Equational Theories and Database Constraints

Starros S. Cosmadakis Paris C. Kanellakis
Laboratory for Computer Science, MIT

Abstract
We present a novel way to formulate database dependencies as sentences of first-order logic, using equational statements instead of Horn clauses. Dependency implication is directly reduced to equational implication. Our approach is powerful enough to express functional and inclusion dependencies, which are the most common database constraints. We present a new proof procedure for these dependencies. We use our equational formulation to derive new upper and lower bounds for the complexity of their implication problems.

1. Introduction
In order to deal formally with the problems of logical database design and data processing, database theory models data as sets of tables (relations). These relations are required to satisfy integrity constraints (dependencies), which intend to capture the semantics of a particular application. Various kinds of dependencies have been proposed in the literature (see [25, 11] for reviews of the area). For example, a functional dependency (FD) is a formal statement of the form EMPLOYEE $\rightarrow$ SALARY, which intuitively states that every employee has a unique salary. An inclusion dependency (IND) is a statement of the form MANAGER $\subseteq$ EMPLOYEE, which intuitively states that every manager is an employee (the more general IND MANAGER.SALARY $\subseteq$ EMPLOYEE.EMPLOYEE.SALARY expresses also the fact that managers make the same salary as managers as they make as employees). FD's and IND's are the most common database constraints. A most general formulation of dependencies as sentences in first order logic (namely Horn clauses) was given in [11]. To handle the central computational problem of dependency implication a particular proof procedure was developed, the chase (see [25] for its wide applicability). Proof procedures for general data dependencies also appear in [26, 2, 3]. The chase was seen to be a special case of a classical theorem proving technique, namely resolution [2, 3].

Alternative methods for theorem proving have been developed in the context of equational theories. This is a fragment of first order logic which has attracted a lot of attention because of its wide applicability in areas such as applicative languages, interpreters, and data types. See [14] for a survey of the area.

Given the formulation of database constraints as first order sentences, one would expect database theory to have been influenced by the developments in equational theories. However, not only did this never happen, but a constant effort has been made to minimize the role of equality in data dependencies (multivalued dependencies, the most widely studied after FD's, do not involve equality). This is even more impressive in view of the fact that the best algorithm for losslessness of joins, a basic computational problem, was derived from an efficient algorithm for congruence closure [10], and the best algorithm for implication of FD's [1] can be seen directly as a special case of an algorithm of [18] for the generator problem in finitely presented algebras.

This paper is a first attempt to rectify this situation. We demonstrate that there is a close connection between dependencies and equational statements. This strongly suggests the possibility of using the tools of equational theories to handle implication of dependencies. We explain our transformation of IND and FD implication into equational implication in Section 3 (Theorem 1). This transformation vastly simplifies arguments about provability of dependencies (compared to arguments using the
chase), and enables us to prove a number of results on implication problems for FD's and IND's.

We illustrate our basic approach with an example: An FD \( A \rightarrow B \) is transformed into a string equation \( a_1 = b_1 \), and an IND CDC\(_{CAB} \) is transformed into the equations \( ai = c, bi = d \). Now we can easily infer the equation \( \chi = d \). This corresponds to inferring the FD C \( \rightarrow \) D. In general, proofs in equational theories have a clean combinatorial structure, due to the existence of a simple, intuitive proof system [4].

A number of results are known about FD and IND implication. For IND's alone and FD's alone we have finite controllability, i.e., implication and finite implication coincide. While FD implication is decidable in linear time [1], IND implication is PSPACE-complete [5]. Syntactic restrictions on the IND's simplify the implication problem: bounded width IND's [5] and typed IND's [6] have polynomial time implication problems. The problem becomes NP-complete for acyclic IND's [24, 9]. The combination of FD's and IND's is not finitely controllable [5], or even decidable [22, 7]. The combination of FD's and unary IND's is also not finitely controllable, but both implication and finite implication can be decided in polynomial time [16]. A fundamental difficulty with IND's is that the chase does not necessarily terminate, and even in special cases delicate analysis is required [15]. A proof procedure for FD and IND implication, which differs from the chase, is presented in [22]. Using a variant of this procedure, inference of unary FD's from typed IND's and acyclic unary FD's is shown decidable in [9]. The chase is guaranteed to terminate if the IND's are acyclic. Thus, acyclic IND's and FJ's are finitely controllable and in exponential time; NP-hardness (even if the IND's are typed) is shown in [9]. Finally, if all possible typed IND's are present, we have a variant of the universal instance assumption [25] known as pairwise consistency [19].

Our results apply to generalizations of FD's called coupled FD's (CFD's). These statements can express the additional fact that two FD's represent the same function in the database. Using our central Theorem 1, we can show:

1. Coupled unary FD's and binary IND's are dual statements. This is a direct consequence of our transformation (Corollary 1.4, Section 3).
2. Completeness of a new proof procedure for CFD's and IND's. This procedure differs from the chase and the formal system in [22], and treats CFD's and IND's in a symmetric fashion (Theorem 3, Section 3).
3. FD and IND implication is undecidable, even with only two FD's (Theorem 6, Section 4).
4. An exponential lower bound for acyclic IND and FD implication. This considerably improves the NP-hardness lower bounds in [9] (Theorem 5, Section 4).
5. Completeness of a proof procedure for CFD implication from a set of CFD's and typed IND's. This generalizes the result in [9] and shows that the problem is decidable for acyclic CFD's (Theorem 5, Section 5).
6. Implication of unary FD's in the presence of pairwise consistency is undecidable. The proof uses a variant of the semidecision procedure from Theorem 5 and a rather involved reduction from the word problem for semigroups (Theorem 6, Section 5).

For finite implication we cannot use the full power of our equational technique. However, we can show:

7. The implication problem for acyclic FD's in the presence of pairwise consistency is finitely controllable (and thus our transformation is also meaningful in the finite case). This does not follow from Theorem 5; an entirely different proof technique has to be developed (Theorem 7, Section 8).
8. A weaker version of our transformation can handle finite implication of FD's and unary IND's. The proof uses the formal system of [16] (Theorem 8, Section 6).

