I = \{w \mid\ & (\exists w_1)(w_1 \in T^1_I \wedge w = \Pi_A(w_1) \wedge |w| > 1)\}, \text{ and} \\ T_M = \{t \mid\ & (\exists t_1)(t_1 \in T^1_M \wedge t[A] = t_1[A])\}. \end{aligned}$$

The following theorem shows that the projection of I-tables commutes with $REDUCE$:

Theorem 3.2.5 *For any I-table* T,

$$\Pi_A(T) = \Pi_A(REDUCE(T)).$$

The correctness of the projection operator is established in the following theorem:

Theorem 3.2.6 *For any reduced I-table T and list of attributes A,*

$$REP(\Pi_A(T)) = \Pi_A(REP(T)).$$

Corollary 3.2.2 *For any I-table T and list of attributes A,*

$$REP(\Pi_A(T))\Pi_A(REP(T)).$$

Theorem 3.2.6 is illustrated in Figure 3.11.

3.2.4 Cartesian product

We first define cartesian product of elements of Σ_{R_1} with elements of Σ_{R_2}, as a mapping, $\times : \Sigma_{R_1}, \Sigma_{R_2} \rightarrow \Sigma_{R_1.R_2}$.

Definition 3.2.7 Let $< U_1, v_1 > \in \Sigma_{R_1}$ and $< U_2, v_2 > \in \Sigma_{R_2}$. Then,

$< U_1, v_1 > \times < U_2, v_2 > = REDUCEREP(< U_1, v_1 > \times^0 < U_2, v_2 >)$, where

$< U_1, v_1 > \times^0 < U_2, v_2 > = < U, v >$,

$U = \{r | (\exists r_1)(\exists r_2)(r_1 \in POSS(U_1) \wedge r_2 \in POSS(U_2) \wedge r = r_1 \times r_2)\}$, *and*

$$v = \bigcup_{r \in U_1} (r \times v_2) \cup \bigcup_{r \in U_2} (v_1 \times r) \cup (v_1 \times v_2).$$

The property: $r_1 \subseteq r_2$ implies $r \times r_1 \subseteq r \times r_2$ and the definition of $REDUCEREP$ allows us to simplify the above definition into the following equivalent definition:

Definition 3.2.8 Let $< U_1, v_1 > \in \Sigma_{R_1}$ and $< U_2, v_2 > \in \Sigma_{R_2}$. Then,

$< U_1, v_1 > \times < U_2, v_2 > = REDUCEREP(< U_1, v_1 > \times^0 < U_2, v_2 >)$, where

$< U_1, v_1 > \times^0 < U_2, v_2 > = < U, v >$,

$U = \{r | (\exists r_1)(\exists r_2)(r_1 \in U_1 \wedge r_2 \in U_2 \wedge r = r_1 \times r_2)\}$, *and*

$$v = \bigcup_{r \in U_1} (r \times v_2) \cup \bigcup_{r \in U_2} (v_1 \times r) \cup (v_1 \times v_2).$$

T

a1	b1
a2	b1
a2	b2
a2	b3
a3	b3
a4	b1
a5	b1
a6	b1

$\Pi_1(T)$

a1
a2
a4
a5
a3
a6

$$REP(T) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b1 \\ a2 & b3 \\ a4 & b1 \\ \hline \end{array} , \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b1 \\ a2 & b3 \\ a5 & b1 \\ \hline \end{array} , \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b1 \\ a3 & b3 \\ a4 & b1 \\ \hline \end{array} , \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b1 \\ a3 & b3 \\ a5 & b1 \\ \hline \end{array} , \right.$$

$$\left. \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b2 \\ a2 & b3 \\ a4 & b1 \\ \hline \end{array} , \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b2 \\ a2 & b3 \\ a5 & b1 \\ \hline \end{array} , \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b2 \\ a3 & b3 \\ a4 & b1 \\ \hline \end{array} , \begin{array}{|c|c|}\hline a1 & b1 \\ a2 & b2 \\ a3 & b3 \\ a5 & b1 \\ \hline \end{array} \right\}$$

$$v_1 = \begin{array}{|c|c|}\hline a6 & b1 \\ \hline \end{array}$$

$$REP(\Pi_1(T)) = \Pi_1(REP(T)) =< U, v >$$

$$U = \left\{ \begin{array}{|c|}\hline a1 \\ a2 \\ a4 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a2 \\ a5 \\ \hline \end{array} \right\} \quad v = \begin{array}{|c|}\hline a3 \\ a6 \\ \hline \end{array}$$

Figure 3.11: Projection

The following theorem shows that the cartesian product of the elements of Σ_{R_1} with the elements of Σ_{R_2} commutes with *REDUCEREP*:

Theorem 3.2.7 *For any* $< U_1, v_1 > \in \Sigma_R$ *and* $< U_2, v_2 > \in \Sigma_R$,

$$< U_1, v_1 > \times < U_2, v_2 > =$$
$$REDUCEREP(< U_1, v_1 >) \times REDUCEREP(< U_2, v_2 >).$$

Next, we define the cartesian product of I-tables. Consider the two I-tables T_1 and T_2 in Figure 3.12 and let $T = T_1 \times T_2$. T_D is obtained by taking the cartesian product of T_D^1 and T_D^2. T_I is obtained in the following manner: The two disjuncts in the single tuple set of T_I^1 combined with the two definite tuples in T_D^2 give us the following disjunctive logical formula:

$$(\text{T(a2,b1)} \wedge \text{T(a2,b2)}) \vee (\text{T(a3,b1)} \wedge \text{T(a3,b2)})$$

Converting this formula into a conjunct of disjuncts, we obtain the following conjunctive formula:

$$(\text{T(a2,b1)} \vee \text{T(a3,b1)}) \wedge (\text{T(a2,b1)} \vee \text{T(a3,b2)}) \wedge$$
$$(\text{T(a2,b2)} \vee \text{T(a3,b1)}) \wedge (\text{T(a2,b2)} \vee \text{T(a3,b2)})$$

which corresponds to the tuple sets of T_I. T_M is obtained by taking the cartesian product of the following pairs of sets:

1. T_D^1 and T_M^2,
2. each tuple set of T_I^1 and T_M^2,
3. T_M^1 and T_M^2,

4. T_M^1 and T_D^2, and

5. T_M^1 and each tuple set of T_I^2.

The cartesian product of T_1 and T_2 is shown in Figure 3.12. Cartesian product of I-tables is formally defined as a mapping, $\times : \Gamma_{R_1}, \Gamma_{R_2} \rightarrow \Gamma_{R_1.R_2}$, as follows:

Definition 3.2.9 Let T_1 and T_2 be two I-tables such that $T_I^1 = \{w_1^1, \ldots, w_m^1\}$ and $T_I^2 = \{w_1^2, \ldots, w_n^2\}$. Let

$E = \{\{t_1, \ldots, t_m\} | (\forall i)(1 \leq i \leq m \rightarrow t_i \in w_i^1)\}$, *and*

$F = \{\{t_1, \ldots, t_n\} | (\forall i)(1 \leq i \leq n \rightarrow t_i \in w_i^2)\}$.

Let the elements of E be $E_1, \ldots, E_e$ and those of F be $F_1, \ldots, F_f$. Let

$$\begin{aligned} A_{ij} = \{t \mid\ & (\exists t_1)(\exists t_2)(t_1 \in T_D^1 \wedge t_2 \in F_l \wedge t = t_1.t_2) \vee \\ & (\exists t_1)(\exists t_2)(t_1 \in E_k \wedge t_2 \in T_D^2 \wedge t = t_1.t_2) \vee \\ & (\exists t_1)(\exists t_2)(t_1 \in E_k \wedge t_2 \in F_l \wedge t = t_1.t_2)\}, \end{aligned}$$

where $1 \leq k \leq e$, $1 \leq l \leq f$, $i = k$ if $e \neq 0$ otherwise $i = 0$, and $j = l$ if $f \neq 0$ otherwise $j = 0$. Let $A_1, \ldots, A_g$ be the distinct A_{ij}s. Then,

$T_1 \times T_2 = REDUCE(T)$, where

$$T_D = \{t \mid (\exists t_1)(\exists t_2)(t_1 \in T_D^1 \wedge t_2 \in T_D^2 \wedge t = t_1.t_2)\},$$

$$T_I = \{w \mid (\exists t_1) \cdots (\exists t_g)(t_1 \in A_1 \wedge \cdots \wedge t_g \in A_g \wedge w = \{t_1, \ldots, t_g\})\}, \text{ and}$$

$$\begin{aligned} T_M = \{t \mid\ & (\exists t_1)(\exists t_2)(t_1 \in T_D^1 \wedge t_2 \in T_M^2 \wedge t = t_1.t_2) \vee \\ & (\exists w)(\exists t_1)(w = \{t_2, \ldots, t_k\} \in T_I^1 \wedge t_1 \in T_M^2 \wedge \\ & (t = t_2.t_1 \vee \cdots \vee t = t_k.t_1)) \vee \\ & (\exists t_1)(\exists t_2)(t_1 \in T_M^1 \wedge t_2 \in T_M^2 \wedge t = t_1.t_2) \vee \end{aligned}$$

$$(\exists t_1)(\exists t_2)(t_1 \in T_M^1 \wedge t_2 \in T_D^2 \wedge t = t_1.t_2) \vee$$
$$(\exists t_1)(\exists w)(t_1 \in T_M^1 \wedge w = \{t_2, \ldots, t_k\} \in T_I^2 \wedge$$
$$(t = t_1.t_2 \vee \cdots \vee t = t_1.t_k))\}.$$

The following theorem shows that the cartesian product of I-tables commutes with *REDUCE*:

Theorem 3.2.8 *For any two I-tables* T_1 *and* T_2,

$$T_1 \times T_2 = REDUCE(T_1) \times REDUCE(T_2).$$

The correctness of the cartesian product operator is established in the following theorem:

Theorem 3.2.9 *For any two reduced I-tables* T_1 *and* T_2,

$$REP(T_1 \times T_2) = REP(T_1) \times REP(T_2).$$

Corollary 3.2.3 *For any two I-tables* T_1 *and* T_2,

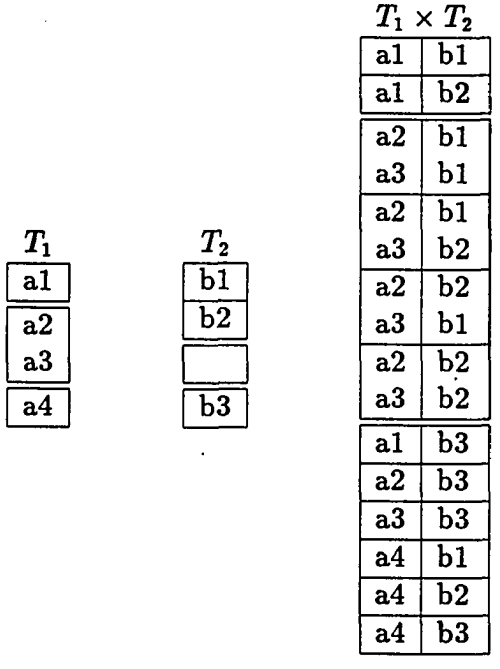
$$REP(T_1 \times T_2) = REP(T_1) \times REP(T_2).$$

Theorem 3.2.9 is illustrated in Figure 3.12.

3.2.5 Union

We first define union on Σ_R, as a mapping, $\cup : \Sigma_R, \Sigma_R \rightarrow \Sigma_R$.

Definition 3.2.10 Let $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$. Then,



$$REP(T_1) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|}\hline \text{a1} \\ \hline \text{a2} \\ \hline \end{array} \; , \; \begin{array}{|c|}\hline \text{a1} \\ \hline \text{a3} \\ \hline \end{array} \right\} \quad v_1 = \boxed{\text{a4}}$$

$$REP(T_2) =< U_2, v_2 >$$

$$U_2 = \left\{ \begin{array}{|c|}\hline \text{b1} \\ \hline \text{b2} \\ \hline \end{array} \right\} \quad v_2 = \boxed{\text{b3}}$$

$$REP(T_1 \times T_2) = REP(T_1) \times REP(T_2) =< U, v >$$

$$U = \left\{ \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a1} & \text{b2} \\ \hline \text{a2} & \text{b1} \\ \hline \text{a2} & \text{b2} \\ \hline \end{array} \; , \; \begin{array}{|c|c|}\hline \text{a1} & \text{b1} \\ \hline \text{a1} & \text{b2} \\ \hline \text{a3} & \text{b1} \\ \hline \text{a3} & \text{b2} \\ \hline \end{array} \right\} \quad v = \begin{array}{|c|c|}\hline \text{a1} & \text{b3} \\ \hline \text{a2} & \text{b3} \\ \hline \text{a3} & \text{b3} \\ \hline \text{a4} & \text{b1} \\ \hline \text{a4} & \text{b2} \\ \hline \text{a4} & \text{b3} \\ \hline \end{array}$$

Figure 3.12: Cartesian Product

$< U_1, v_1 > \cup < U_2, v_2 >= REDUCEREP(< U_1, v_1 > \cup^0 < U_2, v_2 >)$, where
$< U_1, v_1 > \cup^0 < U_2, v_2 >=< U, v >$,

$U = \{r|(\exists r_1)(\exists r_2)(r_1 \in POSS(U_1) \wedge r_2 \in POSS(U_2) \wedge r = r_1 \cup r_2)\}$, *and*

$v = v_1 \cup v_2$.

The property: $r_1 \subseteq r_2$ implies $r \cup r_1 \subseteq r \cup r_2$ and the definition of $REDUCEREP$ allows us to simplify the above definition into the following equivalent definition:

Definition 3.2.11 Let $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$. Then,
$< U_1, v_1 > \cup < U_2, v_2 >= REDUCEREP(< U_1, v_1 > \cup^0 < U_2, v_2 >)$, where
$< U_1, v_1 > \cup^0 < U_2, v_2 >=< U, v >$,

$U = \{r|(\exists r_1)(\exists r_2)(r_1 \in U_1 \wedge r_2 \in U_2 \wedge r = r_1 \cup r_2)\}$, *and*

$v = v_1 \cup v_2$.

The following theorem shows that the union on Σ_R commutes with $REDUCEREP$:

Theorem 3.2.10 *For any* $< U_1, v_1 > \in \Sigma_R$ *and* $< U_2, v_2 > \in \Sigma_R$,

$$< U_1, v_1 > \cup < U_2, v_2 >=$$
$$REDUCEREP(< U_1, v_1 >) \cup REDUCEREP(< U_2, v_2 >).$$

Next, we define union of I-tables. The union of two I-tables is the union of the corresponding components of the two operands. Any redundancies introduced is removed by the $REDUCE$ operator. Formally, union is defined as a mapping, $\cup : \Gamma_R, \Gamma_R \rightarrow \Gamma_R$.

Definition 3.2.12 Let T_1 and T_2 be two domain-compatible I-tables. Then,
$T_1 \cup T_2 = REDUCE(T)$, where

$T_D = \{t|t \in T_D^1 \vee t \in T_D^2\}$,

$T_I = \{w | w \in T_I^1 \vee w \in T_I^2\}$, *and*

$T_M = \{t | t \in T_M^1 \vee t \in T_M^2\}$.

The following theorem shows that the union of I-tables commutes with $REDUCE$:

Theorem 3.2.11 *For any two domain-compatible I-tables* T_1 *and* T_2,

$$T_1 \cup T_2 = REDUCE(T_1) \cup REDUCE(T_2).$$

The correctness of the union operator is established in the following theorem:

Theorem 3.2.12 *For any two domain compatible reduced I-tables* T_1 *and* T_2,

$$REP(T_1 \cup T_2) = REP(T_1) \cup REP(T_2).$$

Corollary 3.2.4 *For any two domain-compatible I-tables* T_1 *and* T_2,

$$REP(T_1 \cup T_2) = REP(T_1) \cup REP(T_2).$$

Theorem 3.2.12 is illustrated in Figure 3.13.

3.2.6 Difference

We first define difference on Σ_R, as a mapping, $- : \Sigma_R, \Sigma_R \rightarrow \Sigma_R$.

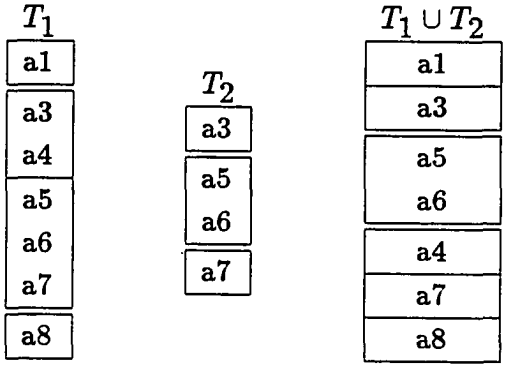
Definition 3.2.13 Let $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$. Then,

$< U_1, v_1 > - < U_2, v_2 > = REDUCEREP(< U_1, v_1 > -^0 < U_2, v_2 >)$, where

$< U_1, v_1 > -^0 < U_2, v_2 > = < U, v >$,

$U = \{r | (\exists r_1)(\exists r_2)(r_1 \in POSS(U_1) \wedge r_2 \in POSS(U_2) \wedge r = (r_1 - r_2) - v_2)\}$, *and*

$v = (\bigcup_{r \in U_1} (r) \cup v_1) - \bigcap_{r \in U_2} (r)$.



$$REP(T_1) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|}\hline a1 \\ a3 \\ a5 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a3 \\ a6 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a3 \\ a7 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a4 \\ a5 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a4 \\ a6 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a4 \\ a7 \\ \hline \end{array} \right\} \quad v_1 = \begin{array}{|c|}\hline a8 \\ \hline \end{array}$$

$$REP(T_2) =< U_2, v_2 >$$

$$U_2 = \left\{ \begin{array}{|c|}\hline a3 \\ a5 \\ \hline \end{array} , \begin{array}{|c|}\hline a3 \\ a6 \\ \hline \end{array} \right\} \quad v_2 = \begin{array}{|c|}\hline a7 \\ \hline \end{array}$$

$$REP(T_1 \cup T_2) = REP(T_1) \cup REP(T_2) =< U, v >$$

$$U = \left\{ \begin{array}{|c|}\hline a1 \\ a3 \\ a5 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a3 \\ a6 \\ \hline \end{array} \right\} \quad v = \begin{array}{|c|}\hline a4 \\ a7 \\ a8 \\ \hline \end{array}$$

Figure 3.13: Union

The property: $r_1 \subseteq r_2$ implies $r - r_2 \subseteq r - r_1$ and the definition of $REDUCEREP$ allows us to simplify the above definition into the following equivalent definition:

Definition 3.2.14 Let $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$. Then,
$< U_1, v_1 > - < U_2, v_2 > = REDUCEREP(< U_1, v_1 > -^0 < U_2, v_2 >)$, where
$< U_1, v_1 > -^0 < U_2, v_2 > = < U, v >$,

$$U = \{r | (\exists r_1)(r_1 \in U_1 \wedge r = r_1 - (\bigcup_{r_2 \in U_2} (r_2) \cup v_2))\}, \text{ and}$$

$$v = (\bigcup_{r \in U_1} (r) \cup v_1) - \bigcap_{r \in U_2} (r).$$

The next theorem shows that the difference on Σ_R commutes with $REDUCEREP$:

Theorem 3.2.13 *For any* $< U_1, v_1 > \in \Sigma_R$ *and* $< U_2, v_2 > \in \Sigma_R$,

$$< U_1, v_1 > - < U_2, v_2 > =$$
$$REDUCEREP(< U_1, v_1 >) - REDUCEREP(< U_2, v_2 >).$$

Next, we define difference of I-tables. Consider two domain-compatible I-tables, T_1 and T_2 and let $T = T_1 - T_2$.

Case 1: $t \in T_D^1$: If t is not in T_D^2 and not in any tuple set of T_I^2 and is not in T_M^2, then include t in T_D. Otherwise, include t in T_M only if $t \notin T_D^2$.

Case 2: $w \in T_I^1$: If no tuple of T_D^2 is in w and no tuple set of T_I^2 has any common elements with w and no tuple of T_M^2 is in w, then include w in T_I. Otherwise, include all the tuples in $w - T_D^2$ in T_M.

Case 3: $t \in T_M^1$: If t does not belong to T_D^2, then include t in T_M.

Finally, remove any redundancies introduced with the $REDUCE$ operator. Formally, difference is defined as a mapping, $- : \Gamma_R, \Gamma_R \rightarrow \Gamma_R$.

