PART

F Finale

In this part, we consider four advanced topics. Two of them (incomplete information and dynamic aspects) have been studied for a while, but for some reason (perhaps their difficulty) they have never reached the maturity of more established areas such as dependency theory. Interest in the other two topics (complex values and object databases) is more recent, and our understanding of them is rudimentary. In all cases, no clear consensus has yet emerged. Our choice of material, as well as our presentation, are therefore unavoidably more subjective than in other parts of this book. However, the importance of these issues for practical systems, as well as the interesting theoretical issues they raise, led us to incorporate a discussion of them in this book.

In Chapter 19, we address the issue of incomplete information. In many database applications, the knowledge of the real world is incomplete. It is crucial to be able to handle such incompleteness and, in particular, to be able to ask queries and perform updates. Chapter 19 surveys various models of incomplete databases, research directions, and some results.

In Chapter 20, we present an extension of relations called complex values. These are obtained from atomic elements using tuple and set constructors. The richer structure allows us to overcome some limitations of the relational model in describing complex data. We generalize results obtained for the relational model; in particular, we present a calculus and an equivalent algebra.

Chapter 21 looks at another way to enrich the relational model by introducing a number of features borrowed and adapted from object-oriented programming, such as objects, classes, and inheritance. In particular, objects consist of a structural part (a data repository) and a behavioral part (pieces of code). Thus the extended framework encompasses behavior, a notion conspicuously absent from relational databases.

Chapter 22 deals with dynamic aspects. This is one of the less settled areas in databases, and it raises interesting and difficult questions. We skim through a variety of issues: languages and semantics for updates; updating views; updating incomplete information; and active and temporal databases.

A comprehensive vision of the four areas discussed in Part F is lacking. The reader should therefore keep in mind that some of the material presented is in flux, and its importance pertains more to the general flavor than the specific results.

19 Incomplete Information

Somebody:	*What are we doing next?*
Alice:	*Who are* we*? Who are* you*?*
Somebody:	We *are you and the authors of the book, and* I *am one of them. This is an instance of incomplete information.*
Somebody:	*It's not much, but we can still tell that* surely *one of us is Alice and that there are* possibly *up to three "Somebodies" speaking.*

In the previous parts, we have assumed that a database always records information that is completely known. Thus a database has consisted of a completely determined finite instance. In reality, we often must deal with incomplete information. This can be of many kinds. There can be missing information, as in "John bought a car but I don't know which one." In the case of John's car, the information exists but we do not have it. In other cases, some attributes may be relevant only to some tuples and irrelevant to others. Alice is single, so the spouse field is irrelevant in her case. Furthermore, some information may be imprecise: "Heather lives in a large and cheap apartment," where the values of *large* and *cheap* are fuzzy. Partial information may also arise when we cannot completely rely on the data because of possible inconsistencies (e.g., resulting from merging data from different sources).

As soon as we leave the realm of complete databases, most issues become much more intricate. To deal with the most general case, we need something resembling a theory of knowledge. In particular, this quickly leads to logics with modalities: Is it *certain* that John lives in Paris? Is it *possible* that he may? What is the *probability* that he does? Does John *know* that Alice is a good student? Does he *believe* so? etc.

The study of knowledge is a fascinating topic that is outside the scope of this book. Clearly, there is a trade-off between the expressivity of the model for incomplete information used and the difficulty of answering queries. From the database perspective, we are primarily concerned with identifying this trade-off and understanding the limits of what is feasible in this context. The purpose of this chapter is to make a brief foray into this topic. We limit ourselves mostly to models and results of a clear database nature. We consider simple forms of incompleteness represented by null values. The main problem we examine is how to answer queries on such databases. In relation to this, we argue that for a representation system of incomplete information to be adequate in the context of a query language, it must also be capable of representing *answers* to queries. This leads to a desirable closure property of representations of incomplete information with respect to query languages. We observe the increase of complexity resulting from the use of nulls.

We also consider briefly two approaches closer to knowledge bases. The first is based on the introduction of disjunctions in deductive databases, which also leads to a form of incompleteness. The second is concerned with the use of modalities. We briefly mention the language KL, which permits us to talk about knowledge of the world.

19.1 Warm-Up

As we have seen, there are many possible kinds of incomplete information. In this section, we will focus on databases that partially specify the state of the world. Instead of completely identifying one state of the world, the database contents are compatible with many possible worlds. In this spirit, we define an *incomplete database* simply as a set of possible worlds (i.e., a set of instances). What is actually stored is a *representation* of an incomplete database. Choosing appropriate representations is a central issue.

We provide a mechanism for representing incomplete information using *null values*. The basic idea is to allow occurrences of variables in the tuples of the database. The different possible values of the variables yield the possible worlds.

The simplest model that we consider is the Codd table (introduced by Codd), or *table* for short. A table is a relation with constants and variables, in which no variable occurs twice. More precisely, let U be a finite set of attributes. A *table* T over U is a finite set of free tuples over U such that each variable occurs at most once. An example of a table is given in Fig. 19.1. The figure also illustrates an alternative representation (using @) that is more visual but that we do not adopt here because it is more difficult to generalize.

The preceding definition easily extends to database schemas. A database table **T** over a database schema **R** is a mapping over **R** such that for each R in **R**, $\mathbf{T}(R)$ is a table over $sort(R)$. For this generalization, we assume that the sets of variables appearing in each table are *pairwise disjoint*. Relationships between the variables can be stated through

R	A	B	C
	0	1	x
	y	z	1
	2	0	v

Table T

R	A	B	C
	0	1	@
	@	@	1
	2	0	@

Alternative representation of T

R	A	B	C
	0	1	2
	2	0	1
	2	0	0

I_1

R	A	B	C
	0	1	2
	3	0	1
	2	0	5

I_2

R	A	B	C
	0	1	2
	2	0	1
	2	0	0

I_3

R	A	B	C
	0	1	1
	2	0	1

I_4

Figure 19.1: A table and examples of corresponding instances

global conditions (which we will introduce in the next section). In this section, we will focus on single tables, which illustrate well the main issues.

To specify the semantics of a table, we use the notion of valuation (see Chapter 4). The incomplete database represented by a table is defined as follows:

$$rep(T) = \{\nu(T) \mid \nu \text{ a valuation of the variables in } T\}.$$

Consider the table T in Fig. 19.1. Then $I_1, \ldots, I_4$ all belong to $rep(T)$ (i.e., are possible worlds).

