Extending Inclusion Dependencies with Conditions

Shuai Ma[a], Wenfei Fan[a,b], Loreto Bravo[c]

[a] *SKLSDE Lab, Beihang University, China*
[b] *University of Edinburgh, UK*
[c] *University of Concepcion, Chile*

Abstract

This paper introduces a class of conditional inclusion dependencies (CINDs), which extends inclusion dependencies (INDs) by enforcing patterns of semantically related data values. We show that CINDs are useful not only in data cleaning, but also in contextual schema matching. We give a full treatment of the static analysis of CINDs, and show that CINDs retain most desired properties of traditional INDs: (a) CINDs are always satisfiable; (b) CINDs are finitely axiomatizable, *i.e.,* there exists a sound and complete inference system for the implication analysis of CINDs; and (c) the implication problem for CINDs has the same complexity as its traditional counterpart, namely, PSPACE-complete, in the absence of attributes with a finite domain; but it is EXPTIME-complete in the general setting. In addition, we investigate the interaction between CINDs and conditional functional dependencies (CFDs), as well as two practical fragments of CINDs, namely *acyclic* CINDs and *unary* CINDs. We show the following: (d) the satisfiability problem for the combination of CINDs and CFDs becomes undecidable, even in the absence of finite-domain attributes. (e) in the absence of finite-domain attributes, the implication problem for acyclic CINDs and for unary CINDs retains the same complexity as its traditional counterpart, namely, NP-complete and PTIME, respectively; but in the general setting, it becomes PSPACE-complete and coNP-complete, respectively; and (f) the implication problem for *acyclic unary* CINDs remains in PTIME in the absence of finite-domain attributes and coNP-complete in the general setting.

Keywords: Satisfiability; Implication; Inclusion dependencies; Data cleaning; Schema matching

1. Introduction

A class of *conditional functional dependencies* (CFDs) has recently been proposed in [1] as an extension of *functional dependencies* (FDs). In contrast to traditional FDs, CFDs hold conditionally on a relation, *i.e.,* they apply only to those tuples that satisfy certain data-value patterns, rather than to the entire relation. CFDs have proven useful in data cleaning [1, 2, 3, 4, 5, 6, 7]: inconsistencies and errors in the data may be captured as violations of CFDs, whereas they may not be detected by traditional FDs.

It has been recognized (*e.g.,* [8, 9]) that to clean relational data, one should make use of not only FDs, but also *inclusion dependencies* (INDs). Furthermore, INDs are commonly used in schema matching systems, *e.g.,* Clio [10]: INDs associate attributes in a source schema with semantically related attributes in a target schema. Nevertheless,

schema matching and data cleaning often need to use inclusion dependencies that hold on a part of data that satisfies certain constant patterns, rather than on the entire relations. As illustrated by the examples below, dependencies with constant patterns cannot be expressed as traditional INDs. These suggest that we extend INDs by incorporating patterns of semantically-related data values, along the same lines as CFDs.

Example 1.1: Consider the two relational schemas, referred to as source and target:

source: order(asin : string, title : string, type : string, price : real)
target: book(isbn : string, title : string, price : real, format : string),
CD(id : string, album : string, price : real, genre : string)

The source database contains a single relation order, specifying items of various types such as books, CDs and DVDs, ordered by customers. The target database has two relations, namely, book and CD, specifying items of books and CDs ordered by customers, respectively. In each relation the underlined attributes indicate a key (a special case of FDs), *e.g.,* asin is a key for order. Example source and target instances D_1 are shown in Fig. 1. Note that keys (FDs) are not the focus of this work, but to make the schemas more rational.

To find schema mappings from the source to the target, or to detect errors across these databases, one might be tempted to use standard INDs such as:

ind_1: order (title, price) $\subseteq$ book (title, price)
ind_2: order (title, price) $\subseteq$ CD (album, price)

These INDs, however, do not make sense: one cannot expect the title and price of each book item in the order table to find a matching CD tuple (*e.g.,* Harry Potter); similarly for the CD items in the order table. Nevertheless, there are indeed inclusion dependencies from the source to the target, as well as on the target database, but only under certain conditions:

	asin	title	type	price
t_1:	a23	Snow White	CD	7.99
t_2:	a12	Harry Potter	book	17.99

(a) Example order data

	isbn	title	price	format
t_3:	b32	Harry Potter	17.99	hardcover
t_4:	b65	Snow White	7.99	paperback

(b) Example book data

	id	album	price	genre
t_5:	b65	Snow White	7.99	a-book

(c) Example CD data

Figure 1: Example instances D_1 of source and target

cind_1: order (title, price, type = 'book') $\subseteq$ book (title, price)
cind_2: order (title, price, type = 'CD') $\subseteq$ CD (album, price)
cind_3: CD (album, price, genre = 'a-book') $\subseteq$ book (title, price, format = 'audio')

Here constraint cind_1 states that for each order tuple t, if its type is 'book', then there must exist a book tuple t' such that tuples t and t' agree on their title and price attributes; similarly for constraint cind_2. Here type = 'book' specifies a condition under which the constraint can be applied. Constraint cind_3 asserts that for each CD tuple t, if its genre is 'a-book' (audio book), then there must be a book tuple t' such that the title and price of t' are identical to the album and price of t, and moreover, the format of t' must be 'audio'. Here format = 'audio' is an additional condition on matched tuples.

Constraints cind_1 and cind_2 specify a form of contextual schema matching studied in [11]. As shown in [11], contextual schema matching often allows us to derive sensible schema mapping from a source to a target, which cannot be found via schema matching specified with traditional INDs.

Such constraints also allow us to detect errors across different relations. For instance,

while D_1 of Fig. 1 satisfies cind_1 and cind_2, it violates cind_3. Indeed, tuple t_5 in the CD table has an 'a-book' genre, but it cannot find a match in the book table with 'audio' format. The violation suggests that there may exist inconsistencies in the CD and book tables in the target database. Such inconsistencies cannot be detected by traditional INDs. Note that the book tuple t_4 is not a match for t_5: although t_5 and t_4 agree on their album (title) and price attributes, the format of t_4 is 'paperback' rather than 'audio' as required by cind_3. □

Like CFDs, dependencies $\mathsf{cind}_1 - \mathsf{cind}_3$ are required to hold only on a subset of tuples satisfying certain patterns. In other words, they apply only *conditionally* to relations order, CD, and book. These dependencies are specified with constants, and hence, cannot be expressed as standard INDs. Although such dependencies are needed for schema matching and data cleaning, to the best of our knowledge, the earlier conference version [12] of this paper is the first to study these constraints.

Contributions. To this end we introduce an extension of INDs, and investigate the static analyses of these constraints.

(1) Our first contribution is a notion of *conditional inclusion dependencies* (CINDs). A CIND is defined as a pair consisting of an IND $R_1[X] \subseteq R_2[Y]$ and a *pattern tableau*, where the tableau enforces binding of semantically related data values across relations R_1 and R_2. For example, ind_1, ind_2, and cind_1–cind_3 given above can all be expressed as CINDs. In particular, traditional INDs are a *special case* of CINDs. This mild extension of INDs captures a fundamental part of the semantics of the data, and suffices to express rules commonly used in data cleaning and schema matching.

(2) Our second contribution consists of complexity results for fundamental problems associated with CINDs, as well as an inference system for reasoning about CINDs.

Given a set of CINDs, the first question one would ask is whether the CINDs are *satisfiable*, *i.e.,* whether they are "dirty" themselves. Indeed, one does not want to enforce the CINDs on a database at running time but find, after repeated failures, that the CINDs cannot possibly be satisfied by a *nonempty* database. Similarly, one does not want to match schema based on CINDs that do not make sense. The satisfiability analysis helps users identify satisfiable sets of CINDs for data cleaning and schema matching.

Another important question concerns the *implication* analysis, which is to decide whether a set of CINDs entails another CIND. The implication analysis is useful in reducing redundant CINDs, and hence improving performance when detecting CIND violations in a database, and speeding up the derivation of schema mappings from CINDs [10].

For traditional INDs, the satisfiability analysis is not an issue: any set of INDs is satisfiable. Their implication analysis is PSPACE-complete, and furthermore, it is *finitely axiomatizable*: there exists a finite, sound and complete set of axioms (see, *e.g.,* [13]).

We show that although CINDs are more expressive than INDs, they retain most desired properties of their IND counterpart: (a) CINDs are always satisfiable; (b) the implication of CINDs is finitely axiomatizable; (c) in the absence of finite-domain attributes, the implication problem for CINDs is PSPACE-complete.

In the real world, it is common to find finite-domain attributes, *e.g.,* Boolean, date, etc. It is hence necessary to get the complexity of these problems right in the general setting when finite-domain attributes may be present. The implication problem of INDs remains intact in the general setting, as their inference does not involve any data values. However, in the gen-

eral setting, the implication problem for CINDs becomes EXPTIME-complete, due to the interaction between the data values in finite domains and constants in pattern tableaux.

(3) Our third contribution consists of complexity bounds for reasoning about the combination of CINDs and CFDs. As remarked earlier, to clean relational data, one may need both CFDs and CINDs in practice.

We show that the presence of finite-domain attributes does not complicate the satisfiability and implication analyses when INDs and FDs (resp. CINDs and CFDs) are taken together. Nonetheless, while a set of INDs and FDs is always satisfiable [13], we show that the satisfiability problem for CINDs and CFDs taken together becomes *undecidable*, even in the absence of finite-domain attributes, due to the presence of data values.

(4) Our fourth contribution consists of complexity bounds for reasoning about two fragments of CINDs: *acyclic* CINDs and *unary* CINDs, which extend *acyclic* INDs and *unary* INDs (see *e.g.,* [13] for details), respectively. Many CINDs found in practice are either acyclic or unary. For instance, the *set* $\{\mathsf{cind}_1, \mathsf{cind}_2\ \mathsf{cind}_3\}$ of CINDs that we have seen earlier is a set of acyclic CINDs.

We show that in the absence of attributes with a finite domain, the implication problem for acyclic CINDs is NP-complete, and it is in PTIME for unary CINDs, the same as their counterparts for classical acyclic and unary INDs, respectively. That is, acyclic CINDs (resp. *unary* CINDs) retain the same complexity as acyclic INDs [14] (resp. unary INDs [15]). Nevertheless, we also show that in the general setting, the implication problem becomes PSPACE-complete for acyclic CINDs, and it is coNP-complete for unary CINDs. This tells us that the increased expressive power of CINDs does not come for free. Nevertheless, these complexity bounds are still lower than their counterparts for general CINDs, namely, PSPACE and EXPTIME, respectively. Therefore, when only acyclic or unary CINDs are needed, we do not have to pay the price of the complexity of the full-fledged CINDs.

We show that further constraining acyclic (or unary) CINDs does not make our lives easier. Indeed, the implication problem for *acyclic unary* CINDs remains coNP-complete in the general setting (while in the absence of attributes with finite domains, it of course remains in PTIME).

These results settle several fundamental problems associated with CINDs, an extension of INDs that is useful in schema matching and data cleaning. (1) We show that one can specify any CINDs without worrying about their satisfiability. (2) We develop a sound and complete inference system for the implication analysis of CINDs, which provides algorithmic insight into reasoning about CINDs. (3) We present a comprehensive picture of complexity bounds on the implication analysis of CINDs, when finite-domain attributes are present or absent, for general CINDs and for practical fragments (acyclic CINDs and unary CINDs). (4) We also show that there exists interaction between CFDs and CINDs.

Remark. (1) This paper is an extension of our earlier work [12] by including (a) the study of acyclic CINDs (Section 5.1), (b) the investigation of unary CINDs (Section 5.2), (c) the complexity bounds on the implication problem for standard INDs, AINDs and UINDs in the presence of finite-domain attributes (Sections 3, 5.1 and 5.2), and (d) the proofs for the sound and complete inference system and for the complexity bounds for the satisfiability and implication analyses of CINDs (Section 3). Some of the proofs are nontrivial and the techniques are interesting in their own right.

The algorithm for the satisfiability checking of CINDs and FDs and its experimental study were reported in [12]; these are left out from this paper in order to focus on the theoretical aspects of CINDs.

(2) CINDs do not introduce a new logical formalism. Indeed, in first-order logic, they can be expressed in a form similar to tuple-generating dependencies (TGDs), which have lately generated renewed interests in data exchange (see [16] for a survey). However, (a) these simple CINDs suffice to capture data consistency and contextual schema matching commonly found in practice, without incurring the complexity of *full-fledged* TGDs (*e.g.,* the undecidability of their implication problem), and (b) no prior work has studied the satisfiability, implication and finite axiomatizability of TGDs in the presence of *constants* or *finite-domain* attributes.

Organization. The remainder of the paper is organized as follows. We define CINDs in Section 2, and investigate their satisfiability and implication problems in Section 3. The interaction between CFDs and CINDs is studied in Section 4. We study acyclic CINDs and unary CINDs in Section 5, followed by related work in Section 6 and topics for future work in Section 7. To improve the readability we defer the detailed proofs to the appendix, but include their sketches in the main paper.

2. Conditional Inclusion Dependencies

A relational database schema $\mathcal{R}$ is a finite collection of relation schemas $(R_1, \ldots, R_n)$, where for each $i \in [1, n]$, R_i is defined over a finite set of attributes, denoted as $\mathsf{attr}(R_i)$. For each attribute $A \in \mathsf{attr}(R_i)$, its domain is specified in R_i, denoted as $\mathsf{dom}(A)$, which is either finite (*e.g.,* bool) or infinite (*e.g.,* string). We use $\mathsf{finattr}(\mathcal{R})$ to denote the set of all the finite-domain attributes that appear in $\mathcal{R}$. We refer to an attribute in terms of the name of the relation and the name of the attribute, and only use the attribute name when it is clear in the context.

An instance I_i of R_i is a finite set of tuples such that for each $t \in I_i$, $t[A] \in \mathsf{dom}(A)$ for all attributes $A \in \mathsf{attr}(R_i)$. A database instance D of $\mathcal{R}$ is a collection of relation instances $(I_1, \ldots, I_n)$, where I_i is an instance of R_i for each $i \in [1, n]$.

Conditional inclusion dependencies. A conditional inclusion dependency (CIND) ψ is defined as a pair

$$(R_a[X; X_p] \subseteq R_b[Y; Y_p],\ T_p),$$

where

(1) X, X_p and Y, Y_p are lists of attributes in $\mathsf{attr}(R_a)$ and $\mathsf{attr}(R_b)$, respectively, such that X and X_p (resp. Y and Y_p) are disjoint;
(2) $R_a[X] \subseteq R_b[Y]$ is a standard IND, referred to as the IND *embedded* in ψ; and
(3) T_p is a tableau, called the *pattern tableau* of ψ, which is a set of tuples over attributes $X_p \cup Y_p$ such that for each tuple $t_p \in T_p$ and each attribute A in $X_p \cup Y_p$, $t_p[A]$ is a constant drawn from $\mathsf{dom}(A)$.

We require that X and X_p (resp. Y and Y_p) are disjoint to make CINDs succinct, and to help eliminate useless constraints. One can readily verify that when X and X_p (resp. Y and Y_p) are not disjoint, we can easily define an equivalent CIND that satisfies the disjointness condition.

We adopt the following conventions and notations. (1) Let $X = [A_1, \ldots, A_m]$ and $Y = [B_1, \ldots, B_m]$. We require that $\mathsf{dom}(A_i) \subseteq \mathsf{dom}(B_i)$ for each $i \in [1, m]$. (2) If a list Z of attributes occurs in both X_p and Y_p, we use Z_L and Z_R to indicate the occurrence of Z in X_p and Y_p, respectively. (3) When both X_p and Y_p are empty lists, T_p is an empty set $\emptyset$. (4) We use $X \cup Y$ to denote the set of all attributes of X and Y, and $X \setminus Y$ to denote the list obtained from list X by removing all the elements in list Y. We denote $X \cup X_p$ as $\mathsf{LHS}(\psi)$ and $Y \cup Y_p$ as $\mathsf{RHS}(\psi)$,

ψ_1: (order[title, price; nil] ⊆ book[title, price; nil], T_1)
ψ_2: (order[title, price; nil] ⊆ CD[album, price; nil], T_2)
ψ_3: (order[title, price; type] ⊆ book[title, price; nil], T_3)
ψ_4: (order[title, price; type] ⊆ CD[album, price; nil], T_4)
ψ_5: (CD[album, price; genre] ⊆ book[title, price; format], T_5)

$T_1 = T_2 = \emptyset$ $\quad T_3$:

type	nil
book	

T_4:

type	nil
CD	

T_5:

genre	format
a-book	audio

Figure 2: Example CINDs

and separate the LHS and RHS attributes in a pattern tuple with '||'. (5) We use nil to denote an empty list. (6) In addition, we adopt the common assumption that each finite or infinite domain contains at least two elements [17], which was used when studying, *e.g.,*FDs [13].

Example 2.1: Constraints ind_1, ind_2, and cind_1–cind_3 given in Examples 1.1 can all be expressed as CINDs, as shown in Fig 2: ψ_1 and ψ_2 for ind_1 and ind_2, and ψ_3–ψ_5 for cind_1–cind_3, respectively. Observe that ind_1 and ind_2 are standard INDs embedded in ψ_1 and ψ_2, respectively. In ψ_5, X is [album, price], Y is [title, price], X_p is [genre], and Y_p is [format]. The standard IND embedded in ψ_5 is CD[album, price] ⊆ book[title, price]. □

Consider CIND $\psi = (R_a[X; X_p] \subseteq R_b[Y; Y_p], T_p)$. In general, the embedded IND may not hold on the entire R_a relation: it applies only to R_a tuples matching certain pattern tuples in T_p. We say that an R_a (resp. R_b) tuple t_1 (resp. t_2) *matches* a pattern tuple $t_p \in T_p$ if $t_1[X_p] = t_p[X_p]$ (resp. $t_2[Y_p] = t_p[Y_p]$).

An instance (I_a, I_b) of (R_a, R_b) *satisfies* the CIND ψ, denoted by $(I_a, I_b) \models \psi$, if and only if for *each* tuple t_1 in the relation I_a, and for *each* pattern tuple t_p in the pattern tableau T_p, if $t_1[X_p] = t_p[X_p]$, then *there exists* a tuple t_2 in the relation I_b such that $t_1[X] = t_2[Y]$, and moreover, $t_2[Y_p] = t_p[Y_p]$.

That is, if $t_1[X_p]$ matches the pattern $t_p[X_p]$, then the standard IND embedded in ψ and the pattern specified by t_p must be satisfied. More specifically, there exists a tuple t_2 such that (1) $t_2[Y] = t_1[X]$, and (2) $t_2[Y_p]$ must match the pattern $t_p[Y_p]$. Note that $t_1[X_p] = t_p[X_p]$ if X_p = nil, and $t_2[Y_p] = t_p[Y_p]$ if Y_p = nil. Intuitively, X_p is used to identify the R_a tuples over which ψ is applied. The pattern on Y_p enforces the matching R_b tuples to have certain values in their Y_p attributes.

We say that a database D satisfies a set Σ of CINDs, denoted by $D \models \Sigma$, if $D \models \psi$ for each $\psi \in \Sigma$. Two sets Σ_1 and Σ_2 of CINDs are said to be *equivalent*, denoted by $\Sigma_1 \equiv \Sigma_2$, if for any instance D, $D \models \Sigma_1$ iff $D \models \Sigma_2$.

Example 2.2: Database D_1 given in Fig. 1 satisfies CINDs ψ_3 and ψ_4. However, the INDs embedded in these CINDs do not necessarily hold. For example, while ψ_4 is satisfied, ind_2 in ψ_4 is not, since item Harry Potter in the order table cannot find a match in the CD table. The pattern X_p in LHS(ψ_4) identifies the order tuples on which ψ_4 has to be enforced, *i.e.,* those CD tuples; similarly for ψ_3.

On the other hand, ψ_5 is *violated* by the database. Indeed, for CD tuple t_5, there exists a pattern tuple t_p in T_5 such that t_5[genre] = t_p[genre] = 'a-book' but there exists no tuple t in table book such that t[format] = 'audio', t[title] = t_5[title] = 'Snow White', and t[price] = t_5[price] = 7.99. Here the genre pattern is to identify CD tuples on which ψ_5 is applicable, while format is a *constraint* on the book tuples that match those CD tuples via the IND embedded in ψ_5. □

Normal form. A CIND $\psi = (R_a[X; X_p] \subseteq R_b[Y; Y_p], T_p)$ is in the *normal form* if T_p only consists of a single pattern tuple t_p. We write ψ as $(R_a[X; X_p] \subseteq R_b[Y; Y_p],\ t_p)$.

It is straightforward to verify that a CIND $\psi = (R_a[X; X_p] \subseteq R_b[Y; Y_p], T_p)$ can be expressed as a set $\Sigma_\psi = \{(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_p) \mid t_p \in$

$T_p\}$ of CINDs in the normal form, which is *equivalent* to $\{\psi\}$, *i.e.,* $\{\psi\} \equiv \Sigma_\psi$. In light of this, in the sequel we shall *w.l.o.g.* consider CINDs in the normal form only.

3. Reasoning about Conditional Inclusion Dependencies

For any constraint language L, there are two fundamental problems associated with it. One is the satisfiability problem, which is to determine whether a given set of constraints in L has conflicts. The other is the implication problem, to derive other constraints from a given set of constraints in L. As remarked in Section 1, to effectively use a constraint language in practice, it is often necessary to answer these two questions at compile time.

One might be tempted to use a constraint language that is more powerful than CINDs, *e.g.,* full-fledged TGDs extended by allowing constants (data values). The question is whether the language allows us to effectively reason about its constraints. We need a constraint language that is powerful enough to express dependencies commonly found in schema matching and data cleaning, while at the same time well-behaved enough so that its associated decision problems are tractable or, at the very least, decidable [16]. For full-fledged TGDs, it was known 30 years ago that the implication problem is *undecidable* even in the *absence* of data values [18].

As found in most database textbooks, standard INDs have several desired properties. (a) INDs are always satisfiable. (b) For INDs, the implication problem is decidable (PSPACE-complete). (c) Better still, INDs are finitely axiomatizable, *i.e.,* there exists a finite set of axioms that is sound and complete for implication of CINDs. The question is: do CINDs still have these properties (which extend INDs by incorporating data values)?

As observed in [18], if TGDs were extended by including data values, their analysis would become more intriguing. Although we are not aware of previous work on the complexity of the static analyses of TGDs with constants, the study of CFDs [1] tells us that data values in the pattern tableaux of dependencies would make our lives much harder: (1) as opposed to standard FDs for which the implication problem is in linear-time, the implication analysis for CFDs is coNP-complete, and (2) while a set of FDs is always satisfiable, the satisfiability problem for CFDs is NP-complete. Moreover, for the satisfiability and implication problems, we have to consider the impact of finite-domain attributes, as a finite domain imposes an additional constraint on how a relation can be populated such that the relation observes the constant patterns and satisfies the dependencies. The interaction between data values in finite domains and constants in pattern tableaux makes the inference complicated.

In this section we study the satisfiability and implication problems for CINDs. We show that despite that CINDs contain constants and are more expressive than INDs, they retain most of the desired properties of INDs. That is, CINDs strike a balance between the expressive power and complexity. Below we first settle the satisfiability problem for CINDs in positive, in Section 3.1. We then study the implication analysis of CINDs in Section 3.2. More specifically, we develop a sound and complete inference system for CINDs in Section 3.2.1, and then establish the complexity of the implication problem in Section 3.2.2. Moreover, we also revisit the implication problem for standard INDs in the presence of finite-domain attributes, an issue that has not been studied.

3.1. The Satisfiability Analysis

One cannot expect to derive sensible schema matches or to effectively clean data

from a set of constraints if the constraints are not satisfiable themselves. Thus before any run-time computation is conducted, we have to make sure that the constraints are satisfiable, or in other words, make sense themselves.

The *satisfiability problem* for a constraint language L is to determine, given a finite set Σ of constraints in L defined on a database schema $\mathcal{R}$, whether there exists a nonempty instance D of $\mathcal{R}$ such that $D \models \Sigma$.

Traditional FDs and INDs do not contain constants, and any set of FDs or INDs is always satisfiable [19]. However, adding data values to constraints may make their satisfiability analysis much harder, *e.g.,* CFDs [1]. To illustrate this, we first review CFDs below, which will also be needed when we study their interaction with CINDs in Section 4.

CFDs. A *conditional functional dependency* (CFD) ϕ on a relation R is a pair $(R : X \rightarrow Y,\ T_p)$, where (1) X and Y are subsets of $\mathsf{attr}(R)$; (2) $R : X \rightarrow Y$ is a standard FD, referred to as the FD *embedded in* ϕ; and (3) T_p is a tableau with all attributes in X and Y, referred to as the *pattern tableau* of ϕ, where for each A in X or Y and each tuple $t_p \in T_p$, $t_p[A]$ is either a constant $a \in \mathsf{dom}(A)$, or an unnamed variable '_', as defined for CINDs given earlier.

