Equational Theories and Database Constraints

by

Stavros Stylianos Cosmadakis

B.S., Massachusetts Institute of Technology (1981)

M.S., Massachusetts Institute of Technology (1983)

Submitted to the Department of Electrical Engineering
and Computer Science in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy in Computer Science

at the

Massachusetts Institute of Technology

August 1985

Signature of Author ....
Department of Electrical Engineering and Computer Science
August 1985

Certified by ....
Paris C. Kanellakis, Thesis Co-Supervisor

Certified by ....
Albert R. Meyer, Thesis Co-Supervisor

Accepted by ....
Arthur C. Smith, Chairman, Departmental Committee on Graduate Students

Equational Theories and Database Constraints

Stavros Stylianos Cosmadakis

Submitted to the Department of Electrical Engineering and Computer Science
on August 1985, in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Computer Science

Abstract

The *implication problem* for database constraints is central in the fields of automated schema design and query optimization and has been traditionally approached with resolution-based techniques. We present a novel approach to database constraints, using *equations* instead of Horn clauses. This formulation enables us to use new techniques for database theory, which derive from universal algebra, equational logic and lattice theory. It also points to the possibility of employing theorem-proving techniques originally developed for equational theories to deal with implication in the context of logical databases.

We apply our approach to study *functional* and *inclusion* dependencies. These constraints can model functional determination and data duplication and they have been extensively proposed as a powerful and realistic feature for semantic data models. We prove completeness of new proof procedures and we derive new upper and lower bounds for the complexity of various implication problems involving these dependencies.

We also present a new class of constraints which are defined equationally, using algebraic operations on set-theoretic partitions. These *partition dependencies* provide an elegant generalization of functional dependencies (in the direction of incorporating *transitive closure*), for which the implication problem remains efficiently solvable.

Thesis Co-Supervisor: Paris C. Kanellakis, Visiting Assistant Professor of Computer Science (on leave from Brown University).

Thesis Co-Supervisor: Albert R. Meyer, Professor of Computer Science.

Keywords: Relational data model, logical databases, dependencies, implication, proof procedures, completeness, equational theories, word problems, lattices.

Chapter One

Introduction

1.1 Functional and Inclusion Dependencies in the Relational Model

The development of the *relational data model* [21, 22] led to major progress in the area of database management. The model and its implementations have contributed significantly both to the increase of programmer productivity [23] and to the fundamental understanding of computation [62].

Among the advantages of the model, which account for its success, are [23]:

1. The sharp, clear boundary it provides between the conceptual and the physical aspects of database management.
2. Its simplicity, which allows users and programmers to have a common understanding of the data and therefore communicate easily about it.
3. The introduction of truly high level language concepts, which enables users to express operations on large pieces of information, without detailed knowledge of its representation or of the access paths to where it is stored.
4. A sound, mathematical foundation, which makes possible the theoretical study of the (often formidable) problems of database design and manipulation.

The relational data model consists of a *structural* part (with a unique data type, the *relation*), a *manipulative* part (with powerful algebraic operators such as *selection*, *projection* and *join*) and an *integrity* part (constraints defining consistent database states, intended to capture the *semantics* of particular applications) [62, 51]. A *relation* is a table with columns named by *attributes* and with rows containing values from some *domain*, each row being a *tuple*. A *database* is a finite set of relations. A *logical database* or *database schema* consists of a *database scheme*, i.e. a finite set D of *relation schemes* (sequences of attributes naming the columns of relations), along with a finite set Σ of *integrity constraints* (*dependencies*), which should be satisfied by all legal *physical databases* (*database instances*).

For an example (invariant throughout the database literature), consider a database of two relations

R,S, where R has attributes EMPLOYEE and MANAGER and S has attributes MANAGER and DEPARTMENT. If we take as our semantic restrictions that "every employee has exactly one manager" and "every manager manages exactly one department", we define the following database schema:

$D = \{R[\text{EMPLOYEE, MANAGER}], S[\text{MANAGER, DEPARTMENT}]\}$
$\Sigma = \{R{:}\text{EMPLOYEE} \rightarrow \text{MANAGER}, S{:}\text{MANAGER} \rightarrow \text{DEPARTMENT}\}$

In this case, our constraints are examples of *functional dependencies* [21, 22, 62, 51]. Formally, a *functional dependency* (FD) is an assertion of the form $R{:}X \rightarrow Y$, where R is the name of a relation and X,Y are sets of attributes from the relation scheme of R. It is satisfied by a database instance iff whenever two tuples of relation R agree on all attributes appearing in X, they also agree on all attributes appearing in Y. Observe that, with no loss of generality, we can take Y to consist of a single attribute.
Functional dependencies form a conceptually simple and naturally occuring class of constraints. For this reason, they have been extensively studied in the literature (see [7, 62, 51] for reviews of the area). Combined with the algebraic operators of the relational model they provide a practical and elegant approach to the problems of database design and manipulation.

At present, a major research effort is underway towards *extending* the relational model. This effort is motivated in large part by the success of the relational methodology and by the demands of specific application domains, in particular Office Automation (see, e.g., [20, 24, 37, 42, 59, 61], which is by no means an exhaustive list). The approach generally taken is to appropriately enrich the integrity part by adding constraints which will enhance the expressive power of the model, while at the same time preserving its original advantages.

Returning to our example, suppose we also want to be able to express simple facts such as "everyone who manages employees belongs to some department". In other words, we want to add to the semantics of our relations that a MANAGER entry in relation R must also appear as a MANAGER entry in relation S. This constraint is formally captured by the *inclusion dependency* [16] $R{:}\text{MANAGER} \subseteq S{:}\text{MANAGER}$. In general, an *inclusion dependency* (IND) is a statement of the form $R{:}A_1 \ldots A_m \subseteq S{:}B_1 \ldots B_m$. Such a statement is satisfied by a database instance iff whenever a tuple with entries $a_1,\ldots,a_m$ for attributes $A_1,\ldots,A_m$ appears in relation R, a tuple with entries $a_1,\ldots,a_m$ for attributes $B_1,\ldots,B_m$ appears in relation S.

Inclusion dependencies make it possible to selectively define what data must be duplicated in what relations and thus they provide a valuable tool for database design [24, 59, 69]. The central notion of *referential integrity* [24, 29] can be expressed using IND's. Together with FD's, IND's form the basis of the structural model of [67]. Descriptions of logical databases written in a variety of languages can be translated into a common language which uses relations, FD's and IND's [45]. Inclusion dependencies have also been employed to map an entity-relationship schema to the relational model [20]. We mention in passing that IND's have been commonly known in Artificial Intelligence applications as *ISA* relationships (cf. [9]).

Although the addition of IND's to the relational model has been recognized as realistic and desirable (because of their conceptual simplicity and expressive power), they have become only recently the object of theoretical investigation [16, 43, 54, 19, 58, 17, 44, 48, 26]. General questions relating to the *implication problem* for IND's and FD's have been studied in [16, 54, 19]. A rather surprising result [54, 19] is that the combination of IND's with FD's is as powerful computationally as first-order predicate calculus. This result can be considered both positive (as it hints to the possibly rich potential of two simple primitive forms) and negative, as it implies inherent computational intractability of the general case. From a more practical standpoint, [43, 17, 44, 26] provide solutions to database design and query optimization problems in the presence of (suitably restricted) IND's and FD's. Also, central notions such as the Universal Instance Assumption [62, 51] have been investigated using IND's [58, 48]. We will review the theoretical work on IND's in more detail in the sequel.

1.2 The Implication Problem

The (*unrestricted*) *implication problem* for a class of dependencies is the following: Given a finite set Σ of dependencies and a dependency σ, test if σ holds in *all* (not necessarily finite) databases which satisfy the dependencies in Σ. By restricting attention to *finite* databases, we obtain the *finite implication problem.*

Solving the implication problem is the main computational task associated with a class of dependencies. As a rule, algorithmic approaches to database schema design and query optimization are based on efficient solutions of the implication problem (see, e.g., [12, 6, 3, 18, 62, 51]). Evidently, if we are concerned with applications then the *finite* implication problem is the one which is most relevant. However, it tends to be much more difficult to deal with. Moreover, for the classes of dependencies for which implication is decidable, it generally happens that finite implication coincides with unrestricted implication.

The problem of dependency implication can be approached in a very general setting by formulating dependencies as sentences in first-order logic, namely as *Horn clauses* [34] (see Section 5.1 of this thesis for some examples). Closely related to this approach is a particular proof procedure, the *chase*; see [52, 11, 62, 51] for its wide applicability (proof procedures for general dependencies also appear in [10, 68, 57]). It has been observed that the chase is a special case of a classical theorem proving technique, namely *resolution* [10, 11]. The chase provides straightforward algorithms for implication of classes of dependencies for which it can be shown to terminate. Furthermore, in these cases the chase produces a *finite counterexample* whenever implication does not hold; it is for this reason that finite implication coincides with unrestricted implication in these cases.

Returning now to functional and inclusion dependencies, what appears to be the fundamental difficulty is precisely that IND's can prevent the chase from terminating. Of course, in the case of general FD's and IND's one cannot hope to circumvent this obstacle, since the implication problem is undecidable [54, 19]. Nevertheless, given the practical importance of these dependencies it makes sense to study the complexity of special cases. The obvious approach that has been suggested is to analyze the chase, but this turns out to be a very delicate task (cf. [43]), which can only give partial results [43, 26]. Thus, it seems that new tools are required in order to make major progress.

The main contribution of this thesis is the introduction of such tools, borrowed from *equational logic*. This is a fragment of first-order logic which has attracted a lot of attention, because of its relevance to areas such as applicative languages, interpreters and data types (see [41] for a survey). However, it does not seem to have been noticed by the database theory community, since a constant effort has been made to minimize the role of equality in dependencies (*multivalued dependencies* (MVD's) [62, 51], the most widely studied after FD's, do not involve equality). The only case where ideas from equational logic were applied in database theory seems to be the best algorithm for *losslessness of joins* (a basic computational problem), which was derived from an efficient algorithm for *congruence closure* [31]. Also, the best algorithm for implication of FD's [6] can be seen directly (as we observe) as a special case of an algorithm of [47] for the *generator problem in finitely presented algebras*.

We use the methods of equational logic to formulate and study implication problems involving

FD's and IND's. We also use equations to define a new class of dependencies (generalizing FD's) and to investigate its implication problem. In the subsequent Sections, we review in more detail the content of each Chapter.

1.3 Chapter Two: The Equational Approach to Dependencies

Let r be a relation over a set of attributes $\mathfrak{U}$, with values taken from a domain $\mathfrak{D}$. Suppose r satisfies the FD $AB \rightarrow C$, i.e. whenever two tuples of r agree on A,B they also agree on C (here and in the sequel we consider single relations, so we can suppress relation names from dependencies). Let x be a variable ranging over the tuples of r and let $a(x)$ ($b(x)$, $c(x)$) be a function which assigns to a tuple x the entry of x at attribute A (B,C). Now since r satisfies $AB \rightarrow C$, it is easy to see that there is a function f (from $\mathfrak{D}^2$ to $\mathfrak{D}$) such that the following sentence is true in r:

$$\forall x. f(a(x), b(x)) = c(x)$$

This observation suggests the following *syntactic* transformation: the FD $AB \rightarrow C$ is rewritten as an equation

$$faxbx = cx,$$

where now the symbol a (b,c) is a *function symbol* of ARITY 1 representing the attribute A (B,C) and f is a function symbol of ARITY 2 corresponding to the FD. Using the standard convention of equational logic, we omit the universal quantifier on the variable x.

We now illustrate how this equational formalism can be used to infer FD's.

Example 1.1: Given the FD's

$$A \rightarrow B_1,\ A \rightarrow B_2,\ B_1B_2 \rightarrow C$$

we can infer the FD $A \rightarrow C$. Using our transformation, the given set of FD's produces the equations

$$f_1ax = b_1x,\ f_2ax = b_2x,\ gb_1xb_2x = cx.$$

From these we can infer the equation

$$gf_1axf_2ax = cx.$$

In general, we can infer an FD such as $A \rightarrow C$ if we can infer an equation $\tau[x/ax] = cx$, where τ is a term over the f's and a variable x (in Example 1.1, τ is the term gf_1xf_2x). The notation $\tau[x/ax]$ means that we substitute ax for x in τ.

Interestingly, this equational formulation can be extended to IND's as well. Suppose relation r satisfies the IND $A_1A_2 \subseteq B_1B_2$, i.e. for each tuple t of r there is a tuple t′ of r such that the values of t′ on B_1,B_2 are the same as the values of t on A_1,A_2 respectively. This means the following sentence is true in r:

$$\forall x\ \exists y.\ [b_1(y) = a_1(x) \wedge b_2(y) = a_2(x)]$$

(as before, x,y are variables ranging over the tuples of r and a_1,a_2,b_1,b_2 are functions corresponding to the attributes A_1,A_2,B_1,B_2).
Consider now the *Skolemization* of the existential quantifier $\exists y$: one obtains the sentence

$$\forall x.\ [b_1(i(x)) = a_1(x) \wedge b_2(i(x)) = a_2(x)],$$

which is true in r for some suitable function $i(x)$ (from tuples to tuples). This suggests transforming the IND $A_1A_2 \subseteq B_1B_2$ into the *set* of equations

$$b_1ix = a_1x,\ b_2ix = a_2x$$

(here i is a function symbol of ARITY 1 corresponding to the IND).

Example 1.2: From the dependencies

$$A_1A_2 \subseteq B_1B_2,\ A_2A_3 \subseteq B_2B_3,\ B_2 \rightarrow B_3$$

we can infer the IND $A_1A_2A_3 \subseteq B_1B_2B_3$ [16, 54]. Using our transformation, the given set of dependencies produces the equations

$$\begin{aligned}&b_1ix = a_1x,\ b_2ix = a_2x,\\ &b_2jx = a_2x,\ b_3jx = a_3x,\\ &fb_2x = b_3x.\end{aligned}$$

From these we can infer

$$b_3ix = fb_2ix = fa_2x = fb_2jx = b_3jx = a_3x,$$

i.e. we can infer the set of equations

$$b_1ix = a_1x,\ b_2ix = a_2x,\ b_3ix = a_3x.$$

In general, we can infer an IND such as $A_1A_2A_3 \subseteq B_1B_2B_3$ if we can infer a set of equations $b_1\tau = a_1x$, $b_2\tau = a_2x$, $b_3\tau = a_3x$, where τ is some term over the i's and a variable x (in Example 1.2, τ is simply ix).

Thus, we can use equational reasoning to obtain a proof procedure for FD's and IND's. The *soundness* and *completeness* of this approach is demonstrated in Theorem 2.1. As a matter of fact, the soundness part (whenever an equation of the appropriate form is implied, the corresponding dependency is implied) is easy and it should already be plausible from the preceding discussion. The difficult part is completeness (whenever a dependency is implied, an equation of the appropriate form is implied). This is proved by a rather delicate induction, which shows that equational reasoning can simulate the chase.

We can also have a slightly different syntactic transformation of dependencies into equations. This transformation, however, does not have a straightforward semantic justification.

Consider the FD's in Example 1.1: We can transform them into the equations

$$f_1a=b_1,\ f_2a=b_2,\ gb_1b_2=c,$$

from which we can infer the equation

$$gf_1af_2a=c.$$

The symbols a,b_1,b_2,c are now *constant symbols* representing the attributes A,B_1,B_2,C.

When approached this way, the implication problem for FD's becomes a special case of the *generator problem for finitely presented algebras* [47], for which [47] gives a polynomial-time algorithm. By inspecting the behaviour of [47]'s algorithm in this special case, we obtain the linear-time algorithm for implication of FD's given in [6].

This alternative transformation can also be extended to IND's. We transform the IND $A_1A_2 \subseteq B_1B_2$ into the set of equations

$$ib_1=a_1,\ ib_2=a_2.$$

Observe that we have now eliminated the variable x, which can play an essential role when IND's are combined with FD's (cf. Example 1.2). For this reason we also need equations of the form

$$fix=ifx,$$

which permit us to move the f's over the i's and vice versa. The soundness and completeness of this approach is also proved in Theorem 2.1.

The equational formulation of dependencies is more redundant than the standard one, since we need to introduce new symbols (f's and i's). On the other hand, inferences of dependencies now give us more information: whenever we infer a dependency σ from a set of dependencies Σ, the associated term τ (cf. Examples 1.1, 1.2) tells us how σ results (in any database satisfying Σ) by "composing" dependencies in Σ.

In the remainder of Chapter 2, we use our equational approach to prove several results relating to

FD and IND implication. We first give a new proof procedure for FD's and IND's (Theorem 2.2). This proof procedure is different in spirit both from the chase and the proof procedure of [54] and it treats FD's and IND's in a symmetric fashion. The equational tools come into play in the proof of *completeness* of this proof procedure. Usually, completeness is proved by constructing a *database* which satisfies a set of dependencies Σ but violates a dependency σ (assuming σ cannot be proved from Σ); see, e.g., [11, 54, 62]. In our case, we consider the set of equations $\mathcal{E}_\Sigma$ obtained from Σ and we construct an *algebra* which satisfies $\mathcal{E}_\Sigma$ but violates any equation that could correspond to σ.

Our second result is a precise characterization of the complexity of *acyclic* IND's and FD's. Intuitively, a set of IND's is *acyclic* [58] if it does not contain any cycles of inclusions, such as $\{R{:}A_1A_2 \subseteq R{:}B_1B_2\}$, $\{R{:}A \subseteq S{:}B,\ S{:}B' \subseteq R{:}A'\}$ and so on. Acyclic sets of IND's have been proposed as a useful tool for database schema design [58]. One can easily observe that the implication problem for acyclic IND's and FD's can be solved in exponential time (the chase terminates in this case). NP-hardness lower bounds for the problem were obtained in [26].

We show that the implication problem for acyclic IND's and FD's *requires* exponential time (Theorem 2.4). The main observation is that, when all FD's are *unary* (i.e. the left-hand side contains a single attribute), the equational inferences of Examples 1.1, 1.2 can be viewed as inferences in semigroups (Corollary 2.3). Such inferences can in turn simulate computations of an automaton with two pushdown stores. Since such automata are universal computing devices, we obtain a tight undecidability result for FD and IND implication (Theorem 2.3). Furthermore, the acyclicity condition on the IND's corresponds to *bounding* the size of one of the pushdown stores, which gives us exponential time.

1.4 Chapter Three: Application to Typed IND's

A usual assumption in database theory is that all database relations are projections of a single *universal* relation (Universal Instance Assumption [62, 51]). In practice this is not always the case, so one has the problem of testing the existence of a universal instance and the problem of adjusting the database relations to maintain the existence of a universal instance as the database is updated. Both of these problems are known to be NP-complete [39]. An alternative, weaker condition we may impose on a multi-relational database is *pairwise consistency*, i.e. *every pair* of the database relations is required to have a universal relation. This condition is easy to test and maintain, as described in numerous works on the subject (see [8] for a review). In fact, if the database scheme is acyclic [8] then pairwise consistency *implies* the existence of a universal instance.

Most of the theoretical work on dependencies is done in the context of databases consisting of a single relation, i.e. it assumes the existence of a universal instance [62, 51]. A natural question, then, is to investigate the effect of the weaker assumption of pairwise consistency on the implication problem, say for functional dependencies. Although the implication problem for FD's is solvable in linear time assuming a universal instance [6], it is not clear even if it is decidable in the context of pairwise consistency.

Let r_1,r_2 be relations over relation schemes $R_1[U_1]$, $R_2[U_2]$ respectively. It is not difficult to see that r_1,r_2 have a universal instance iff the projection of r_1 on $U_1 \cap U_2$ is the same as the projection of r_2 on $U_1 \cap U_2$ [1]. This can be expressed (with a slight abuse of notation) by the pair of IND's

$$R_1{:}U_1 \cap U_2 \subseteq R_2{:}U_1 \cap U_2$$
$$R_2{:}U_1 \cap U_2 \subseteq R_1{:}U_1 \cap U_2.$$

These are examples of *typed* IND's. An IND is *typed* [17, 48] if it has the form $R{:}A_1...A_m \subseteq S{:}A_1...A_m$. By the above observation, we can then formulate the implication problem for FD's in the presence of pairwise consistency as an implication problem for FD's and (typed) IND's.

In this Chapter, we apply the equational techniques of Chapter 2 to study the implication problem for FD's and *typed* IND's. The main tool we develop is a proof procedure for general FD's and IND's (Theorem 3.1). This proof procedure is different from the procedure of Theorem 2.2 and somewhat reminiscent in spirit of the axiomatization of [54]. We prove completeness of the procedure by showing that it captures (indirectly) equational inferences as in Examples 1.1, 1.2.

By analyzing the behaviour of this proof procedure in the case of typed IND's, we obtain a decidability result for typed IND's and FD's satisfying an acyclicity condition (Corollary 3.1). We then further specialize the proof procedure to the case of unary FD's in the presence of pairwise consistency (Lemma 3.2). By a rather complicated analysis of derivations, we show that this implication problem is *undecidable* (Theorem 3.3). This provides a very tight undecidable case of FD and IND implication.

Finally, we use Lemma 3.2 to show that there is no *k-ary* axiomatization (involving only FD's and IND's) for implication of unary FD's under pairwise consistency (Theorem 3.4; the technical notion of a k-ary axiomatization is explained in Chapter 3). This strengthens a previous result of [16] about non-existence of k-ary axiomatizations for FD's and IND's.

1.5 Chapter Four: Finite Implication of FD's and Unary IND's

Given the importance of the finite implication problem, it is natural to ask if our equational approach can be extended to finite implication. Unfortunately, there are difficulties. The *completeness* part of Theorem 2.1 is proved by analyzing a proof procedure (the chase). However, in the case of finite implication of FD's and IND's such a proof procedure does not even exist [54, 19].

Nevertheless, we can have a complete proof procedure for finite implication of FD's and IND's, if we restrict ourselves to IND's with one attribute per side (*unary* IND's). Unrestricted implication becomes rather uninteresting in this case, because FD's and unary IND's do not interact in any non-trivial way (Proposition 4.1). However, in the finite case we have the following interaction:

from $A_0 \rightarrow A_1$ *and* $A_1 \supseteq A_2$ *and...and* $A_{m-1} \rightarrow A_m$ *and* $A_m \supseteq A_0$
derive $A_1 \rightarrow A_0$ *and* $A_2 \supseteq A_1$ *and...and* $A_m \rightarrow A_{m-1}$ *and* $A_0 \supseteq A_m$
(m odd).

It turns out that this is the *only* non-trivial interaction: by turning the above observation into a set of inference rules (one for each odd m) and including the usual inference rules for FD's [5] and IND's [16], we obtain a *complete* axiomatization for FD's and unary IND's in the finite case (Theorem 4.1). The completeness proof is rather long and it involves an intricate construction of a finite counterexample relation. We also remark that this axiomatization leads to a polynomial-time algorithm for finite implication of FD's and unary IND's [44]. The class of FD's and unary IND's is the only known class of dependencies for which unrestricted and finite implication are both solvable without being identical.

Interestingly, the above axiomatization can also be used to prove an analogue of Theorem 2.1 for finite implication of FD's and unary IND's (Theorem 4.2). However, this result is weaker, in the following way. Suppose, for example, that we want to test if the FD $A \rightarrow B$ is implied from a set of dependencies Σ. In the unrestricted case we can show that, if $A \rightarrow B$ is implied, then there is a term τ such that the equation $\tau[x/ax] = bx$ is implied (cf. Example 1.1); i.e., $\tau[x/ax] = bx$ holds in *all* algebras which satisfy the equations corresponding to Σ. In the finite case, we can only show that, for each algebra $\mathcal{A}$ as above, there is a term τ (depending on $\mathcal{A}$) such that the equation $\tau[x/ax] = bx$ holds in

$\mathcal{A}$.

1.6 Chapter Five: Partition Dependencies

We have presented in Chapter 2 an equational formulation of functional dependencies. One can also have another formulation of quite different flavor, using algebraic operations on *partitions* (this seems to be a folklore observation, see e.g. [15, 60]).

Specifically, let r be a relation and for each attribute A let π_A be the following partition of the set of tuples of r: tuples t,s are in the same block of π_A iff they agree on attribute A. Now it is easy to see that r satisfies the FD $A \rightarrow B$ iff

$$\pi_A \leq \pi_B,$$

or, equivalently,

$$\pi_A = \pi_A \cdot \pi_B,$$
$$\pi_B = \pi_A + \pi_B.$$

Here $\leq$ is the usual *refines* relation and $\cdot, +$ are the usual *product* and *sum* operation on partitions.

We are thus led to consider general equations over $\cdot, +$ and the π_A's. We call such equations *partition dependencies* (PD's) [27].

We first compare the expressive power of PD's to that of previously studied database constraints, namely *embedded implicational dependencies* [34]. A first observation is that PD's of the form $\pi_A = \pi_B + \pi_C$ can express *symmetric transitive closure* (Example 5.2). It follows by a simple compactness argument that such PD's cannot be expressed by any set of EID's (Theorem 5.1). On the other hand, PD's are unable to detect complicated patterns of equalities in relations and for this reason they cannot express, for instance, *multivalued* dependencies (Theorem 5.2).

We then study the implication problem for PD's. We observe that the (finite) implication problem for PD's is equivalent to the uniform word problem for (finite) *lattices* (Lemma 5.1). This follows from two deep results of lattice theory, namely that (finite) equivalence relations can represent arbitrary (finite) lattices [66, 56]. Using techniques from universal algebra [36, 47] and lattice theory [28], we show that these word problems are equivalent and they can be solved in polynomial time (Theorem 5.3).

Finally, we examine the problem of testing *consistency* [38, 64] of a database with a set of PD's. Using our polynomial-time algorithm for implication, we show that it can be reduced to testing consistency with a set of FD's [38]. It follows that the problem can be solved in polynomial time (Theorem 5.4).

1.7 Credits

The research reported in this thesis was done in close collaboration with Paris C. Kanellakis, and has been documented in a series of joint publications [25, 26, 44, 27]. Individual credit for the main results goes as follows:

Theorems 2.1, 2.2, 2.3, 2.4 were obtained jointly, and appeared in [25].

Theorems 3.1, 3.2, 3.3 are due to the author of this thesis, and appeared in [25]. Theorem 3.4 was obtained jointly, and appeared in [26].

Theorem 4.1 was obtained jointly, but Paris C. Kanellakis was the main contributor; this result appeared in [44]. Theorem 4.2 was obtained jointly, and appeared in [25].

Theorem 5.3 was obtained jointly, but the author of this thesis was the main contributor; this result appeared in [27]. Theorems 5.1, 5.2, 5.4 were obtained jointly, and appeared also in [27].

The extension to general dependencies outlined in the concluding chapter is due to the author of this thesis.

Chapter Two

The Equational Approach to Dependencies

We present in this Chapter the equational formalization of functional and inclusion dependencies. Section 2.1 gives the necessary definitions and background from database theory and equational logic. In Section 2.2 we present the main Theorem and its Corollaries. We use it in Section 2.3 to prove completeness of a new proof procedure for FD's and IND's. In Section 2.4 we apply the equational formulation to prove new lower bounds for FD and IND implication.

2.1 Definitions

2.1.1 Relational Database Theory

Let $\mathfrak{U}$ be a finite set of *attributes* and $\mathfrak{D}$ a countably infinite set of *values*, such that $\mathfrak{U} \cap \mathfrak{D} = \emptyset$. A *relation scheme* is an object R[U], where R is the *name* of the relation scheme and $U \subseteq \mathfrak{U}$. A *tuple* t over U is a function from U to $\mathfrak{D}$. Let $U = \{A_1,...,A_n\}$ and a_k a value, $k = 1,...,n$; if $t[A_k] = a_k$, we represent tuple t over U as $a_1 a_2 ... a_n$. We represent the restriction of tuple t on a subset X of U as t[X]. A *relation* r over U (named R) is a (possibly infinite) nonempty set of tuples over U. A *database scheme* D is a finite set of relation schemes $\{R_1[U_1],...,R_q[U_q]\}$ and a *database* $d = \{r_1,...,r_q\}$ associates each relation scheme $R_k[U_k]$ in D with a relation r_k over U_k. A database is finite if all of its relations are finite. A database can be visualized as a set of tables, one for each relation, whose headers are the relation schemes (each column headed by an attribute) and whose rows are the tuples.

