Propagating Functional Dependencies with Conditions

Wenfei Fan1,2,3  Shuai Ma1  Yanli Hu1,5  Jie Liu4  Yinghui Wu4
1University of Edinburgh  2Bell Laboratories  3Harbin Institute of Technologies
4Chinese Academy of Sciences  5National 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 

\[ \text{PTIME} \] 

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 \textit{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 \textit{source dependencies}, and those on the view as \textit{view dependencies}.

This problem has been extensively studied when source and view dependencies are functional dependencies (FDs), for views defined in relational algebra (e.g., [6, 8, 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 \textit{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 \textit{conditional functional dependencies} (CFDs) recently proposed [7], as shown below.

\textbf{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:

\[
R_i(\text{AC: string, phn: string, name: string, street: string, city: string, zip: string})
\]

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 \( \text{street} \) (\( f_1 \)), and area code uniquely determines \( \text{city} \) (\( f_2 \)); moreover, area code determines \( \text{city} \) in \( R_3 \) data (\( f_3 \)).

\[
f_1: R_1(\text{zip} \rightarrow \text{street}), \quad f_2: R_1(\text{AC} \rightarrow \text{city}), \quad f_3: R_3(\text{AC} \rightarrow \text{city})
\]

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

\[
\text{select AC, phn, name, street, city, zip, '44' as } \text{AC from } R_1
\]

\[
\text{select AC, phn, name, street, city, zip, '1096' as } \text{AC from } R_2
\]

\[
\text{select AC, phn, name, street, city, zip, '1316' as } \text{AC from } R_3
\]

\[
\text{select AC, phn, name, street, city, zip, 'W1B 1JL' as } \text{AC from } R_1
\]

\[
\text{select AC, phn, name, street, city, zip, '1096' as } \text{AC from } R_2
\]

\[
\text{select AC, phn, name, street, city, zip, '1316' as } \text{AC from } R_3
\]

\[
\text{select AC, phn, name, street, city, zip, 'W1B 1JL' as } \text{AC from } R_1
\]

\[
\text{select AC, phn, name, street, city, zip, 'W1B 1JL' as } \text{AC from } R_2
\]

\[
\text{select AC, phn, name, street, city, zip, 'W1B 1JL' as } \text{AC from } R_3
\]

This concludes the paper.
Define $Q_2$ and $Q_4$ by substituting ‘01’ and ‘31’ for ‘44’, $R_2$ and $R_4$ 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 $t_1$, $t_4$ extracted from $D_2$; in fact, on 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 [7] holds on the view:

$$\varphi_1: R([CC = '44', zip] \rightarrow [street]).$$

That is, for UK customers in the view, $zip$ code uniquely determines $street$. In other words, $\varphi_1$ is an “FD” with a condition: it is to hold only on the subset of tuples in the view that satisfies the pattern $CC = '44'$, rather than on the entire view. It cannot be expressed as a standard FD.

Similarly, from $f_2$ and $f_3$ one 

That is, $f_2$ and $f_3$ hold conditionally on the view: area code determines city for tuples with $CC = '44'$ ($\varphi_2$) or $CC = '31'$ ($\varphi_3$). In other words, the semantics specified by the CFDs on the sources is preserved by the CFDs on the view.

Furthermore, given the following CFDs on the sources:

$$\begin{align*}
\text{cfd}_1 & : R_1([AC = '20'] \rightarrow [city \ldots]), \\
\text{cfd}_2 & : R_2([AC = '20'] \rightarrow [city \ldots]),
\end{align*}$$

then the following CFDs are propagated to the view:

$$\begin{align*}
\varphi_2: R([CC = '44', AC = '20'] \rightarrow [city \ldots]), \\
\varphi_3: R([CC = '31', AC = '20'] \rightarrow [city \ldots]),
\end{align*}$$

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 $\varphi_2$ and $\varphi_3$ 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 instance $D_1$ and $D_2$ of $R_1$ and $R_2$ that satisfy the CFDs $f_2$ and $f_3$, respectively, and for any source instance $D_3$ 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 $\varphi_2$ and $\varphi_3$.

(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 $CC = '44'$, $AC = '20'$ and city = 'EDH', can be rejected without checking the data, since it violates the CFD $\varphi_4$ propagated from the sources.

