Chapter 14

ASSORTED TOPICS

This chapter examines further topics in dependency theory, some limitations of relational algebra, and an extension of the relational model to include computed relations.

14.1 LOGIC AND DATA DEPENDENCIES

In this chapter we establish a connection between the theory of FDs and MVDs and a fragment of propositional logic. We give a way to interpret FDs and MVDs as formulas in propositional logic. For a set of dependencies **C** and a single FD or MVD c, we show that **C** implies c as dependencies if and only if **C** implies c under the logic interpretation. We first prove this result when **C** is FDs alone. We then extend the results to include MVDs, which complicates the proofs of the results considerably.

The correspondence between FDs and propositional formulas is direct. Let $X \rightarrow Y$ be an FD, where

$$X = A_1 A_2 \cdots A_m$$

and

$$Y = B_1 B_2 \cdots B_n.$$

The corresponding logical formula is

$$(A_1 \wedge A_2 \wedge \cdots \wedge A_m) \Rightarrow (B_1 \wedge B_2 \wedge \cdots \wedge B_n).$$

The A's and B's are viewed as propositional variables. The shorthand we use for this logical formula is $X \Rightarrow Y$. If X or Y is empty, we use *true* in place of

the conjunction of variables. Thus, the corresponding logical formula for $X \rightarrow Y$ when $X = \emptyset$ is

$$true \Rightarrow B_1 \wedge B_2 \wedge \cdots \wedge B_n.$$

There is some easy evidence that this correspondence of FDs and formulas will give the desired equivalence of FD implication and logical implication: all the inference rules for FDs are valid inference rules for logic when the FDs are interpreted as logical formulas (see Exercise 14.1).

Example 14.1 Consider the transitivity rule for FDs:

$$X \rightarrow Y \quad \text{and} \quad Y \rightarrow Z \quad \text{imply} \quad X \rightarrow Z.$$

The corresponding rule for logical inference is

$$X \Rightarrow Y \quad \text{and} \quad Y \Rightarrow Z \quad \text{imply} \quad X \Rightarrow Z,$$

which is the transitive rule of inference for logic. If we know $A\,B \rightarrow C$ and $C \rightarrow D\,E$, we may conclude $A\,B \rightarrow D\,E$. Likewise, if we know $A \wedge B \Rightarrow C$ and $C \Rightarrow D \wedge E$, we may infer $A \wedge B \Rightarrow D \wedge E$.

14.1.1 The World of Two-Tuple Relations

In the rest of the material on dependencies and logic, we shall make extensive use of relations with only two tuples. By *the world of two-tuple relations* we mean dependency theory restricted to relations consisting of exactly two tuples. For FDs and MVDs, we shall see that implication in the world of two-tuple relations is the same as implication over unrestricted (finite) relations. The equivalence does not hold for JDs or embedded MVDs.

Let r be a relation over scheme R with exactly two tuples. Call them t_1 and t_2. Relation r can be used to define a truth assignment for the attributes in R, when they are considered as propositional variables.

Definition 14.1 Let r be a relation on scheme R. The *truth assignment for* r, denoted Ψ_r, is the function from R to $\{true, false\}$ defined by

$$\Psi_r(A) = \begin{cases} true \text{ if } t_1(A) = t_2(A) \\ false \text{ if } t_1(A) \neq t_2(A). \end{cases}$$

Example 14.2 Let $r(A\ B\ C\ D)$ be the relation in Figure 14.1. The truth assignment for r is given by

$$\Psi_r(A) = \textit{true}$$
$$\Psi_r(B) = \textit{false}$$
$$\Psi_r(C) = \textit{false}$$
$$\Psi_r(D) = \textit{true}.$$

$r(A$	B	C	$D)$
1	2	4	6
1	3	5	6

Figure 14.1

Lemma 14.1 Let $X \rightarrow Y$ be an FD over R and let r be a relation on R with two tuples, t_1 and t_2. $X \rightarrow Y$ is satisfied by r if and only if $X \Rightarrow Y$ is true under the truth assignment Ψ_r.

Proof (if) Let $X = A_1\, A_2 \cdots A_m$ and let $Y = B_1\, B_2 \cdots B_n$. If Ψ_r makes $X \Rightarrow Y$ true, then Ψ_r must make $A_1 \wedge A_2 \wedge \cdots \wedge A_m$ false or $B_1 \wedge B_2 \wedge \cdots \wedge B_n$ true. If $A_1 \wedge A_2 \wedge \cdots \wedge A_m$ is false, then for some i between 1 and m we have $t_1(X_i) \neq t_2(X_i)$. It follows that r satisfies $X \rightarrow Y$. If $B_1 \wedge B_2 \wedge \cdots \wedge B_n$ is made true by Ψ_r, then $t_1(Y) = t_2(Y)$, and so $X \rightarrow Y$ is again satisfied.

(only if) Left to the reader (see Exercise 14.3).

Example 14.3 Let $r(A\ B\ C\ D)$ be the relation from the last example. The FD $A \rightarrow D$ is satisfied by r, and Ψ_r makes $A \Rightarrow D$ true. Relation r does not satisfy $A \rightarrow B$, and Ψ_r makes $A \Rightarrow B$ false.

Lemma 14.2 Let $r(R)$ be a relation, let F be a set of FDs on R, and let $X \rightarrow Y$ be a single FD on R. If r satisfies F and violates $X \rightarrow Y$, then some two-tuple subrelation s of r satisfies F and violates $X \rightarrow Y$.

Proof The result is fairly immediate, and we do not include a proof. We only recall that if r satisfies F, so does any subrelation of r.

Example 14.4 Figure 14.2 shows a relation r on scheme $A\ B\ C\ D$ that satisfies the set of FDs $F = \{AB \rightarrow D,\ C \rightarrow D\}$ and violates $C \rightarrow B$. Figure 14.3 shows a two-tuple subrelation s of r that satisfies F and violates $C \rightarrow B$.

$r(A$	B	C	$D)$
1	2	4	6
1	2	5	6
1	3	5	6
1	3	4	6

Figure 14.2

$s(A$	B	C	$D)$
1	2	5	6
1	3	5	6

Figure 14.3

There is a certain set of two-tuple relations that we shall need in the proof of the next theorem. Let R be a relation scheme and let $X \subseteq R$. We let 2_X stand for the relation on R that consists of two tuples, t_1 and t_2, where t_1 is all 1's and t_2 is 1's on X and 0's elsewhere. The important point is that 2_X is a two-tuple relation where the tuples agree on exactly X.

Example 14.5 Figure 14.4 shows the relation 2_{AB} on scheme $A\ B\ C\ D$.

$2_{AB}(A$	B	C	$D)$
1	1	1	1
1	1	0	0

Figure 14.4

14.1.2 Equivalence of Implication for Logic and Functional Dependencies

Theorem 14.1 Let F be a set of FDs over scheme R and let $X \rightarrow Y$ be an FD over R. The following are equivalent.

1. F implies $X \rightarrow Y$.
2. F implies $X \rightarrow Y$ in the world of two-tuple relations.
3. F implies $X \Rightarrow Y$ as logical formulas.

Proof Obviously $1 \Rightarrow 2$. By Lemma 14.2, $2 \Rightarrow 1$. For $2 \Rightarrow 3$, we show the contrapositive, $\neg 3 \Rightarrow \neg 2$. Let Ψ be a truth assignment on R that makes every formula in F true, but makes $X \Rightarrow Y$ false. Let

$$Z = \{A \in R \mid \Psi(A) = \mathit{true}\}.$$

Consider the two-tuple relation 2_Z on R. By Lemma 14.1, 2_Z satisfies F (as FDs) but not $X \rightarrow Y$.

The proof is similar for $3 \Rightarrow 2$. Suppose F does not imply $X \rightarrow Y$ for two-tuple relations over R. Let $r(R)$ be a two-tuple relation satisfying the FDs in F and violating $X \rightarrow Y$. By Lemma 14.1, Ψ_r makes F true (as formulas), but makes $X \Rightarrow Y$ false.

14.1.3 Adding Multivalued Dependencies

In this section we add extend Lemmas 14.1 and 14.2 and Theorem 14.1 to include multivalued dependencies. Throughout this section, we shall assume that for an MVD $X \twoheadrightarrow Y$, the left and right sides are disjoint. The correspondence between MVDs and logical formulas is not quite as direct as for FDs. Let R be a relation scheme. Let $X = A_1 A_2 \cdots A_m$, $Y = B_1 B_2 \cdots B_n$, and $Z = C_1 C_2 \cdots C_p$ be a partition of R. The logical formula corresponding to $X \twoheadrightarrow Y$ is

$$(A_1 \wedge A_2 \wedge \cdots \wedge A_m) \Rightarrow ((B_1 \wedge B_2 \wedge \cdots \wedge B_n) \vee (C_1 \wedge C_2 \wedge \cdots \wedge C_p)).$$

For the cases where m, n, or p are 0, we assume the conjunction of 0 propositional variables is *true*. We abbreviate the formula above as $X \Rightarrow (Y \vee Z)$.

Example 14.6 The MVD $B \twoheadrightarrow A\ D$ on scheme $A\ B\ C\ D\ E$ corresponds to the logical formula $B \Rightarrow ((A \wedge D) \vee (C \wedge E))$.

We now introduce a few definitions for MVDs in connection with two-tuple relations. Let $X \twoheadrightarrow Y$ be an MVD on scheme R and let $r(R)$ be a two-tuple relation that satisfies $X \twoheadrightarrow Y$. We say r *actively* satisfies $X \twoheadrightarrow Y$ if the two tuples in r agree on X.

Lemma 14.3 Let R be a relation scheme, and let X, Y, and Z partition R. Let $r = \{t_1, t_2\}$ be a two-tuple relation on R. Relation r actively satisfies $X \twoheadrightarrow Y$ if and only if

1. $t_1(X) = t_2(X)$, and
2. $t_1(Y) = t_2(Y)$ or $t_1(Z) = t_2(Z)$.

Proof Left to the reader (see Exercise 14.5).

We now give the analogue of Lemma 14.1 with MVDs added.

Lemma 14.4 Let c be an FD or MVD over scheme R and let r be a two-tuple relation on R. The dependency c is satisfied by r if and only if c as a logical formula is true under the truth assignment Ψ_r.

Proof Let $r = \{t_1, t_2\}$. If c is an FD, we may appeal to Lemma 14.1, so let c be the MVD $X \twoheadrightarrow Y$, and let $Z = R - XY$.

(if) Ψ_r makes $X \Rightarrow (Y \vee Z)$ true. If Ψ_r makes X false, then $t_1(X) \neq t_2(X)$ in r. Thus r satisfies $X \twoheadrightarrow Y$. If Ψ_r makes X true, it must also make Y true or Z true. It follows that $t_1(X) = t_2(X)$ and either $t_1(Y) = t_2(Y)$ or $t_1(Z) = t_2(Z)$. Hence, by Lemma 14.3, $X \twoheadrightarrow Y$ is satisfied.

(only if) Suppose r satisfies $X \twoheadrightarrow Y$. If $t_1(X) \neq t_2(X)$, then Ψ_r makes X false, and hence makes $X \Rightarrow (Y \vee Z)$ true. If $t_1(X) = t_2(X)$, then r actively satisfies $X \twoheadrightarrow Y$. By Lemma 14.3, $t_1(Y) = t_2(Y)$ or $t_1(Z) = t_2(Z)$. It follows that Ψ_r makes Y true or Z true, so Ψ_r makes $X \Rightarrow (Y \vee Z)$ true.

Lemma 14.5 Let $r = \{t_1, t_2\}$ and $s = \{u_1, u_2\}$ be two-tuple relations over scheme R. Suppose for every attribute $A \in R$, $t_1(A) = t_2(A)$ implies $u_1(A) = u_2(A)$. If $X \twoheadrightarrow Y$ holds actively in r, it also holds actively in s.

Proof Left as Exercise 14.6.

The next lemma is the analogue of Lemma 14.2 with MVDs added. The proof is more complex than that for Lemma 14.2, since it is not the case that any subrelation of a relation satisfying an MVD also satisfies the MVD.

Lemma 14.6 Let r be a relation on scheme R, let $\mathbf{C}$ be a set of FDs and MVDs on R, and let c be a single FD or MVD on R. If r satisfies $\mathbf{C}$ and violates c, then some two-tuple subrelation s of r satisfies $\mathbf{C}$ and violates c.

Proof **Case 1** (c is an FD) Assume that c is $X \rightarrow A$. (Why is it permissible to assume that c has a single attribute on the right side?) By Lemma 14.2, there is at least one two-tuple subrelation of r that violates $X \rightarrow A$. Let s be one such relation, chosen to actively satisfy as many MVDs in $\mathbf{C}$ as any other such subrelation.

Relation s satisfies all the FDs in **C**. Let $W \twoheadrightarrow Y$ be an arbitrary MVD in **C**, where W, Y, and Z partition R. Call the two tuples in s u_1 and u_2. If $u_1(W) \neq u_2(W)$, then s satisfies $W \twoheadrightarrow Y$. Suppose $u_1(W) = u_2(W)$ but s does not actively satisfy $W \twoheadrightarrow Y$. We look at u_1 and u_2 in terms of their W, Y, and Z components. Let $u_1 = \langle w(W), y(Y), z(Z) \rangle$ and $u_2 = \langle w(W), y'(Y), z'(Z) \rangle$, where $y \neq y'$ and $z \neq z'$. Since s violates $X \rightarrow A$, u_1 and u_2 agree on X but not on A, so $A \in Y$ or $A \in Z$. Assume $A \in Y$.

Consider the relation $q(R)$ consisting of the two tuples $v_1 = \langle w, y, z \rangle$ and $v_2 = \langle w, y', z \rangle$. Relation q is a subrelation of r since r satisfies $W \twoheadrightarrow Y$. Now q violates $X \rightarrow A$ (why?), but actively satisfies $W \twoheadrightarrow Y$. By Lemma 14.5, q actively satisfies any MVD that s actively satisfies. The existence of q contradicts the choice of s, since q actively satisfies more MVDs from **C** than s does. The assumption that s violated $W \twoheadrightarrow Y$ must be incorrect. We conclude that s is the desired two-tuple relation.

Case 2 (c is an MVD) Let c be $X \twoheadrightarrow Y$, where X, Y, Z is a partition of R. We know that whatever two-tuple subrelation of r we choose will satisfy the FDs in **C**. Consider tuples in R broken into X, Y, and Z components. Since r violates $X \twoheadrightarrow Y$, there are tuples $t_1 = \langle x, y, z \rangle$ and $t_2 = \langle x, y', z' \rangle$ such that either $\langle x, y', z \rangle$ or $\langle x, y, z' \rangle$ is missing from r. Choose s to be such a pair $\{t_1, t_2\}$ where the number of MVDs that s actively satisfies is maximized.

Suppose s does not satisfy all the MVDs in **C**. Let $U \twoheadrightarrow V$ be an MVD in **C** that s does not satisfy. Let $W = R - UV$. We now look at t_1 and t_2 broken into U, V, and W components. Let $t_1 = \langle u, v, w \rangle$ and let $t_2 = \langle u, v', w' \rangle$. Define

$$V^* = \{A \in V | v(A) \neq v'(A)\}$$

and

$$W^* = \{A \in W | w(A) \neq w'(A)\}.$$

Neither V^* or W^* is empty, or else s would satisfy $U \twoheadrightarrow V$. Relation r satisfies **C**, so $t_3 = \langle u, v', w \rangle$ and $t_4 = \langle u, v, w' \rangle$ must be tuples in r. We consider two subrelations of r, $q_1 = \{t_1, t_3\}$ and $q_2 = \{t_2, t_4\}$. Since $v \neq v'$ and $w \neq w'$, q_1 and q_2 are two-tuple relations. Note that t_1 and t_3 disagree only on V^*, that t_2 and t_4 disagree only on W^*, and that t_1 and t_2 disagree on V^*W^*.

We show that q_1 and q_2 both actively satisfy more MVDs from **C** than s. The pairs t_1, t_3 and t_2, t_4 both agree in every attribute in which the pair t_1, t_2

agrees. Thus, by Lemma 14.5, q_1 and q_2 actively satisfy every MVD that s actively satisfies. Further, q_1 and q_2 both actively satisfy $U \twoheadrightarrow V$.