To formalize the semantics of CFDs, we define an operator $\asymp$ on constants and the symbol '_': $\eta_1 \asymp \eta_2$ if either $\eta_1 = \eta_2$, or one of η_1 and η_2 is '_'. The operator $\asymp$ naturally extends to tuples. For example, (Mayfield, EDI) $\asymp$ (_, EDI) but (Mayfield, EDI) $\not\asymp$ (_, NYC).

An instance I of R *satisfies* the CFD ϕ, denoted by $I \models \phi$, if and only if for *each pair* of tuples t_1, t_2 in the relation I, and for *each* tuple t_p in the pattern tableau T_p, if $t_1[X] = t_2[X] \asymp t_p[X]$, then $t_1[Y] = t_2[Y] \asymp t_p[Y]$. That is, if $t_1[X]$ and $t_2[X]$ are equal to each other and match the pattern $t_p[X]$, then $t_1[Y]$ and $t_2[Y]$ must also be equal to each other and match the pattern $t_p[Y]$. Note that standard FDs are a special case of CFDs in which the pattern tableau contains a single tuple that consists of _ only.

Along the same lines as CINDs in normal form, we say that a CFD $\phi = (R : X \rightarrow Y,\ T_p)$ is in the *normal form* if T_p consists of a single tuple t_p and Y contains a single attribute A, and we write ϕ as $(R : X \rightarrow A,\ t_p)$. We can always rewrite a CFD into an equivalent set of CFDs in the normal form. In the sequel, we only consider CFDs in the normal form.

A set of CFDs, which extend FDs by adding constant patterns, may be unsatisfiable, as illustrated by the following example [1].

Example 3.1: Consider a relation schema $R(A, B)$, and the CFDs below defined on R, refining standard FDs $A \rightarrow B$ and $B \rightarrow A$:

ϕ_1: $(A \rightarrow B, (\mathsf{true} \parallel b_1))$, ϕ_2: $(A \rightarrow B, (\mathsf{false} \parallel b_2))$,
ϕ_3: $(B \rightarrow A, (b_1 \parallel \mathsf{false}))$, ϕ_4: $(B \rightarrow A, (b_2 \parallel \mathsf{true}))$,

where $\mathsf{dom}(A)$ is bool, and b_1, b_2 are two distinct constants in $\mathsf{dom}(B)$. The CFD ϕ_1 (resp. ϕ_2) asserts that for any R tuple t, if $t[A]$ is true (resp. false), then $t[B]$ must be b_1 (resp. b_2). On the other hand, ϕ_3 (resp. ϕ_4) requires that if $t[B]$ is b_1 (resp. b_2), then $t[A]$ must be false (resp. true). Observe that there exists *no* nonempty instance of R that satisfies all these CFDs. Indeed, for any R tuple t, no matter what Boolean value $t[A]$ is, these CFDs together force $t[A]$ to take the other value from the finite domain bool.

Note that if $\mathsf{dom}(A)$ and $\mathsf{dom}(B)$ were infinite, one could find a tuple t such that $t[A]$ is neither true nor false, and $t[B]$ is not b_1 or b_2; then the R instance $\{t\}$ satisfies these CFDs. This tells us that attributes with a finite domain complicate the satisfiability analysis. □

It is known [1] that the satisfiability problem for CFDs is NP-complete. As opposed to CFDs, the satisfiability analysis of CINDs is as trivial as their standard INDs counterpart, despite the increased expressive power of

CINDs. Intuitively, this is because CFDs are a subset of EGDs [13] (universally quantified formulas), while CINDs are TGDs in which the existential quantification allows us to add tuples that match the pattern and satisfy the embedded IND of a CIND. Moreover, the requirement that X and X_p (resp. Y and Y_p) are disjoint in a CIND $(R_a[X; X_p] \subseteq R_b[Y; Y_p],\ t_p)$ simplify the analysis.

Theorem 1. *Any set of CINDs is satisfiable.*

Proof: We show that given a set Σ of CINDs over a database schema $\mathcal{R} = (R_1, \ldots, R_n)$, we can always construct a *nonempty* finite instance D of $\mathcal{R}$ such that $D \models \Sigma$.

We build such an instance D as follows.

(1) We start with the construction of the active domains. For each attribute A, we first collect in $\mathsf{adom}(A)$ all those constants that appear in some pattern of A in the CINDs. We then propagate these constants from $\mathsf{adom}(A)$ to $\mathsf{adom}(B)$ for each attribute B that is connected to A via a CIND of Σ.

More specifically, for each attribute A in some relation of $\mathcal{R}$, we start with $\mathsf{adom}(A) = \emptyset$. For every CIND $(R_i[X; X_p] \subseteq R_j[Y; Y_p],\ t_p)$ in Σ, if $A \in X_p$ or $A \in Y_p$, we include in $\mathsf{adom}(A)$ the constant $t_p[A]$, *i.e.,* we let $\mathsf{adom}(A) = \mathsf{adom}(A) \cup \{t_p[A]\}$. If $\mathsf{dom}(A)$ is still empty after all the CINDs in Σ have been inspected, we let $\mathsf{adom}(A) = \{c\}$ for an arbitrary constant $c \in \mathsf{dom}(A)$.

We propagate these initial values as follows. For each CIND $(R_a[A_1, A_2, \ldots, A_m; X_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_p)$ in Σ, we expand $\mathsf{adom}(B_i)$ by letting $\mathsf{adom}(B_i) = \mathsf{adom}(B_i) \cup \mathsf{adom}(A_i)$ for each $i \in [1, m]$. The propagation process is recursively applied to all attributes, and proceeds until no further changes can be made to $\mathsf{adom}(A)$ of any attribute A. Since it starts with a finite set of values for each $\mathsf{adom}(A)$, it is easy to verify that the process always terminates.

(2) We construct D as follows. For each $R_i(A_1, \ldots, A_k) \in \mathcal{R}$, we define $I_i = \mathsf{adom}(A_1) \times \ldots \times \mathsf{adom}(A_k)$, where $\times$ is the Cartesian product operation. And we define $D = (I_1, \ldots, I_n)$.

It is easy to verify that the database instance D is nonempty, finite and that $D \models \Sigma$. □

3.2. The Implication Analysis

As remarked earlier, the implication analysis allows us to remove redundancies from data quality rules to improve performance, and it is also critical to deriving schema mappings from schema matchings [10, 11]. Recall that (1) a schema mapping is data transformation from instances of a source schema to instances of a target schema while preserving the appropriate information of the source instances; and (2) a schema matching is a pairing of attributes (or groups of attributes) of a source schema and attributes of a target schema such that the pairs are likely to be semantically related. In practice, finding a schema matching is often an early step in building a schema mapping, which is a common task in a variety of data exchange and integration scenarios [11]. It is known that contextual schema matching needs to make use of (contextual) foreign keys, a primitive and special case of CINDs.

The *implication problem* for a constraint language L is to determine, given a finite set Σ of constraints in L and another ψ of L, all defined on the same database schema $\mathcal{R}$, whether Σ entails ψ, denoted by $\Sigma \models \psi$, *i.e.,* whether for all instances D of $\mathcal{R}$, if $D \models \Sigma$ then $D \models \psi$.

Example 3.2: Consider the CINDs given in Fig. 2. Let Σ be $\{\psi_1, \psi_2, \psi_5\}$. One can verify that $\Sigma \models \psi_3$ and $\Sigma \models \psi_4$. That is, CINDs ψ_3 and ψ_4 are redundant, and we only need to focus on CINDs in Σ, and ignore ψ_3 and ψ_4. □

3.2.1. An Inference System

As remarked earlier, for standard INDs the implication problem is not only decidable but also finitely axiomatizable.

We show that CINDs are also finitely axiomatizable, by providing an inference system for CINDs, denoted by $\mathcal{I}$. Given a finite set Σ of CINDs and another CIND ψ, we denote by $\Sigma \vdash_{\mathcal{I}} \psi$ if ψ is provable from Σ using rules of $\mathcal{I}$. As will be seen shortly, these rules are both *sound*, *i.e.,* if $\Sigma \vdash_{\mathcal{I}} \psi$ then $\Sigma \models \psi$, and *complete*, *i.e.,* if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}} \psi$.

Recall that for standard INDs, the inference system consists of three rules: reflexivity, projection-permutation and transitivity [13, 20]. To cope with the richer semantics of CINDs, the inference system $\mathcal{I}$ is more complicated than the inference system for INDs.

The inference system $\mathcal{I}$ is shown in Fig. 3. We briefly illustrate the inference rules (axioms) in $\mathcal{I}$ as follows. Intuitively, rules IR1–IR3 correspond to the inference rules for INDs. IR1 is the reflexivity rule. IR2 shows that the patterns, *i.e.,* X_p and Y_p, can also be permuted, in addition to permutation and projection of the *embedded* IND. IR3 extends the transitivity rule. It requires not only the RHS of the first CIND to match the LHS of the second CIND, but also their pattern tuples to be matched, *i.e.,* $t_{p_1}[Y_p] = t_{p_2}[Y_p]$.

Observe that rules IR1–IR3 do not consider the interaction between the pattern tuples and the embedded INDs, or the complication introduced by finite-domain attributes. Hence, in contrast to classical INDs, these three rules are not enough to characterize the implication analysis of CINDs. In light of these, we need rules IR4–IR8, which do not find a counterpart in the inference system for INDs.

IR4 allows us to instantiate attributes in X and their corresponding attributes in Y. Given a CIND $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_p)$, we can take an attribute A_j from X and the corresponding B_j in Y, assign a value in $\mathsf{dom}(A_j)$ to them, and move the attribute A_j (resp. B_j) to the pattern tuple X_p (resp. Y_p) of the CIND.

IR5 enhances the LHS pattern of a CIND by adding an attribute to the pattern X_p. Consider a CIND $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_p)$. For any attribute $A \in \mathsf{attr}(R_a)$ that is in neither X nor X_p, we can add A to X_p with an arbitrary value from $\mathsf{dom}(A)$. Intuitively, if ψ holds for all data values in $\mathsf{dom}(A)$, then it also holds for a specific value in $\mathsf{dom}(A)$.

IR6 weakens the RHS pattern of a CIND by removing an attribute from Y_p. If $(R_a[X; X_p] \subseteq R_b[Y; Y_p, B], t_p)$ holds, then for each tuple t_a in R_a that satisfies the pattern $t_p[X_p]$, there exists a matching tuple t_b in R_b that satisfies the pattern $t_p[Y_p, B]$. If an attribute is removed from Y_p, the CIND certainly still holds since the same tuple t_b satisfies $t_p[Y_p]$.

Finally, IR7 and IR8 are only needed in the presence of attributes with a finite domain. IR7 says that if we have a set of CINDs that are pairwise identical except for the value $t_p[A]$ of a finite-domain attribute A, and the union of all those $t_p[A]$ values covers the entire domain $\mathsf{dom}(A)$, then we can replace the set of CINDs by a single CIND in which attribute A is removed from X_p. That is, the presence of A in the LHS pattern has no effect at all, and hence, we can just exclude it from the CINDs.

IR8 is the "inverse" of IR4. If IR4 is applied to a CIND ψ to instantiate attributes A and B in the pattern tuple, when $t_p[A]$ ranges over all the values of $\mathsf{dom}(A)$, then IR8 can take all those CINDs and restore ψ. In short, IR8 merges a set of m CINDs if (1) they differ only in the value of $t_{p_i}[A]$, (2) $t_{p_i}[A]$ ranges over all the values in the domain $\mathsf{dom}(A)$, and (3) there exists an attribute B in the RHS of each CIND such that $t_{p_i}[A] = t_{p_i}[B]$.

Example 3.3: Recall ψ_1 and ψ_3 from Exam-

IR1: (*reflexivity*) If X is a list of distinct attributes of R, then $(R[X; \mathsf{nil}] \subseteq R[X; \mathsf{nil}], t_p)$, where $t_p = \emptyset$.

IR2: (*projection and permutation*) If $(R_a[A_1, \ldots, A_m; X_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_p)$, then $(R_a[A_{i_1}, \ldots, A_{i_k}; X'_p] \subseteq R_b[B_{i_1}, \ldots, B_{i_k}; Y'_p], t'_p)$, where (1) $\{i_1, .., i_k\}$ is a list of distinct integers in $\{1, \ldots, m\}$, or $A_1, \ldots, A_m = B_1, \ldots, B_m = \mathsf{nil}$; (2) X'_p and Y'_p are permutations of X_p and Y_p, respectively; and (3) $t'_p = t_p[X'_p \| Y'_p]$.

IR3: (*transitivity*) If $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_{p_1})$ and $(R_b[Y; Y_p] \subseteq R_c[Z; Z_p], t_{p_2})$, then $(R_a[X; X_p] \subseteq R_c[Z; Z_p], t_{p_3})$, where $t_{p_1}[Y_p] = t_{p_2}[Y_p]$, $t_{p_3}[X_p] = t_{p_1}[X_p]$, and $t_{p_3}[Z_p] = t_{p_2}[Z_p]$.

IR4: (*instantiation*) If $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_p)$, then $(R_a[X \setminus \{A_j\}; A_j, X_p] \subseteq R_b[Y \setminus \{B_j\}; B_j, Y_p], t'_p)$, where $X = \{A_1, \ldots, A_m\}$, $Y = \{B_1, \ldots, B_m\}$, $j \in [1, m]$, $t'_p[A_j] \in \mathsf{dom}(A_j)$, $t'_p[B_j] = t'_p[A_j]$, and $t'_p[X_p \| Y_p] = t_p$.

IR5: (*LHS expansion*) If $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_p)$, then $(R_a[X; A, X_p] \subseteq R_b[Y; Y_p], t'_p)$, where $A \in \mathsf{attr}(R_a) \setminus (X \cup X_p)$, $t'_p[A] \in \mathsf{dom}(A)$, and $t'_p[X_p \| Y_p] = t_p$.

IR6: (*RHS reduction*) If $(R_a[X; X_p] \subseteq R_b[Y; B, Y_p], t_p)$, then $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t'_p)$, where $t'_p = t_p[X_p \| Y_p]$.

IR7: (*finite-domain attribute elimination*) If $(R_a[X; A, X_p] \subseteq R_b[Y; Y_p], t_{p_i})$ $(i \in [1, m])$, then $(R_a[X; X_p] \subseteq R_b[Y; Y_p], t_p)$, where $t_{p_1}[X_p \| Y_p] = \ldots = t_{p_m}[X_p \| Y_p] = t_p[X_p \| Y_p]$, $A \in \mathsf{finattr}(R_a)$, and $\mathsf{dom}(A) = \{t_{p_1}[A], \ldots, t_{p_m}[A]\}$.

IR8: (*finite-domain attribute abstraction*) If $(R_a[X; A, X_p] \subseteq R_b[Y; B, Y_p], t_{p_i})$ $(i \in [1, m])$, then $(R_a[A, X; X_p] \subseteq R_b[B, Y; Y_p], t_p)$, where $t_{p_1}[X_p \| Y_p] = \ldots = t_{p_m}[X_p \| Y_p] = t_p[X_p \| Y_p]$, $t_{p_i}[A] = t_{p_i}[B]$ for each $i \in [1, m]$, $A \in \mathsf{finattr}(R_a)$, and $\mathsf{dom}(A) = \{t_{p_1}[A], \ldots, t_{p_m}[A]\}$.

Figure 3: Inference System $\mathcal{I}$ for CINDs

ple 2.1. From ψ_1, we can derive ψ_3 by using rule IR5, *i.e.,* $\{\psi_1\} \vdash_{\mathcal{I}} \psi_3$.

The next example demonstrates that finite-domain attributes make the inference process more intricate. Consider a database that consists of three relations $\mathsf{R_1(ABC)}$, $\mathsf{R_2(EFG)}$, and $\mathsf{R_3(GHK)}$ such that $\mathsf{dom}(B) = \{b_1, b_2\}$ is finite and all the other attributes have an infinite domain, *e.g.,* string. Let $\{\psi'_1, \psi'_2, \psi'_3, \psi'\}$ be a set of CINDs, where

$\psi'_1 = (R_1[A; B] \subseteq R_2[E; G], (b_1 \parallel g))$,
$\psi'_2 = (R_1[A; B] \subseteq R_3[H; G], (b_2 \parallel g))$,
$\psi'_3 = (R_2[E; \mathsf{nil}] \subseteq R_3[H; G], (\mathsf{nil} \parallel g))$, and
$\psi' = (R_1[A; \mathsf{nil}] \subseteq R_3[H; \mathsf{nil}], \emptyset)$.

Then ψ' can be derived from $\{\psi'_1, \psi'_2, \psi'_3\}$:

(1) $(R_2[E; G] \subseteq R_3[H; G], (g \parallel g))$, ψ'_3, IR5
(2) $(R_1[A; B] \subseteq R_3[H; G], (b_1 \parallel g))$, (1), ψ'_1, IR3
(3) $(R_1[A; \mathsf{nil}] \subseteq R_3[H; G], (\mathsf{nil} \parallel g))$, (2), ψ'_2, IR7
(4) $\psi' = (R_1[A; \mathsf{nil}] \subseteq R_3[H; \mathsf{nil}], \emptyset)$, (3), IR6

That is, $\{\psi'_1, \psi'_2, \psi'_3\} \vdash_{\mathcal{I}} \psi'$. □

The soundness and completeness of $\mathcal{I}$. We next show that $\mathcal{I}$ is sound and complete for the implication analysis of CINDs. That is, for any set Σ of CINDs and another CIND φ defined over the same schema $\mathcal{R}$, $\Sigma \models \psi$ iff $\Sigma \vdash_{\mathcal{I}} \psi$.

As we have seen from Example 3.1, the presence of finite-domain attributes compli-

cates the implication analysis of CFDs. This is also the case for CINDs: when some attributes in Σ or φ have a finite domain, the implication analysis is more intricate, as indicated by Example 3.3 and the rules IR7 and IR8 in $\mathcal{I}$. In light of this, we distinguish two settings: (1) in the absence of finite-domain attributes, *i.e.,* when none of the attributes in Σ or φ has a finite domain, and (2) the general setting, when some attributes in Σ or φ may have a finite domain. We next show that a set of the rules of $\mathcal{I}$ is sound and complete in both settings.

In the absence of finite-domain attributes. This is the setting under which the inference system for standard INDs was developed [20] (see Section 6 for a detailed discussion). In this context, our main result about CIND inference is that the inference rules IR1–IR6 of $\mathcal{I}$ make a sound and complete inference system for CINDs. For a set Σ of CINDs and another CIND φ, we use $\Sigma \vdash_{\mathcal{I}(1-6)} \varphi$ to denote that φ can be proved from Σ using rules IR1–IR6 of $\mathcal{I}$.

Theorem 2. *In the absence of finite-domain attributes, the inference rules* IR1–IR6 *of* $\mathcal{I}$ *are sound and complete for the implication analysis of CINDs.*

Proof sketch: For the soundness, we show that given a set $\Sigma \cup \{\psi\}$ of CINDs, if $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$ by using IR1–IR6, then $\Sigma \models \psi$. This can be readily verified by induction on the length of proofs with IR1–IR6, by showing that the application of each of IR1–IR6 is sound, by the definition of CINDs.

For the completeness, we show that given a set of CINDs $\Sigma \cup \{\psi\}$ defined over a relational schema $\mathcal{R}$, if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$, *i.e., ψ* is provable from Σ by using rules IR1–IR6. The proof consists of three parts. (1) We first develop a chase procedure to characterize $\Sigma \models \psi$. The chase process starts with a single-tuple instance of $\mathcal{R}$, and repeatedly adds tuples (one at a time) to the instance by applying CINDs in Σ until no more CINDs can be applied. (2) We then show that the chase process always terminates, and moreover, that if $\Sigma \models \psi$, then the resulting instance satisfies ψ. (3) Based on (1) and (2), we finally show that if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$, by showing that the chase process can be simulated by the applications of rules IR1–IR6. The detailed proof can be found in the appendix. □

Remark. The proof of Theorem 2 is inspired by the proof of their IND counterpart given in [20]. Nonetheless, our proof has to deal with six rules with constant patterns, whereas the inference system for INDs consists of three rules and does not need to consider constant patterns [20].

In the general setting. As indicated by Example 3.3, the presence of finite-domain attributes complicates the inference process for the implication analysis of CINDs, for which rules IR7 and IR8 are used to deal with finite-domain attributes. Indeed, while one can verify that the inference system of [20] is also complete for standard INDs in the presence of finite-domain attributes, whereas the rules IR1–IR6 are no longer complete for CIND implication here, as shown below.

Example 3.4: Consider a database schema $\mathcal{R}$ with relations $R_1(ABCD)$, $R_2(EFGH)$, and $R_3(IJKL)$ such that $\mathsf{dom}(A) = \{a, b, c\}$, $\mathsf{dom}(D) = \{d, e\}$, $\mathsf{dom}(G) = \{d, e, f\}$, $\mathsf{dom}(L) = \{g, h\}$ and all the other attributes have an infinite domain, *e.g.,* string. Consider a set $\Sigma = \{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5\}$ of CINDs and another CIND ψ, all defined on $\mathcal{R}$, where

$\psi_1 = (R_1[BC; AD] \subseteq R_2[EF; GH], (a, d \parallel d, f))$,
$\psi_2 = (R_1[BC; AD] \subseteq R_3[IJ; K], (b, d \parallel k))$,
$\psi_3 = (R_3[IJ; K] \subseteq R_2[EF; GH], (k \parallel d, g))$,
$\psi_4 = (R_1[BC; A] \subseteq R_2[EF; GH], (c \parallel d, h))$,
$\psi_5 = (R_1[BC; D] \subseteq R_2[EF; GH], (e \parallel e, k))$, and

$\psi = (R_1[BCD; \mathsf{nil}] \subseteq R_2[EFG; \mathsf{nil}], \emptyset)$.

Although $\Sigma \models \psi$, one can verify that $\Sigma \not\vdash_{\mathcal{I}(1\text{–}6)} \psi$, by showing that the existence of a tuple t_1 of schema R_1 does not guarantee the existence of a tuple t_2 of schema R_2 such that $t_2[EFG] = t_1[BCD]$, using the chase process for rules IR1–IR6 developed in the detailed proof of Theorem 2. In other words, IR1–IR6 are no longer complete for proving ψ from Σ. In contrast, ψ can be proved from Σ by using IR1–IR8, as follows:

(1) $(R_1[BC; AD] \subseteq R_2[EF; GH], (b, d \parallel d, g)), \psi_2, \psi_3$, IR3
(2) $(R_1[BC; AD] \subseteq R_2[EF; GH], (c, d \parallel d, h)), \psi_4$, IR5
(3) $(R_1[BC; AD] \subseteq R_2[EF; G], (a, d \parallel d)), \psi_1$, IR6
(4) $(R_1[BC; AD] \subseteq R_2[EF; G], (b, d \parallel d))$, (1), IR6
(5) $(R_1[BC; AD] \subseteq R_2[EF; G], (c, d \parallel d))$, (2), IR6
(6) $(R_1[BC; D] \subseteq R_2[EF; G], (d \parallel d))$, (3),(4),(5), IR7
(7) $(R_1[BC; D] \subseteq R_2[EF; G], (e \parallel e)), \psi_5$, IR6
(8) $\psi = (R_1[BCD; \mathsf{nil}] \subseteq R_2[EFG; \mathsf{nil}], \emptyset)$, (6),(7), IR8

That is, $\Sigma \vdash_{\mathcal{I}} \psi$. □

In the general setting, we show that IR1–IR8 are sound and complete.

Theorem 3. *The inference system $\mathcal{I}$ is sound and complete for the implication of CINDs in the general case, where finite-domain attributes may be present.*

Proof sketch: The soundness of $\mathcal{I}$ can be verified by induction on the length of $\mathcal{I}$-proofs.

For the completeness of $\mathcal{I}$, consider a set $\Sigma \cup \{\psi\}$ of CINDs defined on a database schema $\mathcal{R}$. We show that if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}} \psi$. The proof consists of five parts. (1) We first show that it suffices to consider a special form of CINDs. (2) We then develop a chase procedure for the special form of CINDs, which extends the one given in the proof of Theorem 2 to further deal with finite-domain attributes, and (3) we show that the chase process always terminates. (4) In addition, we establish a certain property of the chase procedure, by characterizing conditions under which $\Sigma \cup \{\psi\}$ holds and showing that the chase ensures the satisfaction of the condition. Finally, (5) we show that if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}} \psi$ by using rules IR1–IR8 and the property. We refer the interested reader to the appendix for a detailed proof. □

Unrestricted implication. We have so far only considered *finite* implication, when *finite* databases are considered, *i.e.,* for database instances in which each relation is a finite set of tuples. A CIND ψ is finitely implied by Σ if for every *finite* database D, if $D \models \Sigma$, then $D \models \psi$. For theoretical interest one may also want to consider *unrestricted* implication, where ψ is implied by Σ if for every database D, either *finite* or *infinite*, if $D \models \Sigma$, then $D \models \psi$. To distinguish these, we denote *finite* implication and *unrestricted* implication by $\Sigma \models_{fin} \psi$ and $\Sigma \models_{unr} \psi$, respectively.