Definition 3.2.15 Let T_1 and T_2 be two domain-compatible I-tables. Then, $T_1 - T_2 = REDUCE(T)$, where

$$
\begin{aligned}
T_D = \{t \mid\ & (t \in T_D^1) \wedge (t \notin T_D^2) \wedge \neg(\exists w)(w \in T_I^2 \wedge t \in w) \wedge (t \notin T_M^2)\},\\
T_I = \{w \mid\ & (w \in T_I^1) \wedge\\
& \neg(\exists t)(t \in T_D^2 \wedge t \in w) \wedge\\
& \neg(\exists w_1)(w_1 \in T_I^2 \wedge w \cap w_1 \neq \emptyset) \wedge\\
& \neg(\exists t)(t \in T_M^2 \wedge t \in w)\},\ \textit{and}\\
T_M = \{t \mid\ & ((t \in T_M^1) \vee\\
& (\exists w)(t \in T_D^1 \wedge w \in T_I^2 \wedge t \in w) \vee\\
& (t \in T_D^1 \wedge t \in T_M^2) \vee\\
& (\exists w)(\exists t_1)(w \in T_I^1 \wedge t_1 \in T_D^2 \wedge t_1 \in w \wedge t \in w) \vee\\
& (\exists w_1)(\exists w_2)(w_1 \in T_I^1 \wedge w_2 \in T_I^2 \wedge\\
& w_1 \cap w_2 \neq \emptyset \wedge t \in w_1) \vee\\
& (\exists w)(\exists t_1)(w \in T_I^1 \wedge t_1 \in T_M^2 \wedge t_1 \in w \wedge t \in w)) \wedge\\
& (t \notin T_D^2)\}.
\end{aligned}
$$

NOTE $T - T \neq < \emptyset, \emptyset, \emptyset >$, for any I-table T. A simple example to illustrate this is an I-table T with $T_D = \emptyset$, $T_I = \emptyset$, and $T_M = \{a\}$.

The following theorem shows that the difference of I-tables commutes with $REDUCE$:

Theorem 3.2.14 *For any two domain-compatible I-tables T_1 and T_2,*

$$T_1 - T_2 = REDUCE(T_1) - REDUCE(T_2).$$

The correctness of the difference operator is established in the following theorem:

Theorem 3.2.15 *For any two domain-compatible reduced I-tables* T_1 *and* T_2,

$$REP(T_1 - T_2) = REP(T_1) - REP(T_2).$$

Corollary 3.2.5 *For any two domain-compatible I-tables* T_1 *and* T_2,

$$REP(T_1 - T_2) = REP(T_1) - REP(T_2).$$

Theorem 3.2.15 is illustrated in Figure 3.14.

3.2.7 Intersection

First, we define the intersection on Σ_R, as a mapping $\cap : \Sigma_R, \Sigma_R \rightarrow \Sigma_R$.

Definition 3.2.16 Let $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$. Then,

$< U_1, v_1 > \cap < U_2, v_2 > = REDUCEREP(< U_1, v_1 > \cap^0 < U_2, v_2 >)$, where

$< U_1, v_1 > \cap^0 < U_2, v_2 > = < U, v >$,

$U = \{r | (\exists r_1)(\exists r_2)(r_1 \in POSS(U_1) \wedge r_2 \in POSS(U_2) \wedge r = r_1 \cap r_2)\}$, *and*

$v = (\bigcup_{r \in U_1} (r) \cup v_1) \cap (\bigcup_{r \in U_2} (r) \cup v_2)$.

The property: $r_1 \subseteq r_2$ implies $r \cap r_1 \subseteq r \cap r_2$ and the definition of $REDUCEREP$ allows us to simplify the above definition into the following equivalent definition:

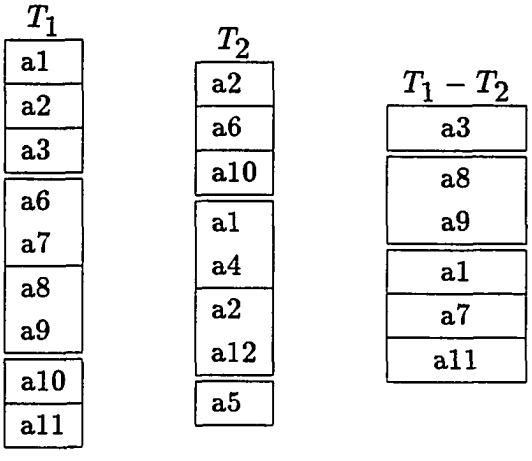
Definition 3.2.17 Let $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$. Then,

$< U_1, v_1 > \cap < U_2, v_2 > = REDUCEREP(< U_1, v_1 > \cap^0 < U_2, v_2 >)$, where

$< U_1, v_1 > \cap^0 < U_2, v_2 > = < U, v >$,

$U = \{r | (\exists r_1)(\exists r_2)(r_1 \in U_1 \wedge r_2 \in U_2 \wedge r = r_1 \cap r_2)\}$, *and*

$v = (\bigcup_{r \in U_1} (r) \cup v_1) \cap (\bigcup_{r \in U_2} (r) \cup v_2)$.



$$REP(T_1) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|}\hline a1 \\ a2 \\ a3 \\ a6 \\ a8 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a2 \\ a3 \\ a6 \\ a9 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a2 \\ a3 \\ a7 \\ a8 \\ \hline \end{array} , \begin{array}{|c|}\hline a1 \\ a2 \\ a3 \\ a7 \\ a9 \\ \hline \end{array} \right\} \quad v_1 = \begin{array}{|c|}\hline a10 \\ a11 \\ \hline \end{array}$$

$$REP(T_2) =< U_2, v_2 >$$

$$U_2 = \left\{ \begin{array}{|c|}\hline a1 \\ a2 \\ a6 \\ a10 \\ \hline \end{array} , \begin{array}{|c|}\hline a2 \\ a4 \\ a6 \\ a10 \\ \hline \end{array} \right\} \quad v_2 = \begin{array}{|c|}\hline a5 \\ a12 \\ \hline \end{array}$$

$$REP(T_1 - T_2) = REP(T_1) - REP(T_2) =< U, v >$$

$$U = \left\{ \begin{array}{|c|}\hline a3 \\ a8 \\ \hline \end{array} , \begin{array}{|c|}\hline a3 \\ a9 \\ \hline \end{array} \right\} \quad v = \begin{array}{|c|}\hline a1 \\ a7 \\ a11 \\ \hline \end{array}$$

Figure 3.14: Difference

The next theorem shows that the intersection on Σ_R commutes with *REDUCEREP*:

Theorem 3.2.16 *For any* $< U_1, v_1 > \in \Sigma_R$ *and* $< U_2, v_2 > \in \Sigma_R$,

$$< U_1, v_1 > \cap < U_2, v_2 > =$$
$$REDUCEREP(< U_1, v_1 >) \cap REDUCEREP(< U_2, v_2 >).$$

Next, we define intersection of I-tables. Consider two domain-compatible I-tables T_1 and T_2 and let $T = T_1 \cap T_2$. Tuples common to the T_D^1 and T_D^2 constitute the tuples of T_D. Tuple sets that belong to T_I^1, or T_I^2, and which are subsets of T_D^2, or T_D^1 constitute the tuple sets of T_I. Tuples which are common to the following pairs of sets constitute T_M:

1. T_D^1 and T_M^2,
2. a tuple set of T_I^1 and T_M^2,
3. T_M^1 and T_M^2,
4. T_M^1 and T_D^2,
5. T_M^1 and a tuple set of T_I^2,
6. T_D^1 and a tuple set of T_I^2,
7. a tuple set of T_I^1 and T_D^2, and
8. a tuple set of T_I^1 and a tuple set of T_I^2.

Formally, the intersection of I-tables is defined as a mapping $\cap : \Gamma_R, \Gamma_R \rightarrow \Gamma_R$.

Definition 3.2.18 Let T_1 and T_2 be two domain-compatible I-tables. Then,

$T_1 \cap T_2 = REDUCE(T)$, where

$$
\begin{aligned}
T_D = \{t \mid\ & t \in T_D^1 \wedge t \in T_D^2\},\\
T_I = \{w \mid\ & (w \in T_I^1 \wedge w \subseteq T_D^2) \vee (w \in T_I^2 \wedge w \subseteq T_D^1)\},\ \textit{and}\\
T_M = \{t \mid\ & (t \in T_D^1 \wedge (\exists w)(w \in T_I^2 \wedge t \in w) \vee\\
& (t \in T_D^1 \wedge t \in T_M^2) \vee\\
& (\exists w)(w \in T_I^1 \wedge t \in T_D^2 \wedge t \in w) \vee\\
& (\exists w_1)(\exists w_2)(w_1 \in T_I^1 \wedge w_2 \in T_I^2 \wedge t \in w_1 \wedge t \in w_2) \vee\\
& (\exists w)(w \in T_I^1 \wedge t \in T_M^2 \wedge t \in w) \vee\\
& (t \in T_M^1 \wedge t \in T_D^2) \vee\\
& (t \in T_M^1 \wedge (\exists w)(w \in T_I^2 \wedge t \in w) \vee\\
& (t \in T_M^1 \wedge T_M^2)\}.
\end{aligned}
$$

Remark: As Figure 3.15 shows, the definitions of difference and intersection are not consistent with the following relationship that holds between the corresponding regular algebraic operators:

$$T_1 \cap T_2 = T_1 - (T_1 - T_2).$$

The next theorem shows that the intersection of I-tables commutes with $REDUCE$:

Theorem 3.2.17 *For any two domain-compatible I-tables T_1 and T_2,*

$$T_1 \cap T_2 = REDUCE(T_1) \cap REDUCE(T_2).$$

The correctness of the intersection operator is established in the following theorem:

Theorem 3.2.18 *For any two domain-compatible reduced I-tables,*

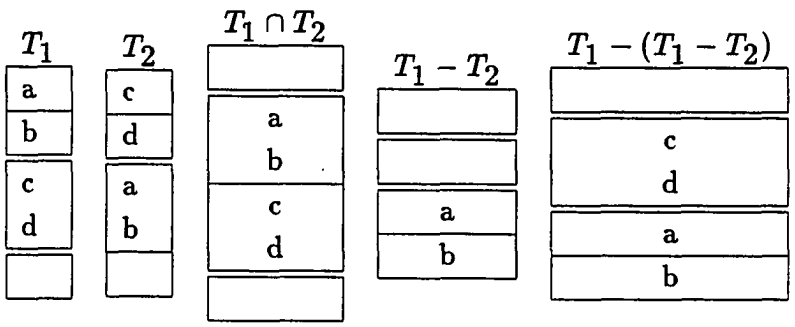


Figure 3.15: $T_1 \cap T_2 \neq T_1 - (T_1 - T_2)$

$$REP(T_1 \cap T_2) = REP(T_1) \cap REP(T_2).$$

Corollary 3.2.6 *For any two domain-compatible I-tables* T_1 *and* T_2,

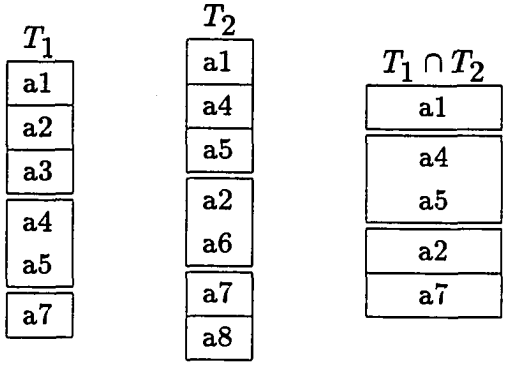
$$REP(T_1 \cap T_2) = REP(T_1) \cap REP(T_2).$$

Theorem 3.2.18 is illustrated in Figure 3.16.

NOTE: An I-table reduces to a relation if its indefinite and maybe components are empty. All the extended relational algebraic operators also reduce to the corresponding regular relational algebraic operators when their operands contain empty indefinite and maybe components. So, the extended relational model and algebra preserve all the features of the conventional relational model and algebra and are merely extensions.

3.3 Queries

Queries can be expressed in terms of the various extended relational algebraic operators defined in Section 3. The I-table accurately models the two bounds on the external interpretation, the interpretation under which the query is referred to



$$REP(T_1) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|}\hline a1 \\ \hline a2 \\ \hline a3 \\ \hline a4 \\ \hline \end{array} \; , \; \begin{array}{|c|}\hline a1 \\ \hline a2 \\ \hline a3 \\ \hline a5 \\ \hline \end{array} \right\} \quad v_1 = \boxed{a7}$$

$$REP(T_1) =< U_2, v_2 >$$

$$U_2 = \left\{ \begin{array}{|c|}\hline a1 \\ \hline a2 \\ \hline a4 \\ \hline a5 \\ \hline \end{array} \; , \; \begin{array}{|c|}\hline a1 \\ \hline a4 \\ \hline a5 \\ \hline a6 \\ \hline \end{array} \right\} \quad v_2 = \begin{array}{|c|}\hline a7 \\ \hline a8 \\ \hline \end{array}$$

$$REP(T_1 \cap T_2) = REP(T_1) \cap REP(T_2) =< U, v >$$

$$U = \left\{ \begin{array}{|c|}\hline a1 \\ \hline a4 \\ \hline \end{array} \; , \; \begin{array}{|c|}\hline a1 \\ \hline a5 \\ \hline \end{array} \right\} \quad v = \begin{array}{|c|}\hline a2 \\ \hline a7 \\ \hline \end{array}$$

Figure 3.16: Intersection

the real world modeled in an incomplete way by the system, of a query [29]. The definite and the indefinite components of an I-table correspond to one of the bounds which is the set of objects for which we can positively say that they belong to the external interpretation of query. The maybe component of an I-table corresponds to the other bound which is the set of objects for which we cannot rule out the possibility of belonging to the external interpretation of the query. We shall use the usual suppliers-parts database for the following two examples.

Example 3.3.1 Consider the I-tables SP and P in Figure 3.17 and the query: Find all the supplier numbers of suppliers who supply "red" parts. The query in the extended relational algebra is

$$\Pi_1(\sigma_{2=3}(SP \times \Pi_1(\sigma_{2="red"}(P)))).$$

Evaluating this expression against the I-database we obtain the answer in Figure 3.17. The answer is interpreted in the following manner: s2 and s4 supply "red" parts and s3 and s5 may supply "red" parts. *Example 3.3.2* Consider another instance of the

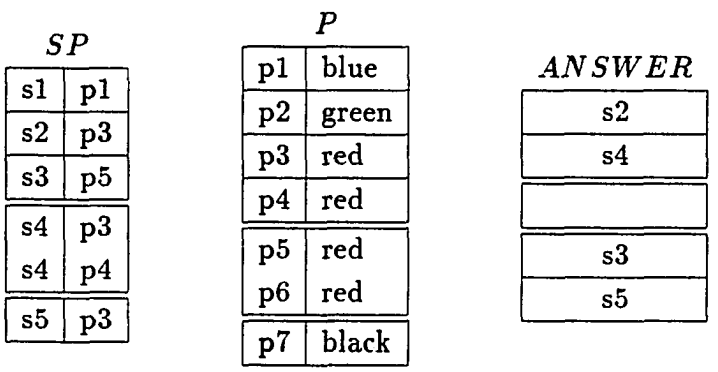


Figure 3.17: I-tables SP, P, $ANSWER$

supplier-parts database in Figure 3.18 and the query: Find all the supplier numbers of

suppliers who do not supply part "p2". The query in the extended relational algebra is

$$\Pi_1(S) - \Pi_1(\sigma_{2="p2"}(SP)).$$

Evaluating this expression against the I-database, we obtain the answer in Figure 3.18. The answer is interpreted in the following manner: s3 does not supply part "p2" and there is a possibility that s7, s8, s9, s10 all do not supply part "p2". As

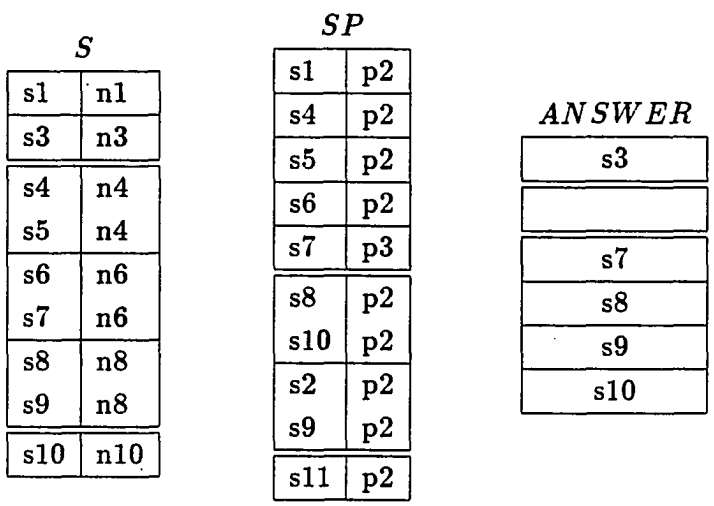


Figure 3.18: I-tables S, SP, $ANSWER$

the above two examples illustrate, queries are posed in the same way as for conventional relational databases. Since we have established the correctness of the extended relational algebraic operators, all possible answers are extracted.

3.4 Non-query Operations

In this section, we present non-query operations on I-tables. We define the insert, delete, and modify operations. These operations allow the user to insert tuples into,

delete tuples from, and modify tuples of I-tables.

Definition 3.4.1 The insert operator is specified as: $ins(C, t, T)$, where $C \in \{D, I, M\}$, t is a tuple if $C \in \{D, M\}$ and is a tuple set if $C = I$, and T is an I-table. The effect of the *ins* operation is to update the I-table T into T' as follows:

$T' = REDUCE(< T_D^1, T_I^1, T_M^1 >)$, where

$T_C^1 = T_C \cup \{t\}$ and $T_X^1 = T_X$ for $X \in \{D, I, M\} - \{C\}$.

Definition 3.4.2 The delete operation is specified as: $del(C, t, T)$, where $C \in \{D, I, M\}$, t is a tuple if $C \in \{D, M\}$ and is a tuple set if $C = I$, and T is an I-table. The effect of the *del* operation is to update the I-table T into T' as follows:

$T' =< T_D^1, T_I^1, T_M^1 >$, where

$T_C^1 = T_C - \{t\}$ and $T_X^1 = T_X$ for $X \in \{D, I, M\} - \{C\}$.

Definition 3.4.3 The modify operation is specified as: $mod(C, t, t', T)$, where $C \in \{D, I, M\}$, t and t' are tuples if $C \in \{D, M\}$ and are tuple sets if $C = I$, and T is an I-table. The effect of the modify operator is to update the I-table T into T' as follows:

$T' = REDUCE(< T_D^1, T_I^1, T_M^1 >)$, where

$T_C^1 = (T_C - \{t\}) \cup \{t'\}$ and $T_X^1 = T_X$ for $X \in \{D, I, M\} - \{C\}$.

The *mod* operation is simply a *del* followed by an *ins*.

4 INDEFINITE DEDUCTIVE DATABASES

In this chapter, we show how the extended relational algebra can be used to implement indefinite deductive databases. First, we present an additional algebraic operator, called *project-union*, which will be used to evaluate non-Horn rules. The project-union operator is actually an extension to the projection operator which could only be used to evaluate Horn rules. Then, we define *I-rules*, which are generalizations of non-Horn rules and describe a method to obtain extended relational algebraic expressions for I-rules. Non-recursive and recursive I-rules are discussed and procedures to evaluate them are described. Finally, we show how to evaluate queries using the extended relational algebra.

4.1 Project-Union

First, we establish the need for an additional operator to evaluate non-Horn rules. Consider the Horn rule:

$$P(x, y) \leftarrow Q(z, x, y).$$

The algebraic expression to evaluate the relation corresponding to the predicate symbol P is:

$$\Pi_{2,3}(Q)$$

where Q is the relation that corresponds to the predicate symbol with the same name. Now consider the non-Horn rule:

$$P(x,y), P(x,z) \leftarrow Q(x,y,z).$$

We cannot use the projection operator to evaluate this rule. To solve this problem, we need to define an extension to the projection operator that can compute the I-table corresponding to the predicate symbol P. The input to such an operator is the I-table Q and two lists of projection attributes, one for each positive literals in the non-Horn rule. The I-table P for the predicate symbol P can be evaluated by applying the extended projection operator, which we shall refer to as *project-union* and shall represent by the symbol $\amalg$, as follows:

$$\amalg_{<<1,2>,<1,3>>}(Q).$$

We need the following definitions:

Definition 4.1.1 A *projection attribute list* is defined to be a list of attribute numbers or constant symbols. For example $< 1, "Math", 3 >$ is a projection attribute list, where 1 and 3 are attribute numbers and $Math$ is a constant symbol.