The logical constraints which determine the set of legal databases are called *database dependencies* [62, 51]. We will be examining two very common types of dependencies.

FD $R{:}A_1...A_n \rightarrow A$ (n>0) is a *functional dependency* [62, 51].
Relation r (named R) satisfies this FD iff,
for tuples t_1, t_2 in r, $t_1[A_1...A_n] = t_2[A_1...A_n]$ implies $t_1[A] = t_2[A]$.

If $n=1$, i.e. the left-hand side contains a single attribute, we have a *unary functional dependency* (u-FD).

IND $S:D_1...D_m \subseteq R:C_1...C_m$ ($m>0$) is an *inclusion dependency* [16].
Relations s,r (named S,R respectively) satisfy this IND iff,
for each tuple t in s, there is a tuple t_1 in r with $t_1[C_k]=t[D_k]$, $k=1,...,m$.

If $m=1$, we have a *unary inclusion dependency* (u-ID).

Equality of two columns headed by attributes A,B in a relation named R can be expressed as a special case of IND's: Use an IND such as $R:AB \subseteq R:AA$. These dependencies are particularly illustrative of our analysis; we will use $A \equiv B$ to denote them.

Database Notation: We use a graph notation to represent an input database scheme *D* and a set of dependencies Σ (*input schema*). We construct a labeled directed graph G_Σ (see Figure 2-1), which has exactly one node a_k^j for each attribute A_k of each relation scheme R_j. For each IND $R_2:D_1...D_m \subseteq R_1:C_1...C_m$ in Σ, the graph G_Σ contains m *black* arcs $(c_1^1,d_1^2),...,(c_m^1,d_m^2)$; each arc is labeled by the name i of the IND. For each FD $R_1:A_1...A_n \rightarrow A$ in Σ, the graph G_Σ contains a group of n *red* arcs $(a_1^1,a^1),...,(a_n^1,a^1)$; the group is labeled by the name f of the FD and its arcs are ordered from 1 to n as listed above.

We also construct two directed graphs I_Σ and F_Σ (see Figure 2-1): The graph I_Σ has one node for each relation scheme name in *D* and arc (R_j,R_k) iff G_Σ contains some black arc (A^j,B^k). The graph F_Σ has one node a for each attribute A of *D* and arc (a,b) iff G_Σ contains some red arc (a^k,b^k). We now define special syntactically restricted forms of input schemata:

.

Acyclic IND's: I_Σ is acyclic [58].
Acyclic FD's: F_Σ is acyclic.
Typed IND's: The black arcs of G_Σ are all of the form (A^j,A^k) for relation names R_j,R_k and attribute A [17, 48].

Typed IND's are between occurrences of the same attribute names in different relation schemes. If we assume that all possible typed IND's are in the input schema, (i.e., with some abuse of notation $R:U \cap U' \subseteq S:U \cap U'$ for all relation schemes R[U], S[U'] in database scheme *D*), then we have *pairwise consistency* PC(*D*) [48].

Implication: We say that Σ *implies* σ ($\Sigma \models \sigma$) if, whenever a database d satisfies Σ, it also satisfies σ. We say that Σ *finitely implies* σ ($\Sigma \models_{\text{fin}} \sigma$) if, whenever a *finite* database d satisfies Σ, it also satisfies σ.
Clearly if $\Sigma \models \sigma$ (*implication*) then $\Sigma \models_{\text{fin}} \sigma$ (*finite implication*), but the converse is not always true. Deciding implication of dependencies is a central problem in database theory.

Since dependencies are sentences in first-order predicate calculus with equality, we have *proof procedures* for the implication problem (we denote provability as $\Sigma \vdash \sigma$). A proof procedure is *sound* if whenever $\Sigma \vdash \sigma$, we have $\Sigma \models \sigma$; and *complete* if it is sound and whenever $\Sigma \models \sigma$, we have $\Sigma \vdash \sigma$.

The standard complete proof procedure for database dependencies is the *chase* [62, 11]. We now present the chase for FD's and IND's (cf. [43]).

Chase: Given an input schema *D*, Σ and a dependency σ, construct a set of tables T, with *D*'s relation schemes as headers. These tables are originally empty and will be filled with symbols from the countably infinite set $\mathfrak{I}$. Whenever we insert a new row of symbols from $\mathfrak{I}$ in a table of T and we do not specify some of the entries of this row, we assume that distinct symbols from $\mathfrak{I}$, which have not yet appeared elsewhere in T, are used to fill these entries. We use t_k^r for the k-th row of table R and $t_k^r[X]$ for this row's entries in the columns of attributes X.

The *initial configuration* of T depends on σ as follows:
(i) If σ is the FD $R{:}A_1 \ldots A_n \rightarrow A$: insert rows t_1^r, t_2^r with the only restriction that $t_1^r[A_k] = t_2^r[A_k]$, $k = 1,\ldots,n$.
(ii) If σ is the IND $S{:}D_1 \ldots D_m \subseteq R{:}C_1 \ldots C_m$: insert t_1^s.

Every dependency in Σ produces a *rule*, as follows:
If f is an FD in Σ the corresponding FD-rule is:
⟨Consider T a database over symbols in $\mathfrak{I}$. If T does not satisfy f, because two symbols x and y are different, then replace y by x in T⟩.
If i is an IND $R{:}X \subseteq S{:}Y$ in Σ the corresponding IND-rule is:
⟨Consider T a database over symbols in $\mathfrak{I}$. If T does not satisfy i, because some $t^r[X]$ does not appear in the table S as some $t^s[Y]$, then insert t^s in S with $t^s[Y] = t^r[X]$.⟩

We will say that $\Sigma \vdash_{\text{chase}} \sigma$, if there is a finite sequence of applications of the FD-rules and IND-rules produced by Σ that transforms T's initial configuration to a final configuration satisfying:

(i) If σ is an FD as above: $t_1^r[A]=t_2^r[A]$.

(ii) If σ is an IND as above: for some j,

$t_1^s[D_k]=t_j^r[C_k]$, $k=1,...,m$.

Proposition 2.1: $\Sigma\vdash_{chase}\sigma$ iff $\Sigma\models\sigma$. ∎

An alternative proof procedure for FD's and IND's is provided by the axiomatization of [54]. If Σ is a set of FD's and IND's and σ is an FD or IND, then $\Sigma\models\sigma$ iff σ can be proved from Σ using the following rules (X,Y denote sets of attributes):

1. (reflexivity) $R:A\rightarrow A$.

2. (augmentation) *from* $R:X\rightarrow A$ *derive* $R:XY\rightarrow A$.

3. (transitivity) *from* $R:X\rightarrow A_k$, $k=1,...,n$, $R:A_1...A_n\rightarrow A$, *derive* $X\rightarrow A$.

4. (IND reflexivity) $R:A_1...A_m\subseteq R:A_1...A_m$.

5. (IND transitivity) *from* $R_1:A_1...A_m\subseteq R_2:B_1...B_m$ *and* $R_2:B_1...B_m\subseteq R_3:C_1...C_m$ *derive* $R_1:A_1...A_m\subseteq R_3:C_1...C_m$.

6. (permutation, projection and redundancy): *from* $R:A_1...A_m\subseteq S:B_1...B_m$ *derive* $R:A_{j_1}...A_{j_p}\subseteq S:B_{j_1}...B_{j_p}$, where $1\leq j_k\leq m$, $k=1,...,p$.

7. (equivalence) *from* $R:AB\subseteq S:CC$ *and* σ *derive* τ, where τ is obtained from σ by substituting A for one or more occurrences of B.

8. (pullback) *from* $R:A_1...A_nA\subseteq S:B_1...B_nB$ *and* $S:B_1...B_n\rightarrow B$ *derive* $R:A_1...A_n\rightarrow A$.

9. (collection) *from* $R:A_1...A_nB_1...B_m\subseteq S:A_1'...A_n'B_1'...B_m'$, $R:B_1...B_mC\subseteq S:B_1'...B_m'C'$ *and* $S:B_1'...B_m'\rightarrow C'$ *derive* $R:A_1...A_nB_1...B_mC\subseteq S:A_1'...A_n'B_1'...B_m'C'$.

10. (attribute introduction) *from* $R:A_1...A_n\subseteq S:B_1...B_n$ *and* $S:B_1...B_n\rightarrow B$ *derive* $R:A_1...A_nN\subseteq S:B_1...B_nB$, where N is a *new* attribute.

Rules 1-3 are the standard rules for FD's [5, 62] (written in our notation) and Rules 4-6 are the rules of [16] for IND's without repeated attributes. The salient rule is *attribute introduction* (Rule 10). Whenever this rule is applied, the attribute N is chosen to be an attribute which does not appear in Σ or in any previous step of the derivation. Rule 10 is sound in the following sense: Whenever the antecedents are true in relations r,s (over relation schemes R,S respectively), there is a relation r' which differs from r only on a new column headed by N and which satisfies the conclusion.

2.1.2 Equational Logic

Let M be a set of symbols and ARITY a function from M to the nonnegative integers $\mathcal{N}$. The set of finite strings over M is M*. Partition M into two sets:

$G = \{g \in M \mid \text{ARITY}(g) = 0\}$ is the set of *generators*,
$O = \{\theta \in M \mid \text{ARITY}(\theta) > 0\}$ is the set of *operators.*

Definition 2.1: $\mathcal{T}(M)$, the set of *terms* over M, is the smallest subset of M* such that,

1) every g in G is a term,
2) if $\tau_1,\dots,\tau_m$ are terms and θ is in O with $\text{ARITY}(\theta) = m$, then $\theta\tau_1\dots\tau_m$ is a term.

A *subterm* of τ is a substring of τ, which is also a term. Let $V = \{x, x_1, x_2, \dots\}$ be a set of *variables.* The set of terms over operators O and generators $G \cup V$ will be denoted by $\mathcal{T}^+(M)$. For terms $\tau_1,\dots,\tau_n$ in $\mathcal{T}^+(M)$ we have a *substitution* $\varphi = \{ (x_k \leftarrow \tau_k) \mid k = 1,\dots,n \}$, which is a function from $\mathcal{T}^+(M)$ to $\mathcal{T}^+(M)$. We use $\varphi(\tau)$ or $\tau[x_1/\tau_1,\dots,x_n/\tau_n]$ for the result of replacing all occurrences of variables x_k in term τ by term τ_k, $k = 1,\dots,n$, where these changes are made simultaneously.

Definition 2.2: A binary relation $\approx$ on $\mathcal{T}(M)$ or $\mathcal{T}^+(M)$ is a *congruence* provided that,

1) $\approx$ is an equivalence relation,
2) if $\text{ARITY}(\theta) = m$ and $\tau_k \approx \tau_k'$, $k = 1,\dots,m$, then $\theta\tau_1\dots\tau_m \approx \theta\tau_1'\dots\tau_m'$.

An *equation* e is a string of the form $\tau = \tau'$, where τ, τ' are in $\mathcal{T}^+(M)$. We use the symbol E for a set of equations. We will be dealing with models for sets of equations, i.e., algebras. We consider each equation e as a sentence of first-order predicate calculus (with equality), where all the variables from V are *universally quantified.*

Definition 2.3: An *algebra* $\mathcal{A}$ is a pair (A,F), where A is a nonempty set and F is a set of functions. Each f in F is a function from A^n to A, for some n in $\mathcal{N}$ which we denote as *type(f).*

Example 2.1:

(a) A *semigroup* $(A,\{+\})$ is an algebra with one binary operator which is *associative*, i.e., for all x,y,z in A we have $(x+y)+z = x+(y+z)$. An example of a semigroup is the set of functions from $\mathcal{N}$ to $\mathcal{N}$, together with the composition operation. In semigroups we use ab instead of a+b. We also omit parentheses, without ambiguity.

(b) $\mathcal{A}_M$ is an algebra with $A=\mathcal{T}(M)$. For each θ in O we define a function θ in F with $type(\theta)=$ ARITY(θ); here we use the same symbol for the syntactic object θ and its interpretation. The function θ maps terms $\tau_1,\ldots,\tau_m$ from $\mathcal{T}(M)$ to the term $\theta\tau_1\ldots\tau_m$, (i.e., $\theta(\tau_1,\ldots,\tau_m)=\theta\tau_1\ldots\tau_m$). This algebra is referred to as the *free algebra* on M. From this example it is clear that we can without ambiguity use both $\theta\tau_1\ldots\tau_m$ and $\theta(\tau_1,\ldots,\tau_m)$ to denote the same term.

(c) Let $\approx$ be a congruence on $\mathcal{T}(M)$. Condition (2) of Definition 2.2 guarantees that the operations in O are well-defined on $\approx$-equivalence (or congruence) classes. Thus we can form a *quotient* algebra $\mathcal{T}(M)/\approx$ with domain $\{[\tau] \mid \tau$ in $\mathcal{T}(M)$, $[\tau]$ is the $\approx$-congruence class of $\tau\}$ and with functions corresponding to the operators in O.

(d) Observations similar to (b),(c) can be made for the set of terms $\mathcal{T}^+(M)$.

Implication: Let e be an equation and $\mathcal{A}$ an algebra. $\mathcal{A}$ *satisfies* e, or is a *model* for e, if e becomes true when its operators and nonvariable generators are interpreted as the functions of $\mathcal{A}$ and its variables take *any* values in the domain of $\mathcal{A}$. The class of all algebras which are models for a set of equations E is called a *variety* or an *equational class.* We say that E implies e ($E\models e$) if the equation e is true in every model of E.

Definition 2.4: An *equational theory* is a set of equalities E (of terms over $\mathcal{T}^+(M)$), closed under *implication.*

See [41] for a survey of equational theories.

We write $E\vdash e$, if there exists a finite proof of e starting from E and using only the following five rules:

$\tau=\tau$,

from $\tau_1=\tau_2$ *deduce* $\tau_2=\tau_1$,

from $\tau_1=\tau_2$ *and* $\tau_2=\tau_3$ *deduce* $\tau_1=\tau_3$,

from $\tau_k=\tau_k'$, $k=1,\ldots,m$, *deduce* $\theta\tau_1\ldots\tau_m=\theta\tau_1'\ldots\tau_m'$ (ARITY$(\theta)=m$),

from $\tau_1=\tau_2$ *deduce* $\varphi(\tau_1)=\varphi(\tau_2)$ (φ is any substitution).

Proposition 2.2: [14, 41] $E\models\tau=\tau'$ iff $E\vdash\tau=\tau'$. ∎

Proofs in the above system can also be viewed as *reduction sequences*, as follows [41]: Whenever $E\models\tau=\tau'$, there is a sequence of terms $\tau_0,\ldots,\tau_m$ such that τ_0 is τ, τ_m is τ', and for $k=0,\ldots,m-1$ the term τ_{k+1} is obtained from τ_k by rewriting a subterm $\varphi(\sigma_1)$ as $\varphi(\sigma_2)$, where $\sigma_1=\sigma_2$ ($\sigma_2=\sigma_1$) is an equation in E and φ is a substitution.

Let Γ be a set of equations over terms in $\mathcal{T}(M)$ (i.e., containing no variables). Consider the equational theory consisting of all equations $\tau=\tau'$ such that $\Gamma\models\tau=\tau'$. By Proposition 2.2 this theory induces a congruence $=_\Gamma$ on $\mathcal{T}(M)$, where $\tau=_\Gamma\tau'$ iff $\Gamma\models\tau=\tau'$. From example (c) above we see that this congruence naturally defines an algebra $\mathcal{T}(M)/=_\Gamma$. If Γ is a finite set, $\mathcal{T}(M)/=_\Gamma$ is known as a *finitely presented algebra* [47].

2.2 Functional and Inclusion Dependencies as Equations

Let Σ be a set of FD's and IND's over a database scheme D and σ an FD or IND. We will transform Σ into two sets of equations E_Σ and $\mathcal{E}_\Sigma$. We will show that $\Sigma\models\sigma$ iff $E_\Sigma\models E_\tau$ iff $\mathcal{E}_\Sigma\models\mathcal{E}_\tau$, for some sets of equations $E_\tau,\mathcal{E}_\tau$ whose form depends on Σ and σ. We assume that D only contains one relation scheme. This simplifies notation, and there is no loss of generality.

Transformation: From the dependencies in Σ construct the following sets of symbols:

$M_f=\{f_k \mid$ for each FD with n attribute left-hand side include one operator f_k of ARITY n$\}$,
$M_i=\{i_k \mid$ for each IND include one operator i_k of ARITY 1$\}$,
$M_a=\{a_k \mid$ for each attribute A_k include one operator a_k of ARITY 1$\}$,
$M_\alpha=\{\alpha_k \mid$ for each attribute A_k include one generator $\alpha_k\}$.

Now let $M=M_f\cup M_i\cup M_a\cup M_\alpha$ and $V=\{x,x_1,x_2,...\}$ be a set of variables. $\mathcal{T}^+(M_f)$ ($\mathcal{T}^+(M_i)$) are the sets of terms constructed using operators in M_f (M_i) and generators in V.

The set E_Σ consists of the following equations (presented in string notation):

1) one equation for each FD $A_1...A_n\rightarrow A$: $f_k a_1 x ... a_n x = ax$,

2) m equations for each IND $B_1...B_m\subseteq A_1...A_m$: $a_1 i_k x = b_1 x$ and ... and $a_m i_k x = b_m x$.

The set $\mathcal{E}_\Sigma$ consists of the following equations:

3) one equation for each FD $A_1...A_n\rightarrow A$: $f_k\alpha_1...\alpha_n=\alpha$,

4) m equations for each IND $B_1...B_m\subseteq A_1...A_m$: $i_k\alpha_1=\beta_1$ and ... and $i_k\alpha_m=\beta_m$,

5) for each pair of symbols f_p in M_f and i_q in M_i the equation $f_p i_q x_1 ... i_q x_n = i_q f_p x_1 ... x_n$ (ARITY(f_p)$=$n).

Note that in $\mathcal{E}_\Sigma$ only equations (5) contain variables. Equations (5) are *commutativity* conditions between the f_k's and the i_k's. We now present Theorem 2.1, which is central to our analysis.

Theorem 2.1: In each of the following three cases, (i),(ii),(iii) are equivalent.

$\equiv$ Case:

i) $\Sigma \models A \equiv B$

ii) $E_\Sigma \models ax = bx$

iii) $\mathcal{E}_\Sigma \models \alpha = \beta$.

FD Case:

i) $\Sigma \models A_1 ... A_n \rightarrow A$

ii) $E_\Sigma \models \tau[x_1/a_1x, ..., x_n/a_nx] = ax$, for some τ in $\mathcal{T}^+(M_f)$

iii) $\mathcal{E}_\Sigma \models \tau[x_1/\alpha_1, ..., x_n/\alpha_n] = \alpha$, for some τ in $\mathcal{T}^+(M_f)$.

IND Case:

i) $\Sigma \models B_1 ... B_m \subseteq A_1 ... A_m$

ii) $E_\Sigma \models a_1\tau = b_1x$ and ... and $a_m\tau = b_mx$, for some τ in $\mathcal{T}^+(M_i)$

iii) $\mathcal{E}_\Sigma \models \tau[x/\alpha_1] = \beta_1$ and ... and $\tau[x/\alpha_m] = \beta_m$, for some τ in $\mathcal{T}^+(M_i)$.

Proof: Observe that the $\equiv$ Case follows immediately from the IND Case, by writing $A \equiv B$ as $AB \subseteq AA$. We use E_τ ($\mathcal{E}_\tau$) to denote the set of equations corresponding to term τ in (ii),(iii).

(ii)$\Rightarrow$(i):

Suppose $E_\Sigma \models E_\tau$, and let relation r satisfy Σ; we will show that r satisfies σ (σ is $A_1 ... A_n \rightarrow A$ in the FD Case and $B_1 ... B_m \subseteq A_1 ... A_m$ in the IND Case). Relation r is, by definition, nonempty and its entries can be assumed w.l.o.g. to be *positive* integers. Let the tuples of r be $t_1, t_2, ...$ (it could contain a countably infinite number of tuples).

For each attribute A in $\mathcal{U}$, define a function $a(.): \mathcal{N} \rightarrow \mathcal{N}$ ($\mathcal{N}$ is the set of *nonnegative* integers) so that, if ν is the index of a tuple in r, then $a(\nu)$ is the entry in tuple t_ν at attribute A; else $a(\nu)$ is 0.

For each FD $C_1 ... C_j \rightarrow C$ in Σ, define a function $f(...): \mathcal{N}^j \rightarrow \mathcal{N}$ so that, if $a_k = t_\nu[C_k]$, $k = 1, ..., j$, then $f(a_1, ..., a_j) = t_\nu[C]$; else $f(a_1, ..., a_j)$ is 0. This is a well-defined function, since r satisfies $C_1 ... C_j \rightarrow C$.

For each IND $D_1 ... D_j \subseteq C_1 ... C_j$ in Σ, define a function $i(.): \mathcal{N} \rightarrow \mathcal{N}$ so that, if ν is the index of a tuple in r, then $i(\nu) = \nu'$, where ν' is the index of the first tuple in r where $t_\nu[D_1 ... D_j] = t_{\nu'}[C_1 ... C_j]$; else $i(\nu)$ is 0. This is also a well-defined function, since r satisfies $D_1 ... D_j \subseteq C_1 ... C_j$.

We have constructed an algebra with domain $\mathcal{N}$ and functions $a(.), ..., f(...), ..., i(.), ...$, which, as is easy to verify, is a model for E_Σ. Let σ be an IND. By interpreting each symbol in τ as an $i(.)$, we see that, when ν is a tuple number, $\tau[x/\nu]$ is another tuple number. Since $E_\Sigma \models E_\tau$, we must have $a_k(\tau[x/\nu]) = b_k(x)$, $k=1,\ldots,m$, which means that r satisfies σ. The case of an FD is similar.

(iii)$\Rightarrow$(ii):

Suppose $\mathcal{E}_\Sigma \models \mathcal{E}_\tau$, and let $\mathcal{M}$ be a model of E_Σ; we will show that $\mathcal{M}$ satisfies E_τ. From $\mathcal{M}$ we construct a model $\mathcal{A}(\mathcal{M})$ for $\mathcal{E}_\Sigma$. The domain of $\mathcal{A}(\mathcal{M})$ is the set of all functions from $\mathcal{M}$ to $\mathcal{M}$, i.e., $\mathcal{M} \rightarrow \mathcal{M}$.

In $\mathcal{A}(\mathcal{M})$ the interpretation of α is the function $a(x)$, which is the interpretation of a(.) in $\mathcal{M}$. The interpretation of i(.) is the function $\lambda h.h(i(x))$, where $i(x)$ is the interpretation of i(.) in $\mathcal{M}$. This is a function from $\mathcal{M} \rightarrow \mathcal{M}$ to $\mathcal{M} \rightarrow \mathcal{M}$. The interpretation of f(...) is the function $\lambda h_1 \ldots h_n . f(h_1(x),\ldots,h_n(x))$, where $f(x_1,\ldots,x_n)$ is the interpretation of f(...) in $\mathcal{M}$. This is a function from $(\mathcal{M} \rightarrow \mathcal{M})^n$ to $\mathcal{M} \rightarrow \mathcal{M}$.

It is straightforward to check that equations (3),(4) hold in $\mathcal{A}(\mathcal{M})$, because $\mathcal{M}$ is a model for E_Σ. Also equations (5) hold in $\mathcal{A}(\mathcal{M})$: For example, if $n=1$ the interpretation of $f(i(h))$ in $\mathcal{A}(\mathcal{M})$ is $f(h(i(x)))$, which is also the interpretation of $i(f(h))$ (h is any element of $\mathcal{M} \rightarrow \mathcal{M}$). Thus $\mathcal{A}(\mathcal{M})$ is a model for $\mathcal{E}_\Sigma$. Since $\mathcal{E}_\Sigma \models \mathcal{E}_\tau$, $\mathcal{A}(\mathcal{M})$ satisfies $\mathcal{E}_\tau$. From this it follows that $\mathcal{M}$ satisfies E_τ.

(i)$\Rightarrow$(iii):

IND Case:

Consider a *chase proof* of $B_1 \ldots B_m \subseteq A_1 \ldots A_m$ from Σ. This chase starts from a single tuple t_1 and generates tuples $t_2,\ldots,t_\nu$, where $t_\nu[A_1 \ldots A_m] = t_1[B_1 \ldots B_m]$. Now a tuple can only be generated by applying an IND-rule on some previously generated tuple. Thus, we can assign (inductively) to each tuple t_p, $p=1,\ldots,\nu$, a term τ_p in $\mathcal{T}^+(M_i)$, as follows:

1. $\tau_1 = x$.
2. If t_p was generated from t_q, $q<p$, by applying the IND-rule corresponding to some IND i in Σ, then $\tau_p = \tau_q[x/ix]$.

The term τ_p records the sequence of applications of IND-rules which produced t_p (starting from t_1).

We will show the following

<u>Claim:</u> For $1 \leq p,q \leq \nu$, C,D in $\mathcal{U}$, if $t_p[C] = t_q[D]$, then $\mathcal{E}_\Sigma \models \tau_p[x/\gamma] = \tau_q[x/\delta]$, where γ,δ are the symbols in M_α corresponding to C,D.

Clearly, the IND Case follows from the Claim: Since $t_\nu[A_1 \ldots A_m] = t_1[B_1 \ldots B_m]$, we have

$\mathcal{E}_\Sigma \models \tau_\nu[x/\alpha_k]=\beta_k$, $k=1,\ldots,m$.

Proof of Claim: Suppose the equality $t_p[C]=t_q[D]$ appears after exactly z steps of the chase. We argue by induction on z.

Basis: $z=0$. Then $p=q=1$, C is D, and the conclusion is straightforward.

Induction Step: Let $t_p[C]=\kappa$, $t_q[D]=\lambda$. The symbols κ,λ were equated by the chase. We distinguish three cases, according to how this happened.

a. κ is a freshly created symbol, identical to λ. This means t_p was created from $t_{p'}$, $p'<p$, using an IND $X_1C'X_2 \subseteq Y_1CY_2$ in Σ ($X_k,Y_k \subseteq \mathcal{U}$, $k=1,2$), and $t_{p'}[C']=t_q[D]$. By the induction hypothesis $\mathcal{E}_\Sigma \models \tau_{p'}[x/\gamma']=\tau_q[x/\delta]$. Now $\tau_p=\tau_{p'}[x/ix]$, where i is the operator corresponding to $X_1C'X_2 \subseteq Y_1CY_2$, and also $i\gamma=\gamma'$ is in $\mathcal{E}_\Sigma$. Thus, $\mathcal{E}_\Sigma \models \tau_{p'}[x/i\gamma]=\tau_q[x/\delta]$, i.e. $\mathcal{E}_\Sigma \models \tau_p[x/\gamma]=\tau_q[x/\delta]$.

b. κ was equated to λ in order to satisfy some FD $C_1 \ldots C_j \rightarrow C$ in Σ. This means $t_p[C_1 \ldots C_j]=t_q[C_1 \ldots C_j]$, and D is C. By the induction hypothesis $\mathcal{E}_\Sigma \models \tau_p[x/\gamma_k]=\tau_q[x/\gamma_k]$, $k=1,\ldots,j$. Also, we have in $\mathcal{E}_\Sigma$ the equation $f\gamma_1 \ldots \gamma_j=\gamma$, where f is the operator in M_i corresponding to the FD $C_1 \ldots C_j \rightarrow C$. Thus, $\mathcal{E}_\Sigma$ implies $f\tau_p[x/\gamma_1] \ldots \tau_p[x/\gamma_j]=\tau_p[x/f\gamma_1 \ldots \gamma_j]$ (by the commutativity conditions (5)) $=\tau_p[x/\gamma]$. Similarly $\mathcal{E}_\Sigma$ implies $f\tau_q[x/\gamma_1] \ldots \tau_q[x/\gamma_j]=\tau_q[x/f\gamma_1 \ldots \gamma_j]=\tau_q[x/\gamma]$, so $\mathcal{E}_\Sigma \models \tau_p[x/\gamma]=\tau_q[x/\gamma]$.

c. There are tuples $t_{p'},t_{q'}$, $p' \leq p$, $q' \leq q$, and C',D' in $\mathcal{U}$ such that $t_{p'}[C']=\kappa$, $t_{q'}[D']=\lambda$, and $t_{p'}[C']$ was equated to $t_{q'}[D']$ at some earlier step. Then by the induction hypothesis $\mathcal{E}_\Sigma$ implies $\tau_p[x/\gamma]=\tau_{p'}[x/\gamma']$, $\tau_q[x/\delta]=\tau_{q'}[x/\delta']$, and $\tau_{p'}[x/\gamma']=\tau_{q'}[x/\delta']$. Thus, $\mathcal{E}_\Sigma \models \tau_p[x/\gamma]=\tau_q[x/\delta]$.

