Temporal Data Exchange and Repair

by

Ladan Golshanara

A dissertation submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York

Doctor of Philosophy
(Department of Computer Science and Engineering)

August 9, 2018

Doctoral Committee:

Dr. Jan Chomicki, Chair
Dr. Bharat Jayaraman
Dr. Oliver Kennedy

To Maman, Baba, Nader and Davood

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my adviser Dr. Jan Chomicki for believing in me and for his relentless pursuit of precision and correctness. Also, for his patience and feedback. I would also like to thank Dr. Wang-Chiew Tan for her help and feedback on some initial work on the data exchange chapter.

I would like to thank Dr. Oliver Kennedy and Dr. Bharat Jayaraman for serving in the committee and for being always kind and encouraging. Finally, I would like to thank my labmates and friends at UB. Specially Gian Pietro for valuable discussions regarding some proofs in this work.

This research has been supported by NSF Grants IIS-1450560 and IIS-1524382.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Temporal Data Exchange and Repair

by

Ladan Golshanara

Committee:

Dr. Jan Chomicki, Chair

Dr. Bharat Jayaraman

Dr. Oliver Kennedy

Temporal data is needed by many organizations and individuals to support audit trails. With temporal data one can represent when a fact is true and for how long. The temporality of facts is also critical in diverse domains, from medical diagnosis to assessing the changing business conditions of companies to taxi and rental bicycle rides. To support temporal database applications suitable database features were recently added to the SQL:2011 standard, and adopted by major database management systems such as DB2, Oracle, and Teradata.

In today's digital world information about an entity can be found at different data sources at different times or in different versions of the same source. Despite the need for frameworks for temporal data curation tasks including temporal data cleaning, temporal data transformation, temporal data exchange and temporal data provenance, there are no principled frameworks in the literature. The goal of this research is to address the challenges that arise when one considers temporal data

in the frameworks of data exchange and repair. The approach and the results can be used in other temporal data curation tasks as well. The frameworks are based on a formal two-tier view of temporal data consisting of the *concrete view* and the *abstract view*. There are two models to show a temporal database in the abstract view: *snapshot model* and *timestamp model*. In the snapshot model, temporal data is shown as an infinite sequence of snapshots. In the timestamp model each fact in the database is associated with every time point in which it is true. There may be infinitely many such points because the database may extend arbitrarily far into the future. In the concrete view each fact is associated with the time interval(s) in which it is true. The concrete view provides efficiency in storing and manipulating data.

In the first part of this dissertation, we show how the framework of data exchange can be systematically extended to temporal data. We first extend the chase procedure for the abstract view using the snapshot model to have a conceptual basis for the data exchange for temporal databases. Considering non-temporal source-to-target tuple generating dependencies and equality generating dependencies, the chase algorithm can be applied on each snapshot independently. Then we define a chase procedure (called *c-chase*) on concrete instances and show the result of c-chase on a concrete instance is semantically aligned with the result of chase on the corresponding abstract instance. In order to interpret intervals as constants while checking if a dependency or a query is satisfied by a concrete database, we will *normalize* the instance with respect to the dependency or the query. To obtain the semantic alignment, the nulls (which are introduced by data exchange and model incompleteness) in the concrete view are annotated with temporal information. Furthermore, we show that the result of the concrete chase provides a foundation for query answering. We define naïve evaluation on the result of the c-chase and show it produces certain answers.

The second part of the dissertation aims to investigate temporal repair checking with respect to temporal functional dependencies. Same as temporal data exchange

framework, building on the relational notion of repair we have defined *concrete repairs*. We have also examined how a temporal repair checking algorithm can handle infinite abstract databases. We show how to reduce the problem of temporal repair checking on infinite temporal databases to the problem of temporal repair checking on finite temporal databases. We also develop two temporal repair construction algorithms: one is an extension of a relational repair construction algorithm while the other algorithm fragments the concrete instance to find the conflicts and leverages time intervals to build a repair. We have done some experiments on both temporal real-life data and synthetic data to compare the performance of our suggested algorithms for normalization and temporal repair construction. The results show that for real-life data, when the number of conflicts w.r.t. temporal functional dependencies is low (compared to the number of tuples in a relation), the algorithm that leverages time intervals is better.

CHAPTER 1

Introduction

Temporal data refers to historical data or data that is dated. Temporal data is needed by many organizations and individuals to support audit trails. With temporal data one can represent when a fact is true and for how long [15]. The temporality of facts is also critical in diverse domains, from medical diagnosis to assessing the changing business conditions of companies [33] to taxi and bicycle rides. To support temporal database applications suitable database features were recently added to the SQL:2011 standard [26], and adopted by major database management systems such as DB2, Oracle, and Teradata.

In today's digital world, information about an entity can often be found in multiple different data sources and in different versions of the same source at different times. Temporal data can also be extracted from unstructured data sources such as text and later be mapped and transformed into a desired format. Some of the temporal data curation tasks have already been addressed in the literature, such as temporal text extraction [32, 35, 40] and temporal entity resolution [8, 9, 28]. In this dissertation we address other critical tasks in the data curation pipeline for temporal data: (1) mapping and exchanging temporal data and (2) managing inconsistency and repairing.

Temporal databases provide a uniform and systematic way of dealing with histori-

cal data [13]. Prior work on temporal databases [24, 38, 13] has provided two views of temporal data: the *abstract temporal view* (or *abstract view* in short) and the *concrete temporal view* (or *concrete view* in short). Abstract view provides representation-independent meaning of a temporal database while concrete view provides a finite representation of temporal data. There are two dominant models for abstract view: *snapshot model* and *timestamp model*.

Snapshot model: Conceptually, we associate to each time point ℓ the state db_ℓ of a database at the time point ℓ. Thus, a temporal database in the abstract view in this model is a sequence of states (*snapshots*). The domain of time points is a totally ordered set which is isomorphic to non-negative integers $\mathbb{N}_0$. For example, consider a database schema $E(name, company)$ and the fact that Ada worked at IBM between time point 5 and 7 (inclusive). In the abstract view the snapshots of E associated with the time points 5 to 7 contain the fact $E(Ada, IBM)$.

Timestamp model: In this model, each relation is augmented with a temporal attribute. The domain of the temporal attribute consists of time points.

$$I_a = \{E(Ada, IBM, 5), E(Ada, IBM, 6), E(Ada, IBM, 7)\}$$

is an abstract database showing that Ada worked in IBM from time point 5 to time point 7.

Due to repetitive data in both of the snapshot model and the timestamp model, storing information in the abstract view is not practical and is meant only to provide the semantics for the concrete view. In the concrete view, temporal data is summarized in a single database instance in which data is time-stamped with a *time interval*[1] that indicates when the fact is true. The concrete view is an extension of the relational model where each relation in a database is augmented with a temporal attribute which takes time intervals as values. For example, in the concrete view the

[1]We assume time intervals have the format $[s, e)$, where $s, e \in \mathbb{N}_0$ and e can be ∞.

information above about *Ada* is usually represented as $E(Ada, IBM, [5, 8))$ where [5, 8) denotes the time points 5, 6 and 7. The fact that Ada has worked in Intel since then can be represented as E(Ada, Intel, [7, ∞)). An infinite time interval, such as [7, ∞), is a useful abstraction when the endpoint is not provided.

In the dissertation we use both views of temporal data. In chapter 3 we use the snapshot model because then we can adopt many notions in relational data exchange for defining the conceptual understanding of temporal data exchange. On the other hand, in chapter 4 we use the timestamp model because it helps to adopt relational repairing algorithms.

1.1 Data Exchange

Data exchange [16] refers to the problem of translating data that conforms to one schema (called the *source schema* R_S) into data that conforms to another schema (called the *target schema* R_T), given a specification of the relationship between the two schemas. This relationship is specified by means of a *schema mapping* consisting of a set of *source-to-target tuple generating dependencies (s-t tgds)* and a set of *tuple generating dependencies (tgds)* and *equality generating dependencies (egds)* on the target schema. Given a schema mapping and a source instance I, the goal of data exchange is to materialize a target instance J that satisfies the specification (i.e. (I, J) satisfies s-t tgds and J satisfies tgds and egds). Such an instance J is called a *solution* for I w.r.t the given schema mapping. For a given source instance, there may be no solution since there may not exist a target instance that satisfies the specification. On the other hand, there may be many solutions. It was shown in [16] that among all solutions of a given source instance, the *universal solutions* are the preferred solutions because they are the most general. In [16], the *chase procedure* is used to find a universal solution. Universal solutions can be used to determine *certain answers* to unions of conjunctive queries posed over a target schema. Certain

answers to a query Q are the tuples that are in the answer of Q in any solution for a source instance w.r.t a schema mapping.

1.2 Data Repairing

A database *db* is consistent w.r.t. a set of integrity constraints Σ defined on its schema if it satisfies all the constraints in Σ ($db \models \Sigma$). Otherwise, the database is inconsistent w.r.t. Σ. Two sources of inconsistency are data exchange and data integration where data from different autonomous sources are integrated. Managing inconsistency in databases has drawn the attention of many researchers for a few decades. There are two approaches to obtain consistent information from an inconsistent database [5, 11]: finding a *repair* (another database instance that satisfies the integrity constraints and *minimally* differs from the original one) and *consistent query answering (CQA)*(query answers that are true in every repair of a given database).

1.2.1 Repair Checking

Repair checking [11] is the problem of determining whether for two given instances one is a repair of the other. The complexity of this decision problem is investigated in [11, 3] for different classes of constraints. In case of functional dependencies, the problem of repair checking is in PTIME.

1.2.2 Repair Construction

Whether the repair checking problem is in PTIME or NP-Complete, there are many papers addressing repair construction by using different heuristics, sampling or using conditional functional dependencies and even the chase procedure as in LLUNATIC data cleaning framework [19].

In case of functional dependencies, a conflict is between two facts in the same relation that violate a functional dependency. To resolve a conflict with respect to

functional dependency, one of the tuples in the conflict is deleted. There might be multiple repairs for an inconsistent database.

1.3 Contributions and Outline

- **Data Exchange:** We first extend the relational data exchange framework to the abstract instances in the snapshot model. Moving to the concrete instances, we introduce the notions of *normalization* of a concrete instance w.r.t. a set of conjunctive formulas and *interval-annotated nulls* to model incompleteness introduced as a result of temporal data exchange. We also study query answering and certain answers on temporal databases and show the result of a concrete chase is enough for obtaining certain answers of a (non-temporal) query.

- **Repair Checking:** We study the notion of repair in temporal databases when the constraints are a set of *temporal functional dependencies (tFDs)*. Developing based on the timestamp model, we will show the problem of repair checking on infinite abstract instances can be reduced to the problem of repair checking on finite abstract instances when the constraints are temporal functional dependencies. The idea is to use a property of abstract instances that is called the *finite change condition* and the smallest time point that satisfies this property. Therefore, the complexity of temporal repair checking w.r.t. temporal functional dependencies is the same as repair checking on relational databases w.r.t. functional dependencies.

- **Repair Construction:** We propose two repair construction algorithms for temporal data: one is based on time-points the other based on time-intervals. Each has its own advantages and disadvantages. We have done some experiments to compare these algorithms.

Outline. Chapter 2 discusses some background on temporal databases. Chapter 3

studies temporal data exchange. Chapter 4 examines temporal data repair checking and construction. Chapter 5 shows our experiments on the proposed normalization algorithms as well as the repair algorithms. Chapter 6 describes the future research directions.

CHAPTER 2

Background and Preliminaries

This section provides background information on relational and temporal databases.

2.1 Relational Databases

We assume an infinite set Const of constants and an infinite set Null of labeled nulls that is disjoint from Const. A *(relational) database schema* $\mathcal{R}$ is a finite sequence $\langle R_1, ..., R_k \rangle$ of relation symbols, each with a fixed arity. Each relation symbol R_i is associated with a *relation schema* which is a set of attributes. A *database instance* over $\mathcal{R}$ is a sequence $\langle R_1^I, ..., R_k^I \rangle$, where each R_i^I is a finite relation of the same arity as R_i. We often use *database* or *instance* to refer to a database instance. We use R_i to denote both the relation symbol and the relation that interprets it.

Labeled nulls are distinguishable nulls that are used to model incompleteness, usually denoted by the symbols N and M (with or without subscript).

2.2 Temporal Databases

Snapshot model: Let $\mathcal{R}$ be a fixed relational database schema. An abstract database instance (abstract instance for short) I_a in the snapshot view is a infinite sequence of relational databases $\langle db_0, db_1, \ldots, \rangle$, where for every $\ell \in \mathbb{N}_0$ there is a relational database *(snapshot)* db_ℓ over $\mathcal{R}$.

Timestamp model: If $\mathcal{R}$ is a database schema, we denote by $\mathcal{R}^+$ the corresponding temporal database schema such that for each n-ary relation $R(A_1, \ldots, A_n)$ in $\mathbf{R}$ there is a $(n+1)$-ary temporal relation, denoted by $R^+(A_1, \ldots, A_n, T)$ in $\mathbf{R}^+$ where T is the temporal attribute and $A_1, \ldots A_n$ are data attributes. The domain of the temporal attribute consists of time points (i.e. a totally ordered set which is isomorphic to non-negative integers $\mathbb{N}_0$). We call $R^+(a_1, \ldots, a_n, \ell)$ where $\ell \in \mathbb{N}_0$ an *abstract fact*, usually denoted by f_a, f'_a, f''_a in this dissertation.

