10 A Larger Perspective

Alice: *fd's, jd's, mvd's, ejd's, emvd's, ind's—it's all getting very confusing.*
Vittorio: *Wait! We'll use logic to unify it all.*
Sergio: *Yes! Logic will make everything crystal clear.*
Riccardo: *And we'll get a better understanding of dependencies that make sense.*

The dependencies studied in the previous chapters have a strong practical motivation and provide a good setting for studying two of the fundamental issues in dependency theory: deciding logical implication and constructing axiomatizations.

Several new dependencies were introduced in the late 1970s and early 1980s, sometimes motivated by practical examples and later motivated by a desire to understand fundamental theoretical properties of unirelational dependencies or to find axiomatizations for known classes of dependencies. This process culminated with a rather general perspective on dependencies stemming from mathematical logic: Almost all dependencies that have been introduced in the literature can be described as logical sentences having a simple structure, and further syntactic restrictions on that structure yield natural subclasses of dependencies. The purpose of this chapter is to introduce this general class of dependencies and its natural subclasses and to present important results and techniques obtained for them.

The general perspective is given in the first section, along with a simple application of logic to obtain the decidability of implication for a large class of dependencies. It turns out that the chase is an invaluable tool for analyzing implication; this is studied in the second section. Axiomatizations for important subclasses have been developed, again using the chase; this is the topic of the third section. We conclude the chapter with a provocative alternative view of dependencies stemming from relational algebra.

The classes of dependencies studied in this chapter include complex dependencies that would not generally arise in practice. Even if they did arise, they are so intricate that they would probably be unusable—it is unlikely that database administrators would bother to write them down or that software would be developed to use or enforce them. Nevertheless, it is important to repeat that the perspective and results discussed in this chapter have served the important function of providing a unified understanding of virtually all dependencies raised in the literature and, in particular, of providing insight into the boundaries between tractable and intractable problems in the area.

10.1 A Unifying Framework

The fundamental property of all of the dependencies introduced so far is that they essentially say, "The presence of some tuples in the instance implies the presence of certain other tuples in the instance, or implies that certain tuple components are equal." In the case of jd's and mvd's, the *new* tuples can be completely specified in terms of the *old* tuples, but for ind's this is not the case. In any case, all of the dependencies discussed so far can be expressed using first-order logic sentences of the form

$$(*) \qquad \forall x_1 \ldots \forall x_n \; [\; \varphi(x_1, \ldots, x_n) \rightarrow \exists z_1 \ldots \exists z_k \psi(y_1, \ldots, y_m) \;],$$

where $\{z_1, \ldots, z_k\} = \{y_1, \ldots, y_m\} - \{x_1, \ldots, x_n\}$, and where φ is a (possibly empty) conjunction of atoms and ψ a nonempty conjunction. In both φ and ψ, one finds *relation atoms* of the form $R(w_1, \ldots, w_l)$ and *equality atoms* of the form $w = w'$, where each of the $w, w', w_1, \ldots, w_l$ is a variable.

Because we generally focus on sets of dependencies, we make several simplifying assumptions before continuing (see Exercise 10.1a). These include that (1) we may eliminate equality atoms from φ without losing expressive power; and (2) we can also assume without loss of generality that no existentially quantified variable participates in an equality atom in ψ. Thus we define an *(embedded) dependency* to be a sentence of the foregoing form, where

1. φ is a conjunction of relation atoms using all of the variables $x_1, \ldots, x_n$;
2. ψ is a conjunction of atoms using all of the variables $z_1, \ldots, z_k$; and
3. there are no equality atoms in ψ involving existentially quantified variables.

A dependency is *unirelational* if at most one relation name is used, and it is *multirelational* otherwise. To simplify the presentation, the focus in this chapter is almost exclusively on unirelational dependencies. Thus, unless otherwise indicated, the dependencies considered here are unirelational.

We now present three fundamental classifications of dependencies.

Full versus embedded: A *full* dependency is a dependency that has no existential quantifiers.

Tuple generating versus equality generating: A *tuple-generating dependency* (tgd) is a dependency in which no equality atoms occur; an *equality-generating dependency* (egd) is a dependency for which the right-hand formula is a *single* equality atom.

Typed versus untyped: A dependency is *typed* if there is an assignment of variables to column positions such that (1) variables in relation atoms occur only in their assigned position, and (2) each equality atom involves a pair of variables assigned to the same position.

It is sometimes important to distinguish dependencies with a single atom in the right-hand formula. A dependency is *single head* if the right-hand formula involves a single atom; it is *multi-head* otherwise.

The following result is easily verified (Exercise 10.1b).

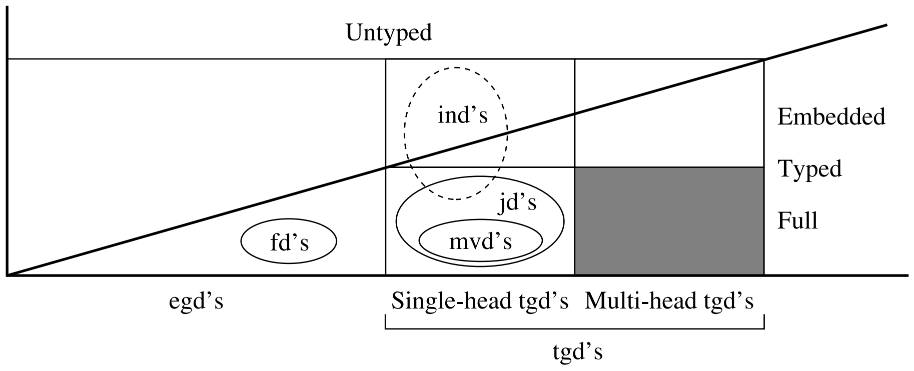

Figure 10.1: Dependencies

Proposition 10.1.1 Each (typed) dependency is equivalent to a set of (typed) egd's and tgd's.

It is easy to classify the fd's, jd's, mvd's, ejd's, emvd's and ind's studied in Chapters 8 and 9 according to the aforementioned dimensions. All except the last are typed. During the late 1970s and early 1980s the class of typed dependencies was studied in depth. In many cases, the results obtained for dependencies and for typed dependencies are equivalent. However, for negative results the typed case sometimes requires more sophisticated proof techniques because it imposes more restrictions.

A classification of dependencies along the three axes is given in Fig. 10.1. The gray square at the lower right indicates that each full multihead tgd is equivalent to a set of single-head tgd's. The intersection of ind's and jd's stems from trivial dependencies. For example, $R[AB] \subseteq R[AB]$ and $\bowtie[AB]$ over relation $R(AB)$ are equivalent [and are syntactically the same when written in the form of $(*)$].

There is a strong relationship between dependencies and tableaux. Tableaux provide a convenient notation for expressing and working with dependencies. (As will be seen in Section 10.4, the family of typed dependencies can also be represented using a formalism based on algebraic expressions.) The tableau representation of two untyped egd's is shown in Figs. 10.2(a) and 10.2(b). These two egd's are equivalent. Note that all egd's can be expressed as a pair $(T, x = y)$, where T is a tableau and $x, y \in var(T)$. If $(T, x = y)$ is typed, unirelational, and x, y are in the A column of T, then this is referred to as an *A-egd*.

Parts (c) and (d) of Fig. 10.2 show two full tgd's that are equivalent. This is especially interesting because, considered as tableau queries, (T', t) properly contains (T, t) (see Exercise 10.4). As suggested earlier, each full tgd is equivalent to some set of full single-head tgd's. In the following, when considering full tgd's, we will assume that they are single head.

Part (e) of Fig. 10.2 shows a typed tgd that is not single head. To represent these within

	A	B	C
S	x	y	w_1
	y	w_2	z
	z	y	w_3
		$x = z$	

(a) $(S, x = z)$

	A	B	C
S'	x	y	w_1
	y	w_2	u
	u	y	w_3
	y	w_4	z
	z	y	w_5
		$x = z$	

(b) $(S', x = z)$

	A	B
T	x	y_1
	x_1	y_1
	x_1	y
t	x	y

(c) (T, t)

	A	B
T'	x	y_1
	x_1	y_1
	x_1	y_2
	x_2	y_2
	x_2	y
t	x	y

(d) (T', t)

	A	B
T_1	x	y_1
	x_1	y_1
	x_1	y_2
	x'	y_2
T_2	x	y_3
	x'	y_3

(e) (T_1, T_2)

Figure 10.2: Five dependencies

the tableau notation, we use an ordered pair (T_1, T_2), where both T_1 and T_2 are tableaux. This tgd is not equivalent to any set of single-head tgd's (see Exercise 10.6b).