FD Case:

Consider, as before, a *chase proof* of $A_1 \ldots A_n \rightarrow A$ from Σ. This chase starts from two tuples t_1,t_2 and generates tuples $t_3,\ldots,t_\nu$; finally, $t_1[A]=t_2[A]$. Again a tuple can only be generated by applying an IND-rule on some previously generated tuple, so we can assign (inductively) to each tuple t_p, $p=1,\ldots,\nu$, a term τ_p in $\mathcal{T}^+(M_i)$, as follows:

1. $\tau_1=x_1$, $\tau_2=x_2$.
2. If t_p was generated from t_q, $q<p$, by applying the IND-rule corresponding to some IND i in Σ, then $\tau_p=\tau_q[x_1/ix_1, x_2/ix_2]$.

Observe that τ_p also records the *tuple* (t_1 or t_2) which produced t_p (apart from the sequence of applications of IND-rules).

We will show the following

Claim: For $1 \leq p,q \leq \nu$, C,D in $\mathcal{U}$, if $t_p[C]=t_q[D]$, then $\mathcal{E}_\Sigma \models \tau_p[x_k/\gamma]=\tau_q[x_k/\delta]$ (k=1,2). If, additionally, t_p is produced from t_1 and t_q is produced from t_2, then $\mathcal{E}_\Sigma$ implies $\tau_p[x_1/\gamma]=\tau_q[x_2/\delta]=\tau[x_1/\alpha_1,...,x_n/\alpha_n]$, for some τ in $\mathcal{T}^+(M_f)$.

Clearly, the IND Case follows from the second part of the Claim: Since $t_1[A]=t_2[A]$, $\mathcal{E}_\Sigma \models \alpha=\tau[x_1/\alpha_1,...,x_n/\alpha_n]$, for some τ in $\mathcal{T}^+(M_f)$.

Proof of Claim: Suppose the equality $t_p[C]=t_q[D]$ appears after exactly z steps of the chase. We argue by induction on z.

Basis: z=0. Then p=q=1, C and D are both some A_k, $1 \leq k \leq n$, and the conclusion is straightforward.

Induction Step: Let $t_p[C]=\kappa$, $t_q[D]=\lambda$. The symbols κ,λ were equated by the chase. We distinguish three cases, according to how this happened.

a. κ is a freshly created symbol, identical to λ. This means t_p was created from $t_{p'}$, $p'<p$, using an IND $X_1C'X_2 \subseteq Y_1CY_2$ in Σ ($X_k,Y_k \subseteq \mathcal{U}$, k=1,2), and $t_{p'}[C']=t_q[D]$. For the first part of the Claim, we argue exactly as in the IND Case. For the second part, note that if t_p is produced from t_1 then so is $t_{p'}$. Therefore we can use the induction hypothesis on $t_{p'},t_q$.

b. κ was equated to λ in order to satisfy some FD $C_1...C_j \rightarrow C$ in Σ. This means $t_p[C_1...C_j]=t_q[C_1...C_j]$, and D is C. The argument for the first part proceeds exactly as in the IND Case. For the second part, note that since $\mathcal{E}_\Sigma$ implies $\tau_p[x_1/\gamma_k]=\tau_k[x_1/\alpha_1,...,x_n/\alpha_n]$, k=1,...,j
(by the induction hypothesis), we have that $\mathcal{E}_\Sigma$ implies
$\tau_p[x_1/\gamma]=\tau_p[x_1/f\gamma_1...\gamma_j]=f\tau_p[x_1/\gamma_1]...\tau_p[x_1/\gamma_j]=f\tau_1[x_1/\alpha_1,...,x_n/\alpha_n]...\tau_j[x_1/\alpha_1...x_n/\alpha_n]=$
$=\tau[x_1/\alpha_1,...,x_n/\alpha_n]$, where τ is $f\tau_1...\tau_j$. Similarly, $\mathcal{E}_\Sigma$ implies
$\tau_q[x_1/\gamma]=\tau[x_1/\alpha_1,...,x_n/\alpha_n]$.

c. There are tuples $t_{p'},t_{q'}$, $p' \leq p$, $q' \leq q$, and C',D' in $\mathcal{U}$ such that $t_{p'}[C']=\kappa$, $t_{q'}[D']=\lambda$, and $t_{p'}[C']$ was equated to $t_{q'}[D']$ at some earlier step. The argument for the first part proceeds exactly as in the IND Case. For the second part, if $t_{p'}$ was produced from t_2, use the induction hypothesis on $t_p,t_{p'}$;

else, if $t_{q'}$ was produced from t_2, use the induction hypothesis on $t_{p'}$, $t_{q'}$; else, use the induction hypothesis on $t_{q'}$, t_q.

This concludes the proof of (i)$\Rightarrow$(iii), so we are done. ∎

We remark here that the (i)$\Rightarrow$(iii) direction can also be proved by showing that each of the rules of [54] (see Subsection 2.1.1) can be simulated using the equational reasoning of Proposition 2.2. We illustrate this simulation with an example:
From $A \rightarrow B$ and $CD \subseteq AB$ the *pullback* rule of [54] derives $C \rightarrow D$. In equational language $f\alpha = \beta$, $i\alpha = \gamma$, $i\beta = \delta$ and $fix = ifx$ imply $f\gamma = fi\alpha = if\alpha = i\beta = \delta$.

Corollary 2.1: Let Σ be a set of FD's and σ an FD. The implication problem $\Sigma \models \sigma$ is equivalent to a *generator problem for a finitely presented algebra* [47].

Proof: $\mathcal{E}_\Sigma$ is now a finite set of equations with no variables. If $\approx$ is the congruence induced by $\mathcal{E}_\Sigma$ on $\mathcal{T}(M)$ then $\mathcal{T}(M)/\approx$ is a finitely presented algebra. The equational implication in Theorem 2.1 is known, in this case, as a generator problem for the finitely presented algebra $\mathcal{T}(M)/\approx$. ∎

Using Corollary 2.1, one can observe that the linear time algorithm of [6] for implication of FD's can be derived in a straightforward way from the algorithm of [47] for the generator problem.

Corollary 2.2: Let Σ be a set of FD's. The implication problem $\Sigma \models A \equiv B$ is a *uniform word problem for a finitely presented algebra* [47]. ∎

If the given FD's are all *unary*, then the equational inferences in the theory E_Σ can be thought of as inferences in *semigroups*. This gives yet another transformation of (unary) FD's and IND's into equations:

Semigroup Transformation: Let Σ be a set of IND's and u-FD's. Construct a set of symbols M_s from M as follows: for each $f_k(.)$ in M_f add one generator f_k in M_s; for each $i_k(.)$ in M_i add one generator i_k in M_s; for each $a_k(.)$ in M_a add one generator a_k in M_s; add one binary operator + in M_s.

The set of equations E_S consists of the associative axiom for + and the following word (string) equations (we omit + and parentheses):
1) one equation for each u-FD $A_1 \rightarrow A$: $f_k a_1 = a$,
2) m equations for each IND $B_1 ... B_m \subseteq A_1 ... A_m$: $a_1 i_k = b_1$ and ... and $a_m i_k = b_m$.

Corollary 2.3: Let Σ be a set of u-FD's and IND's:

$\Sigma \models A \equiv B$ iff $E_S \models a = b$.

$\Sigma \models A_1 \rightarrow A$ iff $E_S \models wa_1 = a$, for some string w in M_S^*.

$\Sigma \models B_1 ... B_m \subseteq A_1 ... A_m$ iff $E_S \models a_1 w = b_1$ and ... and $a_m w = b_m$, for some string w in M_S^*. ∎

Note that the first case is an instance of the *uniform word problem for semigroups.* The other two cases are known as E_S-*unification* problems [41].

2.3 A Proof Procedure for FD's and IND's

We will now describe a proof procedure for FD and IND implication, which exploits the special structure of the equational theory $\mathcal{E}_\Sigma$ (Theorem 2.1). Whenever a dependency σ cannot be proved from a set of dependencies Σ, the procedure provides us (in a natural way) with an *algebra* which satisfies $\mathcal{E}_\Sigma$ but violates any equation that could correspond to σ. Thus, by Theorem 2.1 we have that Σ does not imply σ, i.e. the procedure is complete for FD and IND implication.

The Proof Procedure ***G*****:**

Given a set Σ of FD's and IND's construct their graphical representation G_Σ defined in Subsection 2.1.1. Each attribute name in Σ is associated with one of the nodes of G_Σ.

Rules: Apply some finite sequence of the graph manipulation rules 1,2,3 and 4 of Figure 2-2 on G_Σ. Rules 1 and 2 introduce new unnamed nodes. Rules 3 and 4 identify two existing nodes; the node resulting from this identification is associated with the union of the two sets of attribute names that were associated with each of the identified nodes. Note that rules 1,2 w.l.o.g. need be applied at most once to every left-hand side configuration.

Let G be the resulting graph. Associate a unique new name with every unnamed node in G.

We say that $\Sigma \vdash_G \sigma$ when:

σ is $A \equiv B$: A,B are associated with the same node.

σ is an FD $A_1 ... A_n \rightarrow A$: The node associated with A gets marked by the following algorithm: We mark the nodes associated with $A_1,...,A_n$; whenever nodes $v_1,...,v_j$ are marked and there is a group of red arcs $(v_1,v),...,(v_j,v)$ labeled by the name f of some FD in Σ, we mark v.

σ is an IND $B_1 ... B_m \subseteq A_1 ... A_m$: For k = 1,...,m there is a black directed path from A_k to B_k; moreover, all these paths have the same sequence of labels.

Note that, as expected, the $A \equiv B$ Case is a specialization of the IND Case: if $\Sigma \vdash_G AB \subseteq AA$, then A,B can be identified using Rule 3.

Theorem 2.2: $\Sigma \models \sigma$ iff $\Sigma \vdash_G \sigma$.

Proof:

($\Leftarrow$): Rules 3,4 are obviously sound. Rules 1 and 2 are sound in the sense of the *attribute introduction* rule of [54] (see Subsection 2.1.1), which we illustrate as rule 5 of Figure 2-2.

($\Rightarrow$): Let G be a (possibly infinite) graph obtained by closing G_Σ under Rules 1-4. We will construct from G a model $\mathcal{M}$ of $\mathcal{E}_\Sigma$.
The domain M of $\mathcal{M}$ is the set V of nodes of G, together with a special node $\perp$. The generator $\boldsymbol{\alpha}_k$ is interpreted as the node associated with A_k.
An operator i in $\mathcal{E}_\Sigma$ (corresponding to some IND in Σ) is interpreted as a function $i{:}M \rightarrow M$ as follows: if v is in V and has an outgoing arc (v,w) labeled i, then $i(v)=w$; else $i(v)=\perp$. This function is well-defined, because G is closed with respect to Rule 3.
An operator f of ARITY j in $\mathcal{E}_\Sigma$ (corresponding to some FD in Σ) is interpreted as a function $f{:}M^j \rightarrow M$ as follows: if $v_1,...,v_j$ are in V and there is a group of red arcs $(v_1,v),...,(v_j,v)$ labeled f, then $f(v_1,...,v_j)=v$; else $f(v_1,...,v_j)=\perp$. This function is well-defined, because G is closed with respect to Rule 4.
One can check that $\mathcal{M}$ satisfies the commutativity conditions (5) of $\mathcal{E}_\Sigma$ (because G is closed with respect to Rules 1,2) and $\mathcal{M}$ satisfies equations (3),(4) of $\mathcal{E}_\Sigma$ (because G was constructed starting from G_Σ). Thus, $\mathcal{M}$ is a model of $\mathcal{E}_\Sigma$.
Now suppose we cannot prove σ from Σ. If σ is an FD $A_1...A_n \rightarrow A$, then clearly there is no τ in $\mathcal{T}^+(M_f)$ such that $\tau[x_1/\alpha_1,...,x_n/\alpha_n]=\alpha$ in $\mathcal{M}$. Thus, $\mathcal{M}$ is a *counterexample* to condition (iii) of Theorem 2.1 and therefore Σ does not imply σ. Similarly if σ is an IND. ∎

2.4 Computations as Inferences

It has been known, since at least Post's proof of the unsolvability of the word problem for Thue systems [55, 50], that arbitrary computations can be simulated by inferences in semigroups. Using Corollary 2.3, we show that we can simulate computations by inferences of IND's and *unary* FD's. We thus obtain lower bounds on the complexity of the implication problem for IND's and FD's.

We first describe our machine model: A *deterministic two-stack* machine M is a 5-tuple $(Q,\Pi,q_{start},h,\delta)$, where Q is a finite set of *states*, Π is a finite set of *symbols* $(Q\cap\Pi=\emptyset)$, $q_{start}\in Q$ is the *start state*, $h\in Q$ is the *halt state*, and δ is the *transition function*. Each *move* of M falls into one of the following two types:

1. $\delta(q,\alpha)=(p,\text{POP}_1)$: This means that, if M is in state q and $\alpha\in\Pi$ is the top symbol of STACK_1, then on the next step M goes to state p and *pops* STACK_1.

2. $\delta(q)=(p,\text{PUSH}_1(\beta))$: If M is in state q, then on the next step M goes to state p and *pushes* $\beta\in\Pi$ on STACK_1.

Of course, analogous instructions can manipulate STACK_2.
An *instantaneous description* (ID) of M is a string $x_1...x_nqy_m...y_1$, where $q\in Q$, $x_i,y_i\in\Pi$: the string $x_1...x_n$ is the contents of STACK_1 (the top symbol is x_n); the string $y_m...y_1$ is the contents of STACK_2 (the top symbol is y_m). The relation $w_1\Rightarrow_M w_2$ (ID w_1 *yields* ID w_2 via one step of M) is defined in the standard way [50, 40]. $\Rightarrow^*_M$ is the *reflexive, transitive closure* of $\Rightarrow_M$.

Let us now define a set S of *word equations* (over generators $Q\cup\Pi$) which capture the computation of M:

1. If $\delta(q,\alpha)=(p,\text{POP}_1)$, then $\alpha q=p$ is in S.
   If $\delta(q,\alpha)=(p,\text{POP}_2)$, then $q\alpha=p$ is in S.

2. If $\delta(q)=(p,\text{PUSH}_1(\beta))$, then $q=\beta p$ is in S.
   If $\delta(q)=(p,\text{PUSH}_2(\beta))$, then $q=p\beta$ is in S.

We write $u=_S v$ iff $S\models u=v$. By a standard argument, based on the fact that M is *deterministic* [55, 50], we have

Lemma 2.1: $q_{start}\Rightarrow^*_M h$ iff $q_{start}=_S h$. ∎

To prove our first lower bound, we transform S into another set of equations T which looks like the sets obtained (as in Corollary 2.3) from IND's and u-FD's. The set of generators is now $Q\cup\{A_\alpha,B_\alpha,f_\alpha \mid \alpha\in\Pi\}\cup\{i_\alpha \mid \alpha\in\Pi\}\cup\{j_e \mid e\in S\}$.

1. If $q\alpha=p$ is in S, then $qi_\alpha=p$ is in T.

2. If $\alpha q=p$ is in S, then T contains the equations $q=A_\alpha j_e$, $f_\alpha A_\alpha=B_\alpha$, $B_\alpha j_e=p$, where e is $\alpha q=p$.

Lemma 2.2: $q_{start} =_S h$ iff $q_{start} =_T h$.

Proof: Given a word w over $Q \cup \Pi$ of the form $\alpha_1 ... \alpha_n q \beta_m ... \beta_1$, $q \in \Pi$, $\alpha_i, \beta_i \in \Pi$, define a corresponding word w' to be $f_{\alpha_1} ... f_{\alpha_n} q i_{\beta_m} ... i_{\beta_1}$. We claim that, if w_1, w_2 are words over $Q \cup \Pi$, then $w_1 =_S w_2$ iff $w'_1 =_T w'_2$. The Lemma follows from this claim.

To prove the "only if" direction of the claim, consider the equations in S that can be used to rewrite w_1 as w_2. If $q\alpha = p$ is in S, then $q i_\alpha =_T p$, since $q i_\alpha = p$ is in T. If $\alpha q = p$ is in S, then $f_\alpha q =_T p$, since $f_\alpha q =_T f_\alpha A_\alpha j_e =_T B_\alpha j_e =_T p$. The converse is also straightforward. ∎

Theorem 2.3: The implication problem for IND's and two u-FD's is undecidable.

Proof: Given a deterministic two-stack machine M, it is undecidable if $q_{start} \Rightarrow^*_M h$, even if $|\Pi| = 2$ [53, 40]. By Lemmas 2.1 and 2.2, $q_{start} \Rightarrow^*_M h$ iff $q_{start} =_T h$. By Corollary 2.3, $q_{start} =_T h$ iff $\Sigma \models Q_{start} \equiv H$, where Σ is the set of IND's and FD's which gives rise to T. But now observe that Σ only contains FD's of the form $A_\alpha \rightarrow B_\alpha$, $\alpha \in \Pi$. Since $|\Pi| = 2$, Σ only contains two unary FD's. ∎

Undecidability of the implication problem for IND's and FD's has already been proved [54, 19]. By way of comparison, these reductions use arbitrarily many IND's of the form $D_1 D_2 \subseteq C_1 C_2$ and arbitrarily many u-FD's, while our reduction uses arbitrarily many IND's and only two u-FD's.

To prove our second lower bound, we consider computations of a deterministic two-stack machine M where one of the two stacks has *bounded size*. Let us write $w_1 \Rightarrow^s_M w_2$ iff ID w_2 follows from ID w_1 by a computation of M during which STACK_2 contains at most s symbols.

Let S be the set of word equations described before: this time we transform S into a set T^s of equations which can be obtained (as in Corollary 2.3) from *acyclic* IND's and u-FD's. The set of generators now is $Q^0 \cup ... \cup Q^s \cup \{A_\alpha, B_\alpha, f_\alpha \mid \alpha \in \Pi\} \cup \{i_{\alpha,k} \mid \alpha \in \Pi, k = 1,...,s\} \cup$
$\cup \{j_{e,k} \mid e \in S, k = 0,...,s\}$, where $Q^k = \{q^k \mid q \in Q\}$, $k = 0,...,s$.

1. If $q\alpha = p$ is in S, then $q^{k+1} i_{\alpha,k+1} = p^k$ is in T^s, $k = 0,..., s-1$.
2. If $\alpha q = p$ is in S, then T^s contains the equations $q^k = A_\alpha j_{e,k}$, $f_\alpha A_\alpha = B_\alpha$, $B_\alpha j_{e,k} = p^k$, $k = 0,...,s$, where e is $\alpha q = p$.

It is not hard to see that T^s can be taken to represent a set Σ^s of acyclic IND's and u-FD's: the relation names are $R[A_\alpha B_\alpha \mid \alpha \in \Pi]$, $R^k[Q^k]$, $k = 0,...,s$. It is also easy to see the following

Lemma 2.3: $q_{start} \Rightarrow^{s}_{M} h$ iff $q^0_{start} =_T^s h^0$, iff $\Sigma^s \models R^0$: $Q^0_{start} \equiv H^0$. ∎

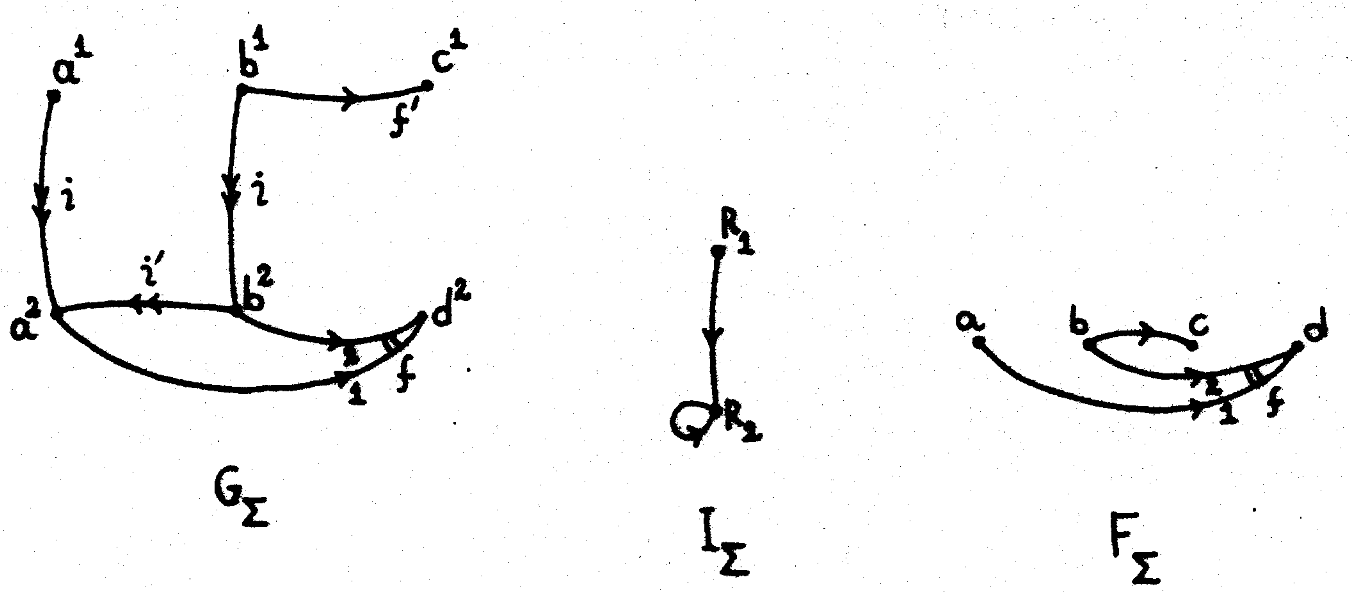
Theorem 2.4: There are constants $c_1, c_2 > 0$ such that the implication problem for acyclic IND's and FD's can be solved in time c_1^n but not in time $c_2^{\sqrt{n/\log n}}$.

Proof: Since the IND's are acyclic, the *chase* gives us a decision procedure, running in exponential time.

To prove the lower bound, let L be any language in DTIME(c^n), $c>0$. We will show that L is polynomial-time reducible to the implication problem for acyclic IND's and u-FD's.

Let M be a deterministic n-AuxiliaryPushdownAutomaton accepting L [40]. Given string x, we construct a deterministic two-stack machine M_x which first puts x on STACK$_2$ and then simulates M. This simulation is done as follows: if M is in state q, its auxiliary storage contains $\alpha_1 \ldots \alpha_n \alpha w$ (α is the symbol scanned) and its stack contains $u\beta$ (β is the top symbol), then the ID of M_x is $u\beta\alpha_{1,\beta} \ldots \alpha_{n,\beta} q\alpha w$. It is not hard to see how M_x can simulate a move of M. Thus, M accepts x iff M_x halts and STACK$_2$ always contains at most $|x|$ symbols, i.e. $x \in L$ iff $q_{start} \Rightarrow^{|x|}_{M_x} h$. Note also that the size of M_x, $|M_x|$, is $O(|x|)$.

Now let $\Sigma^{|x|}$ be the set of acyclic IND's and u-FD's corresponding to M_x. Using Lemma 2.3, $x \in L$ iff $\Sigma^{|x|} \models R^0$: $Q^0_{start} \equiv H^0$. To complete the proof, observe that $\Sigma^{|x|}$ can be computed from x in polynomial time, and that the size of $\Sigma^{|x|}$ is $O(|M_x|\,|x| \log|x|)$, i.e. $O(|x|^2 \log|x|)$. ∎



$$\mathcal{D} = \{ R_1[ABC],\ R_2[ABD] \}$$

$$\begin{aligned} \Sigma = \{\ & R_2 : AB \rightarrow D, \\ & R_1 : B \rightarrow C, \\ & R_2 : AB \subseteq R_1 : AB, \\ & R_2 : A \subseteq R_2 : B \ \} \end{aligned}$$

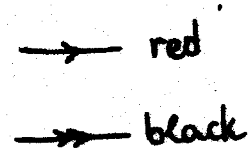


Figure 2-1: Graph notation for FD's and IND's

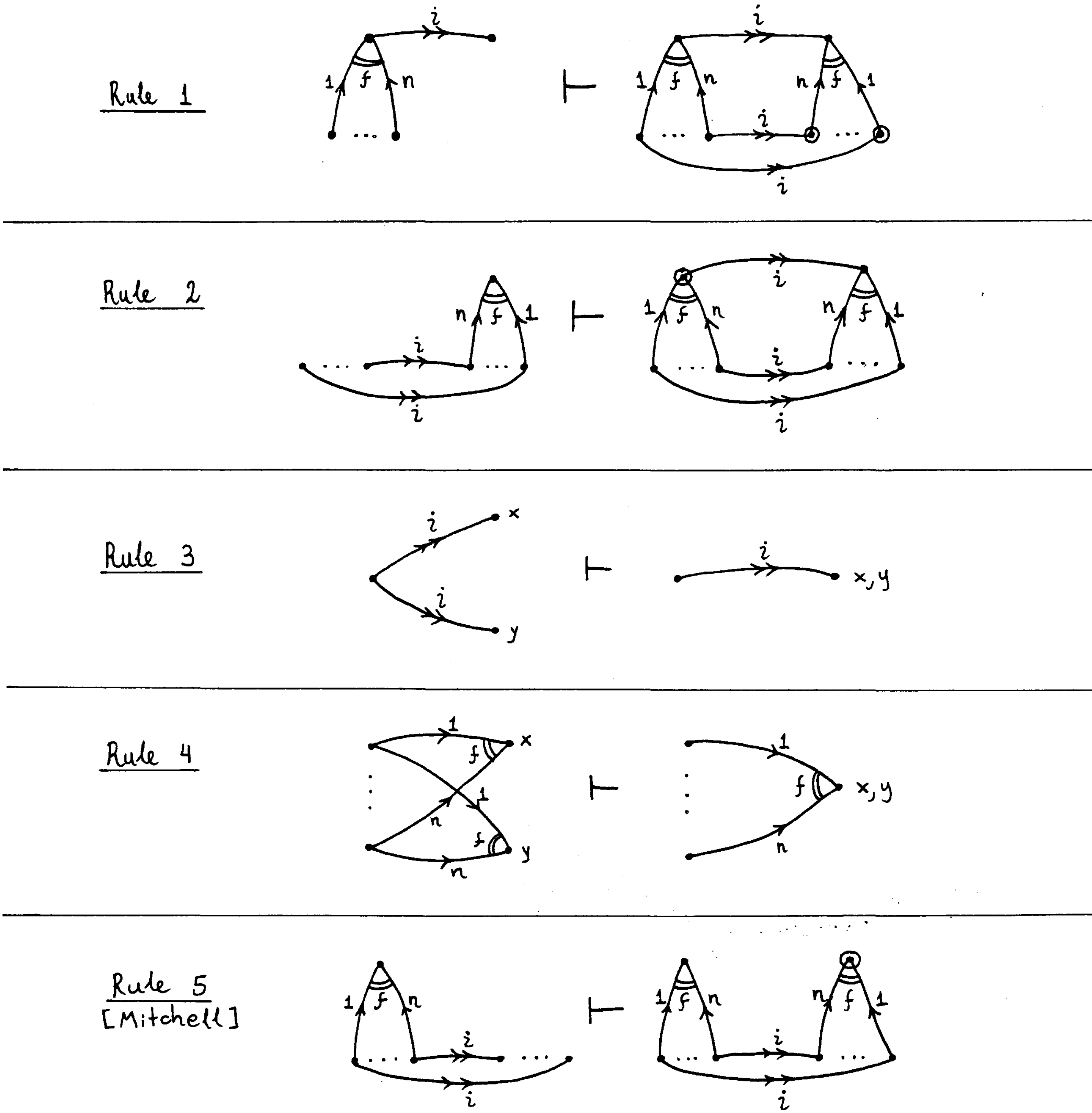


Figure 2-2: Graph rules for FD's and IND's ⊙ : new node

Chapter Three

Application to Typed IND's

In this Chapter we use the tools developed in Chapter 2 (Section 2.2) to study the particular implication problem for FD's and *typed* IND's. We first present a proof procedure for general FD and IND implication (Section 3.1), similar in spirit to the proof procedure of Theorem 2.2. By specializing this proof procedure to typed IND's, we obtain as a corollary that the implication problem for *acyclic* FD's and typed IND's is decidable (Section 3.2). In Section 3.3 we study the special case of inferring FD's under *pairwise consistency*. By analyzing derivations (in the proof procedure of Section 3.1), we show that the problem is undecidable. We also prove that there is no k-ary axiomatization for implication of FD's under pairwise consistency. As a by-product of our techniques, we obtain finite controllability of acyclic unary FD's under pairwise consistency.