The timestamp model and the snapshot model are equivalent [12]. Denote by $f_a[\mathbf{D}]$ and $f_a[T]$ the data attributes and the temporal attribute of an abstract fact f_a respectively. In order to obtain the snapshot model of an abstract database I_a: For any timepoint $\ell \in \mathbb{N}_0$, if there is a fact $f_a \in I_a$ such that $f_a[T] = \ell$, then db_ℓ contains $f_a[\mathbf{D}]$. If there is no fact with time point ℓ, then $db_\ell = \emptyset$.

As an example consider the infinite abstract database

$$I'_a = \{E(Ada, IBM, 2), E(Ada, IBM, 3), ..., E(Ada, Google, 3), E(Ada, Google, 4), ...\}$$

The snapshot representation of I'_a is an infinite sequence of relational databases $\langle db_0, db_1, db_2, db_3, db_4, \ldots \rangle$, where

- $db_0 = db_1 = \emptyset$
- $db_2 : \{E(Ada, IBM)\}$
- $db_3 : \{E(Ada, IBM), E(Ada, Google)\}$
- $db_4 : \{E(Ada, IBM), E(Ada, Google)\}$
- $db_\ell = db_4$, $\ell > 4$

Concrete databases Since an abstract database can be infinite, we use a *concrete temporal database* (concrete database for short) as a space-efficient finite encoding of timestamps. The temporal database schema that is used for concrete databases is

the same as the schema of abstract databases in the timestamp model, that is $\mathcal{R}^+$. However, the domain of the temporal attribute in concrete databases is *time intervals*. Time intervals have the format $[s, e)$, where $s, e \in \mathbb{N}_0$ and e can be ∞.

In order to be able to represent an infinite abstract database with a finite concrete database, the abstract instance should satisfy the *finite change condition* [14]. The finite change condition intuitively says that after a time point, the abstract database is the same as before that timepoint. Even though this condition can be defined on both of the snapshot model and the timestamp model, it is more intuitive in the snapshot model. The finite change condition indicates that there exists $m \in \mathbb{N}_0$ such that $db_m = db_{m+1} = \ldots$.

A (single-dimensional) temporal database is *coalesced* if the facts with identical data attribute values have disjoint (i.e. no overlap) or non-adjacent time intervals [13, 7]. Two intervals $[s, e)$ and $[s', e')$ are adjacent if $s' = e$ (or $s = e'$). Any abstract database can be represented by a unique coalesced concrete database. This is only true if the temporal database has a single temporal attribute [13].

Example 1. Consider the following concrete databases:

- $I_c = \{E(Ada, IBM, [5, 8))\}$
- $I'_c = \{E(Ada, IBM, [5, 7)), E(Ada, IBM, [7, 8))\}$.

The concrete databases I_c and I'_c represents the same abstract database:

$$[\![I_c]\!] = [\![I'_c]\!] = \{E(Ada, IBM, 5), E(Ada, IBM, 6), E(Ada, IBM, 7)\}$$

Note that I_c is coalesced but I'_c is not. △

If a concrete (resp. abstract) database instance does not contain unknown information (nulls) we call it a *complete* concrete (resp. abstract) instance. If I_c is a

complete concrete instance, then we denote by $[\![I_c]\!]$ the abstract database instance that I_c represents:

- in the snapshot model, $[\![I_c]\!]$ is the sequence of snapshots $\langle db_0, db_1, \ldots \rangle$ such that for all $\ell \in \mathbb{N}_0$,

$$db_\ell = \{\ R(\mathbf{a}) \mid \exists s.\exists e.\ \ R^+(\mathbf{a}, [s, e)) \in I_c \text{ and } s \leq \ell < e\}$$

- in the timestamp model $[\![I_c]\!] = \{R^+(\mathbf{a}, \ell) \mid \exists s.\exists e\ R^+(\mathbf{a}, [s, e)) \in I_c \text{ and } s \leq \ell < e\}$

We abuse the symbol $[\![.]\!]$ for both snapshot models and timestamp models.

CHAPTER 3

Temporal Data Exchange

In this chapter whenever we mention abstract databases, we refer to the snapshot model of the abstract databases.

3.1 Preliminaries

We use the symbols I_c, J_c, I'_c, J'_c (resp. I_a, J_a, I'_a, J'_a) to refer to concrete instances (resp. abstract instances).

As in the classic data exchange [16, 4] we assume abstract source instances contain only constants (and in case of concrete instances, constants and time intervals). Thus, the abstract and concrete source instances are complete.

A non-temporal s-t tgd is of the form

$$\sigma_{st} : \forall \mathbf{x}\ \phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$$

and an egd is of the form

$$\sigma_{eg} : \forall \mathbf{x}\ \phi(\mathbf{x}) \rightarrow x_1 = x_2$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors of variables and x_1 and x_2 are variables in $\mathbf{x}$. In the rest of the paper we will usually drop universal quantification.

The s-t tgds and the egds on concrete schemas are augmented with a universally

quantified variable t in each atom in the left-hand-side (lhs for short) and right-hand-side (rhs for short) of the dependency:

$$\sigma_{st}^{+} : \forall \mathbf{x}, t\ \phi(\mathbf{x}, t) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y}, t)$$

$$\sigma_{eg}^{+} : \forall \mathbf{x}, t\ \phi(\mathbf{x}, t) \rightarrow x_1 = x_2$$

The domain (*sort*) of variable t is time intervals.

A *data exchange setting* is a quadruple $\mathcal{M} = (R_S, R_T, \Sigma_{st}, \Sigma_{eg})$ where R_S and R_T are the source and target schemas, respectively; Σ_{st} is a set of s-t tgds and Σ_{eg} is a set of egds. The source and the target schemas are disjoint. The corresponding *temporal setting* (for concrete databases) is $\mathcal{M}^{+} = (R_S^{+}, R_T^{+}, \Sigma_{st}^{+}, \Sigma_{eg}^{+})$.

We use the notation ϕ, σ_{st}, σ_{eg} for a conjunction of atomic formulas, a an s-t tgd and an egd, respectively. Anytime we refer to a *temporal conjunction of atomic formulas* (ϕ^{+}), a *temporal s-t tgd* (σ_{st}^{+}) or a *temporal egd* (σ_{eg}^{+}), we mean that each atom in ϕ, σ_{st} and respectively σ_{eg} is augmented with a variable t (w.l.o.g we assume variable t is the last variable in the atom).

In [16], the *chase procedure* is used to find a universal solution for a data exchange setting $\mathcal{M}$. A solution is universal if it has *homomorphisms* to every other solution. A *homomorphism* h from a relational instance J_1 to another instance J_2, denoted by $h : J_1 \rightarrow J_2$, is a function from the constants and labeled nulls in J_1 to constants and labeled nulls in J_2 such that:

- $h(a) = a$, where a is a constant in J_1.
- $h(N_0) = v$, where N_0 is a labeled null in J_1 and v is either a constant or a labeled null.
- for every $R(v_1, ..., v_n) \in J_1$, $R(h(v_1), ..., h(v_n))$ is in J_2.

A homomorphism h is also used for a mapping from a dependency (such as an s-

E

Name	Company
Ada	IBM
Ada	Google
Bob	IBM

Figure 3.1: A non-temporal source instance

t tgd or an egd) to an instance I such that for every atom $R(x_1, \ldots, x_m)$ in the dependency $R(h(x_1), \ldots, h(x_m))$ is a fact in I. We denote $h(x_1), \ldots, h(x_m)$ by $h(\mathbf{x})$, where $\mathbf{x} = x_1, \ldots, x_m$.

The standard chase modifies an instance by a sequence of *chase steps* until all dependencies are satisfied. A chase step is *fired* by a homomorphism and a dependency. If the dependency is a tgd, a chase step generates new facts in the target instance. Also, fresh labeled nulls are generated at each tgd chase step for each existentially quantified variable. If the dependency is an egd, then the chase step might be successful or not. If the chase step is successful, then some labeled nulls in facts are replaced by other labeled nulls or constants. If one constant is equated to another constant, the chase step fails. For a formal definition of the chase procedure refer to [16].

Example 2. Consider a data exchange setting where the source schema contains only one relation E and the target schema contains a relation Emp. Suppose there are three facts in E (shown in Figure 3.1). Suppose Σ_{st} contains only one s-t tgd:

$$E(n, c) \rightarrow \exists m \; Emp(n, c, m)$$

and $\Sigma_{eg} = \emptyset$. Initially the target instance is empty. By applying three chase steps, three new facts is generated in the target instance (shown in Figure 3.2). The chase procedure stops (after generating the three facts) because the source and target instances are satisfying the s-t tgd. $\triangle$

3.2 Abstract data exchange

In this section we extend the standard chase procedure to abstract instances. As mentioned in Section 3.1, the s-t tgds and egds we consider are same as the ones introduced in [16] which are over relational databases.

Consider a data exchange setting $\mathcal{M} = (R_S, R_T, \Sigma_{st}, \Sigma_{eg})$. Since the s-t tgds and egds are non-temporal, in order to apply the chase procedure on an abstract source instance I_a w.r.t. $\mathcal{M}$, we apply the chase procedure to each snapshot independently, that is

$$chase(I_a, \mathcal{M}) = \langle chase(db_0, \mathcal{M}), chase(db_1, \mathcal{M}), \ldots \rangle$$

The fresh labeled nulls that are produced in a snapshot are distinct from the labeled nulls produced in the other snapshots. Otherwise, it means that the same unknown value appears in different snapshots which is not intended by non-temporal s-t tgds and egds. In the Example 3 some snapshots of an abstract instance is shown.

If the result of at least one of the chase procedures on a snapshot is a failure, then the result of $chase(I_a, \mathcal{M})$ is a failure.

Example 3. Consider a source schema with two relations $E(name, company)$ and $S(name, salary)$. Some snapshots of the abstract view of the temporal database is shown in Figure 3.3.

We have the following non-temporal s-t tgds:

$$\forall n, c\ E(n, c) \rightarrow \exists s Emp(n, c, s)$$

Emp

Name	Company	Manager
Ada	IBM	N
Ada	Google	M
Bob	IBM	N'

Figure 3.2: A non-temporal target instance obtained by chase

I_a

2012	{E(Ada, IBM)}
2013	{E(Ada, IBM), S(Ada, 18k), E(Bob, IBM)}
2014	{E(Ada, Google), S(Ada, 18k), E(Bob, IBM)}
2014	{E(Ada, Google), S(Ada, 18k), E(Bob, IBM)}
2015	{E(Ada, Google), S(Ada, 18k), E(Bob, IBM), S(Bob, 13k)}
...	...
2018	{E(Ada, Google), S(Ada, 18k), S(Bob, 13k)}
...	...

Figure 3.3: Some snapshots in the abstract view of a temporal source instance.

$$\forall n, c, s \; E(n, c) \wedge S(n, s) \rightarrow Emp(n, c, s)$$

and the following egd:

$$\forall n, c, s, s' \; Emp(n, c, s) \wedge Emp(n, c, s') \rightarrow s = s'$$

$\triangle$

A target abstract instance J_a is a *solution* for a source instance I_a with respect to a data exchange setting $\mathcal{M}$ if each snapshot db_ℓ ($\ell \in \mathbb{N}_0$) in (I_a, J_a) is a solution, that is $db_\ell \models (\Sigma_{st} \cup \Sigma_{eg})$, where $\models$ represents the usual semantics of satisfaction of a first order logic formula.

Consider two abstract instances $I_a = \langle db_0, db_1, \ldots \rangle$ and $I'_a = \langle db'_0, db'_1, \ldots \rangle$. There exists a homomorphism h from I_a to I'_a (i.e. $h : I_a \mapsto I'_a$) if:

1. For every $\ell \in \mathbb{N}_0$ there is a homomorphism $h_\ell : db_\ell \mapsto db'_\ell$, $\ell \in \mathbb{N}_0$

2.

$$\forall i, j \in \mathbb{N}_0, \; i \neq j \text{ such that}$$

$$h_i : db_i \mapsto db'_i \text{ and } h_j : db_j \mapsto db'_j,$$

$$\forall \ell \in \mathbb{N}_0. \; \forall N \in \mathsf{Null}(db_\ell), \quad h_i(N) = h_j(N)$$

J_1

db_0	E(Ada, IBM, N)
db_1	E(Ada, IBM, N)

J_2

db'_0	E(Ada, IBM, M_1)
db'_1	E(Ada, IBM, M_2)

Figure 3.4: Two abstract instances with nulls

Example 4. Consider the target schema $Emp(name, company, salary)$. Two instances of the target schema are shown in Figure 3.4. In the instance J_1 the nulls in two consecutive snapshots are the same, representing one unknown value. Though from each snapshot in J_1 there is a homomorphism to the corresponding snapshot in J_2, that is $h_1 : db_0 \mapsto db'_0$ and $h_2 : db_1 \mapsto db'_1$, the homomorphisms do not agree on mapping N, that is $h_1(N) \neq h_2(N)$. △

In the Example 4, there is a homomorphism from the instance J_2 to J_1, but there is no homomorphism from J_1 to J_2.

Definition 3.1. *Universal solution:* A target instance $J_a = \langle db_0, db_1, \ldots \rangle$ is a *universal solution* for I_a with respect to a data exchange setting $\mathcal{M}$ if J_a is a solution and for an arbitrary solution $J'_a = \langle db'_0, db'_1, \ldots \rangle$, there exists a homomorphism $h : J_a \mapsto J'_a$.