Finite versus Unrestricted Implication Revisited

We now reexamine the issues of finite versus unrestricted implication using the logical perspective on dependencies. Because all of these lie within first-order logic, $\models_{\text{fin}}$ is co-r.e. and $\models_{\text{unr}}$ is r.e. (see Chapter 2). Suppose that $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is a set of dependencies and $\{\sigma\}$ a dependency. Then $\Sigma \models_{\text{unr}} \sigma$ ($\Sigma \models_{\text{fin}} \sigma$) iff there is no unrestricted (finite) model of $\sigma_1 \wedge \cdots \wedge \sigma_n \wedge \neg\sigma$. If these are all full dependencies, then they can be rewritten in prenex normal form, where the quantifier prefix has the form $\exists^*\forall^*$. (Here each of the σ_i is universally quantified, and $\neg\sigma$ contributes the existential quantifier.) The family of sentences that have a quantifier prefix of this form (and no function symbols) is called the *initially extended Bernays-Schönfinkel class*, and it has been studied in the logic community since the 1920s. It is easily verified that finite and unrestricted satisfiability coincide for sentences in this class (Exercise 10.3). It follows that finite and unrestricted implication coincide for full dependencies and, as discussed in Chapter 9, it follows that implication is decidable.

On the other hand, because fd's and uind's are dependencies, we know from Theorem 9.2.4 that the two forms of implication do not coincide for (embedded) dependencies, and both are nonrecursive. Although not demonstrated here, these results have been extended to the family of embedded multivalued dependencies (emvd's).

To summarize:

Theorem 10.1.2

1. For full dependencies, finite and unrestricted implication coincide and are decidable.
2. For (typed) dependencies, finite and unrestricted implication do not coincide and are both undecidable. In fact, this is true for embedded multivalued dependencies. In particular, finite implication is not r.e., and unrestricted implication is not co-r.e.

10.2 The Chase Revisited

As suggested by the close connection between dependencies and tableaux, chasing is an invaluable tool for characterizing logical implication for dependencies. In this section we first use chasing to develop a test for logical implication of arbitrary dependencies by full dependencies. We also present an application of the chase for determining how full dependencies are propagated to views. We conclude by extending the chase to work with embedded dependencies. In this discussion we focus almost entirely on typed dependencies, but it will be clear that the arguments can be modified to the untyped case.

Chasing with Full Dependencies

We first state without proof the natural generalization of chasing by fd's and jd's (Theorem 8.4.12) to full dependencies (see Exercise 10.8). In this context we begin either with a tableau T, or with an arbitrary tgd (T, T') or egd $(T, x = y)$. The notion of *applying* a full dependency to this is defined in the natural manner. Lemma 8.4.17 and the notation developed for it generalize naturally to this context, as does the following analog of Theorem 8.4.18:

Theorem 10.2.1 If Σ is a set of full dependencies and T is a tableau (τ a dependency), then chasing $T(\tau)$ by Σ yields a unique finite result, denoted *chase*(T, Σ) (*chase*(τ, Σ)).

Logical implication of (full or embedded) dependencies by sets of full dependencies will now be characterized by a straightforward application of the techniques developed in Section 8.4 (see Exercise 10.8). A dependency τ is *trivial* if

(a) τ is an egd $(T, x = x)$; or

(b) τ is a tgd (T, T') and there is a substitution θ for T' such that $\theta(T') \subseteq T$ and θ is the identity on $var(T) \cap var(T')$.

Note that if τ is a *full* tgd, then (b) simply says that $T' \subseteq T$.

A dependency τ is a *tautology* for finite (unrestricted) instances if each finite (unrestricted) instance of appropriate type satisfies τ—that is, if $\emptyset \models_{\text{fin}} \tau$ ($\emptyset \models_{\text{unr}} \tau$). It is easily verified that a dependency is a tautology iff it is trivial.

The following now provides a simple test for implication by full typed dependencies:

Theorem 10.2.2 Let Σ be a set of full typed dependencies and τ a typed dependency. Then $\Sigma \models \tau$ iff $chase(\tau, \Sigma)$ is trivial.

Recall that the chase relies on a total order $\leq$ on **var**. For egd $(T, x = y)$ we assume that $x < y$ and that these are the least and second to least variables appearing in the tableau; and for full tgd (T, t), $t(A)$ is least in $T(A)$ for each attribute A. Using this convention, we can obtain the following:

Corollary 10.2.3 Let Σ be a set of full typed dependencies.

(a) If $\tau = (T, x = y)$ is a typed egd, then $\Sigma \models \tau$ iff x and y are identical or $y \notin var(chase(T, \Sigma))$.

(b) If $\tau = (T, t)$ is a full typed tgd, then $\Sigma \models \tau$ iff $t \in chase(T, \Sigma)$.

Using the preceding results, it is straightforward to develop a deterministic exponential time algorithm for testing implication of full dependencies. It is also known that for both the typed and untyped cases, implication is complete in EXPTIME. (Note that, in contrast, logical implication for arbitrary sets of initially extended Bernays-Schöfinkel sentences is known to be complete in nondeterministic EXPTIME.)

Dependencies and Views

On a bit of a tangent, we now apply the chase to characterize the interaction of full dependencies and user views. Let $\mathbf{R} = \{R_1, \ldots, R_n\}$ be a database schema, where R_j has associated set Σ_j of full dependencies for $j \in [1, n]$. Set $\Sigma = \{R_i : \sigma \mid \sigma \in \Sigma_i\}$. Note that the elements of Σ are tagged by the relation name they refer to. Suppose that a view is defined by algebraic expression $E : \mathbf{R} \to S[V]$. It is natural to ask what dependencies will hold in the view. Formally, we say that $\mathbf{R} : \Sigma$ *implies* $E : \sigma$, denoted $\mathbf{R} : \Sigma \models E : \sigma$, if $E(\mathbf{I})$ satisfies σ for each $\mathbf{I}$ that satisfies Σ. The notion of $\mathbf{R} : \Sigma \models E : \Gamma$ for a set Γ is defined in the natural manner.

To illustrate these notions in a simple setting, we state the following easily verified result (see Exercise 10.10).

Proposition 10.2.4 Let $(R[U], \Sigma)$ be a relation schema where Σ is a set of fd's and mvd's, and let $V \subseteq U$. Then

(a) $R : \Sigma \models [\pi_V(R)] : X \to A$ iff $\Sigma \models X \to A$ and $XA \subseteq V$.

(b) $R : \Sigma \models [\pi_V(R)] : X \twoheadrightarrow Y$ iff $\Sigma \models X \twoheadrightarrow Z$ for some $X \subseteq V$ and $Y = Z \cap V$.

Given a database schema $\mathbf{R}$, a family Σ of tagged full dependencies over $\mathbf{R}$, a view expression E mapping $\mathbf{R}$ to $S[V]$, and a full dependency γ, is it decidable whether $\mathbf{R} : \Sigma \models E : \gamma$? If E ranges over the full relational algebra, the answer is no, even if the only dependencies considered are fd's.

Theorem 10.2.5 It is undecidable, given database schema $\mathbf{R}$, tagged fd's Σ, algebra expression $E : \mathbf{R} \to S$ and fd σ over S, whether $\mathbf{R} : \Sigma \models E : \sigma$.

Proof Let $\mathbf{R} = \{R[U], S[U]\}$, $\sigma = R : \emptyset \to U$ and $\Sigma = \{\sigma\}$. Given two algebra expressions $E_1, E_2 : S \to R$, consider

$$E = R \cup [E_1(S) - E_2(S)] \cup [E_2(S) - E_1(S)]$$

Then $\mathbf{R} : \Sigma \models E : \sigma$ iff $E_1 \equiv E_2$. This is undecidable by Corollary 6.3.2. ■

In contrast, we now present a decision procedure, based on the chase, for inferring view dependencies when the view is defined using the SPCU algebra.

Theorem 10.2.6 It is decidable whether $\mathbf{R} : \Sigma \models E : \gamma$, if E is an SPCU query and $\Sigma \cup \{\gamma\}$ is a set of (tagged) full dependencies.

Crux We prove the result for SPC queries that do not involve constants, and leave the extension to include union and constants for the reader (Exercise 10.12).

Let $E : \mathbf{R} \to S[V]$ be an SPC expression, where $S \notin \mathbf{R}$. Recall from Chapter 4 (Theorem 4.4.8; see also Exercise 4.18) that for each such expression E there is a tableau mapping $\tau_E = (\mathbf{T}, t)$ equivalent to E.