Definition 4.1.2 Let $A_1, \ldots, A_n$ be n projection attribute lists, where

$$A_i =< a_{i1}, \ldots, a_{im_i} >, 1 \leq i \leq n.$$

Then, $A_1, \ldots, A_n$ are *domain-compatible* if and only if

1. $m_1 = \cdots = m_n = m$, and
2. for each i, $1 \leq i \leq n$, the domains associated with the attributes a_{ij}, $1 \leq j \leq m$, are all the same.

Now we define project-union on Σ_R.

Definition 4.1.3 Let $< U, v > \in \Sigma_R$ and $A_1, \ldots, A_n$ be n domain-compatible projection attribute lists. Then,

$\Pi_{<A_1,\ldots,A_n>}(< U, v >) = REDUCEREP(\Pi^0_{<A_1,\ldots,A_n>}(< U, v >))$, where

$\Pi^0_{<A_1,\ldots,A_n>}(< U, v >) =< U_1, v_1 >$,

$U_1 = \bigcup_{r \in U} (\Pi_{<A_1,\ldots,A_n>}(r))$,

$\Pi_{<A_1,\ldots,A_n>}(r) = \{\{t'_1, \ldots, t'_m\} | (\forall i)(1 \leq i \leq m \rightarrow t'_i \in \{\Pi_{A_1}(t_i), \ldots, \Pi_{A_n}(t_i)\})\}$,

$r = \{t_1, \ldots, t_m\}$,

$v_1 = \{t | (\exists t_1)(t_1 \in v \wedge t \in \{\Pi_{A_1}(t_1), \ldots, \Pi_{A_n}(t_1)\})\}$,

$\Pi_{<a_1,\ldots,a_k>}(t) =< t_{a_1}, \ldots, t_{a_k} >$, and

$$t_{a_i} = \begin{cases} t[a_i] & \text{, if } a_i \text{ is an attribute number} \\ a_i & \text{, if } a_i \text{ is a constant symbol} \end{cases}, \ 1 \leq i \leq k.$$

The next theorem shows that project-union commutes with $REDUCEREP$:

Theorem 4.1.1 *For any $< U, v > \in \Sigma_R$ and domain-compatible projection attribute lists $A_1, \ldots, A_n$,*

$$\Pi_{<A_1,\ldots,A_n>}(< U, v >) = \Pi_{<A_1,\ldots,A_n>}(REDUCEREP(< U, v >)).$$

We now define project-union on I-tables.

Definition 4.1.4 Let T_1 be an I-table and $A_1, \ldots, A_n$ be n domain-compatible projection attribute lists. Then,

$\Pi_{<A_1,\ldots,A_n>}(T_1) = REDUCE(T)$, where

$$T_D = \{t \mid (\exists t_1)(t_1 \in T^1_D \wedge \{t\} = \{\Pi_{A_1}(t_1), \ldots, \Pi_{A_n}(t_1)\}) \vee (\exists w)(w \in T^1_I \wedge \{t\} = \bigcup_{i=1}^{n} \Pi_{A_i}(w))\},$$

$$T_I = \{w \mid (\exists t)(t \in T_D^1 \wedge w = \{\Pi_{A_1}(t), \ldots, \Pi_{A_n}(t)\} \wedge |w| > 1) \vee$$
$$(\exists w_1)(w_1 \in T_I^1 \wedge w = \bigcup_{i=1}^{n} \Pi_{A_i}(w_1) \wedge |w| > 1)\},$$
$$T_M = \{t \mid (\exists t_1)(t_1 \in T_M^1 \wedge t \in \{\Pi_{A_1}(t_1), \ldots, \Pi_{A_n}(t_1)\})\},$$

$\Pi_A(w) = \{\Pi_A(t) | (\exists t)(t \in w)\}$,

$\Pi_{<a_1,\ldots,a_k>}(t) =< t_{a_1}, \ldots, t_{a_k} >$, and

$$t_{a_i} = \begin{cases} t[a_i] & \text{, if } a_i \text{ is an attribute number} \\ a_i & \text{, if } a_i \text{ is a constant symbol} \end{cases}, 1 \leq i \leq k.$$

The following theorem shows that project-union commutes with $REDUCE$:

Theorem 4.1.2 *For any I-table T and domain-compatible projection attribute lists $A_1, \ldots, A_n$.*

$$\Pi_{<A_1,\ldots,A_n>}(T) = \Pi_{<A_1,\ldots,A_n>}(REDUCE(T)).$$

The correctness of the project-union operator is established in the following theorem:

Theorem 4.1.3 *For any reduced I-table T and domain-compatible projection attribute lists $A_1, \ldots, A_n$,*

$$\Pi_{<A_1,\ldots,A_n>}(REP(T)) = REP(\Pi_{<A_1,\ldots,A_n>}(T)).$$

Corollary 4.1.1 *For any I-table T and domain-compatible projection attribute lists $A_1, \ldots, A_n$,*

$$\Pi_{<A_1,\ldots,A_n>}(REP(T)) = REP(\Pi_{<A_1,\ldots,A_n>}(T)).$$

Theorem 4.1.3 is illustrated in Figure 4.1.

NOTE The project-union operator is an extension of the extended projection operator and it reduces to the extended projection operator when

T

a1	c2	c2
a1	b1	c1
a2	b2	c2
a2	b2	c3
a3	b3	c3

$\Pi_{<<1,2>,<1,3>>}(T)$

a1	c2
a1	b1
a1	c1
a2	b2
a2	c2
a2	c3
a3	b3
a3	c3

$$REP(T) =< U, v >$$

$$U = \left\{ \begin{array}{|c|c|c|}\hline a1 & c2 & c2 \\ \hline a1 & b1 & c1 \\ \hline a2 & b2 & c2 \\ \hline \end{array} \; , \; \begin{array}{|c|c|c|}\hline a1 & c2 & c2 \\ \hline a1 & b1 & c1 \\ \hline a2 & b2 & c3 \\ \hline \end{array} \right\}$$

$$v = \begin{array}{|c|c|c|}\hline a3 & b3 & c3 \\ \hline \end{array}$$

$$REP(\Pi_{<<1,2>,<1,3>>}(T)) = \Pi_{<<1,2>,<1,3>>}(REP(T)) =< U_1, v_1 >$$

$$U_1 = \left\{ \begin{array}{|c|c|}\hline a1 & c2 \\ \hline a1 & b1 \\ \hline a2 & b2 \\ \hline \end{array} \; , \; \begin{array}{|c|c|}\hline a1 & c2 \\ \hline a1 & b1 \\ \hline a2 & c2 \\ \hline \end{array} \; , \; \begin{array}{|c|c|}\hline a1 & c2 \\ \hline a1 & c1 \\ \hline a2 & b2 \\ \hline \end{array} \; , \; \begin{array}{|c|c|}\hline a1 & c2 \\ \hline a1 & c1 \\ \hline a2 & c2 \\ \hline \end{array} \; , \; \begin{array}{|c|c|}\hline a1 & c2 \\ \hline a1 & b1 \\ \hline a2 & c3 \\ \hline \end{array} \; , \; \begin{array}{|c|c|}\hline a1 & c2 \\ \hline a1 & c1 \\ \hline a2 & c3 \\ \hline \end{array} \right\}$$

$$v_1 = \begin{array}{|c|c|}\hline a3 & b3 \\ \hline a3 & c3 \\ \hline \end{array}$$

Figure 4.1: Project-Union

1. $n = 1$, and

2. the projection attribute list consists of only attribute numbers.

4.2 I-rules

Here, we introduce I-rules which are generalizations of non-Horn clauses. The need for I-rules is discussed now. Consider the two Horn clauses:

1. $DEPT(x, "Math") \leftarrow TEACHES(x, "231")$, and

2. $DEPT(x, "Math") \leftarrow TEACHES(x, "331")$.

and let

$$TEACHES("John", "231") \vee TEACHES("John", "331")$$

be true in the database. This disjunction actually corresponds to a tuple set in the I-table corresponding to the predicate symbol $TEACHES$. It can easily be observed that $DEPT("John", "Math")$ is a consequence of the database. However, if we consider the algebraic expression to evaluate $DEPT$:

$$\Pi_1(\sigma_{2="231"}(TEACHES)) \cup \Pi_1(\sigma_{2="331"}(TEACHES)),$$

we would obtain $DEPT("John", "Math")$ as a maybe tuple. To avoid such problems, we combine the two Horn clauses into the rule:

$$DEPT(x, "Math") \leftarrow < TEACHES(x, "231"), TEACHES(x, "331") >$$

which is equivalent to the logical formula:

$$DEPT(x, "Math") \vee \neg(TEACHES(x, "231") \vee TEACHES(x, "331")).$$

Such a rule will be referred to as *I-rules*. We now formally define I-rules and certain restrictions on them.

A *conjunct* is a disjunction of positive literals involving the same predicate symbol:

$$P(X_1) \vee \cdots \vee P(X_n),$$

called a *positive conjunct*, or its negation:

$$\neg(P(X_1) \vee \cdots \vee P(X_n)),$$

called a *negative conjunct*. A *ground conjunct* is a conjunct with no variable symbols. We shall surround the literals in a conjunct with angular brackets and separate the literals with commas to be consistent with the syntax of non-Horn clauses. For example $< P(x,y), P(x,z) >$ is a positive conjunct and $\neg < P(x,y), P(x,z) >$ is a negative conjunct. We shall omit the angular brackets if there is only one literal inside it.

An *I-rule* is a disjunction of conjuncts with at most one positive conjunct. A *ground I-rule* is an I-rule with no variable symbols. We shall omit the angular brackets around the positive conjunct of an I-rule. Two examples of I-rules are:

1. $P(x,y), P(x,z) \leftarrow < Q(x,u), Q(x,v) >, R(u,y,v,z)$, and
2. $A(y) \leftarrow S(x,y), < SP(x,"p1"), SP(x,"p2") >$.

The I-rule

$$P_1,\ldots,P_k \leftarrow < Q_{11},\ldots,Q_{1n_1} >,\ldots,< Q_{l1},\ldots,Q_{ln_l} >$$

can be viewed as representing the following collection of non-Horn clauses:

$$\{P_1,\ldots,P_k \leftarrow Q_1,\ldots,Q_l | (\forall i)(1 \leq i \leq l \rightarrow Q_i \in \{Q_{i1},\ldots,Q_{in_i}\})\}$$

We shall impose the following two restrictions on I-rules:

1. *Range-restriction*: The I-rule is said to be *range-restricted*, if each of the non-Horn rules it represents is range-restricted. A non-Horn rule is said to be range-restricted if all the variable symbols appearing in the positive literals, P_is, also appear among the negative literals, Q_is. The I-rule:

$$P(x,y), P(x,w) \leftarrow Q(x,z), < R(z,y), R(w,y) >, S(z,w)$$

is range-restricted because the variables x, y, and w appear on the right hand side of both the non-Horn rules represented by the I-rule, and the I-rule:

$$P(x,y), P(x,z) \leftarrow Q(x,w), < R(y,z), R(x,z) >, S(z,w)$$

is not range restricted because the variable y does not appear in the following non-Horn rule represented by the I-rule:

$$P(x,y), P(x,z) \leftarrow Q(x,w), R(x,z), S(z,w).$$

We shall restrict all the I-rules to be range-restricted.

2. *Projection-consistency*: An I-rule is *projection-consistent* if for each positive literal P_i, $1 \leq i \leq k$, the variable symbols of P_i occur in the same "positions" on the right hand side of the $\leftarrow$ symbol of all non-Horn rules represented by the I-rule. The I-rule:

$$P(x,y), P(x,w) \leftarrow Q(x,z), < R(z,y), R(w,y) >, S(z,w)$$

is projection-consistent because the variables x and y appear in positions 1 and 4 respectively in both the non-Horn rules represented by the I-rule and the variables x and w appear in positions 1 and 6 respectively in both the non-Horn rules represented by the I-rule and the I-rule:

$$P(x,y), P(x,w) \leftarrow Q(x,z), < R(z,y), R(y,w) >, S(z,w)$$

is not projection consistent because the variables x and y appear in positions 1 and 4 respectively in one of the non-Horn rules represented by the I-rule and in positions 1 and 3 in the other non-Horn rule represented by the I-rule. We shall restrict all the I-rules to be projection-consistent.

A *query* is an I-rule with exactly one positive literal. An example of a query is:

$$ANSWER(x) \leftarrow S(x,y), < SP(x,"p1"), SP(x,"p2") >.$$

Queries are also subjected to the range-restriction and projection-consistency restrictions.

4.3 Algebraic Expressions for I-rules

In this section, we present a method to obtain extended relational algebraic expressions for I-rules. Consider the I-rule:

$$P_1, \ldots, P_k \leftarrow < Q_{11}, \ldots, Q_{1n_1} >, \ldots, < Q_{l1}, \ldots, Q_{ln_l} >.$$

Let $L_1, \ldots, L_m$ be the non-Horn clauses represented by the I-rule and let L_i be

$$P_1, \ldots, P_k \leftarrow Q_1(t_1^1, \ldots, t_{m_1}^1), \ldots, Q_l(t_1^l, \ldots, t_{m_l}^l),$$

where t_u^vs are either constant symbols or variable symbols.

We first obtain a selection condition, C_i, for each of the non-Horn clauses L_i, $1 \leq i \leq m$. C_i is obtained as follows:

Step 1 For all the t_u^vs that are constant symbols obtain the following condition:

$$(\sum_{a=1}^{v-1} m_a) + u = t_u^v.$$

Step 2 For all the $t_{u_1}^{v_1}$s and $t_{u_2}^{v_2}$s that are variable symbols such that $u_1 \neq u_2$, $v_1 \neq v_2$, and $t_{u_1}^{v_1} \neq t_{u_2}^{v_2}$ obtain the following condition:

$$(\sum_{a=1}^{v_1-1} m_a) + u_1 = (\sum_{a=1}^{v_2-1} m_a) + u_2.$$

Step 3 C_i is the conjunction of all the conditions obtained in Step 1 and Step 2.

Let $C_1, \ldots, C_m$ be the selection conditions obtained.

Next, we obtain a projection attribute list for each of the positive literals $P(t_1, \ldots, t_{n_i})$. Let t_u^v be equal to t_j, where t_j is a variable symbol. Then a_j, the position of t_u^v, is defined as follows:

$$a_j = (\sum_{p=1}^{v-1} m_p) + u.$$

The projection attribute list for $P(t_1, \ldots, t_{n_i})$ is $< b_1, \ldots, b_{n_i} >$, where

$$b_e = \begin{cases} a_e, & t_e \text{ is a variable symbol} \\ t_e, & t_e \text{ is a constant symbol,} \end{cases} \qquad 1 \leq e \leq n_i.$$

Let $A_1, \ldots, A_k$ be the projection attribute lists for the positive literals $P_1, \ldots, P_k$ respectively.

Note: Range-restricted I-rules ensure the existence of t_u^v on the right hand side of the $\leftarrow$ symbol of the non-Horn clause and projection-consistent I-rules ensure a unique projection attribute list for each positive literal.

Then, the algebraic expression for the I-rule is:

$$\Pi_{<A_1,\ldots,A_k>}(\sigma_{C_1 \vee \cdots \vee C_m}(Q_1 \times \cdots \times Q_l)),$$

where $Q_1, \ldots, Q_l$ are the I-tables corresponding to the predicate symbols with the same names.

Example 4.3.1 Consider the I-rule:

$$P(x,y), P(x,w) \leftarrow Q(x,z), < R(z,y), R(w,y) >, S(z,w).$$

The extended algebraic expression for the I-rule is:

$$P = \Pi_{<<1,4>,<1,6>>}(\sigma_C(Q \times R \times S)),$$

where $C = ((2 = 3) \wedge (3 = 5)) \vee ((2 = 5) \wedge (3 = 6))$.

Example 4.3.2 Consider the I-rule:

$$Answer(y) \leftarrow S(x,y), < SP(x, "p1"), SP(x, "p2") >,$$

which is actually a query. The extended algebraic expression for the query is:

$$ANSWER = \Pi_{<<2>>}(\sigma_{((1=3)\wedge(4="p1"))\vee((1=3)\wedge(4="p2"))}(S \times SP)).$$

4.4 Non-Recursive Indefinite Deductive Databases

The I-table defined by a non-recursive I-rule is computed by the extended relational algebraic expression corresponding to the I-rule.

Example 4.4.1 Consider the non-recursive I-rule:

$$DEPT(x, Math), Dept(x, CS) \leftarrow < Teaches(x, 231), Teaches(x, 331) >,$$

which states that if x teaches the courses numbered 231 or 331, then x belongs to the Math or the CS department. The extended relational algebraic expression for this I-rule is:

$$\Pi_{<<1,"Math">,<1,"CS">>}(\sigma_{(2="231")\vee(2="331")}(T))$$

Evaluating this expression on the I-table $TEACHES$ of Figure 4.2, we obtain the I-table $DEPT$ in Figure 4.3.

$TEACHES$

John	311
Tom	231
Gary	331
David	231
Kevin	231
Craig	231
Craig	331
Joe	231

Figure 4.2: I-table $TEACHES$

4.5 Recursive Indefinite Deductive Databases

Recursion is handled by repeated application of the extended relational algebraic expression associated with a recursive I-rule until no new tuples or tuple sets are generated. This process is guaranteed to terminate as all the databases under consideration are finite. Consider the I-rule:

DEPT

Tom	Math
Tom	CS
Gary	Math
Gary	CS
Craig	Math
Craig	CS
David	Math
David	CS
Kevin	Math
Kevin	CS
Joe	Math
Joe	CS

Figure 4.3: I-table *DEPT*

$$P_1,\ldots,P_k \leftarrow < Q_{11},\ldots,Q_{1n_1} >,\ldots,< Q_{l1},\ldots,Q_{ln_l} >,$$

where at least one of the conjuncts on the right hand side of the symbol $\leftarrow$ involves the predicate symbol present in the positive literals. Let P be the I-table defined by this I-rule and let $Q_1,\ldots,Q_l$ be the I-tables corresponding to the predicate symbols of the conjuncts on the right hand side of the $\leftarrow$ symbol. P is computed by the algorithm shown below:

begin

i := 0;

$P^0 := P_{INIT}$;

$P^* := P_{INIT}$;

repeat

$P^{i+1} := f(P^*, Q_1,\ldots,Q_l)$;

$P^* := P^* \cup P^{i+1}$;

i := i + 1

until (there are no changes to P^*);

$P := P^*$

end

where $f(P^*, Q_1,\ldots,Q_l)$ is the extended relational algebraic expression for the I-rule, and P_{INIT} is the initial instance of the I-table P. P_{INIT} may be present in the database or may be generated by using another I-rule, possibly non-recursive.

Example 4.5.1 Consider the recursive I-rule:

$$BG(x,y), BG(x,z) \leftarrow F(x,u), BG(u,y), M(x,v), BG(v,z),$$

where $BG(x,y)$ stands for "the blood group of x is y", $F(x,y)$ stands for "y is the father of x", and $M(x,y)$ stands for "y is the mother of x". The extended relational algebraic expression for this I-rule is:

$$\Pi_{<<1,4>,<1,8>>}(\sigma_{(1=5)\wedge(2=3)\wedge(6=7)}(F \times BG \times M \times BG)).$$

F

John	David
David	Steve
Mary	Craig
Tom	Doug

BG^0

Steve	A
Pam	B
Craig	A
Liz	O
Doug	O
Lucy	A

M

John	Mary
David	Pam
Mary	Liz
Tom	Lucy

Figure 4.4: A Database Instance

Repeatedly applying the extended relational algebraic expression to the database of Figure 4.4, we obtain I-tables BG^1 and BG^2 in Figure 4.5. BG^1 and BG^2 contain new tuples and tuple sets generated in iterations 1 and 2 respectively. Iteration 3 does not generate any new tuples or tuple sets.

Example 4.5.2 Consider the recursive I-rule:

$$PARTLOC(x,y), PARTLOC(x,z) \leftarrow$$
$$SP(u,x), S(u,y), SUBPART(x,v), PARTLOC(v,z),$$

where $SUBPART(x,y)$ stands for "x is a subpart of y", $SP(x,y)$ stands for "supplier x supplies part y", $S(x,y)$ stands for "supplier x is located in y", and $PARTLOC(x,y)$

BG^1

David	A
David	B
Mary	A
Mary	O
Tom	A
Tom	O

BG^2

John	A
John	B
John	O

Figure 4.5: I-tables BG^1 and BG^2

stands for "part x can be found in location y". The extended algebraic expression for the I-rule is:

$$\Pi_{<<2,4>,<2,8>>}(\sigma_{(1=3)\wedge(2=5)\wedge(6=7)}(SP \times S \times SUBPART \times PARTLOC)).$$

Repeatedly applying the extended algebraic expression against the database in Figure 4.6, we obtain the I-tables $PARTLOC^1$ and $PARTLOC^2$ in Figure 4.7. $PARTLOC^1$ corresponds to the tuples and tuple sets generated in the first iteration and $PARTLOC^2$ corresponds to the tuples and tuple sets generated in the second iteration. The third iteration does not produce any new tuples or tuple sets.