The preceding definition assumes the Closed World Assumption (CWA) (see Chapter 2). This is because each tuple in an instance of $ref(T)$ must be justified by the presence of a particular free tuple in T. An alternative approach is to use the Open World Assumption (OWA). In that case, the incomplete database of T would include all instances that contain an instance of $rep(T)$. In general, the choice of CWA versus OWA does not substantially affect the results obtained for incomplete databases.

We now have a simple way of representing incomplete information. What next? Naturally, we wish to be able to query the incomplete database. Exactly what this means is not clear at this point. We next look at this issue and argue that the simple model of tables has serious shortcomings with respect to queries. This will naturally lead to an extension of tables that models more complicated situations.

Let us consider what querying an incomplete database might mean. Consider a table T and a query q. The table T represents a set of possible worlds $rep(T)$. For each $I \in rep(T)$, q would produce an answer $q(I)$. Therefore the set of possible answers of q is $q(rep(T))$. This is, again, an incomplete database. The answer to q should be a representation of this incomplete database.

More generally, consider some particular representation system (e.g., tables). Such a system involves a language for describing representations and a mapping *rep* that associates a set of instances with each representation. Suppose that we are interested in a particular query language $\mathcal{L}$ (e.g., relational algebra). We would always like to be capable of representing the result of a query in the same system. More precisely, for each representation T and query q, there should exist a computable representation $\overline{q}(T)$ such that

$$rep(\overline{q}(T)) = q(rep(T)).$$

In other words, $\overline{q}(T)$ represents the possible answers of q [i.e., $\{q(I) \mid I \in rep(T)\}$].

If some representation system τ has the property described for a query language $\mathcal{L}$, we will say that τ is a *strong representation system* for $\mathcal{L}$. Clearly, we are particularly interested in strong representation systems for relational algebra and we shall develop such a system later.

Let us now return to tables. Unfortunately, we quickly run into trouble when asking queries against them, as the following example shows.

Example 19.1.1 Consider T of Fig. 19.1 and the algebraic query $\sigma_{A=3}(T)$. There is no table representing the possible answers to this query. A possible answer (e.g., for I_1) is the empty relation, whereas there are nonempty possible answers (e.g., for I_2). Suppose that there exists a table T' representing the set of possible answers. Either T' is empty and $\sigma_{A=3}(I_2)$ is not in $rep(T')$; or T' is nonempty and the empty relation is not in $rep(T')$. This is a contradiction, so no such T' can exist.

The problem lies in the weakness of the representation system of tables; we will consider richer representation systems that lead to a complete representation system for all of relational algebra. An alternative approach is to be less demanding; we consider this next and present the notion of *weak* representation systems.

19.2 Weak Representation Systems

To relax our expectations, we will no longer require that the answer to a query be a representation of the set of all possible answers. Instead we will ask which are the tuples that are surely in the answer (i.e., that belong to *all* possible answers). (Similarly, we may ask for the tuples that are possibly in the answer (i.e., that belong to *some* possible answer). We make this more precise next.

For a table T and a query q, the set of sure facts, $sure(q, T)$, is defined as

$$sure(q, T) = \cap\{q(I) \mid I \in rep(T)\}.$$

Clearly, a tuple is in $sure(q, T)$ iff it is in the answer for every possible world. Observe that the sure tuples in a table T [i.e., the tuples in every possible world in $rep(T)$] can be computed easily by dropping all free tuples with variables. One could similarly define the set $poss(q, T)$ of possible facts.

One might be tempted to require of a weak system just the ability to represent the set of tuples surely in the answer. However, the definition requires some care due to the following subtlety. Suppose T is the table in Fig. 19.1 and q the query $\sigma_{A=2}(R)$, for which $sure(q, T) = \emptyset$. Consider now the query $q' = \pi_{AB}(R)$ and the query $q \circ q'$. Clearly, $q'(sure(q, T)) = \emptyset$; however, $sure(q'(q(rep(T))) = \{\langle 2, 0\rangle\}$. So $q \circ q'$ cannot be computed by first computing the tuples surely returned by q and then applying q'. This is rather unpleasant because generally it is desirable that the semantics of queries be compositional (i.e., the result of $q \circ q'$ should be obtained by applying q' to the result of q). The conclusion is that the answer to q should provide more information than just $sure(q, T)$; the incomplete database it specifies should be equivalent to $q(rep(T))$ with respect to its ability to compute the sure tuples of any query in the language applied to it. This notion of equivalence of two incomplete databases is formalized as follows.

If $\mathcal{L}$ is a query language, we will say that two incomplete databases $\mathcal{I}, \mathcal{J}$ are $\mathcal{L}$ equivalent, denoted $\mathcal{I} \equiv_{\mathcal{L}} \mathcal{J}$, if for each q in $\mathcal{L}$ we have

$$\cap\{q(I) \mid I \in \mathcal{I}\} = \cap\{q(I) \mid I \in \mathcal{J}\}.$$

In other words, the two incomplete databases are undistinguishable if all we can ask for is the set of sure tuples in answers to queries in $\mathcal{L}$.

We can now define weak representation systems. Suppose $\mathcal{L}$ is a query language. A representation system is *weak* for $\mathcal{L}$ if for each representation T of an incomplete database, and each q in $\mathcal{L}$, there exists a representation denoted $\overline{q}(T)$ such that

$$rep(\overline{q}(T)) \equiv_{\mathcal{L}} q(rep(T)).$$

With the preceding definition, $\overline{q}(T)$ does not provide precisely $sure(q, T)$ for tables T. However, note that $sure(q, T)$ can be obtained at the end simply by eliminating from the answer all rows with occurrences of variables.

The next result indicates the power of tables as a weak representation system.

Theorem 19.2.1 Tables form a weak representation system for selection-projection (SP) [i.e., relational algebra limited to selection (involving only equalities and inequalities) and projection]. If union or join are added, tables no longer form a weak representation system.

Crux It is easy to see that tables form a weak representation system for SP queries. Selections operate conservatively on tables. For example,

$$\overline{\sigma_{cond}}(T) = \{t \mid t \in T \text{ and } cond(\nu(t)) \text{ holds}$$
$$\text{for all valuations } \nu \text{ of the variables in } t\}.$$

Projections operate like classical projections. For example, if T is again the table in Fig. 19.1, then

$$\overline{\sigma_{A=2}}(T) = \{\langle 2, 0, v\rangle\}$$

and

$$\overline{(\pi_{AB}(R) \circ \sigma_{A=2}(R))}(T) = \{\langle 2, 0\rangle\}.$$

Let us show that tables are no longer a weak representation system if join or union are added to SP. Consider join first. So the query language is now SPJ. Let T be the table

R	A	B	C
	a	x	c
	a'	x'	c'

where x, x' are variables and a, a', c, c' are constants.