The result below tells us that, however, for CINDs these notions are equivalent. As a result, we can focus on finite implication for CINDs, and use $\Sigma \models \psi$ to denote both $\Sigma \models_{fin} \psi$ and $\Sigma \models_{unr} \psi$.

Proposition 4. *Finite implication and unrestricted implication coincide for CINDs.*

Proof: By definition, unrestricted implication entails finite implication. That is, if $\Sigma \models_{unr} \psi$, then $\Sigma \models_{fin} \psi$. Conversely, it is easy to verify that the inference system $\mathcal{I}$ is sound for unrestricted implication, *i.e.,* if $\Sigma \vdash_{\mathcal{I}} \psi$, then $\Sigma \models_{unr} \psi$. Moreover, by Theorem 3, if $\Sigma \models_{fin} \psi$, then $\Sigma \vdash_{\mathcal{I}} \psi$. From these it follows that finite implication entails unrestricted implication, *i.e.,* if $\Sigma \models_{fin} \psi$, then $\Sigma \models_{unr} \psi$. □

3.2.2. The Complexity of the Implication Analysis

We next establish the computational complexity bounds for the implication analysis of CINDs. We investigate the problem again in

two settings, namely, in the absence of finite-domain attributes and in the general setting when finite-domain attributes may be present.

In the absence of finite-domain attributes. It is known that for standard INDs in this setting, the implication problem is PSPACE-complete [20]. Below we show that the implication problem for CINDs retains the same complexity as their standard counterpart for INDs in this setting.

Theorem 5. *The implication problem for CINDs is PSPACE-complete in the absence of attributes with finite domains.*

Proof sketch: It is known that the implication problem for INDs is PSPACE-complete in the absence of finite-domain attributes [20]. Since CINDs subsume INDs, the implication problem for CINDs is also PSPACE-hard.

To show that the implication problem for CINDs is in PSPACE in the absence of finite-domain attributes, it suffices to give a linear space non-deterministic algorithm for deciding whether $\Sigma \models \psi$, along the same lines as its counterpart for INDs (see [13, 20]). For if it holds, then by Savitch's theorem [21], there is a deterministic quadratic-space algorithm for checking whether $\Sigma \models \psi$, and hence, the implication problem is in PSPACE. We give such an algorithm based on the chase procedure developed in the proof of Theorem 2, in which only a single tuple is kept at all time, and old tuples are replaced by new tuples derived by no-deterministically picking CINDs in Σ (see the appendix for the details of the algorithm). □

In the general setting. When finite-domain attributes are present, the implication analysis of CINDs becomes more involved. Nevertheless, the increased complexity is not incurred only by the presence of finite-domain attributes. Indeed, a close examination of the proofs of [13, 20] reveals that while the PSPACE-completeness for the implication analysis of standard INDs was established for relations with infinite domains only, the proofs remain intact when finite-domain attributes are present. That is, the following result holds.

Proposition 6. *In the presence of finite-domain attributes, the implication problem for INDs remains PSPACE-complete.*

Theorem 5 and Proposition 6 together tell us that neither finite-domain attributes nor constant patterns alone complicate the implication problem. However, below we show that when they are taken together, the implication analysis becomes more intriguing. That is, their interaction makes the implication problem for CINDs harder.

Theorem 7. *In the general setting, the implication problem for CINDs is EXPTIME-complete.*

Proof sketch: To show that the problem is in EXPTIME, we develop an algorithm that, given a set $\Sigma \cup \{\psi\}$ of CINDs defined on instances of a relational schema $\mathcal{R}$, returns 'yes' if and only if $\Sigma \models \psi$. The algorithm is in $O(2^{n^k})$ time, where k is a constant and n is the size of $\mathcal{R}$, Σ and ψ. For the lower bound, we show that the problem is EXPTIME-hard by reduction from the two-player game of corridor tiling problem (TPG-CT), which is EXPTIME-complete [22, 23]. We refer the interested reader to the appendix for a detailed proof. □

4. Interaction between CINDs and CFDs

We have seen that CINDs do not make their satisfiability and implication problems much harder than their traditional counterpart INDs. At the very least, these problems remain decidable for CINDs. In contrast, we show in this

section that when CINDs and CFDs are taken together, the static analyses become far more intriguing. As remarked earlier, in schema matching and data cleaning it is often necessary to use both CINDs and CFDs.

Recall the definition of CFDs given in [1] and presented in Section 3.1. For CFDs the following have been established in [1]. (a) The satisfiability problem for CFDs is NP-complete. (b) The implication problem of CFDs is finitely axiomatizable. (c) The implication problem for CFDs is coNP-complete. (d) The satisfiability and implication problems are in $O(n^2)$ time, where n is the size of the given CFDs, when the CFDs do not involve attributes with a finite domain.

While CFDs alone already complicate the static analyses, we next show that CFDs and CINDs together make our lives much harder.

Implication analysis. It is not surprising that the implication problem for CINDs and CFDs is undecidable and is not finitely axiomatizable, since the problem is already undecidable for standard INDs and FDs (see, *e.g.,* [13]), and CINDs and CFDs subsume INDs and FDs, respectively. The result remains intact even in the absence of attributes with a finite domain, since the undecidability proof of the implication problem for FDs and INDs use attributes with infinite domains only (see [13]).

Proposition 8. *The implication problem for CINDs and CFDs is undecidable, and is not finitely axiomatizable, even in the absence of finite-domain attributes.*

Satisfiability analysis. Given a set of CFDs and a set of CINDs that are separately satisfiable, when they are put together, there may be conflicts among them, which make the set of CFDs and CINDs unsatisfiable, as illustrated by the following example.

Example 4.1: Consider a relation R with $\mathsf{attr}(R) = \{A, B\}$, on which we define a CFD $\phi = (R : A \rightarrow B, (\_ \parallel a))$ and a CIND $\psi = (R[\mathsf{nil}; B] \subseteq R[\mathsf{nil}; B], (\mathsf{nil} \parallel b))$, where a and b are distinct constants. Obviously, there exists a nonempty instance of R that satisfies ϕ and there is an instance satisfying ψ. However, there exists *no* nonempty instance of R that satisfies both ψ and ϕ. To see this, assume that such an instance D exists. Then ψ tells us that as long as D is nonempty, there is a tuple t in D such that $t[B] = b$. In contrast, ϕ requires that $t[B] = a$, violating ψ. □

While the undecidability of the implication problem for CINDs and CFDs is expected, the following result is a little surprising. The undecidability can be verified by reduction from the implication problem for standard FDs and INDs. The undecidability remains intact in the absence of attributes with a finite domain.

Theorem 9. *The satisfiability problem for CFDs and CINDs is undecidable, even in the absence of finite-domain attributes.*

Proof: We show that the satisfiability problem for CINDs and CFDs is undecidable by reduction from the implication problem for standard FDs and INDs, which is undecidable even in the absence of attributes with finite domains (see, *e.g.,* [13]).

More specifically, given an instance of the implication problem, namely, a set Σ of FDs and INDs and an single FD $\varphi = (R_g\colon X \rightarrow A)$ over a database $\mathcal{R}(R_1, \ldots, R_g, \ldots, R_n)$, we define another set Σ' of CINDs and CFDs over $\mathcal{R}$ such that Σ' is satisfiable if and only if $\Sigma \not\models \varphi$.

Let Σ' contain every $\phi \in \Sigma$ as well as the following CINDs:

- $(R_i[\mathsf{nil}; \mathsf{nil}] \subseteq R_g[\mathsf{nil}; A], (\mathsf{nil} \parallel c_1))$ for $i = 1, \ldots, n$; that is, if any R_i is nonempty, then there exists an R_g tuple t such that $t[A] = c_1$;

- $(R_g[X; A] \subseteq R_g[X; A], (c_1 \parallel c_2))$ with $c_1 \neq c_2$ and $c_1, c_2 \in \mathsf{dom}(A)$; that is, if there exists an R_g tuple t_1 with $t_1[A] = c_1$, then there exists an R_g tuple t_2 with $t_2[A] = c_2$ whereas $t_1[X] = t_2[X]$.

We next show that Σ' is satisfiable if and only if $\Sigma \not\models \varphi$.

Assume first that Σ' is satisfiable, then obviously there exists a nonempty instance D of $\mathcal{R}$ such that $D \models \Sigma'$. From this it follows that $D \models \Sigma$ and moreover, that $D \not\models \varphi$, since two tuples t_1, t_2 described above must appear in D.

Conversely, assume that $\Sigma \not\models \varphi$. Then there exists a database D such that $D \models \Sigma$ but $D \not\models \varphi$. Since $D \not\models \varphi$, there must exist two R_g tuples t_1, t_2 such that $t_1[X] = t_2[X]$, $t_1[A] = a$, $t_2[A] = a'$, but $a \neq a'$. Assume *w.l.o.g.* that c_1, c_2 are two constants that do not appear in D, which always exist since $\mathsf{dom}(A)$ is infinite. Define a mapping ρ such that $\rho(a) = c_1$, $\rho(a') = c_2$ and $\rho(d) = d$ for all other constants d in D. Denote the database obtained this way as D'. Then it is easy to verify that D' is a witness that satisfies Σ', *i.e.,* Σ' is satisfiable.

Putting these together, we have shown that Σ' is satisfiable if and only if $\Sigma \not\models \varphi$. □

5. Two Special Classes of CINDs

Proposition 6 and Theorem 7 tell us that it is rather expensive to reason about INDs or CINDs. Indeed, the implication analyses of INDs and CINDs are beyond reach in practice.

However, not all is lost: in practice one often needs only a fragment of INDs or CINDs with a lower complexity. That is, we do not have to pay the price of the complexity of full-fledged INDs or CINDs when we only need certain special cases of the dependencies.

We next investigate two special cases of INDs and CINDs, namely, acyclic and unary inclusion dependencies.

5.1. Acyclic CINDs

One of the well-studied fragments of INDs is identified as sets of acyclic INDs [24] (see *e.g.,* [13]). A set Σ of INDs over a relational schema $\mathcal{R}$ is *acyclic* if there exists no sequence $R_i[X_i] \subseteq S_i[Y_i]$ $(i \in [1, n])$ of INDs in Σ, where $R_{i+1} = S_i$ for each $i \in [1, n-1]$, and $R_1 = S_n$. We refer to such a set of INDs as a set of *AINDs*.

Acyclic CINDs. Along the same lines as AINDs, we define acyclic CINDs.

A set Σ of CINDs is *acyclic* if there exists no sequence $(R_i[X_i; X_{pi}] \subseteq S_i[Y_i; Y_{pi}], t_{pi})$ $(i \in [1, n])$ of CINDs in Σ such that $R_{i+1} = S_i$ for $i \in [1, n-1]$, and $R_1 = S_n$. We refer to such a set of CINDs as a set of ACINDs.

It is common to find a set of CINDs acyclic in practice. For example, the CINDs in a Web database [25] and a drug database [26] are both acyclic. Moreover, the set of CINDs given in Example 2.1 and the set of CINDs in Example 3.3 are both acyclic as well.

We next show that ACINDs indeed make our lives easier, *i.e.,* they allow more efficient static analysis. From Theorem 1, it follows that any set of ACINDs is satisfiable. Hence we shall focus on the implication analysis of ACINDs, *i.e.,* the problem for determining, given a set Σ of ACINDs and a CIND φ, whether $\Sigma \models \varphi$. We study the implication problem for ACINDs in the absence of finite-domain attributes and in the general setting when finite-domain attributes may be present. Note that while Σ is acyclic, $\Sigma \cup \{\varphi\}$ may be cyclic.

In the absence of finite-domain attributes. It has been shown that in this setting, the implication problem for AINDs is NP-complete [14]. We next show that the implication problem for ACINDs retains the same complexity as its standard counterpart, *i.e.,*ACINDs do not complicate the implication analysis in the absence of finite-domain attributes.

Corollary 10. *The implication problem for ACINDs is NP-complete in the absence of attributes with a finite domain.*

Proof: For the lower bounds, note that the implication problem for acyclic INDs is already NP-hard [14]. Since acyclic INDs are a special case of acyclic CINDs, the implication problem for ACINDs is also NP-hard.

We next show that the problem is in NP. Consider the non-deterministic algorithm given in the proof of Theorem 5 for determining whether a set Σ of CINDs implies another CIND ψ (see the appendix). When Σ is a set of ACINDs without finite-domain attributes, the initial tuple t_0 can be replaced by other tuples for at most n times, where n is the number of relations in the database schema $\mathcal{R}$. That is, the linear space non-deterministic algorithm runs in polynomial time. Hence, it is indeed an NP algorithm for deciding whether $\Sigma \models \psi$. As a result, the implication problem for ACIND is in NP without finite-domain attributes. □

In the general setting. The presence of finite-domain attributes does not make the implication analysis of AINDs harder. Indeed, the NP algorithm given in [14] for checking the implication of traditional AINDs still works for AINDs in the presence of finite-domain attributes. Moreover, it is known [14] that the implication analysis of AINDs is NP-hard in the absence of finite-domain attributes; this lower bound obviously carries over to the more general setting when finite-domain attributes may be present. Hence we have the following.

Corollary 11. *The implication problem for AINDs remains NP-complete when finite-domain attributes may be present.*

In contrast, the presence of finite-domain does complicate the implication analysis of ACINDs: the implication problem for ACINDs in this setting becomes PSPACE-complete, up from NP-complete for ACINDs in the absence of finite-domain attributes.

Theorem 12. *In the general setting, the implication problem for ACINDs is PSPACE-complete.*

Proof sketch: We show that the implication problem for ACINDs is in PSPACE in the general setting, by giving a linear space non-deterministic algorithm for determining whether $\Sigma \not\models \psi$, *i.e.,* the complement of $\Sigma \models \psi$. This suffices. For if it holds, then (a) by Immerman–Szelepcsényi theorem [27, 28], there exists a linear space non-deterministic algorithm for deciding whether $\Sigma \models \psi$; and (b) by Savitch's theorem [21], there is a deterministic quadratic-space algorithm for checking whether $\Sigma \models \psi$. Therefore, the problem is in PSPACE. For the lower bound, we show that the problem is PSPACE-hard by reduction from the Q3SAT problem, which is known to be PSPACE-complete (cf. [29]). A detailed proof is given in the appendix. □

5.2. Unary CINDs

Another well-studied fragment of INDs is the class of unary inclusion dependencies, defined as follows [13, 15]. A *unary* IND (UIND) is an IND of the form $R_a[A] \subseteq R_b[B]$, where $A \in \mathsf{attr}(R_a)$ and $B \in \mathsf{attr}(R_b)$. That is, a UIND is an IND in which exactly one attribute appears on each side.

Unary CINDs. We next define and investigate unary CINDs along the same lines as UINDs.

A *unary* CIND (UCIND) is a CIND of the form $(R_a[A; X_p] \subseteq R_b[B; Y_p], t_p)$, where A and B are attributes in R_a and R_b, respectively.

UCINDs are an extension of UINDs by incorporating patterns of data values. Observe that patterns X_p and Y_p in a UCIND may have more than one attribute. It is common to find

UCINDs in practice, just like our familiar unary primary keys and their corresponding foreign keys [13]. Below we give another example.

Example 5.1: Consider a database consisting of three relations: student(SSN, name, dept), course(cno, title, dept), and enroll(SSN, cno, grade). The student relation collects all the student records in a university, and course consists of all the courses offered by the university. In contrast, the enroll relation aims to maintain a complete record of the CS courses registered by students in the CS department.

One would naturally want to design the following CINDs:

(student[SSN; dept] ⊆ enroll[SSN; nil], ('CS' ‖ nil)),
(course[cno; dept] ⊆ enroll[cno; nil], ('CS' ‖ nil)),
(enroll[SSN; nil] ⊆ student[SSN; nil], ∅), and
(enroll[cno; nil] ⊆ course[cno; nil], ∅).

All these CINDs are UCINDs. □

We next investigate the static analysis of UCINDs. By Theorem 1 we do not have to worry about the satisfiability problem for UCINDs. Hence below we focus on the implication problem for UCINDs, *i.e.,* the problem for determining, given a set Σ of UCINDs and another UCIND φ, whether $\Sigma \models \varphi$. We study the problem in the absence of finite-domain attributes and in the general setting.

In the absence of finite-domain attributes. It has been shown that the implication problem for UINDs is in polynomial time (PTIME) in this setting [15]. We next show that the implication analysis of UCINDs can also be conducted efficiently when finite-domain attributes are not present.

Theorem 13. *The implication problem for UCINDs is in polynomial time in the absence of finite-domain attributes.*

Proof sketch: We give a PTIME algorithm for checking whether $\Sigma \models \psi$ when ψ and the CINDs in Σ are unary. The algorithm can be found in the appendix. □

In the general setting. For UINDs, the presence of finite-domain attributes does not complicate the implication analysis. Indeed, a close examination of the PTIME algorithm of [15] for checking traditional UIND implication reveals that it still works in the general setting. Hence we have the following.

Corollary 14. *The implication problem for UINDs remains in polynomial time in the general setting.*

This is, however, no longer the case for UCINDs in this setting.

Theorem 15. *The implication problem for UCINDs is coNP-complete in the general setting.*

Proof sketch: We show that the problem is in coNP, by developing an NP algorithm that, given a set $\Sigma \cup \{\psi\}$ of UCINDs, checks whether $\Sigma \not\models \psi$. We verify that the problem is coNP-hard by reduction from the 3SAT problem to the complement of the problem (*i.e.,* to decide whether $\Sigma \not\models \psi$). It is known that 3SAT is NP-complete (cf. [30]). We defer a detailed proof to the appendix. □

Acyclic UCINDs. One might be tempted to think that it would simplify the implication analysis if we further restrict UCINDs to be acyclic. Unfortunately, this is not the case. Indeed, in the lower bound proof of Theorem 15, the UCINDs used are *acyclic*. From this it follows that the implication problem for acyclic UCINDs remains coNP-complete.

Corollary 16. *The implication problem for acyclic UCINDs is coNP-complete in the general setting.*

6. Related Work

Data dependencies have been studied for relational databases since the introduction of FDs by Codd [31] in 1972 (see, *e.g.,* [13, 32] for details). Recently, data dependencies have generated renewed interests for improving data quality ([1, 2, 3, 4, 5, 6, 7, 33, 34, 35, 36, 37]) and for schema mapping ([10, 11, 16]).

The theory of INDs was established in [20], which developed a sound and complete inference system and the PSPACE-completeness for the implication analysis of INDs. Acyclic INDs were introduced in [24], and their implication problem was shown to be NP-complete in [14]. Unary INDs were studied in [15], which provided a sound and complete inference system for UINDs and FDs, and proved the PTIME bound of the implication problem for UINDs and FDs put together (see [13] for a survey on INDs, AINDs and UINDs). While not explicitly stated, the proofs of these results indicate that the implication analysis was conducted in the absence of finite-domain attributes. In this paper we verify that the complexity bounds for INDs, AINDs and UINDs remain intact in the presence of finite-domain attributes.

CINDs, ACINDs and UCINDs extend INDs, AINDs and UINDs, respectively, by incorporating patterns of data values. For the implication problem in the absence of finite-domain attributes, the lower bounds for CINDs, AINDs and UINDs are inherited from their traditional counterparts, but *not* the upper bounds. When finite-domain attributes may be present, however, none of the results of [14, 15, 18, 20, 24] holds on CINDs. Indeed, the implication problems for CINDs, ACINDs and UCINDs in the general setting have a higher complexity bound than their traditional counterparts.

INDs are a special case of TGDs, which can be expressed as first-order logic sentences of the form:

$$\forall x_1 \ldots \forall x_n[\varphi(x_1, \ldots, x_n) \rightarrow \exists z_1 \ldots \exists z_k \phi(y_1, \ldots, y_m)],$$

where (a) $\{z_1, \ldots, z_k\} = \{y_1, \ldots, y_m\} \setminus \{x_1, \ldots, x_n\}$, (b) φ and ϕ are conjunctions of relation atoms of the form $R(w_1, \ldots, w_l)$ in which $w_1, \ldots, w_l$ are variables (see *e.g.,* [13] for details). In contrast to CINDs, TGDs do not allow constants, and their implication problem is *undecidable* [18]. There have been extensions of TGDs [38] developed for constraint databases, notably constrained tuple-generating dependencies (CTGDs) of the form:

$$\forall \bar{x}(R_1(\bar{x}) \wedge \ldots R_k(\bar{x}) \wedge \xi \rightarrow \exists \bar{y}(R'_1(\bar{x}, \bar{y}) \wedge \ldots \wedge R'_s(\bar{x}, \bar{y}) \wedge \xi'(\bar{x}, \bar{y})),$$

where R_i, R'_j are relation atoms, and ξ, ξ' are arbitrary constraints. While CTGDs support constants and can express CINDs, the increased expressive power comes at a price for static analysis. Indeed, the satisfiability and implication problems are both undecidable for CTGDs.

Closer to our work is the study of CFDs [1]. CFDs extend FDs with pattern tableaux, along the same lines as CINDs. It was shown in [1] that the satisfiability and implication problems for CFDs are NP-complete and coNP-complete, respectively, in the general setting, and they are in PTIME in the absence of finite-domain attributes. Extensions of CFDs have been proposed to support disjunction and negation [39], cardinality constraints and synonym rules [40], built-in predicates ($\neq, <, \leq, >, \geq$) [41], and to specify patterns in terms of value ranges [4]. However, CFDs and their extensions are defined on a single relation and are universally quantified. They cannot express CINDs, and neither CINDs nor their static analyses were studied in [1, 4, 39, 40, 41]. In addition, as we have seen earlier, the satisfiability and implication analysis of CINDs are far more intriguing than their CFD counterparts. An extension of CINDs was recently proposed to support built-in predicates [41], which was

based on the results of this work. Discovering CINDs has been studied in [25].

Research on constraint-based data cleaning has mostly focused on two topics, both proposed by [33]: *repairing* is to find another database that is consistent and minimally differs from the original database (*e.g.,* [8, 9, 42]); and *consistent query answering* is to find an answer to a given query in every repair of the original database (*e.g.,* [33, 43]). A variety of constraint formalisms have been used in data cleaning, ranging from standard FDs and INDs [8, 9, 33], denial constraints (full dependencies) [44], to logic programs (see [34, 35, 36] for surveys). To our knowledge, no prior work has considered CINDs for data cleaning albeit our work [12, 41], and the recent work [26] that makes use of CINDs and CFDs to clean drug data. Moreover, previous work on data cleaning did not study inference, satisfiability and implication analyses of constraints, which are the focus of this paper.

Constraints used in schema matching in practice are typically standard INDs and keys (see, *e.g.,* [10]). Contextual schema matching [11] investigated the applications of contextual foreign keys, a primitive and special case of CINDs, in deriving schema mapping from schema matches. While [11] partly motivated this work, it neither formalized the notion of CINDs nor considered the static analysis of CINDs. There has also been recent work on data exchange (schema mapping) and data integration based on TGDs (see [16, 45, 46] for surveys). However, inference systems and static analyses of constraints are not the focus of the prior work on data exchange and data integration, and none of the results of this work has been established in those lines of research.

The *chase* technique is widely used in implication analysis and query optimization, and has been studied for a variety of dependencies (see, *e.g.,* [13, 15, 18, 47]). Recently it was extended for query reformulation and schema mapping, and a number of sufficient conditions were identified for its termination (see [16, 48] for recent surveys). This work extends the chase technique to study the implication analysis of CINDs, for which the chase process always terminates.

7. Conclusion

We have proposed CINDs, a mild extension of INDs that is important in both contextual schema matching and data cleaning. We have also settled several fundamental problems associated with static analysis of CINDs, from the satisfiability to the finite axiomatizability, to the complexity of the implication problem. Tables 1 and 2 summarize the main results of this work on CINDs, compared with their counterparts for standard INDs. While some proofs for CINDs in the absence of finite-domain attributes are inspired by the proofs of their IND counterparts, most proofs here are more involved. For instance, we have to deal with six inference rules, whereas the inference system for INDs has three rules and does not need to consider constant patterns [20]. In the general setting, our proofs are much more complicated, since we need to cope with the impact of finite-domain attributes and constant patterns together, as indicated by eight inference rules. The techniques for dealing with finite-domain attributes are interesting in their own right.

For the satisfiability analysis, the presence of finite-domain attributes does not complicate the problem of INDs, CINDs, INDs + FDs and CINDs + CFDs. However, while a set of INDs and FDs is always satisfiable [13], the problem of CINDs and CFDs put together is undecidable, because of the presence of data values.