4.6 Example of a Query

Consider the database in Figure 4.8 and the query: Find all the supplier names of suppliers who supply either part "p1" or part "p2". The query as an I-rule is:

$$ANSWER(x) \leftarrow S(x,y), < SP(x,"p1"), SP(x,"p2") >.$$

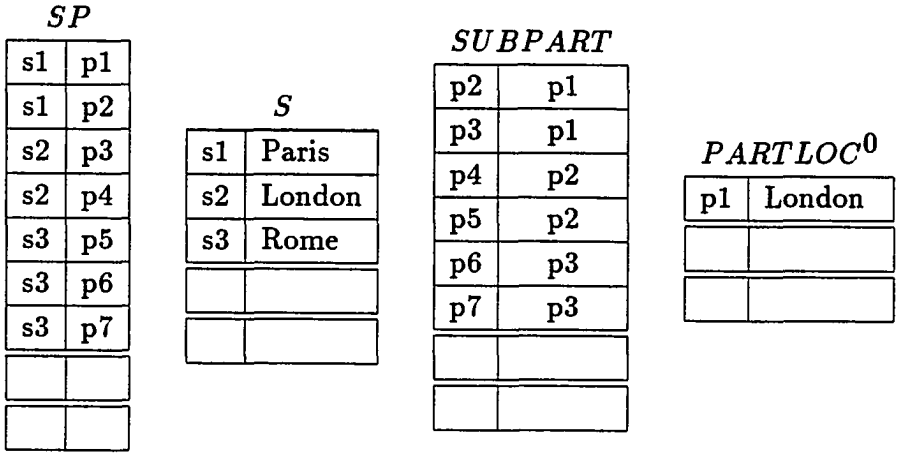


Figure 4.6: A Database

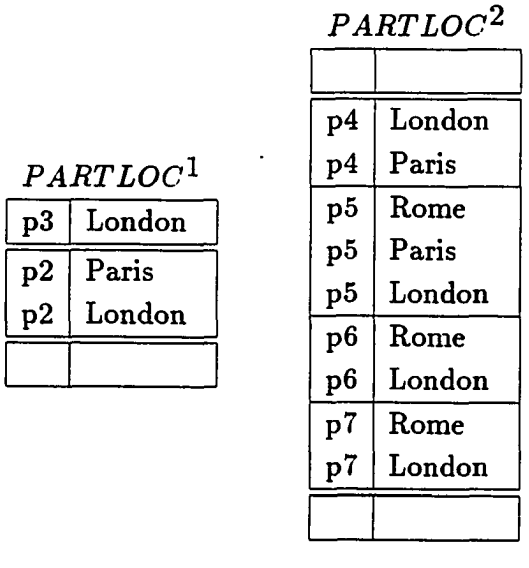


Figure 4.7: I-tables $PARTLOC^1$ and $PARTLOC^2$

The extended relational algebraic expression for the I-rule is:

$$\Pi_2(\sigma_{((1=3)\wedge(4=\text{"}p1\text{"}))\vee((1=3)\wedge(4=\text{"}p2\text{"}))}(S \times SP))$$

Evaluating this expression against the database, we obtain the I-table in Figure 4.9.

S

s1	Jones
s3	Coady
s2	Smith
s2	Blake

SP

s1	p1
s2	p2
s3	p1
s3	p2

Figure 4.8: Database

Jones
Coady
Smith
Blake

Figure 4.9: Answer to Query

The answer is interpreted as: Jones and Coady supply either of the two parts "p1" or "p2" and Smith or Blake supply either of the two parts "p1" or "p2".

4.7 Correctness of Algebraic Approach

Consider the I-rule:

$$P_1,\ldots,P_k \leftarrow < Q_{11},\ldots,Q_{1n_1} >,\ldots,< Q_{l1},\ldots,Q_{ln_l} >.$$

Let $m = n_1 \times \ldots \times n_l$, and let P be the predicate symbol present in the positive literals $P_1,\ldots,P_k$. This I-rule can be easily shown to be equivalent to the following three I-rules:

(1) $A(x_1^1,\ldots,x_{m_l}^l) \leftarrow Q_1(x_1^1,\ldots,x_{m_1}^1),\ldots,Q_l(x_1^l,\ldots,x_{m_l}^l)$

(2) $B(x_1^1,\ldots,x_{m_l}^l) \leftarrow A(x_1^1,\ldots,x_{m_l}^l), < C_1,\ldots,C_m >$

(3) $P_1,\ldots,P_k \leftarrow B(x_1^1,\ldots,x_{m_l}^l)$

where A and B are unique predicate symbols, Q_i is the predicate symbol present in the conjunct $< Q_{i1},\ldots,Q_{in_i} >$, and C_i is the conjunction of the following literals involving the equality predicate symbol:

1. $=(x_u, x_v)$, for variable symbols x_u and x_v such that $u \neq v$ and $x_u = x_v$ on the right hand side of the ith non-Horn rule represented by the I-rule.

2. $=(x_a, a)$, for each constant symbol a on the right hand side of the ith non-Horn rule represented by the I-rule such that x_a is the variable symbol in I-rule (1) in the position of the constant symbol a.

The I-table corresponding to the predicate symbol A can be computed by the cartesian product of the I-tables corresponding to the predicate symbols $Q_1,\ldots,Q_l$. Since the extended cartesian product is shown to be correct in Theorem 3.2.9, we obtain

exactly all the instances of the predicate symbol A defined in rule (1). The I-table corresponding to the predicate symbol B can be computed by the selection operator with the selection condition corresponding to the conditions $C_1, \ldots, C_m$. The input to the selection operator is the I-table corresponding to the predicate symbol A. Again, since the selection operator has been proven to be correct in Theorem 3.2.3, we obtain exactly all the instances of the predicate symbol B defined in rule (2). Finally, the I-table corresponding to the predicate symbol P is computed by the project-union operator with the projection attribute lists corresponding to the arguments of $P_1, \ldots, P_k$. The input to the project-union operator is the I-table corresponding to the predicate symbol B. Again, since the project-union operator has been proven to be correct in Theorem 4.1.3, we obtain exactly all the instances of the predicate symbol P defined in rule (3), which is actually the instances of the predicate symbol P defined in the original I-rule. Finally, since the union operator has been shown to be correct in Theorem 3.2.12, we can use the union operator to obtain exactly all the instances of the predicate symbol P defined by more than one I-rule.

Example 4.7.1 The I-rule:

$$BG(x,u), BG(x,v) \leftarrow F(x,y), BG(y,u), M(x,z), BG(z,v)$$

is equivalent to the three rules:

1. $A(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8) \leftarrow F(x_1,x_2), BG(x_3,x_4), M(x_5,x_6), BG(x_7,x_8)$
2. $B(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8) \leftarrow A(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8), =(x_1,x_5), =(x_2,x_3), =(x_6,x_7)$
3. $BG(x_1,x_4), BG(x_1,x_8) \leftarrow B(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8)$

The extended relational algebraic expressions to evaluate the I-tables corresponding to the predicate symbols A, B, and BG are:

1. $A = F \times BG \times M \times BG$,
2. $B = \sigma_{(1=5)\wedge(2=3)\wedge(6=7)}(A)$, and
3. $BG = \Pi_{<<1,4>,<1,8>>}(B)$

respectively.

Example 4.7.2 The I-rule:

$$P(x) \leftarrow S(x,y), < SP(y,"p1"), SP(y,"p2") >$$

is equivalent to the three rules:

1. $A(x_1,x_2,x_3,x_4) \leftarrow S(x_1,x_2), SP(x_3,x_4)$,
2. $B(x_1,x_2,x_3,x_4) \leftarrow A(x_1,x_2,x_3,x_4), <= (x_2,x_3) \wedge = (x_4,"p1"), = (x_2,x_3) \wedge = (x_4,"p2") >$, and
3. $P(x_1) \leftarrow B(x_1,x_2,x_3,x_4)$.

The extended relational algebraic expressions to evaluate the I-tables corresponding to the predicate symbols A, B, and P are

1. $A = S \times SP$,
2. $B = \sigma_{((2=3)\wedge(4="p1"))\vee((2=3)\wedge(4="p2"))}(A)$, and
3. $P = \Pi_{<<1>>}(B) = \Pi_1(B)$

respectively.

Example 4.7.3 The I-rule:

$$DEPT(x,"Math"), DEPT(x,"CS") \leftarrow$$
$$< TEACHES(x,"231""), TEACHES(x,"331") >$$

is equivalent to the three rules:

1. $A(x_1, x_2) \leftarrow TEACHES(x_1, x_2)$,
2. $B(x_1, x_2) \leftarrow A(x_1, x_2), <= (x_2,"231"), = (x_2,"331") >$, and
3. $DEPT(x_1,"Math"), DEPT(x_1,"CS") \leftarrow B(x_1, x_2)$.

The extended relational algebraic expressions to evaluate the I-tables corresponding to the predicate symbols A, B, and $DEPT$ are:

1. $A = TEACHES$,
2. $B = \sigma_{(2="231")\vee(2="331")}(A)$, and
3. $DEPT = \Pi_{<<1,"Math">,<1,"CS">>}(B)$

respectively.

We now justify the correctness of the algorithm to evaluate recursive I-rules. Recall the definition of *weaker* I-tables. We define a monotonic extended relational algebraic expression as follows:

Definition 4.7.1 An extended relational algebraic expression, f, is said to be *monotonic* if and only if

$$(T_1 \leq T_2) \rightarrow (f(T_1) \leq f(T_2)),$$

for any I-tables T_1 and T_2.

Consider an equation of the form:

$$T = f(T)$$

where $f(T)$ is an extended relational algebraic expression with operand T; perhaps among other operands; such that the arity of T and $f(T)$ are the same. A *least fixed point* of the equation, denoted $LFP(T = f(T))$, is an I-table T^* such that

1. $T^* = f(T^*)$, and

2. if T is any I-table such that $T = f(T)$, then $T^* \leq T$.

Tarski [43] assures that a unique least fixed point exists if f is monotonic. If f is monotonic, then by induction on i, we can show that

$$f^{i-1}(T) \leq f^i(T)$$

where f^i is f applied i times. If all the argument I-tables are finite, then since no new component values are introduced by the extended relational algebraic operators, we know that there is some finite T for which each $f^i(T)$ is a subset. Therefore, there must be some n_0 such that

$$T \leq f(T) \leq f^2(T) \leq \cdots \leq f^{n_0}(T) = f^{n_0+1}(T).$$

It is easy to check that $f^{n_0}(T)$ is the least fixed point, $LFP(T = f(T))$. We now state the following theorem, which is also true for regular relational algebraic expressions:

Theorem 4.7.1 *Any extended relational algebraic expression involving cartesian product, union, selection, and project-union is monotonic.*

5 GENERALIZED RELATIONAL MODEL

In Chapter 3, we defined I-tables to represent disjunctive information of the form $P(t_1) \vee \cdots \vee P(t_n)$, where all the disjuncts in this formula involve the same predicate symbol. In this Chapter, we define a general data structure, called M-table, which is capable of representing more general forms of disjunctive information such as $P_1(t_1) \vee \cdots \vee P_n(t_n)$, where the P_is could be different predicates. The relational algebra is suitably generalized to operate on M-tables. In addition to the generalized relational algebraic operators, we define two new operators, *R-projection* and *merge*, which are used in answering queries.

5.1 M-Tables

In this section, we introduce a data structure, called an *M-table*, which is capable of representing general kinds of disjunctive and maybe information. Then, we present the notion of redundancy in M-tables and define an operator, called *REDUCE*, to remove the redundancy.

A *relation scheme*, R, is a finite list of attribute names, $< A_1, \ldots, A_n >, n \geq 1$. R is said to have *arity* n. With each attribute is associated a domain. An *M-table scheme*, MR, is a finite list of relation schemes, $< R_1, \ldots, R_k >, k \geq 1$. MR is said to be of *order* k.

Definition 5.1.1 An *M-table*, T, over the M-table scheme, $MR =< R_1,\ldots,R_k >$, consists of the two components, $T =< T_{sure}, T_{maybe} >$, where

$$T_{sure} \subseteq \{< u_1,\ldots,u_k > \mid (\forall i)(1 \leq i \leq k \rightarrow u_i \in 2^{D_i^1 \times \cdots \times D_i^{n_i}}) \wedge (\exists i)(1 \leq i \leq k \wedge u_i \neq \emptyset)\},\ \textit{and}$$

$$T_{maybe} \in \{< r_1,\ldots,r_k > \mid (\forall i)(1 \leq i \leq k \rightarrow r_i \in 2^{D_i^1 \times \cdots \times D_i^{n_i}})\},$$

where $D_i^1,\ldots,D_i^{n_i}$ are the domains associated with the attributes of $R_i, 1 \leq i \leq k$. Elements of T_{sure} are sometimes referred to as *mixed tuple sets.* If a mixed tuple set has exactly one tuple in all of its components then it will be referred to as a *definite* tuple. The tuples in the sure components will sometimes be referred to as *sure* tuples. For notational convenience, we say that the mixed tuple set $u =< u_1,\ldots,u_k >$ is a *subset* of another mixed tuple set $v =< v_1,\ldots,v_k >$, written $u \subseteq v$, if and only if $(\forall i)(1 \leq i \leq k \rightarrow u_i \subseteq v_i)$ and u is a *proper subset* of v, written $u \subset v$, if and only if $(u \subseteq v \wedge (\exists i)(1 \leq i \leq k \wedge u_i \subset v_i))$.

An *M-database scheme* is a collection of M-table schemes. We shall restrict a relation scheme to be present in exactly one M-table scheme of the M-database scheme. An *M-database* is a collection of M-tables defined over the M-database scheme.

Example 5.1.1 Consider the scheme

$$MR =<< UNCLE, PERSON >, < AUNT, PERSON >>.$$

Let us assume that the domain of all persons is associated with each of the attributes $UNCLE$, $AUNT$, and $PERSON$. Figure 5.1 shows an M-table, $UNAUN$, defined over MR. $UNAUN_{sure}$ in Figure 5.1 corresponds to the following ground formulas:

1. UN(Tom,Gary)

UNAUN

UNCLE	*PERSON*	*AUNT*	*PERSON*
Tom	Gary		
Craig	John		
Craig	Don		
		Mary	Tom
		Liz	John
		Liz	Don
Sam	John	Sam	John
Chris	Tom	Chris	Tom
Chris	Gary	Chris	Gary
Jeff	Jake	Pam	Bob

Figure 5.1: M-table *UNAUN*

2. UN(Craig,John) $\vee$ UN(Craig,Don)

3. AUN(Mary,Tom)

4. AUN(Liz,John) $\vee$ AUN(Liz,Don)

5. UN(Sam,John) $\vee$ AUN(Sam,John)

6. UN(Chris,Tom) $\vee$ UN(Chris,Gary) $\vee$ AUN(Chris,Tom) $\vee$ AUN(Chris,Gary)

$UNAUN_{maybe}$ in Figure 5.1 corresponds to the following ground atomic formulas:

1. UN(Jeff,Jake)

2. AUN(Pam,Bob)

However, these formulas may or may not be true.

5.2 Redundancy in M-tables

It is quite possible for redundant information to be present in an M-table. We have identified the following two kinds of redundant information and for each we suggest an action to remove the redundancy. Let $T =< T_{sure}, T_{maybe} >$ be an M-table defined over the scheme $MR =< R_1, \ldots, R_k >$.

1. $u =< u_1, \ldots, u_k > \in T_{sure}$, $v =< v_1, \ldots, v_k > \in T_{sure}$, $u \subset v$, and $T_{maybe} =< r_1, \ldots, r_k >$. Here, v is considered redundant and is removed from T_{sure}. In the process, all the tuples in $v_i - u_i$ are included in r_i.

2. $T_{maybe} =< r_1, \ldots, r_k >$, $t \in r_i$, $< u_1, \ldots, u_k > \in T_{sure}$, and $t \in u_i$, for some i, $1 \leq i \leq k$. Here, t is considered redundant and is simply removed from r_i.

We now present an operator, called $REDUCE$, which removes the above mentioned redundancies from M-tables.

Definition 5.2.1 Let $T_1 =< T^1_{sure}, T^1_{maybe} >$ be an M-table over the scheme $MR =< R_1, \ldots, R_k >$, where $T^1_{maybe} =< r_1, \ldots, r_k >$. Then, $REDUCE(T_1) = T$, where

$T_{sure} = \{u | u \in T^1_{sure} \wedge \neg(\exists v)(v \in T^1_{sure} \wedge v \subset u)\}$,

$T_{maybe} =< r^0_1, \ldots, r^0_k >$, and

$$r^0_j = \{t \mid (t \in r_j \vee (\exists u)(\exists v)(u =< u_1, \ldots, u_k > \in T^1_{sure} \wedge v =< v_1, \ldots, v_k > \in T^1_{sure} \wedge u \subset v \wedge t \in (v_j - u_j))) \wedge \neg(\exists w_1) \ldots (\exists w_k)(< w_1, \ldots, w_k > \in T_{sure} \wedge t \in w_j)\}, 1 \leq j \leq k.$$

Example 5.2.1 Figure 5.2 shows an M-table T and $REDUCE(T)$.

T

UNCLE	PERSON	AUNT	PERSON
John	Tom		
John	Tom		
John	Gary		
		Pat	Gary
Pat	Craig	Pat	Gary
Chris	Dan	Chris	Dan
Don	Hugh	Sam	Jill
Tim	Ron	Bob	Ned

$REDUCE(T)$

UNCLE	PERSON	AUNT	PERSON
John	Tom		
		Pat	Gary
Chris	Dan	Chris	Dan
Don	Hugh	Sam	Jill
Tim	Ron	Bob	Ned
John	Gary		
Pat	Craig		

Figure 5.2: $REDUCE(T)$

5.3 Generalized Relational Algebra

In this section, we generalize the relational algebra to operate on M-tables. We also present an operator, called *R-projection*, which projects an M-table onto some of its relation schemes and an operator, called *merge*, which merges various components of a mixed tuple set into one. The $REDUCE$ operator is part of each of these operators to ensure that no redundant information is introduced.

5.3.1 Selection

The selection operator takes in as input an M-table, T, of order k and k selection formulas $F_1,\ldots,F_k$. A mixed tuple set, $< u_1,\ldots,u_k >$, is selected if for every i, $1 \leq i \leq k$, all tuples in u_i satisfy the selection formula F_i. If not all tuples satisfy the respective selection formula then only those tuples which satisfy the selection formula are included in the respective maybe component of the selection.

Definition 5.3.1 Let T_1 be an M-table over the scheme $MR =< R_1,\ldots,R_k >$, where $T^1_{maybe} =< r_1,\ldots,r_k >$. Also, let $F_1,\ldots,F_k$ be selection formula, where the selection formula F_i involves

1. attribute numbers of R_i,
2. arithmetic comparison connectives $<, \leq, >, \geq, =, \neq$, and
3. logical connectives $\wedge$, $\vee$, and $\neg$.

Then, $\sigma_{<F_1,\ldots,F_k>}(T_1) = REDUCE(T)$, where

$$T_{sure} = \{< u_1,\ldots,u_k > \mid < u_1,\ldots,u_k > \in T^1_{sure} \wedge (\forall i)(1 \leq i \leq k \rightarrow (\forall t)(t \in u_i \rightarrow F_i(t)))\},$$

$T_{maybe} =< r_1^0, \ldots, r_k^0 >$,

$$r_j^0 = \{t \mid (t \in r_j \wedge F_j(t)) \vee (\exists u_1) \cdots (\exists u_k)(< u_1, \ldots, u_k > \in T_{sure}^1 \wedge t \in u_j \wedge F_j(t))\}, \text{ and}$$

$F_i(t)$ is F_i with attribute j replaced by $t[j]$.

Example 5.3.1 An example of the selection operator is shown in Figure 5.3.

