Propagating Functional Dependencies with Conditions

Wenfei Fan[1,2,3] Shuai Ma[1] Yanli Hu[1,5] Jie Liu[4] Yinghui Wu[1]
[1]University of Edinburgh [2]Bell Laboratories [3]Harbin Institute of Technologies
[4]Chinese Academy of Sciences [5]National University of Defense Technology

{wenfei@inf,sma1@inf,yanli@sms,yinghui@sms}.ed.ac.uk lj@kg.ict.ac.cn

Abstract

The dependency propagation problem is to determine, given a view defined on data sources and a set of dependencies on the sources, whether another dependency is guaranteed to hold on the view. This paper investigates dependency propagation for recently proposed conditional functional dependencies (CFDs). The need for this study is evident in data integration, exchange and cleaning since dependencies on data sources often only hold *conditionally* on the view. We investigate dependency propagation for views defined in various fragments of relational algebra, CFDs as view dependencies, and for source dependencies given as either CFDs or traditional functional dependencies (FDs). (a) We establish lower and upper bounds, *all matching*, ranging from PTIME to undecidable. These not only provide the *first* results for CFD propagation, but also extend the classical work of FD propagation by giving new complexity bounds in the presence of finite domains. (b) We provide the first algorithm for computing a minimal cover of *all* CFDs propagated via SPC views; the algorithm has the same complexity as one of the most efficient algorithms for computing a cover of FDs propagated via a projection view, despite the increased expressive power of CFDs and SPC views. (c) We experimentally verify that the algorithm is efficient.

1. Introduction

The prevalent use of the Web has made it possible to exchange and integrate data on an unprecedented scale. A natural question in connection with data exchange and integration concerns whether dependencies that hold on data sources still hold on the target data (*i.e.,* data transformed via mapping from the sources). As dependencies (*a.k.a.* integrity constraints) specify a fundamental part of the semantics of the data, one wants to know whether or not the dependencies are propagated from the sources via the mapping, *i.e.,* whether the mapping preserves information.

This is one of the classical problems in database research, referred to as the *dependency propagation problem.* It is to determine, given a view (mapping) defined on data sources and dependencies that hold on the sources, whether or not another dependency is guaranteed to hold on the view? We refer to the dependencies defined on the sources as *source dependencies*, and those on the view as *view dependencies.*

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '08, August 24-30, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

(a) Instance D_1 of R_1, for UK customers

	AC	phn	name	street	city	zip
t_1:	20	1234567	Mike	Portland	LDN	W1B 1JL
t_2:	20	3456789	Rick	Portland	LDN	W1B 1JL

(b) Instance D_2 of R_2, for US customers

	AC	phn	name	street	city	zip
t_3:	610	3456789	Joe	Copley	Darby	19082
t_4:	610	1234567	Mary	Walnut	Darby	19082

(c) Instance D_3 of R_3, for customers in Netherlands

	AC	phn	name	street	city	zip
t_5:	20	3456789	Marx	Kruise	Amsterdam	1096
t_6:	36	1234567	Bart	Grote	Almere	1316

Figure 1: Instances of R_1, R_2, R_3 relations

This problem has been extensively studied when source and view dependencies are functional dependencies (FDs), for views defined in relational algebra (*e.g.,* [7, 9, 15, 16, 12]). It is considered an issue already settled in the 1980s.

It turns out that while many source FDs may not hold on the view as they are, they do hold on the view under *conditions.* That is, source FDs are indeed propagated to the view, not as standard FDs but as FDs with conditions. The FDs with conditions are in the form of *conditional functional dependencies* (CFDs) recently proposed [8], as shown below.

Example 1.1: Consider three data sources R_1, R_2 and R_3, containing information about customers in the UK, US and Netherlands, respectively. To simplify the presentation we assume that these data sources have a uniform schema:

R_i(AC: string, phn: string, name: string,
street: string, city: string, zip: string)

Each tuple in an R_i relation specifies a customer's information (area code AC, phone phn, name and address (street, city, zip code)), for $i \in [1, 3]$. Example instances D_1, D_2 and D_3 of R_1, R_2 and R_3 are shown in Fig. 1.

Consider the following FDs defined on the UK and Holland sources: in instances of R_1, zip code uniquely determines street (f_1), and area code uniquely determines city (f_2); moreover, area code determines city in R_3 data (f_3).

f_1: $R_1(\text{zip} \rightarrow \text{street})$, f_2: $R_1(\text{AC} \rightarrow \text{city})$, f_3: $R_3(\text{AC} \rightarrow \text{city})$.

Define a view V with query $Q_1 \cup Q_2 \cup Q_3$ to integrate the data from the three sources, where Q_1 is

select AC, phn, name, street, city, zip, '44' as CC **from** R_1

Define Q_2 and Q_3 by substituting '01' and '31' for '44', R_2 and R_3 for R_1 in Q_1, respectively. The target schema R has all the attributes in the sources and a country-code attribute CC (44, 01, 31 for the UK, US and Netherlands, respectively).

Now one wants to know whether f_1 on the R_1 source still holds on the target data (view). The answer is negative: Figure 1 tells us that the view violates f_1 due to tuples t_3, t_4 extracted from D_2; indeed, in the US, zip does not determine street. That is, f_1 is *not* propagated to the view as an FD. In contrast, the following CFD [8] holds on the view:

φ_1: $R([\mathsf{CC} = \text{'44'}, \mathsf{zip}\ ] \rightarrow [\mathsf{street}\ ])$.

That is, for UK customers in the view, zip code uniquely determines street. In other words, φ_1 is an "FD" with a condition: it is to hold only on the subset of tuples in the view that satisfies the pattern $\mathsf{CC} = \text{'44'}$, rather than on the entire view. It cannot be expressed as a standard FD.

Similarly, from f_2 and f_3 one *cannot* derive a standard FD on the view to assert that "area code uniquely determines city". Indeed, from tuples t_1 and t_5 in Fig. 1 we can see that 20 is an area code in both the UK and Holland, for London and Amsterdam, respectively. However, not all is lost: the following CFDs are propagated from f_2 and f_3 via the view:

φ_2: $R([\mathsf{CC} = \text{'44'}, \mathsf{AC}\ ] \rightarrow [\mathsf{city}\ ])$,
φ_3: $R([\mathsf{CC} = \text{'31'}, \mathsf{AC}\ ] \rightarrow [\mathsf{city}\ ])$.

That is, f_2 and f_3 hold conditionally on the view: area code determines city for tuples with $\mathsf{CC} = \text{'44'}$ (φ_2) or $\mathsf{CC} = \text{'31'}$ (φ_3). In other words, the semantics specified by the FDs on the sources is preserved by the CFDs on the view.

Furthermore, given the following CFDs on the sources:

cfd_1: $R_1([\mathsf{AC} = \text{'20'}] \rightarrow [\mathsf{city} = \text{'LDN'}])$,
cfd_2: $R_3([\mathsf{AC} = \text{'20'}] \rightarrow [\mathsf{city} = \text{'Amsterdam'}])$,

then the following CFDs are propagated to the view:

φ_4: $R([\mathsf{CC} = \text{'44'}, \mathsf{AC} = \text{'20'}] \rightarrow [\mathsf{city} = \text{'LDN'}])$,
φ_5: $R([\mathsf{CC} = \text{'31'}, \mathsf{AC} = \text{'20'}] \rightarrow [\mathsf{city} = \text{'Amsterdam'}])$,

which carry patterns of semantically related constants. □

No previous algorithms developed for FD propagation are capable of deriving these CFDs from the given source FDs via the view. This highlights the need for investigating dependency propagation, for CFDs as view dependencies.

Applications. The study of dependency propagation is not only of theoretical interest, but also important in practice.

(1) Data exchange [17]. Recall Example 1.1. Suppose that the target schema R and CFDs φ_2 and φ_3 are predefined. Then the propagation analysis assures that the view definition V is a schema mapping from (R_1, R_2, R_3) to R, *i.e.,* for any source instances D_1 and D_3 of R_1 and R_3 that satisfy the FDs f_2 and f_3, respectively, and for any source instance D_2 of R_2, the view $V(D_1, D_2, D_3)$ is an instance of the target schema R and is guaranteed to satisfy φ_2 and φ_3.

(2) Data integration [18]. Suppose that V is a mapping in an integration system, which defines a global view of the sources. Then certain view updates, *e.g.,* insertion of a tuple t with $\mathsf{CC} = \text{'44'}$, $\mathsf{AC} = \text{'20'}$ and $\mathsf{city} = \text{'EDI'}$, can be rejected without checking the data, since it violates the CFD φ_4 propagated from the sources.

(3) Data cleaning. In contrast to FDs that were developed for schema design, CFDs were proposed for data cleaning [8]. Suppose that CFDs φ_1–φ_5 are defined on the target database, for checking the consistency of the data. Then propagation analysis assures that one need not validate these CFDs against the view V. In contrast, if in addition, an FD φ_6: $R(\mathsf{CC}, \mathsf{AC}, \mathsf{phn} \rightarrow \mathsf{street}, \mathsf{city}, \mathsf{zip})$ is also defined on the target, then φ_6 has to be validated against the view since it is not propagated from the source dependencies.

Contributions. In response to the practical need, we provide the *first results* for dependency propagation when view dependencies are CFDs. We study views expressed in various fragments of relational algebra (RA), and source dependencies expressed either as traditional FDs or CFDs.

(1) *Complexity bounds.* We provide a complete picture of complexity bounds on dependency propagation, for source FDs and source CFDs, and for various RA views. Furthermore, we study the problem in two settings: (a) *the infinite-domain setting*: in the absence of finite-domain attributes in a schema, and (b) *the general setting* where finite-domain attributes may be present. We establish upper and lower bounds, *all matching*, for all these cases, ranging from polynomial time (PTIME) to undecidable. We show that in many cases CFD propagation retains the same complexity as its FD counterpart, but in some cases CFDs do make our lives harder by incurring extra complexity.

Previous work on dependency propagation assumes the infinite-domain setting. It is believed that FD propagation is in PTIME for SPCU views [1] (union of conjunctive queries, defined with selection, projection, Cartesian product and union operators). In real world, however, it is common to find attributes with a finite domain, *e.g.,* Boolean, date, etc. It is hence necessary to study the dependency propagation problem *in the presence of finite-domain attributes*, and get the complexity right in the general setting.

In light of this we study the analysis of dependency propagation in the general setting. We show that the presence of finite-domain attributes complicates the analysis, even for *source* FDs and *view* FDs. Indeed, while FD propagation is in PTIME for SPCU views in the infinite-domain setting, this is *no longer* the case in the general setting: the problem already becomes coNP-complete for SC views, source FDs and view FDs! This intractability is unfortunately what one often has to cope with in practice.

To our knowledge this work is the first effort to study the dependency propagation problem in the general setting.

(2) *Algorithms for computing a propagation cover.* In many applications one wants not only to know whether a given view dependency is propagated from source dependencies, but also to find a *cover* of *all* view dependencies propagated. From the cover all view dependencies can be deduced via implication analysis. This is needed for, *e.g.,* processing view updates and detecting inconsistencies, as shown by the data integration and data cleaning examples given above.

Although important, this problem is rather difficult. It is known [9] that even for certain FDs and views defined with a single projection operator, a minimal cover of all view FDs propagated is sometimes necessarily exponentially large, in the infinite-domain setting. A typical method to find a cover is by first computing the closure of all source FDs, and then projecting the closure onto the view schema. While this method always takes exponential time, it is the algorithm recommended by database textbooks [23, 26].

Already hard for FDs and projection views, the propagation cover problem is far more intriguing for CFDs and SPC views. One way around this is by means of heuristic, at a price: it may not always be able to find a cover.

In contrast, we provide an algorithm to compute a minimal cover of all CFDs propagated via SPC views in the absence of finite-domain attributes, by extending a practical algorithm proposed in [12] for computing a cover of FDs propagated via projection views. Despite the increased expressive power of CFDs and SPC views, this algorithm has the same complexity as the algorithm of [12]. The algorithm behaves polynomially in many practical cases. Indeed, exponentially large covers are mostly found in examples intentionally constructed. Further, from this algorithm an effective polynomial-time heuristic is immediate: it computes a minimal cover when the cover is not large, and returns a subset of a cover as soon as the computation reaches a predefined bound, when covers are inherently large.

This is the first algorithm for computing minimal propagation covers via SPC views, for FDs or CFDs.

(3) *Experimental study.* We evaluate the scalability of the propagation cover algorithm as well as minimal covers found by the algorithm. We investigate the impact of the number of source CFDs and the complexity of SPC views on the performance of the algorithm. We find that the algorithm is quite efficient; for example, it takes less than 80 seconds to compute minimal propagation covers when given sets of 2000 source CFDs and SPC views with 50 projection attributes and selection conditions defined in terms of the conjunction of 10 domain constraints. Furthermore, it scales well with the number and complexity of source CFDs and SPC views. The minimal covers found by the algorithm are typically small, often containing less CFDs than the sets of input source CFDs. We contend that the algorithm is a promising method for computing minimal propagation covers of CFDs via SPC views, and may find practical use in data integration, data exchange and data cleaning.

This work not only provides the *first* results for CFD propagation, but also *extends the classical results* of FD propagation, an issue that was considered settled 20 years ago, by investigating the propagation problem in the general and practical setting overlooked by prior work. In addition, for both FDs and CFDs, we give the first practical algorithm for computing minimal propagation covers via SPC views.

Organization. We review CFDs and various fragments of RA in Section 2. We establish complexity bounds on dependency propagation in Section 3. We provide the algorithm for computing minimal propagation covers via SPC views in Section 4. Experimental results are reported in Section 5, followed by related work in Section 6 and topics for future work in Section 7. The proofs are in the appendix.

2. Dependencies and Views

In this section, we review conditional functional dependencies (CFDs [8]) and fragments of relational algebra (RA).

2.1 Conditional Functional Dependencies

CFDs extend FDs by incorporating a pattern tuple of semantically related data values. In the sequel, for each attribute A in a schema R, we denote its associated domain as $\mathsf{dom}(A)$, which is either infinite (*e.g.,* $\mathsf{string}, \mathsf{real}$) or finite (*e.g.,* $\mathsf{Boolean}, \mathsf{date}$).