Assume now that Σ is a set of full dependencies and γ a full tgd. (The case where γ is an egd is left for the reader.) Let the tgd γ over S be expressed as the tableau (W, w). Create a new free instance $\mathbf{Z}$ out of $(\mathbf{T}, t)$ and W as follows: For each tuple $u \in W$, set $\mathbf{T}_u = \nu(\mathbf{T})$ where valuation ν maps t to u, and maps all other variables in $\mathbf{T}$ to new distinct variables. Set $\mathbf{Z} = \cup_{u \in W} \mathbf{T}_u$. It can now be verified that $\mathbf{R} : \Sigma \models E : \gamma$ iff $w \in E(\mathit{chase}(\mathbf{Z}, \Sigma))$. ■

In the case where $\Sigma \cup \{\gamma\}$ is a set of fd's and mvd's and the view is defined by an SPCU expression, testing the implication of a view dependency can be done in polynomial time, if jd's are involved the problem is NP-complete, and if full dependencies are considered the problem is EXPTIME-complete.

Recall from Section 8.4 that a *satisfaction family* is a family $\mathit{sat}(\mathbf{R}, \Sigma)$ for some set Σ of dependencies. Suppose now that SPC expression $E : R[U] \to S[V]$ is given, and that Σ is a set of full dependencies over R. Theorem 10.2.6, suitably generalized, shows that the family Γ of full dependencies implied by Σ for view E is recursive. This raises the natural question: Does $E(\mathit{sat}(R, \Sigma)) = \mathit{sat}(\Gamma)$, that is, does Γ completely characterize the image of $\mathit{sat}(R, \Sigma)$ under E? The affirmative answer to this question is stated next. This result follows from the proof of Theorem 10.2.6 (see Exercise 10.13).

Theorem 10.2.7 If Σ is a set of full dependencies over $\mathbf{R}$ and $E : \mathbf{R} \rightarrow S$ is an SPC expression without constants, then there is a set Γ of full dependencies over S such that $E(sat(\mathbf{R}, \Sigma)) = sat(S, \Gamma)$.

Suppose now that $E : R[U] \rightarrow S[V]$ is given, and Σ is a *finite set* of dependencies. Can a *finite* set Γ be found such that $E(sat(R, \Sigma)) = sat(S, \Gamma)$? Even in the case where E is a simple projection and Σ is a set of fd's, the answer to this question is sometimes negative (Exercise 10.11c).

Chasing with Embedded Dependencies

We now turn to the case of (embedded) dependencies. From Theorem 10.1.2(b), it is apparent that we cannot hope to generalize Theorem 10.2.2 to obtain a decision procedure for (finite or unrestricted) implication of dependencies. As initially discussed in Chapter 9, the chase need not terminate if dependencies are used. All is not lost, however, because we are able to use the chase to obtain a proof procedure for testing unrestricted implication of a dependency by a set of dependencies.

For nonfull tgd's, we shall use the following rule. We present the rule as it applies to tableaux, but it can also be used on dependencies.

tgd rule: Let T be a tableau, and let $\sigma = (S, S')$ be a tgd. Suppose that there is a valuation θ for S that embeds S into T, but no extension θ' to $var(S) \cup var(S')$ of θ such that $\theta'(S') \subseteq T$. In this case σ can be *applied* to T.

Let $\theta_1, \ldots, \theta_n$ be a list of all valuations having this property. For each $i \in [1, n]$, (nondeterministically) choose a *distinct extension*, i.e., an extension θ'_i to $var(S) \cup var(S')$ of θ_i such that each variable in $var(S') - var(S)$ is assigned a distinct new variable greater than all variables in T. (The same variable is not chosen in two extensions $\theta'_i, \theta'_j, i \neq j$.)

The *result of applying* σ to T is $T \cup \{\theta'_i(S') \mid i \in [1, n]\}$.

This rule is nondeterministic because variables not occurring in T are chosen for the existentially quantified variables of σ. We assume that some fixed mechanism is used for selecting these variables when given T, (S, S'), and θ.

The notion of a chasing sequence $T = T_1, T_2, \ldots$ of a tableau (or dependency) by a set of dependencies is now defined in the obvious manner. Clearly, this sequence may be infinite.

Example 10.2.8 Let $\Sigma = \{\tau_1, \tau_2, \tau_3\}$, where

$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
T & w & x & & \\
 & w & & y & \\
\hline
t & & x & y &
\end{array}
\qquad
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
T' & w & & y & z \\
 & & x & y & \\
\hline
t' & w & x & & z
\end{array}
\qquad
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
T'' & & x & & z \\
 & & x & & z' \\
\hline
 & & \multicolumn{3}{c}{z = z'}
\end{array}
$$

$$\tau_1 \qquad\qquad\qquad \tau_2 \qquad\qquad\qquad \tau_3$$

A	B	C	D	
x_1	x_2	x_3	x_4	
x_1	x_5	x_6	x_7	
x_{10}	x_2	x_6	x_{12}	application
x_{11}	x_5	x_3	x_{13}	of τ_1

(a)

A	B	C	D	
x_1	x_2	x_3	x_4	
x_1	x_5	x_6	x_7	
x_{10}	x_2	x_6	x_4	application
x_{11}	x_5	x_3	x_7	of τ_3

(b)

A	B	C	D	
x_1	x_2	x_3	x_4	
x_1	x_5	x_6	x_7	
x_{10}	x_2	x_6	x_4	
x_{11}	x_5	x_3	x_7	
x_1	x_5	x_{20}	x_4	
x_{11}	x_2	x_{21}	x_7	
x_1	x_2	x_{22}	x_7	application
x_{10}	x_5	x_{23}	x_4	of τ_2

(c)

A	B	C	D	
x_1	x_2	x_3	x_4	
x_1	x_5	x_6	x_4	
x_{10}	x_2	x_6	x_4	
x_{11}	x_5	x_3	x_4	
x_1	x_5	x_{20}	x_4	
x_{11}	x_2	x_{21}	x_4	
x_1	x_2	x_{22}	x_4	application
x_{10}	x_5	x_{23}	x_4	of τ_3

(d)

Figure 10.3: Parts of a chasing sequence

We show here only the relevant variables of τ_1, τ_2, and τ_3; all other variables are assumed to be distinct. Here $\tau_3 \equiv B \to D$.

In Fig. 10.3, we show some stages of a chasing sequence that demonstrates that $\Sigma \models_{\text{unr}} A \to D$. To do that, the chase begins with the tableau $\{\langle x_1, x_2, x_3, x_4\rangle, \langle x_1, x_5, x_6, x_7\rangle\}$. Figure 10.3 shows the results of applying $\tau_1, \tau_3, \tau_2, \tau_3$ in turn (left to right). This sequence implies that $\Sigma \models_{\text{unr}} A \to D$, because variables x_4 and x_7 are identified.

Consider now the typed tgd's:

	A	B	C	D
T'''	w	x		z
	w		y	
t''		x	y	z

τ_4

	A	B	C	D
T''''	w	x		
	w		y	
t'''	w	x	y	

τ_5

The chasing sequence of Fig. 10.3 also implies that $\Sigma \models_{\text{unr}} \tau_4$, because (x_{10}, x_2, x_6, x_4) is in the second tableau. On the other hand, we now argue that $\Sigma \not\models_{\text{unr}} \tau_5$. Consider the chasing sequence beginning as the one shown in Fig. 10.3, and continuing by applying the sequence $\tau_1, \tau_3, \tau_2, \tau_3$ repeatedly. It can be shown that this chasing sequence will not terminate and that (x_1, x_2, x_6, v) does not occur in the resulting infinite sequence for any variable v (see Exercise 10.16). It follows that $\Sigma \not\models_{\text{unr}} \tau_5$; in particular, the infinite result of the chasing sequence is a counterexample to this implication. On the other hand, this chasing sequence does not alone provide any information about whether $\Sigma \models_{\text{fin}} \tau_5$. It can be shown that this also fails.

To ensure that all relevant dependencies have a chance to influence a chasing sequence, we focus on chasing sequences that satisfy the following conditions:

(1) Whenever an egd is applied, it is applied repeatedly until it is no longer applicable.

(2) No dependency is "starved" (i.e., each dependency that is applicable infinitely often is applied infinitely often).

Even if these conditions are satisfied, it is possible to have two chasing sequences of a tableau T by typed dependencies, where one is finite and the other infinite (see Exercise 10.14).