3.1 Another Proof Procedure for FD's and IND's

We present in this Section a proof procedure for general FD and IND implication. This procedure is the main tool we use to study the implication problem for typed IND's and FD's. To prove completeness of the procedure, we show that it captures (in an indirect way) equational inferences in the theory E_{Σ} of Theorem 2.1.

Let Σ be a given set of FD's and IND's over a database scheme *D*, containing a single relation scheme $R[\mathcal{U}]$. We represent attribute $A_k \in \mathcal{U}$ by a *node* a_k. An FD $A_1...A_n \rightarrow A$ in Σ is represented as shown in Figure 3-1 by introducing a node $fa_1...a_n$ (we use a different function symbol f for each given FD), a group of directed arcs $(a_1, fa_1...a_n),...,(a_n, fa_1...a_n)$ labeled f and ordered from 1 to n, and an undirected arc $\langle fa_1...a_n, a\rangle$. The undirected arc is the only modification to our graph notation of Section 2.1.1. Its purpose is to represent the equation $fa_1x...a_nx = ax$.
An IND $B_1...B_m \subseteq A_1...A_m$ in Σ is represented (see Figure 3-1) by introducing directed arcs $(a_1,b_1),...,(a_m,b_m)$, labeled i (we use a different label for each given IND).

Let H_Σ be the mixed graph obtained from Σ as described above. Repeatedly apply Rules T(*transitivity*), $E_{1\text{-}2}$ (*equality*), $I_{1\text{-}3}$ (*introduction*) (see Figure 3-2) on H_Σ, in some arbitrary fixed order, until no more rules are applicable. As was the case with Rules 1,2 in Theorem 2.2, the introduction rules need only be applied once for each left-hand side configuration.

Let $H=(N_H,A_H,E_H)$ be the mixed graph obtained this way (N_H is a set of nodes, A_H is a set of labeled directed arcs on N_H, and E_H is a set of undirected arcs on N_H). Notice that each node of H is labeled $Fu_1...u_q$, where F is a term over the function symbols and $u_1,...,u_q$ are nodes representing attributes (by a slight abuse of notation, we write $Fu_1...u_q$ as a shorthand for $F[x_1/u_1,...,x_q/u_q]$). Moreover, every subterm of $Fu_1...u_q$ appears as a node of H.
By a *path labeled* τ, where τ is a term over the i's (and a variable x), we mean a mixed path where the sequence of labels corresponds to τ (see Figure 3-1). In the special case where τ is simply x, the path consists of undirected arcs.

The graph H fully captures implication of FD's and IND's from Σ, as we now show:

Theorem 3.1:
FD Case:
$\Sigma \models A_1...A_n \rightarrow A$ iff there is a node $Fa_1...a_n$ of H such that $\langle Fa_1...a_n, a\rangle \in E_H$.
IND Case:
$\Sigma \models B_1...B_m \subseteq A_1...A_m$ iff there is a path from a_k to b_k labeled τ, $k=1,...,m$, where τ is a term over the i's.

Proof: Let E_Σ be the set of equations of Theorem 2.1. Assume that the various names in E_Σ are consistent with the names in H.

($\Leftarrow$):
Claim:
(i) If $\langle Fu_1...u_p, Gv_1...v_q\rangle \in E_H$, where the u_k's, v_j's are nodes corresponding to attributes and F,G are terms over the f's, then $E_\Sigma \models Fu_1x...u_px = Gv_1x...v_qx$.
(ii) If $(Fu_1...u_p, Gv_1...v_q)$ is a directed arc labeled i, then $E_\Sigma \models Fu_1ix...u_pix = Gv_1x...v_qx$.

Clearly, the "if" direction follows from the Claim, by Theorem 2.1.

Proof of Claim: We prove both (i) and (ii) by *simultaneous* induction on the number of applications of rules that created an (undirected) arc of H.

Basis: No rules were applied. The conclusion is straightforward.

Induction Step: We have to check Rules T, $E_{1\text{-}2}$, $I_{1\text{-}3}$, each of which might have been applied at the last step.

Rules T, E_1 Straightforward.

Rule E_2 The undirected arc $\langle Fu_1...u_p, Gv_1...v_q\rangle$ was created from the undirected arc $\langle F'u_1'...u_{p'}', G'v_1'...v_{q'}'\rangle$, where $(F'u_1'...u_{p'}', Fu_1...u_p)$, $(G'v_1'...v_{q'}', Gv_1...v_q)$ are directed arcs labeled i. By the induction hypothesis, E_Σ implies $F'u_1'x...u_{p'}'x = G'v_1'x...v_{q'}'x$, $F'u_1'ix...u_{p'}'ix = Fu_1x...u_px$, $G'v_1'ix...v_{q'}'ix = Gv_1x...v_qx$. Thus, E_Σ implies $Fu_1x...u_px = Gv_1x...v_qx$.

Rule I_1 The undirected arcs $\langle F_1u_1...u_p, G_1v_1...v_q\rangle,...,\langle F_nu_1...u_p, G_nv_1...v_q\rangle$ create the undirected arc $\langle Fu_1...u_p, Gv_1...v_q\rangle$, where $F = fF_1...F_n$, $G = fG_1...G_n$. By the induction hypothesis, E_Σ implies $F_ku_1x...u_px = G_kv_1x...v_qx$, $k = 1,...,n$. Thus, E_Σ implies
$Fu_1x...u_px = fF_1u_1x...u_px \; ... \; F_nu_1x...u_px = fG_1v_1x...v_qx \; ... \; G_nv_1x...v_qx = Gv_1x...v_qx$.

Rule I_2 The directed arcs $(F_1u_1...u_p, G_1v_1...v_q),...,(F_nu_1...u_p, G_nv_1...v_q)$ (labeled i) create the directed arc $(Fu_1...u_p, Gv_1...v_q)$ (labeled i), where $F = fF_1...F_n$, $G = fG_1...G_n$. By the induction hypothesis, E_Σ implies $F_ku_1ix...u_pix = G_kv_1x...v_qx$, $k = 1,...,n$. Thus, E_Σ implies $Fu_1ix...u_pix = fF_1u_1ix...u_pix \; ... \; F_nu_1ix...u_pix = fG_1v_1x...v_qx \; ... \; G_nv_1x...v_qx = Gv_1x...v_qx$.

Rule I_3 Identical to Rule I_2.

($\Rightarrow$): Let u be a node of H labeled $Fu_1...u_p$, where the u_k's are nodes corresponding to attributes. We denote by $u\tau$ the term $Fu_1\tau...u_p\tau$.

Claim: Suppose E_Σ implies $Fu_1\tau...u_p\tau = Gv_1\rho...v_q\rho$, where the u_k's, v_j's correspond to *arbitrary* nodes of H, F,G are terms over the f's, and τ,ρ are terms over the i's (and a variable x). Also assume $Fu_1...u_p$ is a node of H, and there are nodes w_k, $k = 1,...,p$, such that there is a path from u_k to w_k labeled τ. Then $Gv_1...v_q$ is a node of H and there is a path from $Gv_1...v_q$ to $Fw_1...w_p$ labeled ρ.

The "only if" direction follows easily from the Claim, by Theorem 2.1.

Proof of Claim: If $E_\Sigma \models \sigma = \sigma'$, then there is a sequence of terms $\sigma_0,...,\sigma_m$ such that σ_0 is σ, σ_m is

σ', and for $k=0,...,m-1$ the term σ_{k+1} is obtained from σ_k by rewriting a subterm $\varphi(\theta_1)$ as $\varphi(\theta_2)$, where $\theta_1=\theta_2$ ($\theta_2=\theta_1$) is an equation in E_Σ and φ is a substitution (Proposition 2.2). We call such a sequence a *proof* of the equation $\sigma=\sigma'$.

We define a relation $\prec$ on pairs of terms as follows:

$(\zeta,\zeta')\prec(\eta,\eta')$ iff E_Σ implies $\zeta=\zeta'$ and $\eta=\eta'$, and either
(i) the shortest proof of $\zeta=\zeta'$ is shorter than the shortest proof of $\eta=\eta'$, or
(ii) the above proofs have the same length, and ζ is a *proper* subterm of η, ζ' is a *proper* subterm of η'.

Obviously, $\prec$ is well-founded, so we can argue by induction on $\prec$. Let $\sigma_0,...,\sigma_m$ be a shortest proof of the equation $Fu_1\tau...u_p\tau=Gv_1\rho...v_q\rho$.

Basis: $m=0$. Using I_2, I_1, we see by an easy induction on the structure of F that there is a node $Fw_1...w_p$ and a path from $Fu_1...u_p$ to $Fw_1...w_p$ labeled τ (see Figure 3-3).

Induction Step: We assume that the Claim holds for all equations $\zeta=\zeta'$ implied by E_Σ, where $(\zeta,\zeta')\prec(Fu_1\tau...u_p\tau, Gv_1\rho...v_q\rho)$; we will show that it holds for the equation $Fu_1\tau...u_p\tau=Gv_1\rho...v_q\rho$. We distinguish two cases:

Case 1: For $k=0,...,m-1$, σ_{k+1} is obtained from σ_k by rewriting a *proper* subterm. This means F is $fF_1...F_n$, G is $fG_1...G_n$, and $F_su_1\tau...u_p\tau$ is rewritten as $G_sv_1\rho...v_q\rho$, $s=1,...,n$. Now for $s=1,...,n$, $F_su_1...u_p$ is a node of H and $(F_su_1\tau...u_p\tau, G_sv_1\rho...v_q\rho)\prec(Fu_1\tau...u_p\tau, Gv_1\rho...v_q\rho)$, so by the induction hypothesis $G_sv_1...v_q$ is a node of H and there is a path from $G_sv_1...v_q$ to $F_sw_1...w_p$ labeled ρ (see Figure 3-4). Now by Rules I_2, I_1 and an easy induction on the structure of F_s, there is a path from $F_su_1...u_p$ to $F_sw_1...w_p$ labeled τ; then by Rules I_2, I_1 there is a node $fF_1w_1...w_p$ ... $F_nw_1...w_p$, i.e. a node labeled $Fw_1...w_p$. It follows by Rules I_1, I_3 that there is a node $fG_1v_1...v_q$ ... $G_nv_1...v_q$, i.e. a node $Gv_1...v_q$, and that there is a path from $Gv_1...v_q$ to $Fw_1...w_p$ labeled ρ.

Case 2: For some k, $0\leq k\leq m-1$, σ_k is rewritten into σ_{k+1}. We distinguish four subcases:

Case 2a: $Fu_1\tau...u_p\tau$ is rewritten as $fa_1\xi...a_n\xi$, then as $a\xi$ using an equation $fa_1x...a_nx=ax$ in E_Σ and then as $Gv_1\rho...v_q\rho$. Clearly $(Fu_1\tau...u_p\tau, fa_1\xi...a_n\xi)\prec(Fu_1\tau...u_p\tau, Gv_1\rho...v_q\rho)$, so by the induction hypothesis there is a path from $fa_1...a_n$ to $Fw_1...w_p$ labeled ξ (see Figure 3-5). Since $\langle fa_1...a_n, a\rangle\in E_H$, there is a path from a to $Fw_1...w_p$ labeled ξ. We also have
$(a\xi, Gv_1\rho...v_q\rho)\prec(Fu_1\tau...u_p\tau, Gv_1\rho...v_q\rho)$, so by the induction hypothesis $Gv_1...v_q$ is a node of H

and there is a path from $Gv_1...v_q$ to $Fw_1...w_p$ labeled ρ.

Case 2b: $Fu_1\tau...u_p\tau$ is rewritten as $a\xi$, then as $fa_1\xi...a_n\xi$ using an equation $fa_1x...a_nx=ax$ in E_Σ and then as $Gv_1\rho...v_q\rho$. Clearly $(Fu_1\tau...u_p\tau,\ a\xi)\prec(Fu_1\tau...u_p\tau,\ Gv_1\rho...v_q\rho)$, so by the induction hypothesis there is a path from a to $Fw_1...w_p$ labeled ξ (see Figure 3-6). Since $\langle fa_1...a_n,\ a\rangle\in E_H$, there is a path from $fa_1...a_n$ to $Fw_1...w_p$ labeled ξ. We also have
$(fa_1\xi...a_n\xi,\ Gv_1\rho...v_q\rho)\prec(Fu_1\tau...u_p\tau,\ Gv_1\rho...v_q\rho)$, so by the induction hypothesis $Gv_1...v_q$ is a node of H and there is a path from $Gv_1...v_q$ to $Fw_1...w_p$ labeled ρ.

Case 2c: $Fu_1\tau...u_p\tau$ is rewritten as $a\xi$, then as $bi\xi$ using an equation $ax=bix$ in E_Σ and then as $Gv_1\rho...v_q\rho$. Clearly $(Fu_1\tau...u_p\tau,\ a\xi)\prec(Fu_1\tau...u_p\tau,\ Gv_1\rho...v_q\rho)$, so by the induction hypothesis there is a path from a to $Fw_1...w_p$ labeled ξ (see Figure 3-7). Since there is a directed arc (b, a) labeled i, there is a path from b to $Fw_1...w_p$ labeled $i\xi$. We also have
$(bi\xi,\ Gv_1\rho...v_q\rho)\prec(Fu_1\tau...u_p\tau,\ Gv_1\rho...v_q\rho)$, so by the induction hypothesis $Gv_1...v_q$ is a node of H and there is a path from $Gv_1...v_q$ to $Fw_1...w_p$ labeled ρ.

Case 2d: $Fu_1\tau...u_p\tau$ is rewritten as $bi\xi$, then as $a\xi$ using an equation $ax=bix$ in E_Σ and then as $Gv_1\rho...v_q\rho$. Clearly $(Fu_1\tau...u_p\tau,\ bi\xi)\prec(Fu_1\tau...u_p\tau,\ Gv_1\rho...v_q\rho)$, so by the induction hypothesis there is a path from b to $Fw_1...w_p$ labeled $i\xi$ (see Figure 3-8). Now there is a node c on this path such that the subpath from b to c is labeled i. Since there is a directed arc (b, a) labeled i, by Rules E_1, E_2, T we have $\langle a,\ c\rangle\in E_H$. Thus there is a path from a to $Fw_1...w_p$ labeled ξ. We also have
$(a\xi,\ Gv_1\rho...v_q\rho)\prec(Fu_1\tau...u_p\tau,\ Gv_1\rho...v_q\rho)$, so by the induction hypothesis $Gv_1...v_q$ is a node of H and there is a path from $Gv_1...v_q$ to $Fw_1...w_p$ labeled ρ.

This concludes the Proof of the Claim, so we are done. ∎

We remark here that Theorem 3.1 can be strengthened using the axiomatization of [54] for FD's and IND's (see Subsection 2.1.1). Specifically, we can show that we need not use Rule I_3 in the construction of H. To see this, consider the following sets of dependencies:
$F_H=\{u_1...u_p\to u \mid u_k,\ k=1,...,p$ and u are nodes of H such that $\langle Fu_1...u_p,\ u\rangle\in E_H\}$.
$I_H=\{u_1...u_q\subseteq v_1...v_q \mid u_k,v_k$ are nodes of H such that there is a path from v_k to u_k labeled τ, $k=1,...,q$, where τ is a term over the i's$\}$.
Here we assume that Rule I_3 was not used in the construction of H. Clearly $\Sigma\subseteq F_H\cup I_H$. Moreover, it is straightforward (but lengthy) to verify that $F_H\cup I_H$ is closed under the rules of [54] (using the fact

that H is closed under Rules T, $E_{1\text{-}2}$, $I_{1\text{-}2}$). Therefore, $\Sigma \models A_1 \ldots A_n \rightarrow A$ iff $a_1 \ldots a_n \rightarrow a$ is in F_H and $\Sigma \models B_1 \ldots B_m \subseteq A_1 \ldots A_m$ iff $b_1 \ldots b_m \subseteq a_1 \ldots a_m$ is in I_H. This stronger version, however, is not necessary for our purposes.

3.2 Typed IND's and Acyclic FD's

Suppose we are given a set Σ of FD's and *typed* IND's, over database scheme $D = \{R_k[U_k]: k = 1,\ldots,q\}$, $U_k \subseteq \mathcal{U}$. An attribute A_j of relation scheme R_k is now represented by a node a_j^k of H_Σ (cf. the graph notation of Section 2.1.1). The FD's and IND's in Σ are represented in H_Σ as explained at the beginning of this Section. We use a different label i^{jk} for each typed IND $R_k{:}A_1 \ldots A_m \subseteq R_j{:}A_1 \ldots A_m$ in Σ.

The fact that Σ contains only typed IND's induces a special structure on the graph H (of Theorem 3.1), which we will now analyze. Consider the graph F_Σ of Section 2.1.1. This graph has a node a for each attribute A in $\mathcal{U}$ and a group of red arcs $(a_1,a),\ldots,(a_n,a)$ labeled f for each group of red arcs $(a_1^k,a^k),\ldots,(a_n^k,a^k)$ labeled f of H_Σ. We define two *partial* functions *type, node* on the set of terms (over the a^k's and the f's). If τ is a term, *type*(τ) is the name of a relation scheme in *D* and *node*(τ) is a node of F_Σ. The functions *type, node* are defined inductively as follows:

1. For each attribute A of R_k, *type*(a^k) $= R_k$, *node*(a^k) $=$ a.

2. If *type*(τ_j) $= R_k$ and *node*(τ_j) $= v_j$ for $j = 1,\ldots,n$, where there is a group of red arcs $(v_1,v),\ldots,(v_n,v)$ labeled f in F_Σ, then *type*($f\tau_1 \ldots \tau_n$) $= R_k$, *node*($f\tau_1 \ldots \tau_n$) $=$ v.

The crucial property of H (in the case of typed IND's) is given in the following

Lemma 3.1: The functions *type, node* are defined on all terms that appear as labels of nodes of H. Moreover,

1. If $f\tau_1 \ldots \tau_n$ is a node of H then for $j = 1,\ldots,n$ we have *type*(τ_j) $= R_k$ and *node*(τ_j) $= v_j$, where there is a group of red arcs $(v_1,v),\ldots,(v_n,v)$ labeled f in F_Σ.
2. If $\langle u,v \rangle$ is an undirected arc of H then *type*(u) $=$ *type*(v) and *node*(u) $=$ *node*(v).
3. If (u,v) is a directed arc of H labeled i^{jk} then *type*(u) $= R_j$, *type*(v) $= R_k$ and *node*(u) $=$ *node*(v).

Proof: Straightforward simultaneous induction on the number of applications of rules that produced a node (arc) of H. ∎

Assume now that F_Σ is *acyclic*: It is not hard to see that in this case each node of F_Σ can be the image (under *node*) of at most an *exponential* number of terms (in the size of F_Σ). Therefore by Lemma 3.1 the size of H is at most exponential, and by Theorem 3.1 we obtain

Corollary 3.1: The implication problem for acyclic FD's and typed IND's is decidable. ▮

In particular, implication of an FD can be tested in exponential time, and implication of an IND can be tested in *nondeterministic* exponential time (by guessing appropriate paths of H). Whether these bounds can be improved is an open question.

We remark here that if Σ is a set of FD's and typed IND's over database scheme D and $\Sigma \models \sigma$, where σ is an IND, then σ must be typed. This follows easily from Theorem 3.1 and Lemma 3.1, but can also be seen directly as follows: Consider a database d which associates to each relation scheme R_k of D a single tuple t_k, where $t_k[A_j]=j$, $A_j \in \mathcal{U}$. Clearly d satisfies all FD's and all typed IND's (over D), but violates any IND which is not typed.

3.3 Inference of FD's under Pairwise Consistency

Let Σ be a set of FD's over database scheme D and let PC(D) be the set of all typed IND's over D (recall that PC(D) expresses the fact that the database is *pairwise consistent*). By the remark at the end of the previous Section, PC(D)$\cup\Sigma$ does not imply any new IND's, so we need only be concerned with implication of FD's. Furthermore, observe that if a database d over D satisfies PC(D), then $R_k{:}A_1 \ldots A_n \rightarrow A$ holds in relation R_k iff $R_j{:}A_1 \ldots A_n \rightarrow A$ holds in relation R_j, where $R_k[U_k]$, $R_j[U_j]$ both contain attributes $A_1,\ldots,A_n,A$. For this reason we can suppress relation names from FD's.

In the presence of only typed IND's, every term that appears as label of a node of the graph H (of Theorem 3.1) is of the form $Fa_1^k \ldots a_p^k$, where $type(Fa_1^k \ldots a_p^k) = R_k$; this is an easy consequence of Lemma 3.1. Now suppose we have pairwise consistency, there is a node labeled $Fa_1^k \ldots a_p^k$, and A_m appears in relation scheme R_j, $m=1,\ldots,p$; then there is a directed arc labeled i^{kj} from a_m^k to a_m^j. Thus, by Rule I_2 (and an easy induction on the structure of F) there is a node labeled $Fa_1^j \ldots a_p^j$. This observation allows us to represent the graph H more succinctly, by having only one node a_m for each attribute A_m and a node $Fa_1 \ldots a_p$ for each term
$Fa_1^k \ldots a_p^k$ that appears as a label of a node of H.

This representation can be further simplified if the FD's in Σ are all *unary*. In this case all we need to observe is that the terms that appear as labels of nodes correspond to *paths* in the graph F_Σ (recall that F_Σ is a directed graph with a node a_m for each attribute A_m and an arc (a_k,a_j) for each FD $A_k \rightarrow A_j$ in Σ). Moreover, it is not difficult to see that *all* such paths will appear as labels of nodes. We now give the formal details of this representation.

Let V be the set of nodes of F_Σ. For each attribute A_m, let T_{A_m} be the following (possibly infinite) directed tree:
the set of nodes $P_{A_m} \subseteq a_m V^*$ is the set of all *paths* in F_Σ which start at a_m (denoted as sequences of nodes);
the set of arcs is $\{(sa_k, sa_k a_j) \mid s \in V^*, sa_k \in P_{A_m}, A_k \rightarrow A_j \in \Sigma\}$.

Let $P = \bigcup_{A_m \in \mathcal{U}} P_{A_m}$. Define E to be the smallest set of undirected arcs on P which contains $\langle s,s \rangle$ for all $s \in P$ and $\langle a_k a_j, a_j \rangle$ for all $A_k \rightarrow A_j$ in Σ, and is closed under the following rules:

1. Propagation: If $\langle sa_k, s'a_k \rangle \in E$, then $\langle sa_k a_j, s'a_k a_j \rangle \in E$ for all $A_k \rightarrow A_j$ in Σ.

2. Pseudo-Transitivity: If $\langle s_1,s_2 \rangle$, $\langle s_2,s_3 \rangle$ are in E, $s_k \in P_{A_k}$, and there is a relation scheme in *D* which contains A_1,A_2,A_3, then $\langle s_1,s_3 \rangle$ is in E.

By the preceding remarks and Theorem 3.1, we have

Lemma 3.2: $PC(D) \cup \Sigma \models A_k \rightarrow A_j$ iff $\langle s,a_j \rangle \in E$ for some $s \in P_{A_k}$. ∎

Example 3.1: Figure 3-9 has an example where $D = \{R_0[A_1Q_1Q_2B], R_1[AA_1Q_1], R_2[A_1Q_1A_2Q_2], R_3[A_2Q_2B]\}$ and Σ is $\{A \rightarrow Q_1, A_1 \rightarrow A_2, A_2 \rightarrow B, Q_1 \rightarrow A_2, Q_2 \rightarrow B\}$. In this case, $PC(D) \cup \Sigma \models A \rightarrow B$.

The "only if" direction of Lemma 3.2 can also be proved by a counterexample construction. Suppose $\langle s,a_j \rangle$ is not in E, for any s in P_{A_k}; we will construct a pairwise consistent database d over *D* which satisfies the FD's in Σ but violates $A_k \rightarrow A_j$.
For each attribute A_m in $\mathcal{U}$ the domain of A_m, $\mathcal{D}_{A_m}$, consists of all functions $f: P_{A_m} \rightarrow \{0,1\}$ such that, if $\langle s,s' \rangle \in E$, $s,s' \in P_{A_m}$, then $f(s) = f(s')$.
Let U_n be $A_1 \ldots A_p$. We construct a relation r_n over $R_n[U_n]$ as follows: A tuple $f_1 \ldots f_p$ ($f_\kappa \in \mathcal{D}_{A_\kappa}$) is in r_n iff, for any s in P_{A_κ}, s' in P_{A_λ} $(1 \leq \kappa, \lambda \leq p)$ with $\langle s,s' \rangle \in E$, we have $f_\kappa(s) = f_\lambda(s')$.
It is easy to see that the database d consisting of the relations r_n satisfies the FD's in Σ (by the definition of the set E). We also claim that d is pairwise consistent. The key observation is that, if

$A_{\kappa_1}...A_{\kappa_q}$ is any subset of U_n, then the projection of r_n on $A_{\kappa_1}...A_{\kappa_q}$ consists of exactly those tuples $f_{\kappa_1}...f_{\kappa_q}$ for which $f_B(s)=f_C(s')$ whenever $\langle s,s'\rangle \in E$ (B,C in $A_{\kappa_1}...A_{\kappa_q}$). Finally, one can verify that if $\langle s,a_j\rangle$ is not in E, for any s in P_{A_k}, then d violates $A_k \rightarrow A_j$.

The above construction produces in general an *uncountable* counterexample. Observe, however, that if Σ is *acyclic* then each P_{A_m} is finite, so the counterexample is finite. It follows that for acyclic unary FD's under pairwise consistency, finite implication coincides with (unrestricted) implication:

Theorem 3.2: The class of acyclic unary FD's under pairwise consistency is finitely controllable. ∎

We now make some simple remarks about the set of undirected arcs E. Observe that, if $\langle s_1, s_2\rangle \in E$ and s_1s', s_2s' are in P, then $\langle s_1s', s_2s'\rangle \in E$. This is an easy consequence of Propagation. Also, if $\langle as_1, as_2\rangle \in E$ and sas_1, sas_2 are in P, then $\langle sas_1, sas_2\rangle \in E$. To see this, suppose s is s'b, where b is a node such that $B \rightarrow A$ is in Σ. Then $\langle ba, a\rangle \in E$, so by Propagation $\langle bas_1, as_1\rangle \in E$. Similarly $\langle bas_2, as_2\rangle \in E$. Then by Pseudo-Transitivity $\langle bas_1, bas_2\rangle \in E$. We are now ready to prove the main result of this Section.

Theorem 3.3: The implication problem for unary FD's in the presence of pairwise consistency is undecidable.

Proof: We reduce the uniform word problem for semigroups (Thue systems [50]) to implication of u-FD's under pairwise consistency. We assume that we are given a set S of word equations of the form $\alpha_i\alpha_j=\alpha_k$; the problem is to determine whether $S \models \alpha_1\alpha_2=\alpha_3$. Recall that this happens iff the string α_3 can be obtained from the string $\alpha_1\alpha_2$ by successively replacing a substring w_1 by a substring w_2, where $w_1=w_2$ ($w_2=w_1$) is an equation in S.

For each *given* equation in S, say $\alpha_i\alpha_j=\alpha_k$, we include in our database scheme relation schemes $R_{1\text{-}7}$, $K_{1\text{-}2}$, $R'_{1\text{-}3}$, L, $M_{1\text{-}2}$, as shown in Figure 3-10. The directed arcs represent unary FD's. There are two general-purpose attributes X,Y. For each α_m there are two attributes A_m,B_m, and for each equation there is a set of attributes $Q_{1\text{-}8}$.
If the equation *to be inferred* is $\alpha_1\alpha_2=\alpha_3$, then we include in the database scheme relation schemes $R_{1\text{-}7}$, $K_{1\text{-}2}$, $R'_{1\text{-}3}$, L, $J_{1\text{-}3}$ and FD's as in Figure 3-10 (where now A_i,B_i are A_1, B_1, A_j,B_j are A_2,B_2, A_k,B_k are A_3,B_3, and we have used attributes $Q'_{1\text{-}8}$). We will show that the u-FD $Q'_6 \rightarrow Q$ is implied iff $S \models \alpha_1\alpha_2=\alpha_3$. Let P be a set of nodes and E a set of undirected arcs as in Lemma 3.2.