(3) Data cleaning. In contrast to FDs that were developed for schema design, CFDs were proposed for data cleaning [7]. Suppose that CFDs $\varphi_1$–$\varphi_3$ 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 $\varphi_6$: $R([CC, AC, phn \rightarrow street, city, zip])$ is also defined on the target, then $\varphi_6$ 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 [8] 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 [22, 25].

Already hard for FDs and projection views, the propaga-
tion 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 50 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) [7] 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 $\text{dom}(A)$, which is either infinite (e.g., string, real) or finite (e.g., Boolean, date).

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

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 $R(A \rightarrow B, (x \parallel x))$, 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:

$\varphi_1: R([\text{CC}, \text{zip}] \rightarrow [\text{street}]), (44, \_\_\_\_\_\_\_\_),$

$\varphi_2: R([\text{CC}, \text{AC}] \rightarrow [\text{city}]), (44, \_\_\_\_\_\_\_\_),$

$\varphi_3: R([\text{CC}, \text{AC}] \rightarrow [\text{city}]), (44, 20 \parallel \text{LDN}),$

$f_1: R(zip \rightarrow \text{street}, (\_\_\_\_\_\_)$.

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 $\times$ on constants and ‘$\_\_\_\_\_\_\_\_$’, $n_1 \times n_2$ if either $n_1 = n_2$, or one of $n_1, n_2$ is ‘$\_\_\_\_\_\_\_\_$’. The operator $\times$ naturally extends to tuples, e.g., (Portland, LDS) $\times$ (NYC, LDS) but (Portland, LDS) $\neq$ (NYC, LDS). We say that a tuple $t_1$ matches $t_2$ if $t_1 \times t_2$.

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

Intuitively, $\varphi$ is a constraint defined on the set $D_{\varphi} = \{ t \in D | t[X] \times t_p[X] \}$ 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) $t_1[Y] \times t_p[Y]$. 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 $\varphi$ is defined on the subset $D_{\varphi}$ of $D$ identified by $t_p[X]$, rather than on the entire $D$.

An instance $D$ of $R$ satisfies CFD $R(A \rightarrow B, (x \parallel x))$ 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 $\Sigma$ of CFDs defined on $\mathcal{R}$, denoted by $D \models \Sigma$, if $D \models \phi$ for each $\phi$ in $\Sigma$.

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 $\varphi_1, \varphi_2, \varphi_3$ of Example 2.1. However, if we remove attribute $CC$ from $\varphi_4$, then the view no longer satisfies the modified CFD. Indeed, there are two tuples $t'$ and $t''$ in $V(D_1, D_2, D_3)$ such that $t'$ and $t_1$ of Fig. 1 have identical $AC$ and city values; similarly for $t_2$ and $t_3$ of Fig. 1. Then $t'$ and $t''$ 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
Table 1: Complexity of CFD propagation

<table>
<thead>
<tr>
<th>Σ</th>
<th>View language</th>
<th>Complexity bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Infinite domain only</td>
</tr>
<tr>
<td>Propagation from FDs to CFDs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDs</td>
<td>SPC</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>SC</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>PC</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>SPCU</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>RA</td>
<td>undecidable</td>
</tr>
<tr>
<td>Propagation from CFDs to CFDs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFDs</td>
<td>S</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>SPC</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>SPCU</td>
<td>PTIME</td>
</tr>
<tr>
<td></td>
<td>RA</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

Table 2: Complexity of FD propagation

<table>
<thead>
<tr>
<th>View language</th>
<th>Complexity bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Infinite domain only</td>
</tr>
<tr>
<td>Propagation from FDs to FDs</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>SPCU</td>
</tr>
<tr>
<td></td>
<td>SC</td>
</tr>
<tr>
<td></td>
<td>PC</td>
</tr>
<tr>
<td></td>
<td>SPCU</td>
</tr>
<tr>
<td></td>
<td>RA</td>
</tr>
</tbody>
</table>

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 R, a set Σ of source dependencies on R, and CFD φ on the view, whether or not φ is propagated from Σ via V, denoted by Σ |= V φ, i.e., for any instance D of R, if D |= Σ then V(D) |= φ.

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 R, i.e., in the infinite-domain setting and the general setting.

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

- undecidable for RA views, and
- in PTIME for SPCU 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 FDS 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 coNP-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 Σ ∪ V ψ 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 R, a set Σ of FDS on R, a view V defined by an SC expression, and a FD ψ on V. We show that ϕ is satisfiable iff Σ ∪ V ψ.
In contrast to the \textsc{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 \textsc{fds} to \textsc{cfd}s, and should be thoroughly studied.