If either q_1 or q_2 violates $X \twoheadrightarrow Y$, we are done, for we then have a contradiction to the choice of s. Suppose q_1 and q_2 both satisfy $X \twoheadrightarrow Y$. They both must then actively satisfy $X \twoheadrightarrow Y$, since t_1, t_2, t_3, and t_4 all have the same X-value. By Lemma 14.3, since q_1 actively satisfies $X \twoheadrightarrow Y$, t_1 and t_3 agree on X, and they also agree on either Y or Z. If they agree on Y, then $V^* \subseteq Z$, since t_1 and t_3 only disagree on V^*. If t_1 and t_3 agree on Z, then $V^* \subseteq Y$. A similar argument on q_2 shows that $W^* \subseteq Z$ or $W^* \subseteq Y$.

If $V^* \subseteq Y$ and $W^* \subseteq Y$, then t_1 and t_2 agree on all of Z, which means s satisfies $X \twoheadrightarrow Y$. Thus, that combination of containments cannot hold. Similarly, $V^* \subseteq Z$ and $W^* \subseteq Z$ cannot hold simultaneously. The only remaining possibility for the combination is $V^* \subseteq Y$ and $W^* \subseteq Z$, or $V^* \subseteq Z$ and $W^* \subseteq Y$. By symmetry, we only examine the first possibility. We have $t_3 = \langle x, y', z \rangle$ and $t_4 = \langle x, y, z' \rangle$. One of $\langle x, y', z \rangle$ and $\langle x, y, z' \rangle$ was assumed missing from r in the construction of s, but t_3 and t_4 are both supposed to be in r. We have a contradiction to the supposition that q_1 and q_2 satisfy $X \twoheadrightarrow Y$.

We conclude that at least one of q_1 and q_2 violates $X \twoheadrightarrow Y$, which contradicts the choice of s. Our assumption that s violated some MVD in **C** must have been incorrect, so s is the desired two-tuple relation.

Theorem 14.2 Let **C** be a set of FDs and MVDs over scheme R and let c be a single FD or MVD over r. The following conditions are equivalent.

1. **C** implies c.
2. **C** implies c in the world of two-tuple relations.
3. **C** implies c when dependencies are interpreted as logical formulas.

Proof The proof is similar to that of Theorem 14.1. Lemmas 14.4 and 14.6 take the place of Lemmas 14.1 and 14.2. The details are left to the reader (see Exercise 14.7).

14.1.4 Nonextendibility of Results

The correspondence between logic and data dependencies cannot be extended to include JDs or embedded MVDs. This limitation is not too suprising when we note that implication for these types of dependencies is not the same in the world of two-tuple relations as it is for regular relations (see Exercise 14.9).

We show that no extension of our correspondence works for JDs; the corresponding proof for EMVDs is left as Exercise 14.10.

Suppose the correspondence between logic and data dependencies extended to JDs. Consider the JD $*[AB, BC, AC]$. Suppose that this JD has a corresponding logical formula f. Since $*[AB, BC, AC]$ follows from $A \twoheadrightarrow B$, $A \Rightarrow (B \vee C)$ should imply f. Likewise, considering $B \twoheadrightarrow C$, $B \Rightarrow (A \vee C)$ should imply f. Consider any truth assignment Ψ for $\{A, B, C\}$. If $\Psi(A) =$ *false*, then Ψ makes $A \Rightarrow (B \vee C)$ true, and so Ψ makes f true. If $\Psi(A) =$ *true*, then Ψ makes $B \Rightarrow (A \vee C)$ true, so it also makes f true. Formula f must be a tautology, since every truth assignment makes it true. However, $*[AB, BC, AC]$ does not always hold. We were in error assuming $*[AB, BC, AC]$ has a corresponding logical formula that is consistent with the logical interpretation that we gave to MVDs.

14.2 MORE DATA DEPENDENCIES

Why do we need more types of data dependencies? Are not FDs, MVDs, JDs and their embedded versions enough? There is some evidence that these dependencies do not form a natural class, that there is something missing. The class of sets of instances definable with FDs, MVDs, and JDs is not closed under projection. In Section 9.3 we saw that for a set F of FDs over scheme R, $\pi_X(SAT(F))$ cannot be described always as $SAT(F')$ for a set F' of FDs over X. We saw that a similar remark holds for MVDs. The remark also applies to JDs (see Exercise 14.11). Another problem is that there are no complete sets of inference axioms for embedded MVDs, and there is no known complete set of inference axioms for JDs. It has been shown that no such set of axioms exists for EMVDs, and there is evidence that no such rules exist for JDs. (See Bibliography and Comments at the end of this chapter.) While we do have the chase for determining implication of JDs, the fast implication algorithms for FDs and MVDs are based on inference axioms and not the chase. Also, the chase is an unwieldy tool for generating all dependencies of a given type that are implied by a set of dependencies.

The hope in studying larger classes of data dependencies is that a more general class will be found that contains FDs and JDs, and also avoids the problems mentioned above. Template dependencies and generalized functional dependencies are attempts to find such a class of more general dependencies. Template dependencies generalize JDs, and generalized functional dependencies generalize (you guessed it) FDs. These more general dependencies handle the first problem above. Sets of instances defined by satisfaction of these dependencies are closed under projection, as we shall see. These

generalized dependencies do not do quite as well in solving the inference axiom problem. A complete set of inference axioms exists for template dependencies, but only for "infinite implication." That is, the axioms are complete for reasoning about implication in situations where relations are allowed to be infinite. We shall see that the inference axioms are not complete for implication where relations are restricted to be finite. We shall also see that there are an infinite number of inequivalent template dependencies over schemes of sufficiently large size, so it is generally not possible to generate all the template dependencies implied by a set of template dependencies.

The chase computation can be extended to template dependencies with a few modifications, but the tableaux that result from chasing with template dependencies can be infinite. Even though we are guaranteed to generate the "winning row" in the chase after a finite amount of time (if an implication holds), the chase cannot serve as a basis for a decision procedure for template dependency implication. One might imagine a decision procedure that simultaneously runs the chase to test a given implication and looks for counterexamples to the implication. This plan fails because there can be an infinite counterexample to a relation but no finite counterexample. It is not likely that any modification of this plan will work, for the implication problem has been shown undecidable for a slight generalization of template dependencies.

There are some subcases of the implication problem for template dependencies that are decidable. One is where we seek only implication by a single template dependency. Another case is where the template dependencies are not embedded. In both cases the chase computation terminates. In the latter case, the chase computation terminates even when generalized functional dependencies are added.

14.2.1 Template Dependencies

A template dependency is essentially a statement that a relation is invariant under a certain tableau mapping. When written down, a template dependency looks like a tableau with a special row at the bottom, somewhat like an upside-down tableau query. The special row is called the *conclusion row*; the other rows are the *hypothesis rows*. For a relation r to satisfy a template dependency, whenever there is a valuation ρ that maps the hypothesis rows to tuples in r, ρ also must map the conclusion row to a tuple in r. There is a slight complication to this informal definition, to handle variables in the conclusion row that do not appear in the hypothesis rows.

Example 14.7 Figure 14.5 shows a template dependency τ over scheme $A\ B\ C$. The hypothesis rows are w_1-w_4; w is the conclusion row. Relation r in Figure

14.6 does not satisfy τ, since the valuation ρ that maps w_i to t_i, $1 \leq i \leq 4$, does not map w to any tuple in r. Adding a tuple $t_5 = \langle 1\ 3\ 6 \rangle$ to r makes r satisfy τ, although this fact is tedious to check.

$\tau(A$	B	$C)$
w_1 a	b	c'
w_2 a	b'	c'
w_3 a	b'	c
w_4 a'	b	c
w a	b	c

Figure 14.5

$r(A$	B	$C)$
t_1 1	3	5
t_2 1	4	5
t_3 1	4	6
t_4 2	3	6

Figure 14.6

We now provide a formal definition for a template dependency and its satisfaction. While template dependencies are not exactly the same as tableaux, they are sufficiently similar so that tableau concepts, such as *valuation* and *containment mapping*, apply to the set of hypothesis rows in a template dependency. In the following definition, when we refer to a *row over scheme* R, we mean a tuple of abstract symbols or variables, as in tableaux. We do not make the distinguished-nondistinguished distinction on variables that we did with tableaux, however.

Definition 14.2 A *template dependency* (TD) on a relation scheme R is a pair $\tau = (T, w)$ where $T = \{w_1, w_2, \ldots, w_k\}$ is a set of rows on R, called the *hypothesis rows*, and w is a single row on R, called the *conclusion row*. A relation $r(R)$ *satisfies* TD τ if for every valuation ρ of T such that $\rho(T) \subseteq r$, ρ can be extended in such a way that $\rho(w) \in r$. TD τ is *trivial* if it is satisfied by every relation over R.

TDs are written as shown in Figure 14.5, with the conclusion row at the bottom, separated from the hypothesis rows by a line. For variables, we usually use lowercase letters corresponding to the attribute name or symbol, with the unprimed or unsubscripted version appearing in the conclusion row.

While TDs almost look like tableau mappings turned upside down, there are two differences:

1. A variable in the conclusion row need not appear in any hypothesis row.
2. Variables are not restricted to a single column.

To elaborate on point 1, a TD τ on scheme R where every variable in the conclusion row appears in some hypothesis row is called *full*. Let $w_1, w_2, \ldots, w_k$ be the hypothesis rows of τ and let w be the conclusion row. We say τ is S-*partial*, where S is the set

$$\{A \in R \mid w(A) \text{ appears in one of } w_1, w_2, \ldots, w_k\}.$$

Naturally, if $S = R$, then τ is full. If $S \neq R$, we say τ is *strictly partial*. If τ is S-partial, the conclusion row of τ specifies a tuple with certain values on the attributes in S, but it puts no restriction on the values for attributes in R-S.

Example 14.8 The TD τ on $A\ B\ C$ in Figure 14.7 is $A\ B$-partial.

$\tau(A$	B	$C)$
a'	b'	c'
a'	b	c''
a	b'	c''
a	b	c

Figure 14.7

To elaborate on point 2, a TD where each variable appears in exactly one column is called a *typed* TD. If some variable appears in multiple columns, the TD is called *untyped*. The TDs in Figures 14.5 and 14.7 are typed. For the rest of our treatment of template dependencies, we shall assume that all TDs are typed, unless they are explicitly said to be untyped.

Example 14.9 Figure 14.8 shows an untyped TD τ. This TD assumes that $dom(A) = dom(B)$ and asserts that a relation is transitively closed (when considered as a binary relation in the mathematical sense).

$\tau(A$	$B)$
a	b
b	c
a	c

Figure 14.8

Any join dependency, full or embedded, can be represented as a TD (see Exercise 14.15).

Example 14.10 The MVD $A\ B \twoheadrightarrow C$ over relation scheme $A\ B\ C\ D\ E$ is equivalent to the TD τ in Figure 14.9. TD τ asserts that if a relation has two tuples t_1 and t_2 that agree on $A\ B$, it must also have a tuple t_3 such that $t_3(A\ B\ C) = t_1(A\ B\ C)$ and $t_3(A\ B\ D\ E) = t_2(A\ B\ D\ E)$, which is just a way of stating that the relation satisfies $A\ B \twoheadrightarrow C$.

$\tau(A$	B	C	D	$E)$
a	b	c	d'	e'
a	b	c'	d	e
a	b	c	d	e

Figure 14.9

Not every TD corresponds to a JD or EJD. First note that there are an infinite number of different TDs over a given relation scheme, while there are only a finite set of JDs over the same scheme. It could be that many of the TDs are equivalent, which is certainly the case for one-attribute schemes. For schemes with three or more attributes, we shall see that there are an infinite number of inequivalent TDs. Therefore, some of these TDs must not be equivalent to any JD. For schemes with two attributes, there are only three distinct TDs (see Exercise 14.19). The next example shows that one of those TDs is not equivalent to any JD.

Example 14.11 Consider the TD τ on scheme $A\ B$ in Figure 14.10. This TD is not trivial, for it is easy to construct a relation that does not satisfy τ. (Take the relation consisting of just the hypothesis rows of τ.) The only nontrivial JD over $A\ B$ is $*[A, B]$. However, τ is not equivalent to $*[A, B]$. Relation r in Figure 14.11 satisfies τ but not $*[A, B]$. To see that r satisfies τ, note that any valuation from τ to r maps w_1, w_2, and w_3 to the same tuple of r, and so maps w to that tuple.

	$\tau(A$	$B)$
w_1	a	b'
w_2	a'	b'
w_3	a'	b
w	a	b

Figure 14.10

$r(A$	$B)$
1	3
2	4

Figure 14.11

14.2.2 Examples and Counterexamples for Template Dependencies

In this section we see that there is a strongest TD over any scheme, as well as a weakest, nontrivial, full TD. We shall also exhibit a relation that obeys every strictly partial TD, but violates every full TD, thus showing that a set of strictly partial TDs cannot imply a nontrivial full TD.

Theorem 14.3 For any relation scheme R, there is a strongest TD τ on R. That is, any relation $r(R)$ that satisfies τ also satisfies any other TD τ' on R.

Proof The TD τ we want states that a relation is a column-wise Cartesian product. For example, on the scheme $A\ B\ C\ D$, the Cartesian product TD is shown in Figure 14.12. Any relation that is a Cartesian product satisfies every TD (see Exercise 14.21).

$\tau(A$	B	C	$D)$
a	b_1	c_1	d_1
a_1	b	c_2	d_2
a_2	b_2	c	d_3
a_3	b_3	c_3	d
a	b	c	d

Figure 14.12

While there is no weakest nontrivial TD in general (see Exercise 14.22b), there is a weakest full TD on any scheme with 2 or more attributes. Note that there are only trivial TDs over a scheme with a single attribute.

Theorem 14.4 For any relation scheme R with two or more attributes, there is a weakest nontrivial full TD τ on R. That is, if τ' is another nontrivial full TD on R, any relation $r(R)$ that satisfies τ' also satisfies τ.

Proof Assume $R = A_1 A_2 \cdots A_n$, where $n \geq 2$. For each A_i, τ has two variables in the A_i-column, a_i and b_i. Let $\tau = (T, w)$, where T contains every possible row of a's and b's, except the row of all a's. The conclusion row, w, is the row of all a's. Figure 14.13 shows TD τ when $R = A_1 A_2 A_3$.

Let $\tau' = (T', w')$ be another nontrivial full TD over R. The conclusion row w' cannot appear in T' (see Exercise 14.23). We show that τ' is stronger than τ by exhibiting a containment mapping ψ from τ' to τ. That is, ψ maps variables in τ' to variables in τ in such a way that $\psi(T') \subseteq T$ and $\psi(w') = w$. Thus, for any relation $r(R)$ and for any valuation mapping ρ on T such that $\rho(T) \subseteq r$, $\rho' = \rho \circ \psi$ is a valuation mapping for T' such that $\rho'(T') \subseteq r$. If r satisfies τ', then r contains $\rho'(w')$, which is the same as $\rho(w)$, so r satisfies τ.

For ease of notation while dealing with ψ, assume that the variables in τ' are renamed so that $w = w'$. For the A_i-column of τ', let ψ map a_i to a_i, and let it map any other variable to b_i. Clearly, $\psi(w') = w$. For any hypothesis row v of T', $\psi(v)$ will be a row of a's and b's (other than the row of all a's), so $\psi(v) \in T$. Therefore $\psi(T') \subseteq T$, and we are finished.

$\tau(A_1$	A_2	$A_3)$
b_1	b_2	b_3
b_1	b_2	a_3
b_1	a_2	b_3
b_1	a_2	a_3
a_1	b_2	b_3
a_1	b_2	a_3
a_1	a_2	b_3
a_1	a_2	a_3

Figure 14.13

Theorem 14.5 Let R be a relation scheme, and let $\mathbf{C}$ be a set of strictly partial TDs over R. $\mathbf{C}$ does not imply any nontrivial full TD over R.

Proof Let $R = A_1 A_2 \cdots A_n$. Consider the relation $r(R)$ that contains all tuples of n 0's and 1's, except the row of all 1's. Figure 14.14 shows r for $R = A_1 A_2 A_3$. Since the projection of r onto any proper subset of R is a Cartesian product relation, r satisfies every strictly partial TD. However, r violates the weakest nontrivial full TD, constructed in the proof of Theorem 14.3. The valuation ρ that maps a_i to 1 and b_i to 0, $1 \leq i \leq n$, maps the weakest TD into r, but ρ cannot be extended to the row of all a's. It follows that r violates any nontrivial full TD over R, and so r serves as a counterexample to any proposed implication of a full nontrivial TD by **C**.