2. Definitions

2.1. Equational Theories

Let \( M \) be a set of symbols and ARITY a function from \( M \) to the nonnegative integers \( \mathbb{K} \). The set of finite strings over \( M \) is \( M^* \). Partition \( M \) into two sets:

\[
G = \{ g \in M \mid \text{arity}(g) = 0 \} \quad \text{the generators,}
\]
\[
O = \{ \theta \in M \mid \text{arity}(\theta) > 0 \} \quad \text{the operators.}
\]

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

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

A substring of \( \tau \) is a substring of \( \tau \), which is also a term. Let \( V = \{ x_1, x_2, \ldots \} \) be a set of variables. Then the set of terms over operators \( O \) and generators \( G U V \) will be denoted by \( \mathcal{T}(M) \). For terms \( \tau_1, \ldots, \tau_k \) in \( \mathcal{T}(M) \) we can define the substitution \( \varphi = \{ (x_i, \tau_i) \mid 1 \leq i \leq k \} \) to be a function from \( \mathcal{T}(M) \) to \( \mathcal{T}(M) \). We use \( \varphi(\tau) \) or \( \tau_{x_1, \ldots, x_k} \) for the result of replacing all occurrences of variables \( x_i \) in term \( \tau \) by term \( \tau_i (1 \leq i \leq k) \), where these changes are made simultaneously.

Definition: A binary relation \( \simeq \) on \( \mathcal{S}(M) \) or \( \mathcal{T}(M) \) is a congruence provided that:

1) \( \simeq \) is an equivalence relation,
2) if \( \text{arity}(\theta) = m \) and \( \tau_i \simeq \tau_i' (1 \leq i \leq m) \) then \( \theta \tau_1 \cdots \tau_m \simeq \theta \tau_1' \cdots \tau_m' \).

An equation \( \epsilon \) is a string of the form \( \tau = \tau' \), where \( \tau, \tau' \) 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 \( \epsilon \) as a sentence of first-order predicate calculus (with equality), where all the variables from \( V \) are universally quantified.

274
Definition: An algebra $A=(A,F)$ is a pair, where $A$ is a nonempty set and $F$ a set of functions. Each $f$ in $F$ is a function from $A^n$ to $A$, for some $n$ in $\mathbb{N}$ which we call the type of $f$.

Examples: (a) A semigroup $(A,\cdot)$ is an algebra with one associative binary operator, i.e., for all $x,y,z$ in $A$ $(x+y)+z=x+(y+z)$. An example of a semigroup is the algebra of the set of functions from $A$ to $A$, together with the composition operation. In semigroups we use $ab$ instead of $a+b$ and w.l.o.g. omit parentheses.

(b) $A_M$ is an algebra with $A=\mathcal{P}(M)$. For each $\theta$ in $O$ we define a function $\theta$ in $P$ with $\theta(\psi)=\text{arity}(\theta)$; here we use the same symbol for the syntactic object $\theta$ and its interpretation. The function $\theta$ maps terms $\tau_1,...,\tau_m$ from $\mathcal{P}(M)$ to the term $\theta(\tau_1)...\theta(\tau_m)$, i.e., $\theta(\tau_1)...\theta(\tau_m)D=\theta(\tau_1)...\theta(\tau_m)D$. We will refer to $A_M$ as the free algebra on $M$. From this example it can be shown that we can without ambiguity use both $\theta(\tau_1)...\theta(\tau_m)$ and $\theta(\tau_1)...\theta(\tau_m)$ to denote the same term.

(c) Let $\equiv$ be a congruence on $\mathcal{P}(M)$. Condition 2) of the congruence definition guarantees that the operations in $O$ are well-defined on $\equiv$-equivalence (or congruence) classes. Thus we can form a quotient algebra $\mathcal{P}(M)/\equiv$ with domain $\{[\tau]|\tau \in \mathcal{P}(M)\}$. $[\tau]$ is the $\equiv$-congruence class of $\tau$ and with functions corresponding to $O$'s operators.

(d) Similar observations with (b) and (c) can be made for the set of terms $\mathcal{T}(M)$.

Implication: Let $e$ be an equation and $A$ an algebra. $A$ satisfies $e$, or is a model for $e$, if $e$ becomes true when its operations and nonvariable generators are interpreted as the functions of $A$ and its variables take any values in $A$'s domain. 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 $e$ (written $E\models e$) if equation $e$ is true in every model of $E$.

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

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 $\tau_1 = \tau_2$ deduce $\theta(\tau_1) = \theta(\tau_2)$ ($\theta$ is any substitution).
- From $\tau_1 = \tau_2$ deduce $\psi(\tau_1) = \psi(\tau_2)$ ($\psi$ is any substitution).

Proposition 1: [4] $E\models \tau \iff E\vdash \tau$.

Let $\Gamma$ be a set of equations over terms in $\mathcal{P}(M)$ (i.e., containing no variables). Consider the equational theory consisting of all $\tau = \tau'$ such that, $\Gamma\vdash \tau = \tau'$. By Proposition 1 this theory induces a congruence $\equiv$, on $\mathcal{P}(M)$, where $\tau \equiv \tau'$ if $\Gamma\vdash \tau = \tau'$. From example (c) above we see that this congruence naturally defines an algebra $\mathcal{P}(M)/\equiv$. If $\Gamma$ is a finite set $\mathcal{P}(M)/\equiv$ is known as a finitely presented algebra [18].

2.2. Relational Database Theory

Let $U$ be a finite set of attributes and $\mathcal{G}$ a countably infinite set of values, such that $U \cap \mathcal{G} = \emptyset$. A relation scheme is an object $R[U]$, where $R$ is the name of the relation scheme and $U \subseteq \mathcal{G}$. A tuple $t$ over $U$ is a function from $U$ to $\mathcal{G}$. Let $A_1$ be an attribute in $U$ and $a_1$ a value, where $1 \leq i \leq |U|$; if $\{A_1\} \vdash a_1$ then we represent tuple $t$ over $U$ as $a_1 ... a_n$. We represent the restriction of tuple $t$ on attributes $A_1 ... A_n$ of $U$ as $\{A_1 ... A_n\}_t$. A relation $t$ over $U$ (named $R$) is a (possibly infinite) nonempty set of tuples over $U$. A database scheme $\mathcal{D}$ is a finite set of relation schemes $\{R_1[U], ..., R_n[U] \}$ and a database $d = \{t_1,...,t_m\}$ associates each relation scheme $R_i[U]$ in $\mathcal{D}$ with a relation $t_i$ over $U_i$. 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. We will be examining two very common types of dependencies.