Definition 2.1: A CFD φ on a relation schema R is a pair $R(X \to Y,\ t_p)$, where (1) $X \to Y$ is a standard FD, called the FD *embedded in* φ; and (2) t_p is a tuple with attributes in X and Y, referred to as the *pattern tuple* of φ, where for each A in X (or Y), $t_p[A]$ is either a constant 'a' in $\mathsf{dom}(A)$, or an unnamed variable '_' that draws values from $\mathsf{dom}(A)$. We separate the X and Y attributes in t_p with '$\|$'.

For CFDs on views (*i.e.,* view CFDs) we also allow a special form $R(A \to B, (x \parallel x))$, where A, B are attributes of R and x is a (special) variable. □

Note that traditional FDs are a special case of CFDs, in which the pattern tuples consist of '_' only.

Example 2.1: The dependencies we have seen in Section 1 can be expressed as CFDs. Some of those are given below:

φ_1: $R([\mathsf{CC}, \mathsf{zip}] \to [\mathsf{street}],\ (44, \_ \parallel \_))$,
φ_2: $R([\mathsf{CC}, \mathsf{AC}] \to [\mathsf{city}],\ (44, \_ \parallel \_))$,
φ_4: $R([\mathsf{CC}, \mathsf{AC}] \to [\mathsf{city}],\ (44, 20 \parallel \text{LDN}))$,
f_1: $R_1(\mathsf{zip} \to \mathsf{street},\ (\_ \parallel \_))$.

The standard FD f_1 on source R_1 is expressed as a CFD. □

The semantics of CFDs is defined in terms of a relation $\asymp$ on constants and '_': $\eta_1 \asymp \eta_2$ if either $\eta_1 = \eta_2$, or one of η_1, η_2 is '_'. The operator $\asymp$ naturally extends to tuples, *e.g.,* (Portland, LDN) $\asymp$ (_, LDN) but (Portland, LDN) $\not\asymp$ (_, NYC). We say that a tuple t_1 *matches* t_2 if $t_1 \asymp t_2$.

An instance D of R *satisfies* $\varphi = R(X \to Y,\ t_p)$, denoted by $D \models \varphi$, if for *each pair* of tuples t_1, t_2 in D, if $t_1[X] = t_2[X] \asymp t_p[X]$, then $t_1[Y] = t_2[Y] \asymp t_p[Y]$.

Intuitively, φ is a constraint defined on the set $D_\varphi = \{t \mid t \in D, t[X] \asymp t_p[X]\}$ such that for any $t_1, t_2 \in D_\varphi$, if $t_1[X] = t_2[X]$, then (a) $t_1[Y] = t_2[Y]$, and (b) $t_1[Y] \asymp t_p[Y]$. Here (a) enforces the semantics of the embedded FD, and (b) assures the binding between *constants* in $t_p[Y]$ and *constants* in $t_1[Y]$. Note that φ is defined on the subset D_φ of D identified by $t_p[X]$, rather than on the entire D.

An instance D of R satisfies CFD $R(A \to B, (x \parallel x))$ if for any tuple t in D, $t[A] = t[B]$. As will be seen shortly, these CFDs are used to express selection conditions of the form $A = B$ in a view definition, treating domain constraints and CFDs in a uniform framework.

We say that an instance D of a relational schema $\mathcal{R}$ satisfies a set Σ of CFDs defined on $\mathcal{R}$, denoted by $D \models \Sigma$, if $D \models \phi$ for each ϕ in Σ.

Example 2.2: Recall the view definition V from Example 1.1 and the instances D_1, D_2, D_3 of Fig. 1. The view $V(D_1, D_2, D_3)$ satisfies φ_1, φ_2, φ_4 of Example 2.1. However, if we remove attribute CC from φ_4, then the view no longer satisfies the modified CFD. Indeed, there are two tuples t'_1 and t'_5 in $V(D_1, D_2, D_3)$ such that t'_1 and t_1 of Fig. 1 have identical AC and city values; similarly for t_5 and t'_5 of Fig. 1. Then t'_1 and t'_5 violate the modified CFD: they have the same AC attribute but differ in city. □

2.2 View Definitions

We study dependency propagation for views expressed in various fragments of RA. It is known that the problem is already undecidable for FDs and views defined in RA [1]. In light of this we shall focus on positive fragments of RA, without set difference, in particular SPC and SPCU.

Consider a relational schema $\mathcal{R} = (S_1, \ldots, S_m)$.

SPC. An SPC query (*a.k.a.* conjunctive query) Q on $\mathcal{R}$ is an

Σ	View language	Complexity bounds: Infinite domain only	Complexity bounds: General setting
Propagation from FDs to CFDs			
FDs	SP	PTIME	PTIME
	SC	PTIME	coNP-complete
	PC	PTIME	PTIME
	SPC	PTIME	coNP-complete
	SPCU	PTIME	coNP-complete
	RA	undecidable	undecidable
Propagation from CFDs to CFDs			
CFDs	S	PTIME	coNP-complete
	P	PTIME	coNP-complete
	C	PTIME	coNP-complete
	SPC	PTIME	coNP-complete
	SPCU	PTIME	coNP-complete
	RA	undecidable	undecidable

Table 1: Complexity of CFD propagation

RA expression defined in terms of the selection (σ), projection (π), Cartesian product ($\times$) and renaming (ρ) operators. It can be expressed in the normal form below [1]:

$$\pi_Y(R_c \times E_s), \text{ where } E_s = \sigma_F(E_c),\ E_c = R_1 \times \ldots \times R_n,$$

where (a) $R_c = \{(A_1 : a_1, \ldots, A_m : a_m)\}$, a constant relation, such that for each $i \in [1, m]$, A_i is in Y, A_i's are distinct, and a_i is a constant in $\mathsf{dom}(A_i)$; (b) for each $j \in [1, n]$, R_j is $\rho_j(S)$ for some relation atom in $\mathcal{R}$, and ρ_j is a renaming operator such that the attributes in R_j and R_l are disjoint if $j \neq l$, and A_i does not appear in any R_j; (c) F is a conjunction of equality atoms of the form $A = B$ and $A =$ 'a' for a constant $a \in \mathsf{dom}(A)$.

We also study fragments of SPC, denoted by listing the operators supported: S, P, C, SP, SC, and PC (the renaming operator is included in all these subclasses by default without listing it explicitly). For instance, SC is the class of queries defined with σ, $\times$ and ρ operators.

For example, Q_1 given in Example 1.1 can be expressed as a C query: $\{(\mathsf{CC}\colon 44)\} \times R_1$.

SPCU. SPCU (*a.k.a.* union of conjunctive queries) is an extension of SPC by allowing union ($\cup$). An SPCU query defined on $\mathcal{R}$ can be expressed in normal form $V_1 \cup \ldots \cup V_n$, where V_i's are union-compatible SPC queries. For example, the view V given in Example 1.1 is an SPCU query.

In the sequel we only consider SPC and SPCU queries in the normal form, unless stated otherwise.

3. Complexity on Dependency Propagation

We now give a full treatment of dependency propagation to CFDs. Proofs of the results are given in the appendix.

Formally, the *dependency propagation problem* is to determine, given a view V defined on a schema $\mathcal{R}$, a set Σ of source dependencies on $\mathcal{R}$, and CFD φ on the view, whether or not φ is *propagated from* Σ *via* V, denoted by $\Sigma \models_V \varphi$, *i.e.,* for any instance D of $\mathcal{R}$, if $D \models \Sigma$ then $V(D) \models \varphi$.

That is, φ is propagated from Σ via V if for any source D that satisfies Σ, the view $V(D)$ is guaranteed to satisfy φ.

We study the problem in a variety of settings. (a) We consider views expressed in various fragments of RA: S, P, C, SP, SC, PC, SPC, SPCU. (b) We study the propagation problem when FDs and CFDs are source dependencies, respectively. We refer to the problem as *propagation from FDs to CFDs* when the source dependencies are FDs, and as *propagation from CFDs to CFDs* when the source dependencies are CFDs. (c) We investigate the problem in the absence and in the presence of finite-domain attributes in the schema $\mathcal{R}$, *i.e.,* in the infinite-domain setting and the general setting.

View language	Complexity bounds: Infinite domain only	Complexity bounds: General setting
Propagation from FDs to FDs		
SP	PTIME [16, 1]	PTIME
SC	PTIME [16, 1]	coNP-complete
PC	PTIME [16, 1]	PTIME
SPCU	PTIME [16, 1]	coNP-complete
RA	undecidable [15]	undecidable

Table 2: Complexity of FD propagation

We first study propagation from FDs to CFDs, and then from CFDs to CFDs. Finally, we address a related interesting issue: the emptiness problem for CFDs and views.

3.1 Propagation from FDs to CFDs

In the infinite-domain setting, propagation from FDs to FDs has been well studied, *i.e.,* for source FDs and view FDs. It is known that the propagation problem is

- ∘ undecidable for views expressed in RA [15], and
- ∘ in PTIME for SPCU views [16, 1].

In this setting, CFDs do not make our lives harder.

Theorem 3.1: *In the absence of finite-domain attributes, the dependency propagation problem from FDs to CFDs is*

- ∘ *in* PTIME *for SPCU views, and*
- ∘ *undecidable for RA views.* □

Proof Sketch: (a) For the PTIME bound, we develop an algorithm for testing propagation, via tableau representations of given SPCU views and view CFDs, by extending the chase technique (see [1] for details about chase, and [16] on extensions of chase). We show that the algorithm characterizes propagation and is in PTIME. (b) The undecidability follows from its counterpart for FDs, as CFDs subsume FDs. □

From Theorem 3.1 it follows immediately that propagation from FDs to CFDs is also in PTIME for views expressed in fragments of SPCU, *e.g.,* SPC, SP, SC and PC views.

In the general setting, *i.e.,* when finite-domain attributes may be present, the propagation analysis becomes harder. Below we show that even for propagation from FDs to FDs and for simple SC views, the problem is already intractable.

Theorem 3.2: *In the general setting, the dependency propagation problem from FDs to FDs is co*NP*-complete for SC views.* □

Proof Sketch: There is an NP algorithm that, given source FDs Σ, a view FD ψ and an SC view V, decides whether $\Sigma \not\models_V \psi$ or not. The algorithm extends the chase technique to deal with variables that are associated with finite domain attributes, such that all those variables are instantiated with constants from their corresponding finite domains. There are exponential number of such instantiations that need to be checked. For each instantiation, it can be done in polynomial time by Theorem 3.1. Thus the problem is in coNP.

The lower bound is established by reduction from the 3SAT problem to the complement of the propagation problem, where 3SAT is NP-complete (cf. [10]). Given an instance ϕ of the 3SAT problem, we define source relations $\mathcal{R}$, a set Σ of FDs on $\mathcal{R}$, a view V defined by an SC expression, and a FD ψ on V. We show that ϕ is satisfiable iff $\Sigma \not\models_V \psi$. □

In contrast to the PTIME bound in the infinite-domain setting [16, 1], Theorem 3.2 shows that the presence of finite-domain attributes does complicate the analysis of propagation from FDs to FDs, and should be thoroughly studied.

Theorem 3.3: *In the general setting, the dependency propagation problem from FDs to CFDs is*
- *in* PTIME *for PC views,*
- *in* PTIME *for SP views,*
- *co*NP*-complete for SC views,*
- *co*NP*-complete for SPCU views,*
- *undecidable for RA views.* □

Proof Sketch: (a) The PTIME bounds are verified by providing PTIME algorithms for checking propagation, by extending chase to CFDs. (b) The coNP upper bound is by giving an NP algorithm for deciding $\Sigma \not\models_V \varphi$ for the given source FDs Σ, view CFD φ and SPCU view V. The lower bound follows from Theorem 3.2 for SC views, since CFDs subsume FDs. (c) The undecidability follows from Theorem 3.1, since the general setting subsumes the infinite-domain setting. □

Theorem 3.3 also tells us that in the general setting, propagation from FDs to CFDs is (a) in PTIME for S, P and C views, and (b) coNP-complete for SPC views.

In addition, since FDs are a special case of CFDs, Theorems 3.2 and 3.3 together yield a complete picture of complexity bounds on the dependency propagation problem from FDs to FDs in the general setting.

Corollary 3.4: *In the general setting, the propagation problem from FDs to FDs is in* PTIME *for SP and PC views, and is co*NP*-complete for SPC and SPCU views.* □

3.2 Propagation from CFDs to CFDs

Upgrading source dependencies from FDs to CFDs does not incur extra complexity for propagation analysis, in the infinite-domain setting. That is, the bounds of Theorem 3.1 remain intact, which are the same as for FD propagation.

Theorem 3.5: *In the absence of finite-domain attributes, the dependency propagation problem from CFDs to CFDs is*
- *in* PTIME *for SPCU views, and*
- *undecidable for RA views.* □

Proof Sketch: (a) The PTIME bounds are again verified by developing a polynomial time checking algorithm, via an extension of the chase algorithm in Theorem 3.1 to cope with CFDs instead of FDs. (b) The undecidability follows from Theorem 3.1, since FDs are a special case of CFDs. □

This tells us that propagation from CFDs to CFDs is also in PTIME for SPC, SP, SC and PC views.

When it comes to the general setting, however, the problem becomes intractable even for very simple views.

Corollary 3.6: *In general setting, the dependency propagation problem from CFDs to CFDs is*
- *co*NP*-complete for views expressed as S, P or C queries;*
- *co*NP*-complete for SPCU views; and*
- *undecidable for RA views.* □

Proof Sketch: (a) The implication problem for CFDs is a special case of the dependency propagation problem, when views are the identity mapping, which are expressible as S, P or C queries. It is known that in the presence of finite-domain attributes, CFD implication is already coNP-complete [8]. From this it follows that the propagation problem is already coNP-hard for views expressed as S, P or C queries. (b) The coNP upper bound is verified along the same lines as Theorem 3.3. (c) The undecidability follows from Theorem 3.3, since FDs are a special case of CFDs. □

From Corollary 3.6 it follows that the propagation problem is also coNP-complete for SC, PC, SP and SPC views.