Proposition 3.2. *Let $\mathcal{M} = (R_S, R_T, \Sigma_{st}, \Sigma_{eg})$ be a data exchange setting. Let I_a be an abstract source instance.*

1. *The result of a successful chase$(I_a, \mathcal{M})$ is a universal solution.*

2. *If the result of chase$(I_a, \mathcal{M})$ is a failure then there is no solution.*

Proof. ***Part 1:*** Let $J_a = \langle db_0, db_1, \ldots \rangle$ be the target instance obtained by chase. Let $J'_a = \langle db'_0, db'_1, \ldots \rangle$ be any solution for I_a with respect to $\mathcal{M}$. Based on Theorem 3.3 in [16], the result of a successful chase on each snapshot is a universal solution, meaning that there is a homomorphism h_ℓ from each snapshot db_ℓ in J_a to the corresponding snapshot db'_ℓ in J'_a, $\ell \in \mathbb{N}_0$. Each of the homomorphisms defined from a snapshot in J_a to the corresponding snapshot in J'_a is identity on constants. Now we need to show that these homomorphisms meet the second condition in the Definition 3.1.

J_a	
2012	{Emp(Ada, IBM, N)}
2013	{Emp(Ada, IBM, 18k), Emp(Bob, IBM, N')}
2014	{Emp(Ada, Google, 18k), Emp(Bob, IBM, N'')}
2014	{Emp(Ada, Google, 18k), Emp(Bob, IBM, M)}
2015	{Emp(Ada, Google, 18k), Emp(Bob, IBM, 13k)}
...	...
2018	{Emp(Ada, Google, 18k)}
...	...

Figure 3.5: Some snapshots of the abstract view of the result of $chase(I_a, \mathcal{M})$

The labeled nulls that are produced by the chase procedure in each snapshot in J_a are different from the labeled nulls in other snapshots, that is $\forall i \in \mathbb{N}_0.\ \forall j \in \mathbb{N}_0.\ (Null(db_i) \cap Null(db_j)) = \emptyset$. Therefore, the homomorphisms $h_0, h_1, \ldots$ can be extended in the following way:

$$h'_\ell(N) = \begin{cases} h_0(N) \text{ if } N \in \mathsf{Null}(db_0) \\ h_1(N) \text{ if } N \in \mathsf{Null}(db_1) \\ \ldots \\ h_\ell(N) \text{ if } N \in \mathsf{Null}(db_\ell) \\ \ldots \end{cases}$$

Hence, J_a is a universal solution.

Part 2: Let $I_a = \langle db''_0, db''_1, \ldots, \rangle$. If the result of $chase(I_a, \mathcal{M})$ is a failure, then it means for some $\ell \in \mathbb{N}_0$ the result of $chase(db''_\ell, \mathcal{M})$ is a failure, where db''_ℓ is a snapshot in I_a. Based on the Theorem 3.3 in paper [16], there is no solution for the snapshot db''_ℓ. Therefore there is no target instance J_a such that $(I_a, J_a) \models (\Sigma_{st} \cup \Sigma_{eg})$ (because Σ_{eg} is not satisfied in the ℓ^{th} snapshot of (I_a, J_a)). □

Example 5. The result of applying chase on each snapshot of I_a from Figure 3.3 is shown in Figure 3.5. △

E^+

Name	Company	Time
Ada	IBM	[2012, 2014)
Ada	Google	[2014, ∞)
Bob	IBM	[2013, 2018)

S^+

Name	Salary	Time
Ada	18k	[2013, ∞)
Bob	13k	[2015, ∞)

Figure 3.6: A concrete source instance I_c

3.3 Concrete data exchange

In this section, we define a chase algorithm called *c-chase* for a temporal data exchange setting $\mathcal{M}^+ = (R_S^+, R_T^+, \Sigma_{st}^+, \Sigma_{eg}^+)$ and a concrete source instance. Note that the dependencies in Σ_{st}^+ and Σ_{eg}^+ are implicitly non-temporal because they lack the expressive power to express the temporal phenomena such as an event happened *before* another event.

Example 6. The concrete view of the temporal database shown in Figure 3.3 is shown in Figure 3.6. The s-t tgds and the egd are as follows:

$$\sigma_1 : \forall n, c, t \; E^+(n, c, t) \rightarrow \exists s \; Emp^+(n, c, s, t)$$

$$\sigma_2 : \forall n, c, s, t \; E^+(n, c, t) \wedge S^+(n, s, t) \rightarrow Emp^+(n, c, s, t)$$

and the following egd:

$$\forall n, c, s, s', t \; Emp^+(n, c, s, t) \wedge Emp^+(n, c, s', t) \rightarrow s = s'$$

△

3.3.1 Interval-annotated nulls

The c-chase procedure produces a new type of unknown value for representing unknown values generated as a result of data exchange (that is, existentially quantified variables in the rhs of s-t tgds). The c-chase procedure cannot use labeled nulls

any more. We show the insufficiency of labeled nulls with an example. Consider the concrete fact $Emp(Ada, IBM, N, [0, 2))$, where N is a labeled null showing the salary of Ada is missing during the time interval $[0, 2)$. In the abstract view of this fact, the snapshots db_0 and db_1 contain the fact $Emp(Ada, IBM, N)$. The abstract view of this fact is shown in the Example 4. In Example 4 we have shown that we cannot define homomorphism from an instance in which the same labeled null appears in different snapshots to an instance that has different labeled nulls in each snapshot. The chase on the abstract view generates different labeled nulls in different snapshots. In order to be able to show that the result of the chase on the concrete view has correct semantics (defined by the chase on the abstract view), we introduce *interval-annotated nulls*. These nulls are annotated with the time interval of the concrete facts they occur in. For example, $N^{[s,e)}$ is an interval-annotated null in a concrete fact with the time interval $[s, e)$. The concrete fact $Emp(Ada, IBM, N^{[0,2)}, [0, 2))$ shows that not only the salary of Ada is missing in the time interval $[0, 2)$, but also that it *can be different* at snapshots db_0 and db_1. As another example, consider a concrete fact $Emp(Ada, IBM, N^{[8,\infty)}, [8, \infty))$. The interval-annotated null $N^{[8,\infty)}$ represents the sequence of labeled nulls $\langle N_8, N_9, \ldots\rangle$. In the abstract view, the snapshot db_8 contains the fact $Emp(Ada,\ IBM,\ N_8)$, the snapshot db_9 contains the fact $Emp(Ada,\ IBM,\ N_9)$ and so on.

An interval-annotated null is an expression $N^{[s,e)}$ where N is a label and $[s, e)$ is a time interval which is the *temporal context of* N. Each interval-annotated null $N^{[s,e)}$ (where $e \neq \infty$) corresponds to a finite sequence of distinct labeled nulls $\langle N_s, ..., N_{e-1}\rangle$. In case of $N^{[s,\infty)}$, the interval-annotated null corresponds to the infinite sequence $\langle N_s, N_{s+1}, ...\rangle$ of labeled nulls. In order to choose a labeled null in the sequence of nulls represented by $N^{[s,e)}$ we project on a time point, that is $\Pi_\ell(N^{[s,e)}) = N_\ell$, $s \leq \ell < e$. We denote by $\mathbf{N}^{[s,e)}$, a vector of interval-annotated nulls that occur in a concrete fact with the time interval of $[s, e)$.We extend $[\![.]\!]$ to instances with interval-

annotated nulls. Let I_c be a concrete instance, then $[\![I_c]\!]$ is a sequence of snapshots $\langle db_0, db_1, \ldots \rangle$ such that for all $\ell \in \mathbb{N}_0$:

$$db_\ell = \{\ R(\mathbf{a}, \mathbf{N}) \mid \exists s. \exists e.\ \ R^+(\mathbf{a}, \mathbf{N}^{[s,e)}, [s, e)) \in I_c \text{ and } s \leq \ell < e\}$$

3.3.2 Normalization

In a concrete source instance we have the temporal attribute with time intervals as values. Chase steps use homomorphisms from the lhs of a dependency to an instance to translate data. Informally, we would like to be able to define a homomorphism from a conjunction of atomic formulas $\phi^+(\mathbf{x}, t)$ to a concrete instance I_c whenever there are homomorphisms from $\phi(\mathbf{x})$ to $[\![I_c]\!]$. As an example, suppose we are trying to define a homomorphism from the lhs of σ_2 (in Example 6) to the constants and time intervals in the instance shown in Figure 3.6:

$$h : \{n \mapsto Ada, c \mapsto IBM, s \mapsto 18k, t \mapsto ?\}$$

One cannot map the variable t to a single time interval $h(t)$ such that $E^+(h(n), h(c), h(t))$ and $S^+(h(n), h(s), h(t))$ are some concrete facts in the instance I_c shown in Figure 3.6. In fact no homomorphism can be defined from the lhs of σ_2 to I_c. However, if we consider the abstract view of the same data (shown in Figure 3.3), many homomorphisms can be defined from $E(n, c) \wedge S(n, s)$ to $[\![I_c]\!]$ including:

$$h' : \{n \mapsto Ada, c \mapsto IBM, s \mapsto 18k\}$$

from $E(n, c) \wedge S(n, s)$ to the snapshot associated with time point 2013. We would like to have a concrete instance I_c with the following property:

Definition 3.3. Normalization Property: Let I_c be a concrete instance and Φ^+ be a set of temporal conjunctions respectively. Obtain the corresponding set of con-

junction Φ on schema of snapshots in $[\![I_c]\!]$. The instance I_c has the normalization property w.r.t. Φ^+ when both of the following items hold.

- **Condition 1** $\forall \phi \in \Phi\ \forall \ell \in \mathbb{N}_0$, if $h_\ell : \phi(\mathbf{x}) \mapsto db_\ell$ ($db_\ell \in [\![I_c]\!]$), then there is a homomorphism h from the conjunction of atomic formulas $\phi^+(\mathbf{x}, t) \in \Phi^+$ to I_c such that $\ell \in h(t)$ and h and h_ℓ map the same variable $x \in \mathbf{x}$ to the same constant (that is $\forall x \in \mathbf{x}.\ h(x) = h_\ell(x)$ if $h_\ell(x) = a$).

- **Condition 2** $\forall \phi^+ \in \Phi^+$ if $h : \phi^+(\mathbf{x}, t) \mapsto I_c$ where $h(t) = [s, e)$, then there is a set of homomorphisms $h_s, \ldots, h_{e-1}$ from $\phi(\mathbf{x})$ to consecutive snapshots $db_s, \ldots, db_{e-1}$ such that:

  - $h_s :\ \phi(\mathbf{x}) \mapsto db_s$,
  - $h_{s+1} :\ \phi(\mathbf{x}) \mapsto db_{s+1}$,
  - $\ldots$,
  - $h_{e-1} : \phi(\mathbf{x}) \mapsto db_{e-1}$,
  - $\forall x \in \mathbf{x}$ if $h(x) = a,\ a \in \mathsf{Const}$, then $\forall j \in \{s, \ldots, e-1\}\ h_j(x) = a$.

A concrete instance is *normalized* with respect to a set of temporal conjunctions Φ^+ if it has the normalization property w.r.t. Φ^+. In a normalized concrete instance the time intervals behave as constants (as shown in the Example 7).

Example 7. The instance I'_c shown in Figure 3.7 is normalized (by fragmenting the concrete facts in I_c) with respect to $E^+(n, c, t) \wedge S^+(n, s, t)$ (i.e. the lhs of σ_2). For example, there is a homomorphism h from $E^+(n, c, t) \wedge\ S^+(n, s, t)$ to the concrete instance such that

$$h = \{n \mapsto Ada, c \mapsto Google, s \mapsto 18k, t \mapsto [2014, \infty)\}.$$

Since I'_c is normalized, there are infinitely many homomorphisms $h_\ell, \ell \geq 2014$ from

E^+

Name	Company	Time
Ada	IBM	[2012, 2013)
Ada	IBM	[2013, 2014)
Ada	Google	[2014, ∞)
Bob	IBM	[2013, 2015)
Bob	IBM	[2015, 2018)

S^+

Name	Salary	Time
Ada	18k	[2013, 2014)
Ada	18k	[2014, ∞)
Bob	13k	[2015, 2018)
Bob	13k	[2018, ∞)

Figure 3.7: A normalized concrete source instance I'_c w.r.t. $E^+(n, c, t) \wedge S^+(n, s, t)$

$E(n, c) \wedge S(n, s)$ to snapshots $db_\ell \in [\![I'_c]\!]$ such that:

$$h_\ell(n) = h(n),\ h_\ell(c) = h(c) \text{ and } h_\ell(s) = h(s),\ \ell \in h(t)$$

Also, consider the homomorphism h'_ℓ ($\ell = 2013$) to snapshot db_ℓ:

$$h'_\ell = \{n \mapsto Ada, c \mapsto IBM, s \mapsto 18k\}$$

Since I'_c is normalized, there is a homomorphism h' from $E^+(n, c, t) \wedge\ \ S^+(n, s, t)$ to I_c such that $2013 \in h'(t) = [2013, 2014)$ and $h'(n) = h'_\ell(n)$, $h'(s) = h'_\ell(s)$ and $h'(c) = h'_\ell(c)$. $\triangle$

In the rest of this section, we discuss how to obtain a normalized instance with respect to conjunctions of atomic formulas. Note that the lhs of s-t tgds and egds (discarding the quantification) is conjunctions of atomic formulas.