Claim: The undirected arc $\langle xa_1b_1yxa_2b_2y, xa_3b_3y\rangle$ is in E iff $S\models\alpha_1\alpha_2=\alpha_3$.

Proof of Claim: We will give a characterization of the set E. Let e be an equation $\alpha_i\alpha_j=\alpha_k$ in S, and suppose e gives rise to relation schemes $R_{1\text{-}7}$, $K_{1\text{-}2}$, $R'_{1\text{-}3}$, L, $M_{1\text{-}2}$, as in Figure 3-10. Consider the following sets of undirected arcs which correspond to e (all these arcs are in E):

E_1^e:

$\langle xa_i, a_i\rangle$,

$\langle a_ib_i, b_i\rangle$, $\langle q_1b_i, b_i\rangle$, $\langle a_ib_i, q_1b_i\rangle$,

$\langle b_iy, y\rangle$, $\langle q_2y, y\rangle$, $\langle b_iy, q_2y\rangle$,

$\langle yx, x\rangle$, $\langle q_3x, x\rangle$, $\langle yx, q_3x\rangle$,

$\langle xa_j, a_j\rangle$, $\langle q_4a_j, a_j\rangle$, $\langle xa_j, q_4a_j\rangle$,

$\langle a_jb_j, b_j\rangle$, $\langle q_5b_j, b_j\rangle$, $\langle a_jb_j, q_5b_j\rangle$,

$\langle b_jy, y\rangle$, $\langle q_6y, y\rangle$, $\langle b_jy, q_6y\rangle$,

$\langle xa_k, a_k\rangle$,

$\langle a_kb_k, b_k\rangle$, $\langle q_7b_k, b_k\rangle$, $\langle a_kb_k, q_7b_k\rangle$,

$\langle b_ky, y\rangle$, $\langle q_8y, y\rangle$, $\langle b_ky, q_8y\rangle$.

E_2^e:

$\langle xa_ib_i, q_1b_i\rangle$,

$\langle a_ib_iy, q_2y\rangle$, $\langle q_1b_iy, q_2y\rangle$,

$\langle b_iyx, q_3x\rangle$, $\langle q_2yx, q_3x\rangle$,

$\langle yxa_j, q_4a_j\rangle$, $\langle q_3xa_j, q_4a_j\rangle$,

$\langle xa_jb_j, q_5b_j\rangle$, $\langle q_4a_jb_j, q_5b_j\rangle$,

$\langle a_jb_jy, q_6y\rangle$, $\langle q_5b_jy, q_6y\rangle$,

$\langle xa_kb_k, q_7b_k\rangle$,

$\langle a_kb_ky, q_8y\rangle$, $\langle q_7b_ky, q_8y\rangle$.

E_3^e:

$\langle q_1b_iyx, q_3x\rangle$, $\langle q_2yxa_j, q_4a_j\rangle$, $\langle q_3xa_jb_j, q_5b_j\rangle$, $\langle q_4a_jb_jy, q_6y\rangle$,

$\langle xa_kb_ky, q_8y\rangle$.

E_4^e:

$\langle q_1 b_i y x a_j, q_4 a_j \rangle$, $\langle q_2 y x a_j b_j, q_5 b_j \rangle$, $\langle q_3 x a_j b_j y, q_6 y \rangle$.

E_5^e:

$\langle q_1 b_i y x a_j b_j, q_5 b_j \rangle$, $\langle q_2 y x a_j b_j y, q_6 y \rangle$.

E_6^e:

$\langle x a_i b_i y x a_j b_j y, q_6 y \rangle$.

E_7^e:

$\langle q_6 y, q_8 y \rangle$, $\langle x a_k b_k y, q_6 y \rangle$, $\langle x a_i b_i y x a_j b_j y, q_8 y \rangle$,
$\langle x a_i b_i y x a_j b_j y, x a_k b_k y \rangle$.

It is not difficult to see that for each equation e in S, $k = 1,...,7$, E_k^e is contained in E (compare with Figure 3-9).
Now consider the following set of arcs E′: Let $\langle s_1, s_2 \rangle$ be a member of some E_k^e (for some e,k), and suppose s′ is obtained from s by successively replacing a subsequence $x a_i b_i y x a_j b_j y$ by a subsequence $x a_k b_k y$ (or vice versa), where $\alpha_i \alpha_j = \alpha_k$ is in S. If $s_1 s$, $s_2 s'$ are in P, then put $\langle s_1 s, s_2 s' \rangle$ in E′. Also if s, s′ are in P, then put $\langle s, s' \rangle$ in E′.

By the remarks immediately preceding the statement of Theorem 3.3 (and the fact that $E_k^e \subseteq E$) we have $E' \subseteq E$. Furthermore E′ contains the arcs initially put in E, and clearly it is closed under Propagation. It is also straightforward (albeit a bit tedious) to verify that E′ is closed under Pseudo-Transitivity. Therefore $E \subseteq E'$, and thus $E = E'$. The Claim now follows from this characterization of E.

To finish the Proof, observe that $Q_6' \rightarrow Q$ is implied (Lemma 3.2) iff $\langle x a_1 b_1 y x a_2 b_2 y, x a_3 b_3 y \rangle$ is in E (cf. Figure 3-10). ▮

We will now show that there is no k-ary axiomatization for implication of u-FD's in the presence of pairwise consistency.

Let D be a database scheme and Θ a set of sentences about D (for instance, FD's and IND's). An axiom system for implication of sentences in Θ is *k-ary* [16] iff it is universe-bounded (i.e. only attributes in D are mentioned) and every rule has at most k antecedents, for some fixed integer k. Observe that the axiom system of [54] for implication of FD's and IND's is not k-ary, because Rule 10 violates the boundedness condition (see Subsection 2.1.1).

Let $\Sigma \subseteq \Theta$, σ in Θ. We say that Σ is *closed under implication* iff whenever $\Sigma \models \sigma$ we have $\sigma \in \Sigma$. Also, Σ is *closed under k-ary implication* iff whenever $\Sigma' \models \sigma$, where $\Sigma' \subseteq \Sigma$, $|\Sigma'| \leq k$, we have $\sigma \in \Sigma$. The following characterization for the existence of k-ary axiomatizations is taken from [16]:

Proposition 3.1: There is a k-ary axiomatization for implication of sentences in Θ iff whenever $\Sigma \subseteq \Theta$ is closed under k-ary implication, Σ is closed under implication. ∎

Theorem 3.4: There is no k-ary axiomatization for implication of u-FD's under pairwise consistency (we consider here axiomatizations involving arbitrary FD's and IND's).

Proof: Let $\mathcal{U}$ be $\{A, A_1, \ldots, A_k, Q_1, \ldots, Q_k, B\}$ and let D be a database scheme over $\mathcal{U}$ consisting of relation schemes $R_0[AQ_1 \ldots Q_k B]$, $R_1[AA_1Q_1]$, $R_j[A_{j-1}Q_{j-1}A_jQ_j]$, $j=2,\ldots,k$, $R_{k+1}[A_kQ_kB]$. Let Φ be the following set of FD's over D: $R_1{:}A \rightarrow A_1$, $R_j{:}A_{j-1} \rightarrow A_j$, $j=2,\ldots,k$, $R_j{:}Q_{j-1} \rightarrow A_j$, $j=2,\ldots,k$, $R_{k+1}{:}A_k \rightarrow B$, $R_{k+1}{:}Q_k \rightarrow B$, $R_0{:}Q_j \rightarrow B$, $j=1,\ldots,k$ (cf. Figure 3-9 for the case $k=2$).

Consider the set Φ' of FD's which are consequences of Φ. The set Φ' can be constructed by closing Φ under Rules 1,2,3 of the axiom system of [54] (see Subsection 2.1.1). Let Σ be $\Phi' \cup PC(D)$. We will show that Σ is not closed under implication (of FD's and IND's), but is closed under k-ary implication (of FD's and IND's). Theorem 3.4 will then follow by Proposition 3.1.

For the first part, it is not hard to see that $\Sigma \models \sigma$, where σ is $R_0{:}A \rightarrow B$ (cf. Figure 3-9). Since σ is not in Σ, we are done.

For the second part, suppose $\Sigma' \models \sigma$, where $\Sigma' \subseteq \Sigma$, $|\Sigma'| \leq k$, σ is an IND or an FD. We will show that σ is in Σ.

If σ is an IND, then σ must be typed, by the remark at the end of Section 3.2. Thus σ is in Σ.

Suppose now σ is an FD $R_p{:}C_1 \ldots C_q \rightarrow C_0$, where $0 \leq p \leq k+1$ and all the C_j's are in $\mathcal{U}$. Since all the FD's in Φ are unary, it easily follows from Theorem 3.1 that $\Sigma' \models R_p{:}C_m \rightarrow C_0$, for some m, $1 \leq m \leq q$. We will argue that $R_p{:}C_m \rightarrow C_0$ is in Φ; from this it easily follows that σ is in Φ', i.e. it is in Σ.

Consider the nodes c_m, c_0 of the graph F_Σ (cf. Figure 3-9). If there is no directed path from c_m to c_0, then we can construct a relation r over $\mathcal{U}$ which satisfies all the FD's in Φ (without their relation names) but violates $C_m \rightarrow C_0$. We can then project r over the R_j's to obtain a database d over D which satisfies Σ (and thus also Σ') and violates $R_p:C_m \rightarrow C_0$.

Thus, there is a directed path from c_m to c_0. Since C_m, C_0 also appear in the same relation name, it is easy to check that $R_p:C_m \rightarrow C_0$ is in Φ, *unless* $R_p:C_m \rightarrow C_0$ is $R_0:A \rightarrow B$. However, since $|\Sigma'| \leq k$ one of the FD's $R_1:A \rightarrow A_1$, $R_j:A_{j-1} \rightarrow A_j$, $j=2,\ldots,k$, $R_{k+1}:A_k \rightarrow B$ must be missing from Σ' and therefore we cannot have $\Sigma' \models R_0:A \rightarrow B$ (since there is no directed path from a to b in $F_{\Sigma'}$). This concludes the proof. ∎