Let $q = \pi_{AC}(R) \bowtie \pi_B(R)$. Suppose there is table W such that

$$rep(W) \equiv_{SPJ} q(rep(T)),$$

and consider the query $q' = \pi_{AC}(\pi_{AB}(R) \bowtie \pi_{BC}(R))$. Clearly, $sure(q \circ q', T)$ is

A	C
a	c
a'	c
a	c'
a'	c'

Therefore $sure(q', W)$ must be the same. Because $\langle a', c\rangle \in sure(q', W)$, for each valuation ν of variables in W there must exist tuples $u, v \in W$ such that $u(A) = a'$, $v(C) = c$, $\nu(u)(B) = \nu(v)(B)$. Let ν be a valuation such that $\nu(z) \neq \nu(y)$ for all variables $z, y, z \neq y$. If $u = v$, then $u(A) = a'$ and $u(C) = c$ so $\langle a', c\rangle \in sure(\pi_{AC}(R), W)$. This cannot be because, clearly, $\langle a', c\rangle \notin sure(\pi_{AC}(R), q(rep(T)))$. So, $u \neq v$. Because $\nu(u)(B) = \nu(v)(B)$ and W has no repeated variables, it follows that $u(B)$ and $v(B)$ equal some constant k. But then $\langle a', k\rangle \in sure(\pi_{AB}(R), W)$, which again cannot be because one can easily verify that $sure(\pi_{AB}(R), q(rep(T))) = \emptyset$.

The proof that tables do not provide a weak representation system for SPU follows similar lines. Just consider the table T

R	A	B
	x	b

and the query q outputting two relations: $\sigma_{A=a}(R)$ and $\sigma_{A\neq a}(R)$. It is easily seen that there is no pair of tables W_1, W_2 weakly representing $q(rep(T))$ with respect to SPU. To see this, consider the query $q' = \pi_B(W_1 \cup W_2)$. The details are left to the reader (Exercise 19.7). ■

Naive Tables

The previous result shows the limitations of tables, even as weak representation systems. As seen from the proof of Theorem 19.2.1, one problem is the lack of repeated variables. We next consider a first extension of tables that allows repetitions of variables. It will turn out that this will provide a weak representation system for a large subset of relational algebra.

A *naive* table is like a table except that variables may repeat. A naive table is shown in Fig. 19.2. Naive tables behave beautifully with respect to positive existential queries (i.e., conjunctive queries augmented with union). Recall that, in terms of the algebra, this is SPJU.

Theorem 19.2.2 Naive tables form a weak representation system for positive relational algebra.

Crux Given a naive table T and a positive query q, the evaluation of $\overline{q}(T)$ is extremely simple. The variables are treated as distinct new constants. The standard evaluation of q is then performed on the table. Note that incomplete information yields no extra cost in this case. We leave it to the reader to verify that this works. ■

R	A	B	C
	0	1	x
	x	z	1
	2	0	v

Figure 19.2: A naive table

Naive tables yield a nice representation system for a rather large language. But the representation system is weak and the language does not cover all of relational algebra. We introduce in the next section a representation that is a strong system for relational algebra.

19.3 Conditional Tables

We have seen that Codd tables and naive tables are not rich enough to provide a strong representation system for relational algebra. To see what is missing, recall that when we attempt to represent the result of a selection on a table, we run into the problem that the presence or absence of certain tuples in a possible answer is conditioned by certain properties of the valuation. To capture this, we extend the representation with conditions on variables, which yields conditional tables. We will show that such tables form a strong representation system for relational algebra.

A *condition* is a *conjunct* of *equality atoms* of the form $x = y$, $x = c$ and of *inequality atoms* of the form $x \neq y$, $x \neq c$, where x and y are variables and c is a constant. Note that we only use conjuncts of atoms and that the Boolean *true* and *false* can be respectively encoded as atoms $x = x$ and $x \neq x$.

If formula Φ is a condition, we say that a valuation v *satisfies* Φ if its assignment of constants to variables makes the formula true.

Conditions may be associated with table T in two ways: (1) A *global* condition Φ_T is associated with the entire table T; (2) a *local* condition φ_t is associated with one tuple t of table T. A *conditional table* (*c-table* for short) is a triple (T, Φ_T, φ), where

- T is a table,
- Φ_T is a global condition,
- φ is a mapping over T that associates a local condition φ_t with each tuple t of T.

A c-table is shown in Fig. 19.3. If we omit listing a condition, then it is by default the atom *true*. Note also that conditions Φ_T and φ_t for t in T may contain variables not appearing respectively in T or t.

For our purposes, the global conditions in c-tables could be distributed at the tuple level as local conditions. However, they are convenient as shorthand and when dependencies are considered.

For brevity, we usually refer to a c-table (T, Φ_T, φ) simply as T. A given c-table T represents a set of instances as follows (again adopting the CWA):

T'	A	B	
	$x \neq 2, y \neq 2$		
	0	1	$z = z$
	1	x	$y = 0$
	y	x	$x \neq y$

J_1	A	B
	0	1
	0	0

J_2	A	B
	0	1
	1	0

J_3	A	B
	0	1

J_4	A	B
	0	1
	0	3

Figure 19.3: A c-table and some possible instances

$rep(T) = \{I \mid$ there is a valuation ν satisfying Φ_T such that relation I consists exactly of those facts $\nu(t)$ for which ν satisfies $\varphi_t\}$.

Consider the table T' in Fig. 19.3. Then J_1, J_2, J_3, J_4 are obtained by valuating x, y, z to (0,0,0), (0,1,0), (1,0,0), and (3,0,0), respectively.

The next example illustrates the considerable power of the local conditions of c-tables, including the ability to capture disjunctive information.

Example 19.3.1 Suppose we know that Sally is taking math or computer science (CS) (but not both) and another course; Alice takes biology if Sally takes math, and math or physics (but not both) if Sally takes physics. This can be represented by the following c-table:

Student	*Course*	
	$(x \neq math) \wedge (x \neq CS)$	
Sally	*math*	$(z = 0)$
Sally	*CS*	$(z \neq 0)$
Sally	x	
Alice	*biology*	$(z = 0)$
Alice	*math*	$(x = physics) \wedge (t = 0)$
Alice	*physics*	$(x = physics) \wedge (t \neq 0)$

Observe that there may be several c-table representations for the same incomplete database. Two representations T, T' are said to be equivalent, denoted $T \equiv T'$, if $rep(T) = rep(T')$. Testing for equivalence of c-tables is not a trivial task. Just testing membership of an instance in $rep(T)$, apparently a simpler task, will be shown to be NP-complete. To test equivalence of two c-tables T and T', one must show that for each valuation ν of the variables in T there exists a valuation ν' for T' such that $\nu(T) = \nu'(T')$, and conversely. Fortunately, it can be shown that one need only consider valuations to a set C of constants containing all constants in T or T' and whose size is at most the number of variables in the two tables (Exercise 19.11). This shows that equivalence of c-tables is decidable.