**Theorem 3.3:** In the general setting, the dependency propagation problem from \textsc{fds} to \textsc{cfd}s is

- in \textsc{ptime} for \textsc{pc} views,
- in \textsc{ptime} for \textsc{sp} views,
- \textsc{coNP}-complete for \textsc{sc} views,
- \textsc{coNP}-complete for \textsc{spcu} views,
- undecidable for \textsc{ra} views.

\textbf{Proof Sketch:} (a) The \textsc{ptime} bounds are verified by providing \textsc{ptime} algorithms for checking propagation, by extending chase to \textsc{cfd}s. (b) The \textsc{coNP} upper bound is by giving an \textsc{np} algorithm for deciding $\Sigma \not\models \varphi$ for the given source \textsc{fds} $\Sigma$, view \textsc{cfd} $\varphi$ and \textsc{spcu} view $V$. The lower bound follows from Theorem 3.2 for \textsc{sc} views, since \textsc{cfd}s subsume \textsc{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 \textsc{fds} to \textsc{cfd}s is (a) in \textsc{ptime} for \textsc{s}, \textsc{p} and \textsc{c} views, and (b) \textsc{coNP}-complete for \textsc{sp} views.

In addition, since \textsc{fds} are a special case of \textsc{cfd}s, Theorems 3.2 and 3.3 together yield a complete picture of complexity bounds on the dependency propagation problem from \textsc{fds} to \textsc{cfd}s in the general setting.

**Corollary 3.4:** In the general setting, the propagation problem from \textsc{fds} to \textsc{cfd}s is in \textsc{ptime} for \textsc{sp} and \textsc{pc} views, and is \textsc{coNP}-complete for \textsc{sp} and \textsc{spcu} views.

### 3.2 Propagation from \textsc{cfd}s to \textsc{cfd}s

Upgrading source dependencies from \textsc{cfd}s to \textsc{cfd}s 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 \textsc{fd} propagation.

**Theorem 3.5:** In the absence of finite-domain attributes, the dependency propagation problem from \textsc{cfd}s to \textsc{cfd}s is

- in \textsc{ptime} for \textsc{spcu} views, and
- undecidable for \textsc{ra} views.

\textbf{Proof Sketch:} (a) The \textsc{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 \textsc{cfd}s instead of \textsc{fds}. (b) The undecidability follows from Theorem 3.1, since \textsc{fds} are a special case of \textsc{cfd}s.

This tells us that propagation from \textsc{cfd}s to \textsc{cfd}s is also in \textsc{ptime} for \textsc{sp}, \textsc{sp}, \textsc{sc} and \textsc{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 \textsc{cfd}s to \textsc{cfd}s is

- \textsc{coNP}-complete for views expressed as \textsc{s}, \textsc{p} or \textsc{c} queries;
- \textsc{coNP}-complete for \textsc{spcu} views; and
- undecidable for \textsc{ra} views.

\textbf{Proof Sketch:} (a) The implication problem for \textsc{cfd}s is a special case of the dependency propagation problem, when views are the identity mapping, which are expressible as \textsc{s}, \textsc{p} or \textsc{c} queries. It is known that in the presence of finite-domain attributes, \textsc{cfd} implication is already \textsc{coNP}-complete [7]. From this it follows that the propagation problem is already \textsc{coNP}-hard for views expressed as \textsc{s}, \textsc{p} or \textsc{c} queries. (b) The \textsc{np} upper bound is verified along the same lines as Theorem 3.3. (c) The undecidability follows from Theorem 3.3, since \textsc{fds} are a special case of \textsc{cfd}s.

From Corollary 3.6 it follows that the propagation problem is also \textsc{coNP}-complete for \textsc{sc}, \textsc{pc}, \textsc{sp} and \textsc{spcu} views.

We summarize in Table 1 complexity bounds on propagation from \textsc{fds} to \textsc{cfd}s, and from \textsc{cfd}s to \textsc{cfd}s. To give a complete picture, we present the complexity bounds on propagation from \textsc{fds} to \textsc{cfd}s in Table 2, including results from [15, 16, 1].

### 3.3 Interaction between \textsc{cfd}s and Views

An interesting aspect related to dependency propagation is the interaction between source \textsc{cfd}s and views.

