Chapter 10

QUERY SYSTEMS

To this point, we have seen two systems for manipulating relational databases. In Chapter 1 we saw update commands for adding, deleting, and modifying tuples in relations. In Chapters 2 and 3 we introduced the relational algebra to express selections, restrictions, and combinations of relations in a database. We call a formal system that can express updates to relations an *update system*. A *query* is a computation upon relations that yields other relations. A *query system*, such as the relational algebra, is a formal system for expressing queries. Query systems form the underlying structure of *query languages*: the special purpose programming languages used in database systems to formulate commands. We examine several query languages for relational database systems in Chapter 15.

In this chapter we cover three other query systems. The *tuple relational calculus* is essentially a formalization of the set-former notation we used to define the operators in relational algebra. *Domain relational calculus* is similar, except the variables range over single domain values rather than entire tuples. We shall see that both tuple calculus and domain calculus are equivalent in expressive power to relational algebra. We also introduce a modification of tableaux as a means to express queries. While tableaux cannot express all of the queries representable in relational algebra, the subclass they can represent seems to include many of the queries that might naturally arise in a real application. Furthermore, they lend themselves well to testing of equivalence and transformations, as we shall see in the next chapter.

Although a few query languages are based on relational algebra, most are based on either calculus or tableaux. The main reason is that the algebra is a procedural system, while the other three are non-procedural. That is, an expression in relational algebra gives a set of operations on relations and an order in which to perform them (up to certain associativities). We shall see that the calculi and tableaux simply express *what* the result of the computation should be, but *not how* to carry out the computation. Thus, query languages based on non-procedural systems tend to be higher-level, relieving the user of such languages from having to determine how to derive a desired

answer. The burden for this determination naturally falls to the query language processor of the given database system. In this chapter we shall show that expressions in either calculus can be translated effectively into algebraic expressions. However, the algebraic expressions we end up with can by no means be expected to be efficient means to evaluate the calculus expressions. In the next chapter we explore ways of modifying algebraic expressions to make them easier to evaluate.

We shall also, briefly, introduce conjunctive queries, which are a subclass of domain calculus expressions. Conjunctive queries are similar to tableaux queries, and also lend themselves well to equivalence testing and transformation.

10.1 EQUIVALENCE AND COMPLETENESS

The expressions in the various query systems we shall study can be viewed as mappings from databases to relations. That is, for an expression E, and a database d, we can evaluate E on d and get a particular relation r. We call r the *value* of expression E on d, and denote it $E(d)$.* We would like to say two expressions E_1 and E_2 are *equivalent*, written $E_1 \equiv E_2$, when $E_1(d) = E_2(d)$ for every database state d. The problem is that we must know the scheme of the database to decide equivalence. For example,

$$\pi_{AB}(r \bowtie s) \quad \text{and} \quad \pi_{AB}(r) \bowtie \pi_{AB}(s)$$

would be equivalent if we consider the schemes of r and s to be ABC and ABD, but not if the schemes were $ABCD$ and $ABCE$. We therefore consider equivalence to be relative to a particular database scheme. Sometimes the particular database scheme is immaterial, since the two expressions are equivalent for every database scheme where they are both properly formed. For example,

$$\sigma_{A=a}(r \bowtie s) \quad \text{and} \quad \sigma_{A=a}(r) \bowtie s$$

are equivalent for any database scheme where the relation scheme for r contains A.

The last example suggests a stronger notion of equivalence: $E_1 \equiv E_2$ if $E_1(d) = E_2(d)$ for every database d over every database scheme that is consistent with both E_1 and E_2. Unfortunately, this definition of equivalence is not transitive.

---

*In Chapter 3, before the introduction of databases, we were denoting $E(d)$ as $E(s_1, s_2, \ldots, s_k)$ where d is the database $\{s_1, s_2, \ldots, s_k\}$.

Example 10.1 Consider the following algebraic expressions:

$$E_1 = \pi_{AB}(r \cap s),$$
$$E_2 = \pi_{AB}(r) \cap s, \text{ and}$$
$$E_3 = \pi_{AB}(r) \cap \pi_{AB}(s).$$

E_1 and E_2 are equivalent under the alternative definition above. In any database consistent with E_1 and E_2, the schemes of r and s must both be AB. E_1 requires the two schemes be the same, where E_2 requires the scheme of s be AB. Similarly, E_2 and E_3 can be shown equivalent under the alternative definition. However, E_1 and E_3 are not equivalent, since there are databases consistent with both where the schemes of r and s are both ABC.

The situation can be worse than that given in Example 10.1. Under the alternative definition, there are expressions E_1, E_2, and E_3, with $E_1 \equiv E_2$ and $E_2 \equiv E_3$, where E_1 and E_3 do not even define relations over the same scheme for some mutually consistent databases (see Exercise 10.1). Hence, we shall always assume equivalence is relative to a fixed database scheme.

Once we define the other query systems, we shall discuss equivalence of expressions in different systems. One comparison we shall make between systems is *expressive power*. Query system QS_1 is *as expressive* as query systems QS_2 if for every expression E_2 of QS_2, and every database scheme compatible with E_2, there is an expression E_1 of QS_1 such that $E_1 \equiv E_2$. Note that E_1 may depend upon the particular database scheme. QS_1 and QS_2 are *equally expressive* if each is as expressive as the other. A query system is *complete* if it is as expressive as relational algebra. We shall see that tuple calculus and domain calculus are both complete, while tableau queries and conjunctive queries are not.*

In Chapter 3 we defined the relational algebra $\mathfrak{R}$ for a universe of attributes $\mathbf{U}$, with corresponding domains, a set of relations $\{r_1, r_2, \ldots, r_p\}$, and a set of binary comparators Θ, using constant relations and the operators union, intersection, difference, select, project, natural join, renaming, divide, theta-join, and active complement. The relational algebra with complement also allowed complement. However, we also saw, in Theorem 3.1, that for any relational algebra expression E, there is an equivalent expression E' using only single-attribute, single-tuple constant relations, renaming, select with a single comparison, projection, natural join, union, difference, and possibly complement (if the original expression used complement). The subalgebra of relational algebra using only the constants and operators above

*Traditionally, completeness has been defined as being as expressive as tuple calculus.

is equally expressive with the full relational algebra, and hence complete. Thus when we want to show that some query system QS is as expressive as relational algebra, we need only consider expressions using the subalgebra above. On the other hand, if we want to show relational algebra is as expressive as QS, we may use any of the relational operators, to simplify our task.

Although relational algebra is our benchmark for completeness, we shall see in Chapter 14 that there are some natural computations on relations that cannot be expressed by any algebraic expression (see Exercise 10.2).

10.2 TUPLE RELATIONAL CALCULUS

The tuple relational calculus should appear a natural notation to the reader, since it is quite similar to the set-former expressions used in Chapters 2 and 3 to define some of the operators in relational algebra. Where the relational algebra has relations as its basic units for manipulation, tuple relational calculus (tuple calculus, for short) builds its expressions from tuples.

Recall the definition of divide from Section 3.1. If $r(R)$ and $s(S)$ are relations, with $S \subseteq R$, and $R' = R - S$, then $r \div s$ is the relation

$$r'(R') = \{t \mid \text{for every tuple } t_s \in s \text{ there is a tuple } t_r \in r \text{ with } t_r(R') = t \text{ and } t_r(S) = t_s\}.$$

Tuple calculus expressions will have the form

$$\{x(R) \mid f(x)\},$$

where f is some Boolean predicate on tuple variable x. The expression denotes the relation $r(R)$ that consists of all tuples $t(R)$ where $f(t)$ is true. We shall shortly give a formal definition of the set of legal formulas, but first we give some informal examples.

Consider the database consisting of the three relations in Table 10.1. The database describes replacement parts for aircraft. Relation *pinfo* gives part numbers, other parts of which the part is an immediate subpart (not a subpart of a subpart) and the name of the part. Relation *usedon* gives the quantity of each part that is used on each type of aircraft. Relation *instock* gives the quantity of each part on hand at various repair locations. Some of the relations are incomplete. The value 0 for SUBPARTOF means the part is not a subpart.

Table 10.1 The relations *pinfo*, *usedon*, and *instock*.

pinfo(PART#	SUBPARTOF	PARTNAME)
211	0	coach seat
2114	211	seat cover
2116	211	seat belt
21163	2116	seat belt buckle
21164	2116	seat belt anchor
318	21164	funny little bolt
206	0	overhead console
2061	206	paging switch
2066	206	light switch
2068	206	air nozzle

usedon(PART#	PTYPE	NUSED)
211	707	86
211	727	134
2114	707	86
2114	727	134
2116	707	244
2116	727	296
21164	707	488
21164	727	592

instock(PART#	LOCATION	QUANTITY)
211	JFK	106
211	Boston	28
211	O'Hare	77
2114	JFK	6
2114	O'Hare	28
2116	Boston	341
2116	O'Hare	29
21164	Atlanta	36,391

Example 10.2 For the question "What are the subparts of part number 211," we might express the answer as

$$\{x(\text{PART\# PARTNAME}) \mid x \in \textit{pinfo} \text{ and } x(\text{SUBPARTOF}) = 211\}$$

The value of this expression is given in Table 10.2. The expression may not interpret the question correctly, if subparts of subparts, subparts of subparts of subparts, and so forth, were also meant to be included.

Table 10.2 Subparts of part 211.

(PART#	PARTNAME)
2114	seat cover
2116	seat belt

Example 10.3 The answer to the question "How many coach seats are used on a 727?" can be expressed as

$\{x(\text{NUSED}) | x \in \textit{usedon}$ and $x(\text{PTYPE}) = 727$ and there is a $y \in \textit{pinfo}$ where $x(\text{PART\#}) = y(\text{PART\#})$ and $y(\text{PARTNAME}) =$ "coach seat"$\}$

The value of this expression on the database in Table 10.1 is given in Table 10.3.

Table 10.3 Number of coach seats used on a 727.

(NUSED)
134

10.2.1 Tuple Calculus Formulas

The set of legal tuple calculus formulas will be defined relative to

1. A universal set of attributes **U**, with a domain, $dom(A)$, for each attribute A in **U**;
2. A set Θ of binary comparators on domains; and
3. A set d of relation names $\{r_1, r_2, \ldots, r_p\}$ on schemes $R_1, R_2, \ldots, R_p$, all subsets of **U**.

We first give the rules for building formulas, and then distinguish a subset of legal formulas according to a set of restrictions. We give the intuitive meaning of each formula as we go, but postpone the precise definition of the interpretation of a formula until after the set of legal formulas has been defined.

Tuple variables will generally be lower case letters from the very end of the alphabet, while we reserve t, u and v to stand for individual tuples.

The basic building blocks of formulas are *atoms*, of which there are three kinds:

a1. For any relation name r in d, and for any tuple variable x, $r(x)$ is an atom; $r(x)$ stands for $x \in r$.

a2. For any tuple variables x and y (not necessarily distinct), any comparator $\theta \in \Theta$, and any attributes A and B in $\mathbf{U}$ that are θ-comparable, $x(A)\ \theta\ y(B)$ is an atom.

a3. For any tuple variable x, any comparator $\theta \in \Theta$ and any attributes A and B in U that are θ-comparable, if c is a constant in $dom(A)$, then $c\ \theta\ x(B)$ is an atom; if c is a constant in $dom(B)$, then $x(A)\ \theta\ c$ is an atom.

Example 10.4 For the database of Table 10.1, some atoms are $pinfo(x)$, $x(\text{PART\#}) = y(\text{PART\#})$, and $x(\text{QUANTITY}) \leq 20$.

We use the connectives $\neg$(not), $\wedge$(and), $\vee$(or), $\exists$(there exists) and $\forall$(for all) to recursively build *formulas* from atoms, according to the following six rules. The formulas are similar to those of first-order predicate calculus using $r_1, r_2, \ldots, r_p$ as unary relation symbols.

f1. Any atom is a formula.

f2. If f is a formula, then $\neg f$ is a formula; $\neg f$ is true exactly when f is false.

f3. If f and g are formulas, then $f \wedge g$ and $f \vee g$ are formulas; $f \wedge g$ is true exactly when both f and g are true, $f \vee g$ is true when either f or g is true.

f4. If x is a tuple variable, f is a formula involving x, and R is a subset of $\mathbf{U}$, then $\exists x(R)f$ is a formula. That formula is true if there is some tuple t over R that makes f true when substituted for x in R.

f5. If x is a tuple variable, f is a formula involving x, and R is a subset of $\mathbf{U}$, then $\forall x(R)f$ is a formula. That formula is true if for every tuple t over R, f is true when t is substituted for x.

f6. If f is a formula, then (f) is a formula.

Parentheses are used to override the precedence of the connectives. We assume $\exists$ and $\forall$ are of highest and equal precedence, followed by $\neg$, $\wedge$, and $\vee$ in decreasing precedence.

Example 10.5 The atoms in Example 10.4 are all formulas by f1. By f2,

$$\neg x(\text{QUANTITY}) \leq 20$$

is a formula. By f3 and f6,

$$(x(\text{PART\#}) = y(\text{PART\#}) \vee \neg x(\text{QUANTITY}) \leq 20)$$

is a formula. By f3 again,

$$\mathit{instock}(x) \wedge (x(\text{PART\#}) = y(\text{PART\#}) \vee \neg x(\text{QUANTITY}) \leq 20)$$

is a formula. Finally, by f6 and f4,

$$\exists x(\text{PART\#}\ \text{LOCATION}\ \text{QUANTITY})\ (\mathit{instock}(x) \wedge (x(\text{PART\#}) = y(\text{PART\#}) \vee \neg x(\text{QUANTITY}) \leq 20))$$

is a formula. The parentheses added by f6 are necessary, for the unparenthesized formula

$$\exists x(\text{PART\#}\ \text{LOCATION}\ \text{QUANTITY})\ \mathit{instock}(x) \wedge x(\text{PART\#}) = y(\text{PART\#}) \vee \neg x(\text{QUANTITY}) \leq 20$$

is equivalent to

$$((\exists x(\text{PART\#}\ \text{LOCATION}\ \text{QUANTITY})\ \mathit{instock}\ (x)) \wedge x(\text{PART\#}) = y(\text{PART\#})) \vee x(\text{QUANTITY}) \leq 20$$

by the precedence of the connectives given.

10.2.2 Types, and Free and Bound Occurrences

Before we formally define the interpretation of a formula, we must be precise about what "f is a formula involving x" and "when t is substituted for x" mean. We also want to exclude certain nonsensical formulas, such as

$$\mathit{usedon}(x) \wedge x(\text{LOCATION}) = \text{"JFK"}$$

There is a typing problem with x, for $\mathit{usedon}(x)$ implies x is a tuple variable on PART# PTYPE NUSED, but x(LOCATION) implies a different scheme.

We shall define the *type* of a tuple variable x, that is, the scheme for x. We also define the *mention set* of x, which is the set of attributes x occurs with in a formula. We shall always want the mention set of x to be contained in the type of x. We also define when an occurrence of x is *free* or *bound* in a formula.

The idea of free and bound occurrences of tuple variables is analogous to global and local program variables in a language with nested procedure declaration.

Example 10.6 Consider the program sketched in Figure 10.1. Any mention of X, Y or Z in the body of MAIN refers to the variable declared in declaration 1. Any mention of Y or Z in the body of SUB1 also refers to declaration 1, while any mention of X or W refers to declaration 2. Y and Z are global to SUB1; they reference the same storage location at some procedure outside of SUB1. X and W are local to SUB1; they reference storage locations that are unseen by procedures outside of SUB1, although they can be seen by procedures inside SUB1. In the body of SUB12, X, Y and W are global, but Z is local. Note that in procedure SUB12, every occurrence of Z in declaration 3 and the body can be changed to another variable without changing the meaning of the program, as long as the new variable is not one that is global to SUB12. However, changing every occurrence of W in SUB12 could substantially alter the meaning of the program, since those occurrences of W are global to SUB12. Also note that *occurrences* of variable are global or local. An occurrence of Z in the body of SUB12 is local, while an occurrence in SUB1 is global.

```
        proc MAIN;
(1)       decl X, Y, Z;
          [body of MAIN]
              .
              .
              .
          proc SUB1;
(2)         decl X, W;
            [body of SUB1]
                .
                .
                .
            proc SUB12;
(3)           decl Z;
              [body of SUB12]
              end SUB12;
            end SUB1;
        end MAIN.
```

Figure 10.1

In a formula, free and bound variable occurrences correspond to global and local occurrences of variables in a program. The connectives $\exists$ and $\forall$, called *quantifiers*, correspond to declarations; they bind occurrences of variables in their scope. Quantifiers will also serve to type variables in our formulas, just as declarations can do in programs. We shall define free and bound occurrences recursively, along with $type(x, f)$, the type of variable x in formula f, and $men(x, f)$, the mention set of x in f. Both $type(x, f)$ and $men(x, f)$ are defined only when x has a free occurrence in f (x "occurs free" in f). We also use freedom, boundness, *type*, and *men* to define the class of *legal formulas*, through restrictions on when different connectives can be used.