In particular, finding a minimal representation can be hard. This may affect the computation of the result of a query in various ways: The complexity of computing the answer may depend on the representation of the input; and one may require the result to be somewhat compact (e.g., not to contain tuples with unsatisfiable local conditions).

It turns out that c-tables form a strong representation system for relational algebra.

Theorem 19.3.2 For each c-table T over U and relational algebra query q over U, one can construct a c-table $\overline{q}(T)$ such that $rep(\overline{q}(T)) = q(rep(T))$.

Crux The proof is straightforward and is left as an exercise (Exercise 19.13). The example in Fig. 19.4 should clarify the construction.[1] For projection, it suffices to project the columns of the table. Selection is performed by adding new conjuncts to the local conditions. Union is represented by the union of the two tables (after making sure that they use distinct sets of variables) and choosing the appropriate local conditions. Join and intersection involve considering all pairs of tuples from the two tables. For difference, we consider a tuple in the first table and add a huge conjunct stating that it does not match any tuple from the second table (disjunctions may be used as shorthand; they can be simulated using new variables, as illustrated in Example 19.3.1). ■

To conclude this section, we consider (1) languages with recursion, and (2) dependencies. In both cases (and for related reasons) the aforementioned representation system behaves well. The presentation is by examples, but the formal results can be derived easily.

Languages with Recursion

Consider an incomplete database and a query involving fixpoint. For instance, consider the table in Fig. 19.5. The representation $\overline{tc}(T)$ of the answer to the transitive closure query tc is also given in the same figure. One can easily verify that

$$rep(\overline{tc}(T)) = tc(rep(T)).$$

This can be generalized to arbitrary languages with iteration. For example, consider a c-table T and a relational algebra query q that we want to iterate until a fixpoint is reached.

[1] The representations in the tables can be simplified; they are given in rough form to illustrate the proof technique.

T_1	B	C
	x	c

T_2	B	C	
	y	c	$(y = b)$
	z	w	

$\overline{\pi_B(T_2)}$	B	
	y	$(y = b)$
	z	

T_3	A	B
	a	y

$\overline{\sigma_{B=b}(T_1 \bowtie T_3)}$	A	B	C	
	a	y	c	$(y = x) \wedge (y = b)$

$\overline{T_1 \cup T_2}$	B	C	
	x	c	
	y	c	$(y = b)$
	z	w	

$\overline{T_1 \bowtie T_3}$	A	B	C	
	a	y	c	$(y = x)$

$\overline{T_1 - T_2}$	B	C	
	x	c	$(y \neq b) \wedge (x \neq z)$
	x	c	$(y \neq b) \wedge (w \neq c)$
	x	c	$(y = b) \wedge (x \neq b) \wedge (x \neq z)$
	x	c	$(y = b) \wedge (x \neq b) \wedge (w \neq c)$

Figure 19.4: Computing with c-tables

Then we can construct the sequence of c-tables:

$$\overline{q}(T), \overline{q}^2(T), \ldots, \overline{q}^i(T), \ldots.$$

Suppose now that q is a positive query. We are guaranteed to reach a fixpoint on every single complete instance. However, this does not a priori imply that the sequence of representations $\{\overline{q}^i(T)\}_{i>0}$ converges. Nonetheless, we can show that this is in fact the case. For some i,

$$rep(\overline{q}^i(T)) = rep(\overline{q}^{i+1}(T)).$$

T	A	B
	a	b
	x	c
	c	d

$\overline{tc}(T)$	A	B	
	a	b	
	x	c	
	c	d	
	a	c	$x = b$
	x	d	
	c	c	$x = d$
	a	d	$x = b$

Figure 19.5: Transitive closure of a table

(See Exercise 19.17.) It can also be shown easily that for such i, every $I \in rep(\overline{q}^i(T))$ is a fixpoint of q. The proof is by contradiction: Suppose there is $I \in rep(\overline{q}^i(T))$ such that $q(I) \neq I$, and consider one such I with a minimum number of tuples. Because $rep(\overline{q}^i(T)) = rep(\overline{q}^{i+1}(T))$, $I = q(J)$ for some $J \in rep(\overline{q}^i(T))$. Because q is positive, $J \subseteq I$; so because $q(I) \neq I$, $J \subset I$. This contradicts the minimality of I. So $\overline{q}^i(T))$ is indeed the desired answer.

Thus to find the table representing the result, it suffices to compute the sequence $\{\overline{q}^i(T)\}_{i>0}$ and stop when two consecutive tables are equivalent.

Dependencies

In Part B, we studied dependencies in the context of complete databases. We now reconsider dependencies in the context of incomplete information. Suppose we are given an incomplete database (i.e., a set $\mathcal{I}$ of complete databases) and are told, in addition, that some set Σ of dependencies is satisfied. The question arises: How should we interpret the combined information provided by $\mathcal{I}$ and by Σ?

The answer depends on our view of the information provided by an incomplete database. Dependencies should add to the information we have. But how do we compare incomplete databases with respect to information content? One common-sense approach, in line with our discussion so far, is that more information means reducing further the set of possible worlds. Thus an incomplete database $\mathcal{I}$ (i.e., a set of possible worlds) is more informative than $\mathcal{J}$ iff $\mathcal{I} \subset \mathcal{J}$. In this spirit, the natural use of dependencies would be to eliminate from $\mathcal{I}$ those possible worlds not satisfying Σ. This makes sense for egd's (and in particular fd's).

A different approach may be more natural in the context of tgd's. This approach stems from a relaxation of the CWA that is related to the OWA. Let $\mathcal{I}$ be an incomplete database, and let Σ be a set of dependencies. Recall that tgd's imply the presence of certain tuples based on the presence of other tuples. Suppose that for some $I \in \mathcal{I}$, a tuple t implied by a tgd in Σ is not present in I. Under the relaxation of the CWA, we conclude that t should be viewed as present in I, even though it is not represented explicitly. More generally, the chase (see Chapter 8), suitably generalized to operate on instances rather than tableaux,

I_1

A	B	C
a	b	c
a	b'	c'

I_2

A	B	C
e	f	g
e	f'	g'
e	f	g'
e	f'	g

I_3

A	B	C
a	b	c
g	b	h

J_1

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

J_2

A	B	C
e	f	g
e	f'	g'
e	f	g
e	f'	g

Figure 19.6: Incomplete databases and dependencies

can be used to complete the instance by adding all missing tuples implied by the tgd's in Σ. (See Exercise 19.18.)