**Example 3.1:** Consider a \textsc{cfd} $\phi = R(A \rightarrow B, (x \parallel b_1))$ 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 $\phi$, $V(D)$ is always empty. Indeed, the \textsc{cfd} $\phi$ 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 \textsc{cfd}s are “propagated” to the view, since the view satisfies any \textsc{cfd}s.

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

It turns out that this problem is nontrivial.

**Theorem 3.7:** In the general setting, the emptiness problem is \textsc{coNP}-complete for \textsc{spcu} views.

\textbf{Proof Sketch:} We consider the non-emptiness problem, the complement of the emptiness problem. We show that the non-emptiness problem is \textsc{np}-hard by reduction from the non-tautology problem, a known \textsc{np}-complete problem (cf. [16]). Its upper bound is verified by providing an \textsc{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 \textsc{np}-hard for deciding whether there exists a nonempty database that satisfies a given set of \textsc{cfd}s, in the general setting [7]. This, known as the consistency problem [7], 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 \textsc{np} upper bound for the consistency problem for \textsc{cfd}s remains intact.

In the absence of finite-domain attributes, the implication problem for \textsc{cfd}s becomes tractable [7]. It is also the case for the emptiness problem for \textsc{spcu} views and \textsc{cfd}s: a \textsc{ptime} algorithm can be developed for the emptiness test.

**Theorem 3.8:** Without finite-domain attributes, the emptiness problem is in \textsc{ptime} for \textsc{cfd}s and \textsc{spcu} views.

\textbf{Proof Sketch:} In the absence of finite-domain attributes, one can readily turn the \textsc{np} algorithm given in the proof of Theorem 3.7 into a \textsc{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 \( \phi \) is propagated from a given set \( \Sigma \) of source CFDs via an SPC view \( V \): one can simply compute a minimal cover \( \Gamma \) of all CFDs propagated from \( \Sigma \) via \( V \), and then check whether \( \Gamma \) implies \( \phi \), where CFD implication is already studied in [7].

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 CFDs 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.

We say that a set \( \Sigma \) of CFDs defined on a schema \( R \) implies another CFD \( \varphi \) on \( R \), denoted by \( \Sigma \models \varphi \), if for any instance \( D \) of \( R \), if \( D \models \Sigma \) then \( D \models \varphi \).

A cover of a set \( \Sigma \) of CFDs is a subset \( \Sigma' \) of \( \Sigma \) such that for each CFD \( \varphi \in \Sigma \), \( \Sigma' \models \varphi \). In other words, \( \Sigma' \) is contained in, and is equivalent to, \( \Sigma \). 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 \( \Sigma_{mc} \) of \( \Sigma \) is a cover of \( \Sigma \) such that

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

That is, there is neither redundant attributes in each CFD nor redundant CFDs in \( \Sigma_{mc} \).

We only include nontrivial CFDs in \( \Sigma_{mc} \). A CFD \( R(X \rightarrow A, t_p) \) is nontrivial if either (a) \( A \not\in X \), or (b) \( X = AZ \) but \( t_p \) is not of the form \( (\eta_1, \frac{t_p[Z]}{t_p[A]}) \), where either \( \eta_1 = \eta_2 \) or \( \eta_1 \) is a constant and \( \eta_2 = \ast \).

It is known that without finite-domain attributes, implication of CFDs can be decided in quadratic time [7]. Further, there is an algorithm [7], referred to as MinCover, which computes \( \Sigma_{mc} \) in \( O(|\Sigma|^3) \) time for any given set \( \Sigma \) of CFDs.

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

The propagation cover problem is to compute, given \( V \) and \( \Sigma \), a cover \( \Gamma \) of \( \text{CFD}_\Sigma(V) \). We refer to \( \Gamma \) as a propagation cover of \( \Sigma \) via \( V \), and as a minimal propagation cover if \( \Gamma \) is a minimal cover of \( \text{CFD}_\Sigma(V) \).