First, consider the cases where f is an atomic formula.

a1. If f is $r(x)$, then x is free in f, and $type(x, f) = men(x, f) = R$, where R is the relation scheme for r.
a2. If f is $x(A)\ \theta\ y(B)$, then x and y are both free in f, $type(x, f)$ and $type(y, f)$ are both undefined, $men(x, f) = A$, and $men(y, f) = B$.
a3. If f is $x(A)\ \theta\ c$ or $c\ \theta\ x(A)$, then x is free in f, $type(x, f)$ is undefined, and $men(x, f) = A$.

Atomic formulas, as long as they obey the requirement on comparators and domains, are all legal. Next, consider the cases where f is built from smaller formulas. Assume g and h are both legal formulas.

f2. If $f = \neg g$, then f is legal, and all occurrences of variables in f are free or bound as they are in g. For every variable x that occurs free in f, $type(x, f) = type(x, g)$ and $men(x, f) = men(x, g)$.
f3. If $f = g \wedge h$ or $f = g \vee h$, then all occurrences of variables in f are free or bound as their corresponding occurrences are in g and h. For every variable x that occurs free in f, if $type(x, g)$ and $type(x, h)$ are both defined, they must be equal for f to be legal. If the type of x is defined for only one subformula, say $type(x, g)$, and x occurs free in h, then $type(x, g) \supseteq men(x, h)$ must hold for f to be legal. In either case, $type(x, f) = type(x, g)$. If the type of x is undefined for both subformulas, then $type(x, f)$ is undefined. In all cases, $men(x, f) = men(x, g) \cup men(x, h)$.
f4. If $f = \exists x(R)g$, then x must occur free in g for f to be legal. Furthermore, $type(x,g)$ must be R, if it is defined, and R must contain $men(x,g)$. All occurrences of x in f are bound; $type(x,f)$ and $men(x,f)$ are not defined, since x does not occur free in f. Any oc-

currence of a variable $y \neq x$ is free or bound in f as it was in g; $type(y,f) = type(y,g)$ and $men(y,f) = men(y,g)$.

f5. If $f = \forall x(R)g$, then all restrictions and definitions are the same as in f4.

f6. If $f = (g)$, then f is legal, and freedom, boundness, *type* and *men* are the same as for g.

In a formula such as $\exists x(R)g$ or $\forall x(R)g$, it is useful to distinguish which occurrences of x in g are actually bound by the quantifier. An occurrence of x in $\exists x(R)g$ is bound to $\exists x(R)$ if that occurrence of x is free in g. The same holds for $\exists$ replaced by $\forall$. If the occurrence of x in g is bound, then it must be bound to some quantifier contained in g.

The next five examples refer to the database given in Table 10.1. For these examples let

$$R_1 = \text{PART\# SUBPARTOF PARTNAME},$$
$$R_2 = \text{PART\# PTYPE NUSED, and}$$
$$R_3 = \text{PART\# LOCATION QUANTITY}.$$

Example 10.7 Let f be the formula

$$\forall x(R_3)(\neg\, instock(x) \vee x(\text{QUANTITY}) \leq 100).$$

All occurrences of x are bound; they are in the scope of $x(R_3)$. This formula is true if for every tuple t in *instock*, $t(\text{QUANTITY}) \leq 100$.

As shorthand notation, we shall use

$$\forall x(R) \in r\, f \quad \text{for} \quad \forall x(R)(\neg\, r(x) \vee f).$$

Similarly, we shall use

$$\exists x(R) \in r\, f \quad \text{for} \quad \exists x(R)(r(x) \wedge f).$$

Example 10.8 Let f be the formula

$$\forall x(R_3) \in instock(\exists y(R_3) \in instock$$
$$(x(\text{LOCATION}) = y(\text{LOCATION}) \wedge x(\text{PART\#}) = z(\text{PART\#}))).$$

All occurrences of x and y are bound. Each x is bound to $\forall x(R_3)$; each y is bound to $\forall y(R_3)$. The lone occurrence of z is free; $type(z,f)$ is undefined; $men(z,f) = \text{PART\#}$.

Example 10.9 Let f be the formula

$$\begin{aligned}&\exists x(R_3) \in instock(x(\text{LOCATION}) = \text{"JFK"} \wedge \\ &\quad \forall y(R_2) \in usedon((x(\text{PART\#}) \neq y(\text{PART\#}) \vee y(\text{PTYPE}) \neq \text{"747"}) \vee \\ &\qquad \exists x(R_3) \in instock(x(\text{PART\#}) = y(\text{PART\#}) \wedge x(\text{LOCATION}) \\ &\qquad\qquad = z(\text{LOCATION}))))\end{aligned}$$

(We use double quotes around domain values that come from non-numeric domains, so that values are not confused with variables. For consistency, we use the quotes even when the specific value is numeric.) Let $x_1, x_2, \ldots, x_6$ be the six occurrences of x in the order they occur in f. All occurrences of x are bound, however, x_1, x_2, and x_3 are bound to the first $\exists x(R_3)$, while x_4, x_5, and x_6 are bound to the second $\exists x(R_3)$. All occurrences of y are bound, but the lone occurrence of z is free. Note that the subformula

$$\begin{aligned}\exists x(R_3) \in instock(x(\text{PART\#}) &= y(\text{PART\#}) \wedge x(\text{LOCATION}) \\ &= z(\text{LOCATION}))\end{aligned}$$

could be changed to

$$\begin{aligned}\exists w(R_3) \in instock(w(\text{PART\#}) &= y(\text{PART\#}) \wedge w(\text{LOCATION}) \\ &= z(\text{LOCATION}))\end{aligned}$$

to make f easier to read.

Example 10.10 Let f be the formula

$$\exists x(R_2)\,(usedon(x) \wedge x(\text{LOCATION}) = y(\text{LOCATION})).$$

We see that for the subformulas

$$\begin{aligned}g &= usedon(x) \text{ and} \\ h &= (x(\text{LOCATION}) = y(\text{LOCATION})),\end{aligned}$$

$type(x,g) = R_2$, while $men(x,h) = \text{LOCATION}$. Hence the subformula $g \wedge h$ is not legal, so f is not legal.

Example 10.11 Let f be the formula

$$\exists x(R_2)\,(instock(x) \wedge x(\text{LOCATION}) = y(\text{LOCATION})).$$

for the subformula

$$g = (instock(x) \wedge x(\text{LOCATION}) = y(\text{LOCATION})).$$

$type(x,g) = R_3$, hence formula f is not legal.

Another shorthand notation, which did not arise in the examples, is

$$x(S) = y(S)$$

where S is a set of attributes $A_1 A_2 \cdots A_k$. It stands for the formula

$$(x(A_1) = y(A_1) \wedge x(A_2) = y(A_2) \wedge \cdots \wedge x(A_k) = y(A_k)).$$

When discussing formulas, we shall write $f(x_1, x_2, \ldots, x_n)$ to indicate that there are free occurrences of variables $x_1, x_2, \ldots, x_n$ in f. However, $f(x_1, x_2, \ldots, x_n)$ does not necessarily mean that $x_1, x_2, \ldots, x_n$ are the only variables that occur free in f.

10.2.3 Tuple Calculus Expressions

We denote a tuple calculus $\mathfrak{TC}$ as a sextuple

$$(\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta),$$

where $\mathbf{U}$ is the universe of attributes, $\mathfrak{D}$ is the set of domains, *dom* is a mapping from $\mathbf{U}$ to $\mathfrak{D}$, $\mathbf{R}$ is a set of relation schemes over $\mathbf{U}$, d is a database on the schemes in $\mathbf{R}$, and Θ is a set of comparators that includes at least equality and inequality for every domain in $\mathfrak{D}$. A *tuple calculus expression* over $\mathfrak{TC}$ has the form

$$\{x(R) | f(x)\},$$

where

1. f is a legal formula relative to $\mathbf{U}$, $\mathfrak{D}$, *dom*, $\mathbf{R}$, d and Θ.
2. x is the only tuple variable that occurs free in f.
3. R is a subset of $\mathbf{U}$.
4. If $type(x,f)$ is defined, it is equal to R, otherwise, $R \supseteq men(x,f)$.

By a slight manipulation of notation, we let $dom(R)$ stand for the set of all tuples with scheme R. To define the value for a tuple calculus expression, we need to substitute tuples for tuple variables.

Definition 10.1 Let $f(x)$ be a legal formula. Let R be $type(x,f)$, if $type(x,f)$ is defined, otherwise let R be any subset of U containing $men(x,f)$. Then f *with* t *substituted for* x, denoted $f(t/x)$, is the formula obtained by modifying each atom in f containing a free occurrence of x, as follows.

a1. If the x in $r(x)$ is free, replace $r(x)$ by *true* if $t \in r$, otherwise, replace $r(x)$ by *false*.

a2. If the x in $x(A)\ \theta\ y(B)$ is free, replace $x(A)$ by the constant $c \in dom(A)$ where $t(A) = c$, provided $x \neq y$. The same holds for $y(B)\ \theta\ x(A)$. If $x = y$, that is, the atom is actually $x(A)\ \theta\ x(B)$, replace the entire atom by *true* if $c_1\ \theta\ c_2$, where $c_1 = t(A)$ and $c_2 = t(B)$, otherwise, replace the atom by *false*.

a3. If the x in $x(A)\ \theta\ c$ is free, replace the entire atom by *true* if $c_1\ \theta\ c$, where $c_1 = t(A)$. Otherwise, replace the atom by *false*. Handle $c\ \theta\ x(A)$ in a similar manner.

Note we are slightly extending the definition of formula here to include the Boolean constants *true* and *false* as atoms. Under this extension, if $f(x)$ is a legal formula, so is $f(t/x)$ (see Exercise 10.4).

Example 10.12 Let $f(x)$ be the formula

$$\forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = x(\text{PART\#}) \\ \vee y(\text{QUANTITY}) \leq x(\text{QUANTITY}) \vee x(\text{LOCATION}) = \text{"JFK"})$$

If t is the tuple ⟨2114 O'Hare 28⟩ over scheme R_3 = PART# LOCATION QUANTITY, then $f(t/x)$ is

$$\forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = 2114 \\ \vee y(\text{QUANTITY}) \leq 28 \vee false).$$

Example 10.13 Let $f(y)$ be the formula

$$\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = 2114 \\ y(\text{QUANTITY}) \leq 28 \vee false.$$

If t is the tuple ⟨2116 Boston 341⟩ over scheme R_3, then $f(t/y)$ is

$$\neg\, \text{true} \vee \neg\, false \vee false \vee false.$$

If t = ⟨2114 JFK 6⟩, then $f(t/y)$ is

$$\neg\, true \vee \neg\, true \vee true \vee false.$$

Definition 10.2 Let f be a legal formula with no free tuple variables, but where *true* and *false* may appear as atoms. The *interpretation* of f, denoted $I(f)$, is defined recursively as follows.

f1. If f is *true*, then $I(f) = true$.
If f is *false*, then $I(f) = false$.
f2. If f is $\neg g$, then g must have no free variables. Let $I(f) = false$ if $I(g) = true$, otherwise let $I(f) = true$.
f3. If f is $g \wedge h$ or $g \vee h$, then neither g or h have free variables. If f is $g \wedge h$, let $I(f) = true$ exactly when $I(g) = I(h) = true$, otherwise, $I(f) = false$. If f is $g \vee h$, let $I(f) = false$ exactly when $I(g) = I(h) = false$, otherwise, $I(f) = true$.
f4. If f is $\exists x(R)g$, then x is the only variable that occurs free in g, (see Exercise 10.5). $I(f) = true$ if there is at least one tuple t in $dom(R)$ such that $I(g(t/x)) = true$, otherwise, $I(f) = false$.
f5. If f is $\forall x(R)g$, then x is the only variable that occurs free in g. $I(f) = true$ if for every tuple t in $dom(R)$, $I(g(t/x)) = true$, otherwise $I(f) = false$.
f6. If f is (g), then $I(f) = I(g)$.

Example 10.14 Let f be the formula

$$\exists x(R_3)\,(instock(x) \wedge x(\text{LOCATION}) = \text{"JFK"} \wedge \forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = x(\text{PART\#}) \vee y(\text{QUANTITY}) \leq x(\text{QUANTITY}))),$$

where R_3 = PART# LOCATION QUANTITY, as before. Intuitively, $I(f)$ is true if there is some part such that more of that part is stored at JFK than anywhere else. Let us compute $I(f)$ for the database of Table 10.1. Formula f has the form $\exists x(R_3)\, g(x)$, so we need to know if $I(g(t/x)) = true$ for some $t \in dom(R_3)$. Rather than trying all such tuples, we also note that $g(x)$ has the form $instock(x) \wedge g'(x)$, so we only need check tuples from $dom(R_3)$ that are in *instock*. (Using shorthand notation, f could be written $\exists x(R_3) \in instock\ g'(x)$.)

A little more inspection tells us that we only need try tuples in *instock* where the LOCATION-value is JFK. Let us first try the tuple $t = \langle 2114\ \text{JFK}\ 6\rangle$. We have

$$g(t/x) = (true \wedge true \wedge \forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = 2114 \vee y(\text{QUANTITY}) \leq 6)),$$

which simplifies to

$$\forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = 2114 \vee y(\text{QUANTITY}) \leq 6).$$

This formula has the form $\forall y(R_3)\, h(y)$, so we need to test whether $I(h(u/y)) = true$ for every tuple $u \in dom(R_3)$. Again, we can limit the search: since $h(y)$ has the form $\neg\, instock(y) \vee h'(y)$, we need only consider tuples in *instock*. Getting straight to the point, choosing $u = \langle 2114\ \text{O'Hare}\ 28\rangle$, we have

$$h(u/y) = (\neg\, true \vee \neg\, true \vee false).$$

Clearly $I(h(u/y)) = false$, so $I(g(t/x)) = false$.

We back up and try $t = \langle 211\ \text{JFK}\ 106\rangle$. Now

$$g(t/x) = (true \wedge true \wedge \forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = 211 \vee y(\text{QUANTITY}) \leq 106)),$$

which simplifies to

$$\forall y(R_3)\,(\neg\, instock(y) \vee \neg\, y(\text{PART\#}) = 211 \vee y(\text{QUANTITY}) \leq 106).$$

Again, we have a formula of the form $\forall y(R_3)\, h(y)$, so we have to check that $I(h(u/y)) = true$ for every tuple $u \in dom(R_3)$. As before, we need only test tuples in *instock*. Every choice for u makes $I(h(u/y)) = true$. For example, if $u = \langle 2116\ \text{O'Hare}\ 29\rangle$, we have

$$h(u/y) = (\neg\, true \vee \neg\, false \vee true),$$

so $I(h(u/y)) = true$. If $u = \langle 211\ \text{Boston}\ 28\rangle$, then

$$h(u/y) = (\neg\, true \vee \neg\, true \vee true),$$

so $I(h(u/y)) = true$. Hence $I(g(t/x)) = true$ and it follows $I(f) = true$. Note that $I(f) = false$ if $y(\text{QUANTITY}) \leq x(\text{QUANTITY})$ is changed to $y(\text{QUANTITY}) < x(\text{QUANTITY})$ (see Exercise 10.7).

We can now say what relation a tuple calculus expression defines.

Definition 10.3 Let $E = \{x(R)|f(x)\}$ be a tuple calculus expression over the tuple calculus $\mathfrak{TC} =$

$$(\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta).$$

The *value* of expression E on the current state of database d, denoted $E(d)$, is the relation r on scheme R containing every tuple $t \in dom(R)$ such that

$$I(f(t/x)) = true.$$

Example 10.15 The expression $E =$

$$\{x(\text{PART\# PARTNAME})|\exists y(R_1) \in pinfo\ (y(\text{SUBPARTOF}) = 211 \wedge x(\text{PART\# PARTNAME}) = y(\text{PART\# PARTNAME}))\}$$

is a formalization of the expression in Example 10.2, where $R_1 =$ PART# PARTNAME SUBPARTOF. Let $f(x)$ denote the formula in this expression. To evaluate $E(d)$ for d, the database in Table 10.1, we need to find every tuple $t \in dom(\text{PART\# PARTNAME})$ such that $I(f(t/x)) = true$. Inspection shows that we need only consider tuples in $\pi_{\text{PART\# PARTNAME}}(pinfo)$. Choosing $t = \langle 211 \text{ coach seat}\rangle$,

$$f(t/x) = \exists y(R_1) \in pinfo\ (y(\text{SUBPARTOF}) = 211 \wedge 211 = y(\text{PART\#}) \wedge \text{“coach seat”} = y(\text{PARTNAME})).$$

$I(f(t/x))$ will be true only if there is a tuple $\langle 211$ coach seat$\rangle$ in *pinfo*, which there is not. Thus, $\langle 211$ coach seat$\rangle$ is not in $E(d)$.