CFD (R: $A_1 ... A_n \rightarrow A, S:B_1 ... B_n \rightarrow B$) is a coupled functional dependency. Relations $R.S$ (named $R,S$ respectively), satisfy this CFD if:

- for tuples $t_1, t_2$ in $t$, $t_1[A_1 ... A_n] = t_2[A_1 ... A_n]$ implies $t_1[A] = t_2[A]$ and
- for tuples $t_1, t_2$ in $s$, $t_1[B_1 ... B_n] = t_2[B_1 ... B_n]$ implies $t_1[B] = t_2[B]$ and
- for tuples $t_1$ in $t$, $t_2$ in $s$, $t_1[A_1 ... A_n] = t_2[B_1 ... B_n]$ implies $t_1[A] = t_2[A]$.

If $R = S$, $A = B$, $A_1 = B_1$, ..., $A_n = B_n$ we have a functional dependency (FD). If $n = 1$, i.e., single attribute left hand sides, then we have a binary functional dependency (b-FD). If for an FD we also have $n = 1$ then we call the dependency a unary functional dependency (u-FD). Note that every u-FD is both a b-FD and an FD. For an FD we usually employ the less redundant notation $R:A_1 ... A_n \rightarrow A$.

IND $S:D_1 ... D_m \subseteq C_1 ... C_m$ is an inclusion dependency. Relations $S,R$ (named $S,R$ respectively) satisfy this IND if, for each tuple $t$ in $S$, there is a tuple $t$ in $R$ with $t[C_i] = t[D_i]$ for $1 \leq i \leq m$. If $m = 2$ we have a binary inclusion dependency (b-ID) and if $m = 1$ a unary inclusion dependency (u-ID). Note that u-ID's are in fact special cases of b-ID's, since $S:D \subseteq C$ has the same meaning with $S:D_1 ... D_m \subseteq C_1 ... C_m$.

Equality of two columns headed by attributes $A, B$ in a relation named $R$ can be expressed as a special case of IND's or CFD's: either use a CFD, such as $(R:A \rightarrow A, R:A \rightarrow B)$, or use an IND, such as $R:ABC \subseteq AA$. These dependencies are particularly illustrative of our analysis; we will use $A = B$ to denote them.
Database Notation: We use a graph notation to represent an input database scheme $D$ and set of dependencies $\Sigma$ (input schema). We construct a labeled directed graph $G_\Sigma$ (see Figure 1), which has exactly one node $a_i$ for each attribute $A_i$ of each relation scheme $R_i$. Let $i = R_1:D_1\ldots D_m\subseteq C_1\ldots C_n$ be an IND in $\Sigma$. Then $G_\Sigma$ contains $m$ black arcs $(c_{1j}a_{12j}, \ldots, c_{m,1}a_{m,2j})$; each arc labeled by the name $i$ of the IND. Let $f=(R_1:A_1\ldots A_n \rightarrow B_1\ldots B_m)$ be a CFD in $\Sigma$. Then $G_\Sigma$ contains two groups of $n$ red arcs $(a_{11},b_{11},\ldots,a_{1n},b_{1n})$ and $(a_{21},b_{21},\ldots,a_{2n},b_{2n})$; each group is labeled by the name $f$ of the CFD and each group's arcs are ordered from 1 to $n$ as listed above.

We also consider the following directed graphs $I_\Sigma$ and $F_\Sigma$: $I_\Sigma$ has one node for each relation scheme name in $D$ and arc $(R,S)$ if and only if $G_\Sigma$ contains some black arc $(RA,SB)$. $F_\Sigma$ has one node for each attribute in $D$ and arc $(A,B)$ if and only if $G_\Sigma$ contains some red arc $(RA, RB)$. We now define special syntactic forms of input schemata:

- **Acyclic IND's:** $I_\Sigma$ is acyclic.
- **Acyclic CFD's:** $F_\Sigma$ is acyclic.
- **Typed IND's:** The black arcs of $G_\Sigma$ are all of the form $(RA,SA)$ for relation names $R$, $S$ and attribute $A$.
- **Typed CFD's:** The black arcs of $G_\Sigma$ are all of the form $(RA,SA)$ for relation names $R$, $S$ and attribute $A$.

Implication: We say that $\Sigma$ implies $\sigma$ ($\Sigma \models \sigma$) if, whenever a database $d$ over scheme $D$ satisfies $\Sigma$, it also satisfies $\sigma$. If we restrict ourselves to finite databases we have $\Sigma \models \sigma$. Clearly if $\Sigma \models \sigma$ (implication) then $\Sigma \models \sigma$ (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 proofs as $\Sigma \models \sigma$). A proof procedure is sound if it is sound and complete if it is sound and complete if it is sound and complete. The standard complete proof procedure for database dependencies is the chase. The appropriate chase rules for our analysis are described in [15].

3. Database Constraints as Equations

Let $\Sigma$ be a set of CFD's and IND's over a database scheme $D$ and $\sigma$ a CFD or IND. We will transform $\Sigma$ into two sets of equations $E_\Sigma$ and $\Sigma_e$. We will show that $\Sigma \models \sigma$ iff $E_\Sigma \models \Sigma_e$ for some sets of equations $E_\Sigma, \Sigma_e$ whose form depends on $\Sigma$ and $\sigma$. We assume that $D$ only contains one relation scheme; this simplifies notation, and there is no loss of generality.

**Transformation:** From the dependencies in $\Sigma$ construct the following sets of symbols:

- $M_\Sigma = \{f_i \mid \text{for each CFD with } n \text{ attributes left-hand sides include one operator } f_i \text{ of ARITY } n\}$
- $M_\Sigma = \{i_j \mid \text{for each IND include one operator } i_j \text{ of ARITY } 1\}$
- $M_\Sigma = \{a_k \mid \text{for each attribute } A_k \text{ include one operator } a_k \text{ of ARITY } 0\}$

Now let $M = M_\Sigma \cup M_\Sigma \cup M_\Sigma$ and $V = \{x_1, x_2, \ldots\}$ be a set of variables.