**Challenges.** The example below, taken from [8], shows that the problem is already hard for FDs and simple P views.

**Example 4.1:** Consider a schema \( R \) with attributes \( A_1, A_2, C_1 \) and \( D \), and a set \( \Sigma \) of CFDs on \( R \) consisting of \( A_1 \rightarrow C_1, B_1 \rightarrow C_1, \) and \( C_1, \ldots, C_n \rightarrow D \), for each \( i \in [1, n] \). Consider a view that projects an \( R \) relation onto their \( A_1, B_1 \) and \( D \) attributes, dropping \( C_1 \)’s. Then any cover \( \Sigma_c \) of the set of view CFDs propagated necessarily contains all CFDs of the form \( \eta_1, \ldots, \eta_n \rightarrow D \), where \( \eta_i \) is either \( A_1 \) or \( B_1 \), for \( i \in [1, n] \). Obviously \( \Sigma_c \) contains at least \( 2^n \) CFDs, whereas the size of the input, namely, \( \Sigma \) and the view, is \( O(n) \). Indeed, to derive view CFDs from \( C_1, \ldots, C_n \rightarrow D \), one can substitute either \( A_1 \) or \( B_1 \) for each \( C_i \), leading to the exponential blowup.

In contrast, the dependency propagation problem is in \( \text{PTIME} \) 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, 22, 25], 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 [22, 25] 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 \( R(A \rightarrow B, t_p) \) when \( t_p[A] \) ranges over values from an infinite domain \( A \).

(b) While \( AX \rightarrow A \) is a trivial FD and can be ignored, \( \phi = R(A \rightarrow X, t_p) \) may not be. Indeed, when \( t_p = (\ast, \vec{X}) \) \( a \), \( \phi \) is meaningful: it asserts that for all tuples \( t \) such that \( t[X] = \vec{X} \), the \( A \) column has the same constant \( a \).

(c) While \( X \rightarrow Y \) and \( Y \rightarrow Z \) yield \( X \rightarrow Z \) for FDs, the transitivity rule for CFDs has to take pattern tuples into account and is more involved than its FD counterpart [7].

(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 often 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 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 $R$ and a set $\Sigma$ of source constraints on $R$, PropCFD$_{SPC}$ computes a minimal propagation cover $\Pi_2$ of $\Sigma$ 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 $R \equiv \{S_1, \ldots, S_m\}$ be the source schema. Recall from Section 2 that $V$ defined on $R$ is of the form:

$$\pi_Y(R_x \times E_x), \ E_x = \sigma_F(E_x), \ E_x = R_1 \times \ldots \times R_n,$$

where $R_i$ is a constant relation, $R_i$’s are renamed relation atoms $\rho_j(S)$ for $S \in R$, $Y$ is the set of projection attributes, and $F$ is a conjunction of equality atoms.

Basic results. The constant relation $R_i$ introduces no difficulties: for each $(A_1; a_1) \in R_i$, a CFD $\rho_j(A \rightarrow \lambda, [\lambda \mid a])$ is included in $F$, where $R_i$ is the view schema derived from $V$ and $R$. Thus in the sequel we assume that $V = \pi_Y(E_x)$. The reduction below allows us to focus on $E_x$ instead of $V$. The proof is straightforward, by contradiction.

**Proposition 4.1:** For any SPC view $V$ of the form above, and any set $\Sigma$ of source CFDs, $\Sigma \models \phi$ if and only if $\Sigma \models E_x \phi$ when

- $\phi = \rho_j(A \rightarrow \lambda, (x \mid x))$, denoting $A = B$;
- $\phi = \rho_j(A \rightarrow (\lambda \mid a))$, denoting $A = \lambda'$; or
- $\phi = \rho_j(X \rightarrow A, t_p)$, 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 $\pi$ in the view definition $V$.

**Cartesian product.** Observe that each $R_i$ in $E_x$, $\rho_j(S)$, where $S$ is in $R$. All source CFDs on $S$ are propagated to the view, after their attributes are renamed via $\rho_j$.

**Selection.** The condition $F$ in $\sigma_F$ brings domain constraints into play, which can be expressed as CFDs.

**Lemma 4.2:** (a) If $A = \lambda'$ is in the selection condition $F$, then $\rho_j(A \rightarrow (\lambda \mid a))$ is in $CFD_\Sigma(S, V)$. (b) If $A = B$ is in $F$, then $\rho_j(A \rightarrow B, (x \mid x))$ is in $CFD_\Sigma(S, V)$ for the special variable $x$.

That is, one can incorporate domain constraints $A = \lambda'$ 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 $\lambda'$, 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 $\rho_j(A \rightarrow B, (x \mid x))$ in Section 2).

**Lemma 4.3:** If $\rho_j(A \rightarrow B, (x \mid x))$ and $\rho_j(BX \rightarrow G, t_p)$, then $\rho_j(AX \rightarrow G, t_p')$ is in $CFD_\Sigma(S, V)$, 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 $\Sigma_t$ to denote these CFDs as well as those in $\Sigma$ expressing domain constraints. Based on $\Sigma_t$ we can decide whether $A = B$ or $A = \lambda'$ for atoms $t$ in $Y$ (i.e., $R_1$).