We summarize in Table 1 complexity bounds on propagation from FDs to CFDs, and from CFDs to CFDs. To give a complete picture, we present the complexity bounds on propagation from FDs to FDs in Table 2, including results from [15, 16, 1].

3.3 Interaction between CFDs and Views

An interesting aspect related to dependency propagation is the interaction between source CFDs and views.

Example 3.1: Consider a CFD $\phi = R(A \rightarrow B,\ (\_ \parallel b_1))$ defined on a source $R(A, B, C)$, and an S view $V = \sigma_{B=b_2}(R)$, with $b_2 \neq b_1$. Then for any instance D of R that satisfies ϕ, $V(D)$ is always *empty*. Indeed, the CFD ϕ assures that the B attributes of all tuples t in D have the same constant: $t[B] = b_1$, no matter what value $t[A]$ has. Hence V cannot possibly find a tuple t in D that satisfies the selection condition $B = b_2$. As a result, any source CFDs are "propagated" to the view, since the view satisfies any CFDs. □

This suggests that we consider the *emptiness problem for views and CFDs*: it is to determine, given a view V defined on a schema $\mathcal{R}$ and a set Σ of CFDs on $\mathcal{R}$, whether or not $V(D)$ is always empty for all instances D of $\mathcal{R}$ where $D \models \Sigma$.

It turns out that this problem is nontrivial.

Theorem 3.7: *In the general setting, the emptiness problem is co*NP*-complete for CFDs and SPCU views.* □

Proof Sketch: We consider the non-emptiness problem, the complement of the emptiness problem. We show that the non-emptiness problem is NP-hard by reduction from the non-tautology problem, a known NP-complete problem (cf. [10]). Its upper bound is verified by providing an NP algorithm to check the non-emptiness, by extending chase. From this follows immediately Theorem 3.7. □

The lower bound is not surprising: it is already NP-hard for deciding whether there exists a nonempty database that satisfies a given set of CFDs, in the general setting [8]. This, known as the consistency problem [8], is a special case of the complement of the emptiness problem when the view is the identity mapping. Theorem 3.7 tells us that adding views does not make our lives harder: the NP upper bound for the consistency problem for CFDs remains intact.

In the absence of finite-domain attributes, the implication problem for CFDs becomes tractable [8]. It is also the case for the emptiness problem for SPCU views and CFDs: a PTIME algorithm can be developed for the emptiness test.

Theorem 3.8: *Without finite-domain attributes, the emptiness problem is in* PTIME *for CFDs and SPCU views.* □

Proof Sketch: In the absence of finite-domain attributes, one can readily turn the NP algorithm given in the proof of Theorem 3.7 into a PTIME one, since instantiation of finite-domain attributes, which would require a nondeterministic guess, is no longer needed here. □

4. Computing Covers of View Dependencies

We have studied dependency propagation in Section 3 for determining whether a *given* view CFD is propagated from source CFDs (or FDs). In this section we move on to a related yet more challenging problem, referred to as the propagation cover problem, for finding a minimal cover of *all* view CFDs propagated from source CFDs. As remarked in Section 1, this problem is important for data integration and data cleaning, among other things. Furthermore, an algorithm for finding a propagation cover also readily provides a solution to determining whether a given CFD ϕ is propagated from a given set Σ of source CFDs via an SPC view V: one can simply compute a minimal cover Γ of all CFDs propagated from Σ via V, and then check whether Γ implies ϕ, where CFD implication is already studied in [8].

No matter how important, this problem is hard. As will be seen soon, most prior algorithms for finding FD propagation covers always take exponential time.

The main result of this section is an algorithm for computing a minimal cover of all view CFDs propagated from source CFDs, via SPC views. It is an extension of a practical algorithm proposed in [12], for computing covers of FDs propagated via projection views. It has the same complexity as that of [12], and behaves polynomially in many practical cases. It also yields an algorithm for computing propagation covers when FDs are source dependencies, a special case.

To simplify the discussion we assume the absence of finite-domain attributes, the same setting as the classical work on FD propagation [1, 12, 15, 16]. In this setting, the emptiness problem for CFDs and SPC views, and the CFD propagation problem via SPC views are all in PTIME. We consider *w.l.o.g.* CFDs in the normal form: $(R : X \to A, t_p)$, where A is a single attribute. Indeed, each CFD of the general form given in Section 2 can be converted in linear time to an equivalent set of CFDs in the normal form [8].

In the rest of the section, we first state the propagation cover problem and discuss its challenges in Section 4.1. We then provide basic results in Section 4.2, on which our algorithm is based. The algorithm is presented in Section 4.3.

4.1 The Propagation Cover Problem

Implication and cover. We say that a set Σ of CFDs defined on a schema $\mathcal{R}$ *implies* another CFD φ on $\mathcal{R}$, denoted by $\Sigma \models \varphi$, if for any instance D of $\mathcal{R}$, if $D \models \Sigma$ then $D \models \varphi$.

A *cover* of a set Σ of CFDs is a subset Σ_c of Σ such that for each CFD φ in Σ, $\Sigma_c \models \varphi$. In other words, Σ_c is contained in, and is equivalent to, Σ. For a familiar example, recall the notion of the closure F^+ of a set F of FDs, which is needed for designing normal forms of relational schema (see, *e.g.,* [1]). Then F is a cover of F^+.

A *minimal cover* Σ_{mc} of Σ is a cover of Σ such that

- no proper subset of Σ_{mc} is a cover of Σ, and
- for each CFD $\phi = R(X \to A,\ t_p)$ in Σ_{mc}, there exists no proper subset $Z \subset X$ such that $(\Sigma_{mc} \cup \{\phi'\}) - \{\phi\} \models \phi$, where $\phi' = R(Z \to A,\ (t_p[Z] \parallel t_p[A]))$.

That is, there is neither redundant attributes in each CFD nor redundant CFDs in Σ_{mc}.

We only include nontrivial CFDs in Σ_{mc}. A CFD $R(X \to A,\ t_p)$ is *nontrivial* if either (a) $A \notin X$, or (b) $X = AZ$ but t_p is not of the form $(\eta_1, \overline{d_Z} \parallel \eta_2)$, where either $\eta_1 = \eta_2$, or η_1 is a constant and $\eta_2 =$ '_'.

It is known that without finite-domain attributes, implication of CFDs can be decided in quadratic time [8]. Further, there is an algorithm [8], referred to as MinCover, which computes Σ_{mc} in $O(|\Sigma|^3)$ time for any given set Σ of CFDs.

Propagation cover. For a view V defined on a schema $\mathcal{R}$ and a set Σ of source CFDs on $\mathcal{R}$, we denote by $\mathsf{CFD}_p(\Sigma, V)$ the set of *all* view CFDs propagated from Σ via V.

The *propagation cover problem* is to compute, given V and Σ, a cover Γ of $\mathsf{CFD}_p(\Sigma, V)$. We refer to Γ as a *propagation cover* of Σ via V, and as a *minimal* propagation cover if Γ is a minimal cover of $\mathsf{CFD}_p(\Sigma, V)$.

Challenges. The example below, taken from [9], shows that the problem is already hard for FDs and simple P views.

Example 4.1: Consider a schema R with attributes A_i, B_i, C_i and D, and a set Σ of FDs on R consisting of $A_i \to C_i$, $B_i \to C_i$, and $C_1, \ldots, C_n \to D$, for each $i \in [1, n]$. Consider a view that projects an R relation onto their A_i, B_i and D attributes, dropping C_i's. Then any cover Σ_c of the set of view FDs propagated *necessarily contains* all FDs of the form $\eta_1, \ldots, \eta_n \to D$, where η_i is either A_i or B_i for $i \in [1, n]$. Obviously Σ_c contains at least 2^n FDs, whereas the size of the input, namely, Σ and the view, is $O(n)$. Indeed, to derive view FDs from $C_1, \ldots, C_n \to D$, one can substitute either A_i or B_i for each C_i, leading to the exponential blowup. □

In contrast, the dependency propagation problem is in PTIME in this setting (recall from Section 3). This shows the sharp distinction between the dependency propagation problem and the propagation cover problem.

While we are not aware of any previous methods for finding propagation covers for FDs via SPC views, there has been work on computing *embedded* FDs [12, 23, 26], which is essentially to compute a propagation cover of FDs via projection views. Given a schema R, a set F of FDs on R and a set Y of attributes in R, it is to find a cover F_c of all FDs propagated from F via a projection view $\pi_Y(R)$. An algorithm for finding F_c is by first computing the closure F^+ of F, and then projecting F^+ onto Y, removing those FDs with attributes not in Y. This algorithm always takes $O(2^{|F|})$ time, for computing F^+. As observed in [12], this is the method covered in database texts [23, 26] for computing F_c. A more practical algorithm was proposed in [12], which we shall elaborate shortly.

Already hard for FDs and P views, the propagation cover problem is more intricate for CFDs and SPC views.

(a) While at most exponentially many FDs can be defined on a schema R, there are possibly infinitely many CFDs. Indeed, there are infinitely many CFDs of the form $R(A \to B, t_p)$ when $t_p[A]$ ranges over values from an infinite $\mathsf{dom}(A)$.

(b) While $AX \to A$ is a trivial FD and can be ignored, $\phi = R(AX \to A, t_p)$ may not be. Indeed, when t_p is $(\_, \overline{d_X} \parallel a)$, ϕ is meaningful: it asserts that for all tuples t such that $t[X] \asymp \overline{d_X}$, the A column has the same constant 'a'.

(c) While $X \to Y$ and $Y \to Z$ yield $X \to Z$ for FDs, the transitivity rule for CFDs has to take pattern tuples into account and is more involved than its FD counterpart [8].

(d) As we have seen in the previous section, selection and Cartesian product introduce interaction between domain constraints and CFDs, a complication of SPC views that we do not encounter when dealing with projection views.

4.2 Propagating CFDs via SPC Views

The exponential complexity of Example 4.1 is for the worst case and is only found in examples intentionally constructed. In practice, it is common to find a propagation cover of polynomial size, and thus it is an overkill to use algorithms that always take exponential time. In light of this one wants an algorithm for computing minimal propagation covers that behaves polynomially most of the time, whereas it necessarily takes exponential time only when all propagation covers are exponentially large for a given input.

We next propose such an algorithm, referred to as PropCFD_SPC, by extending the algorithm of [12] for computing a propagation cover of FDs via projection views. Given an SPC view V defined on a schema $\mathcal{R}$ and a set Σ of source CFDs on $\mathcal{R}$, PropCFD_SPC computes a minimal propagation cover Γ of Σ via V, without increasing the complexity of the algorithm of [12], although CFDs and SPC views are more involved than FDs and P views, respectively.

Before we present PropCFD_SPC, we give some basic results behind it. Let $\mathcal{R} = (S_1, \ldots, S_m)$ be the source schema. Recall from Section 2 that V defined on $\mathcal{R}$ is of the form:

$$\pi_Y(R_c \times E_s),\ E_s = \sigma_F(E_c),\ E_c = R_1 \times \ldots \times R_n$$

where R_c is a constant relation, R_j's are renamed relation atoms $\rho_j(S)$ for S in $\mathcal{R}$, Y is the set of projection attributes, and F is a conjunction of equality atoms.

Basic results. The constant relation R_c introduces no difficulties: for each $(A_i : a_i)$ in R_c, a CFD $\mathcal{R}_V(A_i \to A_i, (\_ \parallel a))$ is included in Γ, where $\mathcal{R}_V$ is the view schema derived from V and $\mathcal{R}$. Thus in the sequel we assume that $V = \pi_Y(E_s)$.

The reduction below allows us to focus on E_s instead of V. The proof is straightforward, by contradiction.

Proposition 4.1: *For any SPC view V of the form above, and any set Σ of source CFDs, $\Sigma \models_V \phi$ iff $\Sigma \models_{E_s} \phi$ when*

- *$\phi = \mathcal{R}_V(A \to B, (x \parallel x))$, denoting $A = B$;*
- *$\phi = \mathcal{R}_V(A \to A, (\_ \parallel a))$, denoting $A =$ 'a'; or*
- *$\phi = \mathcal{R}_V(X \to A, t_p)$;*

where $A \in Y$, $B \in Y$, and $X \subseteq Y$. □

We next illustrate how we handle the interaction between CFDs and operators $\times, \sigma$ and π in the view definition V.

Cartesian product. Observe that each R_j in E_c is $\rho_j(S)$, where S is in $\mathcal{R}$. All source CFDs on S are propagated to the view, after their attributes are renamed via ρ_j.

Selection. The condition F in σ_F brings domain constraints into play, which can be expressed as CFDs.

Lemma 4.2: *(a) If $A =$ 'a' is in the selection condition F, then $\mathcal{R}_V(A \to A,\ (\_ \parallel a))$ is in $\mathsf{CFD}_p(\Sigma, V)$. (b) If $A = B$ is in F, then $\mathcal{R}_V(A \to B,\ (x \parallel x))$ is in $\mathsf{CFD}_p(\Sigma, V)$ for the special variable x.* □

That is, one can incorporate domain constraints $A =$ 'a' and $A = B$ enforced by the view V into CFDs. Here (a) asserts that the A column of the view must contain the same constant 'a', and (b) asserts that for each tuple t in the view, $t[A]$ and $t[B]$ must be identical, as required by the selection condition F in the view V (this is why we introduced CFDs of the form $\mathcal{R}_V(A \to B,\ (x \parallel x))$ in Section 2).

Lemma 4.3: *If $\mathcal{R}_V(A \to B,\ (x \parallel x))$ and $\mathcal{R}_V(BX \to G,\ t_p)$, then $\mathcal{R}_V(AX \to G,\ t'_p)$ is in $\mathsf{CFD}_p(\Sigma, V)$, where $t'_p[A] = t_p[B]$, $t'_p[X] = t_p[X]$ and $t'_p[G] = t_p[G]$.* □

That is, we can derive view CFDs by applying the domain constraint $A = B$: substituting A for B in a view CFD yields another view CFD. This also demonstrates how domain constraints interact with CFD propagation.

Let us use Σ_d to denote these CFDs as well as those in Σ expressing domain constraints. Based on Σ_d we can decide whether $A = B$ or $A =$ 'a' for attributes in Y (*i.e.,* R_V).