For the implication analysis, in particular, we have developed a sound and complete inference system for CINDs. We have also pro-

Problems	Dependencies	Infinite domain only
The satisfiability problem	INDs	$O(1)$ ([19])
	CINDs	$O(1)$ (Thm. 1)
	INDs + FDs	$O(1)$ ([13])
	CINDs + CFDs	Undecidable (Thm. 9)
The implication problem	INDs	PSPACE-complete ([13, 20])
	CINDs	PSPACE-complete (Thm. 5)
	INDs + FDs	Undecidable ([13])
	CINDs + CFDs	Undecidable (Cor. 8)
	AINDs	NP-complete ([13, 14])
	ACINDs	NP-complete (Cor. 10)
	UINDs	PTIME ([13, 15])
	UCINDs	PTIME (Thm. 13)
	acyclic UINDs	PTIME ([13, 15])
	acyclic UCINDs	PTIME (Thm. 13)
The finite axiomatizability	INDs	IND1, IND2, IND3 ([13, 20])
	CINDs	IR1– IR6 (Thm. 2)
	INDs + FDs	Not finitely axiomatizable ([13])
	CINDs + CFDs	Not finitely axiomatizable (Cor. 8)

Table 1: Summary of the main results in the absence of finite-domain attributes

Problems	Dependencies	General setting
The satisfiability problem	INDs	$O(1)$ ([19])
	CINDs	$O(1)$ (Thm. 1)
	INDs + FDs	$O(1)$ ([13])
	CINDs + CFDs	Undecidable (Thm. 9)
The implication problem	INDs	PSPACE-complete ([13, 20], Pro. 6)
	CINDs	EXPTIME-complete (Thm. 7)
	INDs + FDs	Undecidable ([13])
	CINDs + CFDs	Undecidable (Cor. 8)
	AINDs	NP-complete ([13, 14], Cor. 11)
	ACINDs	PSPACE-complete (Thm. 12)
	UINDs	PTIME ([13, 15], Cor. 14)
	UCINDs	coNP-complete (Thm. 15)
	acyclic UINDs	PTIME ([13, 15])
	acyclic UCINDs	coNP-complete (Cor. 16)
The finite axiomatizability	INDs	IND1, IND2, IND3 ([13, 20])
	CINDs	IR1– IR8 (Thm. 3)
	INDs + FDs	Not finitely axiomatizable ([13])
	CINDs + CFDs	Not finitely axiomatizable (Cor. 8)

Table 2: Summary of the main results in the general setting

vided a complete picture of complexity bounds for the implication analysis of CINDs and INDs, focusing on the following dichotomies:

- with constant patterns (CINDs) vs. their absence (INDs),
- general CINDs vs. ACINDs and UCINDs;
- the absence of finite-domain attributes vs. the general setting in which finite-domain attributes may be present.

We have investigated the impact of these factors on the implication analysis of inclusion dependencies. (a) For traditional INDs, AINDs, UINDs and acyclic UINDs, the presence of finite-domain attributes does not complicate the implication analysis. (b) The presence of constant patterns alone does not increase the complexity. Indeed, the implication problem for CINDs, ACINDs, UCINDs and acyclic UCINDs in the absence of finite-domain attributes retains the same complexity as its counterpart for INDs, AINDs, UINDs, and acyclic UINDs, respectively. Nevertheless, (c) the presence of both constant patterns and finite-domain attributes makes our lives harder. Indeed, the implication problem for CINDs, ACINDs, UCINDs and acyclic UCINDs in the general setting has a higher complexity than its counterpart for INDs, AINDs, UINDs and acyclic UINDs, respectively. These tell us that it is the interaction between constant patterns and finite-domain attributes that complicates the implication analysis.

There is naturally much more to be done. First, we have shown that when CINDs and CFDs are taken together, both the satisfiability problem and the implication problem become *undecidable*. Nevertheless, it is not known yet whether the problems are decidable for CFDs (or FDs) and fragments of CINDs put together, *e.g.,*AINDs, ACINDs, UINDs, UCINDs, acyclic UINDs and acyclic UCINDs. Second, an extension of CINDs was proposed in [41] by supporting built-in predicates ($\neq, <, \leq, >, \geq$), to capture inconsistencies across different relations. We want to find out the impact of these built-in predicates on the static analyses of ACINDs and UCINDs. Third, it is important and practical to develop effective algorithms for discovering CINDs, as studied in [25], along the same lines as their counterparts for CFDs [3, 4, 49]. Finally, both CFDs and CINDs are useful in schema matching and data cleaning. However, effective and efficient data repairing and schema matching algorithms are yet to be developed when both CINDs and CFDs are brought into the play.

Acknowledgements. Ma is supported in part by NSFC grant 61322207, NGFR 973 grant 2014CB340304 and MOST grant 2012BAH46B04. Fan is supported in part by 973 Programs 2012CB316200 and 2014CB340302, NSFC 61133002, Guangdong Innovative Research Team Program 2011D005 and Shenzhen Peacock Program 1105100030834361 of China, as well as EPSRC EP/J015377/1, UK.

References

[1] W. Fan, F. Geerts, X. Jia, A. Kementsietsidis, Conditional functional dependencies for capturing data inconsistencies, ACM Trans. Database Syst. 33 (2) (2008) Article 6.

[2] G. Cong, W. Fan, F. Geerts, X. Jia, S. Ma, Improving data quality: Consistency and accuracy, in: VLDB, 2007, pp. 315–326.

[3] F. Chiang, R. J. Miller, Discovering data quality rules, PVLDB 1 (1) (2008) 1166–1177.

[4] L. Golab, H. J. Karloff, F. Korn, D. Srivastava, B. Yu, On generating near-optimal tableaux for conditional functional dependencies, PVLDB 1 (1) (2008) 376–390.

[5] W. Fan, H. Gao, X. Jia, J. Li, S. Ma, Dynamic constraints for record matching, VLDB J. 20 (4) (2011) 495–520.

[6] W. Fan, J. Li, S. Ma, N. Tang, W. Yu, Interaction between record matching and data repairing, in: SIGMOD, 2011, pp. 469–480.

[7] W. Fan, J. Li, N. Tang, W. Yu, Incremental detection of inconsistencies in distributed data, in: ICDE, 2012, pp. 318–329.
[8] P. Bohannon, W. Fan, M. Flaster, R. Rastogi, A cost-based model and effective heuristic for repairing constraints by value modification, in: SIGMOD, 2005, pp. 143–154.
[9] J. Chomicki, J. Marcinkowski, Minimal-change integrity maintenance using tuple deletions, Information and Computation 197 (1-2) (2005) 90–121.
[10] L. Haas, M. Hernández, H. Ho, L. Popa, M. Roth, Clio grows up: from research prototype to industrial tool, in: SIGMOD, 2005, pp. 805–810.
[11] P. Bohannon, E. Elnahrawy, W. Fan, M. Flaster, Putting context into schema matching, in: VLDB, 2006, pp. 307–318.
[12] L. Bravo, W. Fan, S. Ma, Extending dependencies with conditions, in: VLDB, 2007, pp. 243–254.
[13] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
[14] S. S. Cosmadakis, P. C. Kanellakis, Functional and inclusion dependencies a graph theoretic approach, in: PODS, 1984, pp. 29–37.
[15] S. S. Cosmadakis, P. C. Kanellakis, M. Y. Vardi, Polynomial-time implication problems for unary inclusion dependencies, Journal of the ACM 37 (1) (1990) 15–46.
[16] P. G. Kolaitis, Schema mappings, data exchange, and metadata management, in: PODS, 2005, pp. 61–75.
[17] S. Ginsburg, E. H. Spanier, On completing tables to satisfy functional dependencies, Theor. Comput. Sci. 39 (1985) 309–317.
[18] C. Beeri, M. Vardi, A proof procedure for data dependencies, Journal of the ACM 31 (4) (1984) 718–741.
[19] R. Fagin, M. Y. Vardi, Armstrong databases for functional and inclusion dependencies, Inf. Process. Lett. 16 (1) (1983) 13–19.
[20] M. A. Casanova, R. Fagin, C. H. Papadimitriou, Inclusion dependencies and their interaction with functional dependencies, JCSS 28 (1) (1984) 29–59.
[21] W. J. Savitch, Relationships between Nondeterministic and Deterministic Tape Complexities, JCSS 4 (1970) 177–192.
[22] B. S. Chlebus, Domino-tiling games, JCSS 32 (3) (1986) 374–392.
[23] P. van Emde Boas, The convenience of tilings, in: Complexity, Logic, and Recursion Theory, Marcel Dekker Inc, 1997.
[24] E. Sciore, Comparing the universal instance and relational data models, Advances in Computing Research 3 (1986) 139–162.
[25] J. Bauckmann, Z. Abedjan, U. Leser, H. Müller, F. Naumann, Discovering conditional inclusion dependencies, in: CIKM, 2012, pp. 2094–2098.
[26] O. Curé, Improving the data quality of drug databases using conditional dependencies and ontologies, J. Data and Information Quality 4 (1) (2012) 3.
[27] N. Immerman, Nondeterministic space is closed under complementation, SIAM J. Comput. 17 (5) (1988) 935–938.
[28] R. Szelepcsényi, The meothod of focing for nondeterministic automata, Bulletin of the EATCS 33 (1987) 96–99.
[29] C. H. Papadimitriou, Computational Complexity, Addison Wesley, 1994.
[30] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.
[31] E. F. Codd, Relational completeness of data base sublanguages., in: Data Base Systems: Courant Computer Science Symposia Series 6, Prentice-Hall, 1972, pp. 65–98.
[32] R. Fagin, M. Y. Vardi, The theory of data dependencies - an overview, in: ICALP, 1984, pp. 1–22.
[33] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, TPLP 3 (4-5) (2003) 393–424.
[34] L. Bertossi, Consistent query answering in databases, SIGMOD Record 35 (2) (2006) 68–76.
[35] J. Chomicki, Consistent query answering: Five easy pieces, in: ICDT, 2007, pp. 1–17.
[36] W. Fan, Dependencies revisited for improving data quality, in: PODS, 2008, pp. 159–170.
[37] W. Fan, J. Li, S. Ma, N. Tang, W. Yu, Towards certain fixes with editing rules and master data, VLDB J. 21 (2) (2012) 213–238.
[38] M. J. Maher, D. Srivastava, Chasing Constrained Tuple-Generating Dependencies, in: PODS, 1996, pp. 128–138.
[39] L. Bravo, W. Fan, F. Geerts, S. Ma, Increasing the expressivity of conditional functional dependencies without extra complexity, in: ICDE, 2008, pp. 516–525.
[40] W. Chen, W. Fan, S. Ma, Incorporating cardinality constraints and synonym rules into conditional functional dependencies, Inf. Process. Lett. 109 (14) (2009) 783–789.

[41] W. Chen, W. Fan, S. Ma, Analyses and validation of conditional dependencies with built-in predicates, in: DEXA, 2009, pp. 576–591.
[42] E. Franconi, A. L. Palma, N. Leone, S. Perri, F. Scarcello, Census data repair: a challenging application of disjunctive logic programming, in: LPAR, 2001, pp. 561–578.
[43] J. Wijsen, Database repairing using updates, TODS 30 (3) (2005) 722–768.
[44] A. Lopatenko, L. Bertossi, Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics, in: ICDT, 2007, pp. 179–193.
[45] M. Arenas, J. Pérez, J. Reutter, C. Riveros, Composition and inversion of schema mappings, SIGMOD Record 38 (3) (2009) 17–28.
[46] M. Lenzerini, Data integration: A theoretical perspective, in: PODS, 2002, pp. 233–246.
[47] D. S. Johnson, A. C. Klug, Testing containment of conjunctive queries under functional and inclusion dependencies, JCSS 28 (1) (1984) 167–189.
[48] A. Deutsch, L. Popa, V. Tannen, Query reformulation with constraints, SIGMOD Record 35 (1) (2006) 65–73.
[49] W. Fan, F. Geerts, J. Li, M. Xiong, Discovering conditional functional dependencies, TKDE 23 (5) (2011) 683–698.

Appendix: Proofs

1. Proof of Theorem 2

Theorem 2. *In the absence of finite-domain attributes, the inference rules* IR1–IR6 *of* $\mathcal{I}$ *are sound and complete for the implication analysis of CINDs.*

Proof: For the soundness, we show that given a set $\Sigma \cup \{\psi\}$ of CINDs, if $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$ by using IR1–IR6, then $\Sigma \models \psi$. This can be readily verified by induction on the length of proofs with IR1–IR6, by showing that the application of each of IR1–IR6 is sound, by the definition of CINDs.

For the completeness, we show that given a set of CINDs $\Sigma \cup \{\psi\}$ over a relational schema $\mathcal{R}$, if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$, *i.e.,* ψ is provable from Σ by using IR1–IR6. We assume *w.l.o.g.* that $\mathcal{R} = (R_1, \ldots, R_n)$, and that ψ is $(R_a[A_1, \ldots, A_m; X_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_{p_\psi})$, in which R_a and R_b are in $\mathcal{R}$.

The proof consists of three parts. (1) We first develop a chase procedure to determine whether $\Sigma \models \psi$. The chase process starts with a single-tuple instance of $\mathcal{R}$, and repeatedly adds tuples (one at a time) to the instance by applying CINDs in Σ until no more CINDs can be applied. (2) We then show that the chase process always terminates, and moreover, that if $\Sigma \models \psi$, then the resulting instance satisfies ψ. (3) Based on these, we finally show that if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$.

(1) We first introduce the chase procedure.

We construct a tuple t_a of schema R_a such that (a) $t_a[A_i] = v_i$ for $i \in [1, m]$; (b) $t_a[A_p] = t_{p_\psi}[A_p]$ for each attribute $A_p \in X_p$ of the CIND ψ; and (c) $t_a[A] = v_0$ for the rest of the attributes $A \in \mathsf{attr}(R_a) \setminus (\{A_1, \ldots, A_m\} \cup X_p)$. Here $v_0, v_1, \ldots, v_m$ are $m + 1$ distinct variables that represent data values not appearing in $\Sigma \cup \{\psi\}$.

The chase process starts with an instance $D_0 := (I_1, \ldots, I_a, \ldots, I_n)$ of $\mathcal{R}$ such that the instance I_a of schema R_a contains the single tuple t_a, and for each $i \in [1, n]$ with $i \neq a$, the instance I_i of schema R_i is empty.

The chase adds tuples to the database D_0, one at a time, by making use of a chase operation APPLY. More specifically, given a CIND $\psi' = (R_i[U; U_p] \subseteq R_j[V; V_p], t_{p_{\psi'}})$ in Σ and an instance D, $\text{APPLY}(D, \psi')$ transforms the instance D into a new one D' by applying CIND ψ' to D as follows.

- For each tuple $t_i \in I_i$ with $t_i[U_p] = t_{p_{\psi'}}[U_p]$, if there exists no tuple $t_j \in I_j$ such that $t_j[V] = t_i[U]$ and $t_j[V_p] = t_{p_{\psi'}}[V_p]$, we say that the CIND ψ' is *applicable* to D. If so then the chase adds a tuple t_j to I_j such that (a) $t_j[V] = t_i[U]$, (b) $t_j[V_p] = t_{p_{\psi'}}[V_p]$, and (c) $t_j[E] = v_0$ for all the other attributes $E \in \mathsf{attr}(R_j) \setminus (V \cup V_p)$.
- If there exists no such tuple t_i for the CIND ψ', the instance D remains the same, *i.e.,* $D = \text{APPLY}(D, \psi')$.

The chase process stops when it reaches an instance D_f such that no more CINDs in Σ are applicable to D_f, *i.e.,* when $D_f = \text{APPLY}(D, \psi')$ for all CINDs ψ' in Σ. We refer to such D_f as *a result of* $\text{APPLY}(D, \Sigma)$.

(2) We next show that the chase process always terminates, and that all results $D_f = \text{APPLY}(D, \Sigma)$ satisfies ψ if $\Sigma \models \psi$.

Observe that in any step of the chase process, the database is defined in terms of a finite set of elements, *i.e.,* $m + 1$ variables $\{v_0, v_1, \ldots, v_m\}$, and the constants appearing in the constant patterns of CINDs in $\Sigma \cup \{\psi\}$. There are only finitely many distinct databases with these elements. From this it follows that the chase process always terminates.

Intuitively, the chase process yields a sequence of databases $D_0, D_1, \ldots, D_f$ such that (a) D_0 is the initial instance, and (b) for each $i \in [0, f-1]$, $D_{i+1} := \text{APPLY}(D_i, \psi')$ by applying a CIND ψ' in Σ to D_i. In addition, $D_i \subseteq D_{i+1}$ for all $0 \leq i < f$, and $\text{APPLY}(D_f, \psi') = D_f$ for all ψ' in Σ. That is, the instance D_f is a fixpoint. When it is clear from the context, we use $\mathsf{Chase}(\Sigma, \psi)$ to denote such a D_f; in other words, $\mathsf{Chase}(\Sigma, \psi)$ denotes an arbitrary fixpoint D_f obtained by such a chase process.

It is easy to see that $\mathsf{Chase}(\Sigma, \psi) \models \Sigma$, no matter what fixpoint is obtained by the chase process and is denoted by $\mathsf{Chase}(\Sigma, \psi)$. Thus if $\Sigma \models \psi$, then $\mathsf{Chase}(\Sigma, \psi) \models \psi$. That is, the initial tuple $t_a \in I_a$ enforces the existence of another tuple $t_b \in I_b$ such that (a) $t_b[B_i] = v_i$ for $i \in [1, m]$, and (b) $t_b[Y_p] = t_{p_\psi}[Y_p]$.

Example 1: Consider a database D_0 with three relations shown below, and the CINDs ψ'_1 and ψ'_3 given in Example 3.3. Let Σ be $\{\psi'_1, \psi'_3\}$ and $\psi = (R_1[A; B] \subseteq R_3[H; G], (b_1 \parallel g))$. The chase process works on the database D_0 as follows.

I_1:	A	B	C
t_1:	v_1	b_1	v_0

I_2:	E	F	G

I_3:	G	H	K

(1) Initial instance D_0

I_1:	A	B	C
t_1:	v_1	b_1	v_0

I_2:	E	F	G
t_2:	v_1	v_0	g

I_3:	G	H	K

(2) Instance D_1 after executing $\text{APPLY}(D_0, \psi'_1)$

I_1:	A	B	C
t_1:	v_1	b_1	v_0

I_2:	E	F	G
t_2:	v_1	v_0	g

I_3:	G	H	K
t_3:	g	v_1	v_0

(3) Final instance D_2 after executing $\text{APPLY}(D_1, \psi'_3)$

Note that $\Sigma \models \psi$, and that there exists a tuple t_3 such that $t_3[H] = t_1[A]$ and $t_3[G] =$ 'g'. □

(3) We finally show that if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi$. To prove this, it suffices to show the following Claim.

Claim *1: Assume that the instance I_j of R_j in the chase process contains a tuple t, where (a) for each $E_i \in \mathsf{attr}(R_j)$ ($i \in [1,k]$), $t[E_i] = v_{j_i}$ and v_{j_i} is in $\{v_1, \ldots, v_m\}$, and (b) $t[Z_p]$ consists of only constants for a list Z_p of distinct attributes in $\mathsf{attr}(R_j)$. Then $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi'$, where ψ' is $(R_a[A_{j_1}, \ldots, A_{j_k}; X_p] \subseteq R_j[E_1, \ldots, E_k; Z_p], t'_p)$, $t'_p[X_p] = t_{p_\psi}[X_p]$ and $t'_p[Z_p] = t[Z_p]$.*

For if it holds, we can conclude that if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi$. Indeed, as remarked earlier, $\mathsf{Chase}(\Sigma, \psi) \models \psi$ for all fixpoints obtained by the chase and denoted by $\mathsf{Chase}(\Sigma, \psi)$. Since the initial D_0 contains the tuple t_a, by the definition of chase, there must be a tuple $t_b \in I_b$ in $\mathsf{Chase}(\Sigma, \psi)$ such that (a) $t_b[B_i] = v_i$ for $i \in [1, m]$, and (b) $t_b[Y_p] = t_{p_\psi}[Y_p]$. Hence by Claim 1, $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi$.

We next prove Claim 1 by induction on the length of the instance sequence generated by the chase process.

Base case. When the length is 1, the sequence consists of the initial instance D_0, which contains a single tuple t_a in the instance I_a of schema R_a. We show that $(R_a[X'; X'_p] \subseteq R_a[X'; X'_p], (t_a[X'_p] \parallel t_a[X'_p]))$ for all $X' \subseteq \{A_1, \ldots, A_m\}$ and $X'_p \subseteq X_p$, by repeatedly using IR4 as follows.

(1) $(R_a[X', X'_p; \mathsf{nil}] \subseteq R_a[X', X'_p; \mathsf{nil}], \emptyset)$, IR1

(2) $(R_a[X'; X'_p] \subseteq R_a[X'; X'_p], (t_a[X'_p] \parallel t_a[X'_p]))$, IR4

Inductive case. Assume that Claim 1 holds for the first $i + 1$ instances $D_0, \ldots, D_i$. We next show that Claim 1 also holds on $D_{i+1} := \textsc{apply}(D_i, \psi')$.

If $D_{i+1} = D_i$, the instance D_i is not changed by $\textsc{apply}(D_i, \psi')$, and the claim obviously holds on D_{i+1} since it holds on D_i.

Assume that $D_{i+1} \neq D_i$. Then there must be a single tuple w inserted into the instance I_j of some schema R_j, *i.e.,* $I_j = I_j \cup \{w\}$, as the result of $\textsc{apply}(D_i, \psi')$ for some $\psi' \in \Sigma$. Assume *w.l.o.g.* that $\psi' = (R_i[C_1, \ldots, C_k; U_p] \subseteq R_j[F_1, \ldots, F_k; V_p], t_{p_{\psi'}})$. Since ψ' is applicable to D_i, there exists a tuple u in I_i such that $u[U_p] = t_{p_{\psi'}}[U_p]$. By the induction hypothesis, there is a CIND $\psi_u = (R_a[A_{p_1}, \ldots, A_{p_h}; X_p] \subseteq R_i[C_{p_1}, \ldots, C_{p_h}; U'_p], t_{p_u})$ such that (a) for each attribute $C \in \mathsf{attr}(R_i)$, if $u[C]$ is a constant, then $C \in U'_p$, and if $u[C]$ is a variable in $\{v_1, \ldots, v_m\}$, then $C \in \{C_{p_1}, \ldots, C_{p_h}\}$, (b) $t_{p_u}[U'_p] = u[U'_p]$, and (c) $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi_u$. Based on the tuples w, u and the CINDs ψ' and ψ_u, we can derive the following.

(a) $t_{p_{\psi'}}[U'_p] = u[U'_p]$, $t_{p_u}[U_p] = u[U_p]$, and $U_p \subseteq U'_p$.

(b) Let $\{C_{q_1}, \ldots, C_{q_g}\} = \{C_1, \ldots, C_k\} \cap \{C_{p_1}, \ldots, C_{p_h}\}$, where $0 \leq g \leq \mathsf{min}(k, h)$. Then $\{A_{q_1}, \ldots, A_{q_g}\} \subseteq \{A_{p_1}, \ldots, A_{p_h}\}$, and for the tuple w, $w[F] \in \{v_1, \ldots, v_m\}$ *iff* $F \in \{F_{q_1}, \ldots, F_{q_g}\}$.

(c) For each attribute $F \in \mathsf{attr}(R_j)$, $w[F]$ is a constant *iff* $F \in V_p$ such that $w[F] = t_{p_{\psi'}}[F]$, or $F = F_i$ ($1 \leq i \leq k$) such that $w[F] = u[F]$ and $C_i \in U_c = U'_p \cap \{C_1, \ldots, C_k\}$. We use V_c to denote the list of attributes in $\mathsf{attr}(R_j)$ that corresponds to the list of attributes U_c of $\mathsf{attr}(R_i)$ in ψ'.

(d) $U_c \cap \{C_{q_1}, \ldots, C_{q_g}\} = \emptyset$ and $U_c \subseteq U'_p$.

We next show that $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi_w$, where ψ_w is the CIND $(R_a[A_{p_1}, \ldots, A_{p_{g'}}; X_p] \subseteq R_j[F_{p_1}, \ldots, F_{p_{g'}}; Z'_p], t_{p_{\psi_w}})$. Observe that (a) $\{F_{p_1}, \ldots, F_{p_{g'}}\} \subseteq \{F_{q_1}, \ldots, F_{q_g}\}$ by (2) above, and (b) $Z'_p \subseteq V_p \cup V_c$ by (3).

In addition, we can derive the following.