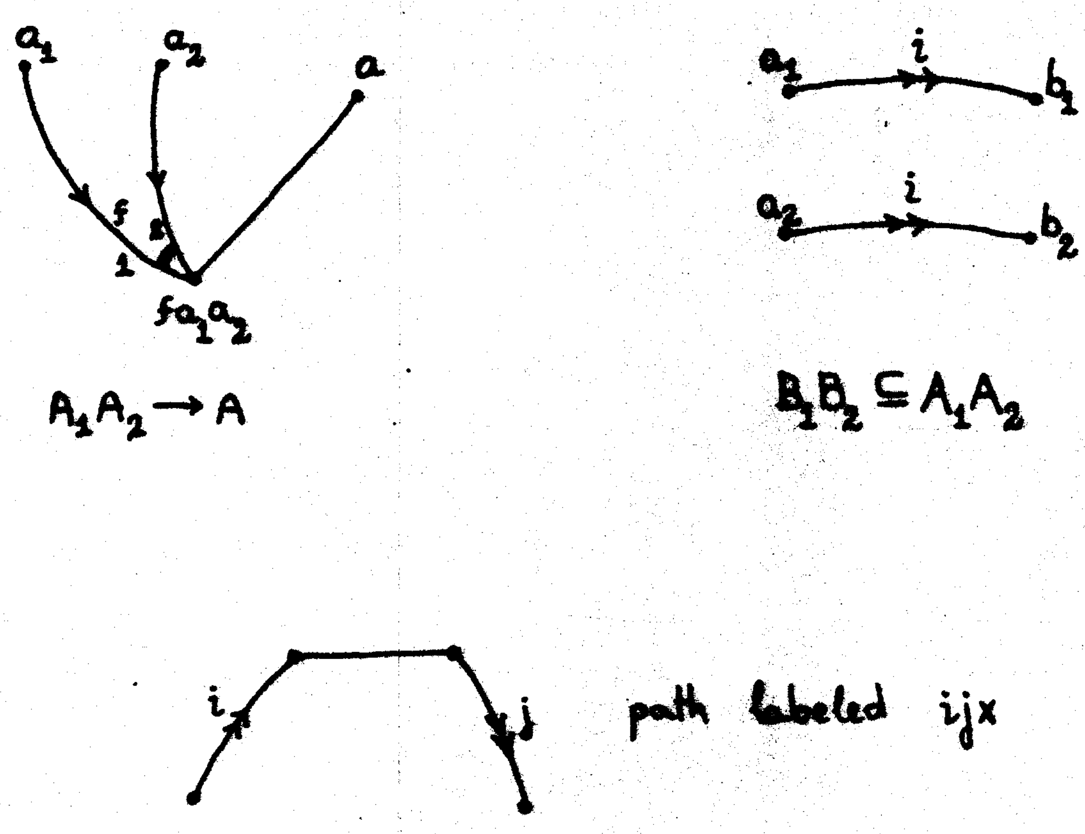


Figure 3-1: Another graph notation for FD's and IND's

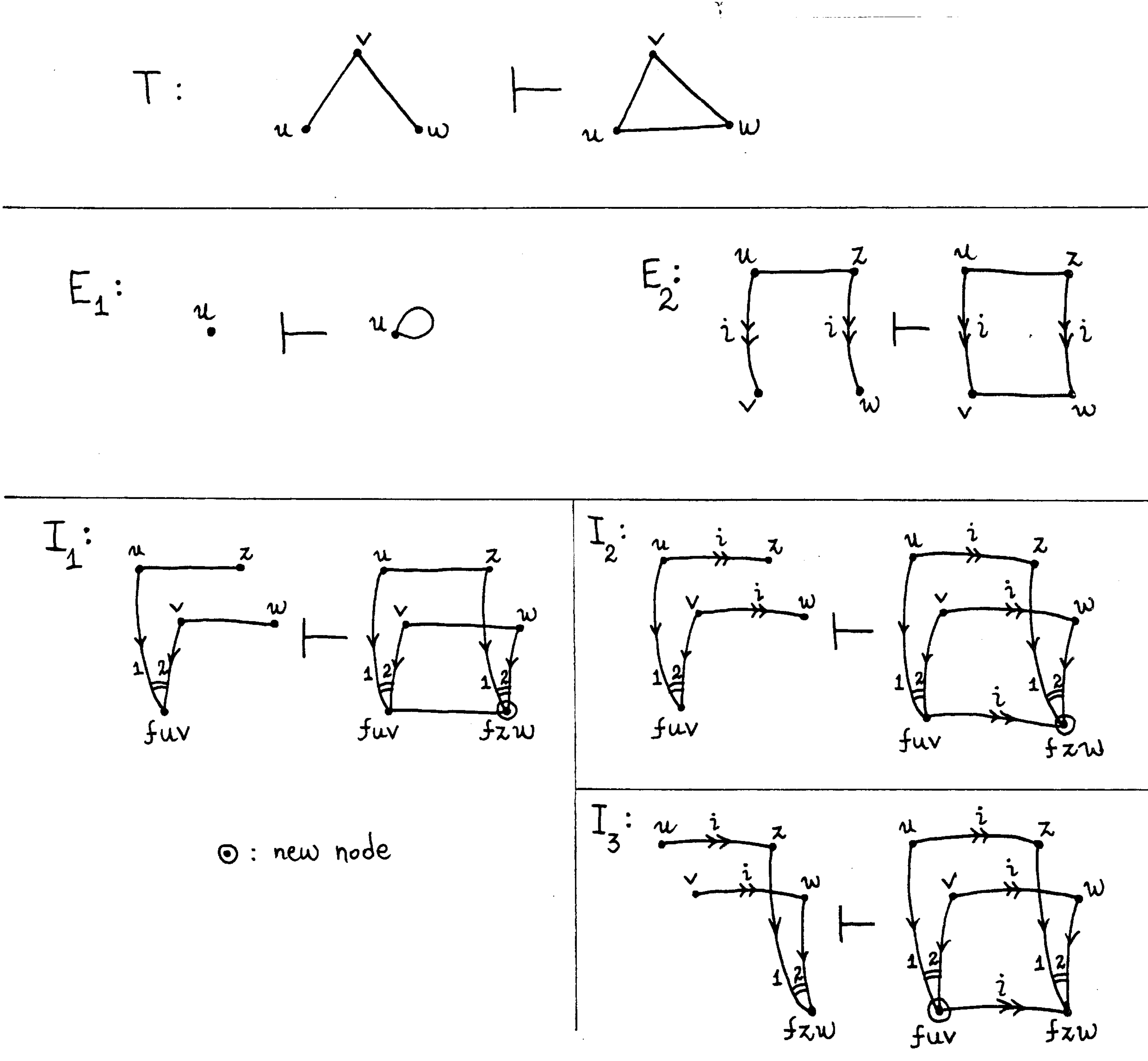


Figure 3-2: Graph rules for FD's and IND's

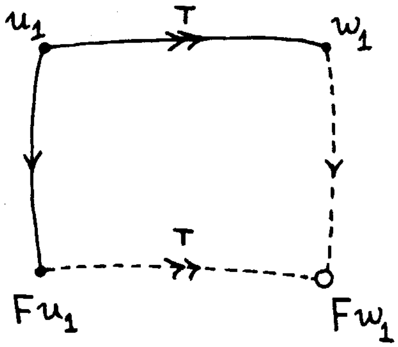


Figure 3-3: Basis case

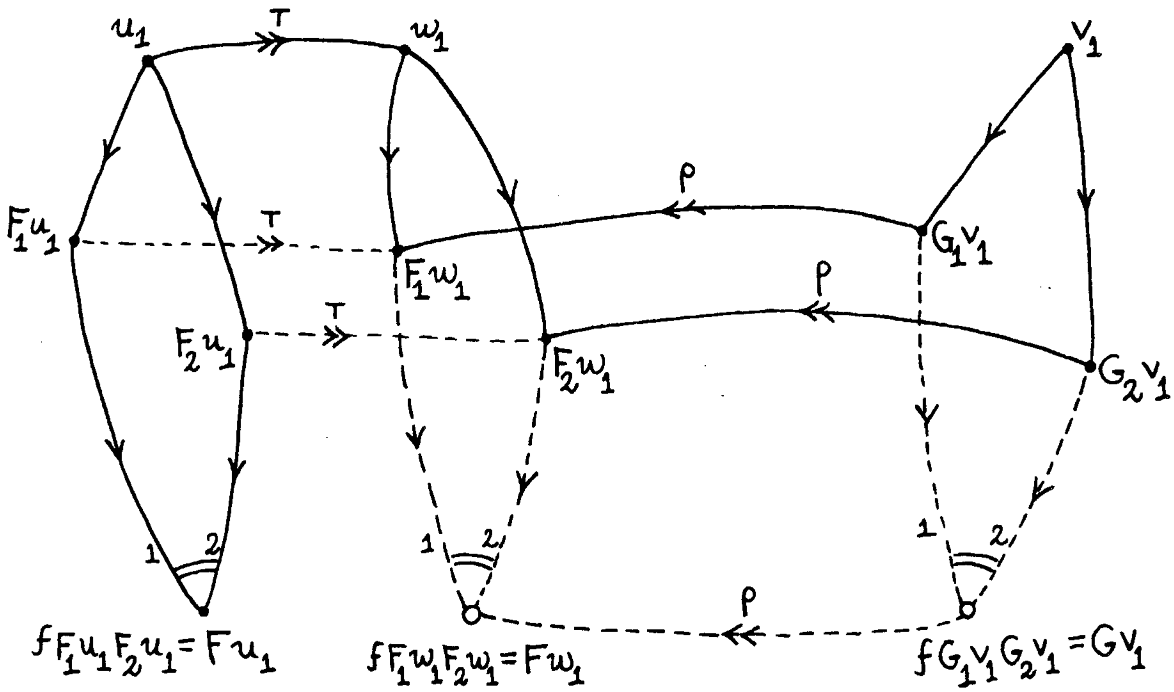


Figure 3-4: Case 1

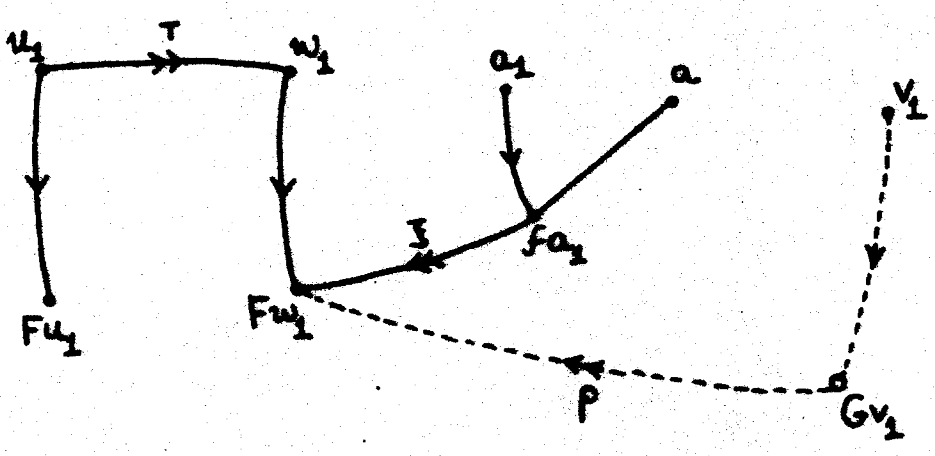


Figure 3-5: Case 2a

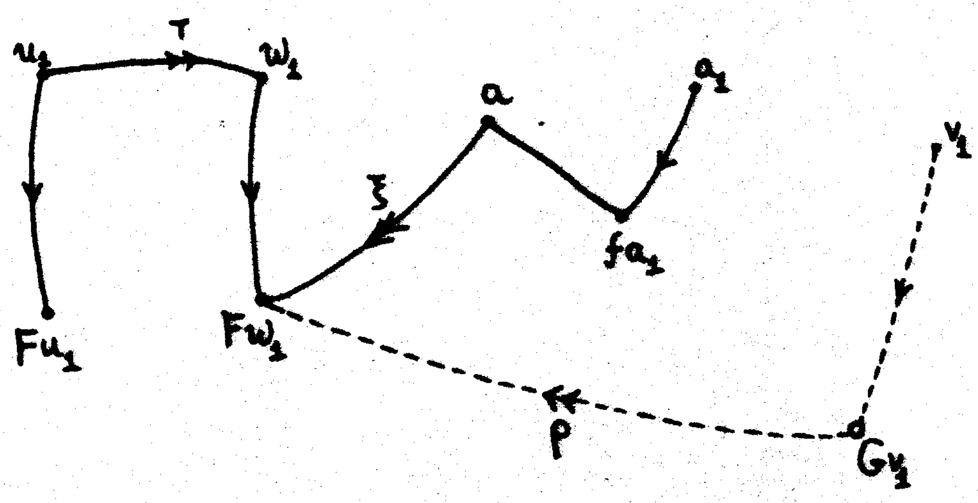


Figure 3-6: Case 2b

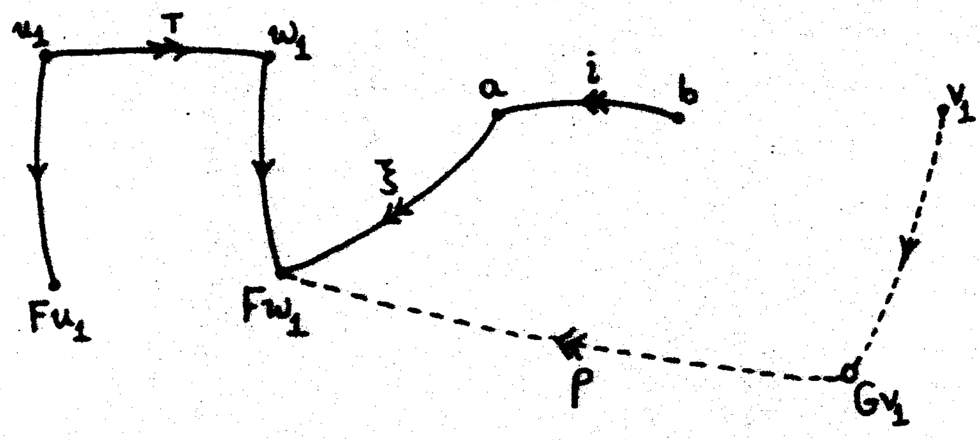


Figure 3-7: Case 2c

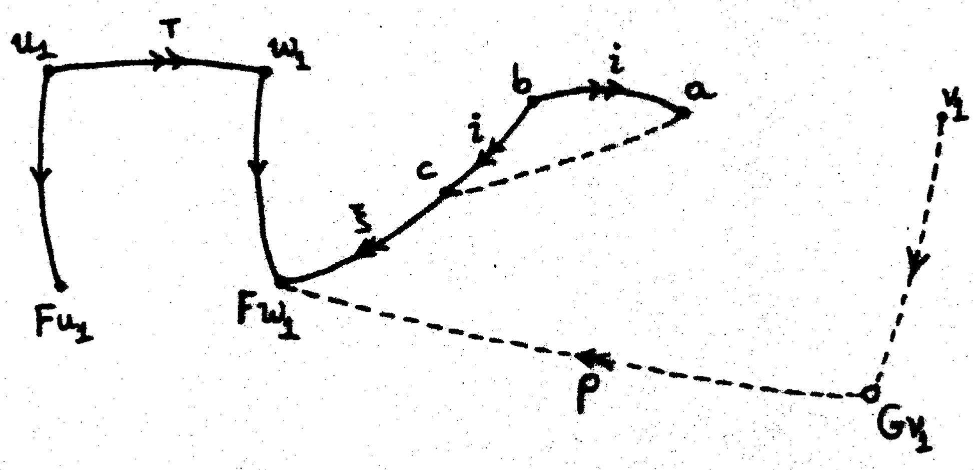


Figure 3-8: Case 2d

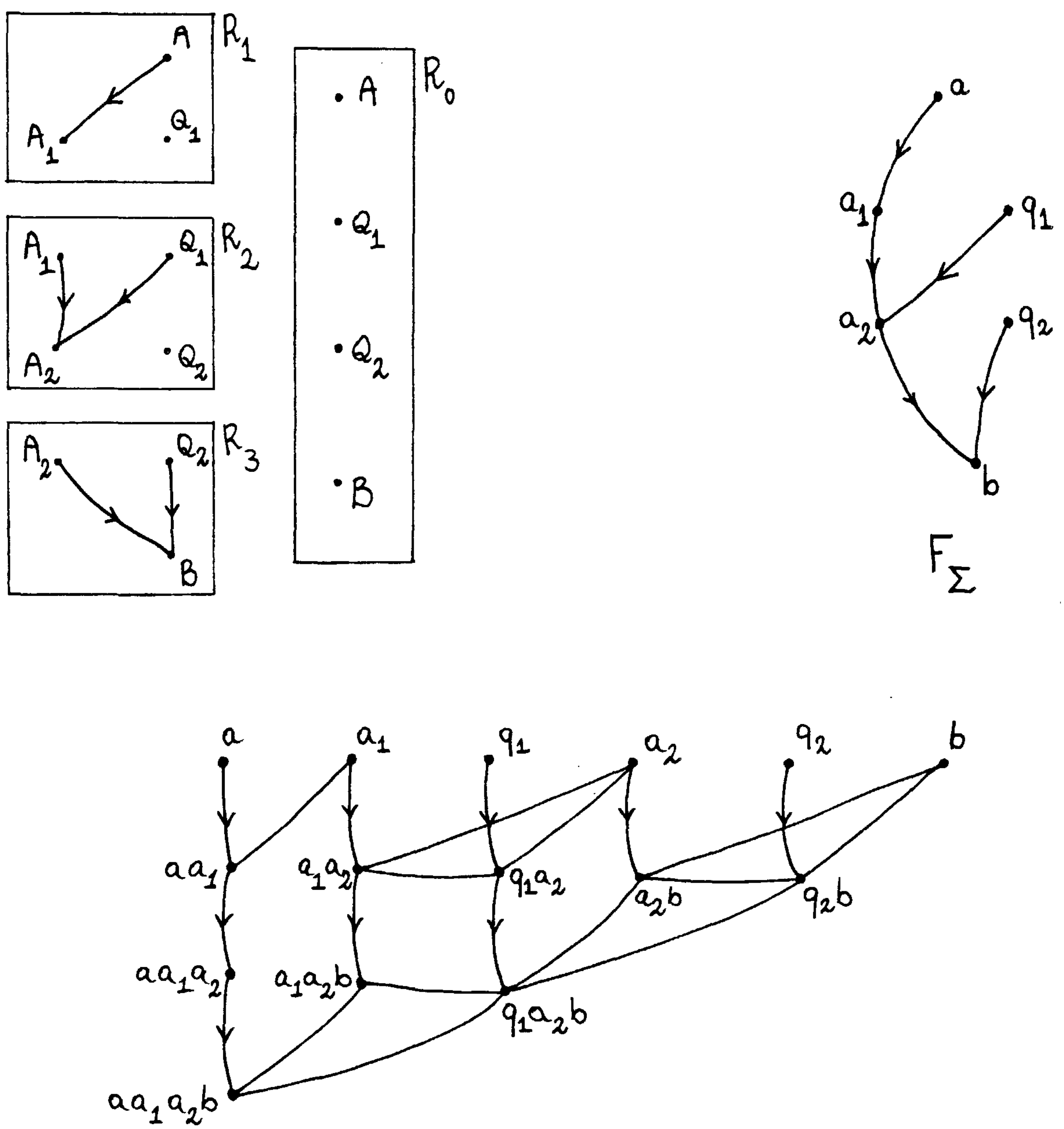


Figure 3-9: Example of FD inference under pairwise consistency

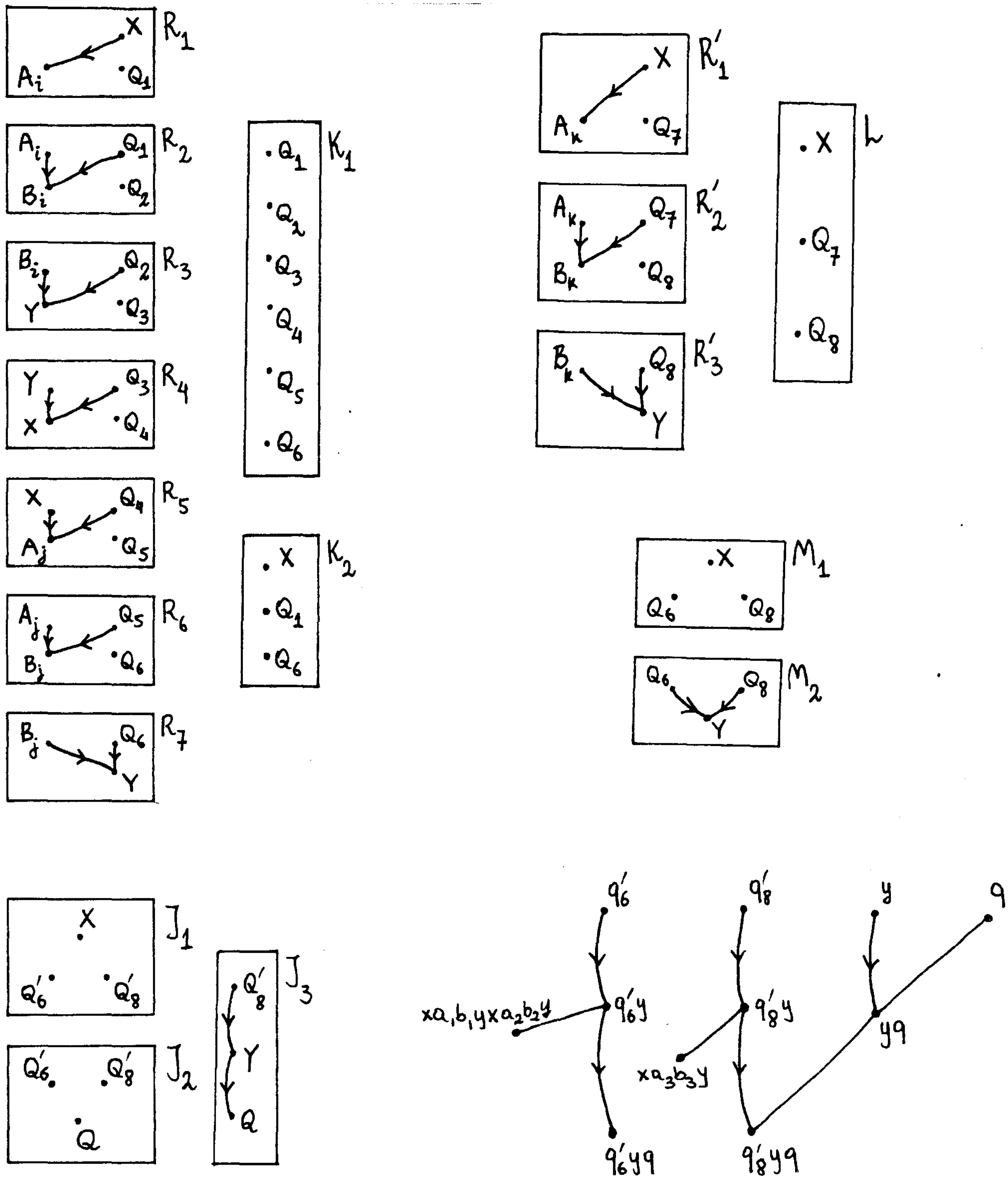


Figure 3-10: Gadgets for Proof of Theorem 3.3

Chapter Four

Finite Implication of FD's and Unary IND's

A natural question is whether our equational approach can handle *finite implication* of database constraints. Ideally, we would like to be able to replace $\models$ by $\models_{\text{fin}}$ throughout Theorem 2.1. It is easily seen that the same arguments can show that (iii)$\Rightarrow$(ii) and (ii)$\Rightarrow$(i) in the finite case (the constructions given map finite counterexamples to finite counterexamples). The argument for (i)$\Rightarrow$(iii), however, breaks down, because it is based on the existence of a complete proof procedure for implication (namely the chase) and such a proof procedure cannot exist for finite implication [54, 19]. As a matter of fact, the same *syntactic* nature of the proofs of Theorems 2.3 and 3.3 prevents us from proving undecidability of finite implication. The weaker proofs of [54, 19], because of their *semantic* nature, can easily be done for the finite case.

However, Theorem 2.4 also holds for the finite case: By the discussion above one can see that $\models$ can be replaced by $\models_{\text{fin}}$ in Theorem 2.1 if we have a *finitely controllable* class of FD's and IND's, i.e. a class where $\models_{\text{fin}}$ is the same as $\models$. Acyclic IND's and FD's provide an easy example of such a class, because the chase in this case constructs a *finite counterexample* if the implication does not hold. Another example of a finitely controllable class is acyclic unary FD's under pairwise consistency (Theorem 3.2).

If $\models_{\text{fin}}$ is different from $\models$, we might still be able to handle the finite case if there is a complete proof procedure for finite implication. In this Chapter we provide such a class: we show that there is a complete proof procedure for finite implication of FD's and *unary* IND's. This proof procedure is then used to prove a (weaker) analogue of Theorem 2.1. for finite implication of FD's and u-ID's.

Let Σ be a set of FD's and u-ID's over a database scheme D containing a single relation scheme $R[\mathcal{U}]$. If σ is an FD or u-ID, we will show that $\Sigma\models_{\text{fin}}\sigma$ iff σ can be proved from Σ using the following set of rules (*). We use X,Y to denote sets of attributes. We denote a u-ID $A\subseteq B$ alternatively as $B\supseteq A$.

Rules (*):

1. (reflexivity) $A \rightarrow A$, $A \in \mathcal{U}$.

2. (augmentation) *from* $X \rightarrow A$ *derive* $XY \rightarrow A$, $A \in \mathcal{U}$.

3. (transitivity) *from* $X \rightarrow A_k$, $k = 1,...,n$, $A_1...A_n \rightarrow A$, *derive* $X \rightarrow A$, $A \in \mathcal{U}$.

4. (u-ID reflexivity) $A \subseteq A$, $A \in \mathcal{U}$.

5. (u-ID transitivity) *from* $A \subseteq B$ *and* $B \subseteq C$ *derive* $A \subseteq C$, $A,B,C \in \mathcal{U}$.

6. (cycle rules) For every odd positive integer m and attributes A_k,
from $A_0 \rightarrow A_1$ *and* $A_1 \supseteq A_2$ *and...and* $A_{m-1} \rightarrow A_m$ *and* $A_m \supseteq A_0$
derive $A_1 \rightarrow A_0$ *and* $A_2 \supseteq A_1$ *and...and* $A_m \rightarrow A_{m-1}$ *and* $A_0 \supseteq A_m$.

Rules 1,2,3 are the standard rules for FD's [5] (written in our notation) and Rules 4,5 are the specialization of the general IND rules of [16] to u-ID's. Thus, Rules 1-5 are sound for general databases (infinite as well as finite). A simple counterexample construction shows that Rules 1-5 are also *complete* for unrestricted implication of FD's and u-ID's. More specifically, FD's and u-ID's *decouple* in the case of unrestricted implication.

Proposition 4.1: Let Σ_F be a set of FD's and Σ_I a set of u-ID's.
1. $\Sigma_F \cup \Sigma_I \models X \rightarrow A$ iff $\Sigma_F \models X \rightarrow A$.
2. $\Sigma_F \cup \Sigma_I \models A \subseteq B$ iff $\Sigma_I \models A \subseteq B$.

Proof: The "if" direction is obvious in both cases. We will show the "only if" direction.

1. Suppose Σ_F does not imply $X \rightarrow A$. Let $X^+ = \{B \mid B \in \mathcal{U}, \Sigma_F \models X \rightarrow B\}$. Consider a relation r consisting of tuples t_k, $k=0,1,2,...$, where $t_0[B]=0$, $B \in \mathcal{U}$, and for $k=1,2,...$, $t_k[B]=k-1$ if $B \in X^+$ and $t_k[B]=k$ otherwise. It is easy to see that r satisfies the FD's in Σ_F (the only tuples to check are t_0,t_1), and obviously r satisfies *all* u-ID's. Now since A is not in X^+, r violates $X \rightarrow A$. Therefore, $\Sigma_F \cup \Sigma_I$ does not imply $X \rightarrow A$.

2. Suppose Σ_I does not imply $A \subseteq B$. Let G_I be a directed graph which has a node a_m for each attribute A_m in $\mathcal{U}$ and a directed arc (a_j,a_k) for each u-ID $A_k \subseteq A_j$ in Σ_I. By our assumption, there is no directed path from b to a in G_I (cf. Rules 4,5). Thus, we can assign to each node u of G_I a number $c(u)$ so that $c(u) \leq c(v)$ whenever there is a directed path from u to v, and $c(b) > c(a)$ (this can be done

by a topological sort of the dag of strongly connected components of G_I [2]). Now consider a relation r consisting of tuples t_k, $k=0,1,2,\ldots$, where for A_m in $\mathcal{U}$ we have $t_k[A_m]=k+c(a_m)$. Clearly r satisfies all u-ID's in Σ_I and violates $A \subseteq B$. Moreover, r satisfies *all* FD's, so $\Sigma_F \cup \Sigma_I$ does not imply $A \subseteq B$. ∎

As a matter of fact, the cycle rules are *not* sound for infinite databases: Consider a relation r over relation scheme R[AB], consisting of tuples t_k, $k=0,1,2,\ldots$, where $t_k[A]=k$, $t_k[B]=k+1$: clearly r satisfies $B \rightarrow A$, $A \supseteq B$, but violates $B \supseteq A$. On the other hand, a simple counting argument shows that the cycle rules are sound in the finite case. Let $|r[A]|$ denote the cardinality of column A of relation r. If the antecedents of a cycle rule hold in r we have $|r[A_0]|=|r[A_1]|=\ldots=|r[A_m]|$. Now if a finite relation r satisfies $|r[A]|=|r[B]|$ and $A \rightarrow B$, it easily follows that it satisfies $B \rightarrow A$. Similarly, from $|r[A]|=|r[B]|$ and $A \supseteq B$ it follows for finite databases that $B \supseteq A$.

In order to analyze the rules (*), we use a graph notation for dependencies similar to the notation of Subsection 2.1.1. If Σ is a set of FD's and u-ID's, G_Σ is a graph which has a node a_m for each attribute A_m, a red arc (a_k,a_j) for each FD $A_k \rightarrow A_j$ in Σ, and a black arc (a_j,a_k) for each u-ID $A_k \subseteq A_j$ in Σ. If between nodes u,v of G_Σ we have red (black) arcs in both directions, we replace them with an undirected red (black) edge. The transitivity and cycle rules imply that, when $A_k \rightarrow A_j$ ($A_k \supseteq A_j$) corresponds to some arc in a directed cycle of G_Σ, we can infer $A_j \rightarrow A_k$ ($A_j \supseteq A_k$). In fact, if Σ is closed under the rules (*) then G_Σ has a good deal of structure, as can be easily verified.

Proposition 4.2: If Σ is a set of FD's and u-ID's closed under the rules (*) then G_Σ has the following properties:

1. Nodes have red (black) self-loops. The red (black) subgraph of G_Σ is transitively closed.
2. The subgraphs induced by the strongly connected components of G_Σ are undirected.
3. In each strongly connected component of G_Σ, the red (black) edges partition the set of nodes into a collection of node-disjoint cliques.
4. If $A_1 \ldots A_n \rightarrow A$ is an FD in Σ and $a_1,\ldots,a_n$ have a common ancestor u in the red subgraph of G_Σ, then G_Σ contains a red arc (u,a). ∎

By a topological sort of the dag of strongly connected components of G_Σ we can assign to each component a unique *scc-number*, smaller than the scc-number of all its descendant components in the dag [2]. Thus every node u in the graph G_Σ of Proposition 4.2 belongs to a unique maximal red (black) clique and a unique strongly connected component. Let scc(u) denote the scc-number of the component of node u.

Figure 4-1 illustrates an example of such a graph G_Σ. There are four strongly connected components, each a black clique, with all black arcs present from components with smaller to components with larger scc-number. The red cliques and red arcs are shown explicitly.

We now give a construction which lies at the heart of our completeness proof.

Lemma 4.1: Let Σ and G_Σ be as in Proposition 4.2 (i.e., closed under the rules (*)). Let the dag of strongly connected components of G_Σ be topologically sorted, so that each component has a unique scc-number. We can construct a finite relation r such that:

1. The u-FD $A \rightarrow B$ holds in r iff it is in Σ. Also all FD's in Σ hold in r.
2. The only repeated symbol in each column of r is 0, and the symbols in r[A] are exactly the integers from 0 to $|r[A]|-1$. Moreover, $|r[A]| \geq |r[B]|$ iff $scc(a) \leq scc(b)$ (thus, the u-ID $A \supseteq B$ holds in r iff $scc(a) \leq scc(b)$, and all u-ID's in Σ hold in r).

Proof: First put in r a tuple of all 0's. Process each strongly connected component of G_Σ in turn, in order of increasing scc-number. Begin processing a component by processing in turn each of its red cliques. To process a red clique κ, add a tuple with all 0's in the columns of the attributes of κ and of the attributes in all red cliques that are descendants of κ in the red subgraph of G_Σ. For now leave all other positions blank.

For every red clique κ keep a *count* of the number of 0's in one of its columns (by the way the construction proceeds all columns of κ have the same number of 0's). Now that one tuple was added for each red clique in the component, in order to terminate processing the component repeat certain of the tuples just added, so as to make the *counts* of all cliques in the component equal, and strictly greater than the *counts* of the cliques of the previous component. This is possible because no red clique is a red descendant of another red clique in the same component, or in a component with larger scc-number. Once a component is processed, no further 0's are added in its columns and its *counts* no longer change.

After adding tuples for all red cliques in all strongly connected components, we examine in turn each column. If the column has s blank positions, we fill them in with the numbers 1 to s, without any repetitions. We illustrate the construction in Figure 4-1.

Now it is easy to check that conditions 1,2 hold:

1. No u-FD in Σ was violated during the construction. Furthermore, all u-FD's not in Σ were violated. To see this, observe that if $A \rightarrow B$ is not in Σ, then the tuple inserted for the red clique of A

and the initial tuple of all 0's disprove $A \rightarrow B$.

We must also verify that all non-unary FD's in Σ are satisfied. Suppose $A_1 \ldots A_n \rightarrow A$ is an FD in Σ violated by r. Since the only repeated symbol in each column is 0, there is a tuple t of r such that $t[A_k]=0$, $k=1,\ldots,n$, $t[A]>0$. Now t was inserted in r while processing a red clique κ, so all 0's in t correspond to attributes that are functionally determined by every attribute B of κ. Since Σ is closed under Rules 1,2,3, it follows that $B \rightarrow A_k$ is in Σ, $k=1,\ldots,n$, and also $B \rightarrow A$ is in Σ. But then r satisfies $B \rightarrow A$, and since $t[B]=0$ and there is an initial tuple of all 0's, we obtain $t[A]=0$, which is a contradiction.

2. By the way r is constructed, the final *counts* are strictly increasing with the scc-numbers, and are equal in all columns of a strongly connected component. ∎

We will now prove our main result:

Theorem 4.1: The rules (*) are sound and complete for finite implication of FD's and u-ID's.

Proof: We have already argued for soundness, so it remains to show completeness. Let Σ be a set of FD's and u-ID's closed under the rules (*), and let σ be an FD or u-ID not in Σ. We will exhibit a finite counterexample relation r which satisfies Σ but violates σ.

Case 1 (σ is an FD):

If σ is unary, then the relation constructed in Lemma 4.1 is the desired counterexample. If σ is not unary, we can use a construction similar to that of Lemma 4.1. In this case the counterexample relation is the union of two relations r_0, r_1.

Let σ be $X \rightarrow A$. The first relation r_0 is a two-tuple relation with one tuple all x's and the other having x's only in the attributes that are functionally determined by X in the set Σ. The remaining positions of this second tuple are initially left blank.

The second relation r_1 contains the symbols 0,1,... (but not x) and is constructed so that the union of r_0 and r_1 has the right number of repetitions of the symbol 0 in r_1 to satisfy all u-ID's in Σ. The construction of r_1 parallels the Proof of Lemma 4.1. The only difference is that now the *counts* are the number of 0's *and* x's in the union of the two relations. When the correct number of blanks have been inserted in all columns, i.e. all columns in a strongly connected component have the same *count* and *count* increases with scc-number, then the blanks can be filled in as in the Proof of Lemma 4.1 and all u-ID's in Σ are satisfied.

Case 2 (σ is a u-ID):

Let σ be $C \supseteq D$. Repeat the construction in the Proof of Lemma 4.1, with the following modification: if the column for attribute A has s blank positions, fill in the blanks with the numbers 1 to s *if there is no black arc* (a,d) in G_Σ; otherwise, fill in the blanks with 1,...,s−1, x. The relation thus constructed satisfies the FD's in Σ, by the same argument as in the Proof of Lemma 4.1. To see that the u-ID's in Σ are also satisfied, observe that $A \supseteq B$ is violated iff either

(i) scc(a)>scc(b), or

(ii) scc(a)$\leq$scc(b), there is no black arc (a,d), and there is a black arc (b,d).

By the properties of G_Σ, this means there is no black arc (a,b), i.e. $A \supseteq B$ is not in Σ. Finally, it is clear that $C \supseteq D$ is violated.

See Figure 4-2 for an example of this construction. ∎

We remark that Theorem 4.1 leads easily to a *polynomial-time* algorithm for finite implication of FD's and u-ID's [44]. We will now use Theorem 4.1 to prove an analogue of Theorem 2.1, this time for *finite* implication of FD's and u-ID's. The notation is taken from Chapter 2.

Theorem 4.2: In each of the following two cases, (i),(ii),(iii) are equivalent:

FD Case:

i) $\Sigma \models_{\text{fin}} A_1 \ldots A_n \to A$.

ii) $E_\Sigma \models_{\text{fin}} \bigvee_{\tau \in \mathcal{T}^+(M_f)} \tau[x_1/a_1x, \ldots, x_n/a_nx] = ax$.

iii) $\mathcal{E}_\Sigma \models_{\text{fin}} \bigvee_{\tau \in \mathcal{T}^+(M_f)} \tau[x_1/\alpha_1, \ldots, x_n/\alpha_n] = \alpha$.

u-ID Case:

i) $\Sigma \models_{\text{fin}} B \subseteq A$.

ii) $E_\Sigma \models_{\text{fin}} \bigvee_{\tau \in \mathcal{T}^+(M_i)} a\tau = bx$.

iii) $\mathcal{E}_\Sigma \models_{\text{fin}} \bigvee_{\tau \in \mathcal{T}^+(M_i)} \tau[x/\alpha] = \beta$.

Proof: The implications (iii)$\Rightarrow$(ii), (ii)$\Rightarrow$(i) can be proved by the same argument as in the Proof of Theorem 2.1. The reason is that the constructions we give map finite counterexamples to finite counterexamples.

(i)$\Rightarrow$(iii): Suppose $\Sigma \models_{\text{fin}} \sigma$, where σ is an FD or u-ID. By Theorem 4.1, there is a *proof* of σ from Σ using the rules (*). Let z be the number of steps of such a proof. We show both the FD and the u-ID Cases by simultaneous induction on z.

Basis: $z=0$. The conclusion is straightforward.

Induction Step: We distinguish six cases, depending on the last rule which was applied to prove σ.

Rules 1,2 Straightforward.

Rule 3 This means the FD's $A_1...A_n \to B_k$, $k=1,...,m$, $B_1...B_m \to A$ can be proved from Σ (in less than z steps); Rule 3 is then applied to derive $A_1...A_n \to A$. By the induction hypothesis, $\mathcal{E}_\Sigma$ finitely implies $\bigvee_{\tau_k \in \mathcal{T}^+(M_f)} \tau_k[x_1/a_1x,...,x_n/a_nx] = b_kx$, $k=1,...,m$, and also $\mathcal{E}_\Sigma$ finitely implies $\bigvee_{\tau \in \mathcal{T}^+(M_f)} \tau[x_1/b_1x,...,x_m/b_mx] = ax$. Thus, $\mathcal{E}_\Sigma$ finitely implies $\bigvee_{\tau,\tau_1,...,\tau_m \in \mathcal{T}^+(M_f)} \tau[x_1/\tau_1[x_1/a_1x,...,x_n/a_nx], \ldots ,x_m/\tau_m[x_1/a_1x,...,x_n/a_nx]] = ax$, i.e. $\mathcal{E}_\Sigma \models_{\text{fin}} \bigvee_{\tau \in \mathcal{T}^+(M_f)} \tau[x_1/a_1x,...,x_n/a_nx] = ax$.

Rule 4 Straightforward.

Rule 5 Similar to Rule 3.

Rule 6 Now the dependencies $A_0 \to A_1$, $A_1 \supseteq A_2$,..., $A_{m-1} \to A_m$, $A_m \supseteq A_0$ (m odd) can be proved from Σ (in less than z steps); then by a cycle rule we derive $A_1 \to A_0$.

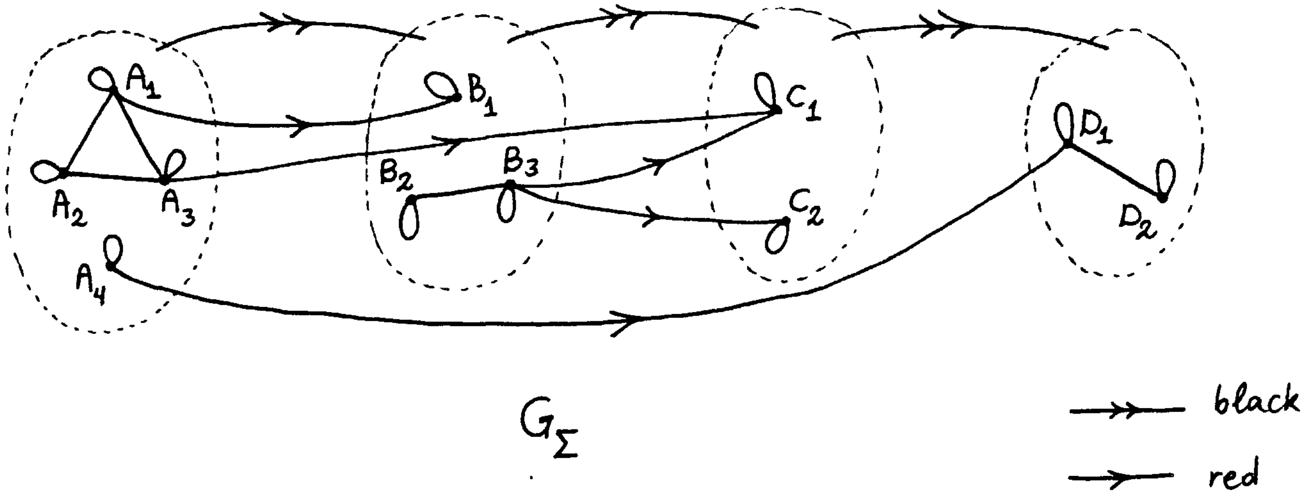
Let $\mathcal{A}$ be a finite model of $\mathcal{E}_\Sigma$. By the induction hypothesis $\mathcal{A}$ satisfies $\rho_0\alpha_0 = \alpha_1$, $\tau_1\alpha_1 = \alpha_2$,..., $\rho_{m-1}\alpha_{m-1} = \alpha_m$, $\tau_m\alpha_m = \alpha_0$, where $\rho_k \in \mathcal{T}^+(M_f)$, $\tau_k \in \mathcal{T}^+(M_i)$ (we write $\tau\alpha$ as a shorthand for $\tau[x/\alpha]$). We will show that there is some ρ' in $\mathcal{T}^+(M_f)$ such that $\mathcal{A}$ satisfies $\rho'\alpha_1 = \alpha_0$.

Observe first that $\mathcal{A}$ satisfies $\rho_0\tau_m\rho_{m-1}...\tau_3\rho_2\tau_1\alpha_1 = \alpha_1$ (concatenation denotes composition). By the commutativity conditions (5) of $\mathcal{E}_\Sigma$, $\rho_0\tau_m\rho_{m-1}...\tau_3\rho_2\tau_1 = \rho_0\rho_{m-1}...\rho_2\tau_m...\tau_3\tau_1$, so $\mathcal{A}$ satisfies $\rho_0\rho_{m-1}...\rho_2\tau_m...\tau_3\tau_1\alpha_1 = \alpha_1$. Now put $\rho_0\rho_{m-1}...\rho_2 = \rho$, $\tau_m...\tau_3\tau_1 = \tau$, $\tau_m...\tau_3\tau_1\alpha_1 = \alpha$.
We now have $\tau\alpha_1 = \alpha$, $\rho\alpha = \alpha_1$. We will argue from these two equations that there exists some ρ' in $\mathcal{T}^+(M_f)$ such that $\mathcal{A}$ satisfies $\rho'\alpha_1 = \alpha$. It will then follow, since $\rho_{m-1}...\rho_2\alpha = \alpha_0$, that $\mathcal{A}$ satisfies $\rho_{m-1}...\rho_2\rho'\alpha_1 = \alpha_0$.

Consider the set $K = \{\rho^k\alpha_1 : k \geq 0\}$ (ρ^k is ρ composed with itself k times). Since $\mathcal{A}$ is finite, K is finite, and therefore there exists a *least* integer q such that $\rho^q\alpha_1 = \rho^s\alpha_1$, for some s greater than q. We will first argue that $q=0$. Assume on the contrary that $q \geq 1$. By commutativity, $\tau\rho^q\alpha_1 = \rho^q\tau\alpha_1 = \rho^q\alpha = \rho^{q-1}\rho\alpha = \rho^{q-1}\alpha_1$, and similarly $\tau\rho^s\alpha_1 = \rho^{s-1}\alpha_1$. But this means $\rho^{q-1}\alpha_1 = \rho^{s-1}\alpha_1$, which contradicts the choice of q.

Since $q=0$, $\mathcal{A}$ satisfies $\alpha_1=\rho^s\alpha_1$, where $s>0$. But now $\alpha=\tau\alpha_1=\tau\rho^s\alpha_1=\rho^s\tau\alpha_1=\rho^{s-1}\rho\alpha=\rho^{s-1}\alpha_1$, i.e. $\mathcal{A}$ satisfies $\rho^{s-1}\alpha_1=\alpha$. This concludes the proof.