More specifically, we partition the attributes into a set EQ of equivalence classes, such that for any $\mathsf{eq} \in \mathsf{EQ}$, and for any attributes A, B in Y, (a) $A, B \in \mathsf{eq}$ iff $A = B$ can be derived from Σ_d; (b) if $A =$ 'a' can be derived from Σ_d and moreover, $A \in \mathsf{eq}$, then for any $B \in \mathsf{eq}$, $B =$ 'a'; we refer to the constant 'a' as the *key* of eq, denoted by $\mathsf{key(eq)}$. If a constant is not available, we let $\mathsf{key(eq)}$ be '_'.

The use of EQ helps us decide whether or not V and Σ always yield empty views (Section 3), which happens if there exists some $\mathsf{eq} \in \mathsf{EQ}$ such that $\mathsf{key(eq)}$ is not well-defined, *i.e.,* when two distinct constants are associated with eq.

It is easy to develop a procedure to compute EQ, referred to as ComputeEQ, which takes Σ and V as input, and returns EQ as output, along with $\mathsf{key(eq)}$ for each $\mathsf{eq} \in \mathsf{EQ}$. If $\mathsf{key(eq)}$ is not well-defined for some eq, it returns a special symbol '$\perp$', indicating the inconsistency in V and Σ.

Projection. To remedy the limitations of closure-based methods for computing propagation covers of FDs via P views, a practical algorithm was proposed in [12] based on the idea of *Reduction by Resolution* (RBR). We extend RBR and the algorithm of [12] to handle CFDs and projection.

To illustrate RBR, we first define a partial order $\leq$ on constants and '_': $\eta_1 \leq \eta_2$ if either η_1 and η_2 are the same constant 'a', or $\eta_2 =$ '_'.

Given CFDs $\phi_1 = R(X \to A, t_p)$ and $\phi_2 = R(AZ \to B, t'_p)$, if $t_p[A] \leq t'_p[A]$ and for each $C \in X \cap Z$, $t_p[C] \asymp t'_p[C]$, then we can derive $\phi = R(XZ \to B, s_p)$ based on CFD implication [8]. Here $s_p = (t_p[X] \oplus t'_p[Z] \parallel t'_p[B])$, and $t_p[X] \oplus t'_p[Z]$ is defined as follows:

- for each $C \in X - Z$, $s_p[C] = t_p[C]$;
- for each $C \in Z - X$, $s_p[C] = t'_p[C]$;
- for each $C \in X \cap Z$, $s_p[C] = \mathsf{min}(t_p[C], t'_p[C])$, *i.e.,* the smaller one of $t_p[C]$ and $t'_p[C]$ if either $t_p[C] \leq t'_p[C]$ or $t'_p[C] \leq t_p[C]$; it is undefined otherwise.

Following [12], we refer to ϕ as an A-resolvent of ϕ_1 and ϕ_2.

Example 4.2: Consider CFDs $\phi_1 = R([A_1, A_2] \to A,\ t_1)$ and $\phi_2 = R([A, A_2, B_1] \to B,\ t_2)$, where $t_1 = (\_, c \parallel a)$ and $t_2 = (\_, c, b \parallel \_)$. Then $\phi = R([A_1, A_2, B_1] \to B,\ t_p)$ is an A-resolvent of ϕ_1 and ϕ_2, where $t_p = (\_, c, b \parallel \_)$. □

Following [12], we define the following. Given $\pi_Y(R)$ and a set Σ of CFDs on R, let U be the set of attributes in R.

- For $A \in (U - Y)$, let $\mathsf{Res}(\Sigma, A)$ denote the set of all non-trivial A-resolvents. Intuitively, it shortcuts all CFDs involving A.
- Denote by $\mathsf{Drop}(\Sigma, A)$ the set $\mathsf{Res}(\Sigma, A) \cup \Sigma[U - \{A\}]$, where $\Sigma[Z]$ denotes the subset of Σ by including only CFDs with attributes in Z.
- Define $\mathsf{RBR}(\Sigma, A) = \mathsf{Drop}(\Sigma, A)$ and inductively, $\mathsf{RBR}(\Sigma, ZA) = \mathsf{Drop}(\mathsf{Drop}(\Sigma, A), Z)$.

Then we have the following result, in which F^+ denotes the closure of F, *i.e.,* the set of all CFDs implied by F.

Proposition 4.4: *For a view $\pi_Y(R)$ and a set Σ of CFDs on R, (a) for each $A \in (U - Y)$, $\mathsf{Drop}(\Sigma, A)^+ = \Sigma^+[U - \{A\}]$; (b) $\mathsf{RBR}(\Sigma, U - Y)$ is a propagation cover of Σ via $\pi_Y(R)$, where U is the set of attributes in R.* □

```
Input: Source CFDs $\Sigma$, and SPC view $V = \pi_Y(E_s)$, $E_s = \sigma_F(E_c)$.
Output: A minimal propagation cover of $\Sigma$ via $V$.

1.  $\Sigma_V := \emptyset$;   $\Sigma :=$ MinCover($\Sigma$);
2.  EQ := ComputeEQ($E_s, \Sigma$);   /* handling $\sigma_F$ */
3.  if EQ = $\perp$   /* inconsistent */
4.  then return $\{R_V(A \to A, (\_ \parallel a)), R_V(A \to A, (\_ \parallel b))\}$;
        /* for some $A \in Y$, distinct $a, b \in$ dom($A$) */
5.  for each $R_j = \rho_j(S)$ in $E_c$ do
6.      $\Sigma_V := \Sigma_V \cup \rho_j(\Sigma)$;   /* handling product '×' */
7.  for each eq $\in$ EQ do   /* applying domain constraints */
8.      pick an attribute rep(eq) in eq
            such that rep(eq) $\in Y$ if eq $\cap Y$ is not empty;
9.      substitute rep(eq) for each $A \in$ eq, in $\Sigma_V$;
10.     eq := eq $\cap Y$;   /* keep only those attributes in $Y$ */
11. $\Sigma_c :=$ RBR($\Sigma_V$, attr($E_s$) $- Y$);   /* handling $\pi_Y$ */
        /* attr($E_s$) is the set of attributes in $E_s$ */
12. $\Sigma_d :=$ EQ2CFD(EQ);   /* put domain constraints as CFDs */
13. return MinCover($\Sigma_c \cup \Sigma_d$);
```

Figure 2: Algorithm PropCFD_SPC

The result is first established in [12] for FDs. The proof of Proposition 4.4 is an extension of its counterpart in [12].

This yields a procedure for computing a propagation cover of Σ via $\pi_Y(R)$, also denoted by RBR. The idea is to repeatedly "drop" attributes in $U - Y$, shortcutting all CFDs that involve attributes in $U - Y$. The procedure takes as input Σ and $\pi_Y(R)$, and returns RBR($\Sigma, U - Y$) as the output.

We shall also need the following lemma.

Lemma 4.5: *If for any source instance D where $D \models \Sigma$, $V(D)$ is empty, then $\mathcal{R}_V(A \to A,\ (\_ \parallel a))$ and $\mathcal{R}_V(A \to A,\ (\_ \parallel b))$ are in $\mathsf{CFD}_p(\Sigma, V)$, for any attribute A in $\mathcal{R}_V$ and any distinct values $a, b \in \mathsf{dom}(A)$.* □

This essentially assures that the view is always empty (recall the emptiness problem from Section 3.3): no tuple t in the view can possibly satisfy the CFDs, which require $t[A]$ to take distinct a and b as its value. As a result any CFD on the view can be derived from these "inconsistent" CFDs.

4.3 An Algorithm for Computing Minimal Covers

We now present algorithm PropCFD_SPC, shown in Fig. 2.

The algorithm first processes selection σ_F (line 2), extracting equivalence classes EQ via procedure ComputeEQ described earlier (not shown). If inconsistency is discovered, it returns a pair of "conflicting" view CFDs that assure the view to be always empty (lines 3-4), by Lemma 4.5. It then processes the Cartesian product, and gets a set Σ_V of CFDs via renaming as described above (lines 5-6). It applies the domain constraints of EQ to Σ_V (line 9), by designating an attribute rep(eq) for each equivalence class eq in EQ (line 8), which is used uniformly wherever attributes in eq appear in CFDs of Σ_V. It also removes attributes in eq that are not in the projection list Y (line 10), since these attributes do not contribute to view CFDs. Next, it handles the projection π_Y, by invoking procedure RBR (line 11), and converts domain constraints of EQ to CFDs via procedure EQ2CFD (line 12). Finally, it returns a minimal cover of the results returned by these procedures, by invoking procedure MinCover of [8]. This yields a minimal propagation cover of V via Σ (line 13).

Procedure RBR of Fig. 3 implements the RBR method: for each $A \in U - Y$, it computes an A-resolvent (lines 4-8) and Drop(Σ, A) (lines 4-11). Only nontrivial CFDs are included (line 8). By dropping all attributes in $U - Y$, it obtains RBR($\Sigma, U - Y$), a cover of Σ and π_Y by Proposition 4.4.

Procedure EQ2CFD of Fig. 4 converts domain constraints

```
Input: An attribute set $X = U - Y$, and a set $\Sigma$ of CFDs on $U$.
Output: A cover $\Gamma$ of $\Sigma^+[Y]$.

1.  let $\Gamma := \Sigma$;
2.  while $X$ is not empty do {
3.      pick $A \in X$; $X := X - \{A\}$;   $C := \emptyset$;
4.      for each CFD $(W \to A, t_1) \in \Gamma$ do
5.          for each CFD $(AZ \to B, t_2) \in \Gamma$ do
6.              if $t_1[A] \preceq t_2[A]$ and $t_1[W] \oplus t_2[Z]$ is well defined
7.              then $\phi := \mathcal{R}_V(WZ \to B,\ (t_1[W] \oplus t_2[Z] \parallel t_2[B]))$;
8.                  if $\phi$ is not trivial then $C := C \cup \{\phi\}$;
9.      for each CFD $\varphi \in \Gamma$ do
10.         if $A$ occurs in $\varphi$ then $\Gamma := \Gamma - \{\varphi\}$;
11.     $\Gamma := \Gamma \cup C$;}
12. return $\Gamma$.
```

Figure 3: Procedure RBR

```
Input: A set EQ of attribute equivalence classes.
Output: A set $\Sigma_d$ of CFDs expressing EQ.

1.  $\Sigma_d := \emptyset$;
2.  for each attribute equivalence class eq $\in$ EQ do
3.      if key(eq) is a constant then
4.          for each attribute $A \in$ eq
5.              $\Sigma_d := \Sigma_d \cup \{R_V(A \to A, (\_ \parallel$ key(eq)$))\}$;
6.      if key(eq)='_' then
7.          for each $A, B \in$ eq do
8.              $\Sigma_d := \Sigma_d \cup \{R_V(A \to B, (x \parallel x))\}$;
9.  return $\Sigma_d$;
```

Figure 4: Algorithm EQ2CFD

enforced by EQ to equivalent CFDs (lines 2-8), by Lemma 4.2. For each eq in EQ, it leverages the constant key(eq) whenever it is available (lines 4-5). When it is not available, it uses the special variable x in the CFDs (lines 6-8).

Example 4.3: Consider sources $R_1(B_1', B_2)$, $R_2(A_1, A_2, A)$, $R_3(A', A_2', B_1, B)$, and view $V = \pi_Y(\sigma_F(R_1 \times R_2 \times R_3))$, where $Y = \{B_1, B_2, B_1', A_1, A_2, B\}$, and F is ($B_1 = B_1'$ **and** $A = A'$ **and** $A_2 = A_2'$). Consider Σ consisting of $\psi_1 = R_2([A_1, A_2] \to A,\ t_1)$ and $\psi_2 = R_3([A', A_2, B_1] \to B,\ t_2)$, for t_1, t_2 given in Example 4.2.

Applying algorithm PropCFD_SPC to Σ and V, after step 10, EQ consists of $\{\{B_1, B_1'\}, \{B_2\}, \{A_1\}, \{A_2\}, \{B\}\}$, and Σ_V consists of ϕ_1, ϕ_2 of Example 4.2. As also given there, procedure RBR returns ϕ of Example 4.2. Procedure EQ2CFD returns $\phi' = R(B_1 \to B_1', (x \parallel x))$, where R is the view schema with attributes in Y. Then the cover returned by the algorithm consists of ϕ and ϕ'. □

Analysis. For the correctness of the algorithm, we show that for each ϕ in $\mathsf{CFD}_p(\Sigma, V)$, $\Gamma \models \phi$, and vice versa, where Γ is the output of the algorithm. For both directions the proof leverages the lemmas and propositions given above.

For the complexity, let $V = \pi_Y(\sigma_F(E_c))$. Then $|Y| \leq |E_c|$ and $|F| \leq (|E_c|^2 + |E_c|)$. We have the following. (a) Procedure ComputeEQ takes $O(|E_c|^4 * |\Sigma|)$ time. (b) EQ2CFD is in $O(|Y|^3)$ time. (c) Procedure RBR has the same complexity as its counterpart in [12]: $O(|E_c|^2 * a^3)$, where a is an upper bound for the cardinality of Γ during the execution of RBR [12]. (d) The rest of the computation takes at most $O(|\Sigma|^3 + a^3 + |E_c|^2)$ time. Since a is no less than $|E_c| * |\Sigma|$, RBR takes at least $O(|E_c|^5 * |\Sigma|^3)$ time. Putting these together, clearly the cost of RBR dominates. That is, the complexity of PropCFD_SPC is the same as the bound on the algorithm of [12]. Note that both Σ and V are defined at the schema level (it has nothing to do with the instances of source databases), and are often small in practice.

A number of practical cases are identified by [12], where RBR is in polynomial time. In all these cases, PropCFD_SPC also behaves polynomially.

We use minimal cover as an optimization technique. First, Σ is "simplified" by invoking $\mathsf{MinCover}(\Sigma)$ (line 1 of Fig. 2), removing redundant source CFDs. Second (not shown), in procedure RBR, to reduce the size of intermediate Γ during the computation, one can change line 11 of Fig. 3 to "$\Gamma := \mathsf{MinCover}(\Gamma \cup C)$". In our implementation, we partition Γ into $\Gamma_1, \ldots, \Gamma_k$, each of a fixed size k_0, and invoke $\mathsf{MinCover}(\Gamma_i)$. This removes redundant CFDs from Γ, to an extent, without increasing the worst-case complexity since it takes $O(|\Gamma| * k_0^2)$ time to conduct.