Choosing $t = \langle 2114$ seat cover$\rangle$,

$$f(t/x) = \exists y(R_1) \in pinfo\ (y(\text{SUBPARTOF}) = 211 \wedge 2114 = y(\text{PART\#}) \wedge \text{“seat cover”} = y(\text{PARTNAME})).$$

$I(f(t/x)) = true$ in this case, so 2114 seatcover is in $E(d)$. The only other choice of t that makes $I(f(t/x)) = true$ is $t = \langle 2116$ seat belt$\rangle$, so $E(d)$ is as given in Table 10.2.

Example 10.16 The expression in Example 10.3 can be formalized to

$$E = \{x(\text{NUSED}) | \exists z(R_2) \in \textit{usedon}\ (x(\text{NUSED}) = z(\text{NUSED}) \wedge z(\text{PTYPE}) = \text{"727"} \wedge \exists y(R_1) \in \textit{pinfo}\ (z(\text{PART\#}) = y(\text{PART\#}) \wedge y(\text{PARTNAME}) = \text{"coach seat"}))\},$$

where R_2 = PART# PTYPE NUSED and R_1 is as in the last example. $E(d)$, for the database given in Table 10.1, is given in Table 10.3. Note that in both this example and the last, an extra tuple variable was necessary going from the informal to the formal version of each expression. The extra variables handle the implicit projection in the informal versions.

Example 10.17 Consider the expression

$$E = \{x(R_2) | \neg \textit{usedon}(x) \vee x(\text{PTYPE}) \neq \text{"707"}\}.$$

where R_2 is as in the last example. If any of the domains for PART#, PTYPE and NUSED is infinite, then $E(d)$ will be an infinite relation, since there will be infinitely many tuples not in *usedon*. The *complement* of $E(d)$, for d the database in Table 10.1, is given in Table 10.4. In this case, but not always, the complement is finite (see Exercise 10.13).

Table 10.4 Complement of $E(d)$.

(PART#	PTYPE	NUSED)
211	727	134
2114	727	134
2116	727	296
21164	727	592

Example 10.18 Again refering to the database d in Table 10.1, consider the expression

$$E = \{x(\text{PARTNAME PTYPE}) | \exists y(R_1) \in \textit{pinfo}\ \exists z(R_2) \in \textit{usedon}\ (y(\text{PART\#}) = z(\text{PART\#}) \wedge x(\text{PARTNAME}) = y(\text{PARTNAME}) \wedge z(\text{NUSED}) > 200 \wedge x(\text{PTYPE}) = z(\text{PTYPE}))\},$$

where R_1 and R_2 are as given in previous examples. $E(d)$ gives all part names and plane types where more than 200 of the part are used on a plane of that type. $E(d)$ is shown in Table 10.5.

Table 10.5 Relation Between Part Name and Plane Type.

(PARTNAME	PTYPE)
seat belt	707
seat belt	727
seat belt anchor	707
seat belt anchor	727

10.3 REDUCING RELATIONAL ALGEBRA WITH COMPLEMENT TO TUPLE RELATIONAL CALCULUS

In this section we show that tuple calculus is as expressive as relational algebra with complement. We shall eventually show that they are equally expressive. In Section 10.4 we shall give an alternative interpretation for tuple calculus formulas. Under the alternative interpretation, tuple calculus and relational algebra without complement are equally expressive.

Theorem 10.1 Let $\mathfrak{R} = (\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta, O)$ be a relational algebra with complement and let $\mathfrak{TC} = (\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta)$ be a tuple calculus. For any algebraic expression E over $\mathfrak{R}$, there is an equivalent tuple calculus expression F over $\mathfrak{TC}$. That is, for any state of d, $E(d) = F(d)$.

Proof As we noted in Section 10, it is sufficient to assume E comes from the subalgebra $\mathfrak{R}'$ where O is replaced by the set of operations renaming, select with a single comparison, projection, natural join, union, difference, and complement, and where only single-attribute, single-tuple constant relations are allowed. The proof proceeds by induction on the number of operators in E.

Basis No operators. Then E is either a constant relation or a single relation from d. If $E = <a{:}A>$, then $F = \{x(A) | x(A) = a\}$. If $E = r$, where r is a relation on R, then $F = \{x(R) | r(x)\}$.

Induction Assume the theorem holds for any relational algebra expression with fewer than k operators. Let E have k operators.

> **Case 1** (renaming): $E = \delta_{A_1, A_2, \ldots, A_m \leftarrow B_1, B_2, \ldots, B_m}(E_1)$. E_1 has less than k operators. Let $\{x(R) | f(x)\}$ be a tuple calculus expression equivalent to E_1. Then F is

$$\{y(S) | \exists x(R)\,(f(x) \wedge g(x,y))\},$$

where $S = (R - A_1A_2 \cdots A_m)B_1B_2 \cdots B_m$ and $g(x, y)$ is the formula that is the "and" of the atoms $y(C) = x(C)$ for each C in $R - A_1A_2 \cdots A_m$ and $y(B_i) = x(A_i)$ for $1 \leq i \leq m$.

Case 2 (select): $E = \sigma_{A\theta c}(E_1)$ or $\sigma_{A\theta B}(E_1)$. Let $\{x(R) | f(x)\}$ be a tuple calculus expression equivalent to E_1. Then F is

$$\{x(R) | f(x) \wedge x(A)\,\theta\, c\}$$

or

$$\{x(R) | f(x) \wedge x(A)\,\theta\, x(B)\}.$$

Case 3 (projection): $E = \pi_X(E_1)$. This case is left to the reader (see Exercise 10.10).

Case 4 (join): $E = E_1 \bowtie E_2$. Let $\{x(QR) | f(x)\}$ be a tuple calculus expression for E_1 and let $\{y(RS) | g(y)\}$ be a tuple calculus expression for E_2, where $QR \cap RS = R$. Then F is

$$\{z(QRS) | \exists x(QR)\, \exists y(RS)\,(f(x) \wedge g(y) \wedge z(QR) = x(QR) \wedge z(RS) = y(RS)\}.$$

Case 5 (union): $E = E_1 \cup E_2$. This case is left to the reader (see Exercise 10.10).

Case 6 (difference): $E = E_1 - E_2$. This case is left to the reader (see Exercise 10.10).

Case 7 (complement): $E = \bar{E}_1$. Let $\{x(R) | f(x)\}$ be a tuple calculus expression for E_1. Then

$$F = \{x(R) | \neg f(x)\}.$$

Example 10.19 Consider the algebraic expression

$$E = \pi_{\text{SUBPARTOFNUSED}}(pinfo \bowtie \sigma_{\text{PTYPE} = 747}(usedon)).$$

for the database in Table 10.1. Equivalent tuple calculus expressions for *pinfo* and *usedon* are

$$\{x(R_1) | pinfo(x)\} \text{ and } \{y(R_2) | usedon(y)\},$$

where R_1 = PART# SUBPARTOF PARTNAME and R_2 = PART# PTYPE NUSED. For $\sigma_{\text{PTYPE} = 747}(usedon)$ we have

$$\{y(R_2) | usedon(y) \wedge y(\text{PTYPE}) = \text{"747"}\}.$$

Letting R = PART# SUBPARTOF PARTNAME PTYPE NUSED, an equivalent expression for $pinfo \bowtie \sigma_{\text{PTYPE} = 747}(usedon)$ is

$$\{z(R) | \exists x(R_1)\ \exists y(R_2)\ (pinfo(x) \wedge usedon(y) \wedge y(\text{PTYPE}) = \text{"747"} \wedge z(R_1) = x(R_1) \wedge z(R_2) = y(R_2)\}.$$

Finally, an equivalent tuple calculus expression for E is

$$\{w(\text{SUBPART NUSED}) | \exists z(R)\ (\exists x(R_1)\ \exists y(R_2)\ (pinfo(x) \wedge usedon(y) \wedge y(\text{PTYPE}) = \text{"747"} \wedge z(R_1) = x(R_1) \wedge z(R_2) = y(R_2)) \wedge w(\text{SUBPART}) = z(\text{SUBPART}) \wedge w(\text{NUSED}) = z(\text{NUSED}))\}.$$

10.4 LIMITED INTERPRETATION OF TUPLE CALCULUS FORMULAS

The interpretation given for tuple calculus formulas presents some practical problems when tuple calculus is considered as the basis for a query system. First, tuple calculus expressions can define infinite relations. Second, it is not clear that arbitrary formulas of the forms

$$\exists x(R) f(x) \quad \text{and} \quad \forall x(R) f(x)$$

can be effectively interpreted. The interpretation given would seem to require searching through all of $dom(R)$, which could be infinite. Even if all the attributes in R have finite domains, $dom(R)$ could be unmanageably large. We shall present an alternative interpretation for formulas where tuples are restricted to being composed of domain values that appear in a formula or in relations mentioned in a formula. The original interpretation shall be called *unlimited*, while the alternative interpretation shall be called *limited*. We

shall also introduce a class of tuple calculus expressions for which both interpretations always yield the same value.

For the following development, we assume, as in Chapter 3, that domains of different attributes are either equal or disjoint. This restriction is for simplification only. We could allow arbitrary intersection of domains, so long as there is a means to specify all the values of a given domain that appear anywhere in a relation.

Definition 10.4 Let f be a tuple calculus formula and let A be an attribute. The *extended active domain* of A relative to f, denoted $edom(A,f)$, is the set of all values from $dom(A)$ that appear in relations mentioned in f or as constants in f.

If no attribute with a domain the same as $dom(A)$ is mentioned in f, it is possible for $edom(A,f) = \emptyset$. For a set of attributes R, we let $edom(R,f)$ consist of all tuples t such that $t(A) \in edom(A,f)$ for every $A \in R$.

Example 10.20 Assume that the attributes PART# and SUBPARTOF, in the database of Table 10.1, have the same domain. Let f be the formula

$$\forall y(R_1) \in pinfo(217 \neq y(\text{SUBPARTOF}) \vee \exists z(R_3) \in instock\ (z(\text{PART\#}) = y(\text{PART\#}) \wedge z(\text{LOCATION}) = x(\text{LOCATION}) \wedge z(\text{QUANTITY}) > 0))),$$

where R_1 and R_3 are as in previous examples. This formula describes all the locations that stock all the subparts of part 217. Using the states of *pinfo* and *instock* in Table 10.1,

$$edom(\text{PART\#}, f) = edom(\text{SUBPARTOF}, f) = \{0, 206, 211, 217, 318, 2061, 2066, 2068, 2114, 2116, 21163, 21164\}.$$

There is an algebraic expression for $edom(\text{PART\#},f)$ that accounts for the current state of *pinfo* and *instock*. It is

$$\pi_{\text{PART\#}}(pinfo) \cup \pi_{\text{PART\#}}(instock) \cup <217{:}\text{PART\#}> \cup \delta_{\text{SUBPARTOF} \leftarrow \text{PART\#}}(\pi_{\text{SUBPARTOF}}(pinfo)).$$

By Theorem 10.1, there is also a tuple calculus expression for $edom(\text{PART\#},f)$. Note that if g is a subformula of a tuple calculus formula f, then $edom(A,g) \subseteq edom(A,f)$ for every attribute A.

We now give the *limited interpretation* of a tuple calculus formula f with no free variables. The limited interpretation of f is denoted $i(f)$, and the definition is the same as for $I(f)$, except for the cases f4 and f5 on $\exists$ and $\forall$. These cases are replaced by

f4′. If f is $\exists x(R)g$, then $i(f) = true$ if there is at least one tuple t in $edom(R,g)$ such that $i(g(t/x)) = true$. Otherwise, $i(f) = false$.

f5′. If f is $\forall x(R)g$, then $i(f) = true$ if for every tuple t in $edom(R,g)$ $i(g(t/x)) = true$. Otherwise, $i(f) = false$.

The *limited evaluation* of a tuple calculus expression $E = \{x(R)|f(x)\}$ is the relation on R consisting of every tuple $t \in edom(R,f)$ such that $i(f(t/x)) = true$.

The previous interpretation and evaluation shall be called *unlimited*. For Examples 10.15, 10.16 and 10.18, the unlimited and limited evaluations yield the same value. However, the value for the expression in Example 10.17 is a finite relation under limited evaluation. Limited evaluation always yields a finite relation, since, given finite relations to start with, $edom(R,f)$ is finite for any formula f. Also, the limited interpretation of formulas such as $\exists x(R)g$ and $\forall x(R)g$ is effective, since $i(g(t/x))$ need only be computed for tuples in $edom(R,g)$.

Example 10.21 The following expression over the database in Table 10.1 always yields the empty relation under limited evaluation:

$$\{x(\text{PART\#})|\forall y(R_3) \in instock\ x(\text{PART\#}) \neq y(\text{PART\#})\}.$$

Tuple variable x ranges only over parts that appear in *instock*.

Example 10.22 Assuming that $dom(\text{PART\#}) = dom(\text{SUBPARTOF})$ in the database of Table 10.1, the expression

$$E = \{x(\text{SUBPARTOF})|\forall y(R_1) \in pinfo\ x(\text{SUBPARTOF}) \neq y(\text{SUBPARTOF})\}$$

does not necessarily yield an empty relation under limited interpretation. Tuple variable x ranges over all parts that appear in either the PART# or SUBPARTOF columns of *pinfo*. If d, the current state of the database, is as given in Table 10.1, then $E(d)$ under limited evaluation is given in Table 10.6.

Table 10.6 $E(d)$ Under Limited Evaluation.

(PART#)
21163
2061
2066
2068

10.4.1 Reducing Relational Algebra to Tuple Calculus with Limited Evaluation

As we have noted, tuple calculus expressions denote only finite relations under limited evaluation. Algebraic expressions without complement denote only finite relations. This similarity is no coincidence.

Theorem 10.2 Let $\mathfrak{R} = (\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta, O)$ be a relational algebra (without complement), and $\mathfrak{IC} = (\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta)$ be a tuple calculus. For any algebraic expression E over $\mathfrak{R}$ there is a tuple calculus expression F over $\mathfrak{IC}$ such that E is equivalent to F under limited evaluation.

Proof The proof follows that of Theorem 10.1 with case 7 of the induction omitted. Every tuple calculus expression used in the rest of the proof has the same value under unlimited and limited interpretation. The details are left to the reader (see Exercise 10.18).

10.4.2 Safe Tuple Calculus Expressions

Tuple calculus under the limited evaluation, although as expressive as relational algebra, leaves something to be desired as the basis for a query language. The value of an expression can depend on columns in relations corresponding to attributes not even mentioned in the expression. Another approach to the problem of keeping values of expressions finite and having effective interpretation of formulas with quantifiers is to use only expressions where the unlimited and limited evaluations are guaranteed to be the same.

Definition 10.5 A tuple calculus expression $\{x(R)|f(x)\}$ is *safe* if the following three conditions hold.

s1. $I(f(t/x)) = true$ implies that $t \in edom(R,f)$.

s2. For every subformula of f of the form $\exists y(S)\ g(y, z_1, z_2, \ldots, z_k)$, $I(g(t/y, u_1/z_1, u_2/z_2, \ldots, u_k/z_k)) = true$ implies $t \in edom(S,g)$, where $y, z_1, z_2, \ldots, z_k$ are all the free tuple variables in g.

s3. For every subformula of f of the form $\forall y(S)\ g(y, z_1, z_2, \ldots, z_k)$, $t \notin edom(S,g)$ implies $I(g,(t/y, u_1/z_1, u_2/z_2, \ldots, u_k/z_k)) = true$ where $y, z_1, z_2, \ldots, z_k$ are all the free tuple variables in g.

Condition s1 guarantees that the value of the expression will be a finite relation under the unlimited evaluation. Conditions s2 and s3 guarantee that the unlimited interpretation of formulas can be made effective, since only a finite number of tuples need be considered to interpret $\exists y(S)g$ or $\forall y(S)g$. If every place a quantifier is used in formula f, it is involved in the shorthand $\exists y(S) \in s\ g(y)$ or $\forall y(S) \in s\ g(y)$, then conditions s2 and s3 will be satisfied. The notation $\exists y(S) \in s\ g(y)$ stands for $\exists y(S)\ h(y)$, where $h(y)$ is $s(y) \wedge g(y)$. Any tuple $t \in s$ is clearly in $edom(S,h)$, and the $s(y)$ atom in $h(y)$ means that $h(t/y)$ cannot interpret to *true* no matter what the other free variables in h are. Similarly, $\forall y(S) \in s\ g(y)$ stands for $\forall y(S)\ h(y)$, where $h(y)$ in this case is $\neg s(y) \vee g(y)$. Any tuple t not in $edom(S,h)$ is surely not in s, so $g(t/y)$ interprets to *true* for any such tuple.

If the formula $f(x)$ actually has the form $r(x) \wedge f'(x)$, for some relation r, then condition s1 is satisfied.