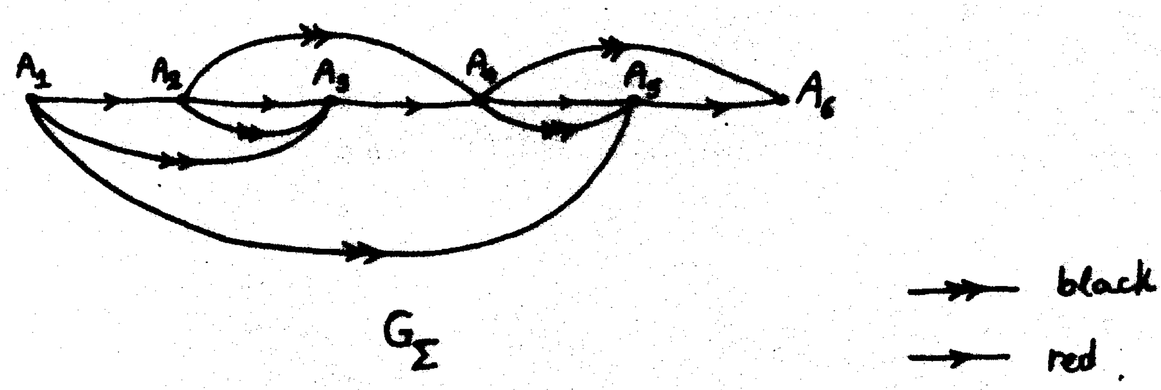
If a cycle rule is applied to derive a u-ID, we argue in an enirely analogous way. ∎



A_1	A_2	A_3	A_4	B_1	B_2	B_3	C_1	C_2	D_1	D_2
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1		0	1	1	
1			0	1	2		1	2	0	0
2			2	0	3		2	3	2	
3			3	2	0	0	0	0	3	
4			4	3	0	0	0	0	4	
5			5	4	4		0	4	5	
6			6	5	5		3	0	6	
7			7	6	6		4	0	7	
8			8	7	7		5	5	0	0
9			9	8	8		6	6	0	0
10			10	9	9		7	7	0	0
11			11	10	10		8	8	0	0

r

Figure 4-1: Construction of a finite counterexample relation



A_1	A_2	A_3	A_4	A_5	A_6
0	0	0	0	0	0
1	0	0	0	0	0
2	1	0	0	0	0
3	2	1	0	0	0
4	3	2	1	0	0
5	4	3	2	1	0
6	x	4	x	2	x

$$r \not\models A_1 \supseteq A_6$$

Figure 4-2: Relation that violates a u-ID

Chapter Five

Partition Dependencies

5.1 Preliminaries

Let D be a database scheme containing a single relation scheme $R[\mathcal{U}]$, $\mathcal{U}=\{A_1,...,A_u\}$. We can express database constraints as formulas of first-order predicate calculus with equality [32]. These formulas have a single relation symbol R of ARITY u which represents the relation R, and no function (or constant) symbols.

Specifically, let us call atomic formulas of the form $Rx_1...x_u$ *relational formulas* and atomic formulas $x=y$ *equalities*. A formula is *typed* iff there are disjoint classes (*types*) of variables such that
1. if $Rx_1...x_u$ appears in the formula, then x_k is of type k, $k=1,...,u$, and
2. if $x=y$ appears in the formula, then x,y have the same type.

Definition 5.1: An *embedded implicational dependency* (EID [34]) is a typed sentence of the form

$$\forall x_1...x_p.\ [(\varphi_1\wedge...\wedge\varphi_n)\Rightarrow\exists y_1...y_q.\ (\psi_1\wedge...\wedge\psi_m)],$$

where each φ_k is a relational formula, each ψ_k is either a relational formula or an equality between two of the x_k's, and each of the x_k's appears in one of the φ_k's.

Example 5.1:
(a) Let $\mathcal{U}=\{A_1,A_2,A,B\}$. The FD $A_1A_2\rightarrow A$ can be expressed as the EID

$$\forall x_1x_2xyx'y'.\ [(Rx_1x_2xy\wedge Rx_1x_2x'y')\Rightarrow x=x'].$$

(b) Let $\mathcal{U}=\{A,B,C\}$. The MVD $A\rightarrow\rightarrow B$ [62, 51] is equivalent to the EID

$$\forall zxyx'y'.\ [(Rzxy\wedge Rzx'y')\Rightarrow Rzxy'].$$

Now let r be a relation over a finite universe of attributes $\mathcal{U}$, and let σ be an EID. As one can easily observe, to decide whether $r\models\sigma$ we do not need to know the particular values appearing in r, but only the *equalities* between these values. As a matter of fact, all that is relevant about two tuples t,s of r is the set of attributes on which they agree. We can capture this information formally by considering, for each attribute A in $\mathcal{U}$, the *partition* π_A which is induced on the set of tuples of r by the values of r in column A: two tuples t,s of r are in the same block of π_A iff they agree on A. The set $\{\pi_A \mid A \in \mathcal{U}\}$ characterizes the EID's satisfied by r.

Although the above observation does not seem to take us very far regarding general EID's, it does lead to an elegant algebraic formulation of FD's [15, 60, 27]. Recall that partitions have a natural partial order $\leq$, and two natural binary operations $\bullet, +$: Given partitions π, π' of a set S,

$\pi \leq \pi'$ iff for every block x of π there is a block x' of π' such that $x \subseteq x'$.

$\pi \bullet \pi' = \{x \mid x = y \cap z \neq \emptyset, y \in \pi, z \in \pi'\}$.

$\pi + \pi' = \{x \mid$ a,b$\in$S are in x iff there is a sequence $x_0,\ldots,x_n$ such that $x_i \in \pi \cup \pi'$ for $i = 0,\ldots,n$, $a \in x_0$, $b \in x_n$, and $x_i \cap x_{i+1} \neq \emptyset$ for $i = 0,\ldots,n-1\}$

Notice that $\pi \bullet \pi'$ is the *coarsest common refinement of* π, π' (in the sense of $\leq$) and $\pi + \pi'$ is their *finest common generalization.* Also $\bullet, +$ are *associative, commutative* and *idempotent* (cf. Section 5.3).

With the above remarks, it is easy to see that an FD such as AB$\rightarrow$CD holds in relation r iff

$$\pi_A \bullet \pi_B \leq \pi_C \bullet \pi_D$$

or, equivalently,

$$\pi_A \bullet \pi_B = \pi_A \bullet \pi_B \bullet \pi_C \bullet \pi_D$$

or, still,

$$\pi_C \bullet \pi_D = \pi_A \bullet \pi_B + \pi_C \bullet \pi_D.$$

Thus, FD's can be expressed equationally using product and sum of partitions. It is then natural to investigate the expressive power of *general* equations one can write using $\bullet, +$.

Definition 5.2:

a. The set of *partition expressions over* $\mathcal{U}$, W($\mathcal{U}$), is the least set satisfying the following closure conditions:

1. $A \in W(\mathcal{U})$, for A in $\mathcal{U}$.

2. If $e, e' \in W(\mathcal{U})$, then $(e \bullet e')$, $(e + e')$ are in $W(\mathcal{U})$.

($\bullet, +$ are meant here as *uninterpreted operator symbols*)

b. A *partition dependency* (PD) is an equation $e = e'$, where $e, e' \in W(\mathcal{U})$.

The above definition gives the *syntax* of PD's. The *semantics* of PD's are given below:

Definition 5.3:

a. Let r be a relation over $\mathcal{U}$, S the set of tuples of r. For A in $\mathcal{U}$,

$\pi_A = \{x \mid t,s \in S$ are in x iff $t[A]=s[A]\}$.

Then L(r) is the set obtained by closing $\{\pi_A \mid A \in \mathcal{U}\}$ under product and sum of partitions.

b. Let $e \in W(\mathcal{U})$. The *meaning* of e in L(r), $\mu_r(e)$, is defined inductively as follows:

1. $\mu_r(A) = \pi_A$, A in $\mathcal{U}$.

2. $\mu_r(e \bullet e') = \mu_r(e) \bullet \mu_r(e')$,
   $\mu_r(e+e') = \mu_r(e) + \mu_r(e')$.

Relation r *satisfies* a PD $e=e'$ (notation: $r \models e=e'$) iff $\mu_r(e)=\mu_r(e')$.

Observe that L(r) is actually a *lattice* [28], generated by the set $\{\pi_A \mid A \in \mathcal{U}\}$. As a matter of fact, $r \models e=e'$ iff L(r) satisfies the equation $e=e'$ (with A interpreted as π_A, $A \in \mathcal{U}$).

From Definition 5.3, we see that we can use the formalism of PD's to express an FD $AB \rightarrow CD$ as the PD $A \bullet B = A \bullet B \bullet C \bullet D$. Clearly $r \models AB \rightarrow CD$ iff $r \models A \bullet B = A \bullet B \bullet C \bullet D$ (here and in the sequel we omit parentheses from PD's wherever possible, for the sake of clarity). Partition dependencies of the above form, which are equivalent to FD's, are of special interest; we call them FPD's.

In the remainder of this Chapter, we investigate various questions concerning PD's. Section 5.2 deals with the *expressive power* of PD's, and compares PD's to EID's from this point of view. In Section 5.3 we give a polynomial-time algorithm for the *implication problem* for PD's. Finally, in Section 5.4 we present a polynomial-time test for *consistency* of a database with a set of PD's.

5.2 Expressive Power

We want to study what properties of a relation r can by expressed using sets of PD's. From the definitions of $\bullet, +$ and Definition 5.3 it it easy to see the following:

1. $r \models C = A \bullet B$ iff for any tuples $t,s \in r$,
   $t[C]=s[C]$ iff $t[A]=s[A]$ *and* $t[B]=s[B]$.

2. $r \models C = A + B$ iff for any tuples $t,s \in r$,

$t[C]=s[C]$ iff there is a sequence $s_0,\ldots,s_n$ of tuples of r with $t=s_0$, $s_n=s$, and for $i=0,\ldots,n-1$, $s_i[A]=s_{i+1}[A]$ *or* $s_i[B]=s_{i+1}[B]$.

From observation (2) above, one sees that *symmetric transitive closure* can be expressed by a PD, as follows:

Example 5.2: Consider a relation r representing an undirected graph. This relation has three attributes: HEAD, TAIL and COMPONENT. For every edge {a,b} in the graph we have in the relation tuples abc, bac, aac, bbc, where c is a number which could vary with {a,b}. These are the only tuples in r. We would like to express that: for each tuple t of r, t[COMPONENT] is the *connected component* in which the arc (t[HEAD], t[TAIL]) belongs. We can do this by insisting that r satisfies the PD COMPONENT = HEAD + TAIL.

We now want to compare the expressive power of PD's to that of previously studied database constraints, namely EID's [34]. Let us say that an EID σ *is expressed* by a set E of PD's iff for any relation r, $r \models \sigma$ iff $r \models E$. From the algebraic properties of $\bullet$, the PD $C=A\bullet B$ is equivalent to $C=C\bullet A\bullet B \wedge A\bullet B=C\bullet A\bullet B$, and therefore it is expressed by the set $\{C \rightarrow AB, AB \rightarrow C\}$. However, because of Example 5.2 above it should come as no surprise [4] that the PD $C=A+B$ cannot be expressed by any set of EID's:

Theorem 5.1: Let $\mathcal{U}=ABC$; the PD $C=A+B$ cannot be expressed by any set of first-order sentences.

Proof: Let Σ be a set of first-order sentences (with a single ternary relation symbol R as the only non-logical symbol) which expresses $C=A+B$. For $k \geq 1$, let φ_k be the following first-order formula, with free variables t,s:

"$t[C]=s[C]$ and there is *no* sequence $s_0,\ldots,s_k$ such that $t=s_0$, $s_k=s$, and for $i=0,\ldots,k-1$, $s_i[A]=s_{i+1}[A]$ or $s_i[B]=s_{i+1}[B]$"

(it is easy to see how to write φ_k without tuple variables). Observe that the relation r in Figure 5-1 (with t,s as indicated) is a *model* for $\Sigma \cup \{\varphi_k\}$: $r \models C=A+B$ so $r \models \Sigma$, and clearly $r \models \varphi_k$. Thus, any finite subset of $\Sigma'=\Sigma \cup \{\varphi_k \colon k \geq 1\}$ has a model, and thus by the Compactness Theorem [32] Σ' has a model, say r'. But this is a contradiction, since r' satisfies Σ and thus r' satisfies $C=A+B$, and on the other hand $r' \models \varphi_k$ *for all* $k \geq 1$ and therefore it does not satisfy $C=A+B$. ∎

On the other hand, an EID as simple as an MVD cannot be expressed by PD's:

Theorem 5.2: Let $\mathcal{U}=ABC$; the MVD $A\to\to B$ cannot by expressed by any set of PD's.

Proof: Let E be a set of PD's which expresses $A\to\to B$ (see Example 5.1 for the meaning of this MVD). Referring to Figure 5-2, relation r_1 satisfies $A\to\to B$, so $L(r_1)\models E$. On the other hand, relation r_2 does not satisfy $A\to\to B$, so $L(r_2)$ does not satisfy E. But this is a contradiction, because $L(r_1)$, $L(r_2)$ are *isomorphic*, and thus they satisfy exactly the same PD's. ∎

5.3 The Implication Problem

Given a finite set E of PD's and a PD δ, we want to know if $E\models\delta$, i.e. if δ holds in every relation that satisfies E. We also want to know if $E\models_{fin}\delta$, i.e. if δ holds in every *finite* relation that satisfies E. We first observe that these questions can be approached as implication problems for *lattices.*

Lemma 5.1:
a. $E\models\delta$ iff $E\models_{lat}\delta$, i.e. iff δ holds in every *lattice* that satisfies E.
b. $E\models_{fin}\delta$ iff $E\models_{lat,fin}\delta$, i.e. iff δ holds in every *finite lattice* that satisfies E.

Proof:
a. ($\Leftarrow$): Suppose $E\models_{lat}\delta$, and let r be a relation that satisfies E. Then $L(r)\models E$, so δ holds in L(r), and thus r satisfies δ.

($\Rightarrow$): Suppose $E\models\delta$, and let L be a lattice satisfying E. By the Representation Theorem for lattices, [28, 66], we may take the elements of L to be partitions of some set X. Thus, each A in $\mathcal{U}$ is interpreted in L as a partition π_A of X (and, of course, $\bullet,+$ in L are partition product and sum respectively). Now consider a relation r over $\mathcal{U}$ containing a tuple t_i for each element i of X (these are the only tuples in r), where $t_i[A]=t_j[A]$ iff i,j are in the same block of π_A, A in $\mathcal{U}$. Clearly r satisfies exactly the same PD's as L. Thus $r\models E$, so by the hypothesis $r\models\delta$, and therefore $L\models\delta$.

b. ($\Leftarrow$): Observe, in the proof of the "if" direction of (a), that if r is *finite* then L(r) is also *finite.*

($\Rightarrow$): Observe, in the proof of the "only if" direction of (a), that if L is *finite* then the set X can be taken to be *finite*, by the Representation Theorem for finite lattices [56]. Then the relation r is also *finite.* ∎

Now $E \models_{lat} \delta$ can be viewed as a (uniform) *word problem*, since a set with two binary operations $\bullet, +$ is a lattice iff the following set of axioms (LA) is satisfied [28]:

1. $x+x=x$, $x \bullet x=x$ (idempotency)

2. $x+y=y+x$, $x \bullet y=y \bullet x$ (commutativity)

3. $x+(y+z)=(x+y)+z$, $x \bullet (y \bullet z)=(x \bullet y) \bullet z$ (associativity)

4. $x+(x \bullet y)=x$, $x \bullet (x+y)=x$ (absorption)

I.e., $E \models_{lat} \delta$ iff δ is implied from $E \cup LA$. We are going to show that $\models_{lat,fin}$ is *equivalent* to $\models_{lat}$, so $\models_{lat,fin}$ can also be viewed as a word problem.

In particular, let δ_σ be the FPD corresponding to an FD σ (δ_σ is $A=A \bullet B$ if σ is $A \rightarrow B$), and let E_Σ be the set of FPD's corresponding to a set of FD's Σ. Since $r \models \sigma$ iff $r \models \delta_\sigma$, $\Sigma \models \sigma$ iff $E_\Sigma \models \delta_\sigma$. Thus, the implication problem for FD's can be reduced, in a straightforward way, to the (uniform) *word problem for idempotent commutative semigroups* (structures with a single associative, commutative and idempotent operator). On the other hand, since $X=Y$ is equivalent to $X=X \bullet Y \wedge Y=Y \bullet X$, we can also reduce the above word problem to the implication problem for FD's.

We now present a polynomial-time algorithm for the (finite) implication problem for PD's. Suppose we are given a set E of PD's, and a PD $e=e'$: by Lemma 5.1, it suffices to test if $E \models_{lat} e=e'$ ($E \models_{lat,fin} e=e'$).

Consider the set $W(\mathcal{U})$ of partition expressions over $\mathcal{U}, \bullet, +$: we define several binary relations on $W(\mathcal{U})$. First, define $\leq_{id}$ (*identically* less-than-or-equal) inductively as follows:

1. $A \leq_{id} A$, A in $\mathcal{U}$.

2. if $p \leq_{id} r$, $q \leq_{id} r$ then $p+q \leq_{id} r$.

3. if $p \leq_{id} r$ *or* $q \leq_{id} r$ then $p \bullet q \leq_{id} r$.

4. if $r \leq_{id} p$, $r \leq_{id} q$ then $r \leq_{id} p \bullet q$.

5. if $r \leq_{id} p$ *or* $r \leq_{id} q$ then $r \leq_{id} p+q$.

(The intended meaning of $\leq_{id}$ is that $p \leq_{id} q$ iff every lattice satisfies $p \leq q$, no matter how the A's in $\mathcal{U}$ are interpreted).

The relation $\leq_{id}$ is reflexive and transitive [28, 65]. Also, if $p_1 \leq_{id} q_1$, $p_2 \leq_{id} q_2$, then $p_1 + p_2 \leq_{id} q_1 + q_2$ and $p_1 \bullet p_2 \leq_{id} q_1 \bullet q_2$.

Now define $=_{id}$ as follows: $p =_{id} q$ iff both $p \leq_{id} q$ and $q \leq_{id} p$.

The relation $=_{id}$ is an *equivalence relation*, and in particular it is a *congruence*: i.e., if $p_1 =_{id} q_1$, $p_2 =_{id} q_2$, then $p_1 + p_2 =_{id} q_1 + q_2$ and $p_1 \bullet p_2 =_{id} q_1 \bullet q_2$. Thus, one can define $\bullet, +$ on the set of *equivalence classes* of $=_{id}$. The structure obtained this way is a *lattice* [28, 65].

We now capture the effect of E. Define the following relation $\rightarrow\rightarrow_E$ on $W(\mathcal{U})$: $p \rightarrow\rightarrow_E q$ iff q can be obtained from p as follows: for $i = 0, \ldots, n$, substitute w_i for some (zero or more) occurences of z_i, where $z_i = w_i$ ($w_i = z_i$) is in E. It is easily verified that $\rightarrow\rightarrow_E$ is a congruence.

Now define $\leq_E$ as the *sum* of $\leq_{id}$, $\rightarrow\rightarrow_E$: $p \leq_E q$ iff there is a sequence of expressions $s_0, \ldots, s_n$ such that $p = s_0$, $s_n = q$, and for $i = 0, \ldots, n-1$, $s_i \leq_{id} s_{i+1}$ *or* $s_i \rightarrow\rightarrow_E s_{i+1}$.

It is easy to see that $\leq_E$ is reflexive and transitive. Also if $p_1 \leq_E q_1$, $p_2 \leq_E q_2$, then $p_1 + p_2 \leq_E q_1 + q_2$ and $p_1 \bullet p_2 \leq_E q_1 \bullet q_2$ (because both $\leq_{id}$ and $\rightarrow\rightarrow_E$ have this property [36]).

Finally, define $=_E$ as follows: $p =_E q$ iff both $p \leq_E q$ and $q \leq_E p$.

The relation $=_E$ is an equivalence relation, and moreover it is a congruence. One can further observe that the equivalence classes of $=_E$ form a lattice L_E under the induced $\bullet, +$: just check the axioms LA, e.g. $p + p =_E p$ because $p + p =_{id} p$, and in general if $p =_{id} q$ then $p =_E q$. Note that L_E satisfies a PD $p = q$ iff $p =_E q$ ($A \in \mathcal{U}$ is interpreted in L_E as the equivalence class of A).

We now show that the relation $=_E$ captures the PD's (finitely) implied by E:

Lemma 5.2: The following statements are equivalent:

a. $e =_E e'$

b. $E \models_{lat} e = e'$

c. $E \models_{lat,fin} e = e'$

Proof: Observe that, from the way $\leq_{id}$ and $\leq_E$ were defined, if $e \leq_E e'$ then $e \leq e'$ in every lattice satisfying E (where $\leq$ is the partial order of the lattice). Thus, (a)$\Rightarrow$(b). To prove (b)$\Rightarrow$(a), recall that L_E satisfies a PD $p = q$ iff $p =_E q$. Thus, if $e \neq_E e'$ then L_E does not satisfy $e = e'$, whereas it satisfies E; i.e., L_E is a *counterexample* to $E \models_{lat} e = e'$.

We now show the equivalence of (b),(c). The direction (b)$\Rightarrow$(c) is obvious. To prove the converse, we adapt an argument of [30] (see also [28]), originally given for the special case $E=\emptyset$.

Suppose E does not imply $e=e'$; we will show that there is a *finite* lattice which satisfies E but violates $e=e'$. Let $\{A_i \mid i=1,...,n\}$ be the set of attributes appearing in E,e,e', and let V be the set of all partition expressions (over the A_i's) of complexity at most as high as the maximum complexity of e,e' and the expressions in E (complexity can be measured by the number of instances of $\cdot, +$). Note that V is finite, since E is finite.

Consider now the subset L of L_E consisting of all finite products of the equivalence classes (under $=_E$) of elements of V, together with the equivalence class of $A_1+...+A_n$. It is not hard to verify that L is a *sublattice* of L_E. But by the equivalence of (a),(b) $e \neq_E e'$, so L satisfies E and violates $e=e'$. Since L is also obviously finite, we are done. ∎

We can now prove our main result:

Theorem 5.3: There is a polynomial-time algorithm for the (finite) implication problem for PD's.

Proof: By Lemmas 5.1, 5.2, it is sufficient to describe a polynomial-time algorithm to test, given E,e,e', whether $e \leq_E e'$.

Let V be the set of all subexpressions of e,e', and of the expressions appearing in E. The following algorithm constructs a set Γ of directed arcs over V such that, whenever $(p,q)\in\Gamma$, $p \leq_{id} q$ or $p \rightarrow\rightarrow_E q$:

begin

$\Gamma \leftarrow \emptyset$

repeat until no new arcs are added

1. Add (A,A), $A \in \mathcal{U}$
2. **if** $(p,r) \in \Gamma$, $(q,r) \in \Gamma$, $p+q \in V$

   then add $(p+q,r)$
3. **if** $(p,r) \in \Gamma$ *or* $(q,r) \in \Gamma$, $p \bullet q \in V$

   then add $(p \bullet q,r)$
4. **if** $(r,p) \in \Gamma$, $(r,q) \in \Gamma$, $p \bullet q \in V$

   then add $(r,p \bullet q)$
5. **if** $(r,p) \in \Gamma$ *or* $(r,q) \in \Gamma$, $p+q \in V$

   then add $(r,p+q)$
6. Add (z,w),(w,z), where $z=w$ in E
7. **if** $(p,r) \in \Gamma$, $(r,q) \in \Gamma$

   then add (p,q)

end

end

Observe that Steps 1-5 in the above algorithm mirror the definition of $\leq_{id}$.

We will now prove the following

<u>Claim:</u> For $p,q \in V$, $p \leq_E q$ iff $(p,q) \in \Gamma$.

Clearly, the Theorem follows from the Claim: to test if $e \leq_E e'$, construct the digraph (V,Γ) and check if it has an arc from e to e'. This can be done in polynomial time.

Proof of Claim:

($\Leftarrow$): Straightforward.

($\Rightarrow$): We first give a set of *rewrite rules* [41] for $\leq_E$:

1. $x+x \rightarrow\rightarrow x$

2. $x \bullet y \rightarrow\rightarrow x$

3. $y \bullet x \rightarrow\rightarrow x$

4. $x \rightarrow\rightarrow x \bullet x$

5. $x \rightarrow\rightarrow x+y$

6. $x \rightarrow\rightarrow y+x$

7. $z \rightarrow\rightarrow w$, where $z=w$ ($w=z$) is in E

Observe, regarding Rules 5,6, that y can be an arbitrary expression.

An easy induction shows that, if $p \leq_{id} q$, then p can be rewritten as q using Rules 1-6. By the definition of $\leq_E$, if $p \leq_E q$ then there is a sequence of expressions $s_0,\ldots,s_n$ such that $p=s_0$, $s_n=q$, and for $i=0,\ldots,n-1$, $s_i \rightarrow\rightarrow s_{i+1}$, i.e. s_{i+1} is obtained from s_i by rewriting a subexpression of s_i according to one of the Rules 1-7. We call such a sequence a *proof* that $p \leq_E q$.

Now we define a relation $\prec$ on pairs of expressions:

$(p_1,q_1) \prec (p_2,q_2)$ iff $p_1 \leq_E q_1$, $p_2 \leq_E q_2$, and either
(i) the shortest proof that $p_1 \leq_E q_1$ is shorter than the shortest proof that $p_2 \leq_E q_2$, or
(ii) the shortest proofs that $p_1 \leq_E q_1$, $p_2 \leq_E q_2$ have the same length, and p_1 is a proper subexpression of p_2, q_1 is a proper subexpression of q_2.

Clearly $\prec$ is well-founded. We proceed by induction on $\prec$.

Basis: There is a proof that $p \leq_E q$ of length 0. Then p is identical to q, and $(p,q) \in \Gamma$.

Induction Step: Let $p,q \in V$, and assume that the Claim holds for $p',q' \in V$ whenever $(p',q') \prec (p,q)$. We will show that the Claim holds for (p,q). Let $s_0,\ldots,s_n$, $n>0$, be a shortest proof that $p \leq_E q$.

Case 1: For $i=0,...,n-1$, s_{i+1} is obtained from s_i by rewriting a *proper* subexpression of s_i according to Rules 1-7. Then $p=p_1\theta p_2$, $q=q_1\theta q_2$ ($\theta\in\{\bullet,+\}$), where $p_i\leq_E q_i$ via proofs at most as long as the proof that $p\leq_E q$, and p_i (q_i) is a proper subexpression of p (q). Thus $(p_i,q_i)\prec(p,q)$, and furthermore $p_i,q_i\in V$, so by the induction hypothesis $(p_i,q_i)\in\Gamma$. It then easily follows that $(p,q)\in\Gamma$.

Case 2: For some i, $0\leq i\leq n-1$, s_i is rewritten into s_{i+1} according to one of the Rules 1-7.

Case 2a: For some i as above, the Rule used is Rule 7. This means p is rewritten to z, $z=w$ ($w=z$) is in E, and w is rewritten to q. Then clearly $(p,z)\prec(p,q)$, and since $z\in V$, by the induction hypothesis $(p,z)\in\Gamma$. Similarly $(w,q)\in\Gamma$. It follows that $(p,q)\in\Gamma$.

Case 2b: For any i as above, the Rule used is one of the Rules 1-6. We consider the *least* such i, and we distinguish cases according to which Rule was used to rewrite s_i to s_{i+1}.

<u>Rule 1</u> This means $p=p_1+p_2$, p_1 rewrites to r, p_2 rewrites to r, and r rewrites to q. Then $p_i\leq_E q$ via proofs shorter than the proof that $p\leq_E q$, so $(p_i,q)\prec(p,q)$. Also $p_i\in V$, so by the induction hypothesis $(p_i,q)\in\Gamma$. It follows that $(p,q)\in\Gamma$.

<u>Rule 2</u> This means $p=p_1\bullet p_2$, p_1 rewrites to r, r rewrites to q. Then $p_1\leq_E q$ via a proof shorter than the proof that $p\leq_E q$, so $(p_1,q)\prec(p,q)$. Also $p_1\in V$, so by the induction hypothesis $(p_1,q)\in\Gamma$. It follows that $(p,q)\in\Gamma$.

<u>Rule 3</u> Similar to Rule 2.