As another optimization technique, one may simplify or better still, minimize input SPC views. This works, but only to an extent: the minimization problem for SPC queries is intractable (see, *e.g.,* [1] for a detailed discussion).

5. Experimental Study

In this section, we present an experimental study of algorithm PropCFD_SPC for computing minimal propagation covers of CFDs via an SPC view. We investigate the effects of the number of source CFDs and the complexity of SPC views on the performance of PropCFD_SPC. We also evaluate the impacts of these factors on the cardinality of the minimal propagation covers computed by PropCFD_SPC.

Experimental Setting. We designed two generators to produce CFDs and SPC views, on which our experiments are based. We considered source relational schemas $\mathcal{R}$ consisting of at least 10 relations, each with 10 to 20 attributes.

(a) CFD generator. Given a relational schema $\mathcal{R}$ and two natural numbers m and n, the CFD generator randomly produces a set Σ consisting of m source CFDs defined on $\mathcal{R}$, such that the average number of CFDs on each relation in $\mathcal{R}$ is n. The generator also takes another two parameters LHS and var% as input: LHS is the maximum number of attributes in each CFD generated, and var% is the percentage of the attributes which are filled with '_' in the pattern tuple, while the rest of the attributes draw random values from their corresponding domains. Note that LHS and var% indicate how complex the CFDs are. The experiments were conducted on various Σ's ranging from 200 to 2000 CFDs, with LHS from 3 to 9 and var% from 40% to 50%.

(b) SPC view generator. Given a source schema $\mathcal{R}$ and three numbers $|Y|, |F|$ and $|E_c|$, the view generator randomly produces an SPC view $\pi_Y(\sigma_F(E_c))$ defined on $\mathcal{R}$ such that the set Y consists of $|Y|$ projection attributes, the selection condition F is a conjunction of $|F|$ domain constraints of the form $A = B$ and $A = $ 'a', and E_c is the Cartesian product of $|E_c|$ relations. Here each constant a is randomly picked from a fixed range $[1, 100000]$ such that the domain constraints may interact with each other. The complexity of an SPC view is determined by Y, F and E_c. In the experiments we considered $|Y|$ ranging from 5 to 50, $|F|$ from 1 to 10, and $|E_c|$ from 2 to 11.

The algorithm was implemented in Java. The experiments were run on a machine with a 3.00GHz Intel(R) Pentium(R) D processor and 1GB of memory. For each experiment, we randomly generated 10 sets of source CFDs (resp. SPC views) with fixed parameters, and ran the experiments 5 times on each of the datasets. The average is reported here.

Experimental results. We conducted two sets of experiments: one focused on the scalability of the algorithm and the cardinality of minimal propagation covers *w.r.t.* various source CFDs, while the other evaluated these *w.r.t.* the complexity of SPC views.

Varying CFDs on the Source. To evaluate the impact of source CFDs on the performance of PropCFD_SPC, we fixed $|Y| = 25$, $|F| = 10$, $|E_c| = 4$, and varied the set Σ of source CFDs. More specifically, we considered Σ with $|\Sigma|$ ranging from 200 to 2000, *w.r.t.* var% = 40% and var% = 50%, while the number of attributes in each CFD ranged from 3 to 9 (LHS = 9). The running time and the cardinality of minimal propagation covers are reported in Fig. 5.

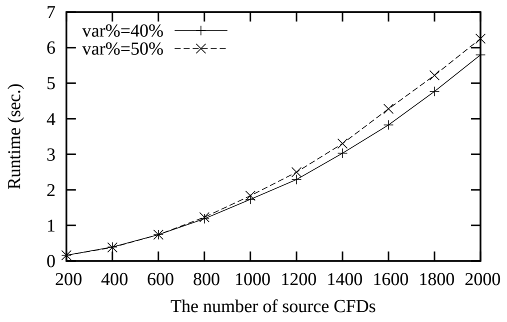
Figure 5(a) shows that PropCFD_SPC scales well with $|\Sigma|$ and is rather efficient: it took less than 7 seconds for $|\Sigma| = 2000$. Further, the algorithm is not very sensitive to (var%, LHS): the results for various (var%, LHS) are quite consistent. As we will see, however, when Y also varies, the algorithm is sensitive to Y and (var%, LHS) taken together.

Figure 5(b) tells us that the more source CFDs are given, the larger the minimal propagation cover found by the algorithm is, as expected. It is interesting to note that the cardinality of the minimal propagation cover is even smaller than the number $|\Sigma|$ of the source CFDs. This confirms the observation of [12]: (minimal) propagation covers are typically much smaller than an exponential of the input size, and thus there is no need to pay the price of the exponential complexity of the closure-based methods to compute covers.

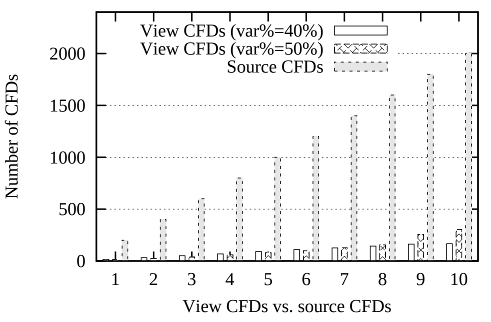
Varying the complexity of SPC views. In the second set of experiments, we evaluated the performance of algorithm PropCFD_SPC *w.r.t.* each of the following parameters of SPC views $\pi_Y(\sigma_F(E_c))$: the set Y of projection attributes, the selection condition F, and the Cartesian product E_c. In each of the experiments, we considered sets Σ of source CFDs with $|\Sigma| = 2000$, *w.r.t.* var% = 40% and var% = 50%, while the number of attributes in each CFD ranged from 3 to 9.

(a) We first evaluated the scalability of PropCFD_SPC with Y: fixing $|F| = 10$ and $|E_c| = 4$, we varied $|Y|$ from 5 to 50. The results are reported in Fig. 6(a), which shows that the algorithm is very sensitive to $|Y|$. On one hand, the larger $|Y|$, the smaller $|U - Y|$, where U is the total number of attributes in E_c (which is determined by $|E_c|$ and the arities of the relations in the source schema $\mathcal{R}$). Note that the outer loop of procedure RBR is dominated by $|U - Y|$: the larger $|U - Y|$ is, the more iterations RBR performs. On the other hand, the larger $|Y|$ is, the more source CFDs are propagated to the view, which has bigger impact on the performance of RBR. As a combination of the above two factors, the running time of PropCFD_SPC does not increase much when $|Y|$ ranges from 5 to 30. However, the running time increases rather rapidly when $|Y|$ is beyond 30. The good news is that even when $|Y|$ is 50 (with $|\Sigma| = 2000$), the algorithm took no more than 80 seconds.

Further, Figure 6(a) shows that when $|Y| < 30$, different settings of var% do not make much difference. However, when $|Y| \geq 30$, their impact on the performance of PropCFD_SPC becomes more obvious. This is because constants may block the transitivity (and thus propagation) of CFDs in procedure RBR (lines 4-8 of Fig. 3); thus more CFDs are propagated to the view when there are less con-



(a) Scalability *w.r.t.* the number of source CFDs



(b) The cardinality of minimal propagation cover

Figure 5: Varying source CFDs

stants (larger var%). When $|Y|$ is small, the impact is not obvious since only a small number of CFDs are propagated to the view. Note that Figures 6(a) and 5(a) are consistent: in the experiments for Fig. 5(a), $|Y|$ was fixed to be 25.

Figure 6(b) shows that when $|Y|$ or var% gets larger, more source CFDs are propagated to the view, as expected. Again it confirms that the minimal covers found are smaller than the source CFDs even when $|Y|$ reached 50.

(b) We next evaluated the scalability of PropCFD_SPC with the selection condition F of SPC views. We fixed $|Y| = 25$ and $|E_c| = 4$, and varied $|F|$ from 1 to 10. The results are reported in Fig. 7. Figure 7(a) shows that when $|F|$ increases, the running time decreases. This is because F introduces domain constraints, which interact with source CFDs, and may either make those CFDs trivial, or combine multiple CFDs into one (see line 9 of Fig. 2). As a result, the larger $|F|$ is, the smaller the set Σ_V is, which is passed to procedure RBR (lines 10-11 of Fig. 2). This leads to the decrease in the running time of RBR, and in turn, the decrease in the running time of PropCFD_SPC. This is rather consistent for the two settings of var%.

Figure 7(b) shows the cardinality of minimal propagation covers *w.r.t.* various $|F|$. The cardinality went up and then down. This is because when $|F|$ increases, to an extent (less than 4 for var% = 40% or 5 for var% = 50%, Fig. 7(b)), more domain constraints are propagated to the view. However, when $|F|$ gets larger, the interaction between domain constraints and CFDs takes a lager toll and as a result, fewer source CFDs are propagated to the view, for the reason given above. This leads to the decrease in the cardinality of the minimal covers when $|F|$ is large. Figure 7(b) also confirms that when var% gets larger, more source CFDs are propagated to the view, which is consistent with Fig. 6(b).

(c) Finally, we evaluated the impact of E_c on the performance of PropCFD_SPC. Fixing $|F| = 10$ and $|Y| = 25$, we varied $|E_c|$ from 2 to 11. The results are reported in Fig. 8.

Figure 8(a) shows that when $|E_c|$ gets larger, the algorithm takes less time. This is because when Y, F and Σ are fixed, increasing $|E_c|$ leads to more CFDs to be dropped from the minimal propagation cover, for involving attributes not in Y. Further, when $|E_c|$ gets larger, *i.e.,* when the total number of attributes involved gets larger, F is less effective in identifying attributes in Y and those not in F. As an example, consider two views defined on the same source: $V_1 = \pi_{AB}(\sigma_{C=D}(R_1(ABCD)))$, and $V_2 = \pi_{AE}(\sigma_{C=H}(R_1(ABCD) \times R_2(EGHL)))$, while the set Σ consists of source CFDs $R_1(A \rightarrow B,\ (\_ \parallel \_))$ and $R_2(E \rightarrow L,\ (\_ \parallel \_))$. Let V_1, V_2 also denote the view schema of V_1, V_2, respectively. Then a minimal cover of the CFDs propagated via V_1 consists of $V_1(A \rightarrow B,\ (\_ \parallel \_))$ whereas no nontrivial CFDs are propagated to the view via V_2.

Figure 8(a) also shows that when $|E_c| \geq 6$, the algorithm is insensitive to E_c, because most of the sources CFDs are dropped, *i.e.,* not propagated to the view (as $|\Sigma|$ is fixed).

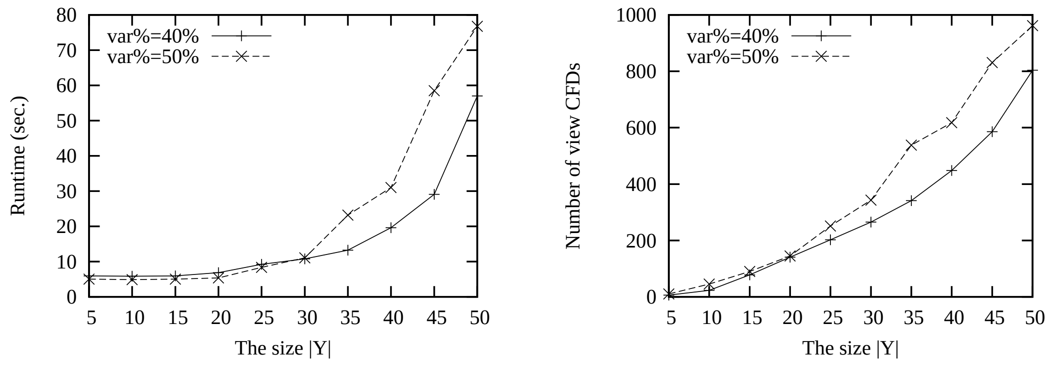
For the same reason, when $|E_c|$ gets larger, the minimal propagation covers get smaller, as shown in Fig. 8(b). However, different from Fig. 6(b) and Fig. 7(b), the number of CFDs propagated in this case is insensitive to different settings of var%. This is because the effect of $|E_c|$ outweighs that of var% in this experiment.

Summary. We have presented several results from our experimental study of algorithm PropCFD_SPC. From the results we find the following. First, the algorithm is quite efficient; for example, it took less than 80 seconds when $|\Sigma| = 2000$, $|Y| = 50$, $|F| = 10$ and $|E_c| = 4$. Second, it scales well with the input set Σ of source CFDs, the selection condition F and the number of relations involved in Cartesian product E_c in the input SPC views. In contrast, the cost increases rapidly when the set Y of projection attributes gets large. Nonetheless, as remarked above, the cost is not unbearable: when $|Y| = 50$ the algorithm still performed reasonably well. Third, we note that minimal propagation covers found by the algorithm are typically small, often smaller than the input set Σ of source CFDs. Finally, while the algorithm is quite sensitive to Y, it is less sensitive to the other complexity factors F and E_c of SPC views. Further, the complexity factors (var%, LHS) of source CFDs do not have significant impact on the performance of the algorithm when the size of Y is less than 30.

6. Related Work

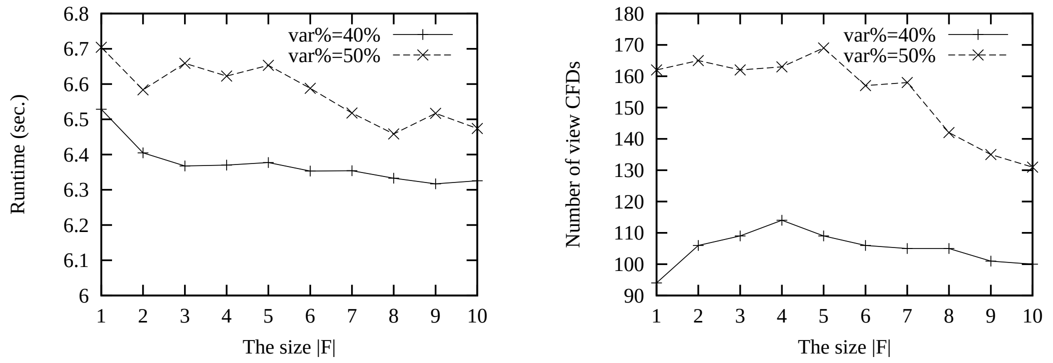
To our knowledge, no previous work has considered (a) CFD propagation, (b) dependency propagation in the general setting, FDs or CFDs, and (c) methods for computing minimal propagation covers for SPC views, even for FDs.