T

A_1	A_2	B_1	B_2
John	A	John	100
John	B	John	200
Craig	C	Pat	100
Craig	D		
Tom	A	Tom	600
Gary	A	John	500
Robin	D	Robin	200
Don	A	Don	100
Don	C		

$\sigma_{<F_1,F_2>}(T)$

A_1	A_2	B_1	B_2
John	A	John	100
John	B	John	200
Tom	A	Tom	600
Gary	A	John	500
Don	A		

$F_1 = ((2 = "A") \vee (2 = "B"))$ and $F_2 = ((1 = "John") \vee (2 \geq "600"))$

Figure 5.3: Selection

5.3.2 Projection

The projection operator takes in as input an M-table, T, of order k and k lists of projection attributes $A_1, \ldots, A_k$. The ith component of a mixed tuple set of T_{sure} is

projected onto A_i and the ith component of T_{maybe} is projected onto A_i, for each i.

Definition 5.3.2 Let T_1 be an M-table over the scheme $MR =< R_1, \ldots, R_k >$ where $T_{maybe} =< r_1, \ldots, r_k >$. Also let $A_1, \ldots, A_k$ be lists of projection attributes, where A_i involves attributes of R_i. Then, $\Pi_{<A_1,\ldots,A_k>}(T_1) = REDUCE(T)$, where T is defined as follows:

$$T_{sure} = \{< u_1, \ldots, u_k > \mid (\exists v_1) \cdots (\exists v_k)(< v_1, \ldots, v_k > \in T^1_{sure} \wedge (\forall i)(1 \leq i \leq k \rightarrow u_i = \Pi_{A_i}(v_i)))\}$$

$$T_{maybe} =< \Pi_{A_1}(r_1), \ldots, \Pi_{A_k}(r_k) > .$$

Example 5.3.2 An example of the projection operator is shown in Figure 5.4.

T

A_1	A_2	A_3	B_1	B_2
John	A	100	John	A
John	A	200		
Tom	A	200	Tom	A
Tom	B	200	Tom	B
Gary	C	300	Gary	C
Gary	D	100	Gary	E
Craig	A	100	Brad	A
Don	A	100		
Jones	A	100	Bill	C
			Bob	D

$\Pi_{<<1,2>,<1>>}(T)$

A_1	A_2	B_1
John	A	John
Tom	A	Tom
Tom	B	
Gary	C	Gary
Gary	D	
Craig	A	Brad
Don	A	
Jones	A	Bill
		Bob

Figure 5.4: Projection

5.3.3 Cartesian Product

Consider the two M-tables, T_1 and T_2, in Figure 5.5 and let T_1 be defined over the scheme $< P, Q >$ and T_2 be defined over the scheme $< R, S >$. Also let $T = T_1 \times T_2$. The sure component of T is computed as follows:

The two mixed tuple sets of T_1 together with the single mixed tuple set of T_2 gives us the following disjunctive formula:

$$(PR(a,e) \wedge PR(c,e)) \vee (PS(a,f) \wedge PS(c,f)) \vee$$
$$(PR(a,e) \wedge QR(d,e)) \vee (PS(a,f) \wedge QS(d,f)) \vee$$
$$(QR(b,e) \wedge PR(c,e)) \vee (QS(b,f) \wedge PS(c,f)) \vee$$
$$(QR(b,e) \wedge QR(d,e)) \vee (QS(b,f) \wedge QS(d,f))$$

Converting this expression into the conjunctive normal form and simplifying, we obtain the four mixed tuple sets of T_{sure}.

The maybe component of T is computed by taking the cross product of sure tuples of T_1 with maybe tuples of T_2, maybe tuples of T_1 with sure tuples of T_2, and maybe tuples of T_1 with maybe tuples of T_2.

Using the above methodology, we obtain the cartesian product in Figure 5.5. The above discussion is formalized into the following definition:

Definition 5.3.3 Let T_1 be an M-table defined over the scheme $MR_1 =< R_1, \ldots, R_k >$ and T_2 be an M-table defined over the scheme $MR_2 =< S_1, \ldots, S_l >$. Then, $T_1 \times T_2$ is an M-table defined over the scheme

$$MR =< R_1.S_1, \ldots, R_1.S_l, \ldots, R_k.S_1, \ldots, R_k.S_l >,$$

where $R_i.S_j$ is the concatenation of the schemes R_i and S_j. Let

$T^1_{sure} = \{< u_{11}, \ldots, u_{1k} >, \ldots, < u_{m1}, \ldots, u_{mk} >\}$,

$T^1_{maybe} =< r_1, \ldots, r_k >$,
$T^2_{sure} = \{< v_{11}, \ldots, v_{1k} >, \ldots, < v_{n1}, \ldots, v_{nl} >\}$, and
$T^2_{maybe} =< s_1, \ldots, s_l >$.
Also let

$$E = \{< u_1, \ldots, u_k > \mid (\exists d_1)(\exists t_1)\cdots(\exists d_m)(\exists t_m)((\forall i)(1 \le i \le m \rightarrow (1 \le d_i \le k \wedge t_i \in u_{id_i})) \wedge < u_1, \ldots, u_k >= collate^k(< t_1, \ldots, t_m >, < d_1, \ldots, d_m >))\},$$

$$F = \{< u_1, \ldots, u_l > \mid (\exists d_1)(\exists t_1)\cdots(\exists d_n)(\exists t_n)((\forall i)(1 \le i \le n \rightarrow (1 \le d_i \le l \wedge t_i \in v_{id_i})) \wedge < u_1, \ldots, u_l >= collate^l(< t_1, \ldots, t_n >, < d_1, \ldots, d_n >))\}.$$

Let $|E| = e$ and $|F| = f$ and let $E_1, \ldots, E_e$ and $F_1, \ldots, F_f$ be the elements of E and F respectively, ordered in any manner. Let

$$EF_{ij,ab} = \{t \mid (\exists t_1)(\exists t_2)(E_i =< u_1, \ldots, u_k > \wedge F_j =< v_1, \ldots, v_l > \wedge t_1 \in u_a \wedge t_2 \in v_b \wedge t = t_1.t_2)\},$$

$1 \le i \le e$, $1 \le j \le f$, $1 \le a \le k$, and $1 \le b \le l$. There exists a one-one mapping, f, from the set of pairs $< i, j >$ of positive integers onto consecutive positive integers. We shall use this mapping to rename the $EF_{ij,ab}$s as $EF_{f(i,j),f(a,b)}$s. Let $c = k \times l$ and let g be the number of distinct EF_{ij}s for a fixed j. Then, $T_1 \times T_2 = REDUCE(T)$, where

$$T_{sure} = \{< u_1, \ldots, u_c > \mid (\exists d_1)(\exists t_1)\cdots(\exists d_g)(\exists t_g)((\forall i)(1 \le i \le g \rightarrow (1 \le d_i \le c \wedge t_i \in EF_{id_i})) \wedge < u_1, \ldots, u_c >= collate^c(< t_1, \ldots, t_g >, < d_1, \ldots, d_g >))\},$$

$T_{maybe} =< r_{11},\ldots,r_{1l},\ldots,r_{k1},\ldots,r_{kl} >,$

$$r_{ij} = \{t \mid (\exists t_1)(\exists t_2)(t_1 \in r_i \wedge (\exists u_1)\cdots(\exists u_l)($$
$$< u_1,\ldots,u_l > \in T^2_{sure} \wedge t_2 \in u_j) \wedge t = t_1.t_2) \vee$$
$$(\exists t_1)(\exists t_2)((\exists u_1)\cdots(\exists u_k)(< u_1,\ldots,u_k > \in T^1_{sure} \wedge$$
$$t_1 \in u_i) \wedge t_2 \in s_j \wedge t = t_1.t_2) \vee$$
$$(\exists t_1)(\exists t_2)(t_1 \in r_i \wedge t_2 \in s_j \wedge t = t_1.t_2)\}, 1 \le i \le k, 1 \le j \le l, \textit{ and}$$

$collate^k(< t_1,\ldots,t_n >,< d_1,\ldots,d_n >)$ is a function that returns a mixed tuple set $< u_1,\ldots,u_k >$ by placing t_i in u_{d_i}, $1 \le i \le n$.

Example 5.3.3 An example of the cartesian product is shown in Figure 5.5.

T_1

P	Q
a	b
c	d
g	h
i	

T_2

R	S
e	f
j	k

$T_1 \times T_2$

P	R	P	S	Q	R	Q	S
a	e	a	f	b	e	b	f
a	e	c	f	b	e	d	f
c	e	a	f	d	e	b	f
c	e	c	f	d	e	d	f
a	j	a	k	b	j	b	k
c	j	c	k	d	j	d	k
g	j	g	k	h	j	h	k
i	j	i	k	h	e	h	k
g	e	g	f				
i	e	i	f				

Figure 5.5: Cartesian Product

5.3.4 Union

The union of two domain compatible M-tables is simply the union of the respective sure and maybe components. $REDUCE$ is applied to the resulting M-table to remove any redundant information.

Definition 5.3.4 Let $T_1 =< T^1_{sure}, T^1_{maybe} >$ and $T_2 =< T^2_{sure}, T^2_{maybe} >$ be two M-tables defined over the scheme $MR =< R_1, \ldots, R_k >$. Then,

$T_1 \cup T_2 = REDUCE(T)$, where

$T_{sure} = T^1_{sure} \cup T^2_{sure}$, and

$T_{maybe} = T^1_{maybe} \cup T^2_{maybe}$.

Example 5.3.4 An example of the union operator is shown in Figure 5.6.

5.3.5 Difference

The difference of two domain-compatible M-tables T_1 and T_2 is computed as follows:

1. If a mixed tuple set, u, of T_1 has no common tuples with any mixed tuple set of T_2 or with any maybe tuple of T_2, then it is included in the sure component of the difference. Otherwise, all the tuples in u that do not appear as a definite tuple in T_2 are included in the corresponding maybe components of the difference.

2. A maybe tuple of T_1 that does not appear as a definite tuple in T_2 is included in the corresponding maybe components of the difference.

The above discussion is formalized in the following definition:

Definition 5.3.5 Let T_1 and T_2 be two M-tables defined over the scheme

T_1

R_1	R_2
a	
b	
c d	
e f	
	g
	h
	i j
k	l
m o	n p
q	r

T_2

R_1	R_2
c	
s	
t u	
	g v
m	n
w y	x z
e	l

$T_1 \cup T_2$

R_1	R_2
a	
b	
c	
s	
t u	
e f	
	g
	h
	i j
k	l
m	n
w y	x z
q	r
o	p
d	v

Figure 5.6: Union

$$MR =< R_1, \ldots, R_k >,$$

where $T^1_{maybe} =< r_1, \ldots, r_k >$ and $T^2_{maybe} =< s_1, \ldots, s_k >$.
Then, $T_1 - T_2 = REDUCE(T)$, where

$$\begin{aligned} T_{sure} = \{< u_1, \ldots, u_k > \mid\ & < u_1, \ldots, u_k > \in T^1_{sure} \wedge \\ & \neg(\exists v_1)\cdots(\exists v_k)(\exists i)((< v_1, \ldots, v_k > \in T^2_{sure} \wedge \\ & 1 \leq i \leq k \wedge u_i \cap v_i \neq \emptyset) \wedge \\ & \neg(\exists i)(1 \leq i \leq k \wedge u_i \cup s_i \neq \emptyset)\}, \end{aligned}$$

$T_{maybe} =< r^0_1, \ldots, r^0_k >$, *and*

$$\begin{aligned} r^0_j = \{t \mid\ & (\exists u_1)\cdots(\exists u_k)(< u_1, \ldots, u_k > \in T^1_{sure} \wedge t \in u_j) \vee t \in r_j) \wedge \\ & \neg(\exists v_1)\cdots(\exists v_k)(< v_1, \ldots, v_k > \in T^2_{sure} \wedge v_j = \{t\} \wedge \\ & (\forall i)(1 \leq i \leq k \wedge i \neq j \rightarrow v_j = \emptyset))\}. \end{aligned}$$

Example 5.3.5 An example of the difference operator is shown in Figure 5.7.

5.3.6 R-projection

The R-projection operator takes in as input an M-table, T, of order k and n relation schemes $R_1, \ldots, R_n$ which are among the relation schemes of T. It returns an M-table over the scheme $< R_1, \ldots, R_n >$. If a mixed tuple set in T_{sure} has empty sets in all the components which do not correspond to any of the R_is then the mixed tuple set is included in the sure component of the R-projection. Otherwise, all the tuples from the components that correspond to the R_is are included in the respective maybe components of the R-projection. T_{maybe} is also projected onto

T_1

R_1	R_2
a	
b	
c d	
	e
	f
	g h
i	j
k m	l
n	o p
q	s
r	t

T_2

R_1	R_2
b	
q	
	g
	o
	p
u	e
k	l
v	w

$T_1 - T_2$

R_1	R_2
a	
c d	
	f
i	j
k	e
m	g
n	h
	l

Figure 5.7: Difference

$< R_1, \ldots, R_k >$ to contribute tuples to the maybe component of the R-projection.

Definition 5.3.6 Let T_1 be an M-table defined over the scheme $MR =< R_1, \ldots, R_k >$ where $T^1_{maybe} =< r_1, \ldots, r_k >$. Also let $R_{i_1}, \ldots, R_{i_n}$ be relation schemes such that

1. $n \leq k$,
2. $R_{i_j} \in \{R_1, \ldots, R_k\}$, $1 \leq j \leq n$, and
3. $R_{i_{j1}} = R_{i_{j2}}$ if and only if $j1 = j2$.

Then, $\Pi_{<R_{i_1},\ldots,R_{i_n}>}(T_1) = REDUCE(T)$, where T is an M-table over the scheme $< R_{i_1}, \ldots, R_{i_n} >$ and is defined as follows:

$$T_{sure} = \{< u_1, \ldots, u_n > \mid (\exists v_1) \cdots (\exists v_k)(< v_1, \ldots, v_k > \in T^1_{sure} \wedge (\forall j)(1 \leq j \leq n \rightarrow u_j = v_{i_j}) \wedge (\forall j)((1 \leq j \leq k \wedge j \notin \{i_1, \ldots, i_n\}) \rightarrow v_j = \emptyset))\},$$

$T_{maybe} =< r^0_1, \ldots, r^0_n >$, *and*

$$r^0_j = \{t \mid (t \in r_{i_j}) \vee (\exists u_1) \cdots (\exists u_k)(< u_1, \ldots, u_k > \in T^1_{sure} \wedge (\exists l)(1 \leq l \leq k \wedge l \notin \{i_1, \ldots, i_n\} \wedge u_l \neq \emptyset) \wedge t \in u_{i_j})\}, 1 \leq j \leq n.$$

Example 5.3.6 An example of the R-projection operator is shown in Figure 5.8.

5.3.7 Merge

The merge operator is defined on M-tables which are defined over the scheme $< R_1, \ldots, R_k >$ where the relation schemes $R_1, \ldots, R_k$ are all domain-compatible.

T

R_1	R_2
a	
b c	
	d
	e
	f g
h	i
j k	l
m	o
n	

$\Pi_{R_2}(T)$

R_2
d
e
f g
i
l
o

Figure 5.8: R-projection

It returns an M-table over the scheme $< R_1 >$. The k components of a mixed tuple set are all merged into one component and the k sets of maybe tuples are also merged into one. The formal definition of merge is presented below:

Definition 5.3.7 Let T_1 be an M-table defined over the scheme

$$MR =<< A_{11}, \ldots, A_{1n} >, \ldots, < A_{k1}, \ldots, A_{kn} >>,$$

such that the domains associated with the attributes $A_{1i}, \ldots, A_{ki}$, for a fixed i, are all the same. Also let

$$T^1_{maybe} = < r_1, \ldots, r_k >.$$

Then, $merge(T_1) = REDUCE(T)$, where $T =< T_{sure}, T_{maybe} >$ is an M-table defined over the scheme $<< A_{11}, \ldots, A_{1n} >>$ and is defined as follows:

$T_{sure} = \{< u > |(\exists u_1) \cdots (\exists u_k)(< u_1, \ldots, u_k > \in T^1_{sure} \wedge u = u_1 \cup \cdots \cup u_k)\}$

$T_{maybe} =< r_1 \cup \cdots \cup r_k >$.

Example 5.3.7 An example of the merge operator is shown in Figure 5.9.

5.4 Queries

Queries can be expressed as a combination of the various generalized relational algebraic operators defined earlier. The M-table accurately models the two bounds on the external interpretation of a query (the interpretation in which the query is referred to the real world modeled in an incomplete way by the system [29]). The sure component of an M-table corresponds to one of the bounds which is the set of objects for which we can positively say that they belong to the external interpretation of the query. The maybe component of an M-table corresponds to the other bound which is the set of objects for which we cannot rule out the possibility of belonging

T

A_1	A_2	B_1	B_2
John	A	John	A
Tom	A	Tom	A
Tom	B	Tom	B
Gary	C	Gary	C
Gary	D	Gary	E
Craig	A	Brad	A
Don	A		
Jones	A	Bill	C
		Bob	D

$merge(T)$

A_1	A_2
John	A
Tom	A
Tom	B
Gary	C
Gary	D
Gary	E
Craig	A
Don	A
Brad	A
Jones	A
Bill	C
Bob	D

Figure 5.9: Merge

to the external interpretation of the query. We now present two examples of queries in the generalized relational model.

Example 5.4.1 Consider the database in Figure 5.10 which consists of the two M-tables: SP defined over the scheme $<< SUPPLIER, PART >>$ and P defined over the scheme $<< PART, COLOR >>$. Also consider the query: Find all the suppliers

SP

SUPPLIER	*PART*
s1	p1
s2	p3
s3	p5
s4	p3
s4	p4
s5	p3

P

PART	*COLOR*
p1	blue
p2	green
p3	red
p4	red
p5	red
p6	red

Figure 5.10: A Database

who supply "red" parts. The query represented in the generalized relational algebra is:

$$ANSWER = \Pi_{<<1>>}(\sigma_{<2=3>}(SP \times \Pi_{<<1>>}(\sigma_{2="red"}(P))))$$

Evaluating this expression against the database in Figure 5.10, we obtain the answer in Figure 5.11. The answer can be interpreted in the following manner: s2 and s4 supply "red" parts and s3 and s5 may supply "red" parts.

Example 5.4.2 Consider the database in Figure 5.12 which consists of two M-tables:

1. SIB defined over the scheme $<< PERSON, SIBLING >>$, and

ANSWER

SUPPLIER
s2
s4
s3
s5

Figure 5.11: Answer to Query

2. $MAFA$ defined over the scheme

$$<< M - ANCESTOR, PERSON >, < F - ANCESTOR, PERSON >>.$$

The M-table SIB represents the sibling relationship and the M-table $MAFA$ represents the mixed relationships male-ancestor and female-ancestor. Consider the query: Find all the siblings of the ancestors, male or female, of "Tom". In the generalized relational algebra, this query is translated as:

$$ANSWER = merge(\Pi_{<<1>,<1>>}(\sigma_{<F_1,F_1>}(SIB \times MAFA))),$$

where F_1 is $(2 = 3) \wedge (4 = "Tom")$. Evaluating this expression against the database of Figure 5.12, we obtain the answer in Figure 5.13. The answer can be interpreted in the following manner: Pam, Gary, and Liz are siblings of ancestors of Tom, at least one of Craig or Don are siblings of the ancestors of Tom, and Bill may be a sibling of an ancestor of Tom.

As the above two examples illustrate, the query is posed in the same way as for conventional relational databases.

SIB

PERSON	*SIBLING*
Gary	Chris
Pam	Mark
Liz	Pat
Craig	Mark
Don	Mark
Bill	Bob

MAFA

M – ANCESTOR	*PERSON*	*F – ANCESTOR*	*PERSON*
Mark	Tom		
		Pat	Tom
Chris	Tom	Chris	Tom
Bob	Tom		

Figure 5.12: A Database

ANSWER

PERSON
Pam
Gary
Liz
Craig
Don
Bill

Figure 5.13: Answer to Query

6 SUMMARY AND CONCLUSION

We have presented a extended relational model to represent indefinite and maybe kinds of incomplete information. The relational algebra has been extended, in a semantically correct way, to manipulate these kinds of information.

The disjunctive information represented in the indefinite component of I-tables and M-tables corresponds to the *inclusive or* variety, i.e., more than one tuple within a tuple set may be in the relation. To handle the *exclusive or* variety of disjunctions, we will have to modify some of the operators defined in this paper.

Query optimization is an issue which needs to be studied in great detail. Some of the techniques used in the conventional relational model may be applied to our extended model too. Combining selections and cartesian products to obtain joins can drastically reduce the size of intermediate I-tables and M-tables. Transforming the extended relational algebraic expressions, as explained in [34,44], can improve running times of queries.