(a) $(R_i[C_{q_1}, \ldots, C_{q_g}; U_c U_p] \subseteq R_j[F_{q_1}, \ldots, F_{q_g}; V_c V_p], t_{p_1})$, where $t_{p_1}[U_p; V_p] = t_{p_{\psi'}}[U_p; V_p]$ and $t_{p_1}[U_c] = t_{p_1}[V_c]$ $u[U_c] = w[U_c]$ (using ψ', IR2, IR4)

(b) $(R_a[A_{q_1}, \ldots, A_{q_g}; X_p] \subseteq R_i[C_{q_1}, \ldots, C_{q_g}; U'_p], t_{p_2})$, where $t_{p_2}[X_p; U'_p] = t_{p_u}[X_p; U'_p]$ (using ψ_u, IR2)

(c) $(R_i[C_{q_1}, \ldots, C_{q_g}; U'_p] \subseteq R_j[F_{q_1}, \ldots, F_{q_g}; V_c V_p], t_{p_3})$, where $t_{p_3}[U'_p] = t_{p_2}[U'_p]$ and $t_{p_3}[V_c V_p] = t_{p_1}[V_c V_p]$ (using (1), IR5)

(d) $(R_a[A_{q_1}, \ldots, A_{q_g}; X_p] \subseteq R_i[C_{q_1}, \ldots, C_{q_g}; U_c U_p], t_{p_4})$, where $t_{p_4}[X_p; U_c U_p] = t_{p_2}[X_p; U_c U_p]$ (using (2), IR6)

(e) $(R_a[A_{q_1}, \ldots, A_{q_g}; X_p] \subseteq R_j[F_{q_1}, \ldots, F_{q_g}; V_c V_p], t_{p_5})$, where $t_{p_5}[X_p] = t_a[X_p]$ and $t_{p_5}[V_c V_p] = t_{p_1}[V_c V_p]$ (using (2), (3), IR3 or alternatively,(1), (4), IR3)

(f) $(R_a[A_{p_1}, \ldots, A_{p_{g'}}; X_p] \subseteq R_j[F_{p_1}, \ldots, F_{p_{g'}}; V_c V_p], t_{p_6})$, where $t_{p_6}[X_p; V_c V_p] = t_{p_5}[X_p; V_c V_p]$ (using (5), IR2)

(g) $(R_a[A_{p_1}, \ldots, A_{p_{g'}}; X_p] \subseteq R_j[F_{p_1}, \ldots, F_{p_{g'}}; Z'_p], t_{p_w})$, where $t_{p_w}[X_p] = t_{p_6}[X_p] = t_a[X_p]$ and $t_{p_w}[Z'_p] = t_{p_6}[Z'_p] = w[Z'_p]$ (using (6), IR6)

This verifies that Claim 1 holds on D_{i+1}. From this it follows that rules IR1–IR6 are complete for the implication of CINDs in the absence of finite-domain attributes. □

2. Proof of Theorem 3

Theorem 3. *The inference system $\mathcal{I}$ is sound and complete for the implication of CINDs in the general case, when finite-domain attributes may be present.*

Proof: The soundness of $\mathcal{I}$ can be verified by induction on the length of $\mathcal{I}$-proofs.

For the completeness of $\mathcal{I}$, consider a set $\Sigma \cup \{\psi\}$ of CINDs defined on a database schema $\mathcal{R}$. We show that if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}} \psi$. Assume that $\mathcal{R} = (R_1, \ldots, R_n)$, and that $\psi = (R_a[A_1, \ldots, A_m; X_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_{p_\psi})$.

The proof consists of five parts. (1) We first show that it suffices to consider a *special form* of CINDs. (2) We then develop a chase procedure for the special form of CINDs, and (3) show that the chase process always terminates. (4) In addition, we establish an important property of the chase procedure, and based on which (5) we show that if $\Sigma \models \psi$ then $\Sigma \vdash_{\mathcal{I}} \psi$ by using IR1–IR8.

(1) We first introduce the special form of CINDs. A CIND $(R_i[U; U_p] \subseteq R_j[V; V_p], t_p)$ is in the special form if (a) the attribute list U *only* consists of attributes with an infinite domain, and (b) the attribute list U_p contains *all* the finite-domain attributes of schema R_i.

As a result, the attribute list V also contains *only* infinite-domain attributes of the schema R_j. However, it is possible that (a) there is an infinite-domain attribute A of schema R_i such that $A \in U_p$, but (b) there is a finite-domain attribute B of schema R_j such that $B \notin V_p$.

It suffices to consider CINDs in this special form only. Indeed, given a CIND φ, we show that there exists a set Σ_φ of CINDs in the special form such that Σ_φ is equivalent to φ. Better still, Σ_φ can be proved from φ by using rules in $\mathcal{I}$ and vice versa, *i.e.,* $\{\varphi\} \vdash_{\mathcal{I}} \Sigma_\varphi$ and $\Sigma_\varphi \vdash_{\mathcal{I}} \varphi$.

First, let's consider CIND $\varphi = (R_i[U, A; U_p] \subseteq R_j[V, B; V_p], t_{p_\varphi})$, where attribute A has a finite-domain $\mathsf{dom}(A) = \{a_1, \ldots, a_k\}$. For each $l \in [1, k]$, we define a new CIND $\varphi_l = (R_i[U; A, U_p] \subseteq R_j[V; B, Y_p], t_{p_{\varphi_l}})$ such that $t_{p_{\varphi_l}}[A] = t_{p_{\varphi_l}}[B] = `a_l'$ and $t_{p_{\varphi_l}}[U_p \parallel V_p] = t_{p_\varphi}[U_p \parallel V_p]$. This is justified by IR4. Let Σ_φ be $\{\varphi_1, \ldots, \varphi_k\}$. It is easy to verify that $\{\varphi\} \equiv \Sigma_\varphi$, *i.e.,* $\Sigma_\varphi \models \{\varphi\}$ and $\{\varphi\} \models \Sigma_\varphi$.

Next, consider $\varphi = (R_i[U; U_p] \subseteq R_j[V; V_p], t_{p_\varphi})$, where there exists an attribute $A \in \mathsf{attr}(R_i) \setminus (U \cup U_p)$ with a finite-domain $\mathsf{dom}(A) = \{a_1, \ldots, a_k\}$. For each $l \in [1, k]$, we construct a CIND $\varphi_l = (R_i[U; A, U_p] \subseteq R_j[V; V_p], t_{p_{\varphi_l}})$ such that $t_{p_{\varphi_l}}[A] = `a_l'$ and $t_{p_{\varphi_l}}[U_p \parallel V_p] = t_{p_\varphi}[U_p \parallel V_p]$. This is justified by IR5. Let Σ_φ be $\{\varphi_1, \ldots, \varphi_k\}$. Then $\{\varphi\} \equiv \Sigma_\varphi$.

This shows how we can convert each CIND φ in $\Sigma \cup \{\psi\}$ into an equivalent set Σ_φ of CINDs in the special form. In addition, $\{\varphi\} \vdash_{\mathcal{I}} \Sigma_\varphi$ by successive applications of IR4 and IR5, and moreover, $\Sigma_\varphi \vdash_{\mathcal{I}} \varphi$ by successive applications of IR7 and IR8. Thus, we can assume *w.l.o.g.* that all the CINDs in $\Sigma \cup \{\psi\}$ are in the special

form.

(2) We now give the chase procedure for determining whether $\Sigma \models \psi$, which extends the one given in the proof of Theorem 2 to further deal with finite-domain attributes.

To deal with the interaction between finite domains and constant patterns in the CINDs, the chase process operates on trees as opposed to relations. In such a tree T, (a) $\mathsf{N}_{\mathsf{root}}$ is its root, (b) each node in T is labeled with a tuple t_j and its schema R_i, denoted by N = '$\mathsf{R_i : t_j}$', and (c) for each leaf node $\mathsf{N}_{\mathsf{leaf}}$ in T, the path from the root $\mathsf{N}_{\mathsf{root}}$ to $\mathsf{N}_{\mathsf{leaf}}$, denoted by PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}}$), encodes an instance of $\mathcal{R}$, such that for each relation schema R in $\mathcal{R}$, the set I_R of tuples of R carried by the nodes on the path is an instance of R.

We now give the detailed chase process.

The chase process starts with a tree T_0 consisting of only a root node $\mathsf{N}_{\mathsf{root}}$ = '$\mathsf{R_a : t_a}$', in which t_a is a tuple of schema R_a such that (a) $t_a[A_i] = v_i$ for $i \in [1, m]$, (b) $t_a[A] = t_{p_\psi}[A]$ for each attribute $A \in X_p$ of CIND ψ, and (c) $t_a[B] = v_0$ for each B in $\mathsf{attr}(R_a) \setminus (\{A_1, \ldots, A_m\} \cup X_p)$, where $v_0, v_1, \ldots, v_m$ are $m + 1$ distinct variables. Observe that for each attribute A in $\mathsf{attr}(R_a)$, if $t_a[A]$ is a variable v_i, then A must have an infinite domain by the definition of the special form of CINDs.

The chase process then repeatedly adds nodes to the tree T_0, a set of nodes at a time, by applying a chase operation APPLY$_f$. To specify APPLY$_f$, we need the following notion.

A CIND $\psi' = (R_i[U; U_p] \subseteq R_j[V; V_p], t_{p_{\psi'}})$ in Σ is said to be *applicable* to a node N = '$\mathsf{R_i : t_i}$' if (a) $t_i[U_p] = t_{p_{\psi'}}[U_p]$; (b) there exists no node N' = '$\mathsf{R_j : t_j}$' with $t_j[V] = t_i[U]$ and $t_j[V_p] = t_{p_{\psi'}}[V_p]$ on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}$); and (c) there exists at least a leaf node $\mathsf{N}_{\mathsf{leaf}}$ such that there is no such node N' on PATH($\mathsf{N}, \mathsf{N}_{\mathsf{leaf}}$). Intuitively, let D denote the database instance represented by PATH($\mathsf{N}, \mathsf{N}_{\mathsf{leaf}}$) on which N appears. Then $D \not\models \psi'$, and hence, we need to enforce ψ' on D.

Given a tree T and the CIND ψ', the chase operation APPLY$_f(T, \psi')$ transforms T into a new tree T' as follows.

- It traverses T starting from its root node $\mathsf{N}_{\mathsf{root}}$ in a breadth-first order, and checks whether there exists a node to which the CIND ψ' is applicable.
- If such a node N is found, then new nodes are added to T to make T', as follows.
  (a) Let $S = \{\mathsf{N}_{\mathsf{leaf}_1}, \ldots, \mathsf{N}_{\mathsf{leaf}_k}\}$ be the set of leaf nodes in T such that for each $i \in [1, k]$, PATH($\mathsf{N}, \mathsf{N}_{\mathsf{leaf}_i}$) exists and moreover, there exists no node N' = '$\mathsf{R_j : t_j}$' with $t_j[V] = t_i[U]$ and $t_j[V_p] = t_{p_{\psi'}}[V_p]$ on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}_i}$).
  (b) Let $\rho(V_f)$ denote an instantiation of the list V_f of all finite-domain attributes in $\mathsf{attr}(R_j) \setminus (V \cup V_p)$. That is, for each attribute $C \in V_f$, $\rho(C)$ is a data value drawn from $\mathsf{dom}(C)$.
  (c) For each possible instantiation $\rho(V_f)$, it generates a *new* node N'_ρ = '$\mathsf{R_j : t}_\rho$' such that $t_\rho[V] = t_i[U]$, $t_\rho[V_p] = t_{p_{\psi'}}[V_p]$, $t_\rho[V_f] = \rho(V_f)$, and $t_\rho[C] = v_0$ for all the other attributes C in $\mathsf{attr}(R_j)$. Observe that there are only a finite number of instantiations for the attribute list V_f, and for each attribute $C \in \mathsf{attr}(R_j)$, $t_\rho[C]$ is a constant if the attribute C has a finite domain.
  (d) For each leaf $\mathsf{N}_{\mathsf{leaf}_i}$ ($i \in [1, k]$) in S and each new node N'_ρ, it adds N'_ρ as a child of $\mathsf{N}_{\mathsf{leaf}_i}$. That is, ψ' is enforced on the database represented by PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}}$).
  Let T' denote the modified tree. Then the same process repeats starting from the root node of T'.
- If there are no nodes to which ψ' is applicable, the tree T remains unchanged, *i.e.,* $T' = \text{APPLY}_f(T, \psi') = T$.

The chase process stops if no CINDs in Σ

are applicable to any nodes in T, *i.e.,* T = $\text{APPLY}_f(T, \psi')$ for each ψ' in Σ.

Intuitively, the chase process augments tree T_0 and generates a sequence $T_0, T_1, \ldots, T_f$ of trees such that (a) for each $l \in [0, f-1]$, T_{l+1} := $\text{APPLY}_f(T_l, \psi')$ for some $\psi' \in \Sigma$, and all the nodes in tree T_l also appear in T_{l+1}, and (b) T_f = $\text{APPLY}_f(T_f, \phi)$ for each ϕ in Σ, *i.e.,* T_f is a fixpoint reached by APPLY_f. We refer to T_f as a *result* of the chase process with Σ and ψ, denoted by $\mathsf{Chase}(\Sigma, \psi)$. Note that there may exist multiple distinct resulting trees T_f, depending on the orders of CINDs in Σ used in the chase process. We use $\mathsf{Chase}(\Sigma, \psi)$ to denote an arbitrary fixpoint obtained by the chase, and study the properties of all such fixpoints.

Example 2: Consider the set $\Sigma \cup \{\psi\}$ of CINDs given in Example 3.4. We first transform each CIND into the special form. For instance, for the CIND ψ, we generate a set Σ_ψ = $\{\psi'_1, \psi'_2, \psi'_3, \psi'_4, \psi'_5, \psi'_6\}$ of CINDs, where

$\psi'_1 = (R_1[BC; AD] \subseteq R_2[EF; G], (a, d \parallel d))$,
$\psi'_2 = (R_1[BC; AD] \subseteq R_2[EF; G], (b, d \parallel d))$,
$\psi'_3 = (R_1[BC; AD] \subseteq R_2[EF; G], (c, d \parallel d))$,
$\psi'_4 = (R_1[BC; AD] \subseteq R_2[EF; G], (a, e \parallel e))$,
$\psi'_5 = (R_1[BC; AD] \subseteq R_2[EF; G], (b, e \parallel e))$,
$\psi'_6 = (R_1[BC; AD] \subseteq R_2[EF; G], (c, e \parallel e))$.

We show the chase process for Σ and ψ'_2 in Fig. 4, where (a) tree T_0 is the initial tree, (b) T_1 is derived by applying ψ_2 to T_0, (c) T_2 is the result of applying $\psi_{3,1}$ to T_1, and (d) T_3 is produced by applying $\psi_{3,2}$ to T_2. Here $\psi_{3,1}$ = $(R_3[IJ; KL] \subseteq R_2[EF; G], (k, g \parallel d, g))$, and $\psi_{3,2}$ = $(R_3[IJ; KL] \subseteq R_2[EF; G], (k, h \parallel d, g))$. These two CINDs are in the special form, derived from ψ_3 by using IR5.

Observe that (1) $\Sigma \models \psi'_2$, and (2) for each leaf $\mathsf{N}_{\mathsf{leaf}}$ in the result $T_3 = \mathsf{Chase}(\Sigma, \psi'_2)$, there exists a node N = 'R_2 : t' on $\text{PATH}(\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}})$ with $t[EF] = (v_1, v_2)$ and $t[G]$ = 'd'. □

(3) We next verify that the chase process always terminates.

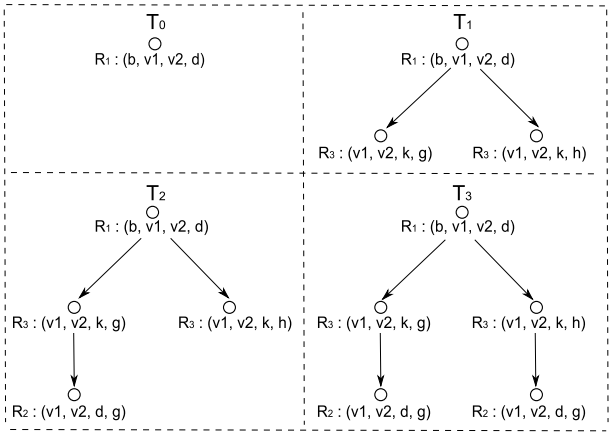

Figure 4: An example chasing process

Observe that in each tree T generated in the chase process, $\text{PATH}(\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}})$ from the root $\mathsf{N}_{\mathsf{root}}$ to each leaf $\mathsf{N}_{\mathsf{leaf}}$ of T represents a database instance D of $\mathcal{R}$. In addition, there exist no nodes N_1 and N_2 on the path such that they are labeled with the same tuple. Hence the depth of T is determined by the maximum instance of $\mathcal{R}$ constructed from the finite set $\{v_0, \ldots, v_m\}$ of variables, the finite set of constants appearing in the constant patterns in $\Sigma \cup \{\psi\}$, and all the constants in the finite domains of $\mathcal{R}$. That is, the depth of T is determined by $\mathcal{R}$ and $\Sigma \cup \{\psi\}$. Similarly, the maximum number of the children of a node in T is bounded by the maximum cardinality of finite domains, which is also determined by $\mathcal{R}$. Hence the size of T is bounded. There exist finitely many distinct trees that are constructed from those variables and constants with a bounded size. The chase process can generate at most finitely many such trees that are distinct, and hence, it must terminate.

(4) We next show a property of the chase procedure, which will be used to show that if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}} \psi$.

We first define an operator $\Upsilon(\mathsf{N})$, where N is a node in a tree T_l ($l \in [0, f]$) generated in the chase process. Given N = 'R_i : t_i', we define

$\Upsilon(\mathsf{N}) = (R_i[C_1, \ldots, C_m; U_p], t_i[U_p])$ if

- for each $j \in [1, m]$, $t_i[C_j] = v_j$, *i.e.,* $t_i[C_1, \ldots, C_m] = (v_1, \ldots, v_m)$; and
- the list U_p consists of all those attributes $C \in \mathsf{attr}(R_i)$ such that $t_i[C]$ is a constant;

whereas $\Upsilon(\mathsf{N})$ is *undefined* if there exist no attributes $C_1, \ldots, C_m$ in R_i such that $t_i[C_1, \ldots, C_m] = (v_1, \ldots, v_m)$.

Observe that when the CIND ψ is enforced, $\Upsilon(\mathsf{N})$ must be defined on some node. We shall use $\Upsilon(\mathsf{N})$ to inspect the existence of nodes satisfying the conditions specified by ψ.

The property is stated as follows.

Claim *2: Let φ be a CIND $(R_a[A_1, \ldots, A_m; X_p] \subseteq R_{b'}[D_1, \ldots, D_m; V_p], t_{p_\varphi})$, T_l be a tree generated in the chase process ($l \in [0, f]$), and $S = \{\mathsf{N}_{\mathsf{leaf}_1}, \ldots, \mathsf{N}_{\mathsf{leaf}_k}\}$ be the set of all the leaf nodes in T. Then $\Sigma \vdash_I \varphi$ if $\Sigma \vdash_I \varphi_j$ for each $j \in [1, k]$, where for a schema R_i in $\mathcal{R}$,*
(a) $\varphi_j = (R_i[C_{1_j}, \ldots, C_{m_j}; U_{p_j}] \subseteq R_{b'}[D_1, \ldots, D_m; V_p], t_{p_{\varphi_j}})$ with $t_{p_{\varphi_j}}[V_p] = t_{p_\varphi}[V_p]$, and
(b) there exists a node $\mathsf{N_j}$ = '$\mathsf{R_i : t_j}$' on $\mathrm{PATH}(\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf_j}})$ such that $\Upsilon(\mathsf{N_j}) = (R_i[C_{1_j}, \ldots, C_{m_j}; U_{p_j}], t_j[U_{p_j}])$ and $t_j[U_{p_j}] = t_{p_{\varphi_j}}[U_{p_j}]$.

Observe the following. (a) $\mathsf{LHS}(\varphi)$ is the same as $\mathsf{LHS}(\psi)$. (b) For each $j \in [1, k]$, $\mathsf{RHS}(\varphi_j)$ is the same as $\mathsf{RHS}(\varphi)$, and φ_j's have the same LHS. As will be seen in part (5) of the proof, we use these to prove $\Sigma \vdash_I \psi$.

We show the claim by induction on the length of the sequence of trees $T_0, T_1, \ldots, T_f$ generated by the chase process.

Base case. For the initial tree T_0, the only leaf node in T_0 is the root $\mathsf{N}_{\mathsf{root}}$ = '$\mathsf{R_a : t_a}$'. In this case, $\Upsilon(\mathsf{N}_{\mathsf{root}}) = (R_a[A_1, \ldots, A_m; X_p], t_{p_\psi}[X_p])$, and φ_{root} and φ are the same CIND $(R_a[A_1, \ldots, A_m; X_p] \subseteq R_a[A_1, \ldots, A_m; X_p], (t_{p_\psi}[X_p] \parallel t_{p_\psi}[X_p]))$. It is obvious that if $\Sigma \vdash_I \varphi_{root}$, then $\Sigma \vdash_I \varphi$.

Inductive case. Assume that Claim 2 holds for the first i trees $T_0, T_1, \ldots, T_i$. We show that it also holds on $T_{i+1} := \mathrm{APPLY}(T_i, \psi')$, *i.e.,* a result of applying some CIND ψ' in Σ to a node N in tree T_i. Assume *w.l.o.g.* that N = '$\mathsf{R_g : t_g}$', and that $\psi' = (R_g[U_1; U_{p_1}] \subseteq R_{g'}[U_2; U_{p_2}], t_{p_{\psi'}})$.

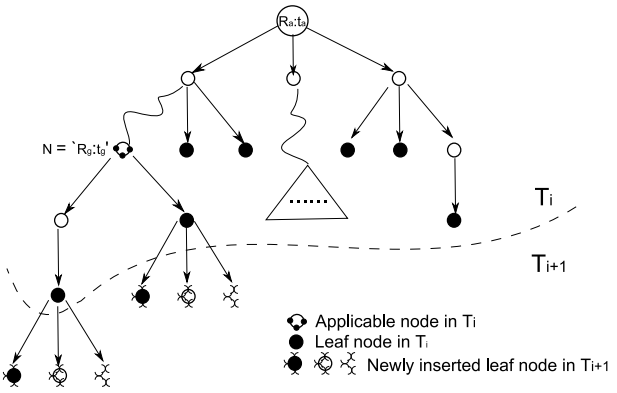

Figure 5: Example trees in the chase process

Here we consider two cases: (a) $T_{i+1} = T_i$ and (b) $T_{i+1} \neq T_i$. If $T_{i+1} = T_i$, the tree T_i is not changed by $\mathrm{APPLY}(T_i, \psi')$, and the claim obviously holds on T_{i+1} since it holds on T_i.

We next focus on the case where $T_{i+1} \neq T_i$. Recall the chase operation $\mathrm{APPLY}(T_i, \psi')$, by applying the CIND ψ' to the node N. Let $S_i = \{\mathsf{N}_{\mathsf{leaf}_1}, \ldots, \mathsf{N}_{\mathsf{leaf}_k}\}$ be the set of all leaf nodes in tree T_i, and let $S_{new} = \{\mathsf{N}_{\mathsf{f}_1}, \ldots, \mathsf{N}_{\mathsf{f}_h}\}$ be the set of newly generated nodes by applying the CIND ψ' to the node N. In T_{i+1}, all the nodes in S_{new} appear as the children of each leaf node of the sub-tree rooted at N in tree T_i.

To illustrate this, an example of T_i and T_{i+1} is shown in Fig. 5. In T_i, the sub-tree rooted at node N = '$\mathsf{R_g : t_g}$' has two leaf nodes. In T_{i+1}, three new nodes are added as the children of each of the two leaf nodes.

To simplify the discussion we assume *w.l.o.g.* that there is a single leaf node $\mathsf{N}_{\mathsf{leaf}_1}$ in the sub-tree rooted at node N; the proof for multiple such leaf nodes is similar. Thus, the set S_{i+1} of leaf nodes in tree T_{i+1} becomes

$S_i \cup S_{new} = \{\mathsf{N}_{\mathsf{f}_1}, \ldots, \mathsf{N}_{\mathsf{f}_h}, \mathsf{N}_{\mathsf{leaf}_2}, \ldots, \mathsf{N}_{\mathsf{leaf}_k}\}$.

We show that the claim holds on T_{i+1}, by considering the following cases.

Case 1. When the operator $\Upsilon(\mathsf{N})$ is undefined on the node N. Then for each node $\mathsf{N}_{\mathsf{f}_j}$ in S_{new} ($j \in [1,h]$), $\Upsilon(\mathsf{N}_{\mathsf{f}_j})$ is also undefined by the definition of APPLY$_f$, which generated those nodes in S_{new} to enforce ψ'. In this case, the claim holds on T_{i+1}. Indeed, those nodes N_j ($j \in [1, k+h-1]$) required by the claim are in the tree T_i, and so is N. Hence $\Sigma \vdash_{\mathcal{I}} \varphi$ since the claim holds on T_i by the induction hypothesis.

Case 2. When $\Upsilon(\mathsf{N})$ is defined on N. Consider $\Upsilon(\mathsf{N}) = (R_g[U'; U'_p],\ t_g[U'_p])$. Since the CIND ψ' is applicable to the node N, we can derive the following. (a) $U_{p_1} \subseteq U'_p$, (b) $t_g[U_{p_1}] = t_{p_{\psi'}}[U_{p_1}]$, (c) $t_g[U'] = (v_1, \ldots, v_m)$, and (d) $t_g[C] = v_0$ for each attribute C of $\mathsf{attr}(R_g)$ that is not in $U' \cup U'_p$.

We distinguish the following cases.

Case 2(a). $U' \not\subseteq U_1$, where U_1 is in $\mathsf{LHS}(\psi'_1)$. By $\Upsilon(\mathsf{N})$ we have that $t_g[U'] = (v_1, \ldots, v_m)$. Thus, if $U' \not\subseteq U_1$, Υ is not defined on those new nodes in S_{new}. Along the same lines as for Case 1 above, one can show that the claim holds on T_{i+1}.