Example 10.23 By what we observed above, the expression

$$\{x(R_2) \mid usedon(x) \wedge \exists y(R_3) \in instock \\ (x(\text{PART\#}) = y(\text{PART\#}) \wedge y(\text{QUANTITY}) \geq 100)\}$$

on the database of Table 10.1 is safe. R_2 and R_3 are the relation schemes of *usedon* and *instock*, as before.

Example 10.24 The expression

$$\{x(R_2) \mid usedon(x) \vee \exists y(R_3) \in instock \\ (x(\text{PART\#}) = y(\text{PART\#}) \wedge y(\text{QUANTITY}) \geq 100)\}$$

is not safe. The unlimited evaluation of this expression for the database in Table 10.1 contains the tuple ⟨2116 707 83⟩, for example, which is not in the extended active domain of R_2 for the formula in the expression.

Example 10.25 The expression

$$\{x(\text{PARTNAME PTYPE}) \mid \exists y(R_1) \in \textit{pinfo}\ \exists z(R_2) \in \textit{usedon}$$
$$(y(\text{PART\#}) = z(\text{PART\#}) \wedge x(\text{PARTNAME}) = y(\text{PARTNAME})$$
$$\wedge\ x(\text{PTYPE}) = y(\text{PTYPE}))\}$$

is safe, even though there is no $r(x)$ term. Any tuple t in the value of the expression must have a PARTNAME-value that appears in *pinfo* and a PTYPE-value that appears in *usedon*. A class of safe expressions, of which this expression is a member, is described in Exercise 10.20.

Lemma 10.1 For any safe tuple calculus expression E, the unlimited and limited evaluations of E are the same.

Proof Left to the reader (see Exercise 10.21).

Theorem 10.3 Given any tuple calculus expression E, there is a safe tuple calculus expression F that is equivalent to E under limited evaluation for E.

Proof Exercise 10.22 shows that for any tuple calculus formula g there is another formula h such that the value of $\{y(X) \mid h(y)\}$ under unlimited evaluation is $\textit{edom}(X,g)$ for a given set of attributes S. Furthermore, $\textit{edom}(S,g) = \textit{edom}(S,h)$, so $\textit{edom}(S,g) = \textit{edom}(S,g \wedge h) = \textit{edom}(S,g \vee \neg h)$.

Given an expression $E = \{x(R) \mid f(x)\}$, we first want to find a formula f' such that $i(f(t/x)) = I(f'(t/x))$ for all $t \in \textit{dom}(R)$. The only place i and I diverge is on subformulas involving quantifiers. We produce f' by modifying such subformulas. For a subformula $\exists y(S)\ g(y)$, let $h(y)$ be as outlined above. Replace this subformula by $\exists y(S)\ (g(y) \wedge h(y))$. We have $i(g(t/y)) = I(g(t/y) \wedge h(t/y))$ for any $t \in \textit{edom}(S,g)$ and $I(g(t/y) \wedge h(t/y)) = \textit{false}$ for any $t \notin \textit{edom}(S,g)$, provided $i(g(t/y)) = I(g(t/y))$. Hence the modified subformula has an unlimited interpretation equal to the limited interpretation of the original subformula.

For a subformula $\forall y(S)\ g(y)$, let $h(y)$ be as above and substitute $\forall y(S)\ (g(y) \vee \neg h(y))$. Assuming $i(g(t/y)) = I(g(t/y))$, $i(g(t/y)) = I(g(t/y) \wedge \neg h(t/y))$ for any $t \in \textit{edom}(S,g)$ and $I(g(t/y) \vee \neg h(t/y)) = \textit{true}$ for any $t \notin \textit{edom}(S,g)$. Hence the modified subformula has an unlimited interpretation equal to the limited interpretation of the original subformula.

If we make these modifications to all such subformulas of f working from the inside out, the end result is a formula f' where $i(f(t/x)) = I(f'(t/x))$. Note that we do not modify subformulas in the h's, so the process does terminate. To complete the proof, let h' be a formula such that the unlimited evaluation of $\{x(R) \mid h'(x)\}$ is $\textit{edom}(R,f)$. The desired expression F is then

$$\{x(R) \mid f'(x) \wedge h'(x)\}.$$

10.5 DOMAIN RELATIONAL CALCULUS

Domain relational calculus is quite similar to tuple relational calculus, except variables represent single domain values rather than entire tuples. Also, relation symbols can now be multiplace rather than simply unary. To keep the notation reasonable, it is necessary to assume a fixed order on the attributes in a relation. Since tuple calculus and domain calculus are so similar, the treatment of domain calculus will be much briefer. We first give some informal examples, again using the database of Table 10.1.

Example 10.26 The expression

$$\{x\,y \mid pinfo(x\,z\,y) \wedge z = 206\}$$

represents all part number-part name pairs that are subparts of part 206. The expression

$$\{x\,y \mid pinfo(x\,206\,y)\}$$

represents the same pairs. The value of both these expressions is shown in Table 10.7.

Table 10.7 Partname-Partnumber Pairs.

(PART#	PARTNAME)
2061	paging switch
2066	light switch
2068	air nozzle

Example 10.27 The expression

$$\{x\,y \mid \exists z\ \exists w\ (usedon(z\,x\,y) \wedge pinfo(z\,w\ \text{“coach seat”})\}$$

represents the number of coach seats used on each plane type. The value of this expression is shown in Table 10.8.

Table 10.8 Number of Coach Seats Used.

(PTYPE	NUSED)
707	86
727	134

A domain calculus $\mathfrak{DC}$ is denoted the same way as a tuple calculus, namely as a sextuple $(\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta)$. *Domain calculus formulas* are built from domain variables using relations, comparators, and the connectives $\neg$, $\wedge$, $\vee$, $\exists$ and $\forall$. The basic building blocks are *atoms*:

a1. If r is a relation in d with scheme $A_1 A_2 \cdots A_n$, then $r(a_1 a_2 \cdots a_n)$ is an atom, where each a_i is either a domain variable or a constant from $dom(A_i)$.
a2. If x and y are domain variables, θ is a comparator and c is an appropriate constant, then $x \theta y$, $x \theta c$ and $c \theta x$ are all atoms.
a3. The Boolean constants *true* and *false* are atoms.

The atoms are combined recursively into formulas:

f1. Any atom is a formula.
f2. If f is a formula, then $\neg f$ is a formula.
f3. If f and g are formulas, so are $f \wedge g$ and $f \vee g$.
f4. If f is a formula, so is $\exists x(A) f$, where A is an attribute in $\mathbf{U}$ and x is a domain variable.
f5. If f is a formula, so is $\forall x(A) f$, where A is an attribute in $\mathbf{U}$ and x is a domain variable.
f6. If f is a formula, so is (f).

The precedence of connectives is the same as for tuple calculus formulas.

Example 10.28 The following domain calculus formula is similar to the tuple calculus formula of Example 10.5:

$$\exists x_1(\text{PART\#})\ \exists x_2(\text{LOCATION})\ \exists x_3(\text{QUANTITY})\ (\mathit{instock}(x_1 x_2 x_3) \wedge (x_1 = y \wedge \neg x_3 \leq 20)).$$

The rules for free and bound occurrences of variables are analogous to those for tuple calculus formulas. To discuss legal domain calculus formulas, we need to type domain variables. The type of a variable x in formula f, denoted $type(x,f)$, is either a domain in $\mathfrak{D}$ or undefined. We again assume for simplicity that the domains of two attributes are either equal or disjoint. We shall not formally define the type of a variable or legal formulas. As with tuple calculus, legality simply requires that there is consistency as to the type of a variable and that a quantified variable occur free in the quantified formula.

Example 10.29 Let $r(A\ B\ C)$ and $s(C\ D)$ be relations. For the formula

$$f = \exists x(E)\, \forall y(A)\, (r(y\ z\ x) \wedge s(x\ x) \wedge y \leq z)$$

to be legal, we must have $dom(C) = dom(D) = dom(E)$ and attributes A and B must be $\leq$-comparable. In that case, $type(z,f) = dom(B)$. If g is the subformula

$$(r(y\ z\ x) \wedge s(x\ x) \wedge y \leq z),$$

then $type(x,g) = dom(C)$ and $type(y,g) = dom(A)$.

When quantifying variables, we could use domains rather than attributes to type them. However, it will be useful to have attributes when reducing domain calculus to relational algebra.

The *substitution* of a domain constant c for a domain variable x that occurs free in a formula f, denoted $f(c/x)$, is analogous to substitution for tuple calculus formulas. We assume $c \in type(x,f)$. Every free occurrence of x in f is replaced by c, and then atoms composed entirely of constants are replaced by *true* and *false* as appropriate.

Example 10.30 Consider the formula

$$\begin{aligned} f \;=\; & usedon(x\ \text{"707"}\ y) \wedge \forall w(\text{QUANTITY}) \\ & (\neg\, instock(x\ \text{"JFK"}\ w) \vee w \geq y \vee x = 2116). \end{aligned}$$

Variable x is free in f; $f(211/x)$ is the formula

$$\begin{aligned} g \;=\; & usedon(211\ \text{"707"}\ y) \wedge \forall w(\text{QUANTITY}) \\ & (\neg\, instock(211\ \text{"JFK"}\ 2) \vee w \geq y \vee false). \end{aligned}$$

Variable y is free in g; $g(86/y)$ is

$$\begin{aligned} & true \wedge \forall w(\text{QUANTITY}) \\ & \quad (\neg\, instock(211\ \text{"JKF"}\ 2) \vee w \geq 86 \vee false), \end{aligned}$$

which simplifies to

$$\forall w(\text{QUANTITY})\, (\neg\, instock(211\ \text{"JFK"}\ 2) \vee w \geq 86).$$

The *unlimited interpretation* of a domain calculus formula f with no free occurrences of variables is denoted $I(f)$. The definition is essentially that for tuple calculus formulas. For a formula $f = \exists x(A)\, g(x)$, $I(f) = true$ if and only if there is a $c \in dom(A)$ such that $I(g(c/x)) = true$. For a formula $f = \forall x(A)\, g(x)$, $I(f) = true$ if and only if for every $c \in dom(A)$, $I(g(c/x)) = true$.

Example 10.31 Let h be the formula

$$\forall w(\text{QUANTITY})\,(\neg\, instock(211\ \text{"JFK"}\ w) \vee w \geq 86)$$

from the end of Example 10.30. To calculate $I(h)$, we need to know $I(h'(c/x))$ for every $c \in dom(\text{QUANTITY})$ where $h'(w)$ is the formula

$$\neg\, instock(211\ \text{"JFK"}\ w) \vee w \geq 86.$$

The first disjunct in $h'(w)$ interprets to *true* for all choices of w except 106, and the second disjunct interprets to *true* for every $c \in dom(\text{QUANTITY})$. Hence $I(h) = true$.

A *domain calculus expression* over $\mathfrak{DC}$ has the form

$$\{x_1(A_1)\, x_2(A_2) \cdots x_n(A_n) | f(x_1, x_2, \ldots, x_n)\}$$

where

1. f is a legal domain calculus formula with exactly the free variables x_1, $x_2, \ldots, x_n$,
2. $A_1, A_2, \ldots, A_n$ are distinct attributes in **U**, and
3. $type(x_i, f) = dom(A_i)$ for $1 \leq i \leq n$.

The *value* of this expression under *unlimited evaluation* is the relation over scheme $A_1\, A_2 \cdots A_n$ containing every tuple $\langle c_1\, c_2 \cdots c_n \rangle$ such that $c_i \in dom(A_i)$, for $1 \leq i \leq n$, and $I(f(c_1/x_1, c_2/x_2, \ldots, c_n/x_n)) = true$. As before, the value of expression E for database d is denoted $E(d)$.

Example 10.32 Using the formula from Example 10.30, we have the domain calculus expression

$$E = \{x(\text{PART\#})\, y(\text{NUSED}) | usedon(x\ \text{"707"}\ y) \wedge \forall w(\text{QUANTITY})\,(\neg\, instock(x\ \text{"JFK"}\ w) \vee w \geq y \vee x = 2116)\}.$$

E denotes the part number and number used for each part used on a 707 such that the quantity of that part *instock* at JFK is at least the number used or the part number is 2116. $E(d)$, under unlimited evaluation, for the state of the database d given in Table 10.1 is shown in Table 10.9.

Table 10.9 Part Number and Number Used.

(PART#	NUSED)
211	86
2116	244

As with tuple calculus, we can give a limited interpretation for formulas and limited evaluation for expressions. The *extended active domain* of an attribute A in a domain calculus formula f, denoted $edom(A,f)$, is the set of all elements of $dom(A)$ that occur as constants in f or in relations mentioned in f. The *limited interpretation* of a domain calculus formula f with no free variables, denoted $i(f)$, differs from the unlimited interpretation only for quantified formulas. If f is $\exists x(A)\, g(x)$, then $i(f) = true$ if and only if there is a constant $c \in edom(A,g)$ such that $I(g(c/x)) = true$. Similarly, if f is $\forall x(A)\, g(x)$, then $i(f) = true$ if for every constant $c \in edom(A,f)\, I(g(c/x)) = true$. To get the limited evaluation of an expression

$$\{x_1(A_1)\, x_2(A_2) \cdots x_n(A_n) | f(x_1, x_2, \ldots, x_n)\},$$

x_i ranges over $edom(A_i,f)$, for $1 \leq i \leq n$, and limited interpretation is used for f.

Example 10.33 Consider the expression

$$\{x(\text{PART\#}) | \forall y(\text{PART\#})\, \forall z(\text{PARTNAME})\, \neg\, pinfo(y\ x\ z) \wedge x \neq 318\}$$

for the database of Table 10.1, where we assume $dom(\text{PART\#}) = dom(\text{SUBPARTOF}) =$ non-negative integers. The unlimited evaluation of this expression yields an infinite relation. The value under limited interpretation is given in Table 10.10.

We can also define the class of *safe* domain calculus expressions. An expression

$$\{x_1(A_1)\, x_2(A_2) \cdots x_n(A_n) | f(x_1, x_2, \ldots, x_n)\}$$

is safe if the following three conditions hold.

s1. If for constants $c_1, c_2, \ldots, c_n$ $I(f(c_1/x_1, c_2/x_2, \ldots, c_n/x_n)) = true$, then $c_i \in edom(A_i, f)$ for $1 \leq i \leq n$.
s2. For each subformula of f of the form $\exists y(A)\, g(y)$, $I(g(c/y)) = true$ implies $c \in dom(A, g)$.
s3. For each subformula of f of the form $\forall y(A)\, g(y)$, $c \notin edom(A, g)$ implies $I(g(c/y)) = true$.

These three conditions serve the same purpose as the corresponding conditions for safe type calculus expressions.

Table 10.10 Value under Limited Interpretation.

(PART#)
21163
2061
2066
2068

Example 10.34 The expression in Example 10.32 is safe. Both x and y must appear in *usedon*, and $\neg$ *instock*(x "JFK" 2) $\vee$ $w \geq y \vee x = 2116$ interprets to *true* if w does not appear in *instock*. The expression in Example 10.33 is not safe. There are values for x that do not appear in *pinfo* that make the formula true.

The following two results are analogs of Lemma 10.1 and Theorem 10.3, and are stated without proof.

Lemma 10.2 For any safe domain calculus expression E, the unlimited and limited evaluations of E are the same.

Theorem 10.4 Given any domain calculus expression E, there is a safe domain calculus expression F that is equivalent to E under limited evaluation for E.

10.6 REDUCTION OF TUPLE CALCULUS TO DOMAIN CALCULUS

As we saw in Example 10.9, any tuple calculus formula can be modified to an equivalent formula where the same tuple variable is not bound in two places. For this section, we assume that in a tuple calculus expression

$$\{x(R) | f(x)\},$$

x does not occur bound in f, nor is any other tuple variable bound in more than one place.