Closest to the study of the CFD propagation problem are [15, 16], which are the first to investigate dependency propagation. The undecidability result for FD propagation via RA views is shown in [15]. An extension of chase is given in [16], for FDs and beyond, based on which the PTIME complexity of FD propagation via SPCU views is derived by [1]. Our proofs of the results in Sections 3 further extend the chase of [16], to accommodate CFDs. As remarked earlier, [15, 16, 1] assume the absence of finite-domain attributes. This work extends [15, 16, 1] by providing complexity bounds for FD propagation in the general setting, and for CFD propagation



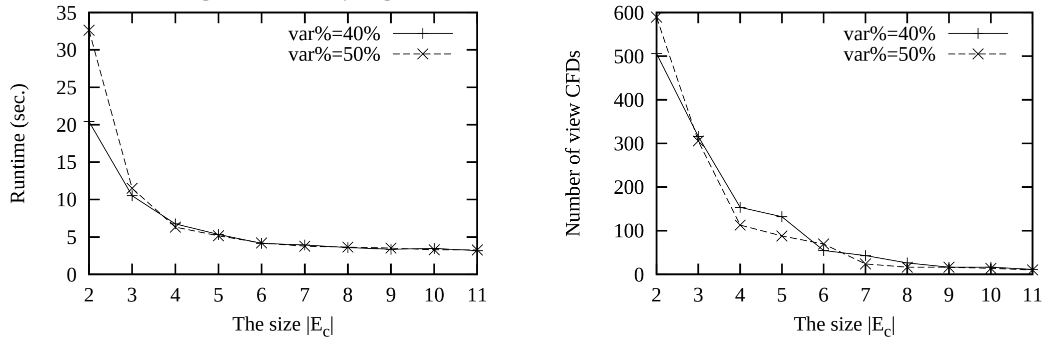
(a) Scalability *w.r.t.* the size $|Y|$ (b) The number of CFDs propagated

Figure 6: Varying the number of projection attributes in Y in SPC views



(a) Scalability *w.r.t.* the size $|F|$ (b) The number of CFDs propagated

Figure 7: Varying the selection condition F in SPC views



(a) Scalability *w.r.t.* the size $|E_c|$ (b) The number of CFDs propagated

Figure 8: Varying the number of relations in the Cartesian product E_c in SPC views

in the infinite-domain setting and in the general setting.

As remarked in Section 4, prior work on propagation covers has focused on FDs and projection views, in the absence of finite-domain attributes [12, 23, 26]. The first algorithm for computing covers *without* computing closures is proposed by [12], based on the RBR method. Our algorithm of Section 4 is inspired by [12], and also uses RBR to handle the interaction between CFDs and the projection operator. This work is the first that deals with selection, Cartesian product and projection operators for computing propagation covers.

Dependency propagation has also been studied for other models [6, 13, 21]. Propagation from XML keys to relational FDs is studied in [6], and composition of views and a powerful set of constraints is investigated for object-oriented databases in [21]. The complexity bounds and techniques developed there do not apply to CFD propagation. An extension of FDs, and their interaction with schema transformation operators (*e.g.,* folding, unfolding) are considered in [13]. Those extended FDs and transformation operators are very different from CFDs and SPC views, respectively.

There has also been a host of work on satisfaction families of dependencies, *e.g.,* [7, 11, 14]. The focus is on the closure problem for dependencies under views. It is to decide, given a set Σ of dependencies and a view V, whether the set Γ of dependencies propagated from Σ via V characterizes the views, *i.e.,* whether the set of databases that satisfy Γ is precisely the set of views $V(D)$ where $D \models \Sigma$. It is shown [11] that FDs are not closed under projection views. This work provides another example for that FDs are not closed under SPCU views: as shown in Section 1, FDs may be propagated to CFDs but not to standard FDs. While CFDs are not closed under SPC views either, the analysis of CFD propagation allows us to preserve certain important semantics of the source data that traditional FDs fail to capture.

The notion of CFDs is proposed in [8], which also studies the implication and consistency problems for CFDs. The propagation problem is *not* considered in [8].

A variety of extensions of classical dependencies have been proposed, for specifying constraint databases [3, 4, 19, 20]. Constraints of [4, 19] cannot express CFDs. More expressive are constraint-generating dependencies (CGDs) of [3] and constrained tuple-generating dependencies (CTGDs) of [20], both subsuming CFDs. A CGD is of the form $\forall \bar{x}(R_1(\bar{x}) \wedge \ldots \wedge R_k(\bar{x}) \wedge \xi(\bar{x}) \rightarrow \xi'(\bar{x}))$, where R_i's are relation atoms, and ξ, ξ' are arbitrary constraints that may carry constants. A CTGD is of the form $\forall \bar{x}(R_1(\bar{x}) \wedge \ldots \wedge R_k(\bar{x}) \wedge \xi \rightarrow \exists \bar{y}(R'_1(\bar{x},\bar{y}) \wedge \ldots \wedge R'_s(\bar{x},\bar{y}) \wedge \xi'(\bar{x},\bar{y})))$, subsuming both CINDs and TGDs. The increased expressive power of CGDs and CTGDs comes at the price of a higher complexity for reasoning about these dependencies. For example, the implication problem for CGDs is already coNP-complete even when all involved attributes have an infinite domain, while in contrast, its CFD counterpart is in quadratic time. Detailed discussions about the connections between CFDs and these extensions can be found in [8]. *No previous work* has studied propagation analysis of these extensions via views.

There has also been recent work on specifying dependencies for XML in terms of description logics [25, 24]. These dependencies also allow constants (concepts). However, the implication problem for such dependencies is undecidable, and as a result, their propagation problem is also undecidable even for views defined as identity mappings.

7. Conclusion

This work is the first effort to study CFD propagation via views. The novelty of the work consists in the following: (a) a complete picture of complexity bounds on CFD propagation, for source dependencies given as either FDs or CFDs, and for views expressed in various fragments of relational algebra; (b) the first complexity results on dependency propagation when finite-domain attributes may be present, a practical and important problem overlooked by previous work; and (c) the first algorithm for computing minimal propagation covers for CFDs via SPC views in the absence of finite-domain attributes, without incurring more complexity than its counterparts for FDs and P views. Our experimental results have verified that the algorithm is efficient. These results are not only of theoretical interest, but also useful for data exchange, integration and cleaning.

A number of issues need further investigation. First, we are currently experimenting with larger datasets and other optimization techniques. Second, when finite-domain attributes are taken into account, the propagation cover algorithm should be generalized. Third, another interesting extension of the propagation cover algorithm is by supporting union. Finally, like CFDs, an extension of inclusion dependencies with conditions, referred to as CINDs, has recently been proposed [5]. It is interesting yet challenging to study propagation of CFDs and CINDs taken together.

Acknowledgments. Wenfei Fan is supported in part by EPSRC GR/S63205/01, GR/T27433/01 and EP/E029213/1.

8. References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.

[3] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-Generating Dependencies. *JCSS*, 59(1):94–115, 1999.

[4] P. D. Bra and J. Paredaens. Conditional dependencies for horizontal decompositions. In *Colloquium on Automata, Languages and Programming*, 1983.

[5] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *VLDB*, 2007.

[6] S. Davidson, W. Fan, C. Hara, and J. Qin. Propagating XML constraints to relations. In *ICDE*, 2003.

[7] R. Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[8] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, to appear.

[9] P. C. Fischer, J. H. Jou, and D.-M. Tsou. Succinctness in dependency systems. *TCS*, 24:323–329, 1983.

[10] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[11] S. Ginsburg and S. Zaiddan. Properties of functional-dependency families. *J. ACM*, 29(3):678–698, 1982.

[12] G. Gottlob. Computing covers for embedded functional dependencies. In *PODS*, 1987.

[13] Q. He and T. Ling. Extending and inferring functional dependencies in schema transformation. In *CIKM*, 2004.

[14] R. Hull. Non-finite specifiability of projections of functional dependency families. *TCS*, 39:239–265, 1985.

[15] A. C. Klug. Calculating constraints on relational expressions. *TODS*, 5(3):260–290, 1980.

[16] A. C. Klug and R. Price. Determining view dependencies using tableaux. *TODS*, 7(3):361–380, 1982.

[17] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, 2005.

[18] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[19] M. J. Maher. Constrained dependencies. *TCS*, 173(1):113–149, 1997.

[20] M. J. Maher and D. Srivastava. Chasing Constrained Tuple-Generating Dependencies. In *PODS*, 1996.

[21] L. Popa and V. Tannen. An equational chase for path-conjunctive queries, constraints, and views. In *ICDT*, 1999.

[22] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[23] A. Silberschatz and H. F. Korth. *Database System Concepts.* McGraw-Hill, 1986.

[24] D. Toman and G. E. Weddell. On reasoning about structural equality in XML: a description logic approach. *TCS*, 336(1):181–203, 2005.

[25] D. Toman and G. E. Weddell. On keys and functional dependencies as first-class citizens in description logics. In *IJCAR*, 2006.

[26] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, 1982.

APPENDIX: Proofs

Tableaux: A Brief Overview

Tableaux have proved useful in studying relational algebra expressions. Below we briefly review the notion of tableaux of [16], which allows multiple rows in the summary.

Consider a relational schema $\mathcal{R} = (R_1, \ldots, R_m)$. A *tableau* T defined on $\mathcal{R}$ is represented as $(\mathsf{Sum}, T_1, \ldots, T_m)$, where for each $i \in [1, m]$, T_i consists of a finite set of free tuples over $R_i(A_1, \ldots, A_k)$, and $k = \mathsf{arity}(R_i)$. For a free tuple $t = (a_1, \ldots, a_k) \in T_i$, a_j is either a constant or a variable for $j \in [1, k]$. If a_j is a constant, then $a_j \in \mathsf{dom}(A_j)$. If a_j is a variable, then $a_j \in \mathsf{Var}$, which is an infinite set of variables. Here Sum is called the *summary*, which also consists of a finite set of free tuples. For a free tuple $s = (w_1, \ldots, w_h) \in \mathsf{Sum}$, where $h = \mathsf{arity}(\mathsf{Sum})$, and for $j \in [1, h]$, w_j is a blank, a constant, or a variable. The variables in Sum are also called distinguished variables, and they must appear in some free tuples from T_i $(1 \le i \le m)$.

Intuitively, the tableau T represents a query, where T_i's specify a pattern, and the summary tuples represent the tuples to be included in the answer to the query, *i.e.,* those for which the pattern specified by T_i's is found in the database (see, *e.g.,* [1] for detailed discussions about tableau queries). For a database instance D of $\mathcal{R}$, we use $T(D)$ to denote the answer to query T in the database D.

For a tableau query T and a query Q in relational algebra, both defined on the same relational schema $\mathcal{R}$, we say that T and Q are *equivalent* iff for any instance D of $\mathcal{R}$, $T(D) = Q(D)$.

The following is known.

Theorem 1: *For every SPC expression E, there exists a tableau T such that E is equivalent to T [16].* □

Note that for an SPC query, one can always find an equivalent tableau query in which the summary consists of a single row [16]. Using the translation rules of [22, 2], it is easy to get the following corollary:

Corollary 2: *Given an SPC expression E, there is a polynomial time algorithm, which transforms E into an equivalent tableau query.* □

Proof of Theorem 3.1

We show that in the absence of finite-domain attributes, the dependency propagation problem from FDs to CFDs is

(a) in PTIME for SPCU views, and
(b) undecidable for RA views.

(a) We first show that it is in PTIME for SPCU views. We develop an algorithm for checking propagation, via tableau representations of given SPCU views and view CFDs, by extending the chase technique. We show that the algorithm characterizes propagation and is in PTIME.

More specifically, assume that the set of source dependencies is Σ, the SPCU view V is $e_1 \cup \ldots \cup e_k$, where e_i is an SPC expression for each $i \in [1, k]$, and the view CFD on V is φ. We show that $\Sigma \models_V \varphi$ can be determined in PTIME. We first show the PTIME bound for $k = 1$, and then extend the proof to $k > 1$.

(a.1) We prove that for $k = 1$, $\Sigma \models_V \varphi$ is in PTIME, *i.e.,* $\Sigma \models_V \varphi$ can be determined in PTIME when V is an SPC expression.

B_{i_1}	$\ldots$	B_{i_g}	B
x_1	$\ldots$	x_g	y_1
x_1	$\ldots$	x_g	y_2

$y_1 = y_2 \asymp t_p[B]$

(a) T_φ for $\varphi = V(B_{i_1} \ldots B_{i_g} \to B,\ t_p)$

Sum

B_1	$\ldots$	B_n
v_{s1}	$\ldots$	v_{sn}

$R_i(A_{i1}, \ldots, A_{in_i})$ for $i \in [1, h]$

A_{i1}	$\ldots$	A_{in_i}
v_{m_i+1}	$\ldots$	$v_{m_i+n_i}$
$v_{m_i+n_i+1}$	$\ldots$	$v_{m_i+2*n_i}$
	$\vdots$	
$v_{m_i+p_i*n_i+1}$	$\ldots$	$v_{m_i+(p_i+1)*n_i}$

(b) T_V for $V(B_1, \ldots, B_n)$

$R_i(A_{i1}, \ldots, A_{in_i})$ for $i \in [1, h]$

A_{i1}	$\ldots$	A_{in_i}
$\rho_1(v_{m_i+1})$	$\ldots$	$\rho_1(v_{m_i+n_i})$
	$\vdots$	
$\rho_1(v_{m_i+p_i*n_i+1})$	$\ldots$	$\rho_1(v_{m_i+(p_1+1)*n_i})$
$\rho_2(v_{m_i+1})$	$\ldots$	$\rho_2(v_{m_i+n_i})$
	$\vdots$	
$\rho_2(v_{m_i+p_i*n_i+1})$	$\ldots$	$\rho_2(v_{m_i+(p_i+1)*n_i})$

(c) Instance I

/* Here $m_i = \Sigma_{j=1}^{i-1}(p_{j-1} + 1) * n_{j-1}$ in (b) and (c) */

Figure 9: Tableau Representations

To do this, we first give tableau representations of the view FD and the SPC view. We also construct a representation of an source instance, in which tuples contain variables. We then present an extension of Chase defined on this instance. Finally, we show that the Chase process is actually a PTIME algorithm for testing propagation, *i.e.,* $\Sigma \models_V \varphi$ iff the chase process terminates on the instance and moreover, it either yields an empty view or leads to a view that satisfies the view dependency φ. Further, the chase process is in PTIME.