$r(A_1$	A_2	$A_3)$
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0

Figure 14.14

14.2.3 A Graphical Representation for Template Dependencies

Testing whether or not a relation r satisfies a TD τ is a tedious task at best, since it involves finding all valuations from the hypothesis rows of τ into the tuples of r. In this section we introduce a graphical representation for relations and TDs that makes finding such valuations somewhat easier, at least for small examples done by hand. Also, the graphical notation removes some extraneous details, and so gives a more concise method for expressing TDs in most cases.

The actual values in relations and the actual variables in TDs are of no importance in testing if a relation satisfies a TD. What is important is equalities among values and among variables. We use undirected graphs to represent relations and TDs. The nodes stand for tuples or rows, as the case may be; labeled edges between nodes indicate where two tuples or rows match.

Definition 14.3 Let r be a relation on scheme $R = A_1 A_2 \cdots A_n$. The *graph of* r, denoted G_r, is an undirected graph with labeled edges constructed as follows. The nodes of G_r are the tuples of r. For two tuples t_1 and t_2 in r, there is an edge (t_1, t_2) in G_r exactly when t_1 and t_2 agree on some attribute

in R. The edge (t_1, t_2) is labeled by the set of attributes on which t_1 and t_2 agree.

Example 14.12 Let r be the relation in Figure 14.6. Figure 14.15 shows G_r, the graph of r. In drawing graphs of relations, we remove any edge from a node to itself (there is such an edge for every node), and sometimes omit edges that can be inferred by transitivity. Thus, we could just as well depict G_r as in Figure 14.16.

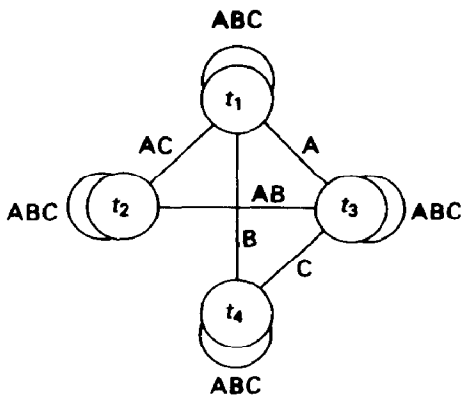

Figure 14.15

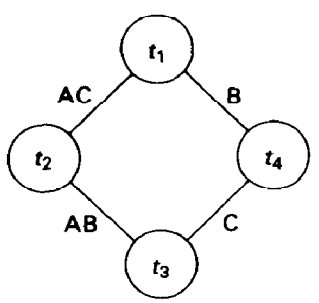

Figure 14.16

The graph for a TD τ, denoted G_τ, is defined similarly, except that we label the node for the conclusion row with a *.

Example 14.13 If τ is the TD from Figure 14.5, then G_τ is shown in Figure 14.17. Again, we omit self-loops and some edges implicit by transitivity.

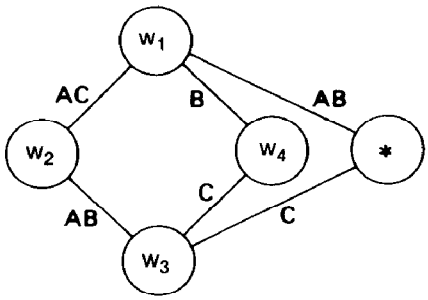

Figure 14.17

Since the order of tuples in a relation or the order of the hypothesis rows in a TD is unimportant, we shall sometimes label the nodes in the graph for a relation or TD arbitrarily. We make the proviso that the conclusion row in a TD will always be labeled *. Apart from *, we need not label the nodes at all, actually. We can also go from graphs to TDs. The transformation gives a unique TD, up to a one-to-one renaming of variables, as long as the scheme is given.

Example 14.14 The graph G in Figure 14.18 gives rise to the TD τ in Figure 14.19, when the scheme is assumed to be $A\ B\ C$.

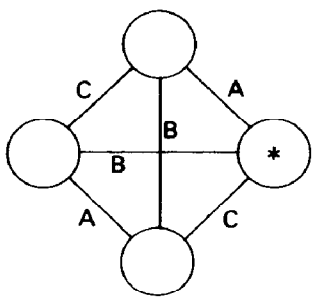

Figure 14.18

$\tau(A$	B	$C)$
a	b'	c'
a'	b	c'
a'	b'	c
a	b	c

Figure 14.19

We now define an analogue for valuation in terms of labeled graphs.

Definition 14.4 Let $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$ be two undirected graphs whose edges are labeled with subsets of some set L. A mapping $h: N_1 \to N_2$ is a *label-preserving homomorphism* (lp-homomorphism) of G_1 to G_2 if for any edge $e = (v, w)$ in E_1, if L_1 is the label of e and L_2 is the label of $(h(v), h(w))$ in G_2, then $L_1 \subseteq L_2$.

Example 14.15 Let G_r be the graph in Figure 14.16 and let G_τ be the graph in Figure 14.17. The function h_1 defined as

$$\begin{aligned} h_1(w_1) &= t_1 \\ h_1(w_2) &= t_2 \\ h_1(w_3) &= t_2 \\ h_1(w_4) &= t_1 \\ h_1(*) &= t_1 \end{aligned}$$

is an lp-homomorphism from G_τ to G_r. (Recall that there are self-loops for all the nodes in G_r, although they are omitted from the figure.) The mapping h_2 defined as

$$\begin{aligned} h_1(w_1) &= t_1 \\ h_1(w_2) &= t_3 \\ h_1(w_3) &= t_3 \\ h_1(w_4) &= t_2 \\ h_1(*) &= t_2 \end{aligned}$$

is not an lp-homomorphism from G_r to G_τ, since, for example, (w_1, w_4) has label B in G_τ, but $(h(w_1), h(w_2)) = (t_1, t_2)$ has label A C.

We can express satisfaction of a TD by a relation in terms of lp-homomorphism between their graphs. In the following theorem, $G_\tau - \{*\}$ means the graph of TD τ without the node for * or its connecting edges.

Theorem 14.6 Let r be a relation over scheme R with graph G_r. Let τ be a TD over R with graph G_τ. Relation r satisfies τ if and only if for any lp-homomorphism h from $G_\tau - \{*\}$ to G_r can be extended to an lp-homomorphism from all of G_τ to G_r.

Proof Left to the reader (see Exercise 14.27).

Example 14.16 Let G_r be the graph in Figure 14.16 and let G_τ be the graph in Figure 14.17. (The same graphs as in the last example.) The mapping h defined as

$$h(w_1) = t_1$$
$$h(w_2) = t_2$$
$$h(w_3) = t_3$$
$$h(w_4) = t_4$$

is an lp-homomorphism from $G_\tau - \{*\}$ to G_r. Any extension of h to all of G_τ is not an lp-homomorphism. Suppose we extend h so that $h(*) = t_4$. This extension is not an lp-homomorphism since $(w_4, *)$ has label $A\ B$ in G_τ, but $(h(w_1), h(*)) = (t_1, t_4)$ has only label A in G_r.

We now give an application of graphical representation of TDs in proving that there are an infinite number of inequivalent full TDs over a scheme of three attributes. We first need the following lemma, which also is proved with the use of graphical representations.

Lemma 14.7 Let τ be a TD over relation scheme R. Let w' be a row over R such that if w' mentions any variable in the conclusion row of τ, some hypothesis row of τ also contains that variable. Form TD τ' by adding w' to the hypothesis rows of τ. Then τ implies τ'.

Proof By the choice of w', we can draw the graph $G_{\tau'}$ as G_τ with a node w' added such that no edges connect w' and *. Consider an arbitrary relation $r(R)$. Any lp-homomorphism h' from $G_{\tau'} - \{*\}$ to G_r can be restricted to an lp-homomorphism h from $G_\tau - \{*\}$ to G_r. If r satisfies τ, then h can be extended to all of G_τ. By the form of the graph $G_{\tau'}$, h' can therefore be extended to all of $G_{\tau'}$. Hence, if r satisfies τ, it also satisfies τ'.

Theorem 14.7 (progressively weaker chain) There is an infinite sequence τ_1, τ_2, τ_3, ... of full TDs such that τ_i implies τ_{i+1} for $i \geq 1$ and no two TDs in the sequence are equivalent.

Proof Consider the infinite graph G in Figure 14.20. For $i \geq 1$, let G_i be the sub-graph of G on nodes $\{*, 1, 2, \ldots, i + 1\}$ and let τ_i be the TD corresponding to G_i. By Lemma 14.7, we have that τ_i implies τ_{i+1} for $i \geq 1$. To complete this proof, we need only show that no pair τ_i, τ_{i+1} of consecutive TDs are equivalent. We do so by exhibiting a relation r that violates τ_i but satisfies τ_{i+1}.

We construct r by treating the hypothesis rows of τ_i as a relation. Note that the graph G_r for r is just G restricted to the nodes $\{1, 2, \ldots, i+1\}$. Relation r is easily seen to violate τ_i. The mapping h defined as $h(j) = j$ for $1 \leq i \leq i+1$ is an lp-homomorphism from $G_i - \{*\}$ to G_r that cannot be extended to *.

We now must show that r satisfies τ_{i+1}. Let h be an arbitrary lp-homomorphism from $G_{i+1} - \{*\}$ to G_r. We prove that h can be extended to all of G_{i+1}. Since $G_{i+1} - \{*\}$ has one more node than G_r, h must map two nodes of $G_{i+1} - \{*\}$ to the same node of G_r. Suppose an odd-numbered node m and an even-numbered node n of $G_{i+1} - \{*\}$ get mapped to the same node of G_r. Node m agrees on A with all odd-numbered nodes of $G_{i+1} - \{*\}$ and n agrees on A with all even-numbered nodes. Since $h(m) = h(n)$, $h(j)$ must agree with $h(k)$ on A for any nodes j and k in $G_{i+1} - \{*\}$. In particular, $h(1)$ and $h(2)$ agree on A, so h can be extended to G_{i+1} by letting $h(*) = h(2)$.

We now show that a contradiction arises if we assume that h never maps an odd-numbered node and an even-numbered node of $G_{i+1} - \{*\}$ to the same node in G_r. Let $h(1) = j$. Consider the case where j is odd. Since $h(2)$ must agree with $h(1)$ on B and we assume $h(1) \neq h(2)$, $h(2)$ is forced to be $j+1$. For $h(3)$, since $h(2)$ and $h(3)$ must agree on C, but $h(2) \neq h(3)$, we must have $h(3) = j+2$. Continuing in this manner, we see that $h(k) = j + k - 1$ for $1 \leq k \leq i+2$. However, we must then have $h(i+2) = j + i + 1 \geq i + 2$, which cannot happen since $i+1$ is the largest-numbered node in G_r.

In the case where j is even, we can show that $h(k) = j - k + 1$ by a similar argument. We again run into a contradiction, since we must then have $h(i+2) = j - i - 1$, which is less than 1 because j is no larger than $i+1$.

We see that h can always be extended to G_{i+1}. Hence, τ_{i+1} satisfies r, and we have shown τ_i and τ_{i+1} inequivalent.

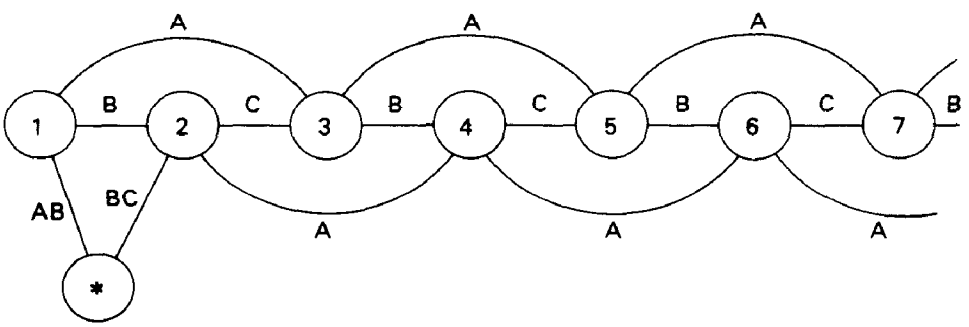

Figure 14.20

14.2.4 Testing Implication of Template Dependencies

In this section we take a short look at problems that arise in computing implications of TDs. The first problem is that certain implications holding for finite relations do not hold when relations are allowed to be infinite. Thus, it is unlikely that a complete set of TD inference axioms exists for finite relations, although such a set exists for arbitrary relations. The next theorem shows that implication is not the same for finite relations and arbitrary relations. By *arbitrary* relations we mean relations that may be finite or infinite.

Theorem 14.8 There is a set **C** of TDs and a single TD τ such that any finite relation that satisfies **C** also must satisfy τ, but there is an infinite relation that satisfies **C** yet violates τ.

Proof The proof is quite long. We sketch the proof here and leave the details to the reader (see Exercise 14.29).

Let $\mathbf{C} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ where τ_i corresponds to graph G_i, $1 \leq i \leq 4$, in Figures 14.21-24. There is a system behind the TDs in **C**. We interpret the graph G_r of a relation r as a directed graph D_r. A subgraph of G_r that matches Figure 14.25 is interpreted as the directed edge $t_1 \rightarrow t_3$ in D_r. TDs τ_1 and τ_2 together say that if D_r has an edge $u \rightarrow v$, then it also has an edge $v \rightarrow w$, for some w. That is, every node in D_r with an incoming edge also has an outgoing edge. (A node with no incoming edges is called a *sink*.) TD τ_3 basically forces D_r to be transitively closed. TD τ_4 comes into play when D_r has a self-loop edge $u \rightarrow u$. TD τ corresponds to the graph G in Figure 14.26.

The property from graph theory that is the mechanism behind this proof is that any finite directed graph that is transitively closed and has no sinks must have a self-loop. The property does not hold for infinite graphs. The proof that **C** implies τ for finite relations basically mimics the proof of the property from graph theory. First, the existence of an lp-homomorphism from $G - \{*\}$ to G_r, for some relation r, implies an edge in D_r. The presence of an edge implies a cycle in D_r reachable from the edge, otherwise, some node would be a sink (so r would violate τ_1 or τ_2). Once the cycle is established, transitivity (application of τ_3) provides the self-loop. The self-loop means τ_4 is applicable. The tuple that τ_4 requires in r is also the tuple that τ requires.

The infinite relation that satisfies **C** but violates τ is

$$r = \{\langle i\, i\, j\, 0\rangle \mid 1 \leq i < j\} \cup \{\langle 0\, i\, i\, i\rangle \mid i \geq 1\}.$$

A proof by cases shows that r satisfies each TD in **C**. However, the lp-homomorphism h from G to G_r defined by

$$h(1) = \langle 0\ 1\ 1\ 1 \rangle$$
$$h(2) = \langle 1\ 1\ 2\ 0 \rangle$$
$$h(3) = \langle 0\ 2\ 2\ 2 \rangle$$
$$h(4) = \langle 2\ 2\ 3\ 0 \rangle$$

cannot be extended to *, since we would need $h(*)(A) = h(*)(D) = 0$, and no such tuple exists in r.