Case 2(b). $U' \subseteq U_1$. We show that $\Sigma \vdash_{\mathcal{I}} \varphi$ if $\Sigma \vdash_{\mathcal{I}} \varphi_j$ for each $j \in [1, k+h-1]$, and for each leaf node $\mathsf{N}_{\mathsf{leaf}} \in S_{i+1}$, there exists a node N_j = '$\mathsf{R}_i : \mathsf{t}_j$' on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}}$) such that $\Upsilon(\mathsf{N}_j) = (R_i[C_{1_j}, \ldots, C_{m_j}; U_{p_j}], t_j[U_{p_j}])$ and $t_j[U_{p_j}] = t_{p_{\varphi_j}}[U_{p_j}]$. It suffices to show that we only need to consider those nodes N_j ($j \in [1, k+h-1]$) in T_i. For if this holds, then the same argument for Case 1 can verify that the claim holds on T_{i+1}. We show this by distinguishing the following cases.

(a) For each leaf node $\mathsf{N}_{\mathsf{leaf}_j}$ ($j \in [2,k]$), the node N_j on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}_j}$) must be in T_i since $\mathsf{N}_{\mathsf{leaf}_j}$ appears in T_i.

(b) For each new leaf nodes $\mathsf{N}_{\mathsf{f}_j}$ ($j \in [1,h]$) in S_{new}, there exist PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{f}_j}$) and PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}_1}$) in tree T_{i+1}, where there is an edge from $\mathsf{N}_{\mathsf{leaf}_1}$ to $\mathsf{N}_{\mathsf{f}_j}$. In this case, the leaf node $\mathsf{N}_{\mathsf{f}_j}$ is the only node appearing on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{f}_j}$), but not on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}_1}$). That is, if the node N_j ($j \in [1,h]$) on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{f}_j}$) is not in tree T_i, it must be the leaf node $\mathsf{N}_{\mathsf{f}_j}$ (see Fig. 5).

If there exists such a node N_j ($j \in [1,h]$) that is not in S_{new}, it must be on PATH($\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{f}_j}$) and hence, there exists PATH($\mathsf{N}_j, \mathsf{N}_{\mathsf{f}_j}$) for each $\mathsf{N}_{\mathsf{f}_j}$ in S_{new}. In this case, we only need to consider this N_j in the tree T_i.

If such a node N_j does not exist, we show that we can use the node N instead of N_j, where N is in T_i. In this case, for each $j \in [1,h]$, the node N_j must be the leaf node $\mathsf{N}_{\mathsf{f}_j}$, and the CIND φ_j must be in the form of $(R_{g'}[U'_2; U_{p_2}, U_f] \subseteq R_{b'}[D_1, \ldots, D_m; V_p], t_{p_{\varphi_j}})$ such that $t_{p_{\varphi_j}}[U_{p_2}] = t_{p_\phi}[U_{p_2}]$ and $t_{p_{\varphi_j}}[U_f] = \rho_j$. Here U_f is the list of all finite-domain attributes in $\mathsf{attr}(R_{g'}) \backslash (U' \cup U_{p_2})$, and $\rho_{U_f} = \{\rho_1, \ldots, \rho_h\}$ is the set of all possible instantiations of U_f.

We show that we can use N instead of $\mathsf{N}_{\mathsf{f}_j}$, and use a CIND φ_g derived below instead of φ_j for $j \in [1,h]$.

- Since $\Sigma \vdash_{\mathcal{I}} \varphi_j$ for each $j \in [1,h]$, we have that $\Sigma \vdash_{\mathcal{I}} \varphi_{g'}$, where $\varphi_{g'} = (R_{g'}[U'_2; U_{p_2}] \subseteq R_{b'}[D_1, \ldots, D_m; V_p], t_{p_{\varphi_{g'}}})$, and $t_{p_{\varphi_{g'}}}[U_{p_2} \parallel V_p] = t_{p_{\varphi_1}}[U_{p_2} \parallel V_p]$, by using IR7.
- By applying IR2 to the CIND ψ', we have that $\Sigma \vdash_{\mathcal{I}} \psi''$, where $\psi'' = (R_g[U'; U_{p_1}] \subseteq R_{g'}[U'_2; U_{p_2}], t_{p_{\psi''}})$, and $t_{p_{\psi''}} = t_{p_{\psi'}}$.
- By applying IR5 to the CIND ψ'', we have that $\Sigma \vdash_{\mathcal{I}} \psi'''$, where $\psi''' = (R_g[U'; U'_p] \subseteq R_{g'}[U'_2; U_{p_2}], t_{p_{\psi'''}})$, $t_{p_{\psi'''}}[U'_p] = t_g[U'_p]$, and $t_{p_{\psi'''}}[U_{p_2}] = t_{p_{\psi''}}[U_{p_2}]$.
- By applying IR3 to ψ''' and $\varphi_{g'}$, we can get that $\Sigma \vdash_{\mathcal{I}} \varphi_g$, where $\varphi_g = (R_g[U';\ U'_p] \subseteq R_{b'}[D_1, \ldots, D_m; V_p], t_{p_{\varphi_g}})$, $t_{p_{\varphi_g}}[U'_p] = t_{p_{\psi'''}}[U'_p] = t_g[U'_p]$ and $t_{p_{\varphi_g}}[V_p] = t_{p_{\varphi_{g'}}}[V_p]$.

Since $\Upsilon(\mathsf{N}) = (R_g[U'; U'_p], t_g[U'_p])$, we can use N and φ_g instead of $\mathsf{N}_{\mathsf{f}_j}$ and φ_j ($j \in [1, h]$), which still satisfy the conditions in the claim.

Hence we only need to use nodes in T_i, on which the claim holds based on the induction hypothesis.

(5) Finally, we show that if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}} \psi$, based on Claim 2. Let T_f be an arbitrary fixpoint obtained by the chase ($\mathsf{Chase}(\Sigma, \psi)$).

Recall that for each leaf $\mathsf{N}_{\mathsf{leaf}}$ of T_f, $\text{PATH}(\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}})$ represents a database instance D of $\mathcal{R}$. Observe that $D \models \psi$. Indeed, $D \models \Sigma$ since no CINDs in Σ are applicable to any nodes in T_f. By $\Sigma \models \psi$, we have that $D \models \psi$.

These tell us that for each leaf node $\mathsf{N}_{\mathsf{leaf}}$ in T_f, there must exist a node $\mathsf{N} = \text{`}\mathsf{R}_\mathsf{b} : \mathsf{t}_\mathsf{b}\text{'}$ on $\text{PATH}(\mathsf{N}_{\mathsf{root}}, \mathsf{N}_{\mathsf{leaf}})$ such that $t_b[B_1, \ldots, B_m] = (v_1, \ldots, v_m)$ and $t_b[Y_p] = t_{p_\psi}[Y_p]$. Here $\Upsilon(\mathsf{N}) = (R_b(B_1, \ldots, B_m; Y'_p), t[Y'_p])$, where $Y_p \subseteq Y'_p$. Hence for each $\mathsf{N}_{\mathsf{leaf}}$, we can verify the following using the inference system $\mathcal{I}$:

$\varphi_1 = (R_b[B_1, \ldots, B_m; Y'_p] \subseteq R_b[B_1, \ldots, B_m; Y'_p], t_{p_{\varphi_1}})$,
where $t_{p_{\varphi_1}}[Y'_{p_L}] = t_{p_{\varphi_1}}[Y'_{p_R}] = t_b[Y'_p]$ IR1

$\varphi_2 = (R_b[B_1, \ldots, B_m; Y'_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_{p_{\varphi_2}})$,
where $t_{p_{\varphi_2}}[Y'_p] = t_b[Y'_p]$ and $t_{p_{\varphi_2}}[Y_p] = t_b[Y_p]$
φ_1, IR6

That is, for each $\mathsf{N}_{\mathsf{leaf}}$, $\Sigma \vdash_{\mathcal{I}} \varphi_2$. Taking this together with the existence of $\mathsf{N} = \text{`}\mathsf{R}_\mathsf{b} : \mathsf{t}_\mathsf{b}\text{'}$, we have that $\Sigma \vdash_{\mathcal{I}} \psi$ by Claim 2. That is, if $\Sigma \models \psi$, then $\Sigma \vdash_{\mathcal{I}} \psi$.

This completes the proof for the completeness of $\mathcal{I}$ for CINDs when finite-domain attributes may be present. □

3. Proof of Theorem 5

Theorem 5. *The implication problem for CINDs is PSPACE-complete in the absence of attributes with finite domains.*

Proof: It is known that the implication problem for INDs is PSPACE-complete in the absence of finite-domain attributes [20]. Since CINDs subsume INDs, the implication problem for CINDs is also PSPACE-hard.

We next show that the implication problem for CINDs is in PSPACE in the absence of finite-domain attributes. We show this by giving a linear space non-deterministic algorithm for deciding whether $\Sigma \models \psi$, along the same lines as its counterpart for INDs (see [13, 20]). If this holds, then by Savitch's theorem [21], there is a deterministic quadratic-space algorithm for checking whether $\Sigma \models \psi$, and therefore, the implication problem is in PSPACE.

Indeed, the chase procedure developed in the proof of Theorem 2 gives such an algorithm. Consider a set $\Sigma \cup \{\psi\}$ of CINDs over a database schema $\mathcal{R} = (R_1, \ldots, R_n)$, where $\psi = (R_a[A_1, \ldots, A_m; X_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_{p_\psi})$. Recall that the chase process starts with an initial database D_0, which contains a single tuple $t_a \in I_a$ such that $t_a[A_i] = v_i$ for all $i \in [1, m]$ and $t_a[X_p] = t_{p_\psi}[X_p]$. As we have seen in the proof of Theorem 2, $\mathsf{Chase}(\Sigma, \psi) \models \psi$, where $\mathsf{Chase}(\Sigma, \psi)$ denotes an arbitrary fixpoint (database) generated by the chase process. Moreover, if $\Sigma \models \psi$, then there must exist a tuple $t_b \in I_b$ in $\mathsf{Chase}(\Sigma, \psi)$ such that $t_b[B_i] = v_i$ for $i \in [1, m]$, and $t_b[Y_p] = t_{p_\psi}[Y_p]$. More specifically, there exists a finite sequence $< t_0, \ldots, t_l >$ of tuples such that $t_0 = t_a$, $t_l = t_b$, and for each $i \in [0, l-1]$, t_{i+1} is obtained by applying a CIND ψ' in Σ to t_i.

Based on the analysis above, we develop a linear space non-deterministic algorithm:

- Initialize a single tuple $t_0 := t_a \in I_a$.
- Replace tuple t_i with another tuple t_{i+1} if t_{i+1} can be derived from t_i by applying a CIND ψ' in Σ to t_i by using rules IR1–IR6. There are possibly multiple such t_{i+1}'s. The algorithm nondeterministically picks one of them.
- Repeat these steps until no more changes can be made.

– If tuple $t_b \in I_b$, return 'yes'; and return 'no' otherwise.

As shown in the proof of Theorem 2, if $\Sigma \models \psi$, then t_b is in $\mathsf{Chase}(\Sigma, \psi)$ and hence, the algorithm returns 'yes'. Conversely, if the algorithm returns 'yes', *i.e.,* t_b is in $\mathsf{Chase}(\Sigma, \psi)$, then by Claim 1, $\Sigma \vdash_{\mathcal{I}(1\text{–}6)} \psi$. By Theorem 2, $\Sigma \models \psi$. Hence the algorithm correctly determines whether $\Sigma \models \psi$. The algorithm is obviously in PSPACE, as it only stores at most a single tuple at any time. As a result, the implication problem for CINDs is in PSPACE in the absence of finite-domain attributes. □

4. Proof of Theorem 7

Before we prove Theorem 7, we first examine the chase process introduced in the proof of Theorem 3. Given a set $\Sigma \cup \{\psi\}$ of CINDs, the chase procedure inspects whether $\Sigma \models \psi$. Below we give its computational complexity.

lemma 1. *Given a set $\Sigma \cup \{\psi\}$ of CINDs defined on a database schema $\mathcal{R}$, the chase procedure given in the proof of Theorem 3 terminates in $O(2^{2^{n^2}})$ time, where n is the size of the input,* i.e., *the size of $\mathcal{R}$, Σ and ψ.*

Proof: The chase procedure is obviously in $O(|T_f|)$ time, where T_f is a result $\mathsf{Chase}(\Sigma, \psi)$ of the chase process, and $|T_f|$ is the number of nodes in T_f. Recall that each root-to-leaf path of T_f represents a database of schema $\mathcal{R}$. Hence the depth of T_f is bounded by the maximum size $|I|$ of a database instance of $\mathcal{R}$. Moreover, the maximum number of children of a node in T_f is also bounded by $|I|$. We show that $|I|$ is in $O(2^{n^2})$ time as follows.

– The cardinality of a finite domain in $\mathcal{R}$ is determined by the schema $\mathcal{R}$ and is hence bounded by n.

– For an infinite domain in $\mathcal{R}$, the chase process uses only those constants appearing in the patterns in Σ or ψ, and the finite set $\{v_0, \ldots, v_m\}$ of variables (bounded by the size of ψ). These are also bounded by the input size n.

Hence the size $|I|$ is bounded by $O(n^n) = O(2^{log(n)*n}) \leq O(2^{n^2})$. And therefore $|T_f|$ is in $O((2^{n^2})^{2^{n^2}}) = O(2^{n^2 * 2^{n^2}}) = O(2^{2^{\log_2 n^2} * 2^{n^2}}) = O(2^{2^{n^2}})$ time. □

We are now ready to give the complexity bound for the implication analysis of CINDs in the general setting.

Theorem 7. *In the general setting, the implication problem for CINDs is EXPTIME-complete.*

Proof: (1) We first show that the problem is in EXPTIME. Given a set $\Sigma \cup \{\psi\}$ of CINDs on a relational schema $\mathcal{R}$, we develop an algorithm in $O(2^{n^k})$ time, where k is a constant and n is the size of $\mathcal{R}$, Σ and ψ, such that it returns 'yes' if and only if $\Sigma \models \psi$. Assume that $\mathcal{R} = (R_1, \ldots, R_n)$, and that $\psi = (R_a[A_1, \ldots, A_m; X_p] \subseteq R_b[B_1, \ldots, B_m; Y_p], t_{p_\psi})$.

Lemma 1 tells us that the chase procedure given in the proof of Theorem 3 cannot be directly used as such an algorithm, since it is doubly exponential. Nevertheless, we shall develop a singly exponential-time algorithm based on the chase procedure. Indeed, the complexity of the chase process is incurred by redundant nodes in the trees generated, as shown in Fig. 4. However, we can use graphs instead of trees to remove the redundancy.

Observe the following. Every node in a tree T is reachable from the root node '$R_a : t_a$'. In addition, if $\Sigma \models \psi$, then from each node in T there exists a path to a node N = '$R_b : t_b$' such that $t_b[B_1, \ldots, B_m] = t_a[A_1, \ldots, A_m]$ and $t_b[Y_p] = t_{p_\psi}[Y_p]$. One can check whether there exists a path from a node to another is solvable in quadratic time [29].

We now develop an EXPTIME algorithm based the chase procedure. The main idea is to

maintain a directed graph $G(V, E)$ and a mapping H. Given a node u in V of G, $H(\mathsf{u})$ is the set of CINDs in Σ that have already been applied to the node u in the chase process. With these two data structures, we can avoid *unnecessary* computations.

Below we first present the algorithm, and then verify the correctness of the algorithm. Finally, we show that the algorithm is in exponential time.

We first present the algorithm.

(a) It initializes the node set $V := \{\mathsf{N}_{\mathsf{root}}\}$, the edge set $E := \emptyset$, and $H(\mathsf{N}_{\mathsf{root}}) := \emptyset$. Here the node $\mathsf{N}_{\mathsf{root}}$ is the root node '$\mathsf{R}_{\mathsf{a}} : \mathsf{t}_{\mathsf{a}}$' of a tree T in the chase process.

(b) For each node u = '$\mathsf{R}_{\mathsf{i}} : \mathsf{t}_{\mathsf{i}}$' in V, it checks whether there exists a CIND $\psi' = (R_i[U; U_p] \subseteq R_j[V; V_p], t_{p_{\psi'}})$ in Σ, but not in $H(\mathsf{u})$, such that $t_i[U_p] = t_{p_{\psi'}}[U_p]$.

(c) If there exists such a CIND ψ' for the node u = '$\mathsf{R}_{\mathsf{i}} : \mathsf{t}_{\mathsf{i}}$', it first generates a set S_{new} of new nodes, and then updates the graph G and the mapping H accordingly.

The set S_{new} is generated along the same lines as the process in the proof of Theorem 3.

- Let V_f be the list of all finite-domain attributes in $\mathsf{attr}(R_j) \setminus (V \cup V_p)$, and $\rho(V_f)$ be an instantiation of V_f. That is, for each attribute $C \in V_f$, $\rho(C)$ is a data value drawn from the finite domain $\mathsf{dom}(C)$.
- For each possible instantiation $\rho(V_f)$, it generates a *new* node u'_ρ := '$\mathsf{R}_{\mathsf{j}} : \mathsf{t}_\rho$' such that $t_\rho[V] = t_i[U]$, $t_\rho[V_p] = t_{p_{\psi'}}[V_p]$, $t_\rho[V_f] = \rho(V_f)$, and $t_\rho[C] = v_0$ for all the other attributes C in $\mathsf{attr}(R_j)$.

Then, for the mapping H, $H(\mathsf{u}) := H(\mathsf{u}) \cup \{\psi'\}$, and for each node u'_ρ in S_{new}, but not in V, $H(\mathsf{u}'_\rho)$ is empty. For the graph $G(V, E)$, its node set V is updated to be $V \cup S_{new}$, and its edge set E is updated as follows. Let $S_{nbr} = \{\mathsf{u}_1, \ldots, \mathsf{u}_k\}$ be the set of neighboring nodes of the node u such that for each node u_l ($l \in [1, k]$), there is an edge $(\mathsf{u}, \mathsf{u}_\mathsf{l})$ in E.

- If S_{nbr} is empty, then for each node u'_ρ in S_{new}, it simply adds an edge $(\mathsf{u}, \mathsf{u}'_\rho)$ to E.
- Otherwise, for each node u_l ($l \in [1, k]$) in S_{nbr} and each node u'_ρ = '$\mathsf{R}_{\mathsf{j}} : \mathsf{t}_\rho$' in S_{new}, (a) if $R_j = R_b$, $t_\rho[B_1, \ldots, B_m] = (v_1, \ldots, v_m)$ and $t_\rho[Y_p] = t_{p_\psi}[Y_p]$, it replaces the edge $(\mathsf{u}, \mathsf{u}_\mathsf{l})$ with a new edge $(\mathsf{u}, \mathsf{u}'_\rho)$, and (b) if not, it replaces the edge $(\mathsf{u}, \mathsf{u}_\mathsf{l})$ with two new edges $(\mathsf{u}, \mathsf{u}'_\rho)$ and $(\mathsf{u}'_\rho, \mathsf{u}_\mathsf{l})$.

(d) The above process repeats until there are no more changes to the node set V and the edge set E of the graph G. We denote the final resulting graph as G_f.

(e) The algorithm finally checks whether $\Sigma \models \psi$ based on the graph G_f.

- Let S_a be the set of nodes in G_f that are reachable from the node $\mathsf{N}_{\mathsf{root}}$ = '$R_a : t_a$' such that $t_a[A_1, \ldots, A_m] = (v_1, \ldots, v_m)$ and $t_a[X_p] = t_{p_\psi}[X_p]$.
- Let S_b be the set of nodes '$R_b : t_b$' in G_f such that $t_b[B_1, \ldots, B_m] = (v_1, \ldots, v_m)$ and $t_b[Y_p] = t_{p_\psi}[Y_p]$.
- The algorithm checks if there exists a node u in S_a such that no nodes u' in S_b are reachable from u, *i.e.,* there exists no $\textsc{path}(\mathsf{u}, \mathsf{u}')$ in graph G_f.
  If there exists such a node, it returns 'no', and returns 'yes' otherwise.

We next show that $\Sigma \models \psi$ iff the algorithm returns 'yes'. Indeed, the algorithm simulates the chase procedure given in the proof of Theorem 3. If it returns 'no', one can readily expand G_f into a tree, which represents a database instance D (see the proof of Theorem 3) such that $D \models \Sigma$, but not $D \models \psi$, *i.e.,* $\Sigma \not\models \psi$. If it returns 'yes', then $\Sigma \models \psi$ by Claim 2 given in the proof of Theorem 3.

To see that the algorithm is in exponential time, observe the following. (a) The number of

nodes in the graph G_f is bounded by the maximum size $|I|$ of a database instance. Therefore, the size of G_f is bounded by $O(2^{n^2})$ as argued in the proof of Lemma 1. (b) For the set Σ of CINDs, the number of equivalent CINDs in the special form is bounded by $|I| = O(2^{n^2})$ as indicated in the proof of Theorem 3, and the number of CINDs equivalent to ψ in the special form is also bounded by $I = O(2^{n^2})$. (c) The size of S_a is bounded by the number of nodes in G_f, *i.e.,* in $O(2^{n^2})$; so is the size of S_b. From these it follows that graph G_f can be constructed in $O(2^{n^2} * 2^{n^2}) = O(2^{n^3})$ time, and the reachability between nodes in S_a and nodes in S_b can be checked in $O(2^{n^2} * 2^{n^2} * (2^{n^2})^2) = O(2^{n^3})$ time [29]. Based on these one can see that the algorithm is indeed in EXPTIME.

(2) We next show that the problem is EXPTIME-hard, by reduction from the two-player game of corridor tiling problem (TPG-CT), which is EXPTIME-complete [22, 23].

An instance of TPG-CT consists of a tiling system $(X, H, V, \vec{t}, \vec{b})$ and a positive integer n, where X is a finite set of tiles (dominoes), $H, V \subseteq X \times X$ are two binary relations, $\vec{t}$ and $\vec{b}$ are two n-vectors of given tiles in X, and n is the number of columns (the width of the corridor). It is to determine whether or not player I has a winning strategy for tiling the corridor. By tiling the corridor we mean that there exists a tiling $\tau : \mathbb{N} \times \mathbb{N} \to X$ and a positive integer m such that for all $x \in [1, n]$ and $y \in [1, m]$, the *tiling adjacency conditions* are observed, *i.e.,*

- if $\tau(x, y) = d$ and $\tau(x + 1, y) = d'$, then $(d, d') \in H$, *i.e.,* horizontally adjacent tiles have matching "colors";
- if $\tau(x, y) = d$ and $\tau(x, y + 1) = d'$, then $(d, d') \in V$, *i.e.,* vertically adjacent tiles have matching colors; and
- $\tau(x, 1) = \vec{t}[x]$ and $\tau(x, m) = \vec{b}[x]$, where $\vec{t}[x]$ (resp. $\vec{b}[x]$) denotes the x-th element of the vector $\vec{t}$ (resp. $\vec{b}$); that is, the given tiles of $\vec{t}$ and $\vec{b}$ are placed on the top and the bottom rows, respectively. The given tiles $\vec{t}$ are placed on the top row by the referee of the game.

Each player in turn places a tile from X in the first free location (column by column from left to right, and row by row from top to bottom), observing the tiling adjacency conditions. Player I wins if either Player II makes an illegal move by placing a tile that violates one of the adjacent conditions, or if the bottom row $\vec{b}$ is placed. Player I has a winning strategy iff Player I can always win no matter how Player II plays. The problem is already EXPTIME-complete when n is *odd* [22, 23], and thus below we assume that n is an *odd* number, and that Player I makes the first move.

Given an instance of TPG-CT $(X, H, V, \vec{t}, \vec{b})$ and n, we define a relational schema $\mathcal{R}$, a set of CINDs Σ and a CIND ψ such that $\Sigma \not\models \psi$ if and only if there is a winning strategy for Player I. If this holds, then the problem is EXPTIME-hard. Indeed, this problem is the complement problem of the implication problem, from which it follows that the implication problem for CINDs is also EXPTIME-hard.

(A) The database schema $\mathcal{R}$ consists of two relation schemas $R(K, A_0, A_1, \ldots, A_n, \mathsf{next}, P, Z)$ and $S(B)$, where for all $i \in [0, n]$, A_i has a finite domain $\mathsf{dom}(A_i) = X$, P has a finite domain $\mathsf{dom}(P) = \{1, \cdots, n\}$, the domains of attributes K and next are positive integers, and Z has a finite domain with two symbols # and !. The attribute B has a finite domain consisting of two distinct values: c and b.