Enforcement of integrity constraints in an I-database is another issue for further study. Let D be a set of integrity constraints. We define $SAT(D)$ as follows:

$$SAT(D) = \{< U, v > \mid < U, v > \in \Sigma_R \wedge (\forall r)(r \in U \rightarrow SATISFIES(r, D)) \wedge (\forall r)(r \subseteq v \rightarrow (\forall r_1)(r_1 \in U \rightarrow SATISFIES(r \cup r_1, D)))\},$$

where $SATISFIES(r, D)$ means that the relation r satisfies all the constraints in D.

Given an I-table, T, and a set of integrity constraints D, we now define the information content of T as $REP(T) \cap SAT(D)$ instead of just $REP(T)$. In order to enforce the integrity constraints D on I-table T, we must define an operator, $subj(T, D)$, which returns an I-table and which satisfies the following condition:

$$REP(subj(T, D)) = REP(T) \cap SAT(T, D).$$

A similar definition can be made for M-tables. The definition of $subj(T, D)$, for any D, is a topic for future research and is under investigation.

Updates to I-tables and M-tables is another topic for future research. Updates to incompletely specified databases have been studied in [1,2,4,12,26]. The insert, delete, and modify operations need to be defined in such a way as to maintain all the integrity constraints on the database. The $REDUCE$ operator needs to be invoked on an insert or a modify to maintain a reduced database. The effect of data dependencies on relational databases with null values has been studied in [23,28,47]. A similar analysis needs to be done for I-tables and M-tables.

With the growing interest in deductive databases [16], definite as well as indefinite, we feel that one needs to consider new models to handle indefinite information. The proof-theoretic approach to indefinite deductive databases is impractical as it is very inefficient to employ theorem provers, especially in the context of large indefinite databases. The conventional relational algebra can be used effectively to implement definite deductive databases. However, it cannot be used in the context of indefinite deductive databases. Our extended relational model has been shown to implement a subclass of indefinite deductive databases in [30].

Another area where the extended relational model can be applied is uncertain/fuzzy databases. By assigning numerical values to tuples in an I-tables and

M-tables, we could enhance the quality of information being modeled.

Finally, we discuss some previous research that is closely related to our work. Lipski presents a general theory of incomplete information databases in [29]. He distinguishes between the internal interpretation of a query which is based on the information present in the database and the external interpretation which is based on the real world truth. Our work is related to answering queries in the external interpretation. Imielinski [22] represents incomplete information in V-tables and C-tables. Null values are treated as variables in V-tables. The relational algebraic operators cartesian product, projection, and positive selection on V-tables are the same as for relations. C-tables are generalizations of V-tables, where each row contains a condition. C-tables are capable of representing more general kinds of incomplete information, including disjunctions. The relational algebra is extended to operate on C-tables. This yields another approach to answering queries in the context of indefinite databases. Grant and Minker [20] work within the framework of database theories which contain the domain closure axiom, the unique name axiom, and the equality axioms. Queries are formulas in first-order logic. An algorithm to check if a candidate answer is indeed an answer is presented. An algorithm to find all minimal answers to queries is also presented. This is yet another approach to answering queries in indefinite databases.

7 BIBLIOGRAPHY

[1] Abiteboul, S. and Grahne, G. "Update semantics for incomplete information". *Proceedings of the 11th International Conference on Very Large Data Bases, Stockholm*, 1985, 1–12.

[2] Abiteboul, S. and Vianu, V. "Transactions in relational databases". *Proceedings of the 10th International Conference on Very Large Data Bases, Singapore*, 1984, 46–56.

[3] Bancilhon, F. and Ramakrishnan, R. "An amateur's introduction to recursive query processing strategies". *Proceedings of ACM SIGMOD International Conference on Management of Data, Washington, DC*, May 1986, 16–52.

[4] Bancilhon, F. and Spyratos, N. "Update semantics of relational views". *ACM Transactions on Database Systems* 6(1981):557–575.

[5] Biskup, J. "A formal approach to null values in database relations". *In Advances in Database Theory, V1*, ed. H. Gallaire, J. Minker, and J. Nicolas, 299–341, New York and London: Plenum Press, 1981.

[6] Biskup, J. "A foundation of Codd's relational maybe-operations". *ACM Transactions on Database Systems* 8(1983):608–636.

[7] Biskup, J. "Extending the relational algebra for relations with maybe tuples and existential and universal null values". *Fundamenta Informaticae* 7(1984):129–150.

[8] Chang, C.L. "On the evaluation of queries containing derived relations in relational databases". *In Advances of Data Base Theory, V1*, ed. H. Gallaire, J. Minker, and J. Nicolas, 235–260, New York: Plenum Press, 1981.

[9] Codd, E.F. "A relational model for large shared data banks". *Communications of the ACM* 13(1970):377–387.

[10] Codd, E.F. "Understanding relations". *Installment #7, FDT Bulletin of ACM Record* 7(1975):23–28.

[11] Codd, E.F. "Extending the database relational model to capture more meaning". *ACM Transactions of Database Systems* 4(1979):397–434.

[12] Fagin, R., Ullman, J.D. and Vardi, M. "On the semantics of updates in databases". *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Atlanta, GA*, 1983. 352–365.

[13] Gallaire, H. and Minker, J. *Logic and Data Bases.* New York: Plenum Press, 1978.

[14] Gallaire, H., Minker, J. and Nicolas, J.M. *Advances in Data Base Theory.* Volume 1. New York: Plenum Press, 1981.

[15] Gallaire, H., Minker, J. and Nicolas, J.M. *Advances in Data Base Theory.* Volume 2. New York: Plenum Press, 1984.

[16] Gallaire, H., Minker, J. and Nicolas, J.M. "Logic and databases: A deductive approach". *ACM Computing Surveys* 16(1984):151–184.

[17] Grant, J. "Null values in a relational database". *Information Processing Letters* 6(1977):156–157.

[18] Grant, J. "Partial values in a tabular database model". *Information Processing Letters* 9(1979):97–99.

[19] Grant, J. "Incomplete information in a relational database". *Fundamenta Informaticae* 3(1980):363–378.

[20] Grant, J., and J. Minker. "Answering queries in indefinite databases and the null value problem". *In Advances in Computing Research, Volume 3*:JAI Press Inc., 1986. 247–267.

[21] Henschen, L. and Naqvi, S. "On compiling queries in recursive first-order data bases". *Journal of the ACM* 31(1984):47–85.

[22] Imielinski, T. "On algebraic query processing in logical databases". *In Advances in Database Theory, Vol. 2.* Ed. H. Gallaire, J. Minker, and J. M. Nicolas, 285–318, New York and London: Plenum Press, 1984.

[23] Imielinski, T. and Lipski, W. "Incomplete information and dependencies in relational databases". *Proceedings of the ACM SIGMOD Conference on Data Management, San Jose, CA*, May 1983, 178–184.

[24] Imielinski, T. and Lipski, W. "Incomplete information in relational databases". *Journal of the ACM* 31(1984):761–791.

[25] Ioannidis, Y.E. and Wong, E. "An algebraic approach to recursive inference". *Proceedings of the First International Conference on Expert Database Systems, Charleston, South Carolina*, 1986, 209–223.

[26] Kuper, G.M. and Ullman, J.D. and Vardi, M. "On the equivalence of logical databases". *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Waterloo, Ontario*, 1984, 221–228.

[27] Levesque, H.J. "The interaction with incomplete knowledge bases: A formal treatment". *Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, B.C.*, August 1981, 240–245.

[28] Lien, E. "Multivalued dependencies with null values in relational databases". *Proceedings of the 5th International Conference on Very Large Data Bases*, 1979, 61–66.

[29] Lipski, W. "On semantic issues connected with incomplete information systems". *ACM Transactions on Database Systems* 4(1979):262–296.

[30] Liu, K.C. and Sunderraman, R. "Applying an extended relational model to indefinite deductive databases". *Proceedings of the 2nd International Symposium on Methodologies for Intelligent Systems, Charlotte, NC*, New York: Elsevier Press, 1987, 175–184.

[31] Liu, K.C. and Sunderraman, R. "An extension to the relational model for indefinite databases". *Proceedings of the ACM-IEEE Computer Society Fall Joint Computer Conference, Dallas, TX*, October 1987, 428–435.

[32] Liu, K.C. and Sunderraman, R. "Indefinite and maybe information in relational databases". January 1988. Submitted to ACM Transactions on Database Systems.

[33] Liu, K.C. and Sunderraman, R. "On representing indefinite and maybe information in relational databases". *Proceedings of the Fourth International Conference on Data Engineering, Los Angeles, California*, 1988, 250–257.

[34] Maier, D. *The Theory of Relational Databases.* Rockville, Maryland: Computer Science Press, 1983.

[35] McKay, D. and Shapiro, S. "Using active connection graphs for reasoning with recursive rules". *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981, 368–374.

[36] Minker, J. "On indefinite databases and the closed world assumption". *In Lecture Notes in Computer Science, N138*, 292–308, New York:Springer-Verlag, 1982.

[37] Minker, J. and Nicolas, J.M. "On recursive axioms in deductive databases". *Information Systems* 8(1982):1–13.

[38] Minker, J. and Perlis, D. "Applications of protected circumscription". *Proceedings of the Conference of Automated Deduction, Napa, CA, May 1984*. New York: Springer-Verlag 1984, 414–425.

[39] Reiter, R. "On closed world databases". *In Logic and Databases*, ed. H. Gallaire, and J. Minker, 56–76, New York: Plenum Press, 1978.

[40] Reiter, R. "Towards a logical reconstruction of relational database theory". *In On Conceptual Modeling* , ed. M. Brodie, J. Mylopoulos, and J. Schmidt, 191–238, New York: Springer-Verlag, 1984.

[41] Reiter, R. "A sound and sometimes complete query evaluation algorithm for relational databases with null values". *Journal of the ACM* 33(1986):349–370.

[42] Schwind, C.B. "Embedding deductive capabilities in relational database systems". *International Journal of Computer and Information Sciences* 13(1984):327–338.

[43] Tarski, A. "A lattice-theoretical fixpoint theorem and its applications". *Pacific J. Mathematics* 5(1955):285–309.

[44] Ullman, J.D. *Principles of Database Systems*. 2nd ed., Potomac, Maryland: Computer Science Press, 1982.

[45] Ullman, J.D. "Implementation of logical query languages for databases". *ACM Transactions on Database Systems* 10(1985):289–321.

[46] Vassiliou, Y. "Null values in data base management - a denotational semantics approach". *Proceedings of the ACM-SIGMOD International Conference on the Management of Data, Boston*, May-June 1979, 162–169.

[47] Vassiliou, Y. "Functional dependencies and incomplete information". *Proceedings of the 6th International Conference on Very Large Data Bases*, 1981, 260–269.

[48] Vielle, L. "Recursive axioms in deductive databases : The query/subquery approach". *Proceedings of the First International Conference on Expert Database Systems, Charleston, SC*, 1986.

[49] Yahya, A. and Henschen, L. "Deduction in non-Horn databases". *Journal of Automated Reasoning* 1(1985):141-160.

[50] Zaniolo, C. "Database relations with null values". *Journal of Computer and System Sciences* 28(1984):142–166.

8 ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Dr. Ken-Chih Liu for all the guidance he has provided throughout my stay at Iowa State. The countless discussions we had over the past five years or so have enriched my understanding of logic and databases.

I would also express my appreciation towards Dr. Oldehoeft, Dr. Schmidt, Dr. Stewart, and Dr. Pohm for serving on my POS committee. Special thanks goes to Dr. Schmidt for the numerous suggestions to improve Chapter 2 and for traveling between Manhattan, Kansas and Ames, Iowa for my preliminary and final examinations.

I would like to thank Mrs. LaDena Bishop for her suggestions to improve the presentation of thesis.

Finally, I must thank my parents for their constant support and encouragement without which I would have found it difficult to achieve my goals.

9 APPENDIX

Here, we present proofs for all the theorems stated in the paper. For convenience, we shall assume that the definite component of an I-table is made of singleton sets of tuples instead of tuples. Consequently, we shall refer to the definite and indefinite components as one component called the *sure* component. This assumption is made only for some of the proofs that follow. We shall assume $T =< T_{sure}, T_{maybe} >$ and $T_i =< T^i_{sure}, T^i_{maybe} >$.

Theorem 3.1.3 $REP(T) = REP(REDUCE(T))$ for any I-table $T \in \Gamma_R$.

Proof: Let $T_{sure} = \{w_1, \ldots, w_k, w_{k+1}, \ldots, w_n\}$ such that

(1) $(\forall i)(k+1 \leq i \leq n \rightarrow (\exists j)(1 \leq j \leq k \wedge w_j \subset w_i))$, and

(2) $(\forall i)(1 \leq i \leq k \rightarrow \neg(\exists j)(1 \leq j \leq k \wedge w_j \subset w_i))$.

Also let $T_1 = REDUCE(T)$. Then, by definition of $REDUCE$, $T^1_{sure} = \{w_1, \ldots, w_k\}$. Let $< MM, M > (REDUCE(T)) =< U_1, v_1 >$, $< MM, M > (T) =< U_2, v_2 >$, $REP(REDUCE(T)) =< U'_1, v'_1 >$, and $REP(T) =< U'_2, v'_2 >$.

Claim 1: $U'_1 = U'_2$.

i) Consider $r_1 = \{t_1, \ldots, t_k\} \in U_1$, where $t_i \in w_i$, $1 \leq i \leq k$. Since (1), there exists $r_2 = \{t_1, \ldots, t_k, t_{k+1}, \ldots, t_n\} \in U_2$, where $t_i \in w_i$, $1 \leq i \leq n$ and $t_j \in \{t_1, \ldots, t_k\}$, $k+1 \leq j \leq n$. Therefore, $r_2 = r_1$ and hence $r_1 \in U_2$, i.e., $U_1 \subseteq U_2$.

ii) Consider $r_1 = \{t_1,\ldots,t_k,t_{k+1},\ldots,t_n\} \in U_2 - U_1$, where $t_i \in w_i$, $1 \leq i \leq n$. Since, $r_1 \in U_2 - U_1$, there exists j, $k+1 \leq j \leq n$, such that $t_j \notin \{t_1,\ldots,t_k\}$. Note that $r_2 = \{t_1,\ldots,t_k\} \in U_1$ and from i) $r_2 \in U_2$. Clearly, $r_2 \subset r_1$. Therefore, for every $r_1 \in U_2 - U_1$ there exists $r_2 \in U_2$ such that $r_2 \subset r_1$.

Therefore, from i) and ii) and definition of $REDUCEREP$, we conclude that $U_1' = U_2'$.

Claim 2: $v_1' = v_2'$.

Let $U_1 = \{r_1,\ldots,r_a\}$ and $U_2 = \{r_1,\ldots,r_a,r_{a+1},\ldots r_{a+b}\}$ such that

(1) $(\forall i)(a+1 \leq i \leq a+b \rightarrow (\exists j)(1 \leq j \leq a \wedge r_j \subset r_i))$, and

(2) $(\forall i)(1 \leq i \leq a \rightarrow \neg(\exists j)(1 \leq j \leq a \wedge r_j \subset r_i))$.

Then, by the definition of $REDUCEREP$ $t \in v_2'$ if and only if

(3) $((t \in v_2) \wedge \neg(\exists r)(r \in U_2' \wedge t \in r))$ or

(4) $(\exists i)(\exists j)(a+1 \leq i \leq a+b \wedge 1 \leq j \leq a \wedge r_j \subset r_i \wedge t \in r_i - r_j) \wedge \neg(\exists r)(r \in U_2' \wedge t \in r)$.

Case 1 : $(t \in v_2) \wedge \neg(\exists r)(r \in U_2' \wedge t \in r)$

iff $(t \in T_{maybe}) \wedge \neg(\exists w)(w \in \{w_1,\ldots,w_k\} \wedge t \in w)$ (By definition of $< MM, M >$)

iff $(t \in T^1_{maybe}) \wedge \neg(\exists w)(w \in T^1_{sure} \wedge t \in w)$ (By definition of $REDUCE$)

iff $(t \in v_1) \wedge \neg(\exists r)(r \in U_1' \wedge t \in r)$ (By definition of $< MM, M >$)

iff $(t \in v_1')$ (By definition of $REDUCEREP$).

Case 2 : $(\exists i)(\exists j)(a+1 \leq i \leq a+b \wedge 1 \leq j \leq a \wedge r_j \subset r_i \wedge t \in r_i - r_j) \wedge \neg(\exists r)(r \in U_2' \wedge t \in r)$

iff $(\exists i)(\exists j)(1 \leq i \leq k \wedge k+1 \leq j \leq n \wedge w_i \subset w_j \wedge t \in w_j - w_i) \wedge$
$\neg(\exists w)(w \in \{w_1, \ldots, w_k\} \wedge t \in w)$ (By definition of $< MM, M >$)
iff $(t \in T^1_{maybe}) \wedge \neg(\exists w)(w \in T^1_{sure} \wedge t \in w)$ (By definition of $REDUCE$)
iff $(t \in v_1) \wedge \neg(\exists r)(r \in U'_1 \wedge t \in r)$ (By definition of $< MM, M >$)
iff $(t \in v'_1)$ (By definition of $REDUCEREP$).

Therefore, from Case 1 and Case 2, we conclude that $v'_1 = v'_2$.

Therefore, $< U'_1, v'_1 >=< U'_2, v'_2 >$, i.e., $REP(T) = REP(REDUCE(T))$ for any I-table T.

Theorem 3.1.4 For any I-tables $T_1 \in \Gamma_R$ and $T_2 \in \Gamma_R$,

$REP(T_1) = REP(T_2)$ if and only if $REDUCE(T_1) = REDUCE(T_2)$.

Proof:

(if) Let $REDUCE(T_1) = REDUCE(T_2)$.
Then, $REP(REDUCE(T_1)) = REP(REDUCE(T_2))$.
By Theorem 3.1.3, $REP(T_1) = REP(T_2)$.

(only if) Let $REDUCE(T_1) \neq REDUCE(T_2)$ and let
$REP(REDUCE(T_1)) =< U_1, v_1 >$ and $REP(REDUCE(T_2)) =< U_2, v_2 >$.

Case 1: $REDUCE(T_1)_D \neq REDUCE(T_2)_D$.
Clearly, in this case, $U_1 \cap U_2 = \emptyset$ and at least one of U_1 or U_2 is non-empty.
Therefore, $REP(REDUCE(T_1)) \neq REP(REDUCE(T_2))$.

Case 2: $REDUCE(T_1)_I \neq REDUCE(T_2)_I$.
Without loss of generality, it can be observed that there exists $r_1 \in U_1$ such that $r_1 \notin U_2$. Consequently,

$$REP(REDUCE(T_1)) \neq REP(REDUCE(T_2)).$$

Case 3: $REDUCE(T_1)_M \neq REDUCE(T_2)_M$.

Without loss of generality, there must exist $t \in REDUCE(T_1)_M$ such that $t \notin REDUCE(T_2)_M$.

Since $t \in REDUCE(T_1)_M$, $t \in v_1$.

Since $t \notin REDUCE(T_2)_M$, $t \notin v_2$.

Therefore, $v_1 \neq v_2$ and hence

$$REP(REDUCE(T_1)) \neq REP(REDUCE(T_2)).$$

Therefore, from Case 1, Case 2, Case 3, and Theorem 3.1.3, we conclude that $REP(T_1) \neq REP(T_2)$.

Theorem 3.2.1 $\sigma_F(< U,v >) = \sigma_F(REDUCEREP(< U,v >))$, for any $< U,v > \in \Sigma_R$.

Proof: Let $U = U_\alpha \cup U_\beta$ and $U_\alpha \cap U_\beta = \emptyset$ such that

(1) $(\forall r_1)(r_1 \in U_\beta \rightarrow (\exists r_2)(r_2 \in U_\alpha \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha \wedge r_2 \subset r_1))$.

Also let $\sigma_F(< U,v >) =< U_1,v_1 >$, $REDUCEREP(< U,v >) =< U',v' >$. and $\sigma_F(REDUCEREP(< U,v >) =< U_2,v_2 >$. Consider $r_i \in U_\beta$. By (1), there exists $r_j \in U_\alpha$ such that $r_j \subset r_i$. Clearly, $\sigma_F(r_j) \subseteq \sigma_F(r_i)$, and hence by the definition of $REDUCEREP$, $U_1 = U_2$. Also,

$$t \in (\sigma_F(r_i) - \sigma_F(r_j)) \vee t \in \sigma_F(v) \text{ if and only if } t \in \sigma_F(v').$$

Therefore, by the definition of $REDUCEREP$, $v_1 = v_2$.