G_1

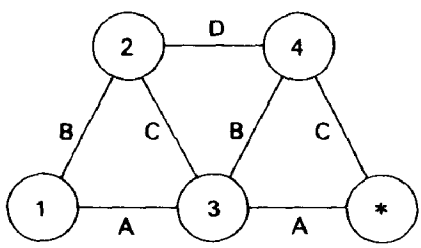

Figure 14.21

G_2

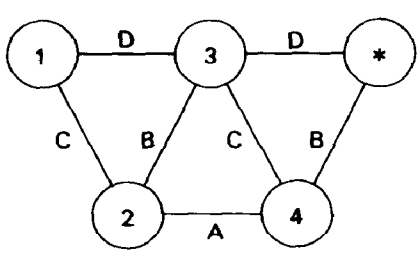

Figure 14.22

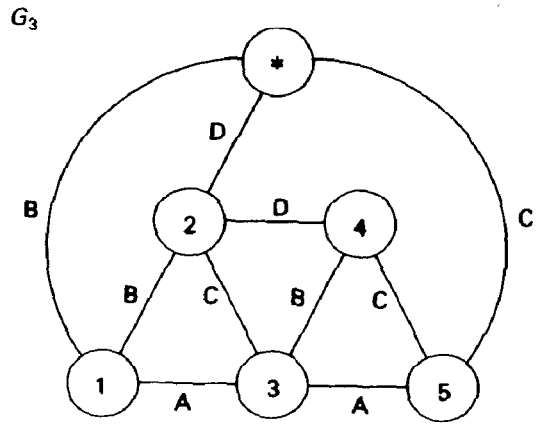

Figure 14.23

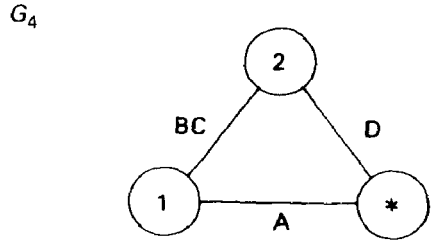

Figure 14.24

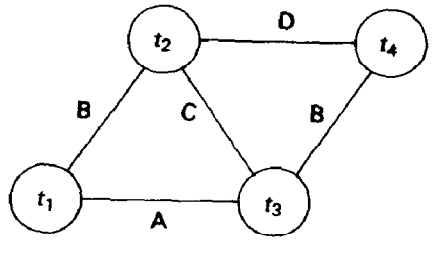

Figure 14.25

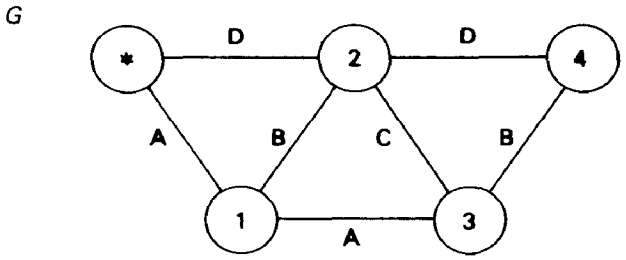

Figure 14.26

There is a special case where the set of TDs implied by a set **C** of TDs is the same for finite and arbitrary relations. If all the TDs in **C** are S-partial for the same S, then if **C** implies τ for finite relations, **C** implies τ for arbitrary relations (see Exercise 14.30). In particular, finite and arbitrary implication are the same for full TDs.

There is a complete set of inference axioms for TDs in arbitrary relations. The axioms are, of course, correct for finite relations, but not complete, by the last theorem. One inference axiom, called *augmentation*, is given by the statement of Lemma 14.7. We give another axiom here, but leave the rest of the set and the proof of completeness as Exercises 14.31 and 14.32.

Weakening If $\tau = (T, w)$ is a TD, and we obtain the row w' from w by changing some variables in w to variables that do not appear in T, then τ implies the TD $\tau' = (T, w')$.

To see that weakening is correct, let r be a relation satisfying τ. If ρ is a valuation such that $\rho(T) \subseteq r$, then $\rho(w) \in r$. We can extend ρ so that $\rho(w') = \rho(w)$, since ρ is unconstrained on any variable in w' not in w. Hence, $\rho(w') \in r$ and so r satisfies τ'.

Example 14.17 The TD τ in Figure 14.27 implies the TD τ' in Figure 14.28 by weakening.

$\tau(A$	B	$C)$
a	b'	c'
a'	b'	c
a'	b	c
a	b	c

Figure 14.27

$\tau(A$	B	$C)$
a	b'	c'
a'	b'	c
a'	b	c
a	b''	c

Figure 14.28

We now look at extending the chase to handle TDs. We want to use a TD $\tau_1 = (T_1, w_1)$ to chase the hypothesis rows of a TD $\tau_2 = (T_2, w_2)$ to see if w_2 can be generated. We need a *T-rule* for chasing with TDs. The definition of the T-rule is straightforward for full TDs.

T-rule Let $\tau = (T_2, w)$ be a full TD on scheme R and let T_2 be a tableau on R. If there exists a valuation ρ on T_1 such that $\rho(T_1) \subseteq T_2$, but $\rho(w)$ is not in T_2, add $\rho(w)$ to T_2.

Example 14.18 Let τ_1 be the TD in Figure 14.29. Let T be the tableau shown in Figure 14.30. To simplify the notation, we shall represent T with just the subscripts of the variables, remembering that the same number stands for different variables in different columns. The simplified version of T is shown in Figure 14.31. Using the valuation ρ from τ_1 to T such that

$$\rho(\langle a\ b'\ c'\rangle) = \langle 3\ 3\ 2\rangle$$
$$\rho(\langle a'\ b\ c'\rangle) = \langle 2\ 3\ 3\rangle$$
$$\rho(\langle a'\ b'\ c\rangle) = \langle 2\ 2\ 2\rangle,$$

we can apply the T-rule for τ_1 to T to add the row

$$\rho(\langle a\ b\ c\rangle) = \langle 3\ 3\ 2\rangle.$$

The resulting TD T' is shown in Figure 14.32.

$\tau_1(A$	B	$C)$
a	b'	c'
a'	b	c'
a'	b'	c
a	b	c

Figure 14.29

$T(A$	B	$C)$
a_1	b_2	c_3
a_3	b_2	c_2
a_2	b_3	c_2
a_2	b_3	c_1
a_2	b_2	c_2

Figure 14.30

$T(A$	B	$C)$
1	2	3
3	2	2
2	3	2
2	3	1
2	2	2

Figure 14.31

$T(A$	B	$C)$
1	2	3
3	2	2
2	3	2
2	3	1
2	2	2
3	3	2

Figure 14.32

A problem arises in extending the T-rule to strictly partial TDs. We have to create new variables in the columns where the conclusion row contains a variable not in any hypothesis row. We extend the T-rule to handle partial TDs.

T-rule (revised) Let $\tau = (T_1, w)$ be an S-partial TD on scheme R, and let T_2 be a tableau on R. If there exists a valuation ρ on T_1 such that $\rho(T_1) \subseteq T_2$, and there is no row in T_2 that matches $\rho(w)$ on S, then add the row w' to T_2, where w' matches $\rho(w)$ on S and $w'(A)$ is a new variable for $A \in R - S$.

Example 14.19 Let τ_2 be the partial TD in Figure 14.33. Using the valuation ρ from τ_2 to tableau T in Figure 14.31 where

$$\rho(\langle a\ b'\ c'\rangle) = \langle 1\ 2\ 3\rangle$$
$$\rho(\langle a'\ b'\ c\rangle) = \langle 3\ 2\ 2\rangle,$$

we can apply the T-rule for τ_2 to add the row $\langle 1\ 4\ 2\rangle$ to T.

τ_2

A	B	C
a	b'	c'
a'	b'	c
a	b	c

Figure 14.33

Since the choice of new variables is arbitrary in the revised T-rule, chasing with partial TDs does not give a unique result. There is little we can do about this problem, and it is not that serious, for we are interested in which combinations of original variables in a tableau get generated during a chase computation. A serious problem is that applying the T-rule can result in an infinite sequence of tableaux where no new combinations of original variables are being generated, but such combinations could be obtained by applying the T-rule in a different manner.

Example 14.20 Let τ_2 be the TD in Figure 14.33 and let τ_3 be the TD in Figure 14.34. Say we start chasing the tableau T in Figure 14.35 (again we show only subscripts). We can first apply the T-rule with τ_3 to generate the row $\langle 3\ 1\ 2\rangle$. We can then apply the T-rule with τ_2, using the new row, to get $\langle 3\ 4\ 1\rangle$. We can continue on indefinitely in this manner, as shown in Figure 14.36, and never generate a row that is $\langle 2\ 1\rangle$ on $B\ C$, although such a row could be generated at any time from T.

τ_3

A	B	C
a'	b	c'
a'	b'	c
a	b	c

Figure 14.34

T

A	B	C
1	1	1
1	2	2

Figure 14.35

$T(A$	B	$C)$	
1	1	1	
1	2	2	
3	1	2	$\cdots \tau_3$
3	4	1	$\cdots \tau_2$
5	4	2	$\cdots \tau_3$
5	6	1	$\cdots \tau_2$
7	6	2	$\cdots \tau_3$

Figure 14.36

We need to guide the chase computation when using the T-rule to insure we generate all possible combinations of original variables. It might seem we could make the restriction that the T-rule may only be applied when it will generate some new combination of original variables. While this restriction would guarantee that the chase process eventually terminates, it can prevent some combinations of original variables from being generated (see Exercise 14.33). Instead, we note that if we repeatedly apply the T-rule for a single TD τ, possibly partial, we will eventually run out of new rows to generate. This observation leads to a more comprehensive rule for chasing with TDs.

T^+-rule Let τ be a TD over scheme R and let T be a tableau on R. Use the T-rule for τ on T as long as it applies.

Example 14.21 If τ_1 is the TD in Figure 14.29 and T is the tableau in Figure 14.31, then using the T^+-rule for τ_1 on T gives the tableau T' in Figure 14.37.

$T'(A$	B	$C)$
1	2	3
3	2	2
2	3	2
2	3	1
2	2	2
3	3	2
3	3	1
3	2	1
2	2	1

Figure 14.37

When chasing under a set of TDs **C**, to ensure that every TD "gets its chance," we make the following definition.

Definition 14.5 Let $\mathbf{C} = \{\tau_1, \tau_2, \ldots, \tau_k\}$ be a set of TDs over scheme R and let T be a tableau on R. *Chasing* T *with* **C** means generating a (possibly infinite) sequence of tableaux $T_0(=T), T_1, T_2, \ldots$, where T_i is obtained by applying the T^+-rule with each of $\tau_1, \tau_2, \ldots, \tau_k$ in sequence to T_{i-1}. The generation of T_i from T_{i-1} is a *stage* in the chase computation. The sequence is finite if it happens that $T_{i-1} = T_i$ for some $i \geq 1$.

The order that the TDs in **C** are applied at each stage in the chase computation is immaterial so far as which combinations of original variables are eventually produced (see Exercise 14.34). Note that if all the TDs in **C** are S-partial for the same S, the chase computation runs for a finite number of stages, because new variables are never introduced in the S-columns. In particular, the chase with full TDs always terminates and, moreover, the final tableau is unique (see Exercise 14.35).

Example 14.22 Let $\mathbf{C} = \{\tau_1, \tau_2\}$, where τ_1 is given in Figure 14.29 and τ_2 is given in Figure 14.31. Tableau T_1 in Figure 14.38 is the result after the first stage of chasing T with **C**.

$T_1(A$	B	$C)$
1	2	3
3	2	2
2	3	2
2	3	1
2	2	2
3	3	2
3	3	1
3	2	1
2	2	1
1	4	2
1	5	1
2	6	3
3	7	3

Figure 14.38

Testing if a set of TDs implies another TD is similar for testing implication of JDs with the chase.

Definition 14.6 Let T_0, T_1, T_2, ... be the sequence of tableaux generated when chasing T_0 with a set of TDs. The *limit* of this sequence is the tableaux

$$T^* = T_0 \cup T_1 \cup T_2 \cup \cdots.$$

Note that T^* might be infinite.

Theorem 14.9 Let **C** be a set of TDs over scheme R and let $\tau = (T, w)$ be an S-partial TD on R. Let T^* be the limit of the sequence $T_0(=T)$, T_1, T_2, ... generated in chasing T with **C**. **C** implies τ on arbitrary relations if and only if T^* contains a row w^* such that $w^*(S) = w(S)$.

Proof We give only a sketch of the proof. For the "if" direction, assume that w^* first appears after the k^{th} stage in the chase computation. That is, w^* is in T_k but not in T_{k-1}. Let r be a relation in $SAT(\mathbf{C})$ and let ρ be a valuation such that $\rho(T) \subseteq r$. Following the chase computation leading up to T_k, we can show that r must contain a tuple t^* such that $\rho(w^*)(S) = t^*(S)$. It follows that $\rho(w)(S) = t^*(S)$ and so r satisfies τ. Notice that the theorem holds in this direction for finite relations instead of arbitrary relations.

For the "only if" direction, we first show that T^*, as a relation, satisfies **C**. If T^* does not contain a row w^* that matches w on S, then T^*, as a relation, is a counterexample to **C** implying τ. Thus, if **C** does imply τ, w^* must exist.

Example 14.23 Let τ be the A C-partial TD in Figure 14.39. Note that the hypothesis rows of τ correspond to the tableau T in Figure 14.30. If **C** is the set of TDs in the last example, we see that **C** implies τ. From that example, we know that chasing the hypothesis rows of τ with **C** gives a row that is $\langle a_1\ c_1 \rangle$ in the A C-columns.

$\tau(A$	B	$C)$
a_1	b_2	c_3
a_3	b_2	c_2
a_2	b_3	c_2
a_2	b_3	c_1
a_2	b_2	c_2
a_1	b_1	c_1

Figure 14.39

14.2.5 Generalized Functional Dependencies

In Section 9.3 we briefly examined the structure of projections of $SAT(F)$ for a set of FDs F. Projecting $SAT(\{A \rightarrow E, B \rightarrow E, CE \rightarrow D\})$ onto the scheme $ABCD$ we came up with the following "curious" dependency that any relation in the projection must satisfy: If t_1, t_2, and t_3 are tuples in $r(ABCD)$ such that

1. $t_1(A) = t_3(A)$
2. $t_1(C) = t_2(C)$
3. $t_2(B) = t_3(B)$

then

4. $t_1(D) = t_2(D)$.

This constraint is an example of a *generalized functional dependency*, which can be written in a notation similar to that for TDs, as shown in Figure 14.40. The rows above the line are again called hypothesis rows. The equality below the line is called simply the *conclusion*. A relation r satisfies this particular generalized functional dependency if any valuation ρ that maps the hypothesis rows into r necessarily has $\rho(d_1) = \rho(d_3)$. We now give a formal definition.

γ

	A	B	C	D
w_1	a_1	b_2	c_1	d_1
w_2	a_2	b_1	c_1	d_2
w_3	a_1	b_1	c_2	d_3

$d_1 = d_3$

Figure 14.40

Definition 14.7 A *generalized functional dependency* (GFD) on a relation scheme R is a pair $\gamma = (T, a = b)$, where T is a set of rows on R, called *hypothesis rows*, and a and b are two variables from the rows in T. The equality $a = b$ is called the *conclusion*. A relation $r(R)$ satisfies the GFD γ if for every valuation ρ of T such that $\rho(T) \subseteq r$, $\rho(a) = \rho(b)$. The GFD γ is *trivial* if it is satisfied by every relation on r; it is *typed* if no variable appears in more than one column of T and a and b come from the same column of T.

We shall assume that all GFDs are typed, henceforth. Figure 14.40 shows how GFDs are written. Not every FD is equivalent to a GFD, but only because GFDs enforce equality in only one column. Any FD with a single attribute on the right side can be expressed as a GFD, so any FD is equivalent to a set of GFDs.

Example 14.24 Consider the FD $A\ B \rightarrow C$ on scheme $A\ B\ C\ D$. Figure 14.41 shows the equivalent GFD for this FD.

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

a	b	c	d
a	b	c'	d'

$c = c'$

Figure 14.41

It is possible to give a complete set of inference axioms for GFDs, but we shall not do so, since such axioms essentially mimic the chase computation. Complete axiomatizations also exist for TDs and GFDs together, although they are for implication on arbitrary relations if partial TDs are allowed. We give one axiom for inferring TDs from FDs.

GT1 Let $X \rightarrow A$ be a nontrivial FD over relation scheme R and let $Y = R - (X\ A)$. This FD implies the TD τ shown in Figure 14.42. Note that we use a little shorthand in τ. For instance, x_1 and x_2 stand for sequences of variables over X that are distinct in every column.

$\tau(X\ \ A\ \ Y)$

w_1	x_1	a_1	y_1
w_2	x_1	a_2	y_2
w_3	x_2	a_2	y_3
w	x_2	a_1	y_3

Figure 14.42

Example 14.25 Figure 14.43 shows a TD τ implied by the FD $A\ B \rightarrow C$ on scheme $A\ B\ C\ D$.

$\tau(A$	B	C	$D)$
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_2	b_2	c_2	d_3
a_2	b_2	c_1	d_3

Figure 14.43

To see that the TD τ in GT1 follows from the FD $X \rightarrow Y$, consider the GFD γ in Figure 14.44 that is equivalent to $X \rightarrow Y$. Let r be an arbitrary relation on R. Let ρ be a valuation that maps w_1-w_3 into r. If r satisfies γ, then $\rho(a_1) = \rho(a_2)$, since the first two rows of γ are the same as the first two rows of τ. It follows that $\rho(w)$ is in τ, since $\rho(w)$ must equal $\rho(w_3)$.