Now consider an infinite chasing sequence $T_1 = T, T_2, \ldots$. Let us denote it by $\overline{T, \Sigma}$. Because egd's may be applied arbitrarily late in $\overline{T, \Sigma}$, for each n, tuples of T_n may be modified as the result of later applications of egd's. Thus we cannot simply take the union of some tail $T_n, T_{n+1}, \ldots$ to obtain the result of the chase. As an alternative, for the chasing sequence $\overline{T, \Sigma} = T_1, T_2, \ldots$, we define

$$chase(\overline{T, \Sigma}) = \{u \mid \exists n\ \forall m > n (u \in T_m)\}.$$

This is nonempty because (1) the "new" variables introduced by the tgd rule are always greater than variables already present; and (2) when the egd rule is applied, the newer variable is replaced by the older one.

By generalizing the techniques developed, it is easily seen that the (possibly infinite) resulting tableau satisfies all dependencies in Σ. More generally, let Σ be a set of dependencies and σ a dependency. Then one can show that $\Sigma \models_{\text{unr}} \sigma$ iff for some chasing sequence $\overline{\sigma, \Sigma}$ of σ using Σ, $chase(\overline{\sigma, \Sigma})$ is trivial. Furthermore, it can be shown that

- if for some chasing sequence $\overline{\sigma, \Sigma}$ of σ using Σ, $chase(\overline{\sigma, \Sigma})$ is trivial, then it is so for *all* chasing sequences of σ using Σ; and
- for each chasing sequence $\overline{\sigma, \Sigma} = T_1, \ldots, T_n, \ldots$ of σ using Σ, $chase(\overline{\sigma, \Sigma})$ is trivial iff T_i is trivial for some i.

This shows that, for practical purposes, it suffices to generate some chasing sequence of σ using Σ and stop as soon as some tableau in the sequence becomes trivial.

10.3 Axiomatization

A variety of axiomatizations have been developed for the family of dependencies and for subclasses such as the full typed tgd's. In view of Theorem 10.1.2, sound and complete recursively enumerable axiomatizations do not exist for finite implication of dependencies. This section presents an axiomatization for the family of full typed tgd's and typed egd's (which is sound and complete for both finite and unrestricted implication). A generalization to the embedded case (for unrestricted implication) has also been developed (see Exercise 10.21). The axiomatization presented here is closely related to the chase. In the next section, a very different kind of axiomatization for typed dependencies is discussed.

We now focus on the full typed dependencies (i.e., on typed egd's and full typed tgd's). The development begins with the introduction of a technical tool for forming the composition of tableaux queries. The axiomatization then follows.

Composition of Typed Tableaux

Suppose that $\tau = (T, t)$ and $\sigma = (S, s)$ are two full typed tableau queries over relation schema R. It is natural to ask whether there is a tableau query $\tau \bullet \sigma$ corresponding to the composition of τ followed by σ—that is, with the property that for each instance I over R,

$$(\tau \bullet \sigma)(I) = \sigma(\tau(I))$$

and, if so, whether there is a simple way to construct it. We now provide an affirmative answer to both questions. The syntactic composition of full typed tableau mappings will be a valuable tool for combining pairs of full typed tgd's in the axiomatization presented shortly.

Let $T = \{t_1, \ldots, t_n\}$ and $S = \{s_1, \ldots, s_m\}$. Suppose that tuple w is in $\sigma(\tau(I))$. Then there is an embedding ν of $s_1, \ldots, s_m$ into $\tau(I)$ such that $\nu(s) = w$. It follows that for each $j \in [1, m]$ there is an embedding μ_j of T into I, with $\mu_j(t) = \nu(s_j)$. This suggests that the tableau of $\tau \bullet \sigma$ should have mn tuples, with a block of n tuples for each s_j.

To be more precise, for each $j \in [1, m]$, let T_{s_j} be $\theta_j(T)$, where θ_j is a substitution that maps $t(A)$ to $s_j(A)$ for each attribute A of R and maps each other variable of T to a new, distinct variable not used elsewhere in the construction. Now set

$$[S](T, t) \equiv \cup\{T_{s_j} \mid j \in [1, m]\} \quad \text{and} \quad \tau \bullet \sigma \equiv ([S](T, t), s).$$

The following is now easily verified (see Exercise 10.18):

Proposition 10.3.1 For full typed tableau queries τ and σ over R, and for each instance I of R, $\tau \bullet \sigma(I) = \sigma(\tau(I))$.

Example 10.3.2 The following table shows two full typed tableau queries and their composition.

$$\begin{array}{c|ccc} & A & B & C \\ \hline T & x & y & z' \\ & x & y' & z \\ & x & y' & z'' \\ \hline t & w & x & y \end{array} \qquad \begin{array}{c|ccc} & A & B & C \\ \hline S & u & v & w' \\ & u' & v & w \\ \hline s & u & v & w \end{array} \qquad \begin{array}{|ccc} A & B & C \\ \hline u & v & p_1 \\ u & p_2 & w' \\ u & p_2 & p_3 \\ u' & v & p_4 \\ u' & p_5 & w \\ u' & p_5 & p_6 \\ \hline u & v & w \end{array}$$

$$\tau \qquad\qquad \sigma \qquad\qquad \tau \bullet \sigma$$

It is straightforward to verify that the syntactic operation of composition is associative.

Suppose that τ and σ are full typed tableau queries. It can be shown by simple chasing arguments that $\{\tau, \sigma\}$ and $\{\tau \bullet \sigma\}$ are equivalent as sets of dependencies. It follows that full typed tgd's are closed under finite conjunction, in the sense that each finite set of full typed tgd's over a relation schema R is equivalent to a single full typed tgd. This property does not hold in the embedded case (see Exercise 10.20).

An Axiomatization for Full Typed Dependencies

For full typed tgd's, $\tau = (T, t)$ and $\sigma = (S, s)$, we say that τ *embeds into* σ denoted $\tau \hookrightarrow \sigma$, if there is a substitution ν such that $\nu(T) \subseteq S$ and $\nu(t) = s$. Recall from Chapter 4 that $\tau \supseteq \sigma$ (considered as tableau queries) iff $\tau \hookrightarrow \sigma$. As a result we have that if $\tau \hookrightarrow \sigma$, then $\tau \models \sigma$, although the converse does not necessarily hold. Analogously, for A-egd's $\tau = (T, x = y)$ and $\sigma = (S, v = w)$, we define $\tau \hookrightarrow \sigma$ if there is a substitution ν such that $\nu(T) \subseteq S$, and $\nu(\{x, y\}) = \{v, w\}$. Again, if $\tau \hookrightarrow \sigma$, then $\tau \models \sigma$.

We now list the axioms for full typed tgd's:

FTtgd1: (triviality) For each free tuple t without constants, $(\{t\}, t)$.

FTtgd2: (embedding) If τ and $\tau \hookrightarrow \sigma$, then σ.

FTtgd3: (composition) If τ and σ, then $\tau \bullet \sigma$.

The following rules focus exclusively on typed egd's:

Tegd1: (triviality) If $x \in var(T)$, then $(T, x = x)$.

Tegd2: (embedding) If τ and $\tau \hookrightarrow \sigma$, then σ.

The final rules combining egd's and full typed tgd's use the following notation. Let $R[U]$ be a relation schema. For $A \in U$, $\overline{A}$ denotes $U - \{A\}$. Given typed A-egd $\tau = (T, x = y)$ over R, define free tuples u_x, u_y such that $u_x(A) = x$, $u_y(A) = y$ and $u_x[\overline{A}] = u_y[\overline{A}]$ consists of distinct variables not occurring in T. Define two full typed tgd's $\tau_x = (T \cup \{u_y\}, u_x)$ and $\tau_y = (T \cup \{u_x\}, u_y)$.

FTD1: (conversion) If $\tau = (T, x = y)$, then τ_x and τ_y.

FTD2: (composition) If (T, t) and $(S, x = y)$, then $([S](T, t), x = y)$.

We now have the following:

Theorem 10.3.3 The set {FTtgd1, FTtgd2, FTtgd3, Tegd1, Tegd2, FTD1, FTD2} is sound and complete for (finite and unrestricted) logical implication of full typed dependencies.