Theorem 3.2.2 $\sigma_F(T) = \sigma_F(REDUCE(T))$ for any I-table T and selection formula F.

Proof: Let T be an I-table such that $T_{sure} = T_\alpha \cup T_\beta$, $T_\alpha \cap T_\beta = \emptyset$,

(1) $(\forall w_1)(w_1 \in T_\beta \rightarrow (\exists w_2)(w_2 \in T_\alpha \wedge w_2 \subset w_1))$, and

(2) $(\forall w_1)(w_1 \in T_\alpha \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha \wedge w_2 \subset w_1))$.

Let $T_1 = \sigma_F(T)$ and $T_2 = \sigma_F(REDUCE(T))$.

i) Consider $w_j \in T_\beta$, such that all the tuples in w_j satisfy F and let $w_i \subset w_j$ for some $w_i \in T_\alpha$. Surely, all tuples in w_i also satisfy F. Therefore, by definition of $REDUCE$ $w_j \notin T^2_{sure}$ and hence $T^1_{sure} = T^2_{sure}$.

ii) Recall that $F(t)$ is F with attribute number i replaced by $t[i]$. $t \in T^1_{maybe}$ if and only if

1) $((t \in T_{maybe}) \wedge F(t) \wedge \neg(\exists w)(w \in T^1_{sure} \wedge t \in w))$ or

2) $((\exists w)(w \in T_{sure} \wedge t \in w \wedge F(t) \wedge \neg(\exists w)(w \in T^1_{sure} \wedge t \in w)))$.

Case 1 : $(t \in T_{maybe}) \wedge F(t) \wedge \neg(\exists w)(w \in T^1_{sure} \wedge t \in w)$
iff $(t \in REDUCE(T)_{maybe}) \wedge F(t)$ (By definition of $REDUCE$)
iff $(t \in T^2_{maybe})$ (By definition of σ_F).

Case 2 : $(\exists w)(w \in T_{sure} \wedge t \in w \wedge F(t) \wedge \neg(\exists w)(w \in T^1_{sure} \wedge t \in w))$
iff $(\exists w)(w \in REDUCE(T)_{sure} \wedge t \in w \wedge F(t) \wedge \neg(\exists w)(w \in T^2_{sure} \wedge t \in w))$
(By definition of $REDUCE$)
iff $(t \in T^2_{maybe})$ (By definition of σ_F).

From Case 1 and Case 2, we conclude that $T^1_{maybe} = T^2_{maybe}$.

From, i) and ii), we conclude that $\sigma_F(T) = \sigma_F(REDUCE(T))$.

Theorem 3.2.3 $\sigma_F(REP(T)) = REP(\sigma_F(T))$, for any reduced I-table T.

Proof: Let T be a reduced I-table such that $T_{sure} = T_\alpha \cup T_\beta$, $T_\alpha \cap T_\beta = \emptyset$,

(1) All tuples in $w_i \in T_\alpha$ satisfy F, and

(2) Not all tuples in $w_i \in T_\beta$ satisfy F.

Let $\sigma_F^0(< MM, M > (T)) =< U_1, v_1 >$ and $< MM, M > (\sigma_F(T)) =< U_2, v_2 >$. Since (1) and (2), $U_2 \subseteq U_1$ and for every $r_i \in U_1 - U_2$, there exists $r_j \in U_2$ such that $r_j \subset r_i$. Also, $t \in (r_i - r_j) \vee t \in v_1$ if and only if $t \in v_2$. Therefore, by the definition of $REDUCEREP$,

$$REDUCEREP(< U_1, v_1 >) = REDUCEREP(< U_2, v_2 >),$$

i.e., $REDUCEREP(\sigma_F^0(< MM, M > (T)))$
$= REDUCEREP(< MM, M > (\sigma_F(T)))$,
i.e., $\sigma_F(< MM, M > (T)) = REP(\sigma_F(T))$.
Therefore, by Theorem 3.2.1,

$$\sigma_F(REDUCEREP(< MM, M > (T))) = REP(\sigma_F(T)),$$

i.e., $\sigma_F(REP(T)) = REP(\sigma_F(T))$.

Corollary 3.2.1 $\sigma_F(REP(T)) = REP(\sigma_F(T))$, for any I-table T.

Proof: Let T be any I-table and let $T_1 = REDUCE(T)$. Then, by Theorem 3.2.3,

$$\sigma_F(REP(T_1)) = REP(\sigma_F(T_1)),$$

i.e., $\sigma_F(REP(REDUCE(T))) = REP(\sigma_F(REDUCE(T)))$.
By Theorem 3.1.3,

$$\sigma_F(REP(T)) = REP(\sigma_F(REDUCE(T)))$$

and by Theorem 3.2.2,

$$\sigma_F(REP(T)) = REP(\sigma_F(T)).$$

Theorem 3.2.4 $\Pi_A(< U,v >) = \Pi_A(REDUCEREP(< U,v >))$, for any $< U,v > \in \Sigma_R$.

Proof: Let $U = U_\alpha \cup U_\beta$ and $U_\alpha \cap U_\beta = \emptyset$ such that

(1) $(\forall r_1)(r_1 \in U_\beta \rightarrow (\exists r_2)(r_2 \in U_\alpha \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha \wedge r_2 \subset r_1))$.

Also let $\Pi_A(< U,v >) =< U_1,v_1 >$, $REDUCEREP(< U,v >) =< U',v' >$, and $\Pi_A(REDUCEREP(< U,v >)) =< U_2,v_2 >$. Consider, $r_i \in U_\beta$. By (1), there exists $r_j \in U_\alpha$ and $r_j \subset r_i$. Clearly, $\Pi_A(r_j) \subset \Pi_A(r_i)$. Therefore, by the definition of $REDUCEREP$, $U_1 = U_2$. Also,

$$t \in (\Pi_A(r_i) - \Pi_A(r_j)) \vee t \in v_1 \text{ iff } t \in v_2.$$

Therefore, by the definition of $REDUCEREP$, $v_1 = v_2$.

Theorem 3.2.5 $\Pi_A(T) = \Pi_A(REDUCE(T))$ for any I-table T and attribute list A.

Proof: Let T be an I-table such that $T_{sure} = T_\alpha \cup T_\beta$, $T_\alpha \cap T_\beta = \emptyset$,

(a) $(\forall w_1)(w_1 \in T_\beta \rightarrow (\exists w_2)(w_2 \in T_\alpha \wedge w_2 \subset w_1))$, and

(b) $(\forall w_1)(w_1 \in T_\alpha \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha \wedge w_2 \subset w_1))$.

Let $T_1 = \Pi_A(T)$ and $T_2 = \Pi_A(REDUCE(T))$.

(i) Consider $w_j \in T_\beta$ and let $w_i \subset w_j$ for some $w_i \in T_\alpha$. Clearly, $\Pi_A(w_i) \subset \Pi_A(w_j)$. Therefore, by the definition of $REDUCE$ $T^1_{sure} = T^2_{sure}$.

(ii) $t \in T^1_{maybe}$
iff $(\exists t_1)(t_1 \in T_{maybe} \wedge t = \Pi_A(t_1) \wedge \neg(\exists w)(w \in T_{sure} \wedge t_1 \in w))$ (By definition of Π_A)

iff $(\exists t_1)(t_1 \in REDUCE(T)_{maybe} \wedge t = \Pi_A(t_1))$ (By definition of $REDUCE$)
iff $t \in T^2_{maybe}$ (By definition of Π_A)
Therefore, $T^1_{maybe} = T^2_{maybe}$.

Therefore, from (i) and (ii) we conclude that $\Pi_A(T) = \Pi_A(REDUCE(T))$.

Theorem 3.2.6 $\Pi_A(REP(T)) = REP(\Pi_A(T))$, for any reduced I-table T.

Proof: Let T be a reduced I-table and let $T_{sure} = T_\alpha \cup T_\beta$ and $T_\alpha \cap T_\beta = \emptyset$ such that

(1) $(\forall w_1)(w_1 \in T_\beta \rightarrow (\exists w_2)(w_2 \in T_\alpha \wedge \Pi_A(w_2) \subseteq \Pi_A(w_1)))$, and

(2) $(\forall w_1)(w_1 \in T_\alpha \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha \wedge \Pi_A(w_2) \subseteq \Pi_A(w_1)))$.

Let $< MM, M > (T) =< U', v' >$, $\Pi^0_A(< MM, M > (T)) =< U_1, v_1 >$, and $< MM, M > (\Pi_A(T)) =< U_2, v_2 >$. Since (1) and (2), $U' = U'_\alpha \cup U'_\beta$ and $U'_\alpha \cap U'_\beta = \emptyset$ such that

(3) $(\forall r_1)(r_1 \in U'_\beta \rightarrow (\exists r_2)(r_2 \in U'_\alpha \wedge \Pi_A(r_2) \subset \Pi_A(r_1)))$, and

(4) $(\forall r_1)(r_1 \in U'_\alpha \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha \wedge \Pi_A(r_2) \subset \Pi_A(r_1)))$.

Also,

(5) $(\forall r_1)(r_1 \in U_2 \rightarrow (\exists r_2)(r_2 \in U'_\alpha \wedge r_1 = \Pi_A(r_2)))$, and

(6) $(\forall r_1)(r_1 \in U'_\alpha \rightarrow (\exists r_2)(r_2 \in U_2 \wedge r_2 = \Pi_A(r_1)))$.

From (3), for any $r_i \in U'_\beta$ there exists $r_j \in U'_\alpha$ and $\Pi_A(r_j) \subset r_i$. Also

$$t \in \Pi_A(r_i) - \Pi_A(r_j) \vee t \in v_1 \text{ if and only if } t \in v_2.$$

Therefore, by the definition of $REDUCEREP$,
$REDUCEREP(< U_1, v_1 >) = REDUCEREP(< U_2, v_2 >)$,

i.e., $REDUCEREP(< \Pi^0_A(< MM, M > (T))) = REDUCEREP(< MM, M > (\Pi_A(T)))$,

i.e., $\Pi_A(< MM, M > (T)) = REP(\Pi_A(T))$.

Therefore, by Theorem 3.2.4,

$\Pi_A(REDUCEREP(< MM, M > (T))) = REP(\Pi_A(T))$,

i.e., $\Pi_A(REP(T)) = REP(\Pi_A(T))$.

Corollary 3.2.2 $\Pi_A(REP(T)) = REP(\Pi_A(T))$, for any I-table T.

Proof: Let T be any I-table and let $T_1 = REDUCE(T)$. Then, by Theorem 3.2.6,

$$\Pi_A(REP(T_1)) = REP(\Pi_A(T_1)),$$

i.e., $\Pi_A(REP(REDUCE(T))) = REP(\Pi_A(REDUCE(T)))$. By Theorem 3.1.3,

$$\Pi_A(REP(T)) = REP(\Pi_A(REDUCE(T)))$$

and by Theorem 3.2.5,

$$\Pi_A(REP(T)) = REP(\Pi_A(T)).$$

Theorem 3.2.7 For any $< U_1, v_1 > \in \Sigma_{R_1}$ and $< U_2, v_2 > \in \Sigma_{R_2}$,

$$< U_1, v_1 > \times < U_2, v_2 > = REDUCEREP(< U_1, v_1 >) \times REDUCEREP(< U_2, v_2 >.$$

Proof: Let $< U_1, v_1 > \times < U_2, v_2 > = < U_1^0, v_1^0 >$ and $REDUCEREP(< U_1, v_1 >) \times REDUCEREP(< U_2, v_2 >) = < U_2^0, v_2^0 >$. Also let $U_1 = U_\alpha^1 \cup U_\beta^1$ and $U_\alpha^1 \cap U_\beta^1 = \emptyset$ such that

(1) $(\forall r_1)(r_1 \in U_\beta^1 \rightarrow (\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha^1 \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$,

and $U_2 = U_\alpha^2 \cup U_\beta^2$ and $U_\alpha^2 \cap U_\beta^2 = \emptyset$ such that

(3) $(\forall r_1)(r_1 \in U^2_\beta \rightarrow (\exists r_2)(r_2 \in U^2_\alpha \wedge r_2 \subset r_1))$, and

(4) $(\forall r_1)(r_1 \in U^2_\alpha \rightarrow \neg(\exists r_2)(r_2 \in U^2_\alpha \wedge r_2 \subset r_1))$.

Now consider $r_j \in U^1_\beta$. By (1), there exists $r_i \in U^1_\alpha$ such that $r_i \subset r_j$. Therefore, $r_i \times r_e \subset r_j \times r_e$ for any $r_e \in U_2$. Similarly, consider $r_j \in U^2_\beta$. By (3), there exists $r_i \in U^2_\alpha$ such that $r_i \subset r_j$. Therefore, $r_i \times r_e \subset r_j \times r_e$ for any $r_e \in U_1$. Hence, $U^0_1 = U^0_2$. Using Lemma 3.1.1, we can conclude that $v^0_1 = v^0_2$.

Theorem 3.2.8 $T_1 \times T_2 = REDUCE(T_1) \times REDUCE(T_2)$, for any two I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be two I-tables. Then,

$REP(T_1 \times T_2)$

$= REP(T_1) \times REP(T_2)$, by Theorem 3.2.9.

$= REP(REDUCE(T_1)) \times REP(REDUCE(T_2))$, by Theorem 3.1.3

$= REP(REDUCE(T_1) \times REDUCE(T_2))$, by Theorem 3.2.9.

Therefore, by Theorem 3.1.4, we conclude that

$$T_1 \times T_2 = REDUCE(T_1) \times REDUCE(T_2).$$

Theorem 3.2.9 $REP(T_1) \times REP(T_2) = REP(T_1 \times T_2)$, for any two reduced I-tables T_1 and T_2.

Proof: Follows from the definition of $T_1 \times T_2$.

Corollary 3.2.3 $REP(T_1) \times REP(T_2) = REP(T_1 \times T_2)$, for any two I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be any two I-tables and let $T^0_1 = REDUCE(T_1)$ and $T^0_2 = REDUCE(T_2)$. Then, by Theorem 3.2.9, $REP(T^0_1) \times REP(T^0_2) = REP(T^0_1 \times T^0_2)$, i.e., $REP(REDUCE(T_1)) \times REP(REDUCE(T_2)) = REP(REDUCE(T_1) \times REDUCE(T_2))$. By Theorem 3.1.3, $REP(T_1) \times REP(T_2) = REP(REDUCE(T_1) \times REDUCE(T_2))$ and by Theorem 3.2.8, $REP(T_1) \times REP(T_2) = REP(T_1 \times T_2)$.

Theorem 3.2.10 $< U_1, v_1 > \cup < U_2, v_2 >= REDUCEREP(< U_1, v_1 >) \cup REDUCEREP(< U_2, v_2 >$, for any $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$.

Proof: Let $U_1 = U_\alpha^1 \cup U_\beta^1$ and $U_\alpha^1 \cap U_\beta^1 = \emptyset$ such that

(1) $(\forall r_1)(r_1 \in U_\beta^1 \rightarrow (\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha^1 \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$.

and let $U_2 = U_\alpha^2 \cup U_\beta^2$ and $U_\alpha^2 \cap U_\beta^2 = \emptyset$ such that

(3) $(\forall r_1)(r_1 \in U_\beta^2 \rightarrow (\exists r_2)(r_2 \in U_\alpha^2 \wedge r_2 \subset r_1))$, and

(4) $(\forall r_1)(r_1 \in U_\alpha^2 \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha^2 \wedge r_2 \subset r_1))$.

Consider $r_i \in U_\beta^1$. By (1), there exists $r_j \in U_\alpha^1$ and $r_j \subset r_i$. Therefore, for any $r_e \in U_2$, $r_j \cup r_e \subset r_i \cup r_e$. Similarly, consider $r_i \in U_\beta^2$. By (3), there exists $r_j \in U_\alpha^2$ and $r_j \subset r_i$. Therefore, for any $r_e \in U_1$ $r_j \cup r_e \subset r_i \cup r_e$. Therefore, by definition of $REDUCEREP$ and Lemma 3.1.1, $< U_1, v_1 > \cup < U_2, v_2 > = REDUCEREP(< U_1, v_1 >) \cup REDUCEREP(< U_2, v_2 >)$.

Theorem 3.2.11 $T_1 \cup T_2 = REDUCE(T_1) \cup REDUCE(T_2)$, for any two domain-compatible I-tables T_1 and T_2.

Proof: Since the extended union operator has $REDUCE$ built into it, we can easily observe that $T_1 \cup T_2 = REDUCE(T_1) \cup REDUCE(T_2)$.

Theorem 3.2.12 $REP(T_1) \cup REP(T_2) = REP(T_1 \cup T_2)$, for any two domain-compatible reduced I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be two domain-compatible reduced I-tables. Also, let $T_{sure}^1 = T_\alpha^1 \cup T_\beta^1$, $T_\alpha^1 \cap T_\beta^1 = \emptyset$, $T_{sure}^2 = T_\alpha^2 \cup T_\beta^2$, and $T_\alpha^2 \cap T_\beta^2 = \emptyset$ such that

(1) $(\forall w_1)(w_1 \in T_\beta^1 \rightarrow (\exists w_2)(w_2 \in T_\alpha^2 \wedge w_2 \subset w_1))$,

(2) $(\forall w_1)(w_1 \in T_\alpha^1 \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha^2 \wedge w_2 \subset w_1))$,

(3) $(\forall w_1)(w_1 \in T^2_\beta \rightarrow (\exists w_2)(w_2 \in T^1_\alpha \wedge w_2 \subset w_1))$, and

(4) $(\forall w_1)(w_1 \in T^2_\alpha \rightarrow \neg(\exists w_2)(w_2 \in T^1_\alpha \wedge w_2 \subset w_1))$.

Let $< MM, M > (T_1) \cup^0 < MM, M > (T_2) =< U_1, v_1 >$, and $< MM, M > (T_1 \cup T_2) =< U_2, v_2 >$. By (1), (2), (3), and (4), $U_2 \subseteq U_1$ and for each $r_i \in U_1 - U_2$, there exists $r_j \in U_2$ and $r_j \subset r_i$. Also, by Lemma 3.1.2,

$$t \in (r_i - r_j) \vee t \in v_1 \text{ if and only if } t \in v_2.$$

Therefore, by definition of *REDUCEREP*,

$$REDUCEREP(< U_1, v_1 >) = REDUCEREP(< U_2, v_2 >),$$

i.e., $REDUCEREP(< MM, M > (T_1) \cup^0 < MM, M > (T_2)) = REDUCEREP(< MM, M > (T_1 \cup T_2))$,
i.e., $< MM, M > (T_1) \cup < MM, M > (T_2) = REP(T_1 \cup T_2)$.
Therefore, by Theorem 3.2.10,

$$REDUCEREP(< MM, M > (T_1)) \cup REDUCEREP(< MM, M > (T_2)) = REP(T_1 \cup T_2),$$

i.e., $REP(T_1) \cup REP(T_2) = REP(T_1 \cup T_2)$.

Corollary 3.2.4 $REP(T_1) \cup REP(T_2) = REP(T_1 \cup T_2)$, for any two domain-compatible I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be any two domain-compatible I-tables and let $T^0_1 = REDUCE(T_1)$ and $T^0_2 = REDUCE(T_2)$. Then, by Theorem 3.2.12, $REP(T^0_1) \cup REP(T^0_2) = REP(T^0_1 \cup T^0_2)$, i.e., $REP(REDUCE(T_1)) \cup REP(REDUCE(T_2)) = REP(REDUCE(T_1) \cup REDUCE(T_2))$. By Theorem 3.1.3, $REP(T_1) \cup REP(T_2) = REP(REDUCE(T_1) \cup REDUCE(T_2))$ and by Theorem 3.2.11, $REP(T_1) \cup REP(T_2) = REP(T_1 \cup T_2)$.

Theorem 3.2.13 $< U_1, v_1 > - < U_2, v_2 >= REDUCEREP(< U_1, v_1 >) - REDUCEREP(< U_2, v_2 >$, for any $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$.

Proof: Let $U_1 = U_\alpha^1 \cup U_\beta^1$ and $U_\alpha^1 \cap U_\beta^1 = \emptyset$ such that

(1) $(\forall r_1)(r_1 \in U_\beta^1 \rightarrow (\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha^1 \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$.