In fact, the chase can be used for both egd's and tgd's. In contrast to tgd's, the effect of chasing with egd's (and, in particular, fd's) may be to eliminate possible worlds that violate them. Note that tuples added by tgd's may lead to violations of egd's. This suggests that an incomplete database $\mathcal{I}$ with a set Σ of dependencies represents

$$\{chase(I, \Sigma) \mid I \in \mathcal{I} \text{ and the chase of } I \text{ by } \Sigma \text{ succeeds}\}.$$

For example, consider Fig. 19.6, which shows the incomplete database $\mathcal{I} = \{I_1, I_2, I_3\}$. Under this perspective, the incorporation of the dependencies $\Sigma = \{A \twoheadrightarrow B, B \rightarrow A\}$ in this incomplete database leads to $\mathcal{J} = \{J_1, J_2\}$.

Suppose now that the incomplete database $\mathcal{I}$ is represented as a c-table T. Can the effect of a set Σ of full dependencies on T be represented by another c-table T'? The answer is yes, and T' is obtained by extending the chase to c-tables in the straightforward way. For example, a table T_1 and its completion T_2 by $\Sigma = \{A \twoheadrightarrow B, C \rightarrow D\}$ are given in Fig. 19.7. The reader might want to check that

$$chase_\Sigma(rep(T_1)) = rep(T_2).$$

T_1	A	B	C	D
	a	b	c	d
	x	e	y	g
	a	b	c	z

T_2	A	B	C	D	
	a	b	c	d	
	x	e	y	g	
	a	b	y	g	$(x = a)$
	a	e	c	d	$(x = a)$

Figure 19.7: c-tables and dependencies

19.4 The Complexity of Nulls

Conditional tables may appear to be a minor variation from the original model of complete relational databases. However, we see next that the use of nulls easily leads to intractability. This painfully highlights the trade-off between modeling power and resources.

We consider some basic computational questions about incomplete information databases. Perhaps the simplest question is the *possibility* problem: "Given a set of possible worlds (specified, for instance, by a c-table) and a set of tuples, is there a possible world where these tuples are all true?" A second question is the *certainty* problem: "Given a set of possible worlds and a set of tuples, are these tuples all true in every possible world?" Natural variations of these problems involve queries: Is a given set of tuples possibly (or certainly) in the answer to query q?

Consider a (c-) table T, a query q, a relation I, and a tuple t. Some typical questions include the following:

(Membership) Is I a possible world for T [i.e., $I \in \mathit{rep}(T)$]?

(Possibility) Is t possible [i.e., $\exists I \in \mathit{rep}(T)(t \in I)$]?

(Certainty) Is t certain [i.e., $\forall I \in \mathit{rep}(T)(t \in I)$]?

(q-Membership) Is I a possible answer for q and T [i.e., $I \in q(\mathit{rep}(T))$]?

(q-Possibility) Is t possibly in the answer [i.e., $\exists I \in \mathit{rep}(T)(t \in q(I))$]?

(q-Certainty) Is t certainly in the answer [i.e., $\forall I \in \mathit{rep}(T)(t \in q(I))$]?

Finally we may consider the following generalizations of the q-membership problem:

(q-Containment) Is T contained in $q(T')$ [i.e., $\mathit{rep}(T) \subseteq q(\mathit{rep}(T'))$]?

(q, q'-Containment) Is $q(T)$ contained in $q'(T)$ [i.e., $\mathit{rep}(q(T)) \subseteq \mathit{rep}(q'(T))$]?

The crucial difference between complete and incomplete information is the large number of possible valuations for the latter case. Because of the finite number of variables in a set of c-tables, only a finite number of valuations are nonisomorphic (see Exercise 19.10). However, the number of such valuations may grow exponentially in the input size. By simple reasoning about all valuations and by guessing particular valuations, we have some easy upper bounds. For a query q that can be evaluated in polynomial time on complete databases, deciding whether $I \in q(\mathit{rep}(T))$, or whether I is a set of possible answers, can be answered in NP; checking whether $q(\mathit{rep}(T)) = \{I\}$, or if I is a set of certain tuples, is in CO-NP.

To illustrate such complexity results, we demonstrate one lower bound concerning the q-membership problem for (Codd) tables.

PROPOSITION 19.4.1 There exists a positive existential query q such that checking, given a table T and a complete instance I, whether $I \in q(\mathit{rep}(T))$ is NP-complete.

Proof The proof is by reduction of graph 3-colorability. For simplicity, we use a query mapping a two-relation database into another two-relation database. (An easy modification of the proof shows that the result also holds for databases with one relation. In particular, increase the arity of the largest relation, and use constants in the extra column to encode several relations into this one.)

We will use (1) an input schema **R** with two relations R, S of arity 5 and 2, respectively; (2) an output schema $\mathbf{R}'$ with two relations R', S' of arity 3 and 1, respectively; and (3) a positive existential query q from **R** to $\mathbf{R}'$. The query q [returning, on each input **I** over **R**, two relations $q_1(\mathbf{I})$ and $q_2(\mathbf{I})$ over R' and S'] is defined as follows:

$$\begin{aligned} q_1 = \{\langle x, z, z'\rangle \mid \exists y([\exists vw(R(x, y, v, w, z) \vee R(v, w, x, y, z))] \\ \wedge [\exists vw(R(x, y, v, w, z') \vee R(v, w, x, y, z'))])\} \\ q_2 = \{z \mid \exists xyvw(R(x, y, v, w, z) \wedge S(y, w))\}. \end{aligned}$$

For each input $G = (V, E)$ to the graph 3-colorability problem, we construct a table **T** over the input schema **R** and an instance $\mathbf{I}'$ over the output schema $\mathbf{R}'$, such that G is 3-colorable iff $\mathbf{I}' \in q(rep(\mathbf{T}))$.

Without loss of generality, assume that G has no self-loops and that E is a binary relation, where we list each edge once with an arbitrary orientation.