Crux Soundness is easily verified. We illustrate completeness by showing that the FTtgd rules are complete for tgd's. Suppose that $\Sigma \models \tau = (T, t)$, where Σ is a set of full typed tgd's and (T, t) is full and typed. By Theorem 10.2.2 there is a chasing sequence of T by Σ yielding T' with $t \in T'$. Let $\sigma_1, \ldots, \sigma_n$ $(n \geq 0)$ be the sequence of elements of Σ used in the chasing sequence. It follows that $t \in \sigma_n(\ldots(\sigma_1(T)\ldots)$, and by Proposition 10.3.1, $t \in (\sigma_1 \bullet \cdots \bullet \sigma_n)(T)$. This implies that $(\sigma_1 \bullet \cdots \bullet \sigma_n) \hookrightarrow (T, t)$. A proof of τ from Σ is now obtained by starting with σ_1 (or $(\{s\}, s)$ if $n = 0$), followed by $n - 1$ applications of FTtgd3 and one application of FTtgd2 (see Exercise (10.18b). ■

The preceding techniques and the chase can be used to develop an axiomatization of unrestricted implication for the family of all typed dependencies.

10.4 An Algebraic Perspective

This section develops a very different paradigm for specifying dependencies based on the use of algebraic expressions. Surprisingly, the class of dependencies formed is equivalent to the class of typed dependencies. We also present an axiomatization that is rooted primarily in algebraic properties rather than chasing and tableau manipulations.

We begin with examples that motivate and illustrate this approach.

Example 10.4.1 Let $R[ABCD]$ be a relation schema. Consider the tgd τ of Fig. 10.4 and the algebraic expression

$$\pi_{AC}(\pi_{AB}(R) \bowtie \pi_{BC}(R)) \subseteq \pi_{AC}(R).$$

It is straightforward to verify that for each instance I over $ABCD$,

$$I \models \tau \text{ iff } \pi_{AC}(\pi_{AB}(I) \bowtie \pi_{BC}(I)) \subseteq \pi_{AC}(I).$$

Now consider dependency σ. One can similarly verify that for each instance I over $ABCD$,

$$I \models \sigma \text{ iff } \pi_{AC}(\pi_{AB}(I) \bowtie \pi_{BC}(I)) \subseteq \pi_{AC}(\pi_{AD}(I) \bowtie \pi_{CD}(I)).$$

$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
T & x & y' & & \\
 & & y' & z & \\
\hline
t & x & & z & \\
\end{array}
\qquad\qquad
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
S & x & y' & & \\
 & & y' & z & \\
\hline
S' & x & & & w' \\
 & & & z & w' \\
\end{array}
$$

$$\tau \qquad\qquad\qquad\qquad \sigma$$

Figure 10.4: Dependencies of Example 10.4.1

The observation of this example can be generalized in the following way. A *project-join* (PJ) expression is an algebraic expression over a single relation schema using only projection and natural join. We describe next a natural recursive algorithm for translating PJ expressions into tableau queries (see Exercise 10.23). (This algorithm is also implicit in the equivalence proofs of Chapter 4.)

ALGORITHM 10.4.2

Input: a PJ expression E over relation schema $R[A_1, \ldots, A_n]$

Output: a tableau query (T, t) equivalent to E

Basis: If E is simply R, then return $(\{\langle x_1, \ldots, x_n\rangle\}, \langle x_1, \ldots, x_n\rangle)$.

Inductive steps:

1. If E is $\pi_X(q)$ and the tableau query of q is (T, t), then return $(T, \pi_X(t))$.
2. Suppose E is $q_1 \bowtie q_2$ and the tableau query of q_i is (T_i, t_i) for $i \in [1, 2]$.

   Let X be the intersection of the output sorts of q_1 and q_2. Assume without loss of generality that the two tableaux use distinct variables except that $t_1(A) = t_2(A)$ for $A \in X$. Then return $(T_1 \cup T_2, t_1 \bowtie t_2)$.

Suppose now that (T, T') is a typed dependency with the property that for some free tuple t, (T, t) is the tableau associated by this algorithm with PJ expression E, and (T', t) is the tableau associated with PJ expression E'. Suppose also that the only variables common to T and T' are those in t. Then for each instance I, $I \models (T, T')$ iff $E(I) \subseteq E'(I)$.

This raises three natural questions: (1) Is the family of PJ inclusions equivalent to the set of typed tgd's? (2) If not, can this paradigm be extended to capture all typed tgd's? (3) Can this paradigm be extended to capture typed egd's as well as tgd's?

The answer to the first question is no (see Exercise 10.24).

The answer to the second and third questions is yes. This relies on the notion of *extended relations* and *extended project-join expressions*. Let $R[A_1, \ldots, A_n]$ be a relation schema. For each $i \in [1, n]$, we suppose that there is an infinite set of attributes $A_i^1, A_i^2, \ldots,$ called *copies* of A_i. The *extended schema* of R is the schema $\overline{R}[A_1^1, \ldots, A_n^1, A_1^2, \ldots, A_n^2, \ldots]$. For an instance I of R, the *extended instance* of $\overline{R}$ corresponding to I, denoted $\overline{I}$, has one "tuple" $\overline{u}$ for each tuple $u \in I$, where $\overline{u}(A_i^j) = u(A_i)$ for each $i \in [1, n]$ and $j > 0$.

An *extended project-join expression* over R is a PJ expression over $\overline{R}$ such that a

$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
T & x & & z & w' \\
 & & & z' & w' \\
 & x' & & z' & w \\
\hline
T' & x & y' & & \\
 & & y' & z & w
\end{array}
\qquad\qquad
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
T & x & & z & w' \\
 & & & z' & w' \\
 & x' & & z' & w \\
\hline
 & \multicolumn{4}{c}{x = x'}
\end{array}
$$

$$\tau \qquad\qquad\qquad\qquad \sigma$$

Figure 10.5: tgd and egd of Example 10.4.3

projection operator is applied first to each occurrence of $\overline{R}$. (This ensures that the evaluation and the result of such expressions involve only finite objects.) Given two extended PJ expressions E and E' with the same target sort, and instance I over R, $E(I) \subseteq_e E'(I)$ denotes $E(\overline{I}) \subseteq E'(\overline{I})$.

An *algebraic dependency* is a syntactic expression of the form $E \subseteq_e E'$, where E and E' are extended PJ expressions over a relation schema R with the same target sort. An instance I over R *satisfies* $E \subseteq_e E'$ if $E(I) \subseteq_e E'(I)$—that is, if $E(\overline{I}) \subseteq E'(\overline{I})$.

This is illustrated next.

Example 10.4.3 Consider the dependency τ of Fig. 10.5. Let

$$E = \pi_{ACD^1}(\overline{R}) \bowtie \pi_{C^1D^1}(\overline{R}) \bowtie \pi_{A^1C^1D}(\overline{R}).$$

Here we use $A, A^1, \ldots$ to denote different copies the attribute A, etc.

It can be shown that, for each instance I over $ABCD$, $I \models \tau$ iff $E_1(\overline{I}) \subseteq_e E_2(\overline{I})$, where

$$E_1 = \pi_{ACD}(E)$$
$$E_2 = \pi_{ACD}(\pi_{AB^1}(\overline{R}) \bowtie \pi_{B^1CD}(\overline{R})).$$

(See Exercise 10.25).

Consider now the functional dependency $A \to BC$ over $ABCD$. This is equivalent to $\pi_{ABC}(\overline{R}) \bowtie \pi_{AB^1C^1}(\overline{R}) \subseteq_e \pi_{ABCB^1C^1}(\overline{R})$.

Finally, consider σ of Fig. 10.5. This is equivalent to $F_1 \subseteq_e F_2$, where

$$F_1 = \pi_{AA^1}(E)$$
$$F_2 = \pi_{AA^1}(\overline{R}).$$

We next see that algebraic dependencies correspond precisely to typed dependencies.

Theorem 10.4.4 For each algebraic dependency, there is an equivalent typed dependency, and for each typed dependency, there is an equivalent algebraic dependency.

Crux Let $R[A_1, \ldots, A_n]$ be a relation schema, and let $E \subseteq_e E'$ be an algebraic dependency over R, where E and E' have target sort X. Without loss of generality, we can assume that there is k such that the sets of attributes involved in E and E' are contained in $\widehat{U} = \{A_1^1, \ldots, A_n^1, \ldots, A_1^k, \ldots, A_n^k\}$. Using Algorithm 10.4.2, construct tableau queries $\tau = (T, t)$ and $\tau' = (T', t')$ over $\widehat{U}$ corresponding to E and E'. We assume without loss of generality that τ and τ' do not share any variables except that $t(A) = t'(A)$ for each $A \in X$.