Assume that the view $V(B_1, \ldots, B_n)$ be defined on h source relations: $R_1(A_{11}, \ldots, A_{1n_1})$, $\ldots$, $R_h(A_{h1}, \ldots, A_{hn_h})$. Let the view dependency CFD φ be $V(B_{i_1} \ldots B_{i_g} \to B,\ t_p)$. Their corresponding tableau representations T_φ and T_V are shown in Fig. 9. In tableau T_φ, if $t_p[B_{i_j}]$ $(j \in [1, g])$ is a constant, then $x_j = t_p[B_{i_j}]$. Otherwise, x_j is a new variable distinct from x_l for $l \in [1, j-1]$. Further, y_1 and y_2 are two new variables distinct from x_j $(j \in [1, g])$. In tableau T_V, for each $j \in [1, n]$, v_{sj} appearing in Sum is a blank, a constant, or some variable appearing in the free tuples for R_i $(i \in [1, h])$.

Note that the CFD φ is defined on the view, and thus B and each B_{i_j} of T_φ are attributes in $\{B_1, \ldots, B_n\}$ of the summary tuple of T_V, for each $j \in [1, g]$.

In order to test whether φ holds on V or not, it suffices to check whether there exist two tuples $t_1, t_2 \in V$ such that either $t_1[B] \neq t_2[B]$ or $t_1[B] \not\asymp t_p[B]$ while $t_1[B_{i_1} \ldots B_{i_g}] = t_2[B_{i_1} \ldots B_{i_g}] \asymp t_p[B_{i_1} \ldots B_{i_g}]$. To check the existence of

t_1, t_2 in V, we construct tuples in source relations shown in Fig. 9, along with two mappings ρ_1 and ρ_2 from constants and variables of T_V to those of T_φ, such that if t_1, t_2 exist, then t_1 is generated by V applied to those source tuples given by ρ_1, and t_2 is generated by V and ρ_2.

Intuitively, ρ_1 and ρ_2 aim to map the summary tuple $(v_{s1}, \ldots, v_{sn})$ of T_V to $(x_1, \ldots, x_k, y_1)$ and $(x_1, \ldots, x_k, y_2)$ of T_φ respectively. We define ρ_1 and ρ_2 based on the correspondences between the attributes in T_V and those in T_φ (*i.e.,* v_{si} and x_l), as follows. We start with the definition of ρ_1.

Case 1. When $B_j = B_{i_l}$ for some $1 \leq j \leq n$ and $1 \leq l \leq g$, *i.e.,* the attribute B_{i_l} in T_φ corresponds to B_j in the summary of T_V, v_{sj} should be mapped to x_l. We define ρ_1 as follows. (a) If v_{sj} is a variable, then let $\rho_1(v_{sj}) = x_l$. (b) If both v_{sj} and x_l are the same constant, then let $\rho_1(v_{sj}) = v_{sj}$. (c) If both v_{sj} and x_l are constants but $v_{sj} \neq x_l$, then $\rho_1(v_{sj})$ is *undefined*.

Case 2. When $B_j = B$ for some $1 \leq j \leq n$, v_{sj} should be mapped to y_1. More specifically, (a) if v_{sj} is a variable, then let $\rho_1(v_{sj}) = y_1$. (b) If v_{sj} is a constant and $v_{sj} \asymp t_p[B]$, then let $\rho_1(v_{sj}) = v_{sj}$. Otherwise, let $\rho_1(v_{sj})$ be *undefined*.

Case 3. Otherwise, B_j does not correspond to any attribute in T_φ. Then we let $\rho_1(v_j) = v_j$. We also let $\rho_1(v_j) = v_j$ for $n \leq j \leq (m_h + (p_h + 1) * n_h)$.

Similarly, we define the mapping ρ_2 to map the summary tuple $(v_{s1}, \ldots, v_{sn})$ of T_V to $(x_1, \ldots, x_k, y_2)$ of T_φ. Again, we consider three cases.

Cases 1 and 2. Here ρ_2 is defined along the same lines as ρ_1.

Case 3. For each constant v_j in T_V that is not covered by Case 1 or 2, where $1 \leq j \leq (m_h + (p_h + 1) * n_h)$, let $\rho_2(v_j) = v_j$.

Case 4. For all those variables v_{j_1}, v_{j_2} in T_V that are not covered by Case 1 or 2, where $1 \leq j_1, j_2 \leq (m_h + (p_h + 1) * n_h)$, we define ρ_2 such that

- $\rho_2(v_{j_1}) \neq v_{j_1}$, $\rho_2(v_{j_2}) \neq v_{j_2}$;
- $\rho_2(v_{j_1}) \neq \rho_2(v_{j_2})$ if $v_{j_1} \neq v_{j_2}$;
- there does not exist any variable v_j in T_V such that $\rho_2(v_{j_1}) = v_j$ or $\rho_2(v_{j_2}) = v_j$.

That is, ρ_2 uses variables different from ρ_1 in this case. It is easy to see that such ρ_2 exists.

If there exists a constant v_j in T_V such that $\rho_1(v_j)$ (resp. $\rho_2(v_j)$) is *undefined*, we say ρ_1 (resp. ρ_2) is *undefined*. If ρ_1 or ρ_2 is *undefined*, it is easy to verify that no tuples in V match the LHS of φ. From this it follows that $\Sigma \models_V \varphi$. If both ρ_1 and ρ_2 are well defined, we create an instance I of the source relation schemas $(R_1, \ldots, R_h)$, such that the instance I_i of R_i is $\rho_1(T_V.R_i) \cup \rho_2(T_V.R_i)$ (shown in Fig. 9 (*c*)) for $i \in [1, h]$.

Chase. Next, we show how to check whether $\Sigma \models_V \varphi$ by performing Chase on the instance I. Without loss of generality, assume a total order $\leq$ on the variables in I.

Consider an FD $\phi = R_i(X \rightarrow Y)$ in Σ, where $1 \leq i \leq h$. We define the chase of I by ϕ, denoted by Chase_ϕ, as follows. For any tuples t, $t' \in I_i$ of I, if $t[X] = t'[X]$ and $t[A] \neq t'[A]$ for some $A \in Y$, then Chase_ϕ applies ϕ to I as follows. (a) If both $t[A]$ and $t'[A]$ are variables, then let $t[A] = t'[A]$ if $t'[A] \leq t[A]$. Otherwise, let $t'[A] = t[A]$. (b) If $t[A]$ is a constant and $t'[A]$ is a variable, then let $t'[A] = t[A]$. (c) If $t[A]$ is a variable and $t'[A]$ is a constant, then let $t[A] = t'[A]$. (d) If both $t[A]$ and $t'[A]$ are (distinct) constants, Chase_ϕ is *undefined*.
Here by $t[A] = t'[A]$ (resp. $t'[A] = t[A]$) means replacing each appearance of $t[A]$ (resp. $t'[A]$) in I with $t'[A]$ (resp. $t[A]$).

This process repeats for all FDs in Σ until one of the following cases happens: (1) no further changes can be incurred to I, (2) the Chase process is *undefined*, or (3) y_1 and y_2 are identified during the process.

Observe that Chase process must terminate. Indeed, there are at most $2 * (m_h + (p_h + 1) * n_h)$ variables in I. Let $|I|$ $= 2 * (m_h + (p_h + 1) * n_h)$, then there are at most $O(|I|^2)$ value assignments, and each can be done in $O(|\Sigma||I|^2)$ time. From this, it follows that the Chase process is in $O(|\Sigma||T|^4)$ time.

We next show that the chase process suffices to determine whether or not $\Sigma \models_V \varphi$. Consider the following cases. If the Chase process is *undefined*, then the view V must be empty. As a result, obviously $\Sigma \models_V \varphi$. If the *chase* terminates due to $y_1 = y_2$, then $\Sigma \models_V \varphi$ since for any two tuples $t_1, t_2 \in V$, we have that $t_1[B] = t_2[B] \asymp t_p[B]$, if $t_1[B_{i_1} \ldots B_{i_g}] = t_2[B_{i_1} \ldots B_{i_g}] \asymp t_p[B_{i_1} \ldots B_{i_g}]$. Otherwise, the chase of I by FDs yields a counterexample for the propagation, and thus $\Sigma \not\models_V \varphi$.

From these it follows that the Chase process is a PTIME algorithm that determines whether $\Sigma \models_V \varphi$, when V is an SPC expression.

(a.2) We next prove that for $k > 1$, $\Sigma \models_V \varphi$ can also be determined in PTIME.

When V is $e_1 \cup \ldots \cup e_k$, V can be represented as a set of tableaux $T = (T_1, \ldots, T_k)$, where T_i is the tableau representation of SPC expression e_i for $i \in [1, k]$. Here to construct source relations such that there exists a pair of view tuples t_1 and t_2 in V that serves as a counterexample for $\Sigma \models_V \varphi$, we have to consider t_1 and t_2 generated by any two of the k SPC expressions, namely, t_1 may be generated via e_i and t_2 may be produced by e_j while i and j are not necessarily the same. In total, there are k^2 combinations of the SPC expressions, and the checking needs to be performed on each of these combination. The Chase given above for $k = 1$ can be easily extended to check each combination. From these it follows that there is a PTIME algorithm to check whether $\Sigma \models_V \varphi$.

(b) The undecidability follows from its counterpart for FDs (see [1]), since CFDs subsume FDs. □

Proof of Theorem 3.2

We show that in the general setting, the dependency propagation problem from FDs to FDs is coNP-complete for SC views.

We first present an NP algorithm that, given source FDs Σ, a view FD φ and an SC view V, decides whether $\Sigma \not\models_V \varphi$ or not. The algorithm is based on tableaux and instance constructed in the same way as in the proof of Theorem 3.1. The difference here is that we have to deal with variables that are associated with finite domain attributes, such that all those variables are instantiated with constants from their corresponding finite domains. There are exponentially many such instantiations that need to be checked. To cope with this, we first guess an instantiation, and then check, *w.r.t.* this instantiation, whether or not $\Sigma \not\models_V \varphi$. The latter can be done in polynomial time by using the PTIME algorithm given in the proof of Theorem 3.1. Note that $\Sigma \not\models_V \varphi$ if and only if there exists an instantiation such that $\Sigma \not\models_V \varphi$ *w.r.t.* the instantiation. From this it follows that this problem is in coNP.

The lower bound is shown by reduction from 3SAT to the complement of the propagation problem (*i.e.,* the problem to decide whether $\Sigma \not\models_V \varphi$), where 3SAT is NP-complete (cf. [10]). More specifically, consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, where all the variables in ϕ are $x_1, \ldots, x_m$, C_j is of the form $y_{k_1} \vee y_{k_2} \vee y_{k_3}$, and moreover, for $i \in [1, 3]$, y_{k_i} is either $x_{p_{ki}}$ or $\overline{x_{p_{ki}}}$, for $p_{ki} \in [1, m]$. We shall construct a database schema $\mathcal{R} = (R_0, R_1, \ldots, R_n)$, a set Σ of FDs on $\mathcal{R}$, a view V defined by an SC expression, and a FD φ on V. Then we show that ϕ is satisfiable if and only if $\Sigma \not\models_V \varphi$.

Let the schema of R_0 be (X, A, Z), where $\mathsf{Dom}(X) = \{1, \ldots, m, \ldots\}$ (*e.g.,* int), $\mathsf{Dom}(A) = \mathsf{Dom}(Z) = \{0, 1\}$, *i.e.,* A and Z are finite domain attributes. Moreover, an FD $\varphi_0 = R_0(X \to A)$ is defined on R_0. Intuitively, for each tuple $t \in R_0$, $t[X]$, $t[A]$ and $t[Z]$ encode a variable, its corresponding truth assignment, and the truth value of ϕ, respectively. The FD φ_0 guarantees that each variable has a unique truth assignment.

Let the schema of R_i be (A_1, A_2, X^i, A^i) for $i \in [1, n]$, where $\mathsf{Dom}(A_1) = \mathsf{Dom}(A_2) = \mathsf{Dom}(A^i) = \{0, 1\}$, and $\mathsf{Dom}(X^i) = \{1, \ldots, m, \ldots\}$ (*e.g.,* int), *i.e.,* A_1, A_2 and A are finite domain attributes. Moreover, we define two FDs $\varphi_{i_1} = R_i(A_1, A_2 \to X^i A^i)$, and $\varphi_{i_2} = R_i(X^i \to A^i)$ on R_i. Intuitively, for each tuple $t \in R_i$, $t[X^i]$ and $t[A^i]$ is to encode a variable and its corresponding truth assignment that make the clause C_i true. Here A_1 and A_2 serve as a "counter" to assure that there are three variables that could satisfy C_i (note that while A_1 and A_2 encode numbers from 0 to 3, one of these is redundant as C_i is defined in terms of 3 literals). The FD φ_{i_1} assures that A_1 and A_2 are used as a "key" such that only variables in C_i and their appropriate truth assignments are considered. Again, the FD φ_{i_2} assures that each variable has a unique truth assignment.

Let the set Σ of source FDs be $\{\varphi_0, \varphi_{1_1}, \varphi_{1_2}, \ldots, \varphi_{n_1}, \varphi_{n_2}\}$.