The set $E_\Sigma$ consists of the following equations (presented both in string and parenthesized notation):

1) two equations for each $a_k = (A_1 \ldots A_n \rightarrow A, B_1 \ldots B_m)$: $f_k a_1 x_1 \ldots a_n x_n = a(x)$ and $f_k b_1 x_1 \ldots b_m x_m = b(x)$;

2) $m$ equations for each $i_k = B_1 \ldots B_m \subseteq A_1 \ldots A_n$: $i_k a_1 x_1 = a_1(x)$ and $i_k a_{1n} x_n = a_n(x)$;

3) two equations for each $f_k = (A_1 \ldots A_n \rightarrow A, B_1 \ldots B_m)$: $f_k a_1 x_1 \ldots a_n x_n = a(x)$ and $f_k b_1 x_1 \ldots b_m x_m = b(x)$;

4) $m$ equations for each $i_k = B_1 \ldots B_m \subseteq A_1 \ldots A_n$: $i_k a_1 x_1 = a_1(x)$ and $i_k a_{1n} x_n = a_n(x)$.

The set $\Sigma_e$ consists of the following equations:

5) for each pair of symbols $f_i$ in $M_f$ and $i_i$ in $M_i$ the equation $f_i (i_i(x_1), \ldots, i_i(x_n)) = g_i (f_i(x_1), \ldots, f_i(x_n))$.

The transformation is illustrated in Figure 2. Note that in $E_\Sigma$ only equations 5) contain variables. Equations 5) are commutativity conditions between $f$ and $i$ operators. We now present Theorem 1, which is central to our analysis.

**Theorem 1:** In each of the following three cases, (i),(ii),(iii) are equivalent.

**Case:**

- $\Sigma \models \Lambda \equiv \beta$.
- $E_\Sigma \models \beta = \alpha$.
- $\Sigma_e \models \alpha = \beta$.

**CFD Case:**

1) $\Sigma \models (A_1 \ldots A_n \rightarrow A, B_1 \ldots B_m)$

- $E_\Sigma \models \tau(x_1, x_2, \ldots, x_n) = \alpha$ and $\tau(x_1, x_2, \ldots, x_n) = \beta$, for some $\tau$ in $\Sigma_e (M_f)$.

- $\Sigma_e \models \tau(x_1, x_2, \ldots, x_n) = \alpha$ and $\tau(x_1, x_2, \ldots, x_n) = \beta$, for some $\tau$ in $\Sigma_e (M_f)$.
IND Case:

i) $\Sigma \Rightarrow B_1...B_nC_A_1...A_m$

ii) $E_{\Sigma} \Rightarrow a_1 = b_1 x$ and \ldots and $a_m = b_m x$, for some $\tau$ in $\mathcal{F}'(M)$.

iii) $E_{\Sigma} \Rightarrow \tau(x/a_1) = \beta_1$ and \ldots and $\tau(x/a_m) = \beta_m$, for some $\tau$ in $\mathcal{F}'(M)$.

Proof Sketch: We use $E_r(\mathcal{E}_r)$ to denote the set of equations corresponding to term $r$ in (ii),(iii).

(iii)$\Rightarrow$(i) Suppose $E_{\Sigma}$, and let relation $r$ satisfy $\Sigma$; we will show that $r$ satisfies $\sigma$. Relation $r$ is, by definition, nonempty and its entries can be written, for example, positive integers. Number its tuples $\mathcal{T}_1, \mathcal{T}_2, \ldots$, (it could contain a countably infinite number of tuples). Define $A(\cdot) : \mathcal{X} \rightarrow \mathcal{K}$ such that, if $x$ is the number of a tuple in $\tau$, then $A(x)$ is the entry in tuple $x$ at attribute $A$, else $A(x)$ is 0. ($\mathcal{X}$ are the nonnegative integers). If $f$ is the CFD $D_1 ... D_k \rightarrow C_1 ... C_k$ in $\Sigma$ define $F(\cdot) : \mathcal{X} \rightarrow \mathcal{K}$ such that, if $x$ is the number of a tuple in $\tau$, then $F(D_1 ... D_k)(x) = D_1(x) \wedge \ldots \wedge D_k(x)$, else $F$ is 0. This is a well defined function since $r$ satisfies $f$. If $i$ is the IND $D_1 ... D_k \rightarrow C_1 ... C_k$ in $\Sigma$ define $I(\cdot) : \mathcal{X} \rightarrow \mathcal{K}$ such that, if $x$ is the number of a tuple in $\tau$ and $x'$ is the number of the first tuple in $\tau$ where $x' \neq x$, then $I(x) = x'$, else $I(x) = 0$. This is also a well defined function since $r$ satisfies $i$. We have constructed an algebra with domain $\mathcal{X}$ and functions $A(\cdot), I(\cdot), F(\cdot), \ldots$, which, as is easy to verify, is a model for $E_{\Sigma}$. Let $\sigma$ be an IND. By interpreting each symbol in $\sigma$ as an $I(\cdot)$, we see that when $x$ is a tuple number (attribute A), the interpretation of $r(x)$ is another tuple number. Since $E_{\Sigma}$ is a model for $E_{\Sigma}$, we must have $A(x) = B(x) 1 \leq i \leq m$, which means that $r$ satisfies $\sigma$. The case of a CFD is similar.

(ii)$\Rightarrow$(i) Suppose $E_{\Sigma}$, and let $\mathcal{M}$ be a model of $E_{\Sigma}$. We will show that $\mathcal{M}$ satisfies $E_{\Sigma}$. From $\mathcal{M}$ we will construct a model $A(\cdot) : \mathcal{M} \rightarrow \mathcal{M}$, i.e., $\mathcal{A} \rightarrow \mathcal{M}$.

In $A(\cdot)$ the interpretation of $\alpha$ will be the function $a(\cdot)$, which is the interpretation of $a(\cdot)$ in $\mathcal{A}$. The interpretation of $I(\cdot)$ will be the function $I(\cdot)$, where $I(x)$ is the interpretation of $I(x)$ in $\mathcal{M}$ (this is a function from $\mathcal{M} \rightarrow \mathcal{M}$ to $\mathcal{M} \rightarrow \mathcal{M}$).