Let $V = \{a_i \mid i \in [1..n]\}$ and $E = \{(b_j, c_j) \mid j \in [1..m]\}$. Let $\{x_j \mid j \in [1..m]\}$ and $\{y_j \mid j \in [1..m]\}$ be two disjoint sets of distinct variables. Then **T** and $\mathbf{I}'$ are constructed as follows:

(a) $\mathbf{T}(R) = \{t_j \mid j \in [1..m]\}$, where t_j is the tuple $\langle b_j, x_j, c_j, y_j, j\rangle$;

(b) $\mathbf{T}(S) = \{\langle i, j\rangle \mid i, j \in \{1, 2, 3\}, i \neq j\}$;

(c) $\mathbf{I}'(R') = \{\langle a, j, k\rangle \mid a \in \{b_j, c_j\} \cap \{b_k, c_k\}$, where each (b, c) pair is an edge in E }; and

(d) $\mathbf{I}'(S') = \{j \mid j \in [1..m]\}$.

Intuitively, for each tuple in $\mathbf{I}(R)$, the second column contains the color of the vertex in the first column, and the fourth column contains the color of the vertex in the third column. The edges are numbered in the fifth column. The role of query q_2 is to check whether this provides an assignment of the three colors $\{1, 2, 3\}$ to vertexes such that the colors of the endpoints of each edge are distinct. Indeed, q_2 returns the edges z for which the colors y, w of its endpoints are among $\{1, 2, 3\}$. So if $q(\mathbf{I})(S') = \mathbf{I}'(S')$, then all edges have color assignments among $\{1, 2, 3\}$ to their endpoints. Next query q_1 checks whether a vertex is assigned the same color consistently in all edges where it occurs. It returns the $\langle x, z, z'\rangle$, where x is a vertex, z and z' are edges, x occurs as an endpoint, and x has the same color assignment y in both z and z'. So if $q_1(\mathbf{I})(R') = \mathbf{I}'(R')$, it follows that the color assignment is consistent everywhere for all vertexes.

For example, consider the graph G given in Fig. 19.8; the corresponding $\mathbf{I}'$ and **T** are exhibited in Fig. 19.9. Suppose that f is a 3-coloring of G. Consider the valuation σ defined by $\sigma(x_j) = f(b_j)$ and $\sigma(y_j) = f(c_j)$ for all j. It is easily seen that $\mathbf{I}' = q(\sigma(T))$. Moreover, it is straightforward to show that G is 3-colorable iff $\mathbf{I}'$ is in $q(rep(\mathbf{T}))$. ■

1	2
2	3
3	4
4	1
3	1

Figure 19.8: Graph G

T(R)

1	x_1	2	y_1	1
2	x_2	3	y_2	2
3	x_3	4	y_3	3
4	x_4	1	y_4	4
3	x_5	1	y_5	5

T(S)

1	2
1	3
2	1
2	3
3	1
3	2

I$'(R')$

1	1	1
1	1	4
1	1	5
1	4	1
1	4	4
1	4	5
2	1	1
2	1	2
2	2	1
2	2	2
	⋮	
4	3	3
4	3	4
4	4	3
4	4	4

I$'(S')$

1
2
3
4
5

Figure 19.9: Encoding for the reduction of 3-colorability

19.5 Other Approaches

Incomplete information often arises naturally, even when the focus is on complete databases. For example, the information in a view is by nature incomplete, which in particular leads to problems when trying to update the view (as discussed in Chapter 22); and we already considered relations with nulls in the weak universal relations of Chapter 11.

In this section, we briefly present some other aspects of incomplete information. We consider some alternative kinds of null values; we look at disjunctive deductive databases; we mention a language that allows us to address directly in queries the issue of incompleteness; and we briefly mention several situations in which incomplete information arises naturally, even when the database itself is complete. An additional approach to representing incomplete information, which stems from using explicit logical theories, will be presented in connection with the view update problem in Chapter 22.

Other Nulls in Brief

So far we have focused on a specific kind of null value denoting values that are unknown. Other forms of nulls may be considered. We may consider, for instance, *nonexisting* nulls. For example, in the tuple representing a CEO, the field DirectManager has no meaning and therefore contains a nonexisting null. Nonexisting nulls are at the core of the weak universal model that we considered in Chapter 11.

It may also be the case that we do not know for a specific field if a value exists. For example, if the database ignores the marital status of a particular person, the spouse field is either unknown or nonexisting. It is possible to develop a formal treatment of such *no-information* nulls. An incomplete database consists of a set of sets of tuples, where each set of tuples is closed under projection. This closure under projection indicates that if a tuple is known to be true, the projections of this tuple (although less informative) are also known to be true. (The reader may want to try, as a nontrivial exercise, to define tables formally with such nulls and obtain a closure theorem analogous to Theorem 19.3.2.)

For each new form of null values, the game is to obtain some form of representation with clear semantics and try to obtain a closure theorem for some reasonable language (like we did for unknown nulls). In particular, we should focus on the most important algebraic operations for accessing data: projection and join. It is also possible to establish a lattice structure with the different kinds of nulls so that they can be used meaningfully in combination.

Disjunctive Deductive Databases

Disjunctive logic programming is an extension of standard logic programming with rules of the form

$$A_1 \vee \cdots \vee A_i \leftarrow B_1, \ldots, B_j, \neg C_1, \ldots, \neg C_k.$$

In datalog, the answer to a query is a set of valuations. For instance, the answer to a query $\leftarrow Q(x)$ is a set of constants a such that $Q(a)$ holds. In disjunctive deductive databases, an answer may also be a disjunction $Q(a) \vee Q(b)$.

Disjunctions give rise to new problems of semantics for logic programs. Although in datalog each program has a unique minimal model, this is no longer the case for datalog with disjunctions. For instance, consider the database consisting of a single statement $\{Q(a) \vee Q(b)\}$. Then there are clearly two minimal models: $\{Q(a)\}$ and $\{Q(b)\}$. This leads to semantics in terms of *sets of minimal models*, which can be viewed as incomplete databases. We can develop a fixpoint theory for disjunctive databases, extending naturally the fixpoint approach for datalog. To do this, we use an ordering over *sets of minimal interpretations* (i.e., sets $\mathcal{I}$ of instances such that there are no I, J in $\mathcal{I}$ with $I \subset J$).