<u>Rule 4</u> Now p rewrites to r, and Rule 4 rewrites r to $r\bullet r$. Observe that the expression $r\bullet r$ will not be rewritten subsequently using Rules 2,3, because in that case we could shorten the proof that $p\leq_E q$ (however, either *subexpression* of $r\bullet r$ may be rewritten). Moreover, if at some later point Rule 5 is applied to rewrite the whole expression s_i as s_i+y, then s_i+y will not be rewritten subsequently using Rule 1. Thus, the expression q eventually obtained is built up, using Rules 4,5,6, by some expressions r_j, $j=1,...,m$, such that r rewrites to r_j for all j, and by some completely new expressions y_k, $k=1,...,m'$, which were introduced by Rules 5,6. Now clearly $(p,r_j)\prec(p,q)$ and $r_j\in V$, so by the induction hypothesis $(p,r_j)\in\Gamma$. It then follows by an easy induction on the structure of q that $(p,q)\in\Gamma$.

<u>Rules 5,6</u> Similar to Rule 4.

This concludes the Proof of the Claim, so we are done. ∎

Since inference of FD's can be seen as a special case of inference of PD's, the problem is actually *polynomial-time complete* [63]. However, in the special case where E is empty [28, 65] it can be solved in *logarithmic space* [40], as we now outline. By Lemma 3, it suffices to describe how to recognize $\leq_{id}$ in logarithmic space.

First, observe the following:

1. $A \leq_{id} A'$ iff A is identical to A', A, A' in $\mathcal{U}$.

2. $A \leq_{id} p' \cdot q'$ iff $A \leq_{id} p'$ *and* $A \leq_{id} q'$, A in $\mathcal{U}$.

3. $A \leq_{id} p' + q'$ iff $A \leq_{id} p'$ *or* $A \leq_{id} q'$, A in $\mathcal{U}$.

4. $p \cdot q \leq_{id} A'$ iff $p \leq_{id} A'$ *or* $q \leq_{id} A'$, A' in $\mathcal{U}$.

5. $p \cdot q \leq_{id} p' \cdot q'$ iff $p \cdot q \leq_{id} p'$ *and* $p \cdot q \leq_{id} q'$.

6. $p \cdot q \leq_{id} p' + q'$ iff $p \leq_{id} p' + q'$ *or* $q \leq_{id} p' + q'$ *or* $p \cdot q \leq_{id} p'$ *or* $p \cdot q \leq_{id} q'$.

7. $p + q \leq_{id} e'$ iff $p \leq_{id} e'$ *and* $q \leq_{id} e'$.

In each of the above cases, the "if" direction is trivial. The "only-if" direction follows in Case 5 because
$p' \cdot q' \leq_{id} p'$ and $p' \cdot q' \leq_{id} q'$, and in Case 7 because $p \leq_{id} p + q$, $q \leq_{id} p + q$. In the remaining cases, the "only-if" direction follows by the definition of $\leq_{id}$.

The above observation gives a *recursive* algorithm to test, given e, e', whether $e \leq_{id} e'$. We now describe how to implement this recursion using only logarithmic auxiliary space.

First, note that the results of intermediate recursive calls need not be stored. For example, consider Case 7: if the recursive call for $p \leq_{id} e'$ returns *false*, then we immediately return *false*; otherwise, we return the result of the recursive call for $q \leq_{id} e'$.

We will also argue that we do not need to store the arguments of previous recursive calls. Thus, all we need to have in storage at any particular point is the arguments of the recursive call which is being evaluated. Since these arguments are *subexpressions* of e, e', we can just have two *pointers* to the appropriate places in the input, and this only takes logarithmic space.

We will now describe how, given two pointers to two subexpressions p, p' of e, e' respectively, we

can find the next recursive call to be evaluated, using only logarithmic additional space. We assume that e,e′ are represented (in the standard way) as binary trees, so that, given a pointer to a node u, we can find a pointer to the father (right son, left son) of u.
We use two auxiliary pointers α,α', initialized to the root of e,e′ respectively. Let $C(e,e')$ be the set of recursive calls generated from the call $e \leq_{id} e'$ ($C(e,e')$ contains either two or four members, depending on which of Cases 2-7 is the relevant one). We will show that we can determine which member of $C(e,e')$ eventually gives rise to the call $p \leq_{id} p'$, using only logarithmic additional space. If this member of $C(e,e')$ turns out to be the call $e_1 \leq_{id} e_1'$, we set the pointers α,α' to the expressions e_1,e_1' respectively and we repeat with $C(e_1,e_1')$. Continuing in this way, we will eventually find e_i,e_i' such that the call $p \leq_{id} p'$ is in $C(e_i,e_i')$. We can then easily determine the next call to be evaluated.

Finally, note that, to determine which member of $C(e,e')$ eventually gives rise to the call $p \leq_{id} p'$, we only need to know whether p (p′) is in the left or in the right subtree of e (e′). This can be found be walking the tree representing e in a depth-first fashion, until we encounter p. This walk can be done using only logarithmic additional space, because all we need to remember is the node v which is currently visited and the node w which was visited immediately before v: if w is the father of v, we next visit the left son of v; if w is the left son of v, we next visit the right son of v; if w is the right son of v, we next visit the father of v.

5.4 Testing Satisfaction

Given a database d over $\mathcal{U}$ and a set of PD's E, we want to test if d is *consistent* with E, i.e if there is a *weak instance* w for d satisfying E. Recall that a relation w over $\mathcal{U}$ is a weak instance for d iff every tuple of relation R[U] of d appears in the projection of w on U. Weak instances have been proposed as a way to model incomplete information in databases [38, 64]. Given a database d and a set of FD's E, we can test if d has a weak instance satisfying E in polynomial time [38]. We now show how this test can be generalized to arbitrary PD's.

First, we replace E by a set E′ of PD's of the form $C=A \bullet B$ or $C=A+B$, where A,B,C are attributes from a universe $\mathcal{U}'$ containing $\mathcal{U}$: this is done by (recursively) replacing $X=Y \bullet Z$ by the PD's $X=C$, $Y=A$, $Z=B$, $C=A \bullet B$, where A,B,C are *new* attribute names. It is easy to check that there is a weak instance for d satisfying E iff there is a weak instance for d satisfying E′.

Let us denote by $p \rightarrow q$, where p,q are partition expressions, the PD $p = p \cdot q$. This slight abuse of notation is consistent, since the FPD $X \rightarrow Y$ is actually *equivalent* to the FD $X \rightarrow Y$. Now a PD $C = A \cdot B$ in E' can be replaced by the FPD's $C \rightarrow AB$, $AB \rightarrow C$, and a PD $C = A + B$ in E' can be replaced by the PD's $A + B \rightarrow C$, $C \rightarrow A + B$. Furthermore, the PD $A + B \rightarrow C$ can be replaced by the FPD's $A \rightarrow C$, $B \rightarrow C$. We now have a set F consisting of FPD's and of PD's of the form $C \rightarrow A + B$, and it is obvious that there is a weak instance for d satisfying E' iff there is a weak instance for d satisfying F.

Now compute (using the algorithm of the previous Section) all *consequences* of F of the form $A \rightarrow B$, A,B in $\mathfrak{U}'$, and add them to F. Furthermore, if now F contains $A \rightarrow B$ and $C \rightarrow A + B$, replace $C \rightarrow A + B$ by $C \rightarrow B$. Let F' be the set of FPD's in F. The crucial fact is given in the following

Lemma 5.3: There is a weak instance for d satisfying F iff there is a weak instance for d satisfying F'.

Proof: The "only if" direction is obvious. For the converse, let w be a weak instance for d satisfying F'. Suppose some PD $C \rightarrow A + B$ in F is violated by tuples t_1, t_2 of w, where $t_1[ABC] = a_1b_1c$, $t_2[ABC] = a_2b_2c$, $a_1 \neq a_2$, $b_1 \neq b_2$. We can remedy this violation by adding to w a tuple s such that $s[AB] = a_1b_2$. To make sure that the relation w_1 obtained still satisfies F', let $A^+ = \{X \mid F' \models A \rightarrow X\}$, $B^+ = \{X \mid F' \models B \rightarrow X\}$: we make $s[A^+] = t_1[A^+]$, $s[B^+] = t_2[B^+]$, and fill in the rest of the attributes of s with distinct new values (not appearing in w). To argue that this is indeed possible, observe first that B is not in A^+ and A is not in B^+ (otherwise $C \rightarrow A + B$ would not appear in F). We also have to make sure that, if $Q \in A^+$ and $Q \in B^+$, then $t_1[Q] = t_2[Q]$. But if Q appears in both A^+ and B^+ we have $F' \models A \rightarrow Q$, $F' \models B \rightarrow Q$, so since $C \rightarrow A + B$ is in F we have $F \models C \rightarrow Q$, and therefore $C \rightarrow Q$ is in F'. This implies that $t_1[Q] = t_2[Q]$, since $t_1[C] = t_2[C]$ and w satisfies F'.

We now repeat the above argument, starting with w_1, to obtain relations w_2, w_3 and so on. The relation w_ω obtained after an infinite number of steps is a weak instance for d satisfying E', because any violation of some PD $C \rightarrow A + B$ appearing at any stage has been taken care of at some later stage. ∎

We can now prove the main result:

Theorem 5.4: There is a polynomial-time algorithm to test whether a given database d is consistent with a set E of PD's.

Proof: Using the polynomial-time algorithm for inference of PD's given in Section 5.3, we can construct the set F′. By Lemma 5.3, we can then use the algorithm of [36] to test if d is consistent with F′. ∎

Observe that the weak instance constructed in the Proof of Lemma 5.3 is in general *infinite*. The problem of testing existence of a *finite* weak instance is open.

r:

	A	B	C
t:	1	2	0
	3	2	0
	3	4	0
	5	4	0
		⋮	
	k-1	k	0
	k+1	k	0
s:	k+1	k+2	0

$r \models \Sigma, \varphi_k$

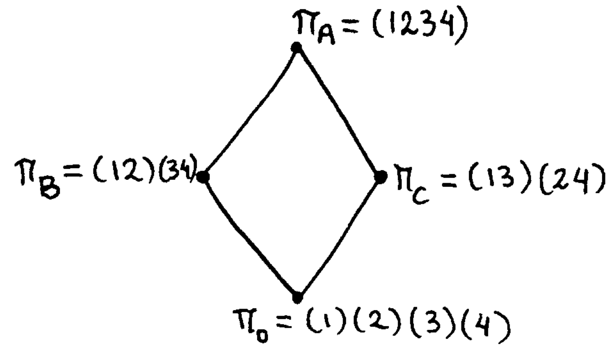
(k even)

Figure 5-1: A model for φ_k

r_1:

	A	B	C
1:	a	b_1	c_1
2:	a	b_1	c_2
3:	a	b_2	c_1
4:	a	b_2	c_2

$r_1 \models A \twoheadrightarrow B$

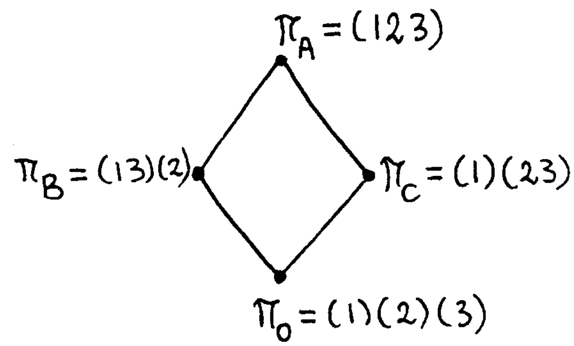


$L(r_1)$

r_2:

	A	B	C
1:	a	b_1	c_1
2:	a	b_2	c_2
3:	a	b_1	c_2

$r_2 \not\models A \twoheadrightarrow B$



$L(r_2)$

Figure 5-2: MVD's are not expressible by PD's

Chapter Six

Directions for Further Investigation

Extending the Equational Approach

Of course, the most obvious question is whether our equational formulation of FD's and IND's can be extended to more general dependencies. We outline some partial results we have at this point, which indicate that such an extension is indeed possible.

Recall that an embedded implicational dependency (EID) is a typed sentence of the form

$$\forall x_1 \dots x_p . [(\varphi_1 \wedge \dots \wedge \varphi_n) \Rightarrow \exists y_1 \dots y_q . (\psi_1 \wedge \dots \wedge \psi_m)],$$

where each φ_k is a relational formula, each ψ_k is either a relational formula or an equality between two of the x_k's, and each of the x_k's appears in one of the φ_k's (cf. Section 5.1). If all the ψ_k's are relational formulas, we have a *tuple generating dependency* (TGD); if all the ψ_k's are equalities, we have an *equality generating dependency* (EGD) [10, 11, 34].

Every EID is obviously equivalent to the conjunction of a TGD and an EGD. Furthermore, it can be shown that every EGD is equivalent to a conjunction of FD's and TGD's [11]. The question then is whether we can have an equational formulation of FD's and TGD's.

Let $\mathfrak{U} = \{A,B,C\}$ and consider the MVD $A \rightarrow\rightarrow B$ (cf. Example 5.1). We can formulate it as the sentence

$$\forall x_1 x_2 . [a(x_1) = a(x_2) \Rightarrow \exists y . (a(y) = a(x_1) \wedge b(y) = b(x_1) \wedge c(y) = c(x_2))].$$

Here x_1, x_2, y are variables ranging over tuples; see Section 1.3. Now Skolemization suggests transforming this MVD into an *equational implication*

$$ax_1 = ax_2 \Rightarrow (aix_1x_2 = ax_1 \wedge bix_1x_2 = bx_1 \wedge cix_1x_2 = cx_2)$$

In this way, we can transform any TGD into an equational implication. In fact, we can even relax the typedness restriction, to obtain a class of constraints which properly includes IND's: specifically, it suffices if only the part of the sentence consisting of the φ_k's is typed.

We can go even further and transform these equational implications into equations. We illustrate how this is done with the implication

$$ax_1 = ax_2 \Rightarrow aix_1x_2 = ax_1.$$

This can be transformed into the set of equations

$$aix_1x_2 = f_a x_1 x_2 a x_1 a x_2$$
$$f_a x_1 x_2 xx = ax_1,$$

where f_a is a new function symbol of ARITY 4.

The above equational formulation of TGD's can be used to prove a generalization of Theorem 2.1, for implication of TGD's from FD's and TGD's (i.e., we actually generalize the IND Case of Theorem 2.1). The proof uses the same ideas as the proof of Theorem 2.1. Unfortunately, the proof of the FD Case does not generalize, because the inductive argument for the completeness part depends critically on the fact that Skolem functions have only one argument (which only happens in the case of IND's).

Designing Normal Form Schemas

An active area of research in logical database design is concerned with canonical representations of the database schema, which avoid potential *update anomalies* (i.e. updates that can result in inconsistent data), and minimize *data redundancy*. Several such representations have been proposed and analyzed, assuming that the only integrity constraints of the database schema are FD's. The general idea is that the database schema should be in a certain *normal form* [22, 7, 62, 51], i.e. certain restrictive conditions should be satisfied by the FD's of the schema and their logical consequences. Given a universe $\mathcal{U}$ of attributes and a finite set Σ of FD's, one can construct a database schema satisfying such restrictions [12, 6]. These algorithms are based on efficient solutions of the implication problem.

An interesting question is to investigate normal forms in the presence of FD's and IND's (cf. [33]). Eventually one would hope to extend the known schema synthesis algorithms to incorporate IND's of some restricted form (for example, unary IND's). The insights we have gained on the implication problem can potentially be useful for this investigation.

Query Equivalence in the Presence of IND's

The problem of optimizing queries has received a lot of attention, because of its central role in all relational database implementations [62]. Given a query Q, the goal is to design an *equivalent* query

Q' which can be processed as efficiently as possible (i.e. contains a minimum number of instances of expensive operators, such as join). Since equivalence of two queries is a data dependency, the problem of testing equivalence of queries in the presence of dependencies can be approached with the standard tools for implication problems [3, 18, 62].

The equivalence of relational database queries in the presence of FD's and IND's has been examined in [43, 48], essentially by extending classical techniques (namely the chase). The authors of [43] show that under reasonable restrictions on the IND's, query equivalence can be reduced to well-understood cases involving only FD's. The approach of [48] is to introduce the *weak instance assumption* [38, 64]; under this restriction, query equivalence in the presence of FD's and typed IND's can be handled by the methods of [43].

Many questions remain unanswered in the area and new techniques seem to be required to handle major new cases. The techniques we have developed for FD and IND implication may be useful in this respect. In particular, it would be interesting to see if the tools we provide for typed IND's can be used to study equivalence of (typed) conjunctive queries [18, 43] in the presence of typed IND's and FD's, without the weak instance assumption of [48].

Expressing Data Distribution

An important consideration in the context of distributed databases is to find ways to preprocess relations stored at different sites, so that a given query can be processed with a minimum amount of data communication between sites. Some work has already been done on characterizing database schemes and queries for which such preprocessing is possible [8, 13]. An interesting research direction is to extend these results to allow for the presence of FD's (conceivably we will be able to preprocess more queries if the database is constrained to satisfy a set of FD's). Since data distribution can be modeled by IND's, these questions can be approached as implication problems involving FD's and IND's.

Performance of Equational Theorem Provers

An interesting practical question is how well theorem provers designed around the Knuth-Bendix method [46] perform on sets of equations obtained from database constraints. We have experimented with the REVE system [35, 49], which has been able to handle various non-trivial inferences of FD's and IND's. However, more work needs to be done in this direction.

References

1. Aho, A.V., Beeri, C. and Ullman, J.D. The Theory of Joins in Relational Databases. *ACM Transactions on Database Systems 4*, 3 (1979), 297-314.

2. Aho, A.V., Hopcroft, J.E. and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

3. Aho, A.V., Sagiv, Y. and Ullman, J.D. Equivalences of Relational Expressions. *SIAM Journal on Computing 8*, 2 (1979), 218-246.

4. Aho, A.V. and Ullman, J.D. Universality of Data Retrieval Languages. Proceedings of the 6th ACM Symposium on Principles of Programming Languages, ACM, 1979, pp. 110-120.

5. Armstrong, W.W. Dependency Structure of Database Relationships. Proceedings IFIP 74, Amsterdam, 1974, pp. 580-583.

6. Beeri, C. and Bernstein, P.A. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Database Systems 4*, 1 (March 1979), 30-59.

7. Beeri, C., Bernstein, P.A. and Goodman, N. A Sophisticate's Introduction to Database Normalization Theory. Proceedings of the 4th VLDB Conference, 1978, pp. 113-124.

8. Beeri, C., Fagin, R., Maier, D. and Yannakakis, M. On the Desirability of Acyclic Database Schemes. *Journal of the ACM 30*, 3 (July 1983), 479-513.

9. Beeri, C. and Korth, H.F. Compatible Attributes in a Universal Relation. Proceedings of the 1st ACM Symposium on Principles of Database Systems, ACM, 1982, pp. 55-62.

10. Beeri, C. and Vardi, M.Y. Formal Systems for Tuple and Equality Generating Dependencies. *SIAM Journal on Computing 13*, 1 (February 1984), 76-98.

11. Beeri, C. and Vardi, M.Y. A Proof Procedure for Data Dependencies. *Journal of the ACM 31*, 4 (October 1984), 718-741.

12. Bernstein, P.A. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Transactions on Database Systems 1* (1976), 277-298.

13. Bernstein, P.A. and Goodman, N. Power of Natural Semijoins. *SIAM Journal on Computing 10*, 4 (November 1981), 751-771.

14. Birkhoff, G. On the Structure of Abstract Algebras. *Proceedings of the Cambridge Philosophical Society 31* (1935), 433-454.

15. Breazu, V. Semantics in Complete Lattices for Relational Database Functional Dependencies. *Analele Stiintifice ale Universitatii "Al.I. Cuza" din Iasi 28* (1982), 177-182.

16. Casanova, M.A., Fagin, R. and Papadimitriou, C.H. Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Sciences 28*, 1 (February 1984), 29-59.

17. Casanova, V. and Vidal, V.M.P. Towards a Sound View Integration Methodology. Proceedings of the 2nd ACM Symposium on Principles of Database Systems, ACM, 1983, pp. 36-47.

18. Chandra, A.K. and Merlin, P.M. Optimal Implementation of Conjunctive Queries in Relational Databases. Proceedings of the 9th ACM Symposium on Theory of Computing, ACM, 1977, pp. 77-90.

19. Chandra, A.K. and Vardi, M.Y. The Implication Problem for Functional and Inclusion Dependencies is Undecidable. *SIAM Journal on Computing 14*, 3 (August 1985), 671-677.

20. Chen, P.P.S. The Entity-Relationship Model: Towards a Unified View of Data. *ACM Transactions on Database Systems 1*, 1 (March 1976), 9-36.

21. Codd, E.F. A Relational Model for Large Shared Data Banks. *Communications of the ACM 13*, 6 (June 1970), 377-387.

22. Codd, E.F. Further Normalization of the Database Relational Model. In *Database Systems*, Rustin, R., Ed., Prentice Hall, 1972, pp. 33-64.

23. Codd, E.F. Relational Database: A Practical Foundation for Productivity. *Communications of the ACM 25*, 2 (February 1982), 109-117.

24. Codd, E.F. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems 4*, 4 (December 1979), 397-434.

25. Cosmadakis, S.S. and Kanellakis, P.C. Equational Theories and Database Constraints. Proceedings of the 17th ACM Symposium on Theory of Computing, ACM, May, 1984, pp. 273-284.

26. Cosmadakis, S.S. and Kanellakis, P.C. Functional and Inclusion Dependencies: A Graph Theoretic Approach. Proceedings of the 3rd ACM Symposium on Principles of Database Systems, ACM, April, 1984, pp. 24-37.

27. Cosmadakis, S.S., Kanellakis, P.C. and Spyratos, N. Partition Semantics for Relations. Proceedings of the 4th ACM Symposium on Principles of Database Systems, ACM, March, 1985, pp. 261-275.

28. Crawley, P. and Dilworth, R.P. *Algebraic Theory of Lattices*. Prentice-Hall, 1973.

29. Date, C. Referential Integrity. Proceedings of the 7th VLDB Conference, 1981, pp. 2-12.

30. Dean, R.A. Component Subsets of the Free Lattice on *n* Generators. *Proceedings of the American Mathematical Society 7* (1956), 220-226.

31. Downey, P.J., Sethi, R. and Tarjan, R.E. Variations on the Common Subexpression Problem. *Journal of the ACM 27*, 4 (October 1980), 758-771.

32. Enderton, H.B. *A Mathematical Introduction to Logic.* Academic Press, 1972.

33. Fagin, R. A Normal Form for Relational Databases that is Based on Domains and Keys. *ACM Transactions on Database Systems 6*, 3 (September 1981), 387-415.

34. Fagin, R. Horn Clauses and Database Dependencies. *Journal of the ACM 29*, 4 (October 1982), 952-985.

35. Forgaard, R. and Guttag, J.V. REVE: A Term Rewriting System Generator with Failure Resistant Knuth-Bendix. Proceedings of an NSF Workshop on the Rewrite Rule Laboratory, NSF, April, 1984, pp. 5-31.

36. Grätzer, G. *Universal Algebra.* Springer-Verlag, New York, 1979.

37. Hammer, M. and McLeod, D. Database Description with SDM: A Semantic Database Model. *ACM Transactions on Database Systems 6*, 3 (September 1981), 351-386.

38. Honeyman, P. Testing Satisfaction of Functional Dependencies. *Journal of the ACM 29*, 3 (July 1982), 668-677.

39. Honeyman, P., Ladner, R.E., Yannakakis, M. Testing the Universal Instance Assumption. *Information Processing Letters 10*, 1 (1980), 14-19.

40. Hopcroft, J.E. and Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

41. Huet, G. and Oppen, D. Equations and Rewrite Rules: a Survey. In *Formal Languages: Perspectives and Open Problems*, Book, R., Ed., Academic Press, 1980, pp. 349-403.

42. Hull, R. and Yap, C.K. The Format Model: A Theory of Database Organization. Proceedings of the 1st ACM Symposium on Principles of Database Systems, ACM, 1982, pp. 205-211.

43. Johnson, D.S. and Klug, A. Testing Containment of Conjunctive Queries Under Functional and Inclusion Dependencies. *Journal of Computer and System Sciences 28*, 1 (February 1984), 167-189.

44. Kanellakis, P.C., Cosmadakis, S.S. and Vardi, M.Y. Unary Inclusion Dependencies Have Polynomial Time Inference Problems. Proceedings of the 15th ACM Symposium on Theory of Computing, ACM, 1983, pp. 264-277.

45. Klug, A. Entity-Relationship Views over Uninterpreted Enterprise Schemas. In *International Conference on Entity-Relationship Approach*, Chen, P.P.S., Ed., North Holland, 1980, pp. 39-59.

46. Knuth, D.E. and Bendix, P.B. Simple Word Problems in Universal Algebras. In *Computational Problems in Abstract Algebra*, Leech, J., Ed., Pergamon, Oxford, 1970, pp. 263-297.

47. Kozen, D. Complexity of Finitely Presented Algebras. Proceedings of the 9th ACM Symposium on Theory of Computing, ACM, May, 1977, pp. 164-177.

48. Laver, K., Mendelzon, A.O. and Graham, M.H. Functional Dependencies on Cyclic Database Schemes. Proceedings of the ACM SIGMOD Conference, ACM, 1983, pp. 79-91.

49. Lescanne, P. Computer Experiments with the REVE Term Rewriting System Generator. Proceedings of the 10th ACM Symposium on Principles of Programming Languages, ACM, January, 1983, pp. 99-108.

50. Lewis, H.R. and Papadimitriou, C.H. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

51. Maier, D. *The Theory of Relational Databases.* Computer Science Press, 1983.

52. Maier, D., Mendelzon, A.O. and Sagiv, Y. Testing Implications of Data Dependencies. *ACM Transactions on Database Systems 4*, 4 (December 1979), 455-469.

53. Minsky, M.L. Recursive Unsolvability of Post's Problem of "Tag" and Other Topics in the Theory of Turing Machines. *Annals of Mathematics 74*, 3 (1961), 437-455.

54. Mitchell, J.C. The Implication Problem for Functional and Inclusion Dependencies. *Information and Control 56*, 3 (March 1983), 154-173.

55. E.L. Post. Recursive Unsolvability of a Problem of Thue. *Journal of Symbolic Logic 13* (1947), 1-11.

56. Pudlak, P. and Tuma, J. Every Finite Lattice Can Be Embedded in a Finite Partition Lattice. *Algebra Universalis 10*, 1 (1980), 74-95.

57. Sadri, F. and Ullman, J.D.,. Template Dependencies: A Large Class of Dependencies in Relational Databases and its Complete Axiomatization. *Journal of the ACM 29*, 2 (April 1982), 363-372.

58. Sciore, E. Inclusion Dependencies and the Universal Instance. Proceedings of the 2nd ACM Symposium on Principles of Database Systems, ACM, 1983, pp. 48-57.

59. Smith, J.M. and Smith, D.C.P. Database Abstractions: Aggregation. *Communications of the ACM 20*, 6 (1977), 405-413.

60. Spyratos, N. The Partition Model: A Deductive Database Model. Tech. Rep. No. 286, INRIA, April, 1984.

61. Tsichritzis, D.C. and Lochovsky, F.H. *Data Models.* Prentice Hall, 1982.

62. Ullman, J.D. *Principles of Database Systems.* Computer Science Press, 1983.

63. Vardi, M.Y. The Implication Problem for FD's is Polynomial-Time Complete. Personal Communication.

64. Vassiliou, Y. *A Formal Treatment of Imperfect Information in Database Management.* Ph.D. Th., University of Toronto, 1980.

65. Whitman, P.M. Free Lattices. *Annals of Mathematics 42* (1941), 325-330.

66. Whitman, P.M. Lattices, Equivalence Relations, and Subgroups. *Bulletin of the American Mathematical Society 52* (1946), 507-522.

67. Wiederhold, G. and El-Masri, R. A Structural Model for Database Systems. Tech. Rep. STAN-CS-79-722, Stanford University, February, 1979.

68. Yannakakis, M. and Papadimitriou, C.H. Algebraic Dependencies. *Journal of Computer and System Sciences 21*, 1 (August 1982), 2-41.

69. Zaniolo, C. Design of Relational Views over Network Schemas. Proceedings of the ACM SIGMOD Conference, ACM, 1979.

Table of Contents

Table of Figures