Consider T (over $\widehat{U}$). For each tuple $s \in T$ and $j \in [1, k]$,

- construct an atom $R(x_1, \ldots, x_n)$, where $x_i = s(A_i^j)$ for each $i \in [1, n]$;
- construct atoms $s(A_i^j) = s(A_i^{j'})$ for each $i \in [1, n]$ and j, j' satisfying $1 \leq j < j' \leq k$.

Let $\varphi(x_1, \ldots, x_p)$ be the conjunction of all atoms obtained from τ in this manner. Let $\psi(y_1, \ldots, y_q)$ be constructed analogously from τ'. It can now be shown (Exercise 10.26) that $E \subseteq_e E'$ is equivalent to the typed dependency

$$\forall x_1 \ldots x_p(\varphi(x_1, \ldots, x_p) \rightarrow \exists z_1 \ldots z_r \psi(y_1, \ldots, y_q)),$$

where $z_1, \ldots, z_r$ is the set of variables in $\{y_1, \ldots, y_q\} - \{x_1, \ldots, x_p\}$.

For the converse, we generalize the technique used in Example 10.4.3. For each attribute A, one distinct copy of A is used for each variable occurring in the A column. ■

An Axiomatization for Algebraic Dependencies

Figure 10.6 shows a family of inference rules for algebraic dependencies. Each of these rules stems from an algebraic property of join and project, and only the last explicitly uses a property of extended instances. (It is assumed here that all expressions are well formed.)

The use of these rules to infer dependencies is considered in Exercises 10.31, and 10.32.

It can be shown that:

Theorem 10.4.5 The family {AD1, ..., AD8} is sound and complete for inferring unrestricted implication of algebraic dependencies.

To conclude this discussion of the algebraic perspective on dependencies, we consider a new operation, direct product, and the important notion of faithfulness.

Faithfulness and Armstrong Relations

We show now that sets of typed dependencies have Armstrong relations,[1] although these may sometimes be infinite. To accomplish this, we first introduce a new way to combine instances and an important property of it.

[1] Recall that given a set Σ of dependencies over some schema R, an Armstrong relation for Σ is an instance I over R that satisfies Σ and violates every dependency not implied by Σ.

AD1: (Idempotency of Projection)
(a) $\pi_X(\pi_Y E) =_e \pi_X E$
(b) $\pi_{sort(E)} E =_e E$

AD2: (Idempotency of Join)
(a) $E \bowtie \pi_X E =_e E$
(b) $\pi_{sort(E)}(E \bowtie E') \subseteq_e E$

AD3: (Monotonicity of Projection)
If $E \subseteq_e E'$ then $\pi_X E \subseteq_e \pi_X E'$

AD4: (Monotonicity of Join)
If $E \subseteq_e E'$, then $E \bowtie E'' \subseteq_e E' \bowtie E''$

AD5: (Commutativity of Join)
$E \bowtie E' =_e E' \bowtie E$

AD6: (Associativity of Join)
$(E \bowtie E') \bowtie E'' =_e E \bowtie (E' \bowtie E'')$

AD7: (Distributivity of Projection over Join)
Suppose that $X \subseteq sort(E)$ and $Y \subseteq sort(E')$. Then
(a) $\pi_{X \cup Y}(E \bowtie E') \subseteq_e \pi_{X \cup Y}(E \bowtie \pi_Y E')$.
(b) If $sort(E) \cap sort(E') \subseteq Y$, then equality holds in (a).

AD8: (Extension)
If $X \subseteq sort(\overline{R})$ and A, A' are copies of the same attribute, then
$\pi_{AA'}\overline{R} \bowtie \pi_{AX}\overline{R} =_e \pi_{AA'X}\overline{R}$.

Figure 10.6: Algebraic dependency axioms

Let R be a relation schema of arity n. We blur our notation and use elements of **dom** × **dom** as if they were elements of **dom**. Given tuples $u = \langle x_1, \ldots, x_n \rangle$ and $v = \langle y_1, \ldots, y_n \rangle$, we define the *direct product* of u and v to be

$$u \otimes v = \langle (x_1, y_1), \ldots, (x_n, y_n) \rangle.$$

The *direct product* of two instances I, J over R is

$$I \otimes J = \{u \otimes v \mid u \in I, v \in J\}.$$

This is generalized to form k-ary direct product instances for each finite k. Furthermore, if $\mathcal{J}$ is a (finite or infinite) index set and $\{I_j \mid j \in \mathcal{J}\}$ is a family of instances over R, then $\otimes\{I_j \mid j \in \mathcal{J}\}$ denotes the (possibly infinite) direct product of this family of instances.

A dependency σ is *faithful* if for each family $\{I_j \mid j \in \mathcal{J}\}$ of nonempty instances,

$$\otimes\{I_j \mid j \in \mathcal{J}\} \models \sigma \text{ if and only if } \forall j \in \mathcal{J}, I_j \models \sigma.$$

(The restriction that the instances be nonempty is important—if this were omitted then no nontrivial dependency would be faithful.)

The following holds because the $\otimes$ operator commutes with project, join, and "extension" (see Exercise 10.29).

Proposition 10.4.6 The family of typed dependencies is faithful.

We can now prove that each set of typed dependencies has an Armstrong relation.

Theorem 10.4.7 Let Σ be a set of typed dependencies over relation R. Then there is a (possibly infinite) instance I_Σ such that for each typed dependency σ over R, $I_\Sigma \models \sigma$ iff $\Sigma \models_{\text{unr}} \sigma$.

Proof Let Γ be the set of typed dependencies over R not in Σ^*. For each $\gamma \in \Gamma$, let I_γ be a nonempty instance that satisfies Σ but not γ. Then $\otimes\{I_\gamma \mid \gamma \in \Gamma\}$ is the desired relation. ■

This result cannot be strengthened to yield finite Armstrong relations because one can exhibit a finite set of typed tgd's with no finite Armstrong relation.

Bibliographic Notes

The papers [FV86, Kan91, Var87] all provide excellent surveys on the motivations and history of research into relational dependencies; these have greatly influenced our treatment of the subject here.

Because readers could be overwhelmed by the great number of dependency theory terms we have used a subset of the terminology. For instance, the typed single-head tgd's (that were studied in depth) are called *template* dependencies. In addition, the typed unirelational dependencies that are considered here were historically called *embedded implicational dependencies* (eid's); and their full counterparts were called *implicational dependencies* (id's). We use this terminology in the following notes.

After the introduction of fd's and mvd's, there was a flurry of research into special classes of dependencies, including jd's and ind's. *Embedded* dependencies were first introduced in [Fag77b], which defined embedded multivalued dependencies (emvd's); these are mvd's that hold in a projection of a relation. Embedded jd's are defined in the analogous fashion. This is distinct from *projected* jd's [MUV84]—these are template dependencies that correspond to join dependencies, except that some of the variables in the summary row may be distinct variables not occurring elsewhere in the dependency. Several other specialized dependencies were introduced. These include *subset dependencies* [SW82], which generalize mvd's; *mutual dependencies* [Nic78], which say that a relation is a 3-ary join; *generalized mutual dependencies* [MM79]; *transitive dependencies* [Par79], which generalize fd's and mvd's; *extended transitive dependencies* [PPG80], which generalize mutual dependencies and transitive dependencies; and *implied dependencies* [GZ82], which form a specialized class of egd's. In many cases these classes of dependencies were introduced in attempts to provide axiomatizations for the emvd's, jd's, or superclasses of them. Although most of the theoretical work studies dependencies in an abstract setting, [Sci81, Sci83] study families of mvd's and ind's as they arise in practical situations.

The proliferation of dependencies spawned interest in the development of a unifying framework that subsumed essentially all of them. Nicolas [Nic78] is credited with first observing that fd's, mvd's, and others have a natural representation in first-order logic. At roughly the same time, several researchers reached essentially the same generalized class of dependencies that was studied in this chapter. [BV81a] introduced the class of tgd's and egd's, defined using the paradigm of tableaux. Chasing was studied in connection with both full and embedded dependencies in [BV84c]. Reference [Fag82b] introduced the class of typed dependencies, essentially the same family of dependencies but presented in the paradigm of first-order logic. Simultaneously, [YP82] introduced the algebraic dependencies, which present the same class in algebraic terms. A generalization of algebraic dependencies to the untyped case is presented in [Abi83].

Related general classes of dependencies introduced at this time are the *general dependencies* [PJ81], which are equivalent to the full typed tgd's, and *generalized dependency constraints* [GJ82], which are the full dependencies.