$\gamma(X$	A	$Y)$
w_1 x_1	a_1	y_1
w_2 x_1	a_2	y_2
$a_1 = a_2$		

Figure 14.44

Axiom GT1 can be generalized from FDs to GFDs (see Exercise 14.39). The derived TD can be used in place of the FD when inferring TDs from a set **C** of FDs and TDs. That is, if we form **C**′ by replacing every FD in **C** with the TD given by GT1, **C** and **C**′ imply the same TDs (see Exercise 14.40).

Extending the chase computation to GFDs is fairly straightforward.

G-rule Let $\gamma = (T_1, a = b)$ be a GFD over scheme R. Let T_2 be a tableau over R. For any valuation ρ such that $\rho(T_1) \subseteq T_2$ and $\rho(a) \neq \rho(b)$, identify $\rho(a)$ and $\rho(b)$ in T_2.

The chase computation under a set **C** of GFDs on a tableau T is just the application of the G-rule for GFDs in **C** until no more variables in T can be equated. The computation terminates, because no new rows or variables are introduced in T. Although not proved here, the result is unique given the proper mechanism for renaming variables when identifying them (such as always choosing the one with the lower subscript to replace the other). Since the result of the chase with GFDs is unique, we may denote it as $chase_{\mathbf{C}}(T)$.

Example 14.26 Let $\mathbf{C} = \{\gamma_1, \gamma_2\}$, where γ_1 and γ_2 are as shown in Figures 14.45 and 14.46. Let T be the tableau in Figure 14.47. Again, we give only the subscripts for the variables in T for simplicity, so, for instance, "2" represents different variables in different columns. We first apply the G-rule for γ_1 to T. There is a valuation ρ_1 from the hypothesis rows of γ_1 to T such that

$$\rho(u_1) = w_3$$
$$\rho(u_2) = w_4$$
$$\rho(u_3) = w_5$$

Thus, we identify $\rho(c_1)$ and $\rho(c_3)$, that is, 2 and 3 in the C-column. The result is tableau T' in Figure 14.48. We adopt the rule of replacing higher subscripts with lower ones. We next apply the G-rule for γ_2 to T' using the valuation ρ_2 where

$$\rho(v_1) = w_2$$
$$\rho(v_2) = w_3$$
$$\rho(v_3) = w_4$$

We can equate 2 and 3 in the A-column to obtain the tableau T^* in Figure 14.49. At this point, no more variables can be identified with the G-rule, so $chase_{\mathbf{C}}(T) = T^*$.

γ_1	A	B	C
u_1	a_1	b_2	c_1
u_2	a_1	b_1	c_2
u_3	a_2	b_1	c_3
	$c_1 = c_3$		

Figure 14.45

γ_2	A	B	C
v_1	a_1	b_1	c_1
v_2	a_2	b_1	c_2
v_3	a_3	b_2	c_1
	$a_1 = a_3$		

Figure 14.46

T	A	B	C
w_1	1	3	1
w_2	2	1	2
w_3	3	1	2
w_4	3	2	2
w_5	2	2	3

Figure 14.47

T'	A	B	C
w_1	1	3	1
w_2	2	1	2
w_3	3	1	2
w_4	3	2	2
w_5	2	2	2

Figure 14.48

T^*	A	B	C
w_1	1	3	1
w_2	2	1	2
w_5	2	2	2

Figure 14.49

We now give a theorem that shows how to test implication of GFDs. In the statement of the theorem, saying that two variables are equated in the chase means that one was renamed to the other, or that they both were ultimately renamed to the same third variable.

Theorem 14.10 Let **C** be a set of GFDs over scheme R and let $\gamma = (T, a = b)$ be a single GFD on R. **C** implies γ if and only if a and b are equated when computing $chase_{\mathbf{C}}(T)$.

Here implication means implication over finite relations, which is the same as implication over arbitrary relations for GFDs (see Exercise 14.41).

Proof The proof is similar to previous proofs, so we omit details. If **C** does not imply γ, then $chase_{\mathbf{C}}(T)$ serves as a counterexample. If a and b are equated during the chase computation, then, by following the steps in the

computation, we can show that any valuation ρ from T into a relation r in $SAT(\mathbf{C})$ must have $\rho(a) = \rho(b)$.

Example 14.27 The GFD γ in Figure 14.50 is implied by the set of GFDs $\{\gamma_1, \gamma_2\}$ from the last example, as the chase computation given there shows.

$\gamma(A$	B	$C)$
w_1 a_1	b_3	c_1
w_2 a_2	b_1	c_2
w_3 a_3	b_1	c_2
w_4 a_3	b_2	c_2
w_5 a_2	b_2	c_3

$$a_2 = a_3$$

Figure 14.50

When using the G-rule and T-rule together, we must exercise some care when we identify variables. The T-rule can create new variables; the G-rule might change an original variable to some new variable. Such a change makes conditions for testing implication hard to state and makes it hard to define the tableau to which the chase converges. To avoid such problems, we maintain an ordering on variables in a tableau. The original variables come first in the ordering; new variables are added to the end of the ordering as they are introduced during the chase computation. Whenever the G-rule identifies variables, the variable earlier in the ordering replaces the variable later in the ordering. In the event that the tableau we are chasing is the hypothesis rows of a TD, we assume that the variables appearing in the conclusion row of the TD occur at the very beginning of the ordering. We can then check directly if the conclusion row is ever generated during the chase, without worrying about variable replacements in the conclusion row.

Definition 14.8 Let $\mathbf{C}$ be a set of TDs and GFDs over scheme R. Let T be a tableau on R. *Chasing* T *with* $\mathbf{C}$ means generating a (possibly infinite) sequence of tableaux T_0 $(=T)$, T_1, T_2, ... where T_i is obtained by applying the T^+-rule for each TD in $\mathbf{C}$, followed by applying the G-rule for GFDs in $\mathbf{C}$ as much as possible, to T_{i-1}. The sequence will be finite if it happens that $T_i = T_{i-1}$ for some $i \geq 1$.

Note that the sequence is always finite if all the TDs in $\mathbf{C}$ are S-partial for the same $S \subseteq R$. The T^+-rule can generate only a finite number of tuples

with new combinations of variables in the S-columns. We cannot define the limit of the chase sequence as we did for TDs alone, since rows can be changed from one stage to the next. We need to identify the rows at a given stage that undergo no subsequent changes.

Definition 14.9 Let $T_0, T_1, T_2, \ldots$ be the sequence of tableaux generated by chasing a tableau T with a set **C** of TDs and GFDs. Relative to this sequence, a row w in T_i is *stabilized* if w appears in T_j for $j \geq i$. Let $STABLE(T_i)$ denote all the stable rows in T_i. The *limit* of the sequence $T_0, T_1, T_2, \ldots$ is the tableau

$$T^* = STABLE(T_0) \cup STABLE(T_1) \cup STABLE(T_2) \cdots.$$

(T^* may be infinite.)

An important point is that T^*, as a relation, satisfies **C**. Note that if some set of rows $w_1, w_2, \ldots, w_k$ in T^* gives rise to a violation of a GFD in **C**, then at least one of them is not stabilized, since it would have the offending value changed by the G-rule.

The next Theorem summarizes implication of TDs and GFDs for arbitrary relations.

Theorem 14.11 Let **C** be a set of TDs and GFDs over scheme R and let T be a tableau on R. Let T^* be the limit of the sequence $T_0(=T), T_1, T_2, \ldots$ generated when chasing T with **C**. We have

1. **C** implies the S-partial TD (T, w) if and only if T^* contains a row w^* such that $w^*(S) = w(S)$, and
2. **C** implies the GFD $(T, a = b)$ if and only if a and b are equated in generating T^*.

Proof Left to the reader (see Exercise 14.43).

It follows from the theorem that TDs by themselves imply only trivial GFDs. Note that the tests for implication of FDs and JDs in Sections 8.6.3 and 8.6.4 are specializations of this theorem.

Example 14.28 Let $\mathbf{C} = \{\tau, \gamma\}$, where τ is the TD in Figure 14.51 and γ is the GFD in Figure 14.52. Let T_0 be the tableau in Figure 14.53. Again, we show only subscripts for simplicity. Consider chasing T_0 with **C**. Applying the T^+-rule for τ gives the tableau T' in Figure 14.54. Applying the G-rule

for γ thrice to T' yields the tableau T_1 in Figure 14.55. No further applications of the T$^+$-rule or G-rule are possible, so T_1 is the limit of this chase computation (see Exercise 14.44). From this limit, Theorem 14.11 allows us to conclude that **C** implies the TD τ' in Figure 14.56 and the GFD γ' in Figure 14.57.

$\tau(A$	B	$C)$
a'	b	c'
a'	b'	c
a	b	c

Figure 14.51

$\gamma(A$	B	$C)$
a	b'	c
a'	b	c
a''	b	c'
$a = a''$		

Figure 14.52

$T(A$	B	$C)$
1	1	2
1	2	3
2	1	3
2	2	1

Figure 14.53

$T'(A$	B	$C)$
1	1	2
1	2	3
2	1	3
2	2	1
3	2	2
4	1	1

Figure 14.54

$T_1(A$	B	$C)$
1	1	2
1	2	3
1	1	3
1	2	1
1	2	2
1	1	1

Figure 14.55

$\tau'(A$	B	$C)$
a_1	b_1	c_2
a_1	b_2	c_3
a_2	b_1	c_3
a_2	b_2	c_1
a_1	b_1	c_1

Figure 14.56

$\gamma'(A$	B	$C)$
a_1	b_1	c_2
a_1	b_2	c_3
a_2	b_1	c_3
a_2	b_2	c_1
$a_1 = a_2$		

Figure 14.57

14.2.6 Closure of Satisfaction Classes Under Projection

Recall the notation introduced in Chapter 8 for expressing the class of all relations on a given relation scheme R that satisfy a set of constraints $\mathbf{C}$, $SAT_R(\mathbf{C})$. Also recall that we may extend a relational operator to sets of relations in an element-wise fashion. Thus, if P is a set of relations on scheme R, and $X \subseteq R$, then

$$\pi_X(P) = \{\pi_X(r) | r \in P\}.$$

In Chapter 9 we briefly considered the question of whether $\pi_X(SAT_R(\mathbf{C}))$ necessarily can be expressed as $SAT_X(\mathbf{C}')$ for $\mathbf{C}$ and $\mathbf{C}'$ coming from given classes of dependencies. The answer was "no" if $\mathbf{C}$ and $\mathbf{C}'$ are both FDs or both MVDs.

In this section we show that if $\mathbf{C}$ is TDs and GFDs, then there is always a set $\mathbf{C}'$ of TDs and GFDs such that $\pi_X(SAT_R(\mathbf{C})) = SAT_X(\mathbf{C}')$. However, to make the equality hold, we must interpret $SAT(\mathbf{C})$ as including both finite and infinite relations satisfying $\mathbf{C}$. We shall see that if the TDs in $\mathbf{C}$ are restricted to be full, then $SAT(\mathbf{C})$ may be interpreted as only the finite relations satisfying $\mathbf{C}$.

For $\mathbf{C}$ a set of TDs and GFDs over R and $X \subseteq R$, $\pi_X(\mathbf{C})$ will mean the set of TDs and GFDs over scheme X that are satisfied by all relations in $\pi_X(SAT(\mathbf{C}))$. Clearly, $\pi_X(SAT_R(\mathbf{C})) \subseteq SAT_X(\pi_X(\mathbf{C}))$, so if there is any set $\mathbf{C}'$ such that $\pi_X(SAT_R(\mathbf{C})) = SAT_X(\mathbf{C}')$, $\pi_X(\mathbf{C})$ will also be such a set. It turns out that $\pi_X(\mathbf{C})$ can be infinite, with no equivalent finite set of TDs and GFDs (see Exercise 14.45), so there is no algorithm guaranteed to generate all of $\pi_X(\mathbf{C})$ in general. The next lemma, however, points out some of the dependencies in $\pi_X(\mathbf{C})$.

For a tableau T over scheme R and $X \subseteq R$, let $T(X)$ be the tableau over scheme X obtained by restricting the rows in T to X.

Definition 14.10 Let $\tau = (T, w)$ be a TD on scheme R. If X is a subset of R such that no two rows of T agree in any column in $R - X$, then we define the *restriction of* τ *to* X, denoted $\tau(X)$, as the TD $\tau' = (T(X), w(X))$ with scheme X.

Example 14.29 Let τ be the TD on scheme $A\ B\ C\ D$ given in Figure 14.58. Figure 14.59 shows the TD $\tau' = \tau(A\ B\ C)$.

$\tau(A$	B	C	$D)$
a'	b	c'	d'
a	b'	c'	d
a	b'	c	d''
a	b	c	d

Figure 14.58

$\tau'(A$	B	$C)$
a'	b	c'
a	b'	c'
a	b'	c
a	b	c

Figure 14.59

Lemma 14.8 Let τ be a TD over scheme R such that $\tau(X)$ is defined. If r is a relation in $SAT(\tau)$, then $\pi_X(r)$ is in $SAT(\tau(X))$.

Proof Left to the reader (see Exercise 14.47).

Definition 14.11 Let $\gamma = (T, a = b)$ be a GFD on scheme R. If X is a subset of R such that no two rows of T agree in any column in $R - X$ and a and b occur in some column of X, we define the *restriction of* γ *to* X, denoted $\gamma(X)$, as the GFD $\gamma' = (T(X), a = b)$.

Lemma 14.9 Let γ be a GFD on R such that $\gamma(X)$ is defined. If r is a relation in $SAT(\gamma)$, then $\pi_X(r)$ is in $SAT(\gamma(X))$.

Proof Assume $\gamma = (T, a = b)$. Let ρ be a valuation from $T(X)$ into $\pi_X(r)$. Since no two rows of T agree outside of X, ρ can be extended to a valuation ρ' from T into r. Moreover, ρ can be extended so that $\rho(w(X)) = \rho'(w)(X)$, for every row w of T (hence for every row $w(X)$ of $T(X)$). Since r satisfies γ, $\rho'(a) = \rho'(b)$. Since a and b appear in $T(X)$, $\rho(a) = \rho(b)$, so $\pi_X(r)$ satisfies $\gamma(X)$.

Theorem 14.12 Let $\mathbf{C}$ be a set of TDs and GFDs over scheme R. If X is a subset of R, then

$$\pi_X(SAT_R(\mathbf{C})) = SAT_X(\pi_X(\mathbf{C})).$$

Proof As we remarked previously, the right set contains the left set. To show the other inclusion, let s be a relation in $SAT_X(\pi_X(\mathbf{C}))$. We exhibit a relation r in $SAT_R(\mathbf{C})$ such that $\pi_X(r) = s$. Form $s'(R)$ by extending each tuple in s to R with new values in the $(R - X)$-columns. Let r be the limit of chasing s' with $\mathbf{C}$. In extending the chase to relations, we do allow identification of values in this instance. However, we show shortly that no identifica-

tions are made in the X-columns of s'. Since r is obtained by chasing with $\mathbf{C}$, r is in $SAT(\mathbf{C})$. We show that $\pi_X(r) = s$.

Let t be any tuple in s and let t' be the extended version of t in s'. Suppose at some stage in the chase computation, some value in the X-columns of t' is changed by application of the G-rule. Interpreting s' as a tableau, we see that $\mathbf{C}$ implies a GFD γ that has s' as hypothesis rows and whose conclusion equates two values in the X-columns of s'. Since all the values in the $(R - X)$-columns of s' are distinct, we apply Lemma 14.9 to show that $\gamma(X)$ is in $\pi_X(\mathbf{C})$. But $\gamma(X)$ has s as its hypothesis rows, and equates two values in s. Therefore, s violates $\pi_X(\mathbf{C})$, a contradiction. We conclude that t' remains unchanged in the X-columns during the chase computation. If t'' is the tuple in r corresponding to t', then $t''(X) = t'(X) = t$. We conclude that $s \subseteq \pi_X(r)$.

Now let t'' be any tuple in r. Again interpreting s' as a tableau, we see that $\mathbf{C}$ implies the TD τ that has s' as hypothesis rows and t'' as the conclusion row. By Lemma 14.8, $\pi_X(\mathbf{C})$ contains the TD $\tau(X)$, which has s as hypothesis rows and $t''(X)$ as the conclusion row. Since s satisfies $\tau(X)$, s must contain $t''(X)$. Hence, $s \supseteq \pi_X(r)$, and so $s = \pi_X(r)$.