Let Φ^+ be a set of temporal conjunctions of the form $\phi^+(\mathbf{x}, t)$. Denote by $|\phi|$ the number of atoms that are in ϕ. We denote by $\mathcal{N}(\Phi^+)$ the normalized form of Φ^+ such that for each formula $\phi^+ \in \Phi^+$ each occurrence of the variable t in ϕ^+ is replaced with a new variable t' in $\mathcal{N}(\Phi^+)$.

Let Φ^+ be temporal conjunctions of atomic formulas. Let I_c be a concrete instance and $\{f_1, \ldots, f_n\} \subseteq I_c$. Let $\phi^* \in \mathcal{N}(\Phi^+)$. We denote by $h : \phi^* \mapsto \{f_1, f_2, \ldots, f_n\}$, where $|\phi^*| = n$, a homomorphism from ϕ^* to I_c such that for every atom $R_i(\mathbf{x}, t_0)$ in

ϕ^*, $R_i(h(\mathbf{x}), h(t_0))$ is f_i, $1 \leq i \leq n$.

Example 8. Let Φ contains a temporal conjunction $\phi^+ = R^+(x,t) \wedge S^+(y,t)$. Then the corresponding $\mathcal{N}(\Phi)$ contains:

$$\phi^* = R^+(x,t_1) \wedge S^+(y,t_2)$$

$\triangle$

The intuitive idea behind using $\mathcal{N}(\Phi^+)$ instead of Φ^+ is to be able to map the temporal variable in each atom in a conjunction in $\mathcal{N}(\Phi^+)$ to a different time interval.

Definition 3.4. Empty intersection property A concrete instance I_c has the *empty intersection property* with respect to a set of temporal conjunctions Φ^+ if for every homomorphism h from a conjunction of atomic formulas $\phi \in \mathcal{N}(\Phi^+)$ to I_c such that $h : \phi^* \mapsto \{f_1, f_2, \ldots, f_n\}$, then $(\bigcap_{i\in\{1,\ldots,n\}} f_i[T]) = \emptyset$ or $\bigcap_{i\in\{1,\ldots,n\}} f_i[T] = \bigcup_{i\in\{1,\ldots,n\}} f_i[T]$.

In the next theorem we will show that an instance has the normalization property with respect to conjunctions of atomic formulas if and only if it has the empty intersection property.

Theorem 3.5. *Let Φ^+ be a set of temporal conjunctions. A concrete instance I_c is normalized with respect to Φ^+ if and only if I_c has the empty intersection property with respect to Φ^+.*

Proof. ***The if direction.*** In this direction, the concrete instance I_c is normalized and we show I_c has the empty intersection property as well. Let h be a homomorphism from $\phi^+(\mathbf{x}, t) \in \Phi^+$ to the instance I_c such that the images of the atoms in ϕ^+ under h are the concrete facts $f_1, ..., f_n$, where $n = |\phi^+|$. The temporal variable t (under h) has to map to a single interval $h(t) = [s, e)$ (otherwise a homomorphism cannot be defined). This means that $\forall i \in \{1, \ldots, n\}, f_i[T] = [s, e)$. Let $\phi^* \in \mathcal{N}(\Phi^+)$ be a

conjunction of atomic formulas that is obtained by replacing each occurrence of the temporal variable t in $\phi^+(\mathbf{x}, t)$ with a new variable. Define h' as follows:

$$h'(x) = h(x), \forall x \in \mathbf{x} \text{ and } h'(t') = h(t), t' \text{ a temporal variable in } \phi^*$$

We have $h' : \phi^* \mapsto \{f_1, \ldots, f_n\}$ (because $h : \phi^+ \mapsto \{f_1, \ldots, f_n\}$ and by construction of ϕ^*). Since $\forall i \in \{1, \ldots, n\}, f_i[T] = [s, e)$, we have $\bigcap_{i\in\{1,\ldots,n\}} f_i[T] = \bigcup_{i\in\{1,\ldots,n\}} f_i[T]$. Thus, I_c has the empty intersection property.

The only if direction Consider a $\phi^* \in \mathcal{N}(\Phi^+)$. Let ϕ^+ be the corresponding temporal conjunction of atomic formulas with the same temporal variable in each atom. Let $B = \{f_{c_1}, \ldots, f_{c_n}\}$ be a subset of I_c. Let h' be a homomorphism $h' : \phi^* \mapsto B$. Since I_c has the empty intersection property the time interval of the facts in B are either equal or the intersection of the time intervals is empty. In the latter case, no homomorphism can be defined from $\phi^+(\mathbf{x}, t)$ to I_c because the variable t in each atom cannot be mapped to a single interval. Therefore, we just consider the former case (that is the $\bigcup_{f_c\in B} f_c[T] = \bigcap_{f_c\in B} f_c[T]$). This means all the facts in B have the same time interval, that is $\forall f_c \in B\ f_c[T] = [s, e)$. Consider any concrete fact $f_c \in B$. W.l.o.g. we assume the interval-annotated nulls in the fact f_c are preceded by all the constants, that is

$$f_c = R(\mathbf{a}, \mathbf{N}^{[s,e)}, [s, e)).$$

By definition of $[\![.]\!]$, for each $f_{c_i} \in B$, $1 \leq i \leq n$, the snapshot $db_\ell \in [\![I_c]\!]$ $(s \leq \ell < e)$, contains the fact

$$f_{a_i} : R(\mathbf{a}, \Pi_\ell(\mathbf{N}^{[\mathbf{s},\mathbf{e})})).$$

Let ϕ be the corresponding conjunction of ϕ^+ over the snapshots. We need to show I_c has the normalization property. Define h as follows:

$$h(x) = h'(x), \forall x \in \mathbf{x} \text{ and } h(t) = [s, e)$$

Since h' is a homomorphism from ϕ^* to B and all the concrete facts in B has the same time interval, h is a homomorphism from ϕ^+ to I_c such that the image of each atom $R_i^+(\mathbf{x}, t)$ under h is a fact in B. Define homomorphisms $h_s, \ldots, h_{e-1}$ from $\phi(\mathbf{x})$ to consecutive snapshots db_s to db_{e-1}. For each $\ell \in \{s, \ldots, e-1\}$, define

$$h_\ell(x) = \begin{cases} h(x) & \text{if } h(x) \text{ is a constant} \\ \Pi_\ell(N^{[s,e)}) & \text{if } h(x) \text{ is an interval-annotated null } N^{[s,e)} \end{cases}$$

For each atom $R_i(\mathbf{x})$ in ϕ, the image of the atom under h_ℓ, that is $R_i(h_\ell(\mathbf{x})) = R(\mathbf{a}, \mathbf{N}_\ell)$ which is the fact f_{a_i} in the snapshot db_ℓ ($s \leq \ell < e$). Thus, condition 2 of the normalization property holds.

Consider any homomorphism h_ℓ, $s \leq \ell < e$. by definition of h' and h_ℓ it follows that:

$$\forall x \in \mathbf{x} \text{ if } h_\ell(x) = a \text{ then } h_\ell(x) = h(x)$$

Also, $\ell \in h(t) = [s, e)$. Thus, condition 1 in the definition of the normalization property holds as well. Therefore, I_c is normalized. □

Let Φ^+ be a set of temporal conjunctions. Let I_c be a concrete instance with n facts that is not normalized. We show the size of a normalized instance w.r.t. Φ^+ (obtained by fragmenting the concrete facts in I_c) is $\mathcal{O}(n^2)$.

Example 9. Suppose I_c is a concrete instance with two facts f_1 and f_2. Let Φ^+ contains one temporal conjunction of atomic formulas ϕ^+ (over schema of I_c). Suppose I_c is not normalized and there is a homomorphism from ϕ^* to $\{f_1, f_2\}$ where $\phi^* \in \mathcal{N}(\Phi^+)$ is the corresponding conjunction for ϕ^+. Let $f_1[T] = [s_1, e_1)$ and $f_2[T] = [s_2, e_2)$. Since I_c is not normalized $f_1[T] \cap f_2[T] \neq \emptyset$. Since the time intervals overlap, one of the following cases holds:

- $s_1 < s_2 < e_1 < e_2$

- $s_2 < s_1 < e_2 < e_1$
- $s_1 < s_2 < e_2 < e_1$
- $s_2 < s_1 < e_1 < e_2$

Here we consider the first item and show how to fragment the facts. In order to make I_c satisfy the empty intersection property, we fragment f_1 and f_2 as follows:

- f_{11}, where $f_{11}[T] = [s_1, s_2)$.
- f_{12}, where $f_{12}[T] = [s_2, e_1)$.
- f_{21}, where $f_{21}[T] = [s_2, e_1)$.
- f_{22}, where $f_{22}[T] = [e_1, e_2)$.

The data attribute values of f_{11} and f_{12} (resp. f_{21} and f_{22}) are same as f_1 (resp. f_2). Observe that any pair of the fragmented facts above satisfy ϕ^* and has disjoint or equal time intervals. $\triangle$

We assume whenever we fragment a concrete fact, the annotation of an interval-annotated null in the concrete fact is changed in the fragmented facts such that the annotation is always equal to the time interval of the fact the interval-annotated null occurs in. So if f_1 contains an interval-annotated null $N^{[s_1,e_1)}$, then f_{11} and f_{12} contain interval-annotated nulls $N^{[s_1,s_2)}$ and $N^{[s_2,e_1)}$ respectively. Each of the facts f_1 and f_2 is fragmented into two facts (with smaller time intervals).

Theorem 3.6. *Let I_c be a concrete instance with n facts that is not normalized w.r.t. a set of temporal conjunctions Φ^+. Let I'_c be a concrete instance that is obtained by fragmenting the facts in I_c such that I'_c is normalized w.r.t. Φ^+. The size of I'_c is $\mathcal{O}(n^2)$ if each fact in I_c needs to be fragmented.*

Proof. In general, a fact $f \in I_c$ (that needs to be fragmented) with time interval $[s_i, e_i)$ is fragmented into k_i number of facts such that k_i is the number of distinct start points and endpoints that are greater than or equal to s_i and smaller than e_i:

$$\underbrace{s_i < ... < s_j < ... < e_m < ... <}_{k_i} e_i$$

An instance I_c that contains n concrete facts have at most $2n$ distinct start points and end points. Therefore, $k_i \leq 2n - 1$. In the worst case (which depends on the set of conjunctions and the time interval of the facts that satisfy a conjunction of atomic formulas) each concrete fact needs to be fragmented considering all the distinct start points and end points in the instance. Therefore, the normalized instance is of size $\mathcal{O}(n^2)$. □

A naïve normalization algorithm, fragments each fact without considering any conjunction of atomic formulas (that is, $\Phi^+ = \emptyset$) and only based on the start points and end points of all the other facts. Such an algorithm needs to sort the start points and endpoints of all the facts. Thus, the time complexity of a A naïve normalization algorithm is $\mathcal{O}(nlogn)$ where n is the number of facts in the original (non-normalized) instance. However, a naïve normalization algorithm generates unnecessary fragments if there is no homomorphism from a conjunction of atomic formulas to any subset of the facts. Figure 3.8 depicts a normalized instance w.r.t. $\Phi^+ = \emptyset$ (generated by a naiïve normalization algorithm) which compared to the normalized instanced with respect to $E^+(n, c, t) \wedge S^+(n, s, t)$ shown in Figure 3.7 has more concrete facts.

We propose an algorithm that fragments the concrete facts in an instance based on Φ^+. The normalization algorithm $norm(I_c, \Phi^+)$ (Algorithm 1) receives a concrete instance I_c and $\mathcal{N}(\Phi^+)$, fragments the concrete facts in I_c and returns a normalized concrete instance I'_c w.r.t. Φ^+. The algorithm first builds the set $\mathcal{S}$ which is a set of sets of concrete facts in I_c that satisfy some formula $\phi^+ \in \mathcal{N}(\Phi^+)$. Then the sets

E^+

Name	Company	Time
Ada	IBM	[2012, 2013)
Ada	IBM	[2013, 2014)
Ada	Google	[2014, 2015)
Ada	Google	[2015, 2018)
Ada	Google	[2018, ∞)
Bob	IBM	[2013, 2014)
Bob	IBM	[2014, 2015)
Bob	IBM	[2015, 2018)

S^+

Name	Salary	Time
Ada	18k	[2013, 2014)
Ada	18k	[2014, 2015)
Ada	18k	[2015, 2018)
Ada	18k	[2018, ∞)
Bob	13k	[2015, 2018)
Bob	13k	[2018, ∞)

Figure 3.8: A normalized concrete source instance obtained by a naïve normalization algorithm

R^+

	A	T
f_1	a	[5,11)

P^+

	A	T
f_2	a	[8,15)
f_4	b	[20, 25)

S^+

	A	T
f_3	a	[7,10)
f_5	b	[18,∞)

Figure 3.9: Input of the normalization algorithm in Example 10

that have at least a concrete fact in common are moved to another set $S_\cap$. The sets that are in $S_\cap$ and have a concrete fact in common are merged until no more merges can be done. After adding the merged sets to $\mathcal{S}$, the concrete facts that are in each set $\Delta \in \mathcal{S}$ are fragmented by sorting the time intervals of the concrete facts in Δ and fragmenting the time intervals such that they do not overlap anymore. Example 10 shows how algorithm $norm(I_c, \Phi^+)$ works.