The translation of a tuple calculus expression E to an equivalent domain calculus expression is straightforward. Let $E = \{x(R)|f(x)\}$. Any tuple variable y appearing in f is associated with a unique relation scheme S. Either y appears in $\exists y(S)$ or $\forall y(S)$, or $y = x$ and $S = R$. Let $S = A_1\, A_2 \cdots A_k$. Tuple variable y is replaced by k domain variables $y_1, y_2, \ldots y_k$. Any atom $r(y)$ becomes $r(y_1 y_2 \cdots y_k)$. Any atom $y(A_i)\, \theta\, a$ or $a\, \theta\, y(A_i)$, for a a constant or another tuple variable component, becomes $y_i\, \theta\, a$ or $a\, \theta\, y_i$, respectively. A quantified subformula $\exists y(S)\, g$ becomes

$$\exists y_1(A_1)\, \exists y_2(A_2) \cdots \exists y_k(A_k)\, g,$$

while $\forall y(S)\, g$ becomes

$$\forall y_1(A_1)\, \forall y_2(A_2) \cdots \forall y_k(A_k)\, g.$$

If $y = x$ and $S = R$, then $x(R)$ at the beginning of the expression is replaced by

$$y_1(A_1) y_2(A_2) \cdots y_k(A_k).$$

Example 10.35 The tuple calculus expression

$$\begin{aligned} E = \{&x(\text{NUSED})|\exists z(R_2)\,(\mathit{usedon}(z) \wedge x(\text{NUSED}) = z(\text{NUSED}) \wedge \\ &z(\text{PTYPE}) = \text{“727”} \wedge y(R_1)\,(\mathit{pinfo}(y) \wedge \\ &z(\text{PART\#}) = y(\text{PART\#}) \wedge y(\text{PARTNAME}) = \text{“coach seat”}))\} \end{aligned}$$

is that of Example 10.16 with the shorthand notation written out. We replace x by x_1, z by z_1, z_2, z_3, and y by y_1, y_2, y_3 to get the domain calculus expression

$$\begin{aligned} F = \{&x_1(\text{NUSED})|\exists z_1(\text{PART\#})\, \exists z_2(\text{PTYPE})\, \exists z_3(\text{NUSED}) \\ &(\mathit{usedon}(z_1\, z_2\, z_3) \wedge x_1 = z_3 \wedge z_2 = \text{“727”} \wedge \\ &\exists y_1(\text{PART\#})\, \exists y_2(\text{SUBPARTOF})\, \exists y_3(\text{PARTNAME}) \\ &(\mathit{pinfo}(y_1\, y_2\, y_3) \wedge z_1 = y_1 \wedge y_3 = \text{“coach seat”}))\}. \end{aligned}$$

F can be simplified to

$$\begin{aligned} \{&x_1(\text{NUSED})|\exists z_1(\text{PART\#})\,(\mathit{usedon}(z_1\, \text{“727”}\, x_1) \\ &\wedge \exists y_2(\text{SUBPARTOF})\,(\mathit{pinfo}(z_1\, y_2\, \text{“coach seat”})))\}. \end{aligned}$$

Theorem 10.5 Let E be a tuple calculus expression and let F be the domain calculus expression obtained from F by the translation given above. Then

1. $E \equiv F$ under unlimited evaluation,
2. $E \equiv F$ under limited evaluation, and
3. If E is safe, then F is safe.

Proof 1. Left to the reader (see Exercise 10.27).

2. The equivalence follows from part 1 and the observation that for any attribute A, if g is a subformula of E and g' is the corresponding subformula of F, $edom(A,g) = edom(A,g')$. The equality follows since g and g' mention exactly the same relations and constants.
3. Any place the formation $\exists y_i(A_i)$ occurs in F, it is actually part of a subformula

$$\exists y_1(A_1)\, \exists y_2(A_2) \cdots \exists y_k(A_k)\, g'(y_1, y_2, \ldots, y_k)$$

that was the translation of a subformula $\exists y(S)\, g(y)$ in E, where $S = A_1\, A_2 \cdots A_k$. Let $c_i \in dom(A_i)$ for $1 \leq i \leq k$, and suppose $I(g'(c_1/y_1, c_2/y_2, \ldots, c_k/y_k)) = true$. By the correspondence of g and g', $I(g(t/y)) = true$ for t the tuple $c_1\, c_2 \cdots c_k$. If E is safe, then $t \in edom(R,g)$. It follows that $c_i \in edom(A_i,g')$. Hence condition s2 of the definition of a safe domain calculus expression is satisified for F. Conditions s1 and s3 can similarly be shown to be satisfied if E is safe.

10.7 REDUCTION OF DOMAIN CALCULUS TO RELATIONAL ALGEBRA

In this section we show that relational algebra with complement is as expressive as domain calculus with unlimited evaluation, and also that relational algebra (without complement) is as expressive as domain calculus with limited evaluation. These results, combined with those of Sections 10.3, 10.4.1 and 10.6, show that relational algebra with complement and tuple and domain calculus under unlimited evaluation are equally expressive. Relational algebra and tuple and domain calculus under limited evaluation are also equally expressive, hence the tuple and domain calculi under limited evaluation are complete.

Theorem 10.6 Let E be an expression over a domain calculus $\mathfrak{DC} = (\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta)$. Let $\mathfrak{R} = (\mathbf{U}, \mathfrak{D}, dom, \mathbf{R}, d, \Theta, O)$ be a relational algebra with

complement. There is an algebraic expression F over $\mathfrak{R}$ that is equivalent to E under unlimited evaluation, provided there are sufficient attributes in $\mathbf{U}$ for each domain in $\mathfrak{D}$.

Proof Let $E = \{x_1(A_1)\, x_2(A_2) \cdots x_n(A_n) | f(x_1, x_2, \ldots, x_n)\}$.
For each subexpression g of f we shall find an algebraic expression F_g for

$$\{y_1(B_1) y_2(B_2) \cdots y_m(B_m) | g(y_1, y_2, \ldots, y_m)\}$$

where $y_1, y_2, \ldots, y_m$ are all the free domain variables in g and the B_i's are chosen to have appropriate domains.

First, replace domain variables in f so that no variable is bound in two places or occurs both free and bound in f. Next, note that every variable is associated with an attribute, either by a quantifier ($\exists x(A)$ or $\forall x(A)$), or by appearing to the left of the bar in the expression E ($x(A)$). For each domain variable x, if A is the attribute associated with x, let $att(x)$ be an attribute B such that $dom(A) = dom(B)$. Furthermore, let $att(x)$ be chosen such that for any domain variable $y \neq x$, $att(y) \neq att(x)$. It is at this step that the requirement for a sufficient number of attributes for each domain arises.

For any attribute A, there is an algebraic expression for $dom(A)$. One such expression is

$$\langle c{:}A \rangle \cup \langle \overline{c{:}A} \rangle,$$

where c is an arbitrary element of $dom(A)$. In what follows, $[A]$ will stand for an algebraic expression for $dom(A)$.

We now recursively define the algebraic expression F_g for each subformula g of f. F_g will be equivalent to the domain calculus expression

$$\{y_1(att(y_1)) y_2(att(y_2)) \cdots y_m(att(y_m)) | g(y_1, y_2, \ldots, y_m)\}.$$

Case 1 Subformula g is an atom of the form $x\, \theta\, y$, $x\, \theta\, x$, $x\, \theta\, c$, or $c\, \theta\, x$, for x and y domain variables and c a domain constant. Let $A = att(x)$ and $B = att(y)$. The algebraic expressions for these atoms are

$$\sigma_{A\theta B}([A] \bowtie [B]),$$
$$\sigma_{A\theta A}([A]),$$
$$\sigma_{A\theta c}([A]), \text{ and}$$
$$\sigma_{c\theta A}([A]), \text{ respectively.}$$

Note the join in the first instance is necessarily a Cartesian product.

Case 2 Subformula g is an atom of the form $r(a_1\ a_2\ \cdots\ a_k)$ where a_i is either a constant or a domain variable. Let $D_1\ D_2\ \cdots\ D_k$ be the relation scheme for r. Let $B_i = att(a_i)$ if a_i is a variable; let $B_i = D_i$ otherwise. The algebraic expression F_g is

$$\pi_X(\delta_N(\sigma_C(r)))$$

where

C is the selection condition made up of the "and" of comparisons $D_i = a_i$ for each a_i that is a constant,
N is the renaming $D_1,D_2, \ldots, D_k \leftarrow B_1,B_2, \ldots, B_k$, and
X is the set of attributes $\{B_i | a_i \text{ is a variable}\}$.

Case 3 Subformula g is $\neg h$. Let F_h be the algebraic expression for h. Then $F_g = \bar{F}_h$.

Case 4 Subformula g is $h \wedge h'$. Let h have free variables $z_1,z_2, \ldots, z_k$, $v_1,v_2, \ldots, v_p$ and let h' have free variables $z_1,z_2, \ldots, z_k$, $w_1,w_2, \ldots, w_q$, where $v_1,v_2, \ldots, v_p$ and $w_1,w_2, \ldots, w_q$ are distinct. F_h and F_h' are the algebraic expressions for h and h'. Let $att(v_i) = B_i$, $1 \leq i \leq p$, and $att(w_i) = C_i$, $1 \leq i \leq q$. Let

$$F_1 = F_h \bowtie [C_1] \bowtie [C_2] \bowtie \cdots \bowtie [C_q]$$

and let

$$F_2 = F_h' \bowtie [B_1] \bowtie [B_2] \bowtie \cdots \bowtie [B_p].$$

F_1 is F_h with columns added for all the attributes in $sch(F_h') - sch(F_h) = C_1C_2 \cdots C_q$. F_2 is F_h' with columns added for all the attributes in $sch(F_h) - sch(F_h') = B_1B_2 \ldots B_p$. Note that $sch(F_1) = sch(F_2) = \{att(y) | y \text{ is free in } g\}$. The algebraic expression F_g is $F_1 \cap F_2$.

Case 5 Subformula g is $h \vee h'$. This case is left to the reader (see Exercise 10.28).

Case 6 Subformula g has the form $\exists x(A)\ h$. F_h is the algebraic expression for h. Let $att(x) = B$. Note that $dom(A) = dom(B)$. The algebraic expression F_g for g is simply $\pi_{X-B}(F_h)$ where $X = sch(F_h)$.

Case 7 Subformula g is $\forall x(A)\ h$. F_h is the algebraic expression for h. Let $att(x) = B$. The algebraic expression F_g for g is $F_h \div [B]$. (This one case is the only reason we ever bothered with division.)

Recursively, using these seven cases, we can build up an expression F_f that is equivalent to

$$\{x_1(att(x_1))\, x_2(att(x_2)) \cdots x_n(att(x_n)) | f(x_1, x_2, \ldots, x_n)\}.$$

We are not assured that $att(x_i)$ is necessarily A_i, so we need one final renaming operation to actually obtain the algebraic expression F equivalent to E.

Example 10.36 $E =$

$$\{x(\text{PART\#})\, y(\text{NUSED}) | usedon(x \text{ "707" } y)$$
$$\forall w(\text{QUANTITY})\,(\neg\, instock(x \text{ "JFK" } w) \vee w \geq y \vee x = 2116)\}$$

is the domain calculus expression from Example 10.32. Let $att(x) =$ PART#, $att(y) =$ NUSED, and $att(w) =$ QUANTITY. To keep things tidy, let $F_P =$ [PART#], $F_N =$ [NUSED], and $F_Q =$ [QUANTITY]. The algebraic expression for $instock(x \text{ "JFK" } w)$ is

$$F_1 = \pi_{\text{PART\# QUANTITY}}(\sigma_{\text{LOCATION} = \text{JFK}}(instock)).$$

(Renaming is unnecessary in this instance.) The algebraic expression for $\neg\, instock(x \text{ "JFK" } w)$ is $\bar{F}_1$. The algebraic expressions for $w \geq y$ and $x = 2116$ are

$$F_2 = \sigma_{\text{QUANTITY} \geq \text{NUSED}}(F_Q \bowtie F_N) \text{ and}$$
$$F_3 = \sigma_{\text{PART\#} = 2116}(F_P).$$

An algebraic expression for

$$\neg\, instock(x \text{ "JFK" } w) \vee w \geq y \vee x = 2116$$

is

$$F_4 = (F_1 \bowtie F_N) \cup (F_2 \bowtie F_P) \cup (F_3 \bowtie F_Q \bowtie F_N).$$

Adding the quantifier $\forall w$(QUANTITY), we get the algebraic expression

$$F_5 = F_4 \div F_Q.$$

The algebraic expression for $usedon(x$ "707" $y)$ is

$$F_6 = \pi_{\text{PART\# NUSED}}(\sigma_{\text{PTYPE} = 707}(usedon)).$$

The equivalent algebraic expression for E is $F = F_6 \cap F_5$. No final renaming is necessary.

Just for fun, we can write F out:

$$\begin{aligned}&\pi_{\text{PART\# NUSED}}(\sigma_{\text{PTYPE} = 707}(usedon)) \cap \\ &\quad(((\pi_{\text{PART\# QUANTITY}}(\sigma_{\text{LOCATION} = \text{JFK}}(instock)) \bowtie F_N) \cup \\ &\qquad(\sigma_{\text{QUANTITY} \geq \text{NUSED}}(F_Q \bowtie F_N) \bowtie F_P) \cup \\ &\qquad(\sigma_{\text{PART\#} = 2116}(F_P) \bowtie F_Q \bowtie F_N)) \div F_Q).\end{aligned}$$

If we look back at the meaning of E given in Example 10.32, we see that a simpler algebraic expression equivalent to E exists, namely

$$\pi_{\text{PART\# NUSED}}(\sigma_{\text{QUANTITY} \geq \text{NUSED} \vee \text{PART\#} = 2116}(usedon \bowtie instock)).$$

Corollary For $\mathfrak{DC}$, $\mathfrak{R}$ and E as given in Theorem 10.6, there is an algebraic expression F over $\mathfrak{R}$ without complement that is equivalent to E under limited evaluation.

Proof The details are left mainly to the reader (see Exercise 10.30).

The proof here is similar to that of Theorem 10.6, with some modifications. The first problem is the correct expression for $[A]$. Expression E can be further modified so that where $\exists x(A)$ or $\forall x(A)$ appear, $att(x) = \text{A}$. Also, in $x_1(A_1)\, x_2(A_2) \cdots x_n(A_n)$, we can choose $att(x_i) = A_i$, $1 \leq i \leq n$, since $A_i \neq A_j$ for $i \neq j$. With this modification of E, any attribute symbol A appears at most once in E. If A appears in $\exists x(A)g$ or $\forall x(A)g$, then $[A]$ should be an algebraic expression for $edom(A,f)$.

The second problem is removing the use of complement from case 3.

The third problem is that the algebraic expression F_g for each subformula g of f will not be equivalent to

$$E_g = \{y_1(att(y_1))\, y_2(att(y_2)) \cdots y_m(att(y_m)) | g(y_1, y_2, \ldots, y_m)\}$$

under limited interpretation, as one might think. Rather, F_g will represent E_g where quantified formulas are given the limited interpretation, but y_i ranges over $[att(y_i)]$, $1 \leq i \leq m$, which may properly contain $edom(att(y_i),g)$.

10.8 TABLEAU QUERIES

This section covers a query system based on tableaux, namely *tableau queries*. Tableau queries are not as expressive as relational algebra, but they can represent any algebraic expression involving only select with an equality comparison, project, and join. With certain extensions, algebraic expressions involving union and difference can be handled. Tableau queries are of particular interest because they can be optimized to minimize the number of joins in the corresponding algebraic expression, although the optimization process can be somewhat expensive computationally. There is a subclass of the tableau queries that can be so optimized efficiently. It is also possible to modify tableau queries to take advantage of FDs and other data dependencies in a database, in a manner similar to the chase computation.

Tableau queries come in two flavors, "untagged" and "tagged." The first flavor is for queries against a single relation. Queries against a multirelation database can also be handled, if the relations are the projection of a common instance. The second flavor is used for queries against multirelation databases where the relations do not necessarily come from a common instance.

We present a class of algebraic expressions we can represent with tableau queries.

Definition 10.6 A *restricted algebraic expression* E is an algebraic expression built up from relations and single-tuple constant relations using

1. select in the form $\sigma_{A=c}$
2. project, and
3. natural join.

Intersection could also be allowed, as it can be expressed by join, and select is not strictly necessary, since it can be expressed by join with a constant relation. While restricted algebraic expressions exclude many interesting queries, there are many queries they can express.

Example 10.37 The query "What are the parts used on a 747?" against the database in Table 10.1 can be expressed by the restricted algebraic expression

$$\pi_{\text{PARTNAME}}(\sigma_{\text{PTYPE}\,=\,747}(\mathit{pinfo} \bowtie \mathit{usedon})).$$

10.8.1 Single Relation Tableau Queries

This section develops tableau queries for a single relation database. While such queries are not in themselves very interesting, they will serve to

demonstrate techniques and proofs that carry over to the multirelation case. Also, single relation tableau queries will serve to represent queries for a multirelation database where all the relations in the database are the projection of a common instance. Since the parts database is getting a little old, we introduce a new database for future examples. The database shown in Table 10.11 describes meals for our little airline. The attributes FL, ME, DT, OP, NM stand for flight, meal, date, option and number. The relation *serves* tells what meals are normally served on each flight. Relation *choice* tells what the different options are for each meal for a given date. Relation *ordered* tells how many of each option have been ordered for a given flight on a given date. Only the portion of the figure to the dashed line is *ordered*. Note that the three relations given join completely. We let *meals* be the relation that is the join of the three relations in the database, which is just *ordered* with the column to the right of the dashed line added.

A *tableau query* is a modified version of a tableau as introduced in Chapter 8. As before, there are distinguished and non-distinguished variables. In addition, there are constants and blanks. Distinguished variables, non-distinguished variables, and constants are collectively called *symbols*. A tableau query also has a *summary*, which is a special row of the tableau. We shall write the summary between two lines above the rows in the tableau query. A tableau query for a relation $r(A_1\ A_2\ \cdots\ A_k)$ must satisfy the following restrictions:

q1. The columns are labeled $A_1, A_2, \ldots, A_k$.
q2. Any variable may appear in only one column.
q3. If a distinguished variable appears in a column, it must also appear in the summary for that column.
q4. Only symbols (no blanks) may appear in the rows.
q5. Only distinguished variables, constants, and blanks may appear in the summary.
q6. If constant c appears in the A-column, then $c \in dom(A_i)$.