Intuitively, an R tuple t encodes a placement of tiles in the game. More specifically, tuple t is the snapshot of the game showing the last $n + 1$ plays, where (a) $t[A_n]$ is the *new* tile placed by a player, (b) $t[A_0, \ldots, A_{n-1}]$ consists of the n tiles placed before $t[A_n]$, (c) $t[P]$ codes the horizontal position of tile $t[A_n]$ in a

I_R:

K	A_0	A_1	A_2	A_3	A_4	next	P	Z
k_1	a_1	a_2	a_3	a_4	b	k_2	1	#
k_2	a_2	a_3	a_4	b	c	k_3	2	#
k_3	a_3	a_4	b	c	d	k_4	3	#
k_4	a_4	b	c	d	e	k_5	4	#
k_5	b	c	d	e	f	k_6	1	#
...								
...								
k_x	x	b_1	b_2	b_3	b_4	k_1	4	#

I_S:

B
b

Figure 6: Encoding of a TPG-GT instance with $n = 4$, $\vec{t} = (a_1, a_2, a_3, a_4)$ and $\vec{b} = (b_1, b_2, b_3, b_4)$ for the proof of Theorem 7

row, (d) $t[K]$ and $t[\mathsf{next}]$ encode a list of such snapshots: $t[K]$ is the "identifier" of the current snapshot, while $t[\mathsf{next}]$ is a pointer to the next one (See Fig. 6). In addition, $t[Z]$ indicates that the game continues when it is #, and that the game should stop if it is !. An illegal move by Player I is indicated by the presence of a tuple s of schema S with $s[B] = `c'$.

We want to show that there exists an instance $D = (I_R, I_S)$ of $\mathcal{R}$ such that D satisfies Σ, but not ψ, if and only if Player I has a winning strategy.

(B) We next define the set Σ of CINDs that encodes the play. We use N_{odd} and N_{even} to denote the set of *even* numbers and the set of *odd* numbers in $[1, n]$, indicating the moves of Player I and Player II, respectively.

Initial condition. The top row of the corridor has to be set to $\vec{t}$. We use a CIND φ_1 to ensure that if I_S is nonempty, then there exists an R tuple t such that $t[A_0, \ldots, A_{n-1}]$ matches $\vec{t}$.

$$\varphi_1 = (S[\mathsf{nil}; \mathsf{nil}] \subseteq R[\mathsf{nil}; A_0, \ldots, A_{n-1}, P, Z],\ t_{p_{\varphi_1}}),$$
$$\text{where } t_{p_{\varphi_1}}[A_0, \ldots, A_{n-1}] = \vec{t} \text{ and } t_{p_{\varphi_1}}[P, Z] = (1, \#);$$

Adjacency constraints. The vertical and horizontal tiling conditions must hold. For each R tuple t, $t[A_n]$ corresponds to the tile directly under tile $t[A_0]$, and $t[A_n]$ is the tile placed next to $t[A_{n-1}]$ in a row. Thus for each tuple $t \in I_R$, we must ensure that $(t[A_0], t[A_n]) \in V$ and that $(t[A_{n-1}], t[A_n]) \in H$.

The adjacency constraints are enforced by two sets Σ_V and Σ_H of CINDs below. These CINDs assert the following: (a) if Player I makes an illegal move, then I_S contains a tuple ('c'); and (b) if any Player makes an illegal move, then the game should stop, by adding an R tuple t' with $t'[Z] =$!.

Σ_V: $(R[\mathsf{nil}; A_0, A_n, P] \subseteq S[\mathsf{nil}; B],\ t^v_{(x,y)})$,
where $t^v_{(x,y)}[A_0, A_n, h \parallel B] = (x, y, h \parallel c)$
for all $(x, y) \in (X \times X) \setminus V$ and all $h \in N_{odd}$

$(R[\mathsf{next}, A_1, \ldots, A_{n-1}; A_0, A_n, P] \subseteq R[K, A_0, \ldots, A_{n-2}; P, A_{n-1}, Z],\ t^v_{(x,y)})$, where $t^v_{(x,y)} = (x, y, h \parallel h + 1, y, !)$
for all $h \in [1, n-1]$ and all $(x, y) \in (X \times X) \setminus V$

Σ_H: $(R[\mathsf{nil}; A_{n-1}, A_n, P] \subseteq S[\mathsf{nil}; B],\ t^h_{(x,y)})$,
where $t^h_{(x,y)} = (x, y, h \parallel c)$ for all $(x, y) \in (X \times X) \setminus H$
and all $h \in N_{odd}$

$(R[\mathsf{next}, A_1, \ldots, A_{n-2}; A_{n-1}, A_n, P]$
$\subseteq R[K, A_0, \ldots, A_{n-3}; P, A_{n-2}, A_{n-1}, Z],\ t^v_{(x,y)})$,
where $t^v_{(x,y)} = (x, y, h \parallel h + 1, x, y, !)$ for all
$h \in [1, n-1]$ and all $(x, y) \in (X \times X) \setminus H$.

Player I has to respond to all possible legal moves of Player II. For each R tuple t_1 that is a legal move, if $t_1[P]$ is even, *i.e.,* if the last move $t_1[A_n]$ was made by Player II, then for each tile $x \in X$ that satisfies the horizontal constraint $(t_1[A_n], x) \in H$ and the vertical constraint $(t_1[A_0], x) \in V$, there must exist a tuple t_2 in I_R with $t_2[K] = t_1[\mathsf{next}]$, $t_2[A_{n-1}] = x$ and moreover, $t_2[A_i] = t_1[A_{i+1}]$ for $i \in [0, n-1]$. That is, all possible *legal* moves of Player II have to be considered. We encode this with a set $\Sigma_\forall$ of CINDs.

$\Sigma_\forall$: $(R[\mathsf{next}, A_1, A_2, \ldots, A_{n-2};\ P, A_0, A_{n-1}, A_n, Z]$
$\subseteq R[K, A_0, A_1, \ldots, A_{n-2};\ P, A_{n-1}, Z],\ t_{(x,y,w)})$,
where $t_{(x,y,w)} = (h, x, y, w, \# \parallel h + 1, w, \#)$, for all
$h \in N_{even}$, and for all $(x, w) \in V$ and $(y, w) \in H$

Play continues unless Player I has won. For each R tuple t_1, if $t_1[P] < n$ and $t_1[Z] = \#$, then there must exist some tuple t_2 such that $t_2[K] = t_1[\mathsf{next}]$, $t_2[A_0, \ldots, A_{n-1}] = t_1[A_1, \ldots, A_n]$ and

$t_2[P] = t_1[P] + 1$. If $t_1[P] = n$ and if the bottom vector $\vec{b}$ is not matched, *i.e.,* if for some $i \in [1, n]$, $t_1[A_i] \neq \vec{b}[i]$, then there must exist some t_2 such that $t_2[K] = t_1[\mathsf{next}]$, $t_2[A_0, \ldots, A_{n-1}] = t_1[A_1, \ldots, A_n]$ and $t_2[P] = 1$. We express this as a set Σ_p consisting of the following CINDs:

$\varphi_h = (R[\mathsf{next}, A_1, \ldots, A_n;\ P, Z] \subseteq R[K, A_0, \ldots, A_{n-1};\ P, Z],$ $t_{p_h})$, where for each $h \in [1, n-1]$, $t_{p_h} = (h, \# \parallel h+1, \#)$.

$\varphi_{(i,x)} = (R[\mathsf{next}, A_1, \ldots, A_i, A_{i+2}, \ldots, A_n;\ P, A_{i+1}, Z] \subseteq R[K,$ $A_0, \ldots, A_{i-1}, A_{i+1}, \ldots, A_{n-1};\ P, A_i, Z],\ t^n_{i,x})$, where $t^n_{(i,x)}$ $= (n, x, \# \parallel 1, x, \#)$, for all $i \in [1, n]$ and all $x \in X \setminus \{\vec{b}[i]\}$.

Observe that if an illegal move was made, a next move t_2 is added with $t_2[Z] =!$ and $t_2[P] = h + 1$, by Σ_H or Σ_V.

The set Σ consists of all the CINDs given above, *i.e.,* $\Sigma = \{\varphi_1\} \cup \Sigma_V \cup \Sigma_H \cup \Sigma_p \cup \Sigma_\forall$. The number of CINDs in Σ is bounded by a polynomial of n and the number of tiles in X.

(C) We define CIND $\psi = (S[\mathsf{nil}; \mathsf{nil}] \subseteq S[\mathsf{nil}; B], (\mathsf{nil} \parallel c))$. Intuitively, if $D \not\models \psi$ then (a) I_S is nonempty, and (b) there exists no tuple $t \in I_S$ such that $t[B] =$ 'c'. That is, Player I does not make illegal move.

The reduction is obviously in polynomial time. We next verify that Player I has a winning strategy iff $\Sigma \not\models \psi$.

First, suppose that $\Sigma \not\models \psi$. Then there exists an instance $D = (I_R, I_S)$ of $\mathcal{R}$ such that $D \models \Sigma$, but $D \not\models \psi$. We give a wining strategy for Player I. Player I begins with the tuple in R enforced by φ_1 in Σ. Such a tuple must exist because I_S must be nonempty (by $D \not\models \psi$) and hence, φ_1 is applicable. At any step in the game, there is a tuple in I_R that represents the last $n + 1$ moves of the play thus far. For each valid tile x_j that Player II places as the next move, represented by t, the CINDs in $\Sigma_\forall$ ensure the existence of a tuple t' with $t[\mathsf{next}] = t'[K]$ and $t'[A_{n-1}] = x_j$ and $t'[A_i] = t[A_{i+1}]$ for $i \in [0, n-1]$, *i.e.,* a response from Player I. By $D \not\models \psi$ and $D \models \Sigma_H \cup \Sigma_V \cup \Sigma_\forall$, the move t' satisfies both the horizontal and the vertical constraints, and it also correctly updates the last $n + 1$ tiles played. Furthermore, by $D \models \Sigma_p$, the play continues until Player I wins. Thus Player I has a winning strategy.

Conversely, suppose that Player I has a winning strategy. We then form an instance $D = (I_R, I_S)$ of $\mathcal{R}$ such that I_R consists of all valid plays in any game, where each tuple codes the horizontal position of the last move in a row and the last $n + 1$ tiles played in the game, and I_S has a single tuple t such that $t[B] =$'b' (*i.e.,* Player I makes no illegal move). It is easy to confirm that $D \models \Sigma$, but $D \not\models \psi$. □

5. Proof of Theorem 12

Theorem 12. *In the general setting, the implication problem for ACINDs is PSPACE-complete.*

Proof: (1) We first show that the problem for ACINDs is in PSPACE in the general setting.

We show this by giving a linear space non-deterministic algorithm for determining whether $\Sigma \not\models \psi$, *i.e.,* the complement of $\Sigma \models \psi$. This suffices. For if it holds, then (a) by Immerman–Szelepcsényi theorem [27, 28], there exists a linear space non-deterministic algorithm for determining whether $\Sigma \models \psi$; and (b) by Savitch's theorem [21], there is a deterministic quadratic-space algorithm for checking whether $\Sigma \models \psi$. From these it follows that the problem is in PSPACE.

Consider a set $\Sigma \cup \{\psi\}$ of acyclic CINDs defined over a database schema $\mathcal{R} = (R_1, \ldots, R_n)$. Assume *w.l.o.g.* that for any two schemas R_i and R_j $(i, j \in [1, n])$, there exist no CINDs in the form of $(R_i[U; U_p] \subseteq R_j[V; V_p], t_p)$ in Σ such that $j < i$. Observe that we can always rearrange the schemas in $\mathcal{R}$ to satisfy this condition since the CINDs in Σ are acyclic.

We also assume *w.l.o.g.* that the CIND ψ is $(R_1[A_1, \ldots, A_m; X_p] \subseteq R_n[B_1, \ldots, B_m; Y_p], t_{p_\psi})$; the proof is similar for the cases where ψ is from R_i to R_j when $i \neq 1$ or $j \neq n$.

The proof consists of three parts. (a) We first introduce notations to be used. (b) We then present the linear space non-deterministic algorithm. (c) Finally, we show that the algorithm is correct and that it runs in PSPACE.

(1.1) Before we present the algorithm, we first introduce the following notations to be used in the algorithm.

(a) Let $\{\Sigma_1, \ldots, \Sigma_{n-1}\}$ be the partition of Σ such that $\bigcup_{i=1}^{n-1} \Sigma_i = \Sigma$, and for each $i \in [1, n-1]$, Σ_i is the set of CINDs of the form $(R_i[U; U_p] \subseteq R_j[V; V_p], t_p)$ such that $j \in [i+1, n]$.

(b) The number of CINDs in Σ_i ($i \in [1, n-1]$) is denoted as n_i. We assume *w.l.o.g.* that for each $i \in [1, n-1]$, $n_i > 0$, *i.e.,* there exists at least one CIND in each Σ_i.

(c) All CINDs in Σ_i ($i \in [1, n-1]$) are sorted in an arbitrary order. We use a pointer p_i to indicate the p_i-th CIND in Σ_i. It is obvious that $1 \leq p_i \leq n_i$ for each $i \in [1, n-1]$.

(d) Given the list $P = [p_1, \ldots, p_{n-1}]$ of pointers, let $\Sigma_{P,\psi} = \{\varphi_1, \ldots, \varphi_{n-1}\}$ be the set of CINDs such that for each $i \in [1, n-1]$, φ_i is the p_i-th CIND in Σ_i.

(1.2) We now present the linear space non-deterministic algorithm for determining whether $\Sigma \not\models \psi$.

(a) It *guesses* an instantiation ρ_1 for the list X_f of all finite-domain attributes in $\mathsf{attr}(R_1) \setminus (X_p)$, where for each attribute C in X_f, $\rho_1(C)$ is a value drawn from the finite $\mathsf{dom}(C)$.

(b) It initializes a database instance $D := (I_1, \ldots, I_n)$, such that for each $i \in [1, n]$, I_i is an empty instance of schema R_i, except that I_1 contains a single tuple t_a of schema R_1, where $t_1[A_1, \ldots, A_m] = (v_1, \ldots, v_m)$, $t_1[X_p] = t_{p_\psi}[X_p]$, and $t_1[X_f] = \rho_1[X_f]$ for the list X_f of all finite-domain attributes in $\mathsf{attr}(R_1) \setminus (X_p)$. Here $v_1, \ldots, v_m$ are m distinct variables. Intuitively, t_1 encodes the LHS of the CIND ψ, and is to serve as a "witness" for $D \not\models \psi$. We assume *w.l.o.g.* that $X_f \cap \{A_1, \ldots, A_m\}$ is *empty* since if not, in the process to be seen shortly, we can simply replace the variable with the constant $\rho[A]$ for each attribute A in $X_f \cap \{A_1, \ldots, A_m\}$.

The algorithm will ensure that for each $i \in [1, n]$, the instance I_i contains at most one tuple, which guarantees that the algorithm uses only linear space.

(c) Starting with $p_i := 1$ for all pointers p_i in P ($i \in [1, n-1]$), the algorithm processes CINDs in $\Sigma_{P,\psi}$ one by one, as follows.

We set $i = 1$, and let ϕ be the p_i-th CIND $(R_i[U; U_p] \subseteq R_j[V; V_p], t_{p_\phi})$ in Σ_i.

- Guess an instantiation ρ_j for the list V_f of all finite-domain attributes in $\mathsf{attr}(R_j) \setminus (V \cup V_p)$, in which for each C in V_f, $\rho_j(C)$ is a value drawn from the finite $\mathsf{dom}(C)$.
- If there is a tuple t_i in I_i, but there exists no tuple t_j in I_j such that $t_j[V] = t_i[U]$, $t_j[V_p] = t_{p_\phi}[V_p]$, it first creates a new tuple t'_j such that $t'_j[V] = t_i[U]$, $t'_j[V_p] = t_{p_\phi}[V_p]$, and $t'_j[V_f] = \rho_j(V_f)$, and then updates the instance $I_j := \{t'_j\}$ to contain this new tuple.
- If $i < (n-1)$, it increases the variable i by 1, and repeats the process above.
- Otherwise, it checks whether there exists a tuple $t_n \in I_n$ such that $t_n[B_1, \ldots, B_m] = t_1[A_1, \ldots, A_m] = (v_1, \ldots, v_m)$ and $t_n[Y_p] = t_{p_\psi}[Y_p]$. Observe that the current database instance $D_P \models \Sigma_{P,\psi}$.

(d) If there exists a pointer p_j ($1 \leq j \leq n-1$) such that $p_j \neq n_j$, the algorithm then adjusts the list $P = [p_1, \ldots, p_{n-1}]$ of pointers by the following pseudo-codes.

```
let j := n − 1, and increase p_{n−1} by 1;
while (j > 1) do
    if p_j = (n_j + 1) then
```

let $p_j := 1$, and increase p_{j-1} by 1;
decrease the variable j by 1.

(e) If there exist no such pointers in P, *i.e.,* $p_j = n_j$ for all $j \in [1, n-1]$, the algorithm stops and returns 'yes' if the tuple t_n is not found in all the cases of the list P of pointers, *i.e.,* from $[1, \ldots, 1]$ to $[n_1, \ldots, n_{n-1}]$. Otherwise, the algorithm starts again from step (a).

(1.3) We next show that the algorithm is in PSPACE and that it is correct.

Observe that at any step of the process, the database instance D contains at most n tuples, where n is the number of relation schemas in $\mathcal{R}$. Hence, it is obvious that the nondeterministic algorithm runs in linear space.

To show the correctness of the algorithm, first observe the following.

(a) The algorithm examines each combination of those CINDs in all Σ_i's ($i \in [1, n-1]$), represented by the pointer list $P = [p_1, \ldots, p_{n-1}]$ of pointers. Recall that for each $i \in [1, n-1]$, the pointer p_i denotes the p_i-th CIND in Σ_i.

For acyclic CINDs Σ, this suffices for determining whether $\Sigma \models \psi$.

(b) The algorithm examines various instantiation of variables for finite-domain attributes, by backtracking. More specifically, it changes the list $P = [p_1, \ldots, p_{n-1}]$ of pointers starting from the last pointer p_{n-1}, and does not change p_i until $p_j \geq n_j$ for each $j > i$ (recall that for each $i \in [1, n-1]$, n_i is the number of CINDs in Σ_i). This allows us to avoid random valuations of finite-domain attributes and moreover, to use the same space in the entire process.

Having seen these, we finally show the correctness of the algorithm, *i.e.,* it returns 'yes' if and only if $\Sigma \not\models \psi$.

First assume that $\Sigma \not\models \psi$. Then there exists a database instance $D = (I_1, \ldots, I_n)$ of $\mathcal{R}$ such that $D \models \Sigma$ but $D \not\models \psi$. By the definition of ψ, there must exist a tuple t_1 in the instance I_1 of schema R_1 such that $t_1[X_p] = t_{p_\psi}[X_p]$, but there exists no tuple t_n in the instance I_n of schema R_n such that $t_n[Y] = t_1[X]$ and $t_n[Y_p] = t_{p_\psi}[Y_p]$. If we choose the instantiation $\rho_1(X_f) = t_1[X_f]$ at step (a), and the instantiations $\rho_j[V_f]$ at step (c) based on the instance D, then for each combination of the list P of pointers, there exists no tuple t_n in the instance I_n of schema R_n such that $t_n[Y] = t_1[X]$ and $t_n[Y_p] = t_{p_\psi}[Y_p]$. Thus the algorithm must stop and return 'yes'.

Conversely, assume that the algorithm returns 'yes'. We construct a nonempty database instance D of $\mathcal{R}$ such that $D \models \Sigma$ but $D \not\models \psi$. Let D be the union of all database instances D_P at step (c), where $D_P \models \Sigma_{P,\psi}$. Then as observed earlier, $D \models \Sigma$ and $D \not\models \psi$.

(2) We next show that the problem is PSPACE-hard by reduction from the Q3SAT problem, which is PSPACE-complete (cf. [29]).

An instance of Q3SAT is a first-order logic sentence $\theta = \forall x_1 \exists x_2 \forall x_3 \cdots Q_m x_m \phi$, where Q_m is $\forall$ if m is odd and it is $\exists$ if m is even; $\phi = C_1 \wedge \cdots \wedge C_n$ is an instance of the 3SAT problem in which all the variables are $x_1, \ldots, x_m$, and for each $j \in [1, n]$, the clause C_j is $y_{j_1} \vee y_{j_2} \vee y_{j_3}$ such that for $i \in [1, 3]$, y_{j_i} is either $x_{p_{ji}}$ or $\overline{x_{p_{ji}}}$ for $p_{ji} \in [1, m]$. Here we use $x_{p_{ji}}$ to denote the occurrence of a variable in the literal l_i of clause C_j. The Q3SAT problem is to decide whether θ is true.

Given an instance θ of Q3SAT, we construct an instance of the implication problem for acyclic CINDs, which consists of a database schema $\mathcal{R}$ with finite-domain attributes, a set Σ of *acyclic* CINDs defined on $\mathcal{R}$ and another CIND ψ on $\mathcal{R}$. We show that $\Sigma \not\models \psi$ if and only if θ is true. This suffices. For it holds, then it is also PSPACE-hard to decide whether $\Sigma \models \psi$.

(A) The database schema $\mathcal{R}$ consists of $m + 2$ relation schemas $R_0(A_1, \ldots, A_m)$, …, $R_m(A_1, \ldots, A_m)$, and $S(B)$. All the attributes in $\mathcal{R}$ have a finite domain $\{0, 1\}$. Intuitively, each R_i is

to encode a quantifier in θ, which is either $\forall$ or $\exists$, for each $i \in [1, m]$. In an instance I_m of schema R_m, each tuple $t[A_1, \ldots, A_m]$ is to carry a truth assignment of the variables $\{x_1, \ldots, x_m\}$ in θ. In addition, we shall use an instance of S to indicate whether θ is satisfied.

(B) The set Σ contains the following ACINDs.

– For each *odd* number $1 \leq i \leq m$, we define two CINDs $\psi_{i,0}$ and $\psi_{i,1}$ from R_{i-1} to R_i as follows:

$\psi_{i,0} = (R_{i-1}[A_1, \ldots, A_{i-1}; \mathsf{nil}] \subseteq R_i[A_1, \ldots, A_{i-1}; A_i], (\mathsf{nil} \parallel 0))$, and

$\psi_{i,1} = (R_{i-1}[A_1, \ldots, A_{i-1}; \mathsf{nil}] \subseteq R_i[A_1, \ldots, A_{i-1}; A_i], (\mathsf{nil} \parallel 1))$.

When $i = 1$, $R_{i-1}[A_1, \ldots, A_{i-1}]$ is $R_0[\mathsf{nil}]$. These CINDs together assert that for each tuple t in an R_{i-1} relation ($i \in \{1, 3, \ldots, m-1\}$), there exist two tuples t_0 and t_1 in the R_i relation such that $t_0[A_1, \ldots, A_{i-1}] = t_1[A_1, \ldots, A_{i-1}] = t[A_1, \ldots, A_{i-1}]$, while $t_0[A_i] = 0$ and $t_1[A_i] = 1$.
Intuitively, we encode a universal quantifier $\forall$ by using these CINDs.

– For each *even* number $1 < i \leq m$, we define a CIND ψ_i from R_{i-1} to R_i:

$\psi_i = (R_{i-1}[A_1, \ldots, A_{i-1}; \mathsf{nil}] \subseteq R_i[A_1, \ldots, A_{i-1}; \mathsf{nil}], (\mathsf{nil} \parallel \mathsf{nil}))$.

This CIND ensures that for each tuple t in an R_{i-1} relation ($i \in \{2, 4, \ldots, m-2\}$), there exists a tuples t' in the R_i relation such that $t'[A_1, \ldots, A_{i-1}] = t[A_1, \ldots, A_{i-1}]$ while $t'[A_i]$ is either 0 or 1.
Intuitively, we encode an existential quantifier $\exists$ by using such CINDs.

– For each clause $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ in the 3SAT instance ϕ, we define CIND $\psi_{S,j}$ from R_m to S:

$\psi_{S,j} = (R_m[\mathsf{nil}; A_{p_{j1}}, A_{p_{j2}}, A_{p_{j3}}] \subseteq S[\mathsf{nil}; B], t_{p_{\psi_{m,j}}})$,

where $t_{p_{\psi_{m,j}}}[B] = 0$, and for each $i \in [1, 3]$, $t_{p_{\psi_{S,j}}}[A_{p_{ji}}] = \xi_j(x_{p_{ji}})$. Here ξ_j is the unique truth assignment of the 3SAT instance ϕ that makes clause C_j false, and $\xi_j(x_{p_{ji}})$ is the truth value of variable $x_{p_{ji}}$ by treating true as 1 and false as 0.
Intuitively, these CINDs assure that for each tuple t in an R_m relation, $t(A_1, \ldots, A_m)$ denotes a truth assignment ξ for the 3SAT instance ϕ, such that for each $i \in [1, m]$, $\xi(x_i) = \mathsf{true}$ if $t[A_i] = 1$, and $\xi(x_i) = \mathsf{false}$ if $t[A_i] = 0$. If the truth assignment ξ makes ϕ false, then there exists a tuple t' in relation S such that $t'[B] = 0$.