Definition 19.5.1 Let $\mathcal{I}$, $\mathcal{J}$ be sets of minimal interpretations. Then

$$\mathcal{J} \sqsubseteq \mathcal{I} \text{ iff } \forall I \in \mathcal{I}\ (\exists J \in \mathcal{J}\ (J \subseteq I)).$$

Consider the following immediate consequence operator. Let P be a datalog program with disjunctions, and let $\mathcal{I}$ be a set of minimal interpretations. A new set $\mathcal{J}$ of interpretations is obtained as follows. For each I in $\mathcal{I}$, $state_P(I)$ is the set of disjunctions of the form $A_1 \vee \cdots \vee A_i$ that are immediate consequences of some facts in I using P. Then $\mathcal{J}$ is the set of of instances J such that for some $I \in \mathcal{I}$, J is a model of $state_P(I)$ containing I. Clearly, $\mathcal{J}$ is not a set of minimal interpretations. The immediate consequence of $\mathcal{I}$, denoted $T_P(\mathcal{I})$, is the set of minimal interpretations in $\mathcal{J}$. Now consider the sequence

$$\begin{aligned}\mathcal{I}_0 &= \emptyset\\ \mathcal{I}_i &= T_P(\mathcal{I}_{i-1}).\end{aligned}$$

It is easy to see that the sequence $\{\mathcal{I}_i\}_{i\geq 0}$ is nondecreasing with respect to the ordering $\sqsubseteq$, so it becomes constant at some point. The semantics of P is the limit of the sequence.

When negation is introduced, the situation, as usual, becomes more complicated. However, it is possible to extend semantics, such as stratified and well founded, to disjunctive deductive databases.

Overall, the major difficulty in handling disjunction is the combinatorial explosion it entails. For example, the fixpoint semantics of datalog with disjunctions may yield a set of interpretations exponential in the input.

Logical Databases and KL

The approach to null values adopted here is essentially a *semantic* approach, because the meaning of an incomplete database is a set of possible instances. One can also use a *syntactic*, proof-theoretic approach to modeling incomplete information. This is done by regarding the database as a set of sentences, which yields the *logical database* approach.

As discussed in Chapter 2, in addition to statements about the real world, logical databases consider the following:

1. *Uniqueness axioms*: State that distinct constants stand for distinct elements in the real world.
2. *Domain closure axiom*: Specify the universe of constants.
3. *Completion axiom*: Specify that no fact other than recorded holds.

Missing in both the semantic and syntactic approaches is the ability to make more refined statements about what the database knows. Such capabilities are particularly important in applications where the real world is slowly discovered through imprecise data. In such applications, it is general impossible to wait for a complete state to answer queries, and it is often desirable to provide the user with information about the current state of knowledge of the database.

To overcome such limitations, we may use languages with modalities. We briefly mention one such language: KL. The language KL permits us to distinguish explicitly between the real world and the knowledge the database has of it. It uses the particular modal symbol K. Intuitively, whereas the sentence φ states the truth of φ in the real world, $K\varphi$ states that the database knows that φ holds.

For instance, the fact that the database knows neither that Alice is a student nor that she is not is expressed by the statement

$$\neg K\, Student(Alice) \wedge \neg K(\neg Student(Alice)).$$

The following KL statement says that there is a teacher who is unknown:

$$\exists x(Teacher(x) \wedge \neg K(Teacher(x))).$$

This language allows the database to reason and answer queries about its own knowledge of the world.

Incomplete Information in Complete Databases

Incomplete information often arises naturally even when the focus is on complete databases. The following are several situations that naturally yield incomplete information:

- *Views*: Although a view of a database is usually a complete database, the information it contains is incomplete relative to the whole database. For a user seeing the view, there are many possible underlying databases. So the view can be seen as a representation for the set of possible underlying databases. The incompleteness of the information in the view is the source of the difficulty in view updating (see Chapter 22).
- *Weak universal relations*: We have already seen how relations with nulls arise in the weak universal relations of Chapter 11.
- *Nondeterministic queries*: Recall from Chapter 17 that nondeterministic languages have several possible answers on a given input. Thus we can think of nondeterministic queries as producing as an answer a set of possible worlds (see also Exercise 19.20).
- *Semantics of negation*: As seen in Chapter 15, the well-founded semantics for datalog$^\neg$ involves 3-valued interpretations, where some facts are neither true nor false but unknown. Clearly, this is a form of incomplete information.

Bibliographic Notes

It was accepted early on that database systems should handle incomplete information [Cod75]. After some interesting initial work on the topic (e.g., [Cod75, Gra77, Cod79, Cod82, Vas79, Vas80, Bis81, Lip79, Lip81, Bis83]), the landmark paper [IL84] laid the formal groundwork for incomplete databases with nulls of the unknown kind and introduced the notion of representation system. That paper assumed the OWA, as opposed to the CWA that was assumed in this chapter. Since then, there has been considerable work on querying incomplete information databases. The focus of most of this work has been a search for the correct semantics for queries applied to incomplete information databases (e.g., [Gra84, Imi84, Zan84, AG85, Rei86, Var86b]).

Much of the material presented in this chapter is from [IL84] (although it was presented there assuming the OWA), and we refer the reader to it for a detailed treatment.

Tables form the central topic of the monograph [Gra91]. Examples in Section 19.1 are taken from there. The naive tables have been called "V-tables" and "e-tables" in [AG85, Gra84, IL84]. The c-tables with local conditions are from [IL84]; they were augmented with global conditions in [Gra84]. The fact that c-tables provide a strong representation system for relational algebra is shown in [IL84]. That this strong representation property extends to query languages with fixpoint on positive queries is reported in [Gra91]. Chasing is applied to c-tables in [Gra91].

There are two main observations in the literature on certainty semantics. The first observation follows from the results of [IL84] (based on c-tables) and [Rei86, Var86b] (based on logical databases). Namely, under particular syntactic restrictions on c-tables and using positive queries, the certainty question can be handled exactly as if one had a complete information database. The second observation deals with the negative effects of the many possible instantiations of the null values (e.g., [Var86b]).

Comprehensive data-complexity analysis of problems related to representing and querying databases with null values is provided in [IL84, Var86b, AKG91]. The program complexity of evaluation is higher by an exponential than the data complexity [Cos83, Var82a]. Such problems were first noted in [HLY80, MSY81] as part of the study of nulls in weak universal instances.

Early investigations suggesting the use of orderings in the spirit of denotational semantics for capturing incomplete information include [Vas79, Bis81]. The first paper to develop this approach is [BJO91], which focused on fd's and universal relations. This has spawned several papers, including an extension to complex objects (see Chapter 20) [BDW88, LL90], mvd's [Lib91], and bags [LW93b]. An important issue in this work concerns which power domain ordering is used (Hoare, Smyth, or Plotkin); see [BDW91, Gun92, LW93a].

The logical database approach has been largely influenced by the work of Reiter [Rei78, Rei84, Rei86] and by that of Vardi [Var86a, Var86b]. The extension of the fixpoint operator of logic programs to disjunctive logic programs is shown in [MR90]. Disjunctive logic programming is the topic of [LMR92]. A survey on deductive databases with disjunctions can be found in [FM92]. The complexity of datalog with disjunction is investigated in [EGM94].