Generally, we denote the summary as w_0, and the rows as $w_1, w_2, \ldots, w_n$.

Example 10.38 Table 10.12 gives a tableau query for the relation *meals* that is the join of the relations in Table 10.11.

To evaluate a tableau query Q for a relation $r(R)$, we must extend the definition of a valuation. A *valuation ρ of a tableau query* Q is a function from the symbols of Q to domain values such that if α is a symbol in the A-column of Q, $\rho(\alpha) \in dom(A)$ and if c is a constant, then $\rho(c) = c$. Valuation ρ can naturally be extended to map rows of Q to tuples in $dom(R)$, and to map the summary of Q to $dom(S)$, where S is the set of columns of Q where the summary does not contain a blank. We then extend ρ to map Q to

Table 10.11 The Relations *serves*, *choice*, and *ordered*.

serves(FL, ME)

FL	ME
56	B
56	L
57	D
106	L
106	D
107	D

choice(ME, DT, OP)

ME	DT	OP
B	15 Aug	eggs
B	15 Aug	waffles
B	16 Aug	eggs
L	15 Aug	sandwich
L	16 Aug	lasagne
L	16 Aug	salad
D	15 Aug	pot roast
D	15 Aug	seafood crepe
D	16 Aug	pot roast
D	16 Aug	flounder
D	16 Aug	chicken

ordered(FL, DT, OP, NM) | (ME)

FL	DT	OP	NM	ME
56	15 Aug	eggs	27	B
56	15 Aug	waffles	23	B
56	16 Aug	eggs	50	B
56	15 Aug	sandwich	50	L
56	15 Aug	salad	50	L
57	15 Aug	pot roast	55	D
57	16 Aug	pot roast	60	D
106	16 Aug	sandwich	80	L
106	16 Aug	lasagne	45	L
106	16 Aug	salad	35	L
106	15 Aug	pot roast	40	D
106	15 Aug	seafood crepe	40	D
106	16 Aug	flounder	40	D
106	16 Aug	chicken	40	D
107	15 Aug	pot roast	60	D
107	16 Aug	chicken	60	D

Table 10.12 Tableau Query for the Relation *meals*.

	Q(FL	DT	OP	NM	ME)
w_0		a_2	a_3	a_4	
w_1	b_1	b_2	a_3	a_4	b_5
w_2	106	a_2	a_3	a_4	b_5
w_3	107	b_6	b_7	b_8	b_4

a relation, namely $\rho(Q)=\{\rho(w_i)|w_i$ is a row in $Q\}$. Note that $\rho(w_0)$ is not included in $\rho(Q)$.

Example 10.39 If Q is the tableau query in Table 10.12 and ρ is the valuation shown in Table 10.13, then $\rho(w_0) = \langle$15 Aug pot roast 40$\rangle$ and $\rho(Q)$ is shown in Table 10.14.

Table 10.13 Valuation Table for ρ.

$\rho(a_2)$ = 15 Aug	$\rho(b_1) = 57$
$\rho(a_3)$ = pot roast	$\rho(b_2)$ = 15 Aug
$\rho(a_4) = 40$	$\rho(b_3) = 60$
$\rho(106) = 106$	$\rho(b_4) = D$
$\rho(107) = 107$	$\rho(b_5) = D$
	$\rho(b_6)$ = 15 Aug
	$\rho(b_7)$ = pot roast
	$\rho(b_8) = 60$

Table 10.14 Valuation Table for $\rho(Q)$.

$\rho(Q)$(FL	DT	OP	NM	ME)
57	16 Aug	pot roast	60	D
106	15 Aug	pot roast	40	D
107	15 Aug	pot roast	60	D

The *value* of tableau query Q on relation r, denoted $Q(r)$, is the relation

$$\{\rho(w_0)|\rho \text{ is a valuation for } Q \text{ and } \rho(Q) \subseteq r\}.$$

Example 10.40 Let Q be the tableau query in Table 10.12 $Q(meals)$, for *meals* the join of the relations in Table 10.11, contains the tuple $\langle$15 Aug pot roast 40$\rangle$; $\rho(w_0) = \langle$15 Aug pot roast 40$\rangle$ and $\rho(Q) \subseteq$ *meals*, for ρ the valuation of Table 10.13. $Q(meals)$ is the answer to the question "For any meal that flight 107 serves, and for any option available for that meal, how many of that option does flight 106 have ordered for that meal on each day?" $Q(meals)$ is shown in Table 10.15.

Table 10.15 $Q(meals)$.

$Q(meals)$(DT	OP	NM)
15 Aug	pot roast	40
15 Aug	seafood crepe	40
16 Aug	flounder	40
16 Aug	chicken	40

It happens that Q is equivalent to the algebraic expression

$$\pi_{\text{DT OP NM}}(\pi_{\text{OP ME}}(\textit{meals}) \bowtie \pi_{\text{DT OP NM}}(\sigma_{\text{FL}=106}(\textit{meals}) \bowtie \pi_{\text{ME}}(\sigma_{\text{FL}=107}(\text{meals}))))).$$

This algebraic expression is equivalent to

$$\pi_{\text{DT OP NM}}(\pi_{\text{OP ME}}(\textit{choice}) \bowtie \pi_{\text{DT OP NM}}(\sigma_{\text{FL}=106}(\textit{ordered})) \bowtie \pi_{\text{ME}}(\sigma_{\text{FL}=107}(\textit{serves})))$$

if *serves*, *choice* and *ordered* are projections of meals.

We can immediately derive a tuple calculus expression equivalent to a given tableau query. Let Q be a tableau query for relation $r(R)$ with summary w_0 and rows $w_1, w_2, \ldots, w_n$. Let S be the set of attributes where w_0 is non-blank. The equivalent tuple calculus expression is

$$E = \{x(S) \mid \exists y_1(R) \in r\ \exists y_2(R) \in r \cdots \exists y_n(R) \in r\ f(x, y_1, y_2, \ldots, y_n)\}$$

where f is the conjunction of the atoms

1. $x(A) = y_i(A)$ where w_0 and w_i have the distinguished variable for the A-column,
2. $y_i(A) = y_j(A)$ where $w_i(A) = w_j(A)$ and $w_i(A)$ is not a constant,
3. $x(A) = c$ where $w_0(A) = c$, c a constant, and
4. $y_i(A) = c$ where $w_i(A) = c$, c a constant.

Expression E is safe, and moreover, is equivalent to an algebraic expression involving at most $n-1$ joins (see Exercise 10.35). Note that the choices for $y_1, y_2, \ldots, y_n$ in r correspond to $\rho(w_1), \rho(w_2), \ldots, \rho(w_n)$ for some valuation ρ such that $\rho(Q) \subseteq r$.

Example 10.41 Let R_m = FL DT OP NM ME. An equivalent tuple calculus expression to tableau query in Figure 10.13 is

$$\{x(\text{DT OP NM}) \mid \exists y_1(R_m)\ \exists y_2(R_m)\ \exists y_3(R_m)\ (x(\text{DT}) = y_2(\text{DT}) \wedge x(\text{OP}) = y_1(\text{OP}) \wedge x(\text{OP}) = y_2(\text{OP}) \wedge x(\text{NM}) = y_2(\text{NM}) \wedge y_1(\text{ME}) = y_3(\text{ME}) \wedge y_2(\text{FL}) = 106 \wedge y_3(\text{FL}) = 107)\}.$$

Note that the atom $y_1(\text{OP}) = y_2(\text{OP})$ could have been included, but would be redundant.

Definition 10.7 Let Q be a tableau query with scheme R and let w be a row of Q. If A is an attribute in R, then w is *matched in the* A*-column* if $w(A) = c$ for some constant c, $w(A) = w_0(A)$ for the summary w_0, or $w(A) = w'(A)$ for another row w' in Q. We call $w(A)$ a *matched symbol*. Let $match(w) = \{A \mid w \text{ is matched in the } A\text{-column}\}$.

Definition 10.8 Let Q be a tableau query on scheme R. Let d be a database over R with scheme $\mathbf{R} = \{R_1, R_2, \ldots, R_p\}$. Q *applies* to database d if for each row w in Q there is a relation scheme $R_i \in \mathbf{R}$ with $match(w) \in R_i$.

Example 10.42 For the tableau query Q in Table 10.12,

$$match(w_1) = \text{DT OP ME}$$
$$match(w_2) = \text{FL DT OP NM}$$
$$match(w_3) = \text{FL ME}.$$

Q applies to the database of Table 10.11. Q would not apply to a database d with database scheme {DT OP, OP ME, FL DT OP NM, FL ME}.

Let Q be a tableau query with summary w_0 and rows $w_1, w_2, \ldots, w_n$ that applies to database d. If the relations of d are all projections of a common instance r, then Q can be evaluated against d as follows. For each row w_i in Q, let $r_i(R_i)$ be a relation in d such that $match(w_i) \subseteq R_i$. For a valuation ρ on Q, we let $\rho(Q) \subseteq d$ mean that $\rho(w_i(R_i)) \subseteq r_i$. That is, ρ maps a portion of w_i containing $match(w_i)$ to a tuple in r_i. We can then let

$$Q(d) = \{\rho(w_0) \mid \rho \text{ is a valuation for } Q \text{ where } \rho(Q) \subseteq d\}.$$

It should be clear that $Q(d) = Q(r)$. For a valuation ρ such that $\rho(Q) \subseteq r$ it also happens that $\rho(Q) \subseteq d$. For any valuation ρ such that $\rho(Q) \subseteq d$, there is a valuation ρ' such that $\rho(w_0) = \rho'(w_0)$ and $\rho'(Q) \subseteq r$. Valuation ρ' always exists, since if $\rho(w_i(R_i)) = t_i \in r_i$, there is a tuple $t \in r$ such that $t(R_i) = t_i$. We can consistently choose ρ' such that $\rho'(w_i) = t$.

The condition that the relations in d be the projections of a common instance r must hold for $Q(d)$ to be well-defined. The choice of relation r_i for row w_i can affect $Q(d)$ if the condition does not hold.

Example 10.43 Let d be the database shown in Figure 10.2. If Q is the tableau query

A	B	C
a_1		
w_1 a_1	b_1	b_2,

then $Q(d)$ can be either $\langle 1{:}A \rangle$ or $\langle 2{:}A \rangle$, depending on whether r_1 or r_2 is chosen for w_1.

r_1		r_2	
A	B	A	C
1	3	2	4

Figure 10.2

10.8.2 Tableau Queries for Restricted Algebraic Expressions

In this section we shall see how to translate a restricted algebraic expression over a relation r into an equivalent tableau query. By the nature of the operators in a restricted algebraic expression, if any subexpression is identically the empty relation, then the entire expression is identically the empty relation. We shall use $Q_\emptyset$ to denote a special tableau query that always evaluates to the empty relation.

For a given restricted algebraic expression E for relation r, we define an equivalent tableau query Q recursively. Let the relation scheme for r be $A_1 A_2 \cdots A_m$.

Case 1 E is r. Then Q is

	A_1	A_2	$\cdots$	A_m
w_0	a_1	a_2	$\cdots$	a_m
w_1	a_1	a_2	$\cdots$	a_m .

Case 2 E is $\langle c_1{:}B_1\ c_2{:}B_2 \cdots c_k{:}B_k \rangle$, where $B_1\ B_2 \cdots B_k \subseteq A_1 A_2 \cdots A_m$ and c_i is a constant in $dom(B_i)$. Q is the tableau query consisting only of a summary w_0, where w_0 has c_i in the B_i-column, $1 \leq i \leq k$, and is blank elsewhere (see Exercise 10.34).

Case 3 E is $\sigma_{A_i=c}(E')$. Let Q' be the tableau query for E'. There are three possibilities for Q.

a. If $Q' = Q_\emptyset$, then $Q = Q_\emptyset$.
b. If w_0' is the summary for Q', and $w_0'(A_i) = c'$, then if $c = c'$, $Q = Q'$. Otherwise, when $c \neq c'$, Q is $Q_\emptyset$.

c. If w_0' is the summary for Q', and $w_0'(A_i) = a_i$, then Q is Q' with every occurrence of a_i replaced by c.

Case 4 E is $\pi_X(E')$. Let Q' be the tableau query for E'. If $Q' = Q_\emptyset$, then $Q = Q_\emptyset$. Otherwise, let w_0' be the summary of Q'. The summary w_0 for Q has $w_0(A_i) = w_0'(A_i)$ for $A_i \in X$ and is blank elsewhere. The rows of Q are the rows of Q', except if a_i is the distinguished variable for the A_i column, and $A_i \notin X$, then a_i is replaced by a new nondistinguished variable.

Case 5 E is $E' \bowtie E''$. Let Q' and Q'' be tableau queries for E' and E''. Assume that no nondistinguished variable appears both in E' and E''. There are three possibilities for Q.

a. If Q' or Q'' is $Q_\emptyset$, then $Q = Q_\emptyset$.
b. If w_0' and w_0'' are the summaries of Q' and Q'', and $w_0'(A_i) = c'$ and $w_0''(A_i) = c''$ for some A_i, and constants c' and c'' are unequal, then Q is $Q_\emptyset$.
c. Otherwise, the summary w_0 has
   i. $w_0(A_i) = c$ if either $w_0'(A_i) = c$ or $w_0''(A_i) = c$ (if a_i is changed to c in the summary, it is changed everywhere).
   ii. $w_0(A_i) = a_i$ if i. does not apply and either $w_0'(A_i) = a_i$ or $w_0''(A_i) = a_i$, and
   iii. $w_0(A_i) =$ blank elsewhere

   and the rows of Q are the rows of Q' and Q''.

Example 10.44 Let E be the restricted algebraic expression

$$\pi_{\text{FL DT}}(\pi_{\text{DT ME}}(\pi_{\text{FL OP ME}}(\sigma_{\text{FL}=56}(meal)) \bowtie \pi_{\text{DT OP}}(meal)) \bowtie meal)$$

for the relation *meal* that is the join of the relations in Table 10.11. We shall translate E to a tableau query, combining steps as we go. A query tableau for $\sigma_{\text{FL}=56}(meal)$ is shown in Figure 10.3. A tableau for $\pi_{\text{FL OP ME}}(\sigma_{\text{FL}=56}(meal))$ is shown in Figure 10.4. Figure 10.5 gives the tableau for $\pi_{\text{FL OP ME}}(\sigma_{\text{FL}=56}(meal)) \bowtie \pi_{\text{DT OP}}(meal)$. Figure 10.6 gives the tableau for $\pi_{\text{DT ME}}(\sigma_{\text{FL OP ME}}(\sigma_{\text{FL}=56}(meal)) \bowtie \pi_{\text{DT OP}}(meal)) \bowtie meal$, and Figure 10.7 gives a tableau query for all of E.

Theorem 10.7 If E is a restricted algebraic expression for relation r, and Q is the tableau query obtained by the method given above, then E is equivalent to Q.

(FL	DT	OP	NM	ME)
56	a_2	a_3	a_4	a_5
56	a_2	a_3	a_4	a_5

Figure 10.3

(FL	DT	OP	NM	ME)
56		a_3		a_5
56	b_1	a_3	b_2	a_5

Figure 10.4

(FL	DT	OP	NM	ME)
56	a_2	a_3		a_5
56	b_1	a_3	b_2	a_5
b_3	a_2	a_3	b_4	b_5

Figure 10.5

(FL	DT	OP	NM	ME)
a_1	a_2	a_3	a_4	a_5
56	b_1	b_6	b_2	a_5
b_3	a_2	b_6	b_4	b_5
a_1	a_2	a_3	a_4	a_5

Figure 10.6

(FL	DT	OP	NM	ME)
a_1	a_2			
56	b_1	b_6	b_2	b_7
b_3	a_2	b_6	b_4	b_5
a_1	a_2	b_8	b_9	b_7

Figure 10.7

Proof We proceed by cases.

Case 1 E is r. The reader should recognize the corresponding tableau query Q as the identity mapping, hence $Q(r) = r$.

Case 2 E is $\langle c_1{:}B_1\ c_2{:}B_2 \cdots c_k{:}B_k\rangle$. If w_0 is the summary of Q and ρ is a valuation of Q, then $\rho(w_0(B_i)) = c_i$, $1 \le i \le k$, so $Q(r) = \langle c_1{:}B_1\ c_2{:}B_2 \cdots c_k{:}B_k\rangle$.