(i)$\Rightarrow$(ii) By induction on the number of steps of a chase proof of $\sigma$ from $\Sigma$.

An alternative proof procedure for IND's and FD's only is given in [22]. We can show that each of the rules in [22] can be simulated using the equational reasoning of Proposition 1 (this provides an alternate proof of the (i) $\Rightarrow$ (iii) step for the FD and IND case). Let us illustrate it with an example: From $A \rightarrow B$ and $CD \rightarrow AB$ the pullback rule of [22] derives $C \rightarrow D$.

In equational language $fa = \beta$, $ia = \gamma$, $ib = \delta$, and $fix = if$ imply $fy = fix = ifx = i\beta$.

Corollary 1.1: Let $\Sigma$ be a set of FD's and $\sigma$ an FD. The implication problem $\Sigma \Rightarrow \sigma$ is equivalent to a generator problem for a finitely presented algebra [18].

Proof: $E_{\Sigma}$ is now a finite set of equations with no variables. If $\approx$ is the congruence induced by $E_{\Sigma}$ on $(M, V)$ then $(M, V)/\approx$ is a finitely presented algebra. The equational implication in Theorem 1 is known, in this case, as a generator problem for the finitely presented algebra $(M, V)/\approx$.

Using Corollary 1.1, one can observe that the linear time algorithm of [5] for FD inference can be derived in a straightforward way from the algorithm of [18] for the generator problem.

Corollary 1.2: Let $\Sigma$ be a set of CFD's. The implication problem $\Sigma \Rightarrow \sigma$ is a uniform word problem for a finitely presented algebra [18].

Semigroup Transformation: Let $\Sigma$ be a set of IND's and b-FD's. Produce the set of symbols $M_f$ from $M$ as follows: for each $f_i(\cdot)$ in $M_f$ add one generator $f_i$ in $M_f$; for each $i(\cdot)$ in $M$ add one generator $i_1$ in $M_f$; for each $a_1(\cdot)$ in $M_f$ add one generator $a_1$ in $M_f$; and one binary operator $+$ in $M_f$.

$E_{\Sigma}$ consists of the associative axiom for + and the following word (string) equations (we omit + and parentheses):

1) two equations for each b-FD $f_i(\cdot) = \alpha_1 ... \alpha_m$:

   \quad $f_1 a_1 = a$ and $f_1 b_1 = b$

2) $m$ equations for each IND $i_1(\cdot)$:

   \quad $a_1 = b_1$ and \ldots and $a_m = b_m$.

Corollary 1.3: Let $\Sigma$ be a set of b-FD's and IND's $\Sigma \Rightarrow \alpha \Rightarrow \beta$ iff $E_{\Sigma}$ is a uniform word problem for semigroups. The other two cases are known as $E_{\Sigma}$-unification problems [14].

By the symmetry in Corollary 1.3, we have

Corollary 1.4 Duality: Let $\Sigma$ be a set of b-FD's and b-IND's. and $\sigma$ a b-FD or b-IND. Transform every b-FD (A $\rightarrow$ B, C $\rightarrow$ D) into the b-IND BDCAC, and every b-IND BDCAC into the b-FD (A $\rightarrow$ B, C $\rightarrow$ D). If this transformation changes $\Sigma$ into $\Sigma'$ and $\sigma$ into $\sigma'$, then $\Sigma \Rightarrow \sigma$ iff $\Sigma' \Rightarrow \sigma'$.
A similar duality theorem for $u$-FD's, $u$-ID's and $\equiv_m$, follows from the analysis in [16]. In [22] it is shown that implication ($\models$) is undecidable for $u$-FD's and $u$-ID's. By Corollary 1.4, it is undecidable for $b$-FD's and $b$-ID's. This was also clear from the form of the undecidability reduction used in [22]. One might imagine formal statements, such as an $m$-FD $(A_1 \rightarrow A_m \rightarrow B_m)$ as duals for INDs $B_1 \rightarrow B_m C A_1 \rightarrow A_1$. Here the duality would hold in the equational theory, but these statements for $m$-FD have no natural meaning in database theory.

We will now describe a proof procedure for $u$-FD and $u$-ID implication, using the special structure of the equational theories from Theorem 1.

The Proof Procedure G: Given a set $\Sigma$ of $u$-FD's and $u$-ID's construct their graphical representation $G_{\Sigma}$ defined in Section 2.2. Each attribute and 4 of $F$ are 3 on $G$. Rules 1 and 2 introduce new unnamed nodes. The black part of $G$ now naturally represents a set of IND's $z$, and the red part a set of CFD's $\gamma$,.

We say that $\Sigma \models G_{\sigma}$ when:

$\sigma$ is $A \equiv B$ and $A,B$ are associated with the same node;
$\sigma$ is a CFD and $\sigma$ can be proved from $Z_{\Sigma}$ for CFD only, the Chase (for CFD only) is an efficient decision procedure;
$\sigma$ is an IND $B_1 \rightarrow B_m C A_1 \rightarrow A_1$, and there are $m$ black directed paths in $Z_{\Sigma}$, all with the same sequence of labels, paths $i$ starting at $A_i$ and ending at $B_i$.

Theorem 2: $\Sigma \models G_{\sigma}$ iff $2 \exists G_{\sigma}$.

Proof Sketch: We outline the proof for $\sigma$ being $A \equiv B$.

($:\Rightarrow$): Rules 3,4 are obviously sound. Rules 1 and 2 are sound in the sense of the attribute introduction rule of [22], which we illustrate as rule 5 of Figure 3.