Importantly, several kinds of constraints that lie outside the dependencies described in this chapter have been studied in the literature. Research on the use of arbitrary first-order logic sentences as constraints includes [GM78, Nic78, Var82b]. A different extension of dependencies based on partitioning relationships, which are not expressible in first-order logic, is studied in [Cos87]. Another kind of dependency is the *afunctional dependency* of [BP83], which, as the name suggests, focuses on the portions of an instance that violate an fd. The *partition dependencies* [CK86] are not first-order expressible and are powerful; interestingly, finite and unrestricted implication coincide for this class of dependencies and are decidable in PTIME. *Order* [GH83] and *sort-set dependencies* [GH86] address properties of instances defined in terms of orderings on the underlying domain elements. There is provably no finite axiomatization for order dependencies, or for sort-set dependencies and fd's considered together (Exercise 9.8).

Another broad class of constraints not included in the dependencies discussed in this chapter is *dynamic* constraints, which focus on how data change over time [CF84, Su92, Via87, Via88]; see Section 22.6.

As suggested by the development of this chapter, one of the most significant theoretical directions addressed in connection with dependencies has been the issue of decidability of implication. The separation of finite and unrestricted implication, and the undecidability of the implication problem, were shown independently for typed dependencies in [BV81a, CLM81]. Subsequently, these results were independently strengthened to projected jd's in [GL82, Var84, YP82]. Then, after nearly a decade had elapsed, this result was strengthened to include emvd's [Her92].

On the other hand, the equivalence of finite and unrestricted implication for full dependencies was observed in [BV81a]. That deciding implication for full typed dependencies is complete in EXPTIME is due to [CLM81]. See also [BV84c, FUMY83], which present numerous results on full and embedded typed dependencies. The special case of deciding implication of a typed dependency by ind's has been shown to be PSPACE-complete [JK84b].

The issue of inferring view dependencies was first studied in [Klu80], where Theorem 10.2.5 was presented. Reference [KP82] developed Theorem 10.2.6.

The issue of attempting to characterize view images of a satisfaction family as a satisfaction family was first raised in [GZ82], where Exercise 10.11b was shown. Theorem 10.2.7 is due to [Fag82b], although a different proof technique was used there. Reference [Hul84] demonstrates that some projections of satisfaction families defined by fd's cannot be characterized by any finite set of full dependencies (see Exercise 10.11c,d). That investigation is extended in [Hul85], where it is shown that if Σ is a family of fd's over U and $V \subseteq U$, and if $\pi_V(sat(U, \Sigma)) \neq sat(V, \Gamma)$ for any set Γ of fd's, then $\pi_V(sat(U, \Sigma)) \neq sat(V, \Gamma)$ for any finite set Γ of full dependencies.

Another primary thrust in the study of dependencies has been the search for axiomatizations for various classes of dependencies. The axiomatization presented here for full typed dependencies is due to [BV84a], which also provides an axiomatization for the embedded case. The axiomatization for algebraic dependencies is from [YP82]. An axiomatization for template dependencies is given in [SU82] (see Exercise 10.22). Research on axiomatizations for jd's is described in the Bibliographic Notes of Chapter 8.

The direct product construction is from [Fag82b]. Proposition 10.4.6 is due to [Fag82b], and the proof presented here is from [YP82]. A finite set of tgd's with no finite Armstrong relation is exhibited in [FUMY83]. The direct product has also been used in connection with tableau mappings and dependencies [FUMY83] (see Exercise 10.19). The direct product has been studied in mathematical logic; the notion of (upward) faithful presented here (see Exercise 10.28) is equivalent to the notion of "preservation under direct product" found there (see, e.g., [CK73]); and the notion of downward faithful is related to, but distinct from, the notion of "preservation under direct factors."

Reference [MV86] extends the work on direct product by characterizing the expressive power of different families of dependencies in terms of algebraic properties satisfied by families of instances definable using them.

Exercises

Exercise 10.1

(a) Show that for each first-order sentence of the form $(*)$ of Section 10.1, there exists an equivalent finite set of dependencies.

(b) Show that each dependency is equivalent to a finite set of egd's and tgd's.

Exercise 10.2 Consider the tableaux in Example 10.3.2. Give $\sigma \bullet \sigma$. Compare it (as a mapping) to σ. Give $\sigma \bullet \tau$. Compare it (as a mapping) to $\tau \bullet \sigma$.

Exercise 10.3 [DG79] Let φ be a first-order sentence with equality but no function symbols that is in prenex normal form and has quantifier structure $\exists^*\forall^*$. Prove that φ has an unrestricted model iff it has a finite model.

Exercise 10.4 This exercise concerns the dependencies of Fig. 10.2.

(a) Show that $(S, x = z)$ and $(S', x = z)$ are equivalent.

(b) Show that (T, t) and (T', t) are equivalent, but that $(T, t) \subset (T', t)$ as tableau queries.

Exercise 10.5 Let $R[ABC]$ be a relation scheme. We construct a family of egd's over R as follows. For $n \geq 0$, let

$$T_n = \{\langle x_i, y_i, z_{2i}\rangle, \langle x_i, y_{i+1}, z_{2i+1}\rangle \mid i \in [0, n]\}$$

and set $\tau_n = (T_n, z_0 = z_{2n+1})$. Note that $\tau_0 \equiv A \to C$.

(a) Prove that as egd's, $\tau_i \equiv \tau_j$ for all $i, j > 0$.

(b) Prove that $\tau_0 \models \tau_1$, but not vice versa.

Exercise 10.6

(a) [FUMY83] Prove that there are exactly three distinct (up to equivalence) full typed single-head tgd's over a binary relation. *Hint:* See Exercise 10.4.

(b) Prove that there is no set of single-head tgd's that is equivalent to the typed tgd (T_1, T_2) of Fig. 10.2.

(c) Exhibit an infinite chain $\tau_1, \tau_2, \ldots$ of typed tgd's over a binary relation where each is strictly weaker than the previous (i.e., such that $\tau_i \models \tau_{i+1}$ but $\tau_{i+1} \not\models \tau_i$ for each $i \geq 1$).

★ **Exercise 10.7** [FUMY83] Let $U = \{A_1, \ldots, A_n\}$ be a set of attributes.

(a) Consider the full typed single-head tgd (full *template dependency*) $\tau_{strongest} = (\{t_1, \ldots, t_n\}, t)$, where $t_i(A_i) = t(A_i)$ for $i \in [1, n]$, and all other variables used are distinct. Prove that $\tau_{strongest}$ is the "strongest" template dependency for U, in the sense that for each (not necessarily full) template dependency τ over U, $\tau_{strongest} \models \tau$.

(b) Let $\tau_{weakest}$ be the template dependency (S, s), where $s(A_i) = x_i$ for $i \in [1, n]$ and where S includes all tuples s' over U that satisfy (1) $s'(A_i) = x_i$ or y_i for $i \in [1, n]$, and (2) $s'(A_i) \neq x_i$ for at least one $i \in [1, n]$. Prove that $\tau_{weakest}$ is the "weakest" full template dependency U, in the sense that for each nontrivial full template dependency τ over U, $\tau \models \tau_{weakest}$.

(c) For $V \subseteq U$, a template dependency over U is *V-partial* if it can be expressed as a tableau (T, t), where t is over V. For $V \subseteq U$ exhibit a "weakest" V-partial template dependency.

Exercise 10.8 [BV84c] Prove Theorems 10.2.1 and 10.2.2.

Exercise 10.9 Prove that the triviality problem for typed tgd's is NP-complete. *Hint:* Use a reduction from tableau containment (Theorem 6.2.3).

Exercise 10.10

(a) Prove Proposition 10.2.4.

(b) Develop an analogous result for the binary natural join.

Exercise 10.11 Let $R[ABCDE]$ and $S[ABCD]$ be relation schemas, and let $V = ABCD$. Consider $\Sigma = \{A \to E, B \to E, CE \to D\}$.

(a) Describe the set Γ of fd's implied by Σ on $\pi_V(R)$.

(b) [GZ82] Show that $sat(\pi_V(R, \Sigma)) \neq sat(S, \Gamma)$. *Hint:* Consider the instance $J = \{\langle a, b_1,$
$c, d_1\rangle, \langle a, b, c_1, d_2\rangle, \langle a_1, b, c, d_3\rangle\}$ over S.