Case 3 E is $\sigma_{A_i=c}(E')$. It should be obvious that Q is equivalent to E for possibilities a and b. For possibility c, suppose t is a tuple in $E(r)$. Then $t(A_i) = c$ and t is a tuple in $E'(r)$. It follows that there is a valuation ρ of Q' where $\rho'(Q') \subseteq r$ such that $\rho(w_0') = t$ and hence $\rho(a_i) = c$. Extending ρ, if necessary, so that $\rho(c) = c$, we see that $\rho(Q) \subseteq r$ and $\rho(w_0) = t$, hence $t \in Q(r)$. Similarly, it follows that if t is a tuple in $Q(r)$, $t \in E(r)$. We conclude E and Q are equivalent.

Case 4 E is $\pi_X(E')$. If $Q' = Q_\emptyset$, then E' is identically the empty relation on $sch(E')$, hence E will be identically the empty relation on X, so Q is correctly chosen. Otherwise, let t be a tuple in $E(r)$. There must be a tuple t' in $E'(r)$ with $t'(X) = t$. Let ρ be a valuation such that $\rho(Q') \subseteq r$ and $\rho(w_0') = t'$. If for any attribute A_i that is projected away by π_X we let $\rho(b_j) = \rho(a_i)$, where b_j is the nondistinguished variable that replaced a_i in Q', then $\rho(Q) \subseteq r$ and $\rho(w_0) = t$, so $t \in Q(r)$. A similar argument shows that any t in $Q(r)$ is also in $E(r)$, so E and Q are equivalent.

Case 5 is left to the reader (see Exercise 10.38).

Although every restricted algebraic expression has an equivalent tableau query, not every tableau query comes from a restricted algebraic expression.

Example 10.45 Consider the tableau query Q in Figure 10.8. Q is not the translation of any restricted algebraic expression using the method previously given. Suppose that Q came from expression E. E must be of the form $E' \bowtie E''$. Let Q' and Q'' be the tableau queries for E' and E''. Which rows of Q came from Q' and which came from Q''? Assume without loss of

Q

A	B
a_1	a_2
a_1	b_2
b_1	b_2
b_1	a_2

Figure 10.8

generality that row w_1 came from Q'. Row w_2 must also come from Q', since it agrees with w_1 on a non-distinguished symbol. Likewise, w_3 comes from Q'. No rows come from Q'', so E'' must be a constant relation, which is a contradiction, since Q contains no constants. Hence Q does not come from any restricted algebraic expression (see Exercise 10.39).

The method given for translating restricted algebraic expressions over a single relation also works for expressions over a database d if all the relations in d are projections of a common instance r. If E is a restricted algebraic expression over d, replace each occurrence of r_i by $\pi_{R_i}(r)$, where R_i is the scheme of r_i, and proceed as before. The resulting tableau query Q is guaranteed to apply to database d. Any row w in Q can be traced to a subexpression $\pi_{R_i}(r)$, which means $match(w) \subseteq R_i$. Suppose, for example, that r is a relation on $ABCD$ and $R_i = AC$. The tableau query for $\pi_{R_i}(r)$ is

$Q'(A$	B	C	$D)$
a_1		a_2	
w' a_1	b_1	a_2	b_2

Row w' eventually becomes w in Q. The nondistinguished symbols in w' can never become matched, either through selection or joining. Hence $match(w)$ in Q will be contained in $R_i = AC$.

Example 10.46 Let E be the expression

$$\pi_{\text{OP ME}}(\sigma_{\text{FL}=106,\ \text{DT}=15\ \text{Aug}}(serves \bowtie choice))$$

for the database in Table 10.11. Figure 10.9 shows a tableau query equivalent to E.

(FL	DT	OP	NM	ME)
		a_3		a_5
106	b_1	b_2	b_3	a_5
b_4	15 Aug	a_3	b_5	a_5

Figure 10.9

10.8.3 Tableau Queries that Come from Algebraic Expressions

Although we shall not characterize all tableau queries that come from restricted algebraic expressions, the following theorem does characterize some of those tableau queries.

Theorem 10.8 Let Q be a tableau query for relation $r(R)$. If Q has at most one matched symbol in every column, then Q can be derived from some restricted algebraic expression by the translation method given.

Proof It suffices to consider only tableau queries with no constants in the summary. If Q has constants $c_1, c_2, \ldots, c_k$ in columns $B_1, B_2, \ldots, B_k$ of the summary, then Q is the tableau query for the expression

$$\langle c_1{:}B_1\, c_2{:}B_2 \cdots c_k{:}B_k \rangle \bowtie E',$$

where E' is the tableau query for Q with all its constants in the summary changed to blanks.

Let $w_1, w_2, \ldots, w_n$ be the rows of Q. For each w_i we form an algebraic expression E_i. Let X_i be the set of columns where w_i has a constant. Let $Y_i = match(w_i) - X_i$. Start with the expression r. For every attribute $A \in X_i$, apply the selection $\sigma_{A=w_i}(A)$ to the expression. Finally, apply the projection π_{Y_i} to get E_i. Let $E = \pi_Y(E_1 \bowtie E_2 \bowtie \cdots \bowtie E_n)$, where Y is the set of columns where the summary of Q is not blank. It is left to the reader to show that E translates into Q by the correct choice of nondistinguished variables at each stage (see Exercise 10.41).

Note that the proof only depends on there not being both a distinguished variable (necessarily matched) and a matched nondistinguished variable in each column. Any number of constants can appear in a column, in addition to another matched symbol.

Example 10.47 Let r be a relation on scheme $ABCD$ and let Q be the tableau query for r shown in Figure 10.10. Q can be derived from the restricted algebraic expression

$$\pi_{AB}(\pi_{ACD}(r) \bowtie \pi_C(\sigma_{B=2}(\sigma_{D=3}(r)) \bowtie \pi_{BD}(r)).$$

The tableau queries Q_1, Q_2 and Q_3 to be used for subexpressions $\pi_{ACD}(r)$, $\pi_C(\sigma_{B=2}(\sigma_{D=3}(r))$, and $\pi_{BD}(r)$ are shown in Figure 10.11.

$A(A \quad B \quad C \quad D)$

a_1	a_2		
a_1	b_1	b_2	b_3
b_4	2	b_2	3
b_5	a_2	b_6	b_3

Figure 10.10

$Q_1(A$	B	C	$D)$
a_1		a_3	a_4
a_1	b_1	a_3	a_4

$Q_2(A$	B	C	$D)$
		a_3	
b_4	2	a_3	3

$Q_3(A$	B	C	$D)$
	a_2		a_4
b_5	a_2	b_6	a_4

Figure 10.11

10.8.4 Tableau Queries for Multirelation Databases

We now modify tableau queries so they can represent restricted algebraic expressions on databases where the relations are not necessarily the projection of some common instance. We associate a particular relation with each row in the query. Actually, we assume that all the relations in the database have distinct schemes, so we simply associate relation schemes with rows.

Let d be a database with scheme $\mathbf{R}$ over R. *A tagged tableau query* Q for d is similar to a regular tableau query with scheme R, except the rows of Q may contain blanks and every row has a *tag*. The tag of row w, denoted $tag(w)$, is some relation scheme $S \in \mathbf{R}$. Row w is blank in exactly the columns in $R - S$. We view a valuation ρ for Q as mapping row w to a tuple over $tag(w)$. The notation $\rho(Q) \subseteq d$ then means that for each row w in Q, $\rho(w) \in s$, where s is the relation in d with scheme $tag(w)$. If w_0 is the summary of Q, we let

$$Q(d) = \{\rho(w_0) | \rho \text{ is a valuation of } Q \text{ such that } \rho(Q) \subseteq d\}.$$

Example 10.48 Harking back to the database d in Table 10.1, we can construct the tagged tableau query Q shown in Figure 10.12. P#, SP, PN, PT, NU, LC, and QY are abbreviations for PART#, SUBPARTOF, PARTNAME, PTYPE, NUSED, LOCATION, and QUANTITY. Tags are shown in parentheses to the right of each row. $Q(d)$ is shown in Figure 10.13.

Q(P#	SP	PN	PT	NU	LC	QY)	
				a_5	a_6	a_7	
b_1	211	b_2					(P# SP PN)
b_1			707	a_5			(P# PT NU)
b_1					a_6	a_7	(P# LC QY)
b_1			727	b_3			(P# PT NU)

Figure 10.12

$Q(d)$(NU	LC	QY)
86	JFK	6
86	O'Hare	28
244	Boston	341
244	O'Hare	29

Figure 10.13

The tags are written in Figure 10.12 only for emphasis. They can be inferred from the non-blank portion of each row.

The translation from restricted algebraic expressions to tagged tableau queries is essentially the same as for the single-relation case. The only difference is the tableau query for a single relation s with scheme $A_1 A_2 \cdots A_k$. The tableau for s has a summary with distinguished symbols in the columns corresponding to $A_1 A_2 \cdots A_k$ and blanks elsewhere. It also has a single row that is identical to the summary.

Example 10.49 The tableau query Q in Figure 10.12 is the translation of the algebraic expression

$$\pi_{\text{NU LC QY}}(\sigma_{\text{SP}=211}(\textit{pinfo}) \bowtie \sigma_{\text{PT}=707}(\textit{usedon}) \bowtie \textit{instock} \bowtie \pi_{\text{P\#}}(\sigma_{\text{PT}=727}(\textit{usedon}))).$$

For a database where all the relations are the projection of a common instance, the tagged tableau query derived from an algebraic expression is equivalent to the untagged version. The two tableaux will be essentially identical, except some unmatched symbols in the untagged version will be blanks in the tagged version.

10.8.5 Tableau Set Queries

The range of algebraic expressions that can be modeled by tableaux can be extended if we allow sets of tableaux to denote a query. In this section we return to regular tableau queries, although the extension described works for tagged tableau queries as well.

Definition 10.9 A *monotonic algebraic expression* E is an algebraic expression built up from relations and single-tuple constant relations using

1. select in the form $\sigma_{A=c}$,
2. project,
3. natural join, and
4. union.

Note that with *union* and *intersection*, selection conditions with $\wedge$ and $\vee$ are possible. Also, *union* can be used to construct multiple-tuple constant relations.

Definition 10.10 Tableau queries Q_1 and Q_2 are *compatible* if they have the same scheme and their summaries are blank in the same columns. A set of tableau queries is compatible if each pair of queries in the set is compatible.

Definition 10.11 A *tableau set query* over scheme R is a set $\mathbf{Q} = \{Q_1, Q_2, \ldots, Q_m\}$ of compatible tableau queries, all with scheme R. For a relation $r(R)$, the *value* of $\mathbf{Q}$ on r, denoted $\mathbf{Q}(r)$ is

$$Q_1(r) \cup Q_2(r) \cup \cdots \cup Q_m(r).$$

Example 10.50 Let *meals* be the join of the relations in Table 10.11. Let $\mathbf{Q}$ be the tableau set query $\{Q_1, Q_2\}$, where Q_1 and Q_2 are shown in Figure 10.14. $\mathbf{Q}$(*meals*) is given in Figure 10.15.

Theorem 10.9 Let E be a monotonic algebraic expression. There is a tableau set query Q that is equivalent to E.

Proof By Theorem 10.7, it is sufficient to show that any monotonic algebraic expression E is equivalent to a monotonic algebraic expression

$$E_1 \cup E_2 \cup \cdots \cup E_m$$

where each E_i is a restricted algebraic expression. The equivalence follows because *select*, *project*, and *join* all distribute over *union*.

Q_1(FL	DT	OP	NM	ME)
a_1		a_3	a_4	
56	15 Aug	a_3	b_2	b_3
a_1	15 Aug	a_3	a_4	b_3

Q_2(FL	DT	OP	NM	ME)
a_1		a_3	a_4	
56	16 Aug	a_3	b_2	b_3
a_1	16 Aug	a_3	a_4	b_3

Figure 10.14

Q(*meals*)(FL	OP	NM)
56	eggs	27
56	waffles	23
56	eggs	50
56	sandwich	50
56	salad	50
106	sandwich	80

Figure 10.15

From Section 2.2 we know that

$$\sigma_{A=c}(E_1 \cup E_2) = \sigma_{A=c}(E_1) \cup \sigma_{A=c}(E_2)$$

for expressions E_1 and E_2. Exercise 2.8 a) gives us

$$\pi_X(E_1 \cup E_2) = \pi_X(E_1) \cup \pi_X(E_2).$$

It is not hard to show that

$$E_1 \bowtie (E_2 \cup E_3) = (E_1 \bowtie E_2) \cup (E_1 \bowtie E_3)$$

as well. Repeated application of these identities will transform a monotonic algebraic expression into the union of restricted algebraic expressions. Note that if E contains k unions, m can be a large as 2^k.

Example 10.51 The tableau set of query Q of Example 10.50 can be derived from the expression

$$\pi_{\text{FL OP NM}}(\pi_{\text{DT OP ME}}(\sigma_{\text{FL}=56}(\sigma_{\text{DT}=15\text{ Aug}}(meals) \cup \sigma_{\text{DT}=16\text{ Aug}}(meals))) \bowtie meals).$$

10.9 CONJUNCTIVE QUERIES

Conjunctive queries are a subset of domain calculus expressions. Although they are not as expressive as domain calculus, they can express many useful queries, and do occur as subexpressions in domain calculus expressions. Their main interest, as with tableau queries, is that they can be optimized effectively.

A *conjunctive query* for database d is a domain calculus expression of the form

$$\{x_1(A_1)\, x_2(A_2) \cdots x_n(A_n) \mid \exists y_1(B_1)\, \exists y_2(B_2) \cdots \exists y_m(B_m)\; f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)\}$$

such that f is the conjunction of atoms of the form $r(a_1\, a_2 \cdots a_p)$, where $r \in d$ and each a_i is either a constant, x_j for some j, or y_k for some k.

Example 10.52 The tagged tableau query Q in Figure 10.12 is equivalent to the conjunctive query

$$\{x_1(\text{NU})\, x_2(\text{LC})\, x_3(\text{QY}) \mid \exists y_1(\text{P\#})\, \exists y_2(\text{PN})\, \exists y_3(\text{NU})\; \textit{pinfo}(y_1\ 211\ y_2) \wedge \textit{usedon}(y_1\ \text{“707”}\ x_1) \wedge \textit{instock}(y_1\, x_2\, x_3) \wedge \textit{usedon}(y_1\ \text{“727”}\ y_3)\}.$$

Every conjunctive query is a safe domain calculus expression. They are as expressive as tagged tableau queries that do not contain constants in the summary (see Exercise 10.45). Conjunctive queries are basically tagged tableau queries where a symbol may appear in multiple columns. Conjunctive queries are easily translated into equivalent algebraic expressions (see Exercise 10.46).

10.10 EXERCISES

10.1 Using the alternative definition of equivalence where expressions must have the same value on all mutually consistent databases, ex-

hibit algebraic expressions E_1, E_2, and E_3, where $E_1 \equiv E_2$ and $E_2 \equiv E_3$, and where there exists a database d that is mutually consistent with E_1 and E_3, but $E_1(d)$ and $E_3(d)$ are relations over different schemes.

10.2* Consider databases with the same scheme as the one in Table 10.1. Prove that there is no expression in relational algebra that denotes the relation r(PART SUBPART') that contains a tuple p s if and only if s is a subpart of p, or a subpart of a subpart, or a subpart of a subpart of a subpart, and so on. That is, s is used somewhere in p.

Many of the following exercises refer to the database pictured in Table 10.16. The *usage* relation gives, for each aircraft type, the number in service, the combined miles of flight (in ten thousands), and the combined hours of flight (in hundreds). The *accidents* relation gives, for each aircraft type and type of accident, the number of accidents and the number of people injured.

Table 10.16 Relations *accidents* and *usage*.

usage(PTYPE	INSERV	TMILES	THOURS)
707	14	7,358	1,839
727	12	6,621	1,642
747	8	3,784	841
A100	3	1,213	397
DC8	21	11,016	2,803

accidents(PTYPE	TACC	NACC	INJURED)
707	takeoff	2	6
727	takeoff	1	3
727	landing	4	17
A100	landing	1	12
A100	inflight	1	6
DC8	inflight	1	26

The domain of INSERV, TMILES, THOURS, NACC, and INJURED is the non-negative integers; *dom*(PTYPE) = {707, 727, 747, A100, DC8, DC10}; and *dom*(TACC) = {takeoff, landing, inflight, taxiing}. The comparators for the non-negative integers are $=$, $\neq$, $<$, $\leq$, $\geq$, $>$. The comparators for the other two domains are just $=$ and $\neq$. Let R_1 be the scheme of *usage* and R_2 be the scheme of *accidents*.

10.3 For each of the formulas, state which are legal. For the legal formulas, state for each tuple variable occurrence whether it is free or

bound; if it is bound, indicate to which quantifier it is bound. Also for the legal formulas, for each tuple variable that occurs free in the formula, give *type* and *men*. Recall R_1 is the scheme of *usage* and R_2 is the scheme of *accidents*.

(a) $x(\text{PTYPE}) = y(\text{PTYPE}) \wedge y(\text{THOURS}) \geq 1{,}000$.