($\Rightarrow$): We assume that we cannot prove $\sigma$, and construct a model for $E_{\Sigma}$ in which $\sigma \neq \beta$; then by Theorem 1 $\Sigma$ does not imply $\sigma$. If $\sigma$ is not provable, then there is a (possibly infinite) graph $G$ which represents $\Sigma$, is closed under the rules, and in which the names $A$ and $B$ correspond to different nodes. We add one special node $\bot$ to $G$. The labels of $G$ are symbols corresponding to INDs ($i$ symbols) or CFDs ($f$ symbols) of $\Sigma$. The groups of red arcs labeled with an $f$ are also ordered. If a node in $G \cup \{\bot\}$ has no outgoing arcs labeled with some $i$, add one going to $\bot$. If an $n$-tuple of nodes does not have a group of $n$ arcs leaving it labeled by $f$ (of ARITY $n$) and ordered 1 to $n$, add such a group going to $\bot$. The resulting graph represents functions interpreting the operators and generators in $E_{\Sigma}$. This is because closure with respect to rules 3 and 4 and the padding of $G$ we performed, guarantees functionality. The node $A$ ($B$) is the interpretation of $\alpha$ ($\beta$). Now closure with respect to rules 1 and 2 guarantees the commutativity conditions of $E_{\Sigma}$, and the fact that $G$ represents $\Sigma$ guarantees equations 3,4 of $E_{\Sigma}$. Thus, there is a model of $E_{\Sigma}$ in which $\sigma \neq \beta$. 

4. Computations as Inferences

It has been known, since at least Post's proof of the unsolvability of the word problem for Thue systems [23, 20], that arbitrary computations can be simulated by inferences in semigroups. By our Corollary 1.3, one can therefore simulate computations by inferences of IND's and unary FD's, and thus obtain lower bounds on the complexity of the implication problem for IND's and CFD's.

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

1. $\delta(q,a) = (p, \text{POP}_1)$: This means that, if $M$ is in state $q$ and $a \in P_1$, then $M$ goes to state $p$ and pops STACK1.
2. $\delta(q) = (p, \text{PUSH}_2(B))$: If $M$ is in state $q$, then on the next step $M$ goes to state $p$ and pushes $B$ on STACK2.

Of course, analogous instructions can manipulate STACK2.

An instantaneous description (ID) of $M$ is a string $x_1 \ldots x_n y_m \ldots y_1$, where $qCQ, x_1, x_n$ are the contents of STACK1 (the top symbol is $x_1$), the string $y_m, y_1$ are the contents of STACK2 (the top symbol is $y_m$), and $w_1 \Rightarrow w_2$ (ID $w_1$ yields ID $w_2$ via one step of M) is defined in the standard way [20, 13]: $w_1^*$ is the reflexive, transitive closure of $\Rightarrow_M$.

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

1. $\text{IF } \delta(q,a) = (p, \text{POP}_1), \text{ then } qa = p \in S$.
2. $\text{IF } \delta(q,a) = (p, \text{PUSH}_2(B)), \text{ then } qa = pB \in S$.

We write $u = v$ iff $\sigma_{start} \Rightarrow_S \text{v}$. By a standard argument, based on the fact that $M$ is deterministic [23, 20], we have

Lemma 1: $q_{start} \Rightarrow_S ^* \text{v}$ iff $q_{start} = s_{start}^* \text{v}$.

To prove our first lower bound, we transform $S$ into another set of equations $T$ which looks like the sets obtained (as in Corollary 1.3) from IND's and $u$-FD's. The set of generators is now
QU{A,,B.,fn ) aEfI}U{i, I aEfI}UIj,l eES).

1. If qa = p is in S, then qa = p is in T.

2. If qa = p is in S, then T contains the equations q = A1 n e, f1 A = B1 1 e = p, where e is aq = p.

Lemma 2: qa * h iff qa * h.

Proof Sketch: Observe that if qa = p is in S then qa = p because /q = A&.., = Blq, Bie = p. 1

Theorem 3: The implication problem for IND's and two u-FD's is undecidable.

Proof Sketch: Given a deterministic two-stack machine M, it is undecidable if qa = p is in S even if [11] = 2, 21, By Lemmas 1 and 2, qa * h iff qa * h, and by Corollary 1.3, qa = p = h iff 21 = Q = h, where 21 is the set of IND'S and FDI's which give rise to T. But now observe that 21 only contains FDI's of the form A1 = B1, aEII, and thus since [11] = 2, 21 only contains two unary FDI's. 1

Undecidability of the implication problem for IND's and FDI's has already been proved [12, 7]. By way of comparison, these reductions use arbitrarily many b-FD's and 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 w2 = M w1 iff ID w1 follows from ID w2 by a computation of M during which STACKS contains at most k symbols.

Let S be the set of word equations described before: this time we transform S into a set T of equations which can be obtained (as in Corollary 1.3) from acyclic IND's and u-FD's. The set of generators now is Q0 U U0 Qk U {A, B, f | aEII} U {q, m, m = 0, ..., k}, where Qm = {qm | q E Q}, m = 0, ..., k.

1. If qa = p is in S then qa = p is in T, m = 0, ..., k.

2. If qa = p is in S then T contains the equations q = A1 n e, f1 A = B1 1 e = p, m = 0, ..., k.

It is not hard to see that T can be taken to represent a set T of acyclic IND's and u-FD's: the relation names would be R [A1 B1 | a ЕII], R Q [Qm], m = 0, ..., k. It is also easy to see the following

Lemma 3: qa = p = h iff qa = p = h. 1

Theorem 4: There are constants c, c > 0 such that, given a set S of acyclic IND's and CFD's and an IND (CFD) a, S = a can be decided in time c2 \sqrt{\text{log} n}.

Proof Sketch: Since the IND's in 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(d^n), d > 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 [13]. Given string x, we construct a deterministic two-stack machine M which first puts x on STACK and then simulates M. This simulation is done as follows: if M is in state q, its auxiliary storage contains qa = p = x (a is the symbol scanned) and its stack contains us (s is the top symbol), then the ID of M is ufp a = r1 a = p; a is not hard to see how Mk can simulate a move of M. Thus, M accepts x iff Mk halts and STACK always contains at most [x] symbols, i.e. x E L iff qa = p = h. Note also that [Mk] is O(k).