Also let

$$REDUCEREP(< U_1, v_1 >) =< U_1^0, v_1^0 >,$$
$$REDUCEREP(< U_2, v_2 >) =< U_2^0, v_2^0 >,$$

$< U_1, v_1 > - < U_2, v_2 >=< U, v >$, and $< U_1^0, v_1^0 > - < U_2^0, v_2^0 >=< U^0, v^0 >$. By Lemma 3.1.1,

$$\bigcup_{r \in U_2} (r) \cup v_2 = \bigcup_{r \in U_2^0} (r) \cup v_2^0.$$

Let $r_{rhs} = \bigcup_{r \in U_2} (r) \cup v_2$. Consider $r_i \in U_\beta^1$. By (1), there exists $r_j \in U_\alpha^1$ and $r_j \subset r_i$. Clearly, $(r_j - r_{rhs}) \subseteq (r_i - r_{rhs})$. Therefore, by the definition of $REDUCREP$, $U = U^0$. Since $\bigcap_{r \in U_2} (r) = \bigcap_{r \in U_2^0} (r)$, by definition of $-^0$ and $REDUCEREP$, $v = v^0$.

Theorem 3.2.14 $T_1 - T_2 = REDUCE(T_1) - REDUCE(T_2)$, for any two domain-compatible I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be two domain-compatible I-tables such that $T_{sure}^1 = T_\alpha^1 \cup T_\beta^1$ and $T_\alpha^1 \cap T_\beta^1 = \emptyset$ such that

(1) $(\forall w_1)(w_1 \in T_\beta^1 \rightarrow (\exists w_2)(w_2 \in T_\alpha^1 \wedge w_2 \subset w_1))$, and

(2) $(\forall w_1)(w_1 \in T_\alpha^1 \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha^1 \wedge w_2 \subset w_1))$.

(i) Consider $w_j \in T^1_\beta$ and let $w_i \subset w_j$ for some $w_i \in T^1_\alpha$. Let w_j not have any common elements with any tuple set of T^2_{sure} or with T^2_{maybe}. Since, $w_i \subset w_j$, w_i also does not have any common elements with any tuple set of T^2_{sure} or with T^2_{maybe}. Therefore, $w_j \notin (T_1 - T_2)_{sure}$. Using Lemma 3.1.2, we conclude that $(T_1 - T_2)_{sure} = (REDUCE(T_1) - REDUCE(T_2))_{sure}$.

(ii) Since $T^2_D = (REDUCE(T_2))_D$, by Lemma 3.1.2 we conclude that $(T_1 - T_2)_{maybe}$ $= (REDUCE(T_1) - REDUCE(T_2))_{maybe}$.

Therefore, by (i) and (ii), we conclude that $T_1 - T_2 = REDUCE(T_1) - REDUCE(T_2)$.

Theorem 3.2.15 $REP(T_1) - REP(T_2) = REP(T_1 - T_2)$, for any two domain-compatible reduced I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be two domain-compatible reduced I-tables, $T = T_1 - T_2$, $< MM, M > (T_1 - T_2) =< U, v >$, $< MM, M > (T_1) =< U_1, v_1 >$, $< MM, M >$ $(T_2) =< U_2, v_2 >$, and $< MM, M > (T_1) -^0 < MM, M > (T_2) =< U', v' >$. Also let $T^1_{sure} = \{w_1, \ldots, w_k, w_{k+1}, \ldots, w_n\}$ such that

(1) w_i, $1 \leq i \leq k$, does not have any common tuples with any component of T_2, and

(2) w_i, $k+1 \leq i \leq n$, has common elements with T^2_D or with a tuple set of T^2_I or with T^2_M.

Therefore, by the definition of difference of I-tables, $T_{sure} = \{w_1, \ldots, w_k\}$. By (1) and (2), $U_1 = U^1_\alpha \cup U^1_\beta$ and $U^1_\alpha \cap U^1_\beta = \emptyset$ such that

(3) $r \in U^1_\alpha$ such that the tuples selected from w_i, $k+1 \leq i \leq n$, are the ones that are common with some component of T_2, and

(4) $r \in U^1_\beta$ such that the tuples selected from w_i, $k+1 \leq i \leq n$, are the ones that are not common with any component of T_2.

Therefore, by the definition of $-^0$, $U' = U'_\alpha \cup U'_\beta$, where

$U'_\alpha = \{r|(\exists r_1)(r_1 \in U^1_\alpha \wedge r = r_1 - (\bigcup_{r_2 \in U_2} (r_2) \cup v_2))\}$, and

$U'_\beta = \{r|(\exists r_1)(r_1 \in U^1_\beta \wedge r = r_1 - (\bigcup_{r_2 \in U_2} (r_2) \cup v_2))\}$.

Clearly, all the relations in U'_β are subsumed by some relations in U'_α. It can also be observed, from (3) and the definition of difference of I-tables that $U = U'_\alpha$. Also,

$t \in v$ if and only if $(t \in v') \vee (\exists r_1)(\exists r_2)(r_1 \in U'_\beta \wedge r_2 \in U'_\alpha \wedge r_2 \subset r_1 \wedge t \in r_1 - r_2)$.

Therefore, by the definition of $REDUCEREP$,

$$REDUCEREP(< U', v' >) = REDUCEREP(< U, v >),$$

i.e., $REDUCEREP(< MM, M > (T_1) -^0 < MM, M > (T_2)) = REDUCEREP(< MM, M > (T_1 - T_2))$,

i.e., $< MM, M > (T_1) - < MM, M > (T_2) = REP(T_1 - T_2)$.

Therefore, by Theorem 3.2.13

$$REDUCEREP(< MM, M > (T_1)) - REDUCEREP(< MM, M > (T_2)) = REP(T_1 - T_2),$$

i.e., $REP(T_1) - REP(T_2) = REP(T_1 - T_2)$.

Corollary 3.2.5 $REP(T_1) - REP(T_2) = REP(T_1 - T_2)$, for any two domain-compatible I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be any two domain-compatible I-tables and let $T^0_1 = REDUCE(T_1)$ and $T^0_2 = REDUCE(T_2)$. Then, by Theorem 3.2.15, $REP(T^0_1) - REP(T^0_2) = REP(T^0_1 - T^0_2)$, i.e., $REP(REDUCE(T_1)) - REP(REDUCE(T_2)) = REP(REDUCE(T_1) - REDUCE(T_2))$. By Theorem 3.1.3, $REP(T_1) - REP(T_2) = REP(REDUCE(T_1) - REDUCE(T_2))$ and by Theorem 3.2.14, $REP(T_1) - REP(T_2) = REP(T_1 - T_2)$.

Theorem 3.2.16 $< U_1, v_1 > \cap < U_2, v_2 >= REDUCEREP(< U_1, v_1 >) \cap REDUCEREP(< U_2, v_2 >$, for any $< U_1, v_1 > \in \Sigma_R$ and $< U_2, v_2 > \in \Sigma_R$.

Proof: Let $U_1 = U_\alpha^1 \cup U_\beta^1$ and $U_\alpha^1 \cap U_\beta^1 = \emptyset$ such that

(1) $(\forall r_1)(r_1 \in U_\beta^1 \rightarrow (\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha^1 \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha^1 \wedge r_2 \subset r_1))$,

and let $U_2 = U_\alpha^2 \cup U_\beta^2$ and $U_\alpha^2 \cap U_\beta^2 = \emptyset$ such that

(3) $(\forall r_1)(r_1 \in U_\beta^2 \rightarrow (\exists r_2)(r_2 \in U_\alpha^2 \wedge r_2 \subset r_1))$, and

(4) $(\forall r_1)(r_1 \in U_\alpha^2 \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha^2 \wedge r_2 \subset r_1))$.

Consider $r_i \in U_\beta^1$. By (1), there exists $r_j \in U_\alpha^1$ and $r_j \subset r_i$. Clearly, $r_j \cap r_e \subset r_i \cap r_e$ for any $r_e \in U_2$. Similarly, consider $r_i \in U_\beta^2$. By (3), there exists $r_j \in U_\alpha^2$ and $r_j \subset r_i$. Clearly, $r_e \cap r_j \subset r_e \cap r_i$ for any $r_e \in U_1$. Therefore, by the definition of *REDUCEREP* and Lemma 3.1.1, $< U_1, v_1 > \cap < U_2, v_2 >= REDUCEREP(< U_1, v_1 >) \cap REDUCEREP(< U_2, v_2 >)$.

Theorem 3.2.17 $T_1 \cap T_2 = REDUCE(T_1) \cap REDUCE(T_2)$, for any two domain-compatible I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be two domain-compatible I-tables such that $T_{sure}^1 = T_\alpha^1 \cup T_\beta^1$ and $T_\alpha^1 \cap T_\beta^1 = \emptyset$ such that

(1) $(\forall w_1)(w_1 \in T_\beta^1 - (\exists w_2)(w_2 \in T_\alpha^1 \wedge w_2 \subset w_1))$, and

(2) $(\forall w_1)(w_1 \in T_\alpha^1 - \neg(\exists w_2)(w_2 \in T_\alpha^1 \wedge w_2 \subset w_1))$,

and let $T_{sure}^2 = T_\alpha^2 \cup T_\beta^2$ and $T_\alpha^2 \cap T_\beta^2 = \emptyset$ such that

(3) $(\forall w_1)(w_1 \in T_\beta^2 \rightarrow (\exists w_2)(w_2 \in T_\alpha^2 \wedge w_2 \subset w_1))$, and

(4) $(\forall w_1)(w_1 \in T_\alpha^2 \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha^2 \wedge w_2 \subset w_1))$.

(i) Consider $w_j \in T^1_\beta$ and let $w_i \subset w_j$ for some $w_i \in T^1_\alpha$. Let $w_j \subseteq T^2_D$. Therefore, $w_i \subseteq T^2_D$ and hence $w_j \notin (T_1 \cap T_2)_{sure}$. Similarly, $w_j \notin (T_1 \cap T_2)_{sure}$, for $w_j \in T^2_\beta$. Therefore, $(T_1 \cap T_2)_{sure} = (REDUCE(T_1) \cap REDUCE(T_2))_{sure}$.

(ii) From (i) and Lemma 3.1.2, we can easily conclude that

$$(T_1 \cap T_2)_{maybe} = (REDUCE(T_1) \cap REDUCE(T_2))_{maybe}.$$

Therefore, from (i) and (ii), we conclude that

$$T_1 \cap T_2 = REDUCE(T_1) \cap REDUCE(T_2).$$

Theorem 3.2.18 $REP(T_1) \cap REP(T_2) = REP(T_1 \cap T_2)$, for any two domain-compatible reduced I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be two reduced domain-compatible I-tables such that $T^1_{sure} = \{w^1_1, \ldots, w^1_k, w^1_{k+1}, \ldots, w^1_n\}$ where

(1) $(\forall i)(1 \leq i \leq k \rightarrow w^1_i \subseteq T^2_D)$, and

(2) $(\forall i)(k+1 \leq i \leq n \rightarrow w^1_i \not\subseteq T^2_D)$,

and $T^2_{sure} = \{w^2_1, \ldots, w^2_l, w^2_{l+1}, \ldots, w^2_m\}$ where

(3) $(\forall i)(1 \leq i \leq l \rightarrow w^2_i \subseteq T^1_D)$, and

(4) $(\forall i)(l+1 \leq i \leq m \rightarrow w^2_i \not\subseteq T^1_D)$.

Also let $< MM, M > (T_1 \cap T_2) =< U', v' >$, $< MM, M > (T_1) =< U_1, v_1 >$, $< MM, M > (T_2) =< U_2, v_2 >$, and $< U_1, v_1 > \cap^0 < U_2, v_2 >=< U, v >$. Let $\{t^1_1, \ldots, t^1_k, t^2_1, \ldots, t^2_l\} \in U'$. Since (1), (2), (3), and (4), we have

$$\{t^1_1, \ldots, t^1_k, t^1_{k+1}, \ldots, t^1_n\} \in U_1, \text{ where}$$

$$(\forall i)(1 \leq i \leq k \rightarrow t_i^1 \notin T_D^2)$$
$$(\forall i)(k+1 \leq i \leq n \rightarrow t_i^1 \in T_D^2)$$

and $\{t_1^2, \ldots, t_l^2, t_{l+1}^2, \ldots, t_m^2\} \in U_2$ where

$$(\forall i)(1 \leq i \leq l \rightarrow t_i^2 \notin T_D^1)$$
$$(\forall i)(l+1 \leq i \leq m \rightarrow t_i^2 \in T_D^1)$$

Therefore, $\{t_1^1, \ldots, t_k^1, t_1^2, \ldots, t_l^2\} \in U$, and hence $U' \subseteq U$. Also, because of (1), (2), (3), and (4), for any $r_1 \in U - U'$, there exists $r_2 \in U'$ such that $r_2 \subset r_1$ and for any $t \in r_1 - r_2$, $t \in v'$. Therefore, by the definition of $REDUCEREP$,

$$REDUCEREP(< U, v >) = REDUCEREP(< U', v' >),$$

i.e., $REDUCEREP(< MM, M > (T_1) \cap^0 < MM, M > (T_2)) = REDUCEREP(< MM, M > (T_1 \cap T_2))$,

i.e., $< MM, M > (T_1) \cap < MM, M > (T_2) = REP(T_1 \cap T_2)$.

Therefore, by Theorem 3.2.16,

$$REDUCEREP(< MM, M > (T_1)) \cap REDUCEREP(< MM, M > (T_2)) = REP(T_1 \cap T_2),$$

i.e., $REP(T_1) \cap REP(T_2) = REP(T_1 \cap T_2)$.

Corollary 3.2.6 $REP(T_1) \cap REP(T_2) = REP(T_1 \cap T_2)$, for any two domain-compatible I-tables T_1 and T_2.

Proof: Let T_1 and T_2 be any two domain-compatible I-tables and let $T_1^0 = REDUCE(T_1)$ and $T_2^0 = REDUCE(T_2)$. Then, by Theorem 3.2.18,

$$REP(T_1^0) \cap REP(T_2^0) = REP(T_1^0 \cap T_2^0),$$

i.e., $REP(REDUCE(T_1)) \cap REP(REDUCE(T_2))$

$= REP(REDUCE(T_1) \cap REDUCE(T_2))$.

By Theorem 3.1.3,

$$REP(T_1) \cap REP(T_2) = REP(REDUCE(T_1) \cap REDUCE(T_2))$$

and by Theorem 3.2.17,

$$REP(T_1) \cap REP(T_2) = REP(T_1 \cap T_2).$$

Theorem 4.1.1 $\Pi_{<A_1,\ldots,A_n>}(< U,v >) = \Pi_{<A_1,\ldots,A_n>}(REDUCEREP(< U,v >))$, for any $< U,v > \in \Sigma_R$ and domain-compatible projection attribute lists $A_1,\ldots,A_n$.

Proof: Let $U = U_\alpha \cup U_\beta$ such that $U_\alpha \cap U_\beta = \emptyset$,

(1) $(\forall r_1)(r_1 \in U_\beta \rightarrow (\exists r_2)(r_2 \in U_\alpha \wedge r_2 \subset r_1))$, and

(2) $(\forall r_1)(r_1 \in U_\alpha \rightarrow \neg(\exists r_2)(r_2 \in U_\alpha \wedge r_2 \subset r_1))$.

Consider $r_1 \in U_\beta$. By (1), there exists $r_2 \in U_\alpha$ such that $r_2 \subset r_1$. Let $r_3 \in \Pi_{<A_1,\ldots,A_n>}(r_1)$. Since, $r_2 \subset r_1$, by definition of Π, there exists $r_4 \in \Pi_{<A_1,\ldots,A_n>}(r_2)$ such that $r_4 \subset r_3$. Also,

$$t \in r_3 - r_4 \text{ if and only if } (\exists t_1)(t_1 \in r_1 - r_2 \wedge t \in \{\Pi_{A_1}(t_1),\ldots,\Pi_{A_n}(t_1)\}.$$

Therefore, by the definition of $REDUCEREP$,

$$\Pi_{<A_1,\ldots,A_n>}(< U,v >) = \Pi_{<A_1,\ldots,A_n>}(REDUCEREP(< U,v >)).$$

Theorem 4.1.2 $\Pi_{<A_1,\ldots,A_n>}(T) = \Pi_{<A_1,\ldots,A_n>}(REDUCE(T))$, for any I-table T and domain-compatible projection attribute lists $A_1,\ldots,A_n$.

Proof: Let $T_{sure} = T_\alpha \cup T_\beta$ such that $T_\alpha \cap T_\beta = \emptyset$,

(1) $(\forall w_1)(w_1 \in T_\beta \rightarrow (\exists w_2)(w_2 \in T_\alpha \wedge w_2 \subset w_1))$, and

(2) $(\forall w_1)(w_1 \in T_\alpha \rightarrow \neg(\exists w_2)(w_2 \in T_\alpha \wedge w_2 \subset w_1))$.

Consider $w_1 \in T_\beta$. By (1), there exists $w_2 \in T_\alpha$ such that $w_2 \subset w_1$. Clearly, $\Pi_{<A_1,...,A_n>}(w_2) \subset \Pi_{<A_1,...,A_n>}(w_1)$. Also,

$$t \in \Pi_{<A_1,...,A_n>}(w_1) - \Pi_{<A_1,...,A_n>}(w_2) \text{ if and only if}$$
$$(\exists t_1)(t_1 \in w_1 - w_2 \wedge t \in \{\Pi_{A_1}(t_1), \ldots, \Pi_{A_n}(t_1)\}).$$

Therefore, by the definition of *REDUCEREP*,

$$\Pi_{<A_1,...,A_n>}(T) = \Pi_{<A_1,...,A_n>}(REDUCE(T)).$$

Theorem 4.1.3 $\Pi_{<A_1,...,A_n>}(REP(T)) = REP(\Pi_{<A_1,...,A_n>}(T))$, for any reduced I-table T and domain-compatible projection attribute lists $A_1, \ldots, A_n$.

Proof: Let T be a reduced I-table, where $T_{sure} = \{w_1, \ldots, w_e\}$ and let T_1 be $\Pi_{<A_1,...,A_n>}(T)$ without the *REDUCE* operator. Let $< MM, M > (T) =< U_1, v_1 >$ and $< MM, M > (T_1) =< U_2, v_2 >$. Consider any $r \in U_1$, where $r = \{t_1, \ldots, t_e\}$, where $t_i \in w_i$, $1 \leq i \leq e$. Let $U \subset U_2$ such that U consists of relations that are related to the tuples $t_1, \ldots, t_e$. By the definition of $< MM, M >$ and $\Pi_{<A_1,...,A_n>}(r)$, it can be observed that $\Pi_{<A_1,...,A_n>}(r) = U$. Therefore, by the definition of *REDUCEREP* and Theorem 3.1.3, we conclude that $\Pi_{<A_1,...,A_n>}(REP(T)) = REP(\Pi_{<A_1,...,A_n>}(T))$.

Corollary 4.1.1 $\Pi_{<A_1,...,A_n>}(REP(T)) = REP(\Pi_{<A_1,...,A_n>}(T))$, for any I-table T and domain-compatible projection attribute lists $A_1, \ldots, A_n$.

Proof: Let T be an I-table and let $T_1 = REDUCE(T)$. Then, by Theorem 4.1.3,

$$\Pi_{<A_1,...,A_n>}(REP(T_1)) = REP(\Pi_{<A_1,...,A_n>}(T_1)).$$

This is equivalent to:

$$\Pi_{<A_1,...,A_n>}(REP(REDUCE(T))) = REP(\Pi_{<A_1,...,A_n>}(REDUCE(T))).$$

Therefore, by Theorem 3.1.3 and Theorem 4.1.2,

$$\Pi_{<A_1,\ldots,A_n>}(REP(T)) = REP(\Pi_{<A_1,\ldots,A_n>}(T)).$$

Theorem 4.7.1 Any extended relational algebraic expression involving cartesian product, union, selection, and project-union is monotonic.

Proof: The semantic definition of weaker I-tables, the commutativity of the extended relational algebraic operators with REP, and the monotonicity of the regular algebraic operators allow us to conclude:

1. $(T_1 \leq T_2) \rightarrow (\sigma_F(T_1) \leq \sigma_F(T_2))$,
2. $(T_1 \leq T_2) \rightarrow (\Pi_{A_1,\ldots,A_n}(T_1) \leq \Pi_{A_1,\ldots,A_n}(T_2))$,
3. $(T_1 \leq T_2) \wedge (T_3 \leq T_4) \rightarrow (T_1 \times T_3) \leq (T_2 \times T_4)$
4. $(T_1 \leq T_2) \wedge (T_3 \leq T_4) \rightarrow (T_1 \cup T_3) \leq (T_2 \cup T_4)$

A simple induction on the number of operators in the extended relational algebraic expression allows us to conclude the theorem.