(b) $x(\text{TACC}) \wedge x(\text{INSERV})$.

(c) $x(\text{TACC}) \wedge \mathit{usage}(x)$.

(d) $\exists x(R_1)\,(\mathit{usage}(x) \wedge \exists x(R_2)$
$(\mathit{accidents}(x) \wedge x(\text{PTYPE}) = x(\text{PTYPE}) \wedge x(\text{NACC}) =$
$y(\text{NACC}))))$.

(e) $\forall x(R_1) \in \mathit{accidents}\,(x(\text{PTYPE}) = y(\text{PTYPE}) \wedge$
$\exists z(R_1R_2)\,(z(\text{INSERV}) \geq y(\text{INSERV}) \wedge$
$z(\text{PTYPE}) \geq x(\text{PTYPE})))$.

(f) $\forall x(R_1)\,\exists x(R_2)\,(x(\text{NACC}) \leq 6)$.

(g) $\forall x(R_2) \in \mathit{accidents}(\neg x(\text{PTYPE TACC}) =$
$y(\text{PTYPE TACC}) \vee (x(\text{NACC}) \geq y(\text{NACC}) \wedge$
$x(\text{INJURED}) \geq y(\text{INJURED})))$.

10.4 Assuming the atoms *true* and *false* are both legal formulas, show that $f(t/x)$ is legal if $f(x)$ is legal.

10.5 Show that if legal formula $f = \exists x(R)g$ has no free variables, then x is the only variable that occurs free in g.

10.6 Using the database of Table 10.16, give the (unlimited) interpretation of each of the following formulas, where R_1 and R_2 are the schemes of *usage* and *accidents*.

(a) $\forall x(R_1)\,(x(\text{INSERV}) < 60)$.

(b) $\forall x(R_1)\,(\neg \mathit{usage}(x) \vee x(\text{INSERV}) < 60)$.

(c) $\forall x(R_2) \in \mathit{accidents}\,(x(\text{PTYPE}) \neq \text{"727"} \wedge$
$(x(\text{TACC}) = \text{"landing"} \vee x(\text{TACC}) = \text{"takeoff"})$.

(d) $\exists x(R_1) \in \mathit{usage}\ \exists y(R_2) \in \mathit{accidents}$
$(x(\text{PTYPE}) = y(\text{PTYPE}) \wedge y(\text{TACC}) = \text{"inflight"} \wedge$
$x(\text{TMILES}) \geq 5{,}000 \wedge y(\text{NACC}) \leq 1)$.

(e) $\exists x(R_1) \in \mathit{usage}\ \exists y(R_1) \in \mathit{usage}\ \forall z(R_2) \in \mathit{accidents}$
$\forall w(R_2) \in \mathit{accidents}\,(\neg(x(\text{PTYPE}) = z(\text{PTYPE}) \wedge$
$y(\text{PTYPE}) = w(\text{PTYPE})) \vee z(\text{TACC}) \neq w(\text{TACC}) \vee$
$z(\text{NACC}) \leq w(\text{NACC}))$.

Hint: There is a join going on here.

10.7 Let f be the formula

$\exists x(R_3)\,(\mathit{instock}(x) \wedge x(\text{LOCATION} = \text{"JFK"} \vee$
$\forall y(R_3)\,(\neg \mathit{instock}(y) \vee \neg y(\text{PART\#}) = x(\text{PART\#}) \vee$
$y(\text{QUANTITY}) < x(\text{QUANTITY}))$

for the database in Table 10.1, where R_3 = PART# LOCATION QUANTITY. Show that $I(f) = \textit{false}$ for any state of the relation *instock*.

10.8 Using the database in Table 10.16, give the value of $\{x(R_1) | f(x)\}$ for each of the following choices for f (under the unlimited evaluations). (You can write an infinite relation on a single sheet of paper as follows. Write the first tuple on half the sheet. Write the next two tuples on half of what remains. Write the next four tuples on half of what now remains. Continue in this manner until you have written all the tuples. *Warning*: This method does not work for uncountably infinite relations.)

(a) $f(x) = \textit{usage}(x) \wedge x(\text{THOURS}) \geq 1{,}000$.

(b) $f(x) = \textit{usage}(x) \wedge \forall y(R_2) \in \textit{accidents}$
$(x(\text{PTYPE}) \neq y(\text{PTYPE}))$.

(c) $f(x) = \exists y(R_1) \in \textit{usage}$
$(x(\text{PTYPE INSERV TMILES}) = y(\text{PTYPE INSERV TMILES}))$

(d) $f(x) = \textit{usage}(x) \wedge \forall y(R_2) \in \textit{accident}$
$(x(\text{PTYPE}) \neq y(\text{PTYPE}) \vee \forall z(R_2) \in \textit{accident}$
$(y(\text{PTYPE}) = z(\text{PTYPE}) \vee y(\text{TACC}) \neq z(\text{TACC})$
$\vee\, y(\text{NACC}) > z(\text{NACC})))$.

(e) $f(x) = \exists y(R_1) \in \textit{usage}\ (x(\text{PTYPE}) = y(\text{PTYPE}) \wedge$
$\exists z(R_i) \in \textit{usage}(x(\text{INSERV TMILES THOURS}) =$
$z(\text{INSERV TMILES THOURS}) \wedge$
$z(\text{INSERV}) \leq 5))$.

(f) $f(x) = \exists y(R_2) \in \textit{accidents}(x(\text{PTYPE}) = y(\text{PTYPE}) \wedge$
$\exists z(R_1) \in \textit{usage}(x(\text{INSERV TMILES THOURS}) =$
$z(\text{INSERV TMILES THOURS}) \wedge$
$z(\text{INSERV}) \leq 5))$.

10.9 For the database in Table 10.16, give tuple calculus expressions that will provide the answers to the following questions:

(a) Which plane types have no inflight accidents listed?

(b) Which plane type has logged the most miles and how many miles is that?

(c) Which plane types have more hours but less inflight accidents than other plane types? (Assume that there is an attribute PTYPE′ with the same domain as PTYPE and that no entry for a plane type and accident type in accidents means no accidents of that type.)

10.10 Complete cases 3, 5, and 6 of Theorem 10.1.

10.11 Give equivalent tuple calculus expressions for the following algebraic expressions over the database in Table 10.16.
(a) $\pi_{\text{PTYPE}}(\overline{usage \bowtie accidents})$.
(b) $\sigma_{\text{PTYPE}=707 \vee \text{INSERV} \leq 10}(\overline{usage})$.
(c) $\overline{accidents}$.
(d) ⟨707:PTYPE 12:INSERV 1,213:TMILES⟩

10.12 Let E be the algebraic expression $E_1 \div E_2$, where $sch(E_1) = RS$ and $sch(E_2) = S$. Give a tuple calculus expression equivalent to E in terms of formulas in the expressions $\{x(RS) | f(x)\}$ and $\{y(S) | g(y)\}$ for E_1 and E_2.

10.13 Give a tuple calculus expression E for the database d in Table 10.16 such that $E(d)$ and its complement are both infinite under the unlimited interpretation.

10.14 For the following formulas, let r be a relation on $ABCD$ and S a relation on CDE where $dom(A) = dom(C) = dom(E) =$ positive integers. For each formula f, give an algebraic or tuple calculus expression for $edom(A,f)$.
(a) $r(x) \wedge x(c) \leq 10$.
(b) $\exists z(ABCD)\, \exists y(CDE)\, (z(B) = x(B) \wedge z(D) = y(D) \wedge x(CD) = y(CD))$.
(c) $\exists x(ABCD)\, (x(C) \leq z(C) \vee (z(E) \leq 15 \wedge z(E) \geq 5))$

10.15 Repeat Exercise 10.6 using the limited interpretation of formulas.

10.16 Repeat Exercise 10.8 using the limited evaluation of expressions.

10.17 Give a tuple calculus expression E for the database d in Table 10.16 such that $E(d)$ is finite under both the unlimited and limited interpretations, but such that the value is different under the two interpretations.

10.18 Show that every tuple calculus expression used in the proof of Theorem 10.1, except for case 7, has the same value under unlimited and limited evaluation (provided you did Exercise 10.10 properly).

10.19 Which expressions in Exercise 10.8 are safe?

10.20 Let $r_1, r_2, \ldots, r_p$ be relations on schemes $R_1, R_2, \ldots, R_p$. Let E be any expression of the form

$$\{x(S) | \exists y_1(R_1) \in r_1\, \exists y_2(R_2) \in r_2 \cdots \exists y_p(R_p) \in r_p \\ (f(x,y_1,y_2, \ldots, y_p) \wedge g(x,z_1,z_2, \ldots, z_q))\}$$

where

1. f is an arbitrary legal tuple calculus formula with no quantifiers,
2. $\{z_1,z_2, \ldots, z_q\}$ is a subset of $\{y_1,y_2, \ldots, y_p\}$, and

3. g is the conjunction of atoms, where for every attribute $A \in S$ there is an atom $x(A) = z_i(A)$ or $x(A) = c$ in g.

Show that E is safe. Is E safe if some of the $\exists$'s are changed to $\forall$'s? If f may contain quantifiers?

10.21 Prove Lemma 10.1. You may wish to define safe tuple calculus formulas as those satisfying conditions s2 and s3 of the definition of safe expression.

10.22 Let R be a set of attributes. Show that for a legal tuple calculus formula g there is another formula h such that the value of $\{y(R)|g(y)\}$ under unlimited evaluation is $edom(R,g)$, and such that $edom(R,g) = edom(R,h)$.

10.23 Give the unlimited and limited evaluations of the following domain calculus expressions for the database in Table 10.16

(a) $\{x(\text{PTYPE})\, y(\text{INSERV})|(\exists z(\text{TMILES})\ \exists w(\text{THOURS})$
$(usage(x\ y\ z\ w) \wedge (z \geq 4{,}000 \vee w \geq 1{,}000))\}$.

(b) $\{x(\text{TACC})|\forall y(\text{PTYPE})\ \forall z(\text{NACC})\ \forall w(\text{INJURED})$
$\neg\, accidents(y\ x\ z\ w)\}$.

(c) $\{x(\text{PTYPE})\, y(\text{INSERV})|y \geq 10 \wedge \forall z_1(\text{TMILES})\ \forall z_2(\text{THOURS})$
$(\neg\, usage(x\ y\ z_1\ z_2)\ \exists w(\text{INJURED})$
$accidents(x\ \text{"inflight"}\ 1\ w))\}$.

10.24 Which expressions in Exercise 10.23 are safe?

10.25 Give domain calculus expressions to answer the questions in Exercise 10.9.

10.26 Convert the tuple calculus expressions in Exercise 10.8 to equivalent domain calculus expressions, and simplify where possible.

10.27 Prove part 1 of Theorem 10.5.

10.28 Complete case 5 of the proof of Theorem 10.6.

10.29 For each domain calculus expression E in Exercise 10.23 give an equivalent algebraic expression, assuming unlimited evaluation of E.

10.30* Complete the proof of the corollary to Theorem 10.6.

10.31 Which of the tuple calculus expressions in Exercise 10.23 have equivalent restricted algebraic expressions?

10.32 Compute $Q(meals)$ for the following tableau queries, where *meals* is the join of the relations in Table 10.12.

(a) Q

FL	DT	OP	NM	ME
a_1	a_2			
a_1	a_2	eggs	b_1	b_2

(b) Q(FL DT OP NM ME)

FL	DT	OP	NM	ME
a_1	a_2		a_4	
a_1	15 Aug	b_1	b_2	b_3
a_1	a_2	b_1	a_4	b_4

(c) Q(FL DT OP NM ME)

FL	DT	OP	NM	ME
57	a_2			
57	15 Aug	b_1	b_2	b_3
57	a_2	b_1	a_4	b_4

(d) Q(FL DT OP NM ME)

FL	DT	OP	NM	ME
a_1		a_3		a_4
a_1	b_1	b_2	b_3	a_4
b_4	b_5	a_3	b_6	a_4

10.33 Give algebraic expressions equivalent to the tableau queries in Exercise 10.32.

10.34 What is the difference between tableau queries Q and Q' below?

$Q(A)$
5
b_1

$Q'(A)$
5

10.35 Let R be a relation scheme and let $A_1 A_2 \cdots A_m \subseteq R$. Consider a tuple calculus expression

$$E = \{x(A_1A_2 \cdots A_m) \mid \exists y_1(R) \in r\, \exists y_2(R) \in r \cdots \exists y_n(R) \in r \\ x(A_1) = b_1 \wedge x(A_2) = b_2 \wedge \cdots \wedge x(A_m) = b_m \wedge \\ g(y_1, y_2, \ldots, y_n)\}$$

where b_i is either a constant in $dom(A_i)$ or $y_j(A_i)$ for some $1 \leq j \leq n$ and, g is the conjunction of atoms of the forms $y_i(A_k) = y_j(A_k)$ and $y_i(A_k) = c$, c a constant, in $dom(A_k)$.

(a) Show that E is safe.

(b) Show that E is equivalent to an algebraic expression with at most n-1 natural joins and no theta-joins.

(c) Show that E is equivalent to a tableau query.

(d) Show that if g is an arbitrary formula without quantifiers that may contain atoms of the form $y_i(A_k)\ \theta\ y_j(A_l)$ and $y_i(A_k) = c$, then (a) and (b) can be shown, but not (c).

10.36 Let Q be a tableau query that applies to database d. Assuming the relations of d are projections of a single relation r, show how to derive a tuple calculus expression on database d that is equivalent to Q.

10.37 Give equivalent tableau queries for the following restricted algebraic expressions on the database of Table 10.12, assuming the relations are all projections of a common instance.

(a) $\pi_{\text{OP ME}}(\sigma_{\text{FL}=106}(\textit{serves}) \bowtie \sigma_{\text{DT}=15\ \text{Aug}}(\textit{choice}))$.

(b) $\pi_{\text{OP ME}}(\sigma_{\text{FL}=106}(\textit{serves}) \bowtie \sigma_{\text{DT}=15\ \text{Aug}}(\textit{ordered}) \bowtie \textit{choice})$.

(c) ⟨B:ME eggs:OPTION⟩ $\bowtie$ *ordered*.

(d) $\pi_{\text{FL}}(\sigma_{\text{DT}=15\ \text{AUG OP}=\text{salad}}(\textit{serves} \bowtie \textit{choice}))$.

10.38 Prove case 5 of Theorem 10.7.

10.39* Show that the tableau query Q in Figure 10.8 is not equivalent to any tableau query Q' that comes from a restricted algebraic expression.

10.40 Give a tableau query that comes from a restricted algebraic expression, but that is not characterized by Theorem 10.8.

10.41 Complete the proof of Theorem 10.8.

10.42 Repeat Exercise 10.37 without the assumption that the relations are projected from a common instance.

10.43 Let r be a relation on scheme $ABCD$. Give tableau set queries for the following algebraic expressions.

(a) $\sigma_{A=6 \vee (B=7 \wedge C=2)}(r)$

(b) $(\sigma_{A=6}(r) \cup \sigma_{A=7}(r)) \bowtie \pi_{CD}(\sigma_{A=B}(r))$.

10.44 Let E be an algebraic expression involving relation r, single-tuple constant relations, select in the form $\sigma_{A=c}$, project, join, union, and difference. Are there necessarily tableau set queries $\mathbf{Q}_1$ and $\mathbf{Q}_2$ such that

$$E(r) = \mathbf{Q}_1(r) - \mathbf{Q}_2(r)?$$

10.45 Show that there is a conjunctive query equivalent to any tagged tableau query that does not have constants in its summary.

10.46 Show that a conjunctive query

$$\{x_1(A_1)x_2(A_2)\cdots x_n(A_n) \mid \exists y_1(B_1)\,\exists y_2(B_2)\cdots \exists y_m(B_m)\ f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)\},$$

where f is the conjunction of k atoms, has an equivalent algebraic expression using renaming, $k-1$ equijoins, and a single projection.

10.11 BIBLIOGRAPHY AND COMMENTS

Codd [1971a, 1972b] proposed relational calculus as the benchmark for query language completeness and showed its equivalence to relational algebra. The concept of safe expressions, as well as the general structure of the equivalence proofs, are from Ullman [1980]. However, Codd's and Ullman's proofs are for the case where relations have ordered columns and no attribute names.

Conjunctive queries are due to Chandra and Merlin [1976], and are the basis for tableau queries introduced by Aho, Sagiv, and Ullman [1979a, 1979b]. Tableau set queries are due to Sagiv and Yannakakis [1978], who also handle the difference operator in a limited form. Klug [1980b, 1980c] has extended tableau queries to handle inequality comparisons.

As the structure of tuple and domain calculus indicates, there is a close connection between predicate logic and databases. Gallaire and Minker [1979] have edited a collection of papers on that connection. Jacobs [1979, 1980a, 1980b] has dealt extensively with the use of logic in database theory.

In Chapter 14 we shall see that there are natural operations on relations that cannot be expressed in relational algebra, thus questioning the aptness of the definition of a complete query system.