Now let S be the set of acyclic IND's and u-FD's corresponding to M. Using Lemma 3, x E L iff 0 = R Q = h. To complete the proof, observe that S can be computed from x in polynomial time, and that 0 = O(Mx | x | \text{log} x), i.e. O(Mx | |x| |x|). 1

5. An Application to Typed IND's

We show how the tools developed in Section 3 can be applied to the particular case of inferring a CFD from CFD's and typed IND's. We first give a formal system for implication, similar in spirit to the formal system of Theorem 2. The semi-decision procedure thus obtained becomes a decision procedure if the CFD's are acyclic, generalizing a result of [9] about acyclic unary FDI's. On the other hand, by analyzing derivations we can show that the general problem is undecidable, even if only unary FDI's and all possible typed IND's are given.

Let S be a given set of CFD's and typed IND's over database scheme D = {Rk [Uk], k = 1, ..., q}, U C II. We represent attribute AEU by a node a A C FD (Rk : AB + C, Rj : A'B' + C') in S is represented as shown in Figure 4a by introducing nodes fa, ba, fa, ba, (we use a different function symbol f for each given CFD). directed arcs (ai, ak), (bi, bk) labeled 1 (we use a different label for each given IND).

Let H be the mixed graph obtained from S as described above. Repeatedly apply Rules 1, 2, 3, 4, 5, 6 to H, in some arbitrary fixed order, until no more rules are applicable. As was the case with Rules 1, 2 in Theorem 2, the introduction rules need only be applied once for each left-hand side configuration.

Proof Sketch: Since the IND's in S are acyclic, the chase gives us a decision procedure, running in exponential time.
Let \( H = (N, A, G) \) be the mixed graph obtained this way (\( N \) is a set of nodes, \( A \) is a set of labeled directed arcs on \( N \), and \( G \) is a set of undirected arcs on \( N \)): notice that each node of \( H \) is labeled \( F(a, ..., b) \), where \( F \) is a term over the function symbols and \( a, ..., b \) are nodes representing attributes. \( F(a, ..., b) \) is a shorthand for \( F[x, a, ..., q] \).

Moreover, every subterm of \( F(a, ..., b) \) appears as a node of \( H \). The graph \( H \) fully captures implication of CFD's from \( Z \):

**Theorem 5**: \( Z \models \sigma \) iff there are nodes \( F(a, ..., b) \) and \( F(a', ..., b') \) of \( H \) such that \( \langle F(a, ..., b), c, \rangle \in E \) and \( \langle F(a', ..., b'), c, \rangle \in E \).

**Proof**: Omitted. See [8].

If \( \Sigma \) consists of acyclic CFD's and typed IND's it is easy to see that the graph \( H \) is finite, and in particular its size is exponential in the size of \( \Sigma \). We thus obtain an exponential time upper bound; whether it can be improved is an open question.

**Corollary 5.1**: Inferring a CFD from acyclic CFD's and typed IND's is decidable in exponential time.

We can also show that inferring a CFD from (general) CFD's and typed IND's is undecidable, even if the set of IND's is restricted to be \( PC(D) \) (pairwise consistency) and all CFD's are unary FD's. Let \( \Sigma \) be a set of u-FD's. Note that if a database \( d \) over \( D \) satisfies \( PC(D) \), then \( R_j \models X \rightarrow Y \) holds in relation \( r \) iff \( R_j \models X \rightarrow Y \) holds in \( r \), where \( R_j[U_1].R_j[U_2] \) both contain attributes \( X \) and \( Y \). For this reason we can suppress relation names from FD's.

6. **Equational Theories and Finite Implication**

We now examine to what extent the tools we developed can handle finite implication of database constraints. Ideally, we would like to be able to replace \( \models \) by \( \models_{fin} \) throughout Theorem 1. However, our proof does not work anymore: To be sure, 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); on the other hand, the argument for (i) \( \Rightarrow \) (iii) hinges on the existence of a complete formal system for implication (namely the chase), and such a formal system cannot exist for finite implication [22, 7]. Incidentally, the same syntactic nature of the proofs of Theorems 3 and 6 prevents us from proving undecidability of finite implication. The weaker proofs of [22, 7], because of their semantic nature, can easily be done for the finite case.

Thus, we have to contend ourselves with partial results about restricted cases. First, by the discussion above one can see that \( \models \) can be replaced by \( \models_{fin} \) in Theorem 1 if we have a finitely controllable class of CFD's and IND's, i.e. a class where \( \models_{fin} \) is the same as \( \models \). An easy example of such a class is provided by CFD's and acyclic IND's, because the chase in this case constructs a finite counterexample if the implication does not hold (thus, Theorem 4 also holds for the finite case). Another such class is given in the following

**Theorem 7**: The implication problem for acyclic FD's under pairwise consistency is finitely controllable.

**Proof Sketch**: We will only consider unary FD's for simplicity. Let \( \Sigma \) consist of unary FD's; we will show that if \( PC(D) \cup \Sigma \) does not imply \( X \rightarrow Y \), then there is a finite pairwise consistent database \( d \) which satisfies \( \Sigma \) but violates \( X \rightarrow Y \). We use the notation of Lemma 4. Define \( d \) as follows:

For each attribute \( A \), the domain of \( A \), \( D_A \), consists of all functions \( f \) such that, if \( \langle p, q \rangle \in E \), then \( f(p) = f(q) \).

Example: Figure 5 has an example where \( D = \{ R_1[A, Y], R_2[N, X], R_3[A, X, B] \} \), and \( \Sigma = \{ A \rightarrow Y, Y \rightarrow B, X \rightarrow B \} \).
Since \( \Sigma \) is acyclic, each \( P_\chi \) is finite, and thus \( d \) is finite. It is not hard to reason that \( d \) satisfies the FD's in \( \Sigma \) (by the definition of the set \( F \)). We also claim that \( d \) is pairwise consistent: The key fact is that, if \( XY \ldots Z \in E \), then the tuples \( (XY \ldots Z) \in d \) are exactly the tuples \( f_1f_2 \ldots f_k \) for which \( f_k(p) = f_k(q) \) whenever \( p, q \in E \) (\( A \subseteq U \)). Finally, if \( p, q \in E \) for any \( p \) in \( P_\chi \) (Lemma 4), then one can verify that \( d \) violates \( X \ldots Y \).

Observe that the construction given above provides us with an alternative proof of the "only-if" direction of Lemma 4 (the counterexample obtained is, in general, uncountable).