Example 10. Consider a schema with three relation symbols R^+, P^+, S each with attributes A and T. Consider an instance of I_c of this schema with five facts as shown in Figure 3.9. Let $\mathcal{N}(\Phi^+)$ contains two conjunctions of atomic formulas:

$$\phi_1 : R^+(x, t_1) \wedge P^+(y, t_2) \text{ and}$$

$$\phi_2 : P^+(x, t_1) \wedge S^+(y, t_2)$$

The algorithm first builds the set $\mathcal{S}$. In this example R is the only relation in the

Algorithm 1: $norm(I_c, \Phi^+)$

Input : Concrete instance I_c and Φ^+.
Output: Normalized instance I'_c w.r.t. Φ^+

$I'_c = I_c$;
build $\mathcal{N}(\Phi^+)$;

$$\mathcal{S} = \{\Delta \mid \Delta = \{f_1, \ldots, f_m\}, f_1, \ldots, f_m \in I_c \text{ such that } \bigcap_{f \in \Delta} f[T] \neq \emptyset$$

and $\exists \phi^+$ such that $\phi^+ \in \mathcal{N}(\Phi^+)$ and $m = |\phi^+|$ and there is a homomorphism h s.t. $h : \phi^* \mapsto \Delta\}$

$S_\cap = \{\Delta \in \mathcal{S} \mid \exists \Delta' \in \mathcal{S}. \exists f$ such that $f \in (\Delta \cap \Delta')\}$;

$\mathcal{S} = \mathcal{S} \backslash S_\cap$;

while $\exists \Delta_1, \Delta_2 \in S_\cap$ *such that* $(\Delta_1 \neq \Delta_2$ *and* $\Delta_1 \cap \Delta_2 \neq \emptyset)$ **do**
    $\Delta' = \Delta_1 \cup \Delta_2$;
    $S_\cap = (S_\cap \backslash \Delta_1, \Delta_2) \cup \{\Delta'\}$;
end
$\mathcal{S} = \mathcal{S} \cup S_\cap$;
for *each* $\Delta \in \mathcal{S}$ **do**
    $TP_\Delta = \langle tp_1, tp_2, \ldots, tp_m \rangle$, where tp_i is a distinct start point or end point in the facts in Δ and m is the number of distinct start points and end points in Δ;
    Sort TP_Δ in ascending order of time points;
    for *each* $f \in \Delta$ *such that* $f[T] = [s_i, e_i)$*:* **do**
        $TP_f = \langle s_i, \ldots, e_i \rangle$ is a sub-sequence of TP_Δ from time point s_i to time point e_i;
        $k = |TP_f| - 1$;
        Fragment the fact f_c to k facts such that

$$frg = \forall j \in \{1, \ldots, k\} \;\; f_j[\mathbf{D}] = f_c[\mathbf{D}] \text{ and } f_j[T] = [TP_f[j],\ TP_f[j+1])$$

        $I'_c = I'_c \backslash \{f\} \cup \; frg$
    end
end
Output: I'_c

instance, so

$$\mathcal{S} = \{\{f_1, f_2\}, \{f_2, f_3\}, \{f_4, f_5\}\}$$

Each set Δ in $\mathcal{S}$ satisfies a conjunction of atomic formulas (treating time intervals as constants) and in each Δ the intersection of the time intervals of the facts is not empty. The algorithm continues by building the set $S_\cap$ which is a subset of $\mathcal{S}$ and contains the sets of facts that have a common fact with each other. In this example

$$S_\cap = \{\{f_1, f_2\}, \{f_2, f_3\}\}.$$

After building $S_\cap$, the algorithm removes the sets in $S_\cap$ from $\mathcal{S}$ and merges the sets in $S_\cap$ that have common facts. In this example after merging the sets in $S_\cap$ we have:

$$S_\cap = \{\{f_1, f_2, f_3\}\}$$

After adding $S_\cap$ to $\mathcal{S}$:

$$\mathcal{S} = \{\{f_1, f_2, f_3\}, \{f_4, f_5\}\}$$

In this example there are two sets Δ_1 and Δ_2 in $\mathcal{S}$. The algorithm sorts the distinct start points and end points of the facts in Δ_1 and Δ_2:

- $TP_{\Delta_1} : \langle 5, 7, 8, 10, 11, \infty \rangle$
- $TP_{\Delta_2} : \langle 15, 18, 20, \infty \rangle$

Here we just show how the fact f_1 is fragmented.

- f_{11}, where $f_{11}[T] = [5, 7)$
- f_{12}, where $f_{12}[T] = [7, 8)$
- f_{13}, where $f_{13}[T] = [8, 10)$
- f_{14}, where $f_{14}[T] = [10, 11)$

R^+

	A	T
f_{11}	a	[5,7)
f_{12}	a	[7,8)
f_{13}	a	[8,10)
f_{14}	a	[10,11)

P^+

	A	T
f_{21}	a	[8,10)
f_{22}	a	[10,11)
f_{23}	a	[11,15)
f_4	b	[20, 25)

S^+

	A	T
f_{31}	a	[7,8)
f_{31}	a	[8,10)
f_{51}	b	[18,20)
f_{52}	b	[20,25)
f_{53}	b	[25,∞)

Figure 3.10: Output of the normalization algorithm

At the end the algorithm removes f_1 from the instance I'_c and adds the fragmented facts. The other facts in Δ_1 and Δ_2 are fragmented the same way as well. The final normalized instance is shown in Figure 3.10. $\triangle$

Theorem 3.7. *Let $I'_c = norm(I_c, \Phi^+)$. The instance I'_c is normalized.*

Proof. We will show I'_c has the empty intersection property. Therefore, based on Theorem 3.5, I'_c is normalized.

Let ϕ^* be a conjunction of atomic formula in $\mathcal{N}(\Phi^+)$. Let h be a homomorphism from ϕ^* to a set of concrete facts $f_{c_1}, \ldots, f_{c_n}$ in I'_c. We need to show either $(\bigcap_{i\in\{1,..,n\}} f_{c_i}[T]) = \emptyset$ or $\bigcap_{i\in\{1,..,n\}} f_{c_i}[T] = \bigcup_{i\in\{1,..,n\}} f_i[T]$.

Let $B = \{f_{c_1}, \ldots f_{c_n}\}$. Denote by $br(f)$ the set of fragmented facts of a concrete fact $f \in I_c$ obtained by the algorithm. Each fact f_{c_i} in B is either obtained by fragmenting a concrete fact f_i in I_c (that is $f_{c_i} \in br(f_i)$) or is a concrete fact in I_c (that is $f_{c_i} = f_i$). Define h' to be h except that the temporal variable t_i in each atom R_i is mapped to $h'(t_i) = f_i[T]$. Since $f_{c_i}[\mathbf{D}] = f_i[\mathbf{D}]$ and the temporal attribute values of f_{c_i} and f_i do not matter when considering $\phi^* \in \mathcal{N}(\Phi^+)$, h' is a homomorphism from ϕ^* to $\{f_1, f_2, \ldots f_n\}$.

If $(\bigcap_{i\in\{1,..,n\}} f_i) = \emptyset$, then it is obvious that $(\bigcap_{i\in\{1,..,n\}} f_{c_i}[T]) = \emptyset$. So we consider the case that $(\bigcap_{i\in\{1,..,n\}} f_i) \neq \emptyset$. Thus, there is a set in $\mathcal{S}$ such that $f_1, \ldots f_n \in \Delta$.

Let TP_Δ contains the sorted distinct start points and end points of the facts in Δ, that is $\langle \ell_1, \ldots, \ell_m \rangle$, where m is the number of distinct start points and end points in

Δ. Consider an arbitrary fact f_{c_i} in B. Let $f_{c_i}[T] = [s, e)$. By construction of the fact f_{c_i} by the algorithm (which is obtained by fragmenting a concrete fact $f_i \in \Delta$), we have $\langle \ell_1, \ldots, s, e, \ldots, \ell_m \rangle$. Observe that time point e is the immediate timepoint after s in TP_Δ. Therefore, if any fact f' in B has another start point (that is $f'[T] = [s', e')$ and $s \neq s'$), then the $\bigcap_{f \in \Delta} f[T] = \emptyset$. If all the facts in B have the same interval then $\bigcup_{f \in B} f[T] = \bigcap_{f \in B} f[T]$. Therefore, I'_c has the empty intersection property.

□

The time complexity of the normalization algorithm $norm(I_c, \Phi^+)$ by the assumption of fixing Φ^+ is polynomial in the size of I_c. Naïve normalization algorithm has a better time complexity but the size of the normalized instance is bigger because of the possibility of unnecessary fragments. In general there is a trade off between the cardinality of a normalized instance and the time complexity of a normalization algorithm. In Chapter 5 we have done some experiments on naïve normalization and $norm(I_c, \Phi^+)$ to show this trade off.

Note that even algorithm $norm(I_c, \mathcal{N}(\Phi^+))$ might fragment a fact unnecessarily because during merge the facts that are not directly satisfying a conjunction of atomic formulas are merged together. For example, the fragments f_{11} and f_{12} in Figure 3.10 can be coalesced into a fact $R^+(a, [5, 8))$ and the instance is still normalized. The reason is that time point 7 which causes these fragments is the start point of the fact f_3 and the facts f_1 and f_3 are not directly satisfying a conjunction of atomic formulas but they are in $\Delta_1 = \{f_1, f_2, f_3\}$. The only reason that they are in the same set in $\mathcal{S}$ is that the fact f_2. However, since the start point of f_3 (i.e. 7) is less that the start point of f_2 (i.e. 8) the fragmentation of f_1 by considering time point 7 is unnecessary. In order to find a minimum normalized instance, more computational time should be spent to find such cases and coalesce them.

3.3.3 Concrete chase (c-chase)

Putting everything together, in this section we define the concrete chase. Considering the lhs of all s-t tgds, first the concrete source instance needs to be normalized w.r.t. the lhs of the s-t tgds. Then all *s-t tgd c-chase steps* are applied sequentially to get a target instance. Then the target instance needs to be normalized w.r.t. the lhs of the egds. Finally a *concrete solution* is obtained by applying a successful sequence of *egd c-chase steps*. In the rest of this chapter, whenever we say a concrete instance is normalized w.r.t. Σ_{st} (resp. Σ_{eg}) it means it is normalized w.r.t. the lhs of Σ_{st} (resp. Σ_{eg}). The lhs of the s-t tgds and egds are considered as conjunctions of atomic formulas.

Definition 3.8. *c-chase step:*

- *(s-t tgd):* Let $\sigma^+ : \forall \mathbf{x}, t\ \phi^+(\mathbf{x}, t) \rightarrow \exists \mathbf{y}\ \psi^+(\mathbf{x}, \mathbf{y}, t)$ be an s-t tgd. Let I_c be the concrete normalized source instance and J_c be a concrete target instance (initially $J_c = \emptyset$). Let h be a homomorphism from lhs of σ to I_c such that there is no extension h' of h from $\phi^+(\mathbf{x}, t) \wedge \psi^+(\mathbf{x}, \mathbf{y}, t)$ to (I_c, J_c). We say σ^+ *can be applied to* (I_c, J_c) *with* h. Let J'_c be the union of J_c with the set of facts obtained by (a) extending h to h' so that each variable in $\mathbf{y}$ is assigned to a fresh null annotated with $h(t)$, followed by (b) applying h' to the rhs of σ^+. We say the result of *applying* σ^+ *to* (I_c, J_c) *is* (I_c, J'_c) and write $(I_c, J_c) \xmapsto{\sigma^+, h} (I_c, J'_c)$.

- *(egd):* Let $\sigma^+ : \forall \mathbf{x}, t\ \phi^+(\mathbf{x}, t) \rightarrow x_1 = x_2$ be an egd. Let J_c be the concrete normalized target instance such that $(I_c, J_c) \models \Sigma^+_{st}$. Let h be a homomorphism from σ to J_c such that $h(x_1) \neq h(x_2)$. We say that σ^+ *can be applied to* J_c *with* h. We distinguish two cases:

  - If both $h(x_1)$ and $h(x_2)$ are constant then *the result of applying* σ^+ *to* J_c *with* h is a *failure* and it is denoted by $J_c \xmapsto{\sigma^+, h} \perp$.

EMP^+

Name	Company	Salary	Time
Ada	IBM	$N^{[2012,2013)}$	[2012, 2013)
Ada	IBM	18k	[2013, 2014)
Ada	Google	18k	[2014, ∞)
Bob	IBM	$N^{[2013,2015)}$	[2013, 2015)
Bob	IBM	13k	[2015, 2018)

Figure 3.11: The concrete view of c-chase$(I_c, \mathcal{M}^+)$

– Otherwise, let J'_c be J_c where we identify $h(x_1)$ and $h(x_2)$ as follows: if one is a constant, then the interval-annotated null is replaced everywhere by the constant; if both are interval-annotated nulls, then one is replaced everywhere by the other. We say J'_c is *the result of applying* σ^+ *to* J_c *with* h, denoted by $J_c \xrightarrow{\sigma^+,h} J'_c$.

Note that in an egd c-chase step, the annotated nulls have the same time interval because the only way a homomorphism can be defined to the lhs of an egd chase step is to map variable t to a single time interval. We assumed that all the interval annotated nulls in a fact are annotated with the fact's time interval.

A concrete chase is a finite sequence of s-t tgd chase steps followed by egd chase steps. We call the result of a successful concrete chase a *concrete solution*. If an egd c-chase step fails, then the result of c-chase is a failure.