A related but simpler approach to incomplete information is the use of "or-sets." As a simple example, a tuple $\langle Joe, \{20, 21\}\rangle$ might be used to indicate that Joe has age either 20 or 21. This approach is introduced in [INV91a, INV91b] in the context of complex objects; subsequent works include [Rou91, LW93a].

One will find in [Lev84b, Lev84a] entry points to the interesting world of knowledge bases (from the viewpoint of incompleteness of information), including the language KL. A related, active area of research, called reasoning about knowledge, extends modal operators to talk about the knowledge of several agents about facts in the world or about each other's knowledge. This may be useful in distributed databases, where sites may have different knowledge of the world. The semantics of such statements is in terms of an extension of the possible worlds semantics, based on Kripke structures. An introduction to reasoning about knowledge can be found in [Hal93, FHMV95].

Finally, nonapplicable nulls are studied in [LL86]; open nulls are studied in [GZ88]; and weak instances with nonapplicable nulls are studied in [AB87b].

Exercises

Exercise 19.1 Consider the c-table in Example 19.3.1. Give the c-tables for the answers to these queries: (1) Which students are taking Math? (2) Which students are not taking Math? (3) Which students are taking Biology? In each case, what are the sets of sure and possible tuples of the answer?

Exercise 19.2 Consider the c-table T' in Fig. 19.3. Show that each I in $rep(T')$ has two tuples. Is T' equivalent to some 2-tuple c-table?

Exercise 19.3 Consider the naive table in Fig. 19.2. In the weak representation system described in Section 19.1, compute the naive tables for the answers to the queries $\sigma_{A=C}(R)$, $\pi_{AB}(R) \bowtie \pi_{AC}(R)$. What are the tuples surely in the answers to these queries?

Exercise 19.4 A ternary c-table T represents a directed graph with blue, red, and yellow edges. The first two columns represent the edges and the last the colors. Some colors are unknown. The local conditions are used to enforce that a blue edge cannot follow a red one on a path. Give a datalog query q stating that there is a cycle with no two consecutive edges of the same color. Give c-tables such that (1) there is surely such a cycle; and (2) there may be one but it is not sure. In each case, compute the table strongly representing the answer to q.

Exercise 19.5 Let T be the Codd table in Fig. 19.1. Compute strong representations of the results of the following queries, using c-tables: (a) $\sigma_{A=3}(R)$; (b) $q_1 = \delta_{BC\to AB}(\pi_{BC}(R))$; (c) $q_1 \cup \pi_{AB}(R)$; (d) $q_1 \cap \pi_{AB}(R)$; (e) $q_1 - \pi_{AB}(R)$; (f) $q_1 \bowtie \pi_{BC}(R)$.

Exercise 19.6 Consider the c-table $T_4 = T_1 \cup T_2$ of Fig. 19.4. Compute a strong representation of the transitive closure of T_4.

Exercise 19.7 Complete the proof that Codd tables are not a weak representation system with respect to SPU, in Theorem 19.2.1.

Exercise 19.8 Example 19.1.1 shows that one cannot strongly represent the result of a selection on a table with another table. For which operations of relational algebra applied to tables is it possible to strongly represent the result?

Exercise 19.9 Prove that naive tables are not a weak representation system for relational algebra.

Exercise 19.10 Prove that, given a c-table T without constants, $rep(T)$ is the closure under isomorphism of a finite set of instances. Extend the result for the case with constants.

Exercise 19.11 Provide an algorithm for testing equivalence of c-tables.

★ **Exercise 19.12** Show that there exists a datalog query q such that, given a naive table T and a tuple t, testing whether t is possibly in the answer is NP-complete.

Exercise 19.13 Prove Theorem 19.3.2.

Exercise 19.14 Prove that for each c-table T_1 and each set of fd's and mvd's, there exists a table T_2 such that $chase_\Sigma(rep(T_1)) = rep(T_2)$. *Hint:* Use the chase on c-tables.

★ **Exercise 19.15** Show that there is a query q in polynomial time for which deciding, given I and a c-table T, (a) whether $I \in q(rep(T))$, or whether I is possible, are NP-complete; and (b) whether $q(rep(T)) \subseteq \{I\}$, or whether I is certain, are co-NP-complete.

Exercise 19.16 Give algorithms to compute, for a c-table T and a relational algebra query q, the set of tuples $sure(q, T)$ surely in the answer and the set of tuples $poss(q, T)$ possibly in the answer. What is the complexity of your algorithms?

Exercise 19.17 Let T be a c-table and q a positive existential query of the same arity as T. Show that the sequence $\overline{q}^i(T)$ converges [i.e., that for some i, $\overline{q}^i(T) \equiv \overline{q}^{i+1}(T)$]. *Hint:* Show that the sequence converges in at most m stages, where $m = max\{i \mid q^i(I) = q^{i+1}(I), I \in \mathcal{I}\}$ and where $\mathcal{I}$ is a finite set of relations representing the nonisomorphic instances in $rep(T)$.

Exercise 19.18 Describe how to generalize the technique of chasing by full dependencies to apply to instances rather than tableau. If an egd can be applied and calls for two distinct constants to be identified, then the chase ends in failure. Show that for instance I, if the chase of I by Σ succeeds, then $chase(I, \Sigma) \models \Sigma$.

Exercise 19.19 Show that for datalog programs with disjunctions in heads of rules, the sequence $\{\mathcal{I}_i\}_{i \geq 0}$ of Section 19.5 converges. What can be said about the limit in model-theoretic terms?

♠ **Exercise 19.20** [ASV90] There is an interesting connection between incomplete information and *nondeterminism*. Recall the nondeterministic query languages based on the *witness* operator W, in Chapter 17. One can think of nondeterministic queries as producing as an answer a set of possible worlds. In the spirit of the sure and possible answers to queries on incomplete databases, one can define for a nondeterministic query q the deterministic queries $sure(q)$ and $poss(q)$ as follows:

$$sure(q)(I) = \cap\{J \mid J \in q(I)\}$$
$$poss(q)(I) = \cup\{J \mid J \in q(I)\}$$

Consider the language $FO + W$, where a program consists of a finite sequence of assignment statements of the form $R := \varphi$, where φ is a relational algebra expression or an application of W to a relation. Let $sure(FO + W)$ denote all deterministic queries that can be written as $sure(q)$ for some $FO + W$ query q, and similarly for $poss(FO + W)$. Prove that

(a) $poss(FO + W) =$ NP, and

(b) $sure(FO + W) =$ CO-NP.