Theorem 14.12 is true for $SAT(\mathbf{C})$ interpreted as finite relations satisfying $\mathbf{C}$ if $\mathbf{C}$ contains only full TDs and GFDs. Looking back in the proof, the relation r generated by chasing s' with $\mathbf{C}$ will be finite if $\mathbf{C}$ meets this restriction. In particular, the "finite relation" version of the theorem is true for $\mathbf{C}$ consisting of FDs and JDs (although $\pi_X(\mathbf{C})$ may have dependencies that are not JDs or FDs).

14.3 LIMITATIONS OF RELATIONAL ALGEBRA

We have used relational algebra as the "yardstick" for a complete query system. In this section we show that this definition for complete can be disputed, since there are some natural operators on relations that cannot be expressed by relational algebra. To be exact, we show that there is no algebraic expression E that specifies the transitive closure of a two-attribute relation.

Definition 14.12 Let r be a relation on a two-attribute relation scheme, call it A_1A_2, where $dom(A_1) = dom(A_2)$. The *transitive closure of* r, denoted r^+, is the smallest relation on A_1A_2 such that $r \subseteq r^+$ and r^+ satisfies the untyped TD

$\tau(A_1$	$A_2)$
a	b
b	c
a	c

Note that this definition is symmetric in A_1 and A_2.

Example 14.30 If r is the relation in Figure 14.60, then Figure 14.61 shows r^+.

$r(A$	$B)$
1	2
2	1
2	3

Figure 14.60

$r^+(A$	$B)$
1	2
2	1
2	3
1	1
1	3

Figure 14.61

In the next theorem, we shall construct something that looks like a domain calculus expression, except we shall use a different set of atoms than usual. Assume r is a relation on scheme A_1A_2 where $dom(A_1) = dom(A_2)$. We use the atom $r^i(a\ b)$ to mean that b is i "steps away" from a in r. More precisely, for $i \geq 0$, $r^i(a\ b)$ is true when there are values $a_1, a_2, \ldots, a_{i-1}$ in $dom(A_1)$ such that $\langle a\ a_1\rangle$, $\langle a_1\ a_2\rangle$, $\langle a_2\ a_3\rangle$, $\ldots$, $\langle a_{i-1}\ b\rangle$ are all tuples in r. We let $r^0(a\ b)$ mean $a = b$ and a appears in r. That is, r^0 is equality on values in r. Finally, for $i < 0$, $r^i(a\ b)$ if and only if $r^{-i}(b\ a)$. Note that $r^i(a\ b)$ and $r^j(b\ c)$ together imply $r^{i+j}(a\ c)$ and that $r^1(a\ b)$ if and only if $\langle a\ b\rangle \in r$.

Example 14.31 For relation r in Figure 14.60, $r^0(1\ 1)$, $r^1(2\ 1)$, $r^2(1\ 1)$, and $r^2(1\ 3)$ are all true, while $r^0(4\ 4)$, $r^1(1\ 1)$, and $r^2(1\ 2)$ are all false.

Theorem 14.13 Let A_1 and A_2 be two attributes with the same domain. There is no relational algebra expression E involving a relation $r(A_1A_2)$ such that $E(r) = r^+$ for all states of r.

Proof Assume that the domain of A_1 and A_2 is the positive integers with only the comparators $=$ and $\neq$. While we do not allow inequality comparisons in selections, we shall use the order of the integers in our arguments. It suffices to show that there is some state of r for which $E(r) \neq r^+$. We restrict our attention to states of the form $\{\langle 1\ 2\rangle, \langle 2\ 3\rangle, \ldots, \langle p-1\ p\rangle\}$ for $p > 1$. We denote this state of r by $[p]$.

We begin by showing that for any relational algebra expression over r and for sufficiently large p, there is a domain calculus-like expression

$$E_p = \{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m) | f(b_1, b_2, \ldots, b_m)\}$$

such that $E([p]) = E_p([p])$. The b's in E_p are assumed to range over $\{1, 2, \ldots, p\}$. Formula f is constructed of atoms of the form $r^i(a_1\ a_2)$, where a_i is a constant or one of the b's, $1 \leq i \leq 2$, and the connectives $\wedge$, $\vee$ and $\neg$. A *literal* of f is an atom or its negation and a *clause* of f is a conjunction of literals. In the remainder of the proof, b's and d's are variables, c's are constants and a's are either.

The important property that E_p will have is that the form of f will depend on E but not on p. The value of p will appear in f as a constant, but the number of literals and clauses in f is independent of p.

The proof that E_p exists is done by induction on the number of operators in E. By Theorem 3.1, we shall assume that E contains only the relation symbol r, single-attribute single-tuple constant relations, selection with a single comparison, natural join, union, difference, renaming, and projection. We further assume that any projection removes exactly one attribute.

Basis If E has no operators, then E is r or $\langle c{:}B\rangle$ for some constant c. In the first case,

$$E_p = \{b_1(A_1)\, b_2(A_2) | r^1(b_1\ b_2)\}.$$

In the second case,

$$E_p = \{b_1(A_1) | r^0(b_1\ c)\},$$

as long as p is sufficiently large that c appears in $[p]$ (that is, $p \geq c$).

Induction We consider the form of E by cases. In all the cases, we assume that E' and E'' are subexpressions of E with corresponding calculus-like expressions

$$E_p' = \{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m) | f'(b_1, b_2, \ldots, b_m)\}$$

and

$$E_p'' = \{d_1(D_1)\, d_2(D_2) \cdots d_n(D_n) | f''(d_1, d_2, \ldots, d_n)\}.$$

1. *Selection* $E = \sigma_C(E')$. E_p has the form

$$\{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m) | f'(b_1, b_2, \ldots, b_m) \wedge g\}$$

where g is $r^0(b_i\, b_j)$, $\neg r^0(b_i\, b_j)$, $r^0(b_i\, c)$ or $\neg r^0(b_i\, c)$ depending on if the selection condition C is $B_i = B_j$, $B_i \neq B_j$, $B_i = c$, or $B_i \neq c$, respectively.

2. *Join* $E = E' \bowtie E''$. Assume that $B_1, B_2, \ldots, B_k$ are the same as $D_1, D_2, \ldots, D_k$ in E_p' and E_p''. E_p is then

$$\{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m)\, d_{k+1}(D_{k+1}) \cdots d_n(D_n) | \\ f'(b_1, b_2, \ldots, b_m) \wedge f''(b_1, b_2, \ldots, b_k, d_{k+1}, \ldots, d_n)\}.$$

3. *Union* $E = E' \cup E''$. For E to be a legal expression, E_p'' must have the form

$$\{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m) | f''(b_1, b_2, \ldots, b_m)\}.$$

E_p is

$$\{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m) | f'(b_1, b_2, \ldots, b_m) \vee \\ f''(b_1, b_2, \ldots, b_m)\}.$$

4. *Difference* $E = E' - E''$. E_p'' must be as in Case 3. E_p is

$$\{b_1(B_1)\, b_2(B_2) \cdots b_m(B_m) | f'(b_1, b_2, \ldots, b_m) \wedge \neg f''(b_1, b_2, \ldots, b_m)\}.$$

5. *Renaming* $E = \delta_{B_i \leftarrow D}(E')$. E_p is

$$\{b_1(B_1)\, b_2(B_2) \cdots b_i(D) \cdots b_m(B_m) | f'(b_1, b_2, \ldots, b_m)\}.$$

6. *Projection* $E = \pi_X(E')$, where $X = B_1 B_2 \cdots B_{m-1}$. This is the hard case to handle. Assume $m > 1$ and that $f'(b_1, b_2, \ldots, b_m)$ has the form

$$f_1(b_1, b_2, \ldots, b_m) \vee f_2(b_1, b_2, \ldots, b_m) \vee \cdots \vee f_q(b_1, b_2, \ldots, b_m),$$

where each f_i is a clause (f' is then said to be in *disjunctive normal form* or DNF). It is an elementary theorem of logic that f' can be put into DNF if it does not already have that form. E_p could be represented as

$$\{b_1(B_1)\, b_2(B_2) \cdots b_{m-1}(B_{m-1}) \mid \exists\, b_m(B_m) f'(b_1, b_2, \ldots, b_m)\},$$

but we do not want the existential quantifier. This expression is equivalent to

$$\begin{aligned}\{&b_1(B_1)\, b_2(B_2) \cdots b_{m-1}(B_{m-1}) \mid \\ &(\exists\, b_m(B_m) f_1(b_1, b_2, \ldots, b_m)) \vee \\ &(\exists\, b_m(B_m) f_2(b_1, b_2, \ldots, b_m)) \vee \cdots \vee \\ &(\exists\, b_m(B_m) f_q(b_1, b_2, \ldots, b_m))\},\end{aligned}$$

so we consider only the case where f' is itself a single clause.

Before we attempt to remove the existential quantifier, we do some manipulations of f'. For every atom that mentions b_m, we move b_m to the first slot, if it is not already there, using the identity $r^i(a\, b_m) = r^{-i}(b_m\, a)$. We can leave out any literal of the form $r^0(b_m\, b_m)$ or $\neg\, r^i(b_m\, b_m)$, $i \neq 0$, as they are always true for $[p]$. Likewise, $\neg\, r^0(b_m\, b_m)$ or $r^i(b_m\, b_m)$, $i \neq 0$, can be replaced by $\neg\, r^0(b_1\, b_1)$, as they are always false for $[p]$.

Two possibilities remain.

6.1 There is no literal of the form $r^i(b_m\, a)$ in f'. That is, any atom mentioning b_m is negated in f'. Let $f(b_1, b_2, \ldots, b_{m-1})$ be the conjunction of all the literals in f' that do not mention b_m. We claim that when p is sufficiently large, for any $m - 1$ constants $c_1, c_2, \ldots, c_{m-1}$ chosen from $\{1, 2, \ldots, p\}$,

$$f(c_1, c_2, \ldots, c_{m-1}) \equiv \exists\, b_m f'(c_1, c_2, \ldots, c_{m-1}, b_m).$$

The right side implies the left side, since $c_1, c_2, \ldots, c_{m-1}$ must satisfy every literal that does not mention b_m. In the other direction, if p is sufficiently large, there is always some constant c_m that makes every literal of the form $\neg\, r^i(b_m\, c)$ in $f(c_1, c_2, \ldots, c_{m-1}, b_m)$ true when b_m is replaced with c_m. Each such literal can prohibit only a single value for c_m. Since the number of literals in f' is fixed, but we allow p to be as large as necessary, there is always a choice for c_m. Thus, if $f(c_1, c_2, \ldots, c_{m-1})$ is

true, so is $\exists\, b_m f'(c_1, c_2, \ldots, c_{m-1}, b_m)$, by the choice of c_m for b_m. In this case we have

$$E_p = \{b_1(B_1)\, b_2(B_2) \cdots b_{m-1}(B_{m-1}) | f(b_1, b_2, \ldots, b_{m-1})\}.$$

6.2 The other possibility is that there is some literal of the form $r^i(b_m\, a)$ in f'. In this case, to form f, we remove $r^i(b_m\, a)$ and make a replacement for any other atom mentioning b_m. Since we have the relative position of b_m and a, we can convert any reference to b_m to a reference to a.

If the literal that mentions b_m is $r^j(b_m\, a')$, we replace it with $r^{j-i}(a\, a')$. Certain simplifications can then be made. For example, any literal of the form $r^k(c\, c')$ can be removed if $c + k = c'$, or replaced by $\neg r^0(b_1\, b_1)$ if $c + k \neq c'$. Similarly, any literal of the form $r^k(a\, b)$ can also be replaced by $\neg r^0(b_1\, b_1)$ if $|k| \geq p$.

To finish forming f, we must add a few more literals if the a in $r^i(b_m\, a)$ is actually b_k for some $1 \leq k \leq m - 1$. If $i > 0$, we must conjoin the literals $\neg r^l(1\, b_k)$, $0 \leq l < i$. Since b_k is at least i steps away from b_m, it must be at least i steps away from 1. If $i < 0$, we conjoin $\neg r^l(b_k\, p)$, $1 \leq l \leq -i$.

We leave Exercise 14.48 to show that

$$f(b_1, b_2, \ldots, b_{m-1}) \equiv \exists b_m f'(b_1, b_2, \ldots, b_m).$$

The final expression, as in the last subcase, is

$$E_p = \{b_1(B_1)\, b_2(B_2) \cdots b_{m-1}(B_{m-1}) | f(b_1, b_2, \ldots, b_{m-1})\}.$$

We have completed the case for projection. We now know that if E is a relational algebra expression as in the hypothesis of the theorem, for a sufficiently large choice of p, there is an expression

$$E_p = \{b_1(A_1)\, b_2(A_2) | f(b_1, b_2)\}$$

such that $E_p([p]) = [p]^+$. We may assume that f is in DNF.

We now argue that E_p cannot correctly compute $[p]^+$. It is important that the form of f is the same regardless of the choice of p (as long as p is sufficiently large that we can form f correctly). In particular, the number of clauses in f is independent of p. The only place p enters the construction (other than as a constant in E) is in the literals $\neg r^l(b_m\, p)$ that we add in subcase 6.2. Thus, we can convert E_p to $E_{p'}$, $p' \geq p$, by replacing each $\neg r^l(b_m\, p)$ by $\neg r^l(b_m\, p')$. Another property of f is that, by our simplifications, f con-

tains no atom of the form $r^i(a_1\ a_2)$ for $|i| \geq p$, nor any atom of the form $r^i(c_1\ c_2)$.

Suppose each clause of f contains an unnegated literal $r^i(a_1\ a_2)$ where a_i, $1 \leq i \leq 2$, is b_1, b_2, or a constant, but not both a_1 and a_2 are constants. The number of clauses in f is independent of p. Say there are k of them, so we are dealing with k unnegated literals. If p is sufficiently larger than k, then there are choices c_1 and c_2 for b_1 and b_2 such that $\langle c_1\ c_2\rangle \in [p]^+$, but replacing b_1 and b_2 by c_1 and c_2 makes each of the k unnegated literals false. For example, if $\bar{i}$ is the magnitude of the largest superscript of any of the unnegated literals, and $\bar{c}$ is the largest constant appearing in any of them, let $c_1 = \bar{i} + \bar{c} + 1$ and let $c_2 = 2\bar{i} + \bar{c} + 2$. By such a choice of c_1 and c_2, we have $f(c_1, c_2)$ is false, so $\langle c_1\ c_2\rangle$ is not in $E_p([p])$, but $\langle c_1\ c_2\rangle$ is in $[p]^+$, a contradiction.

If every clause of f does not contain an unnegated literal, then there is some clause in which every literal has the form $\neg\, r^i(a_1\ a_2)$, where a_i, $1 \leq i \leq 2$, is b_1, b_2 or a constant, but not both a_1 and a_2 are constants. Since there are a fixed number of literals in this clause, if p is large enough, we can choose values c_1 and c_2 for b_1 and b_2 such that $\langle c_1\ c_2\rangle$ is not in $[p]^+$, but all the atoms in the clause are false. Since all the atoms are false, all the literals are true, so the clause is true and $f(c_1, c_2)$ is true. (How do we pick c_1 and c_2?) Hence, $\langle c_1\ c_2\rangle \in E_p([p])$, a contradiction. We see that in any case, there is some p for which $E_p([p]) \neq [p]^+$. Since E_p is equivalent to E, E cannot compute transitive closures correctly for all states of r.

There have been several proposals for extending relational algebra so it can express more operations on relations. These proposals generally involve addition of programming language constructs or fixed-point operators. The query language QBE, which we cover in the next chapter, includes constructs specifically for dealing with transitive closures of relations (for relations whose closures are anti-symmetric).

14.4 COMPUTED RELATIONS

14.4.1 An Example

Consider a relation *schedule*(FLIGHT# FROM TO DEPARTS ARRIVES) containing flight information for our mythical airline. Suppose we want to create a relation *length*(FLIGHT# FLYTIME) that gives the duration of each flight. One approach is to use a database command to extract tuples

from the relation, perform a calculation in some general-purpose programming language, and use another database command to insert tuples into *length*. That is, we embed calls to the database system within programs in some standard programming language. If the times in *schedule* are local, the program doing the duration calculation has to know what the time zone is for each city served. It would simplify the program if the database system could connect each city with its time zone. We can keep a relation *inzone*(CITY ZONE) giving the time zone for each city served. We can then define two virtual relations

$$fromzone = \delta_{CITY \leftarrow FROM,ZONE \leftarrow FZONE}(inzone)$$
$$tozone = \delta_{CITY \leftarrow TO,ZONE \leftarrow TZONE}(inzone)$$

and use them to define a third virtual relation

$$zonetimes = \pi_{FLIGHT\# \; DEPARTS \; FZONE \; ARRIVES \; TZONE}(schedule \bowtie fromzone \bowtie tozone).$$

The program can then access *zonetimes* to compute *length* without having to look up time zones for cities.