(a) The CINDs from R_0 to R_1:
$\psi_{1,0} = (R_0[\mathsf{nil}; \mathsf{nil}] \subseteq R_1[\mathsf{nil}; A_1], (\mathsf{nil} \parallel 0))$,
$\psi_{1,1} = (R_0[\mathsf{nil}; \mathsf{nil}] \subseteq R_1[\mathsf{nil}; A_1], (\mathsf{nil} \parallel 1))$.

(b) The CIND from R_1 to R_2:
$\psi_2 = (R_1[A_1; \mathsf{nil}] \subseteq R_2[A_1; \mathsf{nil}], (\mathsf{nil} \parallel \mathsf{nil}))$.

(c) The CINDs from R_2 to R_3:
$\psi_{3,0} = (R_2[A_1, A_2; \mathsf{nil}] \subseteq R_3[A_1, A_2; A_3], (\mathsf{nil} \parallel 0))$,
$\psi_{3,1} = (R_2[A_1, A_2; \mathsf{nil}] \subseteq R_3[A_1, A_2; A_3], (\mathsf{nil} \parallel 1))$.

(d) The CIND from R_3 to R_4:
$\psi_4 = (R_3[A_1, A_2, A_3; \mathsf{nil}] \subseteq R_4[A_1, A_2, A_3; \mathsf{nil}], (\mathsf{nil} \parallel \mathsf{nil}))$.

(e) The CINDs from R_4 to S:
$\psi_{S,1} = (R_m[\mathsf{nil}; A_1, A_2, A_3] \subseteq S[\mathsf{nil}; B], (0, 0, 0 \parallel 0))$,
$\psi_{S,2} = (R_m[\mathsf{nil}; A_2, A_3, A_4] \subseteq S[\mathsf{nil}; B], (0, 1, 0 \parallel 0))$.

Figure 7: A (partial) example reduction for the proof of Theorem 12

The set Σ has no more than $2m + n$ of CINDs in total. Note that Σ is acyclic.

For example, consider the following instance of the Q3SAT problem: $\theta = \forall x_1 \exists x_2 \forall x_3 \exists x_4 (C_1 \wedge C_2)$, where $C_1 = x_1 \vee x_2 \vee x_3$, and $C_2 = x_2 \vee \bar{x_3} \vee x_4$. Then the set Σ for θ consists of 7 CINDs, as shown in Fig. 7.

(C) We define CIND $\psi = (R_0[\mathsf{nil}; \mathsf{nil}] \subseteq S[\mathsf{nil}; B], (\mathsf{nil} \parallel 0))$. It ensures that if the R_0 relation is not *empty*, then there exists a tuple t in relation S such that $t[B] = 0$.

We next show that the Q3SAT instance θ is true if and only if $\Sigma \not\models \psi$.

First, assume that θ is satisfiable. We define an instance D of $\mathcal{R}$ as follows. For $i \in [1, m]$, the instance I_i of R_i in D consists of all truth as-

signments for $x_1, \ldots, x_m$ that satisfy the 3SAT instance ϕ. The instance I_S of S in D consists of a single tuple s with $s[B] = 1$. One can readily verify that $D \models \Sigma$ but $D \not\models \psi$. Hence $\Sigma \not\models \psi$.

Conversely, assume that $\Sigma \not\models \psi$. Then there exists an instance D of $\mathcal{R}$ such that $D \models \Sigma$ but $D \not\models \psi$. By $D \not\models \psi$, the instance of I_0 of schema R_0 in D is nonempty, and hence so is the instance I_i of R_i in D for all $i \in [1, m]$, by $D \models \Sigma$. Observe that the instance I_m of R_m encodes truth assignments for $x_1, \ldots, x_m$. By $D \models \Sigma$, we know that I_{m-1} includes all the truth assignments required by the quantifiers in θ. By $D \not\models \psi$ again, the instance I_S of S in D does not have any tuple s with $s[B] = 0$. Hence by the definition of the CINDs $\psi_{m,j}$, none of those truth assignments in I_m violates the 3SAT instance ϕ. Therefore, θ is true. □

6. Proof of Theorem 13

Theorem 13. *The implication problem for UCINDs is in polynomial time in the absence of finite-domain attributes.*

Proof: It suffices to give a PTIME algorithm for checking whether $\Sigma \models \psi$ or not. Similar to the *upper bound* proof of Theorem 7, the algorithm converts the problem to the graph reachability problem, *i.e.,* checking whether there exists a path from a node to another in a graph. Recall that the graph reachability problem is solvable in quadratic time [29].

The proof consists of three parts. We first present the algorithm. We then show that the algorithm is correct. Finally, we show that the algorithm is in PTIME.

Consider a set $\Sigma \cup \{\psi\}$ of UCINDs over a database schema $\mathcal{R} = (R_1, \ldots, R_n)$, where the UCIND $\psi = (R_a[A; X_p] \subseteq R_b[B; Y_p], t_{p_\psi})$.

(1) The algorithm simulates the chase procedure given in the proof of Theorem 2, fine-tuned to leverage unary CINDs.

(a) The algorithm first builds a directed graph $G(V, E)$, based on which it checks whether $\Sigma \models \psi$. A node in graph G is in the form of $(R_i[C; U_p], t[U_p])$ $(1 \le i \le n)$ such that:
(i) R_i is a schema in $\mathcal{R}$,
(ii) C is a single attribute in $\mathsf{attr}(R_i)$,
(iii) U_p is a list of attributes in $\mathsf{attr}(R_i)$, and
(iv) $t[U_p]$ is a partial tuple of R_i defined on U_p.

The set V of nodes in G includes the following: (a) a single node $\mathsf{u_a} = (R_a[A; X_p], t_{p_\psi}[X_p])$, which corresponds to the LHS of the UCIND ψ; and (b) for each UCIND $\psi' = (R_i[C; U_p] \subseteq R_j[F; V_p], t_p)$ in Σ, a node $\mathsf{u} = (R_j[F; V_p], t_p[V_p])$, which denotes the RHS of ψ'.

The set E of edges contains a directed edge $(\mathsf{u_1}, \mathsf{u_2})$ for all nodes $\mathsf{u_1} = (R_i[C; U_p], t_i[U_p])$ and $\mathsf{u_2} = (R_j[F; V_p], t_j[V_p])$ in V if there exists a UCIND $(R_i[C; U'_p] \subseteq R_j[F; V'_p], t_p)$ in Σ such that $U'_p \subseteq Z_p$, $V_p \subseteq V'_p$, $t_p[U'_p] = t_i[U'_p]$, and $t_p[V_p] = t_j[V_p]$.

(b) The algorithm then checks whether $\Sigma \models \psi$, based on graph G.

Let S_b be the set of nodes that are of the form $\mathsf{u_b} = (R_b[B; U_p], t_b[U_p])$ in G such that $Y_p \subseteq Z_p$ and $t_b[Y_p] = t_{p_\psi}[Y_p]$. Recall that Y_p and $t_{p_\psi}[Y_p]$ are from the UCIND ψ.

The algorithm checks whether there exists a node $\mathsf{u_a}$ in the node set S_b such that there is a path from $\mathsf{u_a}$ to $\mathsf{u_b}$ in graph G (recall that $\mathsf{u_a}$ denotes the LHS of ψ). If there exists such u, it returns 'yes', and it returns 'no' otherwise.

(2) We now verify the correctness of the algorithm that returns 'yes' iff $\Sigma \models \psi$.

First assume that the algorithm returns 'yes'. Then there must exist a path from the node $\mathsf{u_a}$ to a node $\mathsf{u_b} = (R_b[B; U_p], t_b[U_p])$ in the graph G, where $Y_p \subseteq Z_p$ and $t_b[Y_p] = t_{p_\psi}[Y_p]$. Along the same lines as the proof of Theorem 2, one can construct a proof to show

that $\Sigma \vdash_{\mathcal{I}(1-6)} \psi$ based on IR1–IR6 in the inference system $\mathcal{I}$ (see Section. 3.2.1). By Theorem 2, IR1–IR6 are sound and complete for the implication analysis of CINDs in the absence of finite-domain attributes, by which we have $\Sigma \models \psi$.

Conversely, assume that the algorithm returns 'no'. We show that $\Sigma \not\models \psi$ by constructing a database instance D of $\mathcal{R}$ such that $D \models \Sigma$, but $D \not\models \psi$. The instance D is constructed step by step as follows.

(a) Initialize $D := \{I_1, \ldots, I_n\}$ such that $I_1 = \ldots = I_n = \emptyset$.

(b) Create a tuple t_a such that $t_a[A] = v$, $t_a[X_p] = t_{p_\psi}[X_p]$, and $t_a[A'] = v_0$ for all the other attributes A' in $\mathsf{attr}(R_a)$, and let $I_a = I_a \cup \{t_a\}$. Here v and v_0 are two distinct variables.

(c) For each node $\mathsf{u} = (R_i[C; U_p], t_i[U_p])$ in the graph G such that there exists a path from nodes $\mathsf{u_a}$ to u, construct a tuple t such that $t[C] = v$, $t[U_p] = t_i[U_p]$, and $t[C'] = v_0$ for all the other attributes C' in $\mathsf{attr}(R_i)$, and let $I_i = I_i \cup \{t\}$.

(d) Extend the instance D by using the chase procedure given in the proof of Theorem 2 until D reaches a fixpoint.

Then as argued in the proof of Theorem 2 about the chase procedure, one can verify that $D \models \Sigma$, but $D \not\models \psi$. Therefore, $\Sigma \not\models \psi$.

(3) We next show that the algorithm is in polynomial time.

It is obvious that the graph G can be built in polynomial time. Observe that the size of the set S_b is bounded by the number of nodes in the graph G, of which the size is bounded by a polynomial in the size of $\Sigma \cup \{\psi\}$. Based on these, it is easy to verify that the algorithm is indeed in PTIME.

Putting these together, we conclude that the implication problem for UCINDs is in PTIME, in the absence of finite-domain attributes. □

7. Proof of Theorem 15

Theorem 15. *The implication problem for UCINDs is coNP-complete in the general setting.*

Proof: (1) We first show that the problem is in coNP. Consider a set $\Sigma \cup \{\psi\}$ of UCINDs defined on a database schema $\mathcal{R} = (R_1, \ldots, R_n)$.

To show that the problem is in coNP, it suffices to give NP algorithms for checking whether $\Sigma \not\models \psi$. We first show that UCINDs can be transformed into two normal forms. We then present two NP algorithms based on the forms of ψ, and show that the algorithms are indeed correct.

(A) We show that UCINDs can be expressed in certain "normal" forms. Consider a UCIND $\varphi = (R_i[C; U_p] \subseteq R_j[F; V_p], t_{p_\varphi})$ such that the attribute C has a finite domain $\mathsf{dom}(C) = \{c_1, \ldots, c_k\}$. Let $\Sigma_\varphi = \{\varphi_1, \ldots, \varphi_k\}$, where for each $l \in [1, k]$, $\varphi_l = (R_i[\mathsf{nil}; C, U_p] \subseteq R_j[\mathsf{nil}; F, V_p], t_{p_{\varphi_l}})$ such that $t_{p_{\varphi_l}}[U_p \parallel V_p] = t_{p_\varphi}[U_p \parallel V_p]$ and $t_{p_{\varphi_l}}[C] = t_{p_{\varphi_l}}[F] = `c_l'$. It is easy to verify that $\Sigma_\varphi \equiv \{\varphi\}$, by IR4 and IR8 in the inference system $\mathcal{I}$ for CINDs (see Section. 3.2.1).

As a result, given a set Σ of UCINDs, we can derive an equivalent set Σ' of CINDs by using the transformations above.

Note that the number of CINDs in Σ' is bounded by a *polynomial* of the size of $\mathcal{R}$ and the number of UCINDs in Σ. In light of this, we can assume *w.l.o.g.* that all the UCINDs in $\Sigma \cup \{\psi\}$ are of one of the following forms:

- $(R_i[\mathsf{nil}; U_p] \subseteq R_j[\mathsf{nil}; V_p], t_p)$; and
- $(R_i[C; U_p] \subseteq R_j[F; V_p], t_p)$, where attributes C and F have an infinite domain.

Given a set $\Sigma \cup \{\psi\}$ of CINDs in these two forms, we develop two NP algorithms for checking whether $\Sigma \not\models \psi$, depending on the form of the CIND ψ. We use Σ_1 and Σ_2 to denote those CINDs in Σ in the first form and those in the second one, respectively, where $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$.

(B) We now provide an NP algorithm for the first case, where the CIND ψ is of the form $(R_a[\mathsf{nil}; X_p] \subseteq R_b[\mathsf{nil}; Y_p], t_{p_\psi})$.

To treat Σ_1 and Σ_2 uniformly, we further transform the CINDs of Σ_2 into CINDs in the first form. More specifically, for each CIND $\varphi = (R_i[C; U_p] \subseteq R_j[F; V_p], t_{p_\varphi})$ in Σ_2, we define a set Σ_φ of CINDs in the first form as follows.

- Let $\mathsf{adom} := \{a_1, \ldots, a_h\}$ be the set of constants appearing in either $\Sigma \cup \{\psi\}$ or in the finite domains of $\mathcal{R}$.
- Let Σ_φ to be the set $\{\varphi_0, \varphi_1, \ldots, \varphi_h\}$, where (a) $\varphi_0 = (R_i[\mathsf{nil}; U_p] \subseteq R_j[\mathsf{nil}; V_p], t_{p_\varphi})$, and (b) for each $l \in [1, h]$, $\varphi_l = (R_i[\mathsf{nil}; C, U_p] \subseteq R_j[\mathsf{nil}; F, V_p], t_{p_{\varphi_l}})$ such that $t_{p_{\varphi_l}}[U_p \parallel V_p] = t_{p_\varphi}[U_p \parallel V_p]$ and $t_{p_{\varphi_l}}[C] = t_{p_{\varphi_l}}[F] = `a_l'$.

Here CIND φ_0 is derived from φ by IR2, and the other CINDs in Σ_φ are derived from φ by IR4 in the inference system $\mathcal{I}$. Observe that the total number of CINDs in Σ_φ is bounded by a polynomial in the size of $\mathcal{R}$, Σ, and ψ.

Let Σ_2' be the union of Σ_φ's when φ ranges over all CINDs in Σ_2. We show that it suffices to consider CINDs in $\Sigma_1 \cup \Sigma_2'$. Indeed, a close examination of the chase procedure given in the proof of Theorem 3 tells us that the chase process for $\Sigma_1 \cup \Sigma_2 \cup \{\psi\}$ is equivalent to the one for $\Sigma_1 \cup \Sigma_2' \cup \{\psi\}$ since whenever a CIND in $\Sigma_1 \cup \Sigma_2$ is applied, we can use one in $\Sigma_1 \cup \Sigma_2'$ instead to reach the same result, and vice versa. Recall that the chase procedure determines whether $\Sigma \models \psi$ for CINDs in the general setting.

We now give the details of the NP algorithm, which is a non-deterministic extension of the PTIME algorithm given in the proof of Theorem 13, to handle finite-domain attributes. Given the set $\Sigma_1 \cup \Sigma_2' \cup \{\psi\}$ of CINDs, it extends the PTIME algorithm as follows.

- It adds an extra step after generating the node set V but before generating the edge set E. For each node $\mathsf{u} = (R_i[\mathsf{nil}; U_p], t[U_p])$ in V, it *guesses* an instantiation ρ_u for the list U_f of all finite-domain attributes in $\mathsf{attr}(R_i) \setminus (U_p)$ such that for each attribute $C' \in U_f$, $\rho_u(C')$ is a data value drawn from the finite $\mathsf{dom}(C')$. Note that the format of nodes is a little different from those used in the PTIME algorithm. However, this has no impact on the algorithm itself.
- The NP algorithm returns an answer *opposite* to that of the PTIME algorithm. That is, if there exists *no* node $\mathsf{u_b}$ in the set S_b such that there exists a path from the node $\mathsf{u_a}$ to $\mathsf{u_b}$ in the graph G, the algorithm returns 'yes', and it returns 'no' otherwise. Recall that $\mathsf{u_a}$ and $\mathsf{u_b}$ denote the LHS and the RHS of ψ, respectively. This is because the NP algorithm checks whether $\Sigma \not\models \psi$, while the PTIME algorithm checks whether $\Sigma \models \psi$.

It is easy to see that the algorithm is in NP.

We next show that algorithm returns 'yes' if and only if $\Sigma \not\models \psi$. Assume first that the NP algorithm returns 'yes'. Along the same lines as the argument for the 'no' answer of the PTIME algorithm given in the proof of Theorem 13, we can construct a nonempty instance D of $\mathcal{R}$ such that $D \models \Sigma$ but $D \not\models \psi$, *i.e.,* $\Sigma \not\models \psi$. Here when populating D, it suffices to randomly *guess* an instantiation for the finite-domain attributes of the new tuple to be inserted into D (the one shown in the NP algorithm will do).

Conversely, assume that $\Sigma \not\models \psi$. We show that there exists a set of instantiations for the node set V such that the algorithm returns 'yes'. Since $\Sigma \not\models \psi$, there exists a database instance $D = (I_1, \ldots, I_n)$ such that $D \models \Sigma$ and $D \not\models \psi$. That is, there exists a tuple $t_a \in I_a$ such that $t_a[X_p] = t_{p_\psi}[X_p]$, but there exists no tuple $t_b \in I_b$ such that $t_b[Y_p] = t_{p_\psi}[Y_p]$. The instantiations are defined as follows. For the node $\mathsf{u_a} = (R_a[\mathsf{nil}; X_p], t_{p_\psi}[X_p])$ in V, define an instantia-

tion ρ_{u_a} such that $\rho_{u_a}[X_f] = t_a[X_f]$ for the list X_f of all finite-domain attributes in $\mathsf{attr}(R_a) \setminus X_p$. For all the other nodes $\mathsf{u} = (R_i[\mathsf{nil}; U_p], t[U_p])$ in V, if there exists a tuple $t_u \in I_i$ such that $t_u[U_p] = t[U_p]$, define an instantiation ρ_u such that $\rho_u[U_f] = t_u[U_f]$ for the list U_f of all finite-domain attributes in $\mathsf{attr}(R_u) \setminus U_p$. Otherwise, let $\rho_u[U_f]$ be defined in terms of arbitrary values in the domains. The algorithm must return 'yes' for this specific graph G since there exist no tuples $t_b \in I_b$ such that $t_b[Y_p] = t_{p_\psi}[Y_p]$.

(C) We next present an NP algorithm for the second case where the CIND ψ is of the form $(R_a[A; X_p] \subseteq R_b[B; Y_p], t_{p_\psi})$.

This case is simpler. Indeed, the chase procedure given in the proof of Theorem 3 tells us that those CINDs in Σ_1 can be simply left out. In this case, the NP algorithm is the same as the one for the first case, except that only those CINDs in Σ_2 are considered when generating the graph G. Here the nodes have the same format as those used in the PTIME algorithm given in the proof of Theorem 13.

An argument similar to the one for the first case can verify that this NP algorithm returns 'yes' if and only if $\Sigma \not\models \psi$.

(2) We next show that the problem is coNP-hard by reduction from the 3SAT problem to the complement of the problem (*i.e.,* to decide whether $\Sigma \not\models \psi$). It is known that 3SAT is NP-complete (cf. [30]).

Consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, where $x_1, \ldots, x_m$ are all the variables in ϕ, and for each $j \in [1, n]$, C_j is of the form $y_{j_1} \vee y_{j_2} \vee y_{j_3}$ such that for $i \in [1, 3]$, y_{j_i} is either $x_{p_{ji}}$ or $\overline{x_{p_{ji}}}$ for $p_{ji} \in [1, m]$. Here $x_{p_{ji}}$ denotes the occurrence of a variable in the literal l_i of clause C_j. The 3SAT problem is to decide whether ϕ is satisfiable.

Given an instance ϕ of 3SAT, we construct an instance of the implication problem for unary CINDs, which consists of a database schema $\mathcal{R}$ with finite-domain attributes, and a set $\Sigma \cup \{\psi\}$ of UCINDs defined on $\mathcal{R}$. We show that $\Sigma \not\models \psi$ if and only if ϕ is satisfiable.

(A) The database schema $\mathcal{R}$ consists of two relation schemas $R(B, A_1, \ldots, A_m)$ and $S(B, C)$, where all attributes have a finite domain $\{0, 1\}$.

Intuitively, a tuple $t(A_1, \ldots, A_m)$ in an instance I_R of schema R denotes a truth assignment ξ_t of the 3SAT instance ϕ, such that for each $i \in [1, m]$ $\xi_t(x_i) = \mathsf{true}$ if $t[A_i] = 1$, and $\xi_t(x_i) = \mathsf{false}$ if $t[A_i] = 0$. As will be seen shortly, an instance I_S of schema S is used to indicate whether ϕ is satisfiable.

(B) The set Σ consists of n UCINDs given as follows. For each $j \in [1, n]$, let ξ_j be the *unique* truth assignment of the 3SAT instance ϕ that makes the clause $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ false. Then we define a UCIND $\varphi_j = (R[B; A_{p_{j1}}, A_{p_{j2}}, A_{p_{j3}}] \subseteq S[B; C], t_{p_{\varphi_j}})$, where $t_{p_{\varphi_j}}[C] = 0$, and for each $i \in [1, 3]$, $t_{p_{\varphi_j}}[A_{p_{ji}}] = 1$ if $\xi_j(x_{p_{ji}}) = \mathsf{true}$, and $t_{p_{\varphi_j}}[A_{p_{ji}}] = 0$ otherwise.

These UCINDs assert that for a tuple t in an R relation, if its carries a truth assignment ξ_t that makes ϕ false, then there must exist a tuple t' in the S relation such that $t'[B] = t[B]$ and $t'[C] = 0$. Observe that if there exists a database instance $D = (I_R, I_S)$ such that I_R is nonempty, I_S is empty, and $D \models \Sigma$, then the 3SAT instance ϕ must be satisfiable.

(C) The UCIND $\psi = (R[B; \mathsf{nil}] \subseteq S[B; C], (\mathsf{nil} \,||\, 0))$. It enforces that for each tuple t in relation I_R, there exists a tuple t' in relation I_S such that $t'[B] = t[B]$ and $t'[C] = 0$.

As an example, consider an instance $\phi = C_1 \wedge C_2 \wedge C_3$ of the 3SAT problem, where $C_1 = x_1 \vee x_2 \vee x_3$, $C_2 = x_2 \vee \bar{x_3} \vee x_4$ and $C_3 = x_3 \vee \bar{x_4} \vee \bar{x_5}$. The reduction for ϕ is shown in Fig. 8.

The reduction above is obviously in polynomial time.

(a) The database schema $\mathcal{R} = (R(B, A_1, \ldots, A_5), S(B, C))$.

(b) The set of UCINDs $\Sigma = \{\varphi_1, \varphi_2, \varphi_3\}$, where
$\varphi_1 = (R[B; A_1, A_2, A_3] \subseteq S[B; C], (0, 0, 0 \parallel 0))$,
$\varphi_2 = (R[B; A_2, A_3, A_4] \subseteq S[B; C], (0, 1, 0 \parallel 0))$, and
$\varphi_3 = (R[B; A_3, A_4, A_5] \subseteq S[B; C], (0, 1, 1 \parallel 0))$.

(c) The UCIND $\psi = (R[B; \mathsf{nil}] \subseteq S[B; C], (\mathsf{nil} \parallel 0))$.

Figure 8: An example reduction for the proof of Theorem 15

We next show that $\Sigma \not\models \psi$ if and only if the 3SAT instance ϕ is satisfiable.

We first assume that ϕ is satisfiable, and show that $\Sigma \not\models \psi$. It suffices to construct a database D such that $D \models \Sigma$ but $D \not\models \psi$. Since ϕ is satisfiable, there exists a truth assignment ξ that satisfies ϕ. Based on ξ, we define a tuple t on R such that (a) for each $i \in [1, m]$, $t[A_i] = 1$ if $\xi_t(x_i) = \mathsf{true}$, and $t[A_i] = 0$ if $\xi_t(x_i) = \mathsf{false}$, and (b) $t[B] = 1$. Let the instance $D = (I_R, I_S)$ such that $I_R = \{t\}$ and $I_S = \emptyset$. Then $D \models \Sigma$ but $D \not\models \psi$.

Conversely, we assume that $\Sigma \not\models \psi$, and show that ϕ is satisfiable. It suffices to find a truth assignment ξ that satisfies ϕ. Since $\Sigma \not\models \psi$, there exists a database instance $D = (I_R, I_S)$ such that $D \models \Sigma$ but $D \not\models \psi$. By $D \not\models \psi$, there is a tuple t in I_R but there exists no tuple t' in I_S with $t'[B] = t[B]$ and $t'[C] = 0$. Define a truth assignment ξ for ϕ such that for each $i \in [1, m]$, $\xi(x_i) = \mathsf{true}$ if $t[A_i] = 1$, and $\xi(x_i) = \mathsf{false}$ if $t[A_i] = 0$. Then ξ satisfies ϕ since otherwise, there must exist a tuple t' in I_S with $t'[B] = t[B]$ and $t'[C] = 0$ by the definition of those UCINDs in Σ. Hence ϕ is satisfiable. □