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

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

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

**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 Reducibility 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 ‘$\bot$’: $\eta_1 \leq \eta_2$ if either $\eta_1$ and $\eta_2$ are the same constant ‘$\lambda'$’, or $\eta_2 = \bot$.

Given CFDs $\phi_1 = R(X \rightarrow A, t_p)$ and $\phi_2 = R(AZ \rightarrow B, t_p)$, if $t_p[A] \leq t_p[A]$ and for each $C \in X \subseteq Z, t_p[C] \leq t_p[C]$, then we can derive $\phi = R(XZ \rightarrow B, s_p)$ based on CFD implication [7]. Here $s_p = (t_p[X] \oplus t_p[Z] \parallel t_p[B])$, and $t_p[X] \parallel t_p[Z]$ is defined as follows:

- for each $C \in X \setminus Z$, $s_p[C] = t_p[C]$;
- for each $C \in Z \setminus X$, $s_p[C] = t_p[C]$;
- for each $C \in X \cap Z$, $s_p[C] = \min(t_p[C], t_p[C])$, 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 $\phi$ as an $\mathbf{A}$-resolvent of $\phi_1$ and $\phi_2$.

**Example 4.2:** Consider CFDs $\phi_1 = R([A_1, A_2] \rightarrow A, t_1)$ and $\phi_2 = R([A_1, A_2, B_1] \rightarrow B, t_2)$, where $t_1 = (\langle c \mid a \rangle)$ and $t_2 = (\langle c, b \parallel \bot \rangle)$. Then $\phi = R([A_1, A_2, B_1] \rightarrow B, t_3)$ is an $\mathbf{A}$-resolvent of $\phi_1$ and $\phi_2$, where $t_3 = (\langle c, b \parallel \bot \rangle)$.

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

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

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 $\Sigma$ of CFDs on $R$, (a) for each $A \in (U \setminus Y)$, $\mathbf{Drop}(\Sigma, A) = F^+[U \setminus \{A\}]$; (b) $\mathbf{RBR}(\Sigma, U \setminus Y)$ is a propagation cover of $\Sigma$ via $\pi_Y(R)$, where $U$ is the set of attributes in $R$. \(\square\)
Input: Source CFDs Σ, and SPC view V = πY(E1), Eα = σ_F(Eα). Output: A minimal propagation cover of Σ via V.

1. ΣV := Φ; Σ := MinCover(Σ);
2. EQ := ComputeEQ(Ε1, Σ); /* handling σ_F */
3. if EQ = Φ /* inconsistent */
   then return \{R_1(A → A, \{a\}), R_2(A → A, \{b\})\}; /* for some A ∈ Y, distinct a, b ∈ dom(A) */
4. for each \( \alpha = R_{\alpha}(A, \text{dom}(A)) \) in EQ do
   5. \( \Sigma_v := \text{MinCover(Σ),} \) /* handling product \( \times' \) */
6. for each eq ∈ EQ do /* applying domain constraints */
   7. pick an attribute rep(eq) in eq such that rep(eq) ∈ Y if eq \( \cap Y \) is not empty;
   8. substitute rep(eq) for each \( A \in \text{eq} \), in \( \Sigma_v \);
   9. \( \Sigma_v := \text{MinCover(Σ_v, } \text{eq} \) /* keep only those attributes in Y */
10. \( \Sigma_Y := \text{RBR(Σ_v, attr(E)}_v) \); /* handling π_Y */
11. \( \Sigma_L := \text{E}_{\Sigma_Y} \); /* putting domain constraints as CFDs */
12. \( \Sigma_d := \text{EQ2CFD(Σ_L)}; /* put domain constraints as CFDs */
13. return MinCover(Σ_v, \( \Sigma_d \));

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 \( \Sigma \) 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 \( \Sigma \) and \( \pi_Y(R) \), and returns RBR(\( \Sigma, U \times Y \)) as the output.

We shall also need the following lemma.

Lemma 4.5: If for any source instance \( D \) where \( D = \Sigma \), \( V(D) \) is empty, then \( RCV(A → A, \{a\}) \) and \( RCV(A → A, \{b\}) \) are in CFDs(\( \Sigma, V \)), for any attribute \( A \in \pi_Y \) and any distinct values \( a, b \in \text{dom}(A) \).