If \( \text{fin} \) is different from \( \text{inf} \), we might still be able to handle the finite case if there is a complete formal system for finite implication. A class with this property is FD's with unary IND's [15]: The formal system consists of standard rules for FD's and IND's [24,5] and of the following cycle rules:

\[
\begin{align*}
&\text{from } A_0 \supseteq A_1 \supseteq A_2 \ldots \supseteq A_k \supseteq A_{k+1} \supseteq A_0 \text{ and } A_L \supseteq A_R \\
&\text{derive } A_1 \supseteq A_0 \text{ and } A_2 \supseteq A_1 \ldots \supseteq A_{k+1} \supseteq A_0 \quad (k \text{ odd}).
\end{align*}
\]

However, it turns out that we cannot replace \( \text{inf} \) by \( \text{fin} \) in Theorem 1, but have to settle for something weaker. Let \( \Sigma \) be a set of FD's and u-ID's, \( \sigma \) an FD (u-ID): \( \Sigma \models_{\text{fin}} \sigma \) can be characterized as follows (\( V \) stands for an infinitary disjunction of equations).

**Theorem 8** (\( \sigma \) is an FD): In each of the following two cases, (i),(ii),(iii) are equivalent:

**FD Case:**

1. \( \Sigma \models_{\text{fin}} A_1 \ldots A_n \rightarrow \Lambda \).
2. \( \nu \models_{\text{fin}} \forall X \in T \cdot (\forall x_1 \ldots x_n \cdot (\nu[\alpha] = \beta)) \).
3. \( \nu \models_{\text{fin}} \forall X \cdot (\forall x_1 \ldots x_n \cdot (\nu[\alpha] = \beta)) \).

**u-ID Case:**

1. \( \Sigma \models_{\text{fin}} BC \rightarrow A \).
2. \( \nu \models_{\text{fin}} \forall x \in T \cdot (\nu[\alpha] = \beta) \).
3. \( \nu \models_{\text{fin}} \forall x \cdot (\nu[\alpha] = \beta) \).

**Proof Sketch:** The implications (ii)\( \Rightarrow \) (i), (ii)\( \Rightarrow \) (ii) can be proved by the same argument as in Theorem 1. We show (i)\( \Rightarrow \) (iii) by induction on the length \( m \) of a proof of \( \sigma \) from \( \Sigma \). The basis case \( (m = 0) \) is obvious. For the induction step, we only check the cycle rule corresponding to \( k = 1 \) (the argument generalizes to arbitrary \( k \)). We write \( \tau(\alpha) \) as a shorthand for \( \tau[\alpha] \).

Proof by induction on \( m \). If \( \chi \) is a finite model of \( \Sigma \) then \( \chi \) satisfies \( \rho(\alpha) = \alpha_n \), \( \tau(\alpha) = \alpha_n \), for some \( \rho \in \mathcal{E} \), \( \tau \in \mathcal{E} \). We are ready to apply the \( m+1 \) step of the derivation, which will be the cycle rule step for \( k = 1 \). Consider now the set \( K = \{ \rho^k(\alpha) \mid k \geq 0 \} \) (\( k^k \) is \( \rho \) composed with itself \( k \) times): if \( \alpha_0 \in K \), then \( \chi \) satisfies \( \rho^k(\alpha_0) = \alpha_n \), for some \( \rho \in \mathcal{E} \) and \( \tau \in \mathcal{E} \). This term \( \rho^k \) gives us the proof for the \( m+1 \) step. If such a term did not exist, then let \( k \) be the least integer such that \( \rho^k(\alpha_0) = \alpha_n \), for some \( \rho \in \mathcal{E} \), \( k > 0 \) (\( K \) is finite since \( \chi \) is finite): by commutativity, \( \tau(\rho^k(\alpha_0)) = \rho(\tau(\alpha_0)) = \rho(\tau(\alpha_0)) = \rho(\tau(\alpha_0)) = \rho(\tau(\alpha_0)) \), similarly \( \tau(\rho^k(\alpha_0)) = \rho^k(\alpha_0) \). This means \( \rho^k(\alpha_0) = \rho^k(\alpha_0) \), which contradicts the choice of \( k \). For \( k = 1 \) we get a similar contradiction.

Finally, observe that if the FD's are also unary, we have (by analogy to Corollary 1.3) the finite \( F_\chi \) unification problem.

7. **Conclusions and Open Problems**

We have demonstrated a close relationship between implication of equations and implication of database constraints. We used this relationship to derive better bounds for the implication of FD's and IND's, which are the most common database dependencies.

An interesting practical question is how well conventional theorem proving systems perform on database dependency questions [17, 12]. Some theoretical questions also remain unresolved. For the common case of FD's and typed IND's, there is a considerable gap between the exponential upper bounds (for unary IND's and FD's, and for typed IND's and acyclic FD's) and the NP-hardness lower bound of [9]. The undecidability of finite implication for FD's in the presence of pairwise consistency is open, as well as the finite controllability of acyclic FD's and (general) typed IND's.

**References**


---

**Figure 1**

\[ \Sigma = \{ (R_1: AB \rightarrow B, R_2: AB \rightarrow D), \]
\[ R_1: B \rightarrow C, \]
\[ R_4: AB \subseteq R_2: AB, \]
\[ R_2: A \subseteq R_2: B \} \]

\[ D = \{ R_1: [ABC], R_2: [ABD] \} \]
\[ \Sigma : \quad \begin{align*} &\{ (A \rightarrow B, C \rightarrow D) \} \\
&\text{cd} \leq \text{ab} \\
&\text{AB'} \rightarrow C' \end{align*} \]

**RULE 1**

**RULE 2**

**RULE 3**

**RULE 4**

**RULE 5 [22]**

\[ \xi : \quad \begin{align*} &f \alpha x = b x \\
&f c x = d x \\
&a i x = c x \\
&b i x = d x \\
&f' c x b x = c' x \\
&f \alpha = b \\
&f y = b \\
&i x = g \\
&i b = b \\
&f a' b = y' \\
&f i x = i f x \\
&f' i x_1 i x_2 = i f' x_1 x_2 \end{align*} \]

**Figure 2**

**Figure 3**

@: new node