We next define the SC expression V to be $e \times e_{0_1} \times e_{0_2} \times e_1 \times \ldots \times e_n$, where

- ∘ $e = R_0$,
- ∘ $e_{0_1} = \sigma_{X=1}(R_0) \times \ldots \times \sigma_{X=m}(R_0)$,
- ∘ $e_{0_2} = \sigma_{R_0.X=R_1.X^1 \wedge R_0.A=R_1.A^1}(R_0 \times R_1) \times \ldots \times \sigma_{R_0.X=R_n.X^n \wedge R_0.A=R_n.A^n}(R_0 \times R_n)$, and
- ∘ for $j \in [1, n]$, $e_j = \sigma_{X^j=p_{k1} \wedge A^j=a_1 \wedge A_1=0 \wedge A_2=0}(R_j)$ $\times \sigma_{X^j=p_{k2} \wedge A^j=a_2 \wedge A_1=0 \wedge A_2=1}(R_j)$ $\times \sigma_{X^j=p_{k3} \wedge A^j=a_3 \wedge A_1=1 \wedge A_2=0}(R_j)$ $\times \sigma_{X^j=p_{k1} \wedge A^j=a_1 \wedge A_1=1 \wedge A_2=1}(R_j)$,
  such that for $i \in \{1, 2, 3\}$, $a_i = 1$ if $y_{ki} = x_{p_{ki}}$ and $a_i = 0$ if $y_{ki} = \overline{x_{p_{ki}}}$.

Intuitively, (a) the subexpression e_{0_1} assures that R_0 contains a truth assignment for all variables of ϕ, *i.e.,* $x_1, \ldots, x_m$ appear in e_{0_1}; (b) e_{0_2} is to assure that the truth assignments of variables in R_0 and the truth assignments of variables in R_j are consistent, for $j \in [1, n]$; and (c) for each $j \in [1, n]$, e_j encodes the clause C_j: it enumerates all the truth assignments that make C_j true (note that when $A_1 = 1$ and $A_2 = 1$, the truth assignment is redundant: it repeats the assignment when $A_1 = 0$ and $A_2 = 0$).

Based on the definition of V and Σ, it is easy to verify the following: for any instance I of $(R_0, R_1, \ldots, R_n)$, if $V(I)$ is not empty then the instance of R_0 in I contains a truth assignment that makes ϕ true.

Finally, we define φ to be $V(X, A \to Z)$, where attributes X, A, Z come from the subexpression e.

Now we verify that ϕ is satisfiable if and only if the view FD φ is not propagated from the source FDs Σ via V.

Suppose that $\Sigma \not\models_V \varphi$. Then there exists an instance I such that $I \models \Sigma$, while $V(I) \not\models \varphi$. Since $V(I) \not\models \varphi$, we have that $V(I)$ is not empty. As remarked earlier, the instance of R_0 in I contains a truth assignment that makes ϕ true. Thus ϕ is satisfiable.

Conversely, if ϕ is satisfiable, then there is a truth assignment ξ that makes ϕ true. We construct an instance I of $(R_0, R_1, \ldots, R_n)$ such that $I \models \Sigma$ but $V(I) \not\models \varphi$. Let the instance of R_0 in I consist of $2m$ tuples $\{t_1, \ldots, t_{2m}\}$. For each $i \in [1, \ldots, m]$, let (a) $t_i[X] = i$, $t_i[A] = \xi(x_i)$ and $t_i[Z] = 0$; and (b) $t_{m+i}[X] = i$, $t_{m+i}[A] = \xi(x_i)$ and $t_{m+i}[Z] = 1$. That is, t_i and t_{m+i} agree on their X and A attributes, *i.e.,* they encode the same truth assignment for X_i; however, they differ in their Z attributes. For $j \in [1, n]$, let the instance of R_j in I contain exactly four tuples as specified by e_j, which are all true assignments that make C_j true only. It is easy to verify that $I \models \Sigma$ and $V(I) \not\models \varphi$. From this it follows that φ is not propagated from Σ via V.

Therefore, the problem is coNP-complete. □

Proof of Theorem 3.3

We show that in the general setting, the dependency propagation problem from FDs to CFDs is

(a) in PTIME for PC views,
(b) in PTIME for SP views,
(c) coNP-complete for SC views,
(d) coNP-complete for SPCU views,
(e) undecidable for RA views.

(a) We develop a PTIME algorithm that, given source FDs Σ, a view CFD φ and an PC view V, checks whether $\Sigma \models_V \varphi$ or not. In fact the PTIME algorithm given in the proof of Theorem 3.1 suffices here. To see this, assume that V is $\pi_{B_1, \ldots, B_m}(R_1 \times \ldots \times R_n)$, and that φ is $V(B_{i_1} \ldots B_{i_g} \to B, t_p)$, where $B_{i_j}, B \in \{B_1, \ldots, B_m\}$ for $j \in [1, g]$. It is easy to verify that if $\Sigma \models_V \varphi$, all of $B_{i_1} \ldots B_{i_g}$ and B must be attributes of the same relation R_i ($1 \le i \le n$). Therefore, without loss of generality, we assume that there is only one relation R involved in the PC view V. We construct an instance I as in the proof of Theorem 3.1. In this case, I only need to contain two tuples of R. As a result, the instantiations of finite domain variables are not necessary because each domain has at least two elements: we can simply construct the two tuples with distinct values whenever necessary, regardless of what values they are. Then the algorithm given in the proof of Theorem 3.1 suffices to determine whether or not $\Sigma \models_V \varphi$.

(b) Similar to the proof of (a), one can verify that the algorithm given in the proof of Theorem 3.1 suffices to determine whether $\Sigma \models_V \varphi$ or not in this case. Indeed, when V is defined as an SP expression, only one source relation is involved in V. Then the argument for (a) suffices to show that the algorithm in the proof of Theorem 3.1 is a PTIME algorithm for determining the propagation in this case.

(c, d) We first show the dependency propagation problem from FDs to CFDs is in coNP for SPCU views. Then we show the dependency propagation problem from FDs to CFDs is in coNP-hard for SC views.

The coNP upper bound is verified by giving an NP algorithm for deciding $\Sigma \not\models_V \varphi$ for any given source FDs Σ, view CFD φ and SPCU view V. The proof is similar to that of Theorem 3.2. The only difference here is that we represent the CFD as a tableau, instead of an FD. Again we have deal with variables associated with finite domain attributes, such that all those variables are instantiated with constants from their corresponding finite domains. There are exponentially many such instantiations that need to be checked. To cope with this, we first guess an instantiation, and then check, *w.r.t.* this instantiation, whether or not $\Sigma \not\models_V \varphi$. The latter can be done in polynomial time by using the PTIME algorithm given in the proof of Theorem 3.1. Note that $\Sigma \not\models_V \varphi$ if and only if there exists an instantiation such that $\Sigma \not\models_V \varphi$ *w.r.t.* the instantiation. From this it follows that this problem is in coNP.

The coNP lower bound follows from the proof of Theorem 3.2 for SC views, since CFDs subsume FDs.

(e) The undecidability follows from Theorem 3.1, since the general setting subsumes the infinite-domain setting. □

Proof of Theorem 3.5

We next show that in the absence of finite-domain attributes, the dependency propagation problem from CFDs to CFDs is

(a) in PTIME for SPCU views, and

(b) undecidable for RA views.

(a) The PTIME bound is again verified by developing a polynomial time checking algorithm. The algorithm is similar to that given in the proof of Theorem 3.1. The only difference here is that a minor extension of the chase rules is used here to cope with CFDs instead of FDs.

(b) The undecidability follows from Theorem 3.1, since FDs are a special case of CFDs. □

Proof of Corollary 3.6

We show that in general setting, the dependency propagation problem from CFDs to CFDs is

- coNP-complete for views expressed as S, P or C queries;
- coNP-complete for SPCU views; and
- undecidable for RA views.

(a, b) We first show that the propagation problem is coNP-hard for views expressed as S, P, or C queries. Then we show the propagation problem for SPCU views is in coNP.

We show the lower bound by reduction from the implication problem for CFDs. The *implication problem* for CFDs is to determine, given a set Σ of CFDs and a single CFD φ defined on a relation schema R, whether or not Σ entails φ, denoted by $\Sigma \models \varphi$, *i.e.,* whether or not for all instances I of R, if $I \models \Sigma$ then $I \models \varphi$. It is known that in the presence of finite-domain attributes, CFD implication is already coNP-complete [8]. Observe that the implication problem for CFDs is a special case of the dependency propagation problem by limiting the views to the identity mappings, which are expressible as S, P, C or SPCU queries. From these it follows that the propagation problem is coNP-hard for views expressed as S, P or C queries.

The coNP upper bound is verified along the same lines as the proof of Theorem 3.3 for SPCU views. The only difference here is that we need to extend chase to deal with source CFDs instead of source FDs.

(c) The undecidability follows from Theorem 3.3, since FDs are a special case of CFDs. □

Proof of Theorem 3.7

We show that in the general setting, the emptiness problem is coNP-complete for CFDs and SPCU views.

To do this, we first show that the emptiness problem for CFDs and SPCU views is coNP-hard. We then show that this problem is in coNP.

The lower bound is verified by reduction from the satisfiability problem for CFDs to the complement of the emptiness problem. The *satisfiability problem* for CFDs is to determine, given a set Σ of CFDs defined on a relation schema R, whether or not there exists a nonempty instance I of R such that $I \models \Sigma$. It is known that the satisfiability problem for CFDs is NP-complete [8]. Obviously the satisfiability problem is a special case of the complement of the emptiness problem (*i.e.,* the non-emptiness problem), where views are defined as the identity mapping. Putting these together, we have that the emptiness problem is coNP-hard.

The upper bound is verified by providing an NP algorithm to check the non-emptiness based on an extended Chase process. More specifically, assume that the SPCU expression V is $e_1 \cup \ldots \cup e_k$, where for each $i \in [1, k]$, e_i is an SPC expression. Note that if V is not empty, then there must exist at least one e_i $(1 \leq i \leq k)$ such that e_i is not empty.

To present the NP algorithm, we first give a tableau representation T_{e_i} for each SPC view e_i. We then extend the Chase technique to handle T_{e_i}. Finally, we show that the Chase process is actually an NP algorithm for checking the non-emptiness.

The construction of tableau T_{e_i} follows the same way as that of T_V given in the proof of Theorem 3.1. Without loss of generality, we assume that there is a total order $\leq$ on the variables in T_{e_i}. Variables in T_{e_i} that are associated with finite domain attributes are instantiated with constants from their corresponding finite domains. There are exponentially many such instantiations. To cope with this, we first guess an instantiation; we then check, *w.r.t.* this instantiation, whether or not V yields a non-empty instance on some source database. To show that the algorithm is in NP, it suffices to show that the checking step can be done in PTIME, for each instantiation.

For each instantiation of tableau T_{e_i}, the following Chase process is performed to apply each CFD $\varphi = R(X \rightarrow A, t_p)$ in Σ. We next extend the chase rules for CFDs. There are two cases to consider, depending on $t_p[A]$.

Case 1. The pattern value $t_p[A]$ is an unnamed variable '_'. In this case, for any free R tuples t, t' of T_{e_i}, if $t[X] = t'[X]$, $t[X] \asymp t_p[X]$ but $t[A] \neq t'[A]$, we do the following. (a) If both $t[A]$ and $t'[A]$ are variables, then we let $t[A] = t'[A]$ if $t'[A] \leq t[A]$, or $t'[A] = t[A]$ if $t[A] \leq t'[A]$. (b) If $t[A]$ is a constant and $t'[A]$ is a variable, then let $t'[A] = t[A]$. (c) If $t[A]$ is a variable and $t'[A]$ is a constant, then let $t[A] = t'[A]$. (d) If both $t[A]$ and $t'[A]$ are (distinct) constants, then we say that the Chase process is *undefined.*

Case 2. The pattern value $t_p[A]$ is a constant. In this case, for any free R tuple t of T_{e_i}, if $t[X] \asymp t_p[X]$ and $t[A] \neq t_p[A]$, we do the following. (a) If $t[A]$ is a variable, then we let

$t[A] = t_p[A]$. (b) If $t[A]$ is a (distinct) constant, then we say that the Chase process is *undefined*.

Here by $t[A] = t'[A]$ (resp. $t'[A] = t[A]$, $t[A] = t_p[A]$) we mean replacing each appearance of $t[A]$ (resp. $t'[A]$, $t[A]$) in T_{e_i} with $t'[A]$ (resp. $t[A]$, $t_p[A]$).

This process repeats for all source CFDs in Σ until either no further changes can be incurred to T_{e_i}, or the Chase process becomes *undefined* for some CFD. It is easy to see that one of these cases must happen.

We now show that the Chase process actually yields an algorithm for checking the non-emptiness. If the Chase process terminates because no changes can be made to T_{e_i}, we can easily construct a source instance I such that $I \models \Sigma$ and $V(I)$ is non-empty. Indeed, the instance I can be constructed by instantiating variables in the final chasing result of T_{e_i} with pairwise different constants. If the Chase process terminates because it becomes *undefined* for some CFD, then the view must be empty *w.r.t.* the instantiation. From these it follows that e_i is non-empty if and only if there exists an instantiation such that the Chase process terminates because of no changes.

Finally we verify that the Chase process above is in NP. It suffices to show that for each instantiation, the Chase process for T_{e_i} terminates in PTIME. Without loss of generality, we assume that T_{e_i} is the same as T_V shown in Fig. 9. There are at most $(m_h + (p_h + 1) * n_h)$ variables in T_{e_i}. Let $|T_{e_i}| = (m_h + (p_h + 1) * n_h)$, then there are at most $O(|T_{e_i}|^2)$ value assignments, and each can be done in $O(|\Sigma||T_{e_i}|^2)$ time. Thus the Chase process must terminate in $O(|\Sigma||T_{e_i}|^4)$ time.

Putting these together, we have that the emptiness problem is coNP-complete for CFDs and SPCU views. $\Box$

Proof of Theorem 3.8

We show that without finite-domain attributes, the emptiness problem is in PTIME for CFDs and SPCU views.

In the absence of finite-domain attributes, one can readily turn the NP algorithm given in the proof of Theorem 3.7 into a PTIME one, since instantiation of finite-domain attributes, which would require a nondeterministic guess, is no longer needed here. $\Box$