Example 11. The result of concrete chase on the concrete input instance shown in Figure 3.6 with the schema mappings in the Example 6 is shown in Figure 3.11. △

We have shown the result of a successful chase on the abstract view is a universal solution (Proposition 3.2). Since in practice concrete instances are used the aim is to show the result of c-chase on a concrete instance has the correct semantics as if we were able to apply chase on the abstract view of that instance. This is shown in Figure 3.12. Following the Fagin et al. [16] approach to prove that the result of the chase procedure is a universal solution, we first show a property of a c-chase step in

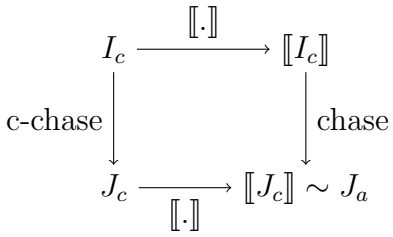

Figure 3.12: Correspondence between concrete chase on I_c and chase on $[\![I_c]\!]$

Lemma 3.9. Using this lemma we show in Theorem 3.10 that if J_c is the result of a successful c-chase, then $[\![J_c]\!]$ is a universal abstract solution. The proof steps in Lemma 3.9 follows the proof steps of Lemma 3.4 in [16]. The main difference is that the notion of homomorphism is defined from abstract instances to abstract instances (not on concrete instances). Meanwhile, a concrete chase uses homomorphisms from a temporal dependency to a concrete instance. Therefore, in the proof we have to deal with a homomorphism from a temporal dependency to an instance and its effects on a homomorphism from an abstract instance to another abstract instance.

Lemma 3.9. *Let $K_c = (I_c, J_c)$ be a concrete normalized instance w.r.t. a dependency σ^+. Let $K_c \overset{\sigma^+,h}{\longmapsto} K'_c$ be a chase step. Let $K_a = \langle db''_0, db''_1, \ldots \rangle$ be an abstract instance such that $K_a \models \sigma$ and $h' : [\![K_c]\!] \mapsto K_a$. Then there is a homomorphism $g : [\![K'_c]\!] \mapsto K_a$.*

Proof. Let $[\![K_c]\!] = \langle db_0, db_1, \ldots \rangle$. Let $[\![K'_c]\!] = \langle db'_0, db'_1, \ldots \rangle$. Having the homomorphism $h' : [\![K_c]\!] \mapsto K_a$ means that there is a homomorphism from each snapshot in $[\![K_c]\!]$ to the corresponding snapshot in K_a,

$$h'_\ell : db_\ell \mapsto db''_\ell.$$

Case 1: *σ^+ is an s-t tgd.* Then $h : \phi^+(\mathbf{x}, t) \mapsto K_c$. Suppose $h(t) = [s, e)$. Based on Theorem 3.5 there are homomorphisms $h_s, \ldots, h_{e-1}$ from $\sigma = \phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$

to $db_s, \ldots db_{e-1}$, respectively in $[\![K_c]\!]$. Consider h_ℓ $(s \leq \ell < e)$:

$$h_\ell : \phi(\mathbf{x}) \mapsto db_\ell.$$

Composing homomorphisms yields homomorphisms, thus:

$$h'_\ell \circ h_\ell : \phi(\mathbf{x}) \mapsto db''_\ell.$$

Since $db''_\ell \models \sigma$, then there exists a homomorphism h'' such that

$$h''_\ell : \phi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y}) \mapsto db''_\ell,$$

where h''_ℓ is an extension of $h'_\ell \circ h_\ell$ such that $h''_\ell(\mathbf{x}) = h'_\ell(h_\ell(\mathbf{x}))$. For each variable $y \in \mathbf{y}$ in ψ^+, a fresh interval-annotated null is generated in K'_c that is annotated with $h(t) = [s, e)$. Denote by $N'^{[s,e)}$ the interval-annotated null generated in the chase step $K_c \xrightarrow{\sigma^+, h} K'_c$. Therefore, by definition of $[\![.]\!]$, there is a labeled null N'_ℓ in $db'_\ell \in [\![K'_c]\!]$ $(s \leq \ell < e)$. Define g_ℓ on $\mathsf{Null}(db'_\ell)$ as follows: $g_\ell(N_\ell) = h'_\ell(N_\ell)$, if $N_\ell \in \mathsf{Null}(db_\ell)$, and $g_\ell(N'_\ell) = h''_\ell(y)$ for $y \in \mathbf{y}$.

In order to show $g : [\![K'_c]\!] \mapsto K_a$ we need to show there is a homomorphism between the corresponding snapshots. Hence, we need to show that g_ℓ is a homomorphism from db'_ℓ to db''_ℓ , $s \leq \ell < e$. For the facts of db'_ℓ that are also in db_ℓ this is true because h' is a homomorphism; thus $h'_\ell : db_\ell \mapsto db''_\ell$. Let $R^+(\mathbf{x_0}, \mathbf{y_0}, t)$ be an arbitrary atom in ψ^+. Therefore, the atom $R(\mathbf{x_0}, \mathbf{y_0})$ is in ψ. Then $R^+(h(\mathbf{x_0}), h(\mathbf{y_0}), h(t))$ is a fact in K'_c. By definition of $[\![.]\!]$ there is a fact $R(h(\mathbf{x_0}), \pi_\ell(h(\mathbf{y_0})) = R(h(\mathbf{x_0}), \mathbf{N}_{\mathbf{y}_\ell})$ in db'_ℓ. Based on Theorem 3.5 we showed $h(\mathbf{x_0}) = h_\ell(\mathbf{x_0})$. By replacing h with h_ℓ in R and taking the image of this fact under g_ℓ we have:

$$R(g_\ell(h_\ell(\mathbf{x_0})), g_\ell(\mathbf{N}_\ell)) = R(h''_\ell(\mathbf{x_0}), h''_\ell(\mathbf{y_0})).$$

The homomorphism h'' maps all the atoms of $\phi \wedge \psi$, in particular $R(\mathbf{x_0}, \mathbf{y_0})$ into facts in db''_ℓ. Thus g_ℓ is a homomorphism. The only remaining thing is to show

$$\begin{aligned}
&\forall i, j \in \mathbb{N}_0 \text{ such that } i \neq j \\
&\text{and } g_i : db_i \mapsto db'_i \quad \text{and } g_j : db_j \mapsto db'_j, \\
&\forall N \in \mathsf{Null}(db'_\ell)\ g_i(N) \neq g_j(N)
\end{aligned}$$

For the nulls in db'_ℓ that are already in db_ℓ this is true because h' is a homomorphism. A null $N'^{[s,e)}$ (replacing $y \in \mathbf{y}$) generated by the concrete chase step $K_c \xrightarrow{\sigma^+,h} K'_c$ results in the labeled nulls $\langle N'_s, N'_{s+1}, \ldots, N'_{e-1} \rangle$ in the snapshots $db'_s, \ldots db'_{e-1}$ respectively. The homomorphism g'_ℓ is an extension of g_ℓ such that

$$g'_\ell(N) = \begin{cases}
h_s(N) \text{ if } N \in \mathsf{Null}(db_s) \\
h_{s+1}(N) \text{ if } N \in \mathsf{Null}(db_{s+1}) \\
\ldots \\
h_\ell(N) \text{ if } N \in \mathsf{Null}(db_\ell) \\
\ldots \\
h_{e-1}(N) \text{ if } N \in \mathsf{Null}(db_{e-1}).
\end{cases}$$

Case 2: σ^+ is an egd. In this case the difference between K_c and K'_c is that some interval-annotated nulls in K_c are replaced with other interval-annotated nulls or constants. But there is no new constant or interval-annotated null generated in K'_c.

If σ^+ can be applied on K_c with h, based on Theorem 3.5 we know there is a homomorphism h_ℓ from $\phi(x)$ in $db_\ell \in [\![K_c]\!]$. As in case 1:

$$h'_\ell \circ h_\ell : \phi(x) \mapsto db''_\ell, \quad \ell \in \mathbb{N}_0, \quad db''_\ell \in K_a.$$

Since each snapshot db''_ℓ ($\ell \in \mathbb{N}_0$) in K_a satisfies the egd σ,

$$h'_\ell(h_\ell(x_1)) = h'_\ell(h_\ell(x_2)).$$

We take g_ℓ to be h'_ℓ. We need to show h'_ℓ is still a homomorphism from db'_ℓ to db''_ℓ. The only way that h'_ℓ can fail to be a homomorphism on db'_ℓ is if h'_ℓ maps $h_\ell(x_1)$ and $h_\ell(x_2)$ into two different constants or labeled nulls of db''_ℓ, which is not the case because $h'_\ell(h_\ell(x_1)) = h'_\ell(h_\ell(x_2))$. □

Theorem 3.10. *Assume a data exchange setting where Σ^+_{st} consists of s-t tgds and Σ^+_{eg} consists of egds.*

1. *Let (I_c, J_c) be the result some successful finite concrete chase of $(I_c, \emptyset)$ with $\Sigma^+_{st} \cup \Sigma^+_{eg}$. Then $[\![J_c]\!]$ is a universal solution.*

2. *If there exists some failing chase of $(I, \emptyset)$ with $\Sigma^+_{st} \cup \Sigma^+_{eg}$, then there is no solution.*

Proof. part 1: The proof is based on Lemma 3.9 and the proof of Theorem 3.3 in [16]. Let J_a be an arbitrary solution (for example the result of chase on $[\![I_c]\!]$). Then $([\![I_c]\!], J_a)$ satisfies $\Sigma_{st} \cup \Sigma_{eg}$. The identity mapping $id : ([\![I_c]\!], \emptyset) \mapsto ([\![I_c]\!], J_a)$ is a homomorphism. By applying lemma 3.9 at each s-t tgd c-chase steps, we have $h_1 : ([\![I_c]\!], [\![J'_c]\!]) \to ([\![I_c]\!], J_a)$. Then by applying lemma 3.9 at each egd chase step, we have $h : ([\![I_c]\!], [\![J_c]\!]) \to ([\![I_c]\!], J_a)$. In particular h is a homomorphism from $[\![J_c]\!]$ to J_a. Thus, $[\![J_c]\!]$ is a universal solution.

part 2: Let $(I_c, J_c) \xrightarrow{\sigma^+, h} \emptyset$ be the last egd c-chase step of a failing c-chase. Then σ^+ must be an egd in Σ^+_{eg}, say $\phi(\mathbf{x}, t) \mapsto (x_1 = x_2)$ and $h : \phi(\mathbf{x}, t) \mapsto J_c$ is a homomorphism such that $h(x_1)$ and $h(x_2)$ are *two distinct constants a_1 and respectively,* a_2. Suppose $h(t) = [s, e)$. Let $[\![J_c]\!] = \langle db_0, db_1, \ldots, \rangle$. As in proof of Lemma 3.9 we have a homomorphism h_ℓ from lhs $\sigma : \phi(\mathbf{x}) \to x_1 = x_2$ to db_ℓ ($s \leq \ell < e$):

$$h_\ell : \phi(\mathbf{x}) \mapsto db_\ell$$

Assume by contradiction that there exists a solution $J_a = \langle db'_0, db'_1, \ldots, \rangle$. Since J_a is a solution, $db'_\ell \models \sigma$, $(s \leq \ell < e)$. The identity homomorphism

$$id : (\llbracket I_c \rrbracket, \emptyset) \mapsto (\llbracket I_c \rrbracket, J_a)$$

implies, by Lemma 3.9, the existence of homomorphism $g : (\llbracket I_c \rrbracket, \llbracket J_c \rrbracket) \mapsto (\llbracket I_c \rrbracket, J_a)$. In particular, g is also a homomorphism from $\llbracket J_c \rrbracket$ to J_a, which means:

$$g_\ell : \ db_\ell \mapsto db'_\ell, \quad \ell \in \mathbb{N}_0$$

Then

$$g_\ell \circ h_\ell : \phi(\mathbf{x}) \mapsto db''_\ell \quad s \leq \ell < e$$

Since $db'_\ell \models \sigma$, it must be the case that $g_\ell(h_\ell(x_1)) = g_\ell(h(x_2))$ and thus $g_\ell(a_1) = g_\ell(a_2)$. Homomorphisms are identity on Const, and so $a_1 = a_2$, which is a contradiction. □

Corollary 3.11. *Assume a data exchange setting in which Σ^+_{st} consists of s-t tgds and Σ^+_{eg} consists of egds. Let I_c be a normalized concrete source instance w.r.t. Σ_{st}. Let J_c be the result of c-chase on I_c. Let J_a be the result of chase on $\llbracket I_c \rrbracket$. Then $\llbracket J_c \rrbracket$ is homomorphically equivalent to J_a, that is $\llbracket J_c \rrbracket \sim J_a$.*

3.4 Query Answering

In addition to finding a universal solution for a data exchange problem, another important issue in data exchange is query answering over the target schema [16]. When queries are posed over the target schema, different answers may be obtained depending on the solution that is considered. To cope with the multiplicity of query results the notion of certain answers is used, where the answers are the intersection of all the answers to the query on each possible solution [16].