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 \( \sigma_{\Sigma} \) (line 1), 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 \( \Sigma_V \) of CFDs via renaming as described above (lines 5-6). It applies the domain constraints of EQ to \( \Sigma_V \) (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 \( \Sigma_V \). 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 \( \pi_Y \), 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 [7].

This yields a minimal propagation cover of \( V \) via \( \Sigma \), (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(\( \Sigma, A \)) (lines 4-11). Only nontrivial CFDs are included (line 8). By dropping all attributes in \( U - Y \), it obtains RBR(\( \Sigma, U \times Y \)), a cover of \( \Sigma \) and \( \pi_Y \) by Proposition 4.4.

Procedure EQ2CFD of Fig. 4 converts domain constraints 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 \( R_1(B_1', B_2'), \ R_2(A_1, A_2, A), \ R_3(A_1', A_2', B_1, B) \), and view \( V = \pi_Y(σ_{π_{\Sigma}}(R_1 \times R_2 \times R_3)) \), where \( Y = \{B_1, B_2, B_1', A_1, A_2, B_1', B_2, A_1, A_2, B\} \), and \( F = \langle B_1 = B_1' \) and \( A = A' \rangle \) and \( \psi \) is \( R_2((A_1', A_2', B_1, B_2), \) for \( e_1, e_2 \) given in Example 4.2.

Applying algorithm PropCFD_SPc to \( \Sigma \) and \( V \), after step 10, EQ consists of \( \{\{B_1, B_2\}, \{B_1\}, \{A_1, A_2\}, \{B\}\} \), and \( \Sigma_V \) consists of \( \phi_1, \phi_2 \) of Example 4.2. As also given there, procedure RBR returns \( \phi \) of Example 4.2. Procedure EQ2CFD returns \( \phi' = R(B_1 → B_1', (x \parallel x)), \) where \( R \) is the view schema with attributes in \( Y \). Then the cover returned by the algorithm consists of \( \phi \) and \( \phi' \).

Analysis. For the correctness of the algorithm, we show that for each \( \phi \) in CFDs(\( \Sigma, V \)), \( \Gamma = \phi \), and vice versa, where \( \Gamma \) is the output of the algorithm. For both directions the proof leverages the lemmas and propositions given above.

For the complexity, let \( V = \pi_Y(σ_{π_{\Sigma}}(E)) \). Then \( |Y| \leq |E| \) and \( |F| \leq (|E_1|^2 + |E_2|) \). We have the following. (a) Procedure ComputeEQ takes \( O(|E_1|^4 + |\Sigma|) \) time. (b) Procedure EQ2CFD is in \( O(|Y|^3) \) time. (c) Procedure RBR has the same complexity as its counterpart in [12]: \( O(|E_1|^2 + a^2) \), where \( a \) is an upper bound for the cardinality of \( \Gamma \) during the execution of RBR [12]. (d) The rest of the computation takes at most \( O(|E|^2 + a^2 + |E_1|^2) \). Since \( a \) is no less than \( |E_1| \), RBR takes at least \( O(|E|^2 + |E_1|^2) \). 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 \( \Sigma \) and \( V \) are defined at the schema level (it has nothing to do with the instances of \( \Sigma \)).
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, \( \Sigma \) is “simplified” by invoking MinCover(\( \Sigma \)) (line 1 of Fig. 2), removing redundant source CFDs. Second (not shown), in procedure RBR, to reduce the size of intermediate \( \Gamma \) during the computation, one can change line 11 of Fig. 3 to “\( \Gamma := \text{MinCover}(\Gamma \cup C) \)”. In our implementation, we partition \( \Gamma \) into \( \Gamma_1, \ldots , \Gamma_k \), each of a fixed size \( k_0 \), and invoke MinCover(\( \Gamma_i \)). This removes redundant CFDs from \( \Gamma \), to an extent, without increasing the worst-case complexity since it takes \( O(|\Gamma| + k_0^3) \) 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 \( R \) consisting of at least 10 relations, each with 10 to 20 attributes.