Suppose we want *length* to be a virtual relation, so its state is always consistent with that of *schedule*. We need to do the computation of *length* entirely within the database system. We could have a relation *lasts*(DEPARTS FZONE ARRIVES TZONE FLYTIME) that gives the duration for all combinations of departure and arrival times and zones. If we had *lasts*, we could define *length* as

$$\pi_{FLIGHT\# \; FLYTIME}(zonetimes \bowtie lasts).$$

While such an approach is conceivable, *lasts* will be a huge relation if it includes all possible combinations of times and zones. Even if we took this approach, we would probably need a program to calculate all the tuples in *lasts*.

What would be nice is if *lasts* only contained the tuples needed to correctly compute *length*. There is no way to know in advance what tuples will be needed in *lasts*, so it would be desirable to compute the proper tuples upon demand. That is, we would like to implicitly embed calls to programs within database commands. Rather than associating any stored extension with *lasts*, we instead associate a program that calculates FLYTIME, given values for DEPARTS, FZONE, ARRIVES, and TZONE, so tuples can be created upon demand when computing *length*.

We call a relation whose extension is a function a *computed* relation. To distinguish computed relations from relations with stored extensions, we call the latter *tabular* relations.

We cannot use the computed relation *lasts* in arbitrary expressions, for example

$$dtimes = \pi_{DEPARTS\ FZONE}(\sigma_{ARRIVES=1:10p,FLYTIME=2:20}(lasts)).$$

The procedure associated with *lasts* does not work in the right direction for use in computing *dtimes*. We can associate other programs with *lasts* to generate tuples when values on other sets of attributes are given. To handle *dtimes*, we would need a procedure that generates all DEPARTS FZONE TZONE-values for a given ARRIVES FLYTIME-value. While such a procedure is not unreasonable, we probably would not want to deal with a procedure that generates all DEPARTS FZONE FLYTIME-values for a given ARRIVES TZONE-value. In general, while we can associate several procedures with a computed relation to handle different sets of input attributes, it is unlikely that there will be a procedure for every set of attributes.

If we associate a procedure with *lasts* to compute an ARRIVES-value from a DEPARTS FZONE TZOME FLYTIME-value, we can evaluate the expression

$$\pi_{TO\ FROM\ ARRIVES}(\sigma_{DEPARTS=8:15a,FLYTIME=2:40}(lasts \bowtie fromzone \bowtie tozone))$$

to get the arrival times in various cities of a flight that departs at 8:14a and lasts 2 hours and 40 minutes. It may not be immediately apparent that we can evaluate this expression given the procedural extension of *lasts*, but we shall show shortly that it is possible.

The procedures associated with computed relations need not return a value for every possible value on the input attributes. For *lasts*, for any set of four attributes, there is a function that will return a value on a fifth attribute, so long as there is a legal value. However, there may not always be a legal value for the fifth attribute to go with a given value on the other four. For instance, if we have ⟨9:20a Pacific 1:20p 2:40⟩ as a value on DEPARTS FZONE ARRIVES FLYTIME, there is no value for TZONE that will form a legal tuple in *lasts*.

Sometimes we cannot evaluate a given expression involving a computed relation because certain procedures are absent from the relation's extension.

In some of those cases, however, we may still be able to test if a particular tuple is in the relation represented by the expression. Suppose the only procedures associated with *lasts* are ones that generate a fifth value, given four others. Consider again the expression

$$dtimes = \pi_{DEPARTS\ FZONE}(\sigma_{ARRIVES=1:10p.FLYTIME=2:20}(lasts)).$$

While with the specified computed extension for *lasts* we cannot evaluate *dtimes*, we can test if a particular tuple, say ⟨10:50a Eastern⟩, is in *dtimes*. We can make this test because it amounts to evaluating the expression

$$\sigma_{DEPARTS=10:50a,FZONE=\text{Eastern}}(\pi_{DEPARTS\ FZONE}(\sigma_{ARRIVES=1:10p,FLYTIME=2:20}(lasts))),$$

for which the given procedural extension of *lasts* is adequate.

The next section examines the problem of determining, given a restricted algebraic expression involving computed relations, if the expression can be effectively evaluated, tested for membership, or neither.

14.4.2 Testing Expressions Containing Computed Relations

We shall not worry much about the exact mechanism for specifying the procedures that make up the extension of a computed relation. We are mainly concerned with which sets of attributes can be used to determine values for other attributes.

To develop the theory of computed relations, it helps to imagine that any computed relation $r(R)$ does have a tabular extension, although that extension might be infinite. We shall use simply r to denote that extension. For X, $Y \subseteq R$, we say there is a *computed dependency* (CD) of Y on X in r, written $X =: Y$, if given an X-value x, there is a procedure to compute $\pi_Y(\sigma_{X=x}(r))$. While not essential to the following discussion, it may clarify things to assume that a CD $X =: Y$ implies the FD $X \rightarrow Y$. That is, there is a function to compute a single Y-value given an X-value, so $\pi_Y(\sigma_{X=x}(r))$ contains at most one tuple. A *determining set* for r is any left side of a CD on r.

Example 14.32 If for the computed relation *lasts* in the previous section we could compute a fifth value from any four, *lasts* would satisfy the CDs

DP FZ AR TZ =: FT
DP FZ AR FT =: TZ
DP FZ TZ FT =: AR
DP AR TZ FT =: FZ
FZ AR TZ FT =: DP,

using the abbreviations DP, FZ, AR, TZ and FT for DEPARTS, FZONE, ARRIVES, TZONE and FLYTIME. To recast remarks in the previous section, it is conceivable that *lasts* could satisfy the CD

AR FT =: DP FZ TZ,

but it is unlikely that the extension of *lasts* would contain a procedure that would give rise to

AR TZ =: DP FZ FT.

We now define the terms *listable* and *decidable* for algebraic expressions involving computed relations. We limit ourselves to expressions with single-tuple constant relations, project, select on equality, and natural join. Recall that these constraints define the class of restricted algebraic expressions, for which equivalent tagged tableau queries exist. We actually define listable and decidable for tagged tableau queries. Recall the notation associated with a tagged tableau query. For a tagged tableau query Q, a database d, and a valuation ρ of Q, $\rho(Q) \subseteq d$ means that ρ maps every row w in Q to a tuple in the relation in d with scheme $tag(w)$. We then have

$$Q(d) = \{\rho(w_0) | \rho(Q) \subseteq d\},$$

where w_0 is the summary of Q. Since we imagine every computed relation to have an extension, even if we cannot effectively compute it, $Q(d)$ makes sense even when d contains computed relations.

To make our definitions, we need the algorithm MARK given in Figure 14.62. MARK marks symbols in a tableau query, succeeding if it marks all the symbols in the summary, and marks all matched symbols and a determining set for each row that is tagged with a computed relation. We assume MARK has global access to two set variables, *TABULAR* and *COMPUTED*, that give the sets of tabular and computed relations in the database.

Input: A tagged tableau query Q, and an array *CDEP* of sets of computed dependencies for each relation in *COMPUTED*.
Output: *true* if all symbols of Q get marked; *false* otherwise.
MARK(Q, *CDEP*)
begin

1. *Initialization.*
   Mark all constant symbols in Q;
   for each row $w \in Q$ **do**
     if $tag(w) \in$ *TABULAR*
     then mark every symbol in w;

2. *Computation.*
   2.1. *Propagate Marks.*
     while changes occur **do**
       for each marked variable a in Q **do**
         mark all copies of a in Q, including those in the summary;
   2.2. *Apply CDs.*
     for each row $w \in Q$ **do**
       if $tag(w) \in$ *COMPUTED*
       then begin
         for each CD $X =: Y$ in $CDEP(tag(w))$ **do**
           if all the symbols in the X-columns of w are marked
           then mark all the symbols in the Y-columns of w
       end;

3. *Return Results*
   if all the symbols in the summary are marked **and**
     every row tagged with a computed relation is marked on
     all its matched variables and on some determining set
   then return(*true*)
   else return(*false*)

end.

Figure 14.62

Example 14.33 Consider the tagged tableau query Q_1 in Figure 14.63, which corresponds to one of the expressions of the last section (where FR abbreviates FROM). Assume the first five CDs from the last example hold on *lasts* and that relations *fromzone* and *tozone* are tabular. Figure 16.64 shows

Q_1 after the initialization step in MARK. We use the symbol $\times$ for marks. Figure 14.65 shows Q_1 after propagation of marks. Applying the CD DR FZ TZ FT =: AR, we may mark a_5 in the first row, as shown in Figure 14.66. Finally, in the second pass through the computation step, the mark on a_5 is propagated to the summary, so MARK returns *true*.

Q_1(FR	TO	DR	FZ	AR	TZ	FT)	
a_1	a_2			a_5			
		8:14a	b_1	a_5	b_2	2:41	(*lasts*)
a_1			b_1				(*fromzone*)
	a_2				b_2		(*tozone*)

Figure 14.63

Q_1(FR	TO	DR	FZ	AR	TZ	FT)	
a_1	a_2			a_5			
		8:14a $\times$	b_1	a_5	b_2	2:41 $\times$	(*lasts*)
a_1 $\times$			b_1 $\times$				(*fromzone*)
	a_2 $\times$				b_2 $\times$		(*tozone*)

Figure 14.64

Q_1(FR	TO	DR	FZ	AR	TZ	FT)	
a_1 $\times$	a_2 $\times$			a_5			
		8:14a $\times$	b_1 $\times$	a_5	b_2 $\times$	2:41 $\times$	(*lasts*)
a_1 $\times$			b_1 $\times$				(*fromzone*)
	a_2 $\times$				b_2 $\times$		(*tozone*)

Figure 14.65

Q_1(FR	TO	DR	FZ	AR	TZ	FT)	
a_1 $\times$	a_2 $\times$			a_5			
		8:14a $\times$	b_1 $\times$	a_5 $\times$	b_2 $\times$	2:41 $\times$	(*lasts*)
a_1 $\times$			b_1 $\times$				(*fromzone*)
	a_2 $\times$				b_2 $\times$		(*tozone*)

Figure 14.66

Example 14.34 The tagged tableau query Q_2, in Figure 14.67, also corresponds to an expression given in the last section. Assume again that *lasts* has the first five CDs in Example 14.32. If we apply MARK to Q_2, the two constants in the first row get marked, but no other marks are made. Thus, MARK returns *false* for Q_2.

Q_2(DR FZ AR TZ FT)

a_1	a_2				
a_1	a_2	1:10p	b_1	2:20	(*lasts*)

Figure 14.67

Definition 14.13 A tagged tableau query Q is *listable* relative to a set of CDs *CDEP* if MARK(Q, *CDEP*) = *true*.

Example 14.35 The query Q_1 given in Example 14.33 is listable, while the query Q_2 in Example 14.34 is not.

We have done things in a rather backwards manner. Usually we define a property, and then give an algorithm to test for it. Here we have given the algorithm, and defined the property from it. We shall now show that the term "listable" is well-chosen—that for a listable query Q and a database d, we can indeed compute $Q(d)$. We leave it to the reader to show that there is no way, in general, to evaluate $Q(d)$ if Q is not listable (see Exercise 14.51). To simplify the argument, we assume that each CD $X =: Y$ implies the corresponding FD $X \rightarrow Y$. Hence, for any relation r with CD $X =: Y$, we may assume a function f_{XY} that returns a Y-value for any X-value, or possibly a special value κ (for *kill*) that indicates r contains no tuple with the given X-value. The argument can be generalized to allow several Y-values for a given X-value (see Exercise 14.53).

To evaluate a tagged tableau query Q on a database d, we need to find every valuation ρ of Q such that $\rho(Q) \subseteq d$. Let Q_{tab} be the set of rows in Q whose tags are tabular relations and let Q_{com} be the rows with tags that are computed relations. The possibilities for ρ are limited by its value on rows in Q_{tab}. It is a straightforward enumeration to find every valuation ρ of Q_{tab} such that $\rho(w) \in tag(w)$ for every row w in Q_{tab}, that is, $\rho(Q_{tab}) \subseteq d$. We assume that we start with such a valuation ρ for Q_{tab}, and attempt to extend it to a valuation for all of Q. We know that $\rho(c)$ must equal c for every constant c, so we can extend ρ to all the constants in Q. At this point ρ is defined on the symbols that are marked after the initialization stage of running

MARK on Q. The strategy of this argument is to show that we can extend ρ to any symbol marked during that computation.

Any symbol a that is marked during the propagation step must already have $\rho(a)$ defined. If we apply a CD $X =: Y$ on a row w to mark the symbols in the Y-columns, the X-columns must already be marked. Hence, we know the value of $\rho(w(X))$. Using the function f_{XY} corresponding to the CD, we can get a Y-value $y = f_{YX}(\rho(w(X)))$. If y is actually κ, we know there is no way to extend ρ so that $\rho(w) \in tag(w)$. Also, no extension is possible if ρ is already defined on $w(A)$, $A \in Y$, and $\rho(w(A)) \neq y(A)$. If neither of these cases arises, then we may extend ρ to $w(Y)$ by letting $\rho(w(Y)) = y$.

If we continue on in this manner, we either find at some point ρ cannot be extended, or it is extended to all the symbols that get marked by MARK. The valuation ρ might not be defined on every symbol in some row $w \in Q_{com}$. Let $tag(w) = r(R)$. Even if ρ is not defined on all of w, since MARK succeeds on Q, for some determining set X of r, ρ is defined. Thus, if ρ is defined on exactly $w(Z)$, for some $Z \subseteq R$, we can determine if there is a tuple t in r such that $\rho(w(Z)) = t(Z)$. Since any variable in $w(R - Z)$ is not marked, it cannot be matched, or Q would not be listable. Theoretically, then, ρ could be extended to all of w by letting $\rho(w) = t$. We may not actually be able to determine the value of t on attributes in $R - Z$, but it is sufficient to know that appropriate values exist. Since the summary, say w_0, of Q gets marked, $\rho(w_0)$ is defined, and is a tuple in $Q(d)$.

We have argued that any valuation ρ found by the method above can be extended to all of Q in such a manner that $\rho(Q) \subseteq d$. Starting with some valuation ρ such that $\rho(Q) \subseteq d$, it is not hard to see that if we restrict ρ to Q_{tab} and then extend it by the method above, we end up with a valuation that agrees with ρ on all the marked symbols of Q, and hence has the same value on the summary. Therefore, we can find every tuple in $Q(d)$.

We now turn to the definition of decidable. Let MARK′ be the algorithm MARK of Figure 14.62 modified so that all the symbols in the summary of Q are marked during the initialization step.

Definition 14.14 A tagged tableau query Q is *decidable* relative to a set of CDs *CDEP* if MARK′$(Q, CDEP) = true$.

Example 14.36 Referring back to Example 14.34, let Q_2 be the tagged tableau query in Figure 14.67. Running MARK′ on Q_2, we mark all the symbols in the summary and the constants during initialization, as shown in Figure 14.68. Propagating the marks on a_1 and a_2, and applying the CD

DR FZ AR FT =: TZ results in all the symbols of Q_2 being marked. Hence MARK′ returns *true* for Q_2 and so Q_2 is decidable.

Q_2(DR	FZ	AR	TZ	FT	)
a_1 ×	a_2 ×				
a_1 ×	a_2 ×	1:10p ×	b_1	2:20 ×	(*lasts*)

Figure 14.68

We now argue that for a decidable tagged tableau query Q and a tuple t, we can effectively decide if $t \in Q(d)$. Let R be the scheme of $Q(d)$, and assume t is a tuple on R. Deciding if $t \in Q(d)$ is the same as determining if $\sigma_{R=t}(Q(d))$ has any tuples. Let w_0 be the summary of Q. We can form a query Q' for $\sigma_{R=t}(Q(d))$ by replacing any variable a_i in the A_i-column of w_0 by the constant $t(A_i)$. If $w(A_i)$ is already a constant and not equal to $t(A_i)$, we can stop, for we know t is not in $Q(d)$. Q' is just Q with summary variables replaced by constants, so MARK succeeds on Q' exactly when MARK′ succeeds on Q. Therefore, if Q is decidable, Q' is listable, and we can effectively decide if $t \in Q(d)$ by computing $Q'(d)$.