Let q be a non-temporal k-ary query, for $k \geq 0$. Let I_a be an abstract instance,

that is $I_a = \langle db_0, db_1, \ldots \rangle$. Let $\mathcal{M}$ be a data exchange setting. The certain answers of q w.r.t. I_a and $\mathcal{M}$, denoted by $certain(q, I_a, \mathcal{M})$, is the sequence of sets of certain answers of q on each snapshot

$$certain(q, I_a, \mathcal{M}) = \langle certain(q, db_0, \mathcal{M}), certain(q, db_1, \mathcal{M}), \ldots \rangle$$

where $certain(q, db_\ell, \mathcal{M})$ ($\ell \in \mathbb{N}_0$) is the set of k-tuples r of constants from db_ℓ, such that for every solution db'_ℓ of the snapshot db_ℓ w.r.t. a schema mapping $\mathcal{M}$, $r \in q(db'_\ell)$:

$$certain(q, db, \mathcal{M}) = \bigcap_{db' \text{ is a solution for } db w.r.t. \mathcal{M}} q(db')$$

Naïve evaluation [1, 4, 22, 16] is a technique commonly used in the literature to find certain answers for unions of conjunctive queries on naïve tables. It has been shown [4, 16] that naïve evaluation of unions of conjunctive queries on a universal solution db' for a relational source instance db gives certain answers. Denote by $q(db)_\downarrow$ the result of naïve evaluation of query q on db which is obtained by treating the labeled nulls as new constants (that is $N = N$, $N \neq M$, and $N \neq a$, where $a \in \mathsf{Const}$). It is shown that $certain(q, db, \mathcal{M}) = q(db')_\downarrow$ where db' is a universal solution for db w.r.t. a data exchange setting [4, 16]. Denote by $q(J_a)_\downarrow$ the result of naïve evaluation of the query q on a universal solution $J_a = \langle db'_0, db'_1, \ldots \rangle$ for a source instance I_a w.r.t. a data exchange setting. Thus,

$$ceratin(q, I_a, \mathcal{M}) = q(J_a)_\downarrow = \langle q(db'_0)_\downarrow, q(db'_1)_\downarrow, \ldots \rangle.$$

Let q be a query on the target schema in the abstract view. Denote by q^+ the corresponding query (obtained by augmenting all the atoms in the query q by a free variable t) for the target schema in the concrete view.For concrete instances, . Given a union of conjunctive queries q^+ and a concrete solution J_c for a source instance I_c

w.r.t. a data exchange setting, the naïve evaluation of q^+ on J_c, denoted by $q^+(J_c)_\downarrow$ is:

$$q^+(J_c)_\downarrow = \bigcup_{q' \text{ is a disjunct of } q^+} q'(J_c)_\downarrow$$

where $q'(J_c)_\downarrow$ is defined as follows:

1. Normalize instance J_c w.r.t. q'. We denote the normalized instance by J'_c
2. Each interval-annotated null $N^{[s,e)}$ in J_c is replaced with a fresh constant $cn^{[s,e)}$ everywhere it occurs. The result of this step is J''_c.
3. Query q is evaluated by finding all homomorphisms from variables in q to J''_c. In particular, the variable t is mapped to a time interval. The result of this step is denoted by $q(J''_c)$.
4. Tuples with fresh constants are dropped from $q(J''_c)$ to yield $q'(J_c)_\downarrow$.

The following theorem shows that naïve evaluation on a concrete solution produces the same answers as naïve evaluation on the corresponding abstract solution under the semantic mapping.

Theorem 3.12. *Let J_c be a concrete solution for a source instance I_c w.r.t. $\mathcal{M}$. Let q^+ be a union of conjunctive queries over the concrete target schema and q the corresponding union of conjunctive queries on abstract target schema. Then $[\![q^+(J_c)_\downarrow]\!] = q([\![J_c]\!])_\downarrow$.*

Proof. Let $[\![J_c]\!] = \langle db_0, db_1, \ldots \rangle$. Let $(a_1, \ldots, a_k, [s, e))$ be a $(k+1)$-ary tuple in $q^+(J_c)_\downarrow$. Then, $[\![q^+(J_c)_\downarrow]\!] = \langle \mathbf{r_0}, \mathbf{r_1}, \ldots \rangle$ where $\mathbf{r}_\ell$ ($\ell \in \mathbb{N}_0$) is a set of k-ary tuples defined as follows:

$$\mathbf{r}_\ell = \{\ (a_1, \ldots, a_k) \mid \exists i. \exists j.\ (a_1, \ldots a_k, [i, j)) \in q^+(J_c)_\downarrow,\ i \leq \ell < j\}$$

Since $(a_1, \ldots, a_k, [s, e)) \in q^+(J_c)_\downarrow$ there is a homomorphism h from a conjunctive query q' that is a disjunct of q^+ to J_c. W.l.o.g. we assume $q' : \exists y \phi^+(\mathbf{x}, y, t)$. The proof can be easily extended when there is more than one existentially quantified variable. Also, the snapshots $\mathbf{r_s}$ to $\mathbf{r_{e-1}}$ in $[\![q^+(J_c)]\!]$ contain the tuple $h(\mathbf{x})$ by definition.

Let $R^+(\mathbf{x}, y, t)$ be an arbitrary atom in ϕ^+. Then $R^+(h(\mathbf{x}), h(y), h(t))$ is a fact in J_c. Depending on whether $h(y)$ is a constant or an interval-annotated null we consider two cases:

Case 1: $h(y) = a^*$ where $a^* \in \mathsf{Const}$. In this case the snapshots db_s to db_{e-1} in $[\![J_c]\!]$ contains the fact $R(a_1, \ldots, a_k, a^*)$. Define homomorphisms $h_s, \ldots h_{e-1}$ as follows: $h_\ell(\mathbf{x}) = h(\mathbf{x})$ and $h_\ell(y) = h(y)$, $s \leq \ell < e$. Then $h_s, \ldots, h_{e-1}$ are homomorphisms from $\phi(\mathbf{x}, y)$ to $db_s, \ldots, db_{e-1}$, respectively because $R(h_\ell(\mathbf{x}), h_\ell(y))$ is a fact in db_ℓ, $s \leq \ell < e$. Hence, $h_\ell(\mathbf{x}) = (a_1, \ldots, a_k)$ is in $q(db_\ell)$.

Case 2: $h(y) = N^{[s,e)}$. In this case, define homomorphisms $h_s, \ldots h_{e-1}$ as follows: $h_\ell(\mathbf{x}) = h(\mathbf{x})$ and $h_\ell(y) = \pi_\ell(h(y)) = N_\ell$. Then $h_s, \ldots, h_{e-1}$ are homomorphisms from $\phi(\mathbf{x}, y)$ to $db_s, \ldots, db_{e-1}$, respectively. Therefore, the tuple $(a_1, \ldots, a_k)$ is in $q'(db_\ell)$ and consequently in $q^+(J_c)_\downarrow$.

For the other direction, let $(a_1, \ldots, a_k)$ be a tuple in the result of q on consecutive snapshots $db_s, db_{s+1}, \ldots db_{e-1}$, that is

$$(a_1, \ldots, a_k) \in q(db_\ell),\ s \leq \ell < e$$

Therefore, there exists a conjunctive query $\exists y \phi(\mathbf{x}, y)$ that is a disjunct of q and there exists a homomorphism h_ℓ

$$h_\ell :\ \phi(\mathbf{x}, y) \mapsto db_\ell.$$

That means if $R(\mathbf{x}, y)$ is an atom in ϕ then $R(h_\ell(\mathbf{x}), h_\ell(y))$ is in db_ℓ. Observe that in this direction we also assume one existentially variable y in the conjunctive query. But this assumption is without loss of any generality.

Based on the value of $h_\ell(y)$ we consider two cases:

Case 1: $h_\ell(y) = a^*$, where $a^* \in \mathsf{Const}$. Thus, $R^+(a_1, \ldots, a_k, a^*, [s, e))$ is a fact in J_c. Define a homomorphism h on variables in $\phi^+(\mathbf{x}, y, t)$ (in a disjunct in q^+) as follows:

$$h(z) = \begin{cases} h_\ell(z) & \text{if } h_\ell(z) \text{ is a constant} \\ [s, e) & \text{if } z = t \end{cases}$$

h is a homomorphism from $\phi^+(\mathbf{x}, y, t)$ to J_c because it maps an arbitrary atom such as $R^+(\mathbf{x}, y, t)$ in ϕ^+ to a fact $R^+(h(\mathbf{x}), h(y), h(t))$ in J_c. Therefore, $(a_1, \ldots a_k, [s, e))$ is in $q^+(J_c)_\downarrow$ which means $\mathbf{r}_\ell$, $s \leq \ell < e$ contains the tuple $(a_1, \ldots a_k)$.

Case 2: $h_\ell(y) = N_\ell$, $s \leq \ell < e$. In this case by definition of $[\![J_c]\!]$, $R^+(a_1, \ldots, a_k, N^{[s,e)}, [s, e))$ is a fact in J_c.

$$h(z) = \begin{cases} h_\ell(z) & \text{if } h_\ell(z) \text{ is a constant} \\ N^{[s,e)} & \text{if } h_\ell(z) = N_\ell \\ [s, e) & \text{if } z = t \end{cases}$$

Same as previous case, h is a homomorphism from $\phi^+(\mathbf{x}, y, t)$ to J_c and $\mathbf{r}_\ell$ in $[\![q^+(J_c)]\!]$ contains the tuple $(a_1, \ldots a_k)$.

□

Corollary 3.13. *Let J_c be the result of concrete chase on a concrete source instance I_c w.r.t. a temporal schema mapping $\mathcal{M} = (R_S, R_T, \Sigma_{st}, \Sigma_{eg})$. Let q^+ be a union of conjunctive queries over the target schema. Then certain$(q, [\![I_c]\!], \mathcal{M}) = [\![q^+(J_c)_\downarrow]\!]$.*

3.5 Related Work

Here we overview the previous relevant work on data exchange and temporal databases. The formal foundations of data exchange were developed by Fagin et al. in [16]. The authors showed that the chase algorithm, previously used for checking

implication of data dependencies, can be used to produce a universal solution for instances of the data exchange problem. Universal solutions map homomorphically to other solutions for the source instance. This property makes them the preferred solutions to query answering. Universal solutions and, in general, solutions of instances of data exchange problem can contain incomplete information. Representing incomplete information and evaluating queries over them are more complex than in the complete case as shown by Imielinski and Lipski [22] as well as Abiteboul et al. in [1]. The gap between the theoretical work on incomplete information and what has been used in practice is discussed by Gheerbrant et al. [20] and Libkin [30]. The chase algorithm proposed by Fagin et al. in [16] produces labeled nulls for incomplete values. Relations containing labeled nulls are called *naïve tables* [1, 22]. In data exchange the semantics of answering queries is defined in terms of *certain answers* [4, 16]. Certain answers [22] are tuples that belong to the answer of the posed query no matter which solution is used. Fagin et al. showed that whenever a universal solution can be computed in polynomial time (for the class of dependencies identified in [16]), certain answers to unions of conjunctive queries can also be computed in polynomial time in data complexity in [16]. Computing certain answers for queries that have more than one inequality, however, is a coNP-complete problem [16]. Data exchange and incomplete information and other possible semantics for query answering are discussed in detail by Libkin in [29].

The formal foundations of temporal data models and query languages were studied by Chomicki in [10] and by Chomicki and Toman in [13]. Abstract versus concrete temporal views were first developed in the context of the semantics of temporal query languages [38]. These kinds of views of temporal data were also used in program debugging and dynamic program analysis [27]. However, Chomicki [10] and Chomicki and Toman [13] did not discuss incomplete temporal information and its possible semantics. The notion of normalization was previously used in the context of query

answering in temporal databases [27, 39] on tuples with the same schema that agree on non-temporal attribute values. The normalization algorithms that we introduced change the concrete instance w.r.t. conjunctions of atomic formulas. Koubarakis proposed a unified framework for both finite and infinite, definite and indefinite temporal data [25, 24]. His suggested framework extends *conditional tables* (a.k.a. c-tables) [22] and can be used to store facts such as roomA is booked from 2 to *sometime between 5 to 8*. He used global conditions to define the constraints on the start point or end point of a time interval. In his framework, the temporal attribute values can be unknown. C-tables are a generalization of naïve tables where a table is associated with global and local conditions specified by logic formulas [22]. In Koubarakis framework the indefinite (incomplete) temporal data are not the result of data exchange. His framework, does not deal with schema mappings or integrity constraints. We proposed interval-annotated nulls for unknown values in concrete data exchange to align the semantics of temporal data exchange with the data exchange on abstract vie w. Also in our framework the value of the temporal attribute is known because the schema mappings are non-temporal. Therefore, there is no condition on the temporal attribute or non-temporal attributes as a result of data exchange. Naïve tables are sufficient for representing incomplete information in temporal data exchange with non-temporal schema mappings.

3.6 Conclusion

In this chapter, we proposed a framework for data exchange on temporal data which relies on the distinction between the abstract and concrete view of the data. Abstract view is responsible for the semantics while the concrete view is used in the implementations. We considered a basic case where the schema mapping is non-temporal. We first extended the standard chase procedure on abstract instances. Defining chase on the abstract instances provides a conceptual tool on how a concrete

chase should work. Then we defined a concrete chase on concrete instances. We showed normalization of the concrete instance is necessary to define homomorphisms from the lhs of a dependency with a shared temporal variable among the atoms to a concrete instance. We finished the chapter by showing the result of the concrete chase is a good candidate to be materialized and used for answering queries.

CHAPTER 4

Temporal Repair Checking and Repair Construction

A database db is consistent w.r.t. a set of integrity constraints Σ defined on its schema if it satisfies all the constraints in Σ ($db \models \Sigma$). Data may be collected from imprecise sources such as social media with imprecise procedures such as natural-language processing. Inconsistency may also arise during data exchange and data integration where data from different autonomous (and even separately consistent) sources are integrated. Arenas et al. [5] introduced a principled approach to managing inconsistency via the notions of *repairs* and *consistent query answering*. Repairing a database means bringing the database in accordance with a given set of integrity constraints by applying some minimal changes. If a database can be repaired in more than one way, then the *consistent answer* to a query is defined as the intersection of the query answers on all repaired versions of the database. If repairs can be obtained only by tuple deletion, then they are called *subset repairs*. Subset repairs are the maximal consistent subsets of the original database instance.