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

(b) SPC view generator. Given a source schema \( R \) and three numbers \( |Y|, |F|, |E_c| \), the view generator randomly produces an SPC view \( \gamma(Y(\sigma_F(E_c))) \) defined on \( R \) 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 \( A = \ast \), 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 \( \Sigma \) of source CFDs. More specifically, we considered \( \Sigma \) with \( |\Sigma| \) ranging from 200 to 2000, w.r.t. \( \text{var}\% = 40\% \) and \( \text{var}\% = 50\% \), while the number of attributes in each CFD ranged from 3 to 9 (\( \text{LHS} = 9 \)). 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 (\( \text{var}\%, \text{LHS} \)): the results for various (\( \text{var}\%, \text{LHS} \)) are quite consistent. As we will see, however, when \( Y \) also varies, the algorithm is sensitive to \( Y \) and (\( \text{var}\%, \text{LHS} \)) 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 of 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 \( \gamma(Y(\sigma_F(E_c))) \): the set \( Y \) of projection attributes, the selection condition \( F \), and the Cartesian product \( E_c \). In each of the experiments, we considered sets \( \Sigma \) of source CFDs with \( |\Sigma| = 2000 \), w.r.t. \( \text{var}\% = 40\% \) and \( \text{var}\% = 50\% \), 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 \( |F| = 10, |E_c| = 4 \), 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 \(|U \setminus Y|\), where \( U \) is the total number of attributes in \( E_c \) (which is determined by \( |E_c| \)) and the attibutes of the relations in the source schema \( R \). Note that the outer loop of procedure RBR is dominated by \( |U \setminus Y| \); the larger \(|U \setminus Y|\) 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 \(|Y| \geq 30\), different settings of \( \text{var}\% \) 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-
stems (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| = 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 F 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 on the performance of PropCFD_SPC. Fixing |F| = 10 and |Y| = 25, we varied |E| from 2 to 11. The results are reported in Fig. 8.

Figure 8(a) shows that when |E| gets larger, the algorithm takes less time. This is because when Y, F and Σ are fixed, increasing |E| leads to more CFDs to be dropped from the minimal propagation cover, for involving attributes not in Y. Further, when |E| 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: V1 = πAB(σC>D(R1(ABCD))) and V2 = πAE(σC=H(R1(ABCD) × R2(ECHL))), while the set Σ consists of source CFDs R1(A → B, (C, D)) and R2(E → L, (C, H)). Let V1, V2 also denote the view schema of V1, V2, respectively. Then a minimal cover of the CFDs propagated via V1 consists of V1(A → B, (C, D)) whereas no nontrivial CFDs are propagated to the view via V2.

Figure 8(a) also shows that when |E| ≥ 6, the algorithm is insensitive to E, because most of the sources CFDs are dropped, i.e., not propagated to the view (as |Σ| is fixed).

For the same reason, when |E| 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| 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 |Σ| = 2000, |Y| = 50, |F| = 10 and |E| = 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 in the input SPC views. In contrast, the cost increases rapidly when the set of 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 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 SPCV 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 extend [15, 16, 1] by providing complexity bounds for FD propagation in the general setting, and for CFD propagation.
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, 22, 25]. 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 [5, 13, 21]. Propagation from XML keys to relational FDs is studied in [5], 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., [6, 11, 14]. The focus is on the closure problem for dependencies under views. It is to decide, given a set \( \Sigma \) of dependencies and a view \( V \), whether the set \( \Gamma \) of dependencies propagated from \( \Sigma \) via \( \Gamma \) characterizes the views, i.e., whether the set of databases that satisfy \( \Gamma \) is precisely the set of views \( \Gamma(D) \) 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 al-

![Figure 6: Varying the number of projection attributes in Y in SPC views](image1)

![Figure 7: Varying the selection condition F in SPC views](image2)

![Figure 8: Varying the number of relations in the Cartesian product Ec in SPC views](image3)
A variety of extensions of classical dependencies have been proposed, for specifying constraint databases [2, 3, 19, 20]. Constraints of [3, 19] cannot express CFDs. More expressive are constraint-generating dependencies (CGDs) of [2] and constrained tuple-generating dependencies (CTGDs) of [20], both subsuming CFDs. A CGD is of the form $\forall x(R_1(x) \land \ldots \land R_k(x) \land \xi(x) \rightarrow \eta(x))$, where $R_i$’s are relation atoms, and $\xi, \eta$ are arbitrary constraints that may carry constants. A CTGD is of the form $\forall x(R_1(x) \land \ldots \land R_k(x) \land \xi \rightarrow \exists y(R'_1(x, y) \land \ldots \land R'_k(x, y) \land \xi'(x, y)))$, 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 [7]. 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 [24, 23]. 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 SPV 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 [4]. 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