The definitions of listable and decidable implicitly assume that domains are infinite. If a domain is finite, we can enumerate its values as part of an evaluation strategy. For example, if we had a computed relation $r(A\ B)$ with CD A =: B, and $dom(A)$ is finite, we could compute the extension of r by enumerating the domain of A. Exercise 14.55 shows how to incorporate information on finite domains into the definitions of listable and decidable.

14.5 EXERCISES

14.1 Show that all the inference rules for FDs are valid rules of inference for logic when FDs are interpreted as propositional formulas.

14.2* Choose any complete set of inference rules for FDs. Show that these rules, when interpreted as inference rules in propositional logic, are a complete set of inference rules for the subtheory of propositional logic consisting only of formulas of the form $X \Rightarrow Y$.

14.3 Complete the proof of Lemma 14.1.

14.4 Show that all the inference rules for MVDs alone and MVDs with FDs are valid rules of inference for logic when FDs and MVDs are interpreted as propositional formulas.

14.5 Prove Lemma 14.3.

14.6 Prove Lemma 14.5.

14.7 Prove Theorem 14.2.

14.8 Let F be a set of FDs, each with a single attribute on the right side. Define

$$M_F = \{X \twoheadrightarrow Y | X \rightarrow A \in F\}.$$

M_F is F converted to MVDs. Use Theorem 14.2 to prove the following theorem of Beeri.

Theorem 14.14 Let F be a set of FDs and let M be a set of MVDs. For an MVD $X \twoheadrightarrow Y$, $F \cup M$ implies $X \twoheadrightarrow Y$ if and only if $M_F \cup M$ implies $X \twoheadrightarrow Y$.

One consequence of this theorem is that any procedure for implication of MVDs by MVDs can easily be converted to a procedure for the inference of MVDs by FDs and MVDs.

14.9 Show that inference in the world of two-tuple relations is not the same as inference over regular relations for JDs and embedded MVDs.

14.10* Show that the correspondence of FDs and MVDs with formulas in propositional logic can not be extended to embedded MVDs in such a way as to preserve equivalence of implication.

14.11 Let J be a set of JDs over scheme R. Let X be a subset of R. Show that $\pi_X(SAT(J))$ cannot necessarily be expressed as $SAT(J')$ for a set J' of JDs over X.

14.12 Give the smallest relation containing the relation r below that satisfies the TD τ in Figure 14.5.

$r(A$	B	$C)$
1	3	5
1	3	6
1	3	7
2	3	5
2	4	5
1	4	6

14.13 Give the smallest relation containing relation r above that satisfies the TD τ in Figure 14.7.

14.14 Show that the TD τ in Figure 14.5 is not equivalent to any JD.

14.15 Prove that any JD, full or properly embedded, is equivalent to some TD.

Definition 14.15 A (typed) TD is *simple* if each column has at most one repeated variable.

14.16 (a) Prove that if a TD is simple and full, then it is equivalent to some JD.
(b) Give a simple TD that is not equivalent to any full or embedded JD.

14.17* Show that any set of full TDs over the same scheme is equivalent to a single TD.

14.18* Give a set of partial TDs over the same scheme that is not equivalent to a single TD.

14.19* Prove that there are only three distinct TDs over a relation scheme with two attributes. (They are the trivial TD, the Cartesian product TD, and the TD in Figure 14.10.)

14.20 Show that a TD can be expressed as a quantified predicate calculus formula with a single predicate symbol (for the relation). Note that if the TD is full, the formula uses no existential quantifiers.

14.21 Prove that any relation that is a column-wise Cartesian product satisfies every TD over its scheme.

14.22 (a) Consider the set of all S-partial TDs over a scheme R, where $S \subseteq R$ and S has two or more attributes. Show that there is a strongest TD and a weakest nontrivial TD in this set.
(b) Prove that there is no weakest nontrivial TD over any scheme of three or more attributes.

14.23 Prove that a full TD is trivial if and only if the conclusion row is also a hypothesis row.

14.24 Characterize those TDs that are equivalent to JDs in terms of the graphs of the TDs.

14.25 Give two different graphs for the TD τ in Figure 14.13.

14.26 Is every TD whose graph can be drawn as a triangle equivalent to an EMVD?

14.27 Prove Theorem 14.6.

14.28* Exhibit a chain of progressively stronger TDs, along the lines of Theorem 14.7. Hint: Consider cutting the graph G in Figure 14.20 at node i, and then overlaying nodes 1 and i.

14.29* Finish the proof of Theorem 14.8.

14.30 Let **C** be a set of S-partial TDs on scheme R and let τ be an arbitrary TD on R. Show that if there is an infinite relation that satisfies **C** but violates τ, then there is a finite relation with the same property.

14.31 Show that the following inference axioms for TDs are correct. Let $\tau = (T, w)$ be a TD.
(a) *Renaming* If τ' is formed from τ by a one-to-one renaming of symbols, then τ implies τ'.
(b) *Identification of variables* If τ' is formed by replacing all oc-

currences of some variable in T of τ with a variable from the same column of T, then τ implies τ'. (Note that this axiom does not allow identifying a variable that only appears in w with a variable in T.)

(c) *Transitivity* Let $\tau_1 = (T_1, w_1)$ and $\tau_2 = (T_2, w_2)$ where $T_1 \supseteq T_2 \cup \{w_2\}$. TDs τ_1 and τ_2 together imply $\tau_3 = (T_1 - \{w_2\}, w_1)$.

(d) *Reflexivity* The TD $(\{w\}, w)$ holds for any row w.

14.32* Prove that Augmentation (Lemma 14.7), Weakening (Section 14.3.4), Renaming, Identification of variables, Transitivity, and Reflexivity (from the last exercise) are a complete set of inference axioms for implication of TDs on arbitrary relations.

14.33* Show that if the T-rule is restricted to generate only rows that contain new combinations of original variables in a tableau, then some combinations of original symbols will not be obtained that would be obtained without the restriction. That is, provide an example tableau and some TDs where the T-rule cannot generate any new combination of original variables immediately, but will do so eventually.

14.34 Let **C** be a set of TDs over scheme R and let T be a tableau on R. Show that chasing T with **C** produces the same combinations of original variables no matter what the order in which the TDs in **C** are used.

14.35 Let **C** be a set of full TDs over a scheme R and let T be a tableau on R. Show that chasing T with **C** always gives a unique result for the tableau at the last stage.

14.36 Referring to Example 14.23, show that neither TD in **C** by itself implies τ.

14.37 Which of the following relations satisfies the GFD γ in Figure 14.40?

r_1

A	B	C	D
1	3	5	7
1	4	6	8
2	3	6	7

r_2

A	B	C	D
1	2	3	5
1	2	4	6

r_3

A	B	C	D
1	2	4	5
1	3	4	6

14.38 Give a syntactic characterization for when a GFD is trivial.

14.39 Find an inference axiom along the lines of GT1 that gives a nontrivial TD implied by a nontrivial GFD.

14.40 Show that axiom GT1 can be used to replace FDs with TDs when considering the TDs implied by a set of FDs and GFDs.

14.41 Prove that implication of GFDs is the same for finite and arbitrary relations.

14.42 Note that row w_1 can be removed from the GFD γ in Figure 14.50 to get an equivalent GFD. State and prove a general result about superfluous rows in TDs and GFDs.

14.43 Sketch the proof of Theorem 14.11. Be sure to indicate why our method for renaming variables when applying the G-rule is adequate.

14.44 In doing a chase computation with TDs and GFDs, show that if the sequence of tableaux generated is finite, then the last tableau in the sequence is the limit of the sequence.

14.45* Let **C** be a finite set of TDs and GFDs. Demonstrate that $\pi_X(\mathbf{C})$ can have an infinite number of dependencies, and also not be equivalent to any finite set of dependencies. What if **C** happens to contain only FDs and JDs?

14.46 (a) Prove that there is a set of GFDs equivalent to $\pi_X(\mathbf{C})$ when **C** contains only FDs.

(b) Prove that there is a set of TDs equivalent to $\pi_X(\mathbf{C})$ when **C** contains only JDs.

14.47 Prove Lemma 14.8.

14.48 In Case 6.2 of the proof of Theorem 14.13, show that

$$f(b_1, b_2, \ldots, b_{m-1}) \equiv \exists b_m \, f'(b_1, b_2, \ldots, b_m).$$

14.49 (a) Let *lasts* and *zonetimes* be as defined in Section 14.4.1. Assume *connects*(*F*#1 *F*#2) is a relation giving all pairs of connecting flights. Give an algebraic expression that defines a relation giving total duration and layover time for each pair of connecting flights.

(b) What CDs must *lasts* satisfy in order to be able to evaluate your answer to part (a)?

14.50 Let $r_1(A\ B\ C)$ and $r_2(C\ D)$ be tabular relations. Let $s_1(B\ D\ E)$ be a computed relation with CDs $B =: D\ E$ and $D\ E =: B$. Let $s_2(C\ E\ I)$ be a computed relation with CD $C\ E =: I$. Convert each of the following restricted algebraic expressions to a tagged tableau query, and say which of the resulting queries are listable.

(a) $r_1 \bowtie s_1$
(b) $r_2 \bowtie s_1$
(c) $\pi_{BC}(r_2 \bowtie s_1)$
(d) $\pi_{BE}(r_2 \bowtie s_1)$
(e) $\sigma_{E=1}(r_2 \bowtie s_1)$
(f) $r_1 \bowtie s_1 \bowtie s_2$
(g) $\pi_{ACE}(r_1 \bowtie r_2 \bowtie s_2)$
(h) $s_1 \bowtie s_2$
(i) $\pi_{BI}(s_1 \bowtie s_2)$
(j) $\pi_{DI}(s_1 \bowtie s_2)$

14.51 Show that in general there is no effective method to evaluate a query that is not listable. Assume that all domains are infinite.

14.52* Does tableau query equivalence preserve listability?

14.53 Give a method to evaluate a listable query without the assumption that a CD $X =: Y$ implies the FD $X \rightarrow Y$.

14.54 Which of the tableau queries you produced in Exercise 14.50 are decidable?

14.55 Show how information on finite domains can be brought into the theory of computed relations by using a single-attribute tabular relation $r(A)$ for every attribute A with a finite domain. The extent of r will be $dom(A)$.

14.6 BIBLIOGRAPHY AND COMMENTS

The connection between FDs and propositional formulas was first exhibited by Delobel and Casey [1973], and was fully developed by Fagin [1977b]. Sagiv [1980] showed the connection between MVDs and formulas. The material in this chapter on the connection between FDs and MVDs together and propositional formulas, particularly the material on two-tuple relations and the proof of Lemma 14.6, is from Sagiv, Delobel, Parker, and Fagin [1981]. Namibar [1979] also points out the connection between MVDs and logical formulas.

Template dependencies were introduced by Sadri and Ullman [1980a], although they deal only with the typed case. They give a complete axiomatization for TDs and also show how to extend the chase to TDs. Sadri and Ullman [1980b] also show how to include FDs with the TDs in the chase. The material on examples and counterexamples for TDs, graphical representations, and implication on finite versus infinite relations is from Fagin, Maier, Ullman, and Yannakakis [1981]. GFDs were defined by Sadri [1980a, 1980b, 1980c], who gives inference rules for TDs and GFDs together, ex-

tends the chase to GFDs, shows satisfaction classes are closed under projection, and defines a normal form with respect to TDs and GFDs.

There have been numerous dependency classes defined in attempts to overcome problems with existing classes or to produce a more expressive language of relational constraints. Nicolas [1978a, 1978b] introduced *mutual dependencies,* which are a kind of join dependency, and showed that FDs, MVDs and mutual dependencies can all be expressed in first-order logic. Maier and Mendelzon [1979] generalized mutual dependencies. Tanaka, Kambayashi, and Yajima [1979b] explore the properties of EMVDs. Parker and Parsaye-Ghomi [1980], and Sagiv and Walecka [1979] proved that there is no finite complete axiomatization of the EMVDs by themselves. Both proofs work by showing that for any k there is an inference axiom that says a certain k EMVDs imply some other EMVD τ, but any $k - 1$ of those EMVDs only imply other EMVDs that are implied by the EMVDs singly. Thus, that inference axiom cannot be derived from any set of inference axioms with $k - 1$ or less hypothesis EMVDs. The latter pair of authors define the class of *subset dependencies,* which properly include the EMVDs, and which has a finite complete axiomatization. A subset dependency is a statement of the form

$$\pi_Z(\sigma_{X=x}(r)) \subseteq \pi_Z(\sigma_{Y=y}(r)).$$

Sciore [1982], attempting to axiomatize JDs, had to generalize that class of dependencies slightly in order to do so, and conjectures there is no finite axiomatization of JDs. His dependencies were essentially TDs with the restriction that any column may contain at most two repeated variables.

Several people independently came up with classes of dependencies similar to TDs and GFDs. Fagin [1980a] used Horn clauses to define *embedded implicational dependencies* (EIDs), which are equivalent to untyped TDs and GFDs (where we consider any typed dependency to also be an untyped dependency). Among other results, he shows that any set **C** of EIDs has an Armstrong relation, that is, a relation that satisfies exactly the EIDs in **C**. Fagin also shows closure of satisfaction classes of EIDs under projection. Yannakakis and Papadimitriou [1980] give the class of *algebraic dependencies,* which are set inequalities on algebraic expressions formed with projection and equijoin. They show that algebraic dependencies are equivalent to EIDs. Grant and Jacobs [1980] use logic to define a class of dependencies that are equivalent to "full" EIDs. That is, their class is equivalent to untyped full TDs and untyped GFDs. Paredaens and Janssens [1981] define *generalized dependencies,* which are equivalent to TDs and GFDs.

Beeri and Vardi [1980a, 1980b, 1980c] define *tuple generating dependen-*

cies and *equality generating dependencies,* which correspond to untyped TDs and untyped GFDs, respectively. They prove a number of complexity and decidability results, such as finite and infinite implication are undecidable for EIDs and implication of S-partial TDs is decidable. They also show the inequivalence of finite and infinite implication for untyped dependencies. Chandra, Harel, and Makowsky [1981] show undecidability of infinite implication for typed EIDs and untyped TDs. They also show that implication of "full" EIDs is decidable, but exponential-time complete. There are recent reports from several researchers that implication for typed TDs is undecidable, but the decidability of EMVDs has yet to be resolved.

Hull [1981] explores the properties of EIDs, and proves that satisfaction classes of EIDs are closed under join.

Several types of dependencies of a flavor different from TDs and GFDs have been proposed. Lipski and Marek [1979] discuss constraints involving cardinalities. Ginsburg and Hull [1981] consider constraints involving ordered domains. Casanova [1981] presents a class of dependencies called *subset dependencies,* but they are not the same as the subset dependencies of Sagiv and Walecka. Rather, they are interrelational constraints that connect projections of two relations.

The proof that transitive closure is not expressible in relational algebra is due to Aho and Ullman [1979], and they give some proposals for extensions of the algebra. Banchilon [1978], Paredaens [1978], Chandra [1981], and Chandra and Harel [1980a, 1980b] have all proposed other notions of query language completeness. The language specification for QBE (Query-by-Example) given by Zloof [1976] includes specific constructs for dealing with transitive closures.

The section on computed relations is adapted from Maier and Warren [1981a]. The ISBL query language, presented by Hall, Hitchcock, and Todd [1975] and Todd [1975, 1976], allows some forms of computed relations.

Exercises 14.2, 14.8, and 14.10 are from Sagiv, Delobel, Parker, and Fagin [1981]. The original proof of Theorem 14.14 is by Beeri [1980]. Exercises 14.17, 14.18, 14.19, 14.28, and 14.29 are from Fagin, Maier, Ullman, and Yannakakis [1981]. Answers to Exercises 14.32 and 14.33 can be found in Sadri and Ullman [1980a]. Exercise 14.39 is from Beeri and Vardi [1980a]. Exercise 14.45 on finite specification is taken from Hull [1981]. Exercise 14.46 is from Sadri [1980a].