★(c) [Hul84] Show that there is no finite set Υ of full dependencies over S such that $\pi_V(sat(R, \Sigma)) = sat(S, \Upsilon)$ *Hint:* Say that a satisfaction family $\mathcal{F}$ over R has rank n if $\mathcal{F} = sat(R, \Gamma)$ for some Γ where the tableau in each dependency of Γ has $\leq n$ elements. Suppose that $\pi_V(sat(R, \Sigma))$ has rank n. Exhibit an instance J over V with $n + 1$ elements such that (a) $J \notin \pi_V(sat(R, \Sigma))$, and (b) J satisfies each dependency σ that is implied for $\pi_V(R)$ by Σ, and that has $\leq n$ elements in its tableau. Conclude that $J \in sat(V, \Gamma)$, a contradiction.

★(d) [Hul84] Develop a result for mvd's analogous to part (c).

Exercise 10.12 [KP82] Complete the proof of Theorem 10.2.6 for the case where Σ is a set of full dependencies and γ is a full tgd. Show how to extend that proof (a) to the case where γ is an egd; (b) to include union; and (c) to permit constants in the expression E. *Hint:* For (a), use the technique of Theorem 8.4.12; for (b) use union of tableaux, but permitting multiple output rows; and for (c) recall Exercise 8.27b.

Exercise 10.13 [Fag82b] Prove Theorem 10.2.7.

Exercise 10.14 Exhibit a typed tgd τ and a set Σ of typed dependencies such that $\Sigma \models \tau$, and there are two chasing sequences of τ by Σ, both of which satisfy conditions (1) and (2), in the definition of chasing for embedded dependencies in Section 10.2, where one sequence is finite and the other is infinite.

Exercise 10.15 Consider these dependencies:

$$\begin{array}{c|ccc} & A & B & C \\ \hline & x & y & \\ & x & & z \\ \hline & & y & z \end{array} \qquad \begin{array}{c|ccc} & A & B & C \\ \hline & x & & z \\ & & y & z \\ \hline & x & y & \end{array} \qquad AC \to B$$

$$\tau_1 \qquad\qquad\qquad \tau_2 \qquad\qquad\qquad \tau_3$$

(a) Starting with input $T = \{\langle 1, 2, 3\rangle, \langle 1, 4, 5\rangle\}$, perform four steps of the chase using these dependencies.

(b) Prove that $\{\tau_1, \tau_2, \tau_3\} \not\models_{\text{unr}} A \to B$.

★**Exercise 10.16**

(a) Prove that the chasing sequence of Example 10.2.8 does not terminate; then use this sequence to verify that $\Sigma \not\models_{\text{unr}} \tau_5$.

(b) Show that $\Sigma \not\models_{\text{fin}} \tau_5$.

(c) Exhibit a set Σ' of dependencies and a dependency σ' such that the chasing sequence of σ' with Σ' is infinite, and such that $\Sigma' \not\models_{\text{unr}} \sigma'$ but $\Sigma' \models_{\text{fin}} \sigma'$.

♠**Exercise 10.17** [BV84c] Suppose that $\overline{T}, \overline{\Sigma}$ is a chasing sequence. Prove that *chase*$(\overline{T}, \Sigma)$ satisfies Σ.

Exercise 10.18 [BV84a] (a) Prove Proposition 10.3.1. (b) Complete the proof of Theorem 10.3.3.

Exercise 10.19 [FUMY83] This exercise uses the direct product construction for combining full typed tableau mappings. Let R be a fixed relation schema of arity n. The direct product of free tuples and tableaux is defined as for tuples and instances. Given two full typed tgd's $\tau = (T, t)$ and $\tau' = (T', t')$ over relation schema R, their *direct product* is

$$\tau \otimes \tau' = (T \otimes T', t \otimes t').$$

(a) Let τ, σ be full typed single-head tgd's over R. Prove that $\tau \otimes \sigma$ is equivalent to $\{\tau, \sigma\}$.

(b) Are $\tau \otimes \sigma$ and $\tau \bullet \sigma$ comparable as tableau queries under $\subseteq$, and, if so, how?

(c) Show that the family of typed egd's that have equality atoms referring to the same column of R is closed under finite conjunction.

Exercise 10.20 [FUMY83]

(a) Let τ and τ' be typed tgd's. Prove that $\tau \models_{\text{unr}} \tau'$ iff $\tau \models_{\text{fin}} \tau'$. *Hint:* Show that chasing will terminate in this case.

(b) Prove that there is a pair τ, τ' of typed tgd's for which there is no typed tgd τ'' equivalent to $\{\tau, \tau'\}$. *Hint:* Assume that typed tgd's were closed under conjunction in this way. Use part (a).

★ **Exercise 10.21** [BV84a] State and prove an axiomatization theorem for the family of typed dependencies.

Exercise 10.22 [SU82] Exhibit a set of axioms for template dependencies (i.e., typed single-head tgd's), and prove that it is sound and complete for unrestricted logical implication.

Exercise 10.23 Prove that Algorithm 10.4.2 is correct. (See Exercise 4.18a).

Exercise 10.24

(a) Consider the full typed tgd

$$\tau = (\{\langle x, y'\rangle, \langle x', y'\rangle, \langle x', y\rangle\}, \langle x, y\rangle).$$

Prove that there is no pair E, E' of (nonextended) PJ expressions such that τ is equivalent to $E \subseteq E'$ [i.e., such that $I \models \tau$ iff $E(I) \subseteq E'(I)$].

(b) Let τ be as in Fig. 10.5. Prove that there is no pair E, E' of (nonextended) PJ expressions such that τ is equivalent to $E \subseteq E'$.

Exercise 10.25 In connection with Example 10.4.3,

(a) Prove that τ is equivalent to $E_1 \subseteq_e E_2$.

(b) Prove that $A \to BC$ is equivalent to $\pi_{ABC}(\overline{R}) \bowtie \pi_{AB^1C^1}(\overline{R}) \subseteq_e \pi_{ABCB^1C^1}(\overline{R})$.

(c) Prove that σ is equivalent to $F_1 \subseteq_e F_2$.

★ **Exercise 10.26** Complete the proof of Theorem 10.4.4.

Exercise 10.27 An extended PJ expression E is *shallow* if it has the form $\pi_X(\overline{R})$ or the form $\pi_X(\pi_{Y_1}(\overline{R}) \bowtie \cdots \bowtie \pi_{Y_n}(\overline{R}))$. An algebraic dependency $E \subseteq_e E'$ is *shallow* if E and E' are shallow. Prove that every algebraic dependency is equivalent to a shallow one.

Exercise 10.28 [Fag82b] A dependency σ is *upward faithful* (with respect to direct products) if, for each family of nonempty instances $\{I_j \mid j \in \mathcal{J}\}$,

$$\forall j \in \mathcal{J}, I_j \models \sigma \text{ implies } \otimes\{I_j \mid j \in \mathcal{J}\} \models \sigma.$$

Analogously, σ is *downward faithful* if

$$\otimes\{I_j \mid j \in \mathcal{J}\} \models \sigma \text{ implies } \forall j \in \mathcal{J}, I_j \models \sigma.$$

(a) Show that the constraint

$$\forall x, y, y', z, z'(R(x, y, z) \wedge R(x, y', z') \rightarrow (y = y' \vee z = z'))$$

is downward faithful but not upward faithful.

(b) Show that the constraint

$$\forall x, y, z(R(x, y) \wedge R(y, z) \rightarrow R(x, z))$$

is upward faithful but not downward faithful.

Exercise 10.29 [Fag82b, YP82] Prove Proposition 10.4.6.

Exercise 10.30 [Fag82b] The direct product operator $\otimes$ is extended to instances of database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$ by forming, for each $i \in [1, n]$, a direct product of the relation instances associated with R_i. Let $\mathbf{R} = \{P[A], Q[A]\}$ be a database schema. Show that the empty set of typed dependencies over $\mathbf{R}$ has no Armstrong relation. *Hint:* Find typed dependencies σ_1, σ_2 over $\mathbf{R}$ such that $\emptyset \models (\sigma_1 \vee \sigma_2)$ but $\emptyset \not\models \sigma_1$ and $\emptyset \not\models \sigma_2$.

★ **Exercise 10.31** [YP82] Let $R[ABCD]$ be a relation schema. The pseudo-transitivity rule for multivalued dependencies (Chapter 8) implies, given $A \rightarrow\!\!\!\rightarrow B$ and $B \rightarrow\!\!\!\rightarrow C$, that $A \rightarrow\!\!\!\rightarrow C$. Express this axiom in the paradigm of algebraic dependencies. Prove it using axioms {AD1, . . . , AD7} (without using extended relations).

★ **Exercise 10.32** Infer the three axioms for fd's from rules {A1, . . . , A8}.

Exercise 10.33 [YP82] Prove that {A1, . . . , A8} is sound.