In this chapter, we first adopt a basic decision problem *repair checking* [11, 3] (is a database a repair of another database?) to obtain *temporal repair checking*. A repair checking algorithm (represented by $RepChk(db, db', \Sigma)$) receives two databases with the same schema as input (here db and db') and returns true if db' is a repair

of *db*. The class of integrity constraints that we consider throughout the chapter is temporal functional dependencies [13]. Then, we propose two algorithms for repairing a temporal database. Note that this dissertation does not propose a criterion of goodness to compare the repairs generated by each algorithm. The goal of comparing these algorithms is mainly to contrast a time point based repair algorithm with a time interval based repair algorithm.

4.1 Preliminaries

In this chapter we use the timestamp model for the abstract view. The reason is that the relations in the timestamp model are the same as the relations in the relational model (except that abstract relations can be infinite) and we can re-use relational repair checking algorithms by providing the way of handling the infinity. Recall from Chapter 2 that in the timestamp model, the same schema is used for both the abstract and concrete databases.

Let $f_c = R(\mathbf{a}, [s, e))$ be a concrete fact in a concrete instance I_c, we denote by $[\![f_c]\!]$ the set of the *abstract facts* that it represents: $[\![f_c]\!] = \{R(\mathbf{a}, \ell) \mid \ell \in [s, e)\}$.

We assume the instances in this chapter are complete (i.e. without nulls). We use $f[\bar{D}]$ and $f[T]$ to refer to data attribute values and respectively the temporal attribute value of the fact f.

Definition 4.1. *Switch point:* Let I_a be an abstract instance in the snapshot model. The smallest time point for which I_a satisfies the finite change condition is called the *switch point* of I_a.

Definition 4.2. *Temporal functional dependencies (tFDs):* Consider the relation schema $R(X, Y, Z, T)$, where X, Y, Z are sets of attributes and T is the temporal attribute. A *temporal functional dependency* $X, T \rightarrow Y$ holds in an abstract database I_a in the timestamp model if the classical functional dependency $X, T \rightarrow Y$ holds in

I_a[13].

Every tFD can be written as a first-order sentence. For example, tFD $X, T \rightarrow Y$ can be written as the following first-order sentence.

$$\forall \bar{x}, \bar{y}, \bar{y}', \bar{z}, \bar{z}', t \; R(\bar{x}, \bar{y}, \bar{z}, t) \wedge R(\bar{x}, \bar{y}', \bar{z}', t) \rightarrow \bar{y} = \bar{y}'$$

A *temporal key constraint* over relation R is $X, T \rightarrow R$ which shows the values of the attributes X and T uniquely identifies a fact in R.

Definition 4.3. *Conflict:* The abstract facts $f_{a_1}, \ldots, f_{a_n}$ are in a conflict w.r.t. an integrity constraint σ if $f_{a_1}, \ldots, f_{a_n}$ witness a violation of σ, that is $f_{a_1}, \ldots, f_{a_n} \not\models \sigma$.

The concrete facts $f_{c_1}, \ldots, f_{c_n}$ are in a conflict w.r.t. an integrity constraint σ if there exist $f_{a_1} \in [\![f_{c_1}]\!], \ldots, f_{a_n} \in [\![f_{c_n}]\!]$ such that $f_{a_1}, \ldots, f_{a_n} \not\models \sigma$.

For the tFDs, a conflict involves only two facts in the same relation. Note that when two abstract facts f_a and f'_a are in a conflict w.r.t. a tFD, then $f_a[T] = f'_a[T]$. On the other hand, if two concrete facts f_c and f'_c are in a conflict w.r.t. a tFD, then $[\![f_c]\!][T] \cap [\![f'_c]\!][T] \neq \emptyset$.

As our integrity constraints are tFDs, it suffices to consider a database with a single relation $R(A_1, \ldots, A_n, T)$. In a general database, our results can be applied to each relation individually. In the rest of this chapter whenever we refer to a database instance (abstract or concrete) it only has one relation.

Let Σ be a set of tFDs and $R(X, Y, T)$ be a concrete normalized relation w.r.t. Σ. A tFD $XT \rightarrow Y$ holds in R if the tFD $XT \rightarrow Y$ holds in R' which is obtained by replacing the distinct intervals in R with distinct constants.

4.2 Temporal Repair Checking

In this section, we identify the cases where the relational repair checking cannot be used for temporal databases and discuss the notion of *temporal repair checking*,

which addresses the shortcomings of the relational repair framework.

If an abstract relation is finite, then a relational repair checking algorithm can be used. In the rest of the paper, we just consider the temporal databases that are infinite. Therefore, abstract databases contain an infinite relation and concrete databases contain a fact with time interval of the form $[s, \infty)$, $s \in \mathbb{N}_0$.

We define abstract repairs based on relational repairs.

Definition 4.4. *[Abstract repair]:* An instance I'_a is an abstract repair of I_a w.r.t. a set of integrity constraints Σ if

- $I'_a \models \Sigma$;
- The set difference between I_a and I'_a is minimal;
- The snapshot view of I'_a satisfies the finite change condition.

Example 12. Consider the relation schema E(Name, WorksFor, Time). Let $Name, Time \rightarrow WorksFor$ be a temporal functional dependency over relation E. Consider the concrete database I_c with two facts:

$$I_c : \{f_1 : E(Ada, IBM, [1, 8)), f_2 : E(Ada, Google, [2, \infty))\}.$$

The concrete database I_c is inconsistent because $[\![I_c]\!]$ is inconsistent. The concrete facts f_1 and f_2 are in a conflict. Note that unlike in relational databases, removing a concrete fact does not produce a repair here because the difference between $[\![I_c]\!]$ and $[\![f_1]\!]$ (resp. $[\![I_c]\!]$ and $[\![f_2]\!]$) is not minimal. In the Example 13, some of the repairs of I_c are listed. $\triangle$

We define *concrete repairs* based on the abstract repairs.

Definition 4.5. Let I_c and I'_c be two concrete databases over the same schema. A concrete database I'_c is a *concrete repair* of I_c if $[\![I'_c]\!]$ is an abstract repair of $[\![I_c]\!]$.

Definition 4.6. *Temporal repair checking ($TRepChk(I_c, I'_c, \Sigma)$):* Given concrete databases I_c and I'_c over the same schema, is I'_c a concrete repair of I_c?

Example 13. Consider the instance I_c from the Example 12. According to the definition of concrete repairs (Definition 4.5) any of the following instances is a concrete repair for I_c:

1. $I'_1 = \{E(Ada, IBM, [1, 2)), E(Ada, Google, [2, \infty))\}$,
2. $I'_2 = \{E(Ada, IBM, [1, 8)), E(Ada, Google, [8, \infty))\}$,
3. $I'_3 = \{E(Ada, IBM, [1, 7)), E(Ada, Google, [7, \infty))\}$,
4. $I'_4 = \{E(Ada, IBM, [1, 2)), E(Ada, Google, [2, 6)),$
   $E(Ada, IBM, [6, 8)), E(Ada, Google, [8, \infty))\}$

△

Observe that the cardinality of some of these concrete repairs is smaller than that of the others.

Next we define *e^*-reduction* to reduce a concrete instance with infinite semantics to a concrete instance with finite semantics.

Definition 4.7. *[e^*-reduction]:* An *e^*-reduction* with a time point e^* (or *reduction* if e^* is known) is a function from concrete facts to concrete facts (i.e. e^*-reduction: $\mathcal{F}_c \mapsto \mathcal{F}_c$, where $\mathcal{F}$ is a set of concrete facts). The result of applying e^*-reduction on a concrete fact f_c is another concrete fact f'_c such that:

1. f'_c is not defined if $f_c[T] = [s, \infty)$ and $s \geq e^*$,
2. $f'_c[\mathbf{D}] = f_c[\mathbf{D}]$ and $f'_c[T] = [s, e^*)$ if $f_c[T] = [s, \infty)$ and $s < e^*$,
3. $f'_c = f_c$ if $f_c[T] = [s, e)$, $e \neq \infty$.

We call f'_c the *reduced fact* of f_c.

Intuitively, an e^*-reduction is identity on the concrete facts with finite end points and it only reduces the concrete facts with ∞ as the end point to concrete facts with e^* as the end point as long as e^* is greater than the start point of the facts. Otherwise, if $s \geq e^*$, the resulting concrete fact f'_c is meaningless.

The e^*-reduction can be lifted from concrete facts to concrete instances $\mathcal{I}_c$ such that the result of applying e^*-reduction on a concrete instance $I_c \in \mathcal{I}_c$ is another concrete instance I_{e^*} where

$$I_{e^*} = e^*\text{-reduction}(I_c) = \bigcup_{f_c \in I_c} e^*\text{-reduction}(f_c)$$

We call I_{e^*} the *reduced instance* of I_c.

Example 14.

$$I_c : \{f_1 = E(Ada, IBM, [7, \infty)), f_2 = E(Ada, Google, [7, \infty))\}$$

An e^*-reduction of I_c with any timepoint $e^* \leq 7$ is not defined. An e^*-reduction of I_c with timepoint 8 is:

$$I_8 = \{E(Ada, IBM, [7, 8)),\ E(Ada, Google, [7, 8))\}$$

Note that if $NameTime \rightarrow WorksFor$ is the temporal key constraint (tkc) of relation E, then the facts in I_c are in a conflict. Also, note that the facts in the reduced instance I_8 are also in a conflict w.r.t. the tkc. △

Proposition 4.8 shows whenever the time point e^* is greater than all the start points in a concrete instance I_c, then $[\![I_{e^*}]\!] \subset I_c$, where $I^*_e = e^*\text{-reduction}(I_c)$.

Proposition 4.8. *Let $I_{e^*} = e^*$-reduction(I_c), where time point e^* is greater than all the start points in I_c. Then $[\![I_{e^*}]\!] \subset I_c$.*

A relational repair checking algorithm $RepChk(db, db', \Sigma)$ where Σ is a set of functional dependencies, iteratively goes over each relational fact f_r in $db - db'$ and checks if $db' \cup \{f_r\} \models \Sigma$. The issue that makes relational repair checking algorithms inapplicable on concrete databases is that the concrete databases might have infinite semantics. The goal in temporal repair checking $TRepChk(I_c, I'_c, \Sigma)$ is to use a reduction on a concrete instance to obtain another concrete instance that has finite semantics and keeps the conflicts i.e. a *lossless reduction*. In this way we can use the relational repair checking algorithms after the reduction has been applied.

Definition 4.9. *[Lossless reduction:]* An e^*-reduction of a concrete instance I_c is lossless w.r.t. a set of integrity constraints if whenever the facts $f_{c_1}, \ldots, f_{c_n}$ in I_c are in a conflict then their reduced facts $f'_{c_1}, \ldots, f'_{c_n}$ in the reduced instance I_{e^*} are also in a conflict.

To find a lossless reduction w.r.t. a set of tFDs one needs the switch point of a concrete instance.

Example 15. Consider the coalesced concrete database

$$I_c = \{E(Ada, IBM, [2, \infty)), E(Ada, Google, [3, \infty))\}.$$

The maximum of all start points and finite end points is 3, therefore, the timepoint 3 is the switch point based on Theorem 4.10. Note that if I_c is not coalesced, Theorem 4.10 fails. For example, consider

$$I'_c = \{E(Ada, IBM, [2, \infty)), E(Ada, Google, [3, 18)), E(Ada, IBM, [18, \infty))\}.$$

Note that $[\![I'_c]\!] = [\![I_c]\!]$. However, the time point 18 is not the switch point of I'_c because the smallest time point for which $[\![I'_c]\!]$ (in the snapshot model) satisfies the finite change condition is time point 3. $\triangle$

The finite change condition was defined in the snapshot model of temporal databases where the abstract instance is an infinite sequence of relational databases. Theorem 4.10 shows the switch point of a coalesced concrete instance I_c is the maximum of all start points and finite end points.

Theorem 4.10. *Let I_c be a coalesced concrete database. The switch point of I_c is the maximum of all start points and finite end points of the intervals in I_c (ignoring $e = \infty$).*

Proof. The proof uses the equivalence of the timestamp and the snapshot models. If f_c is a concrete fact in I_c such that $f_c[T] = [s, e)$, then the snapshots $db_s, db_{s+1}, \ldots, db_{e-1}$ of $[\![I_c]\!]$ contain the relational fact $f_c[\mathbf{D}]$. Note that for every abstract fact $f \in f_c$, $f[\mathbf{D}] = f_c[\mathbf{D}]$ by definition of the semantics mapping. A concrete database that has finite semantics $[\![I_c]\!] : \langle db_1, \ldots, db_n \rangle$ can be seen as an infinite sequence of relational databases $[\![I_c]\!] : \langle db_1, \ldots, db_n, db_{n+1}, \ldots \rangle$ such that $db_{n+1} = db_{n+2} = \ldots = \emptyset$.

Consider the snapshot view of $[\![I_c]\!]$, that is $\langle db_0, \ldots, db_{FP}, db_{FP+1}, \ldots \rangle$. Let FP be the switch point of $[\![I_c]\!]$, that is the smallest time point that satisfies the finite change condition. We will show FP is the maximum of all start points and finite end points in I_c.

Since FP is the switch point of $[\![I_c]\!]$ we have $db_{FP} = db_{FP+1} = \ldots$ and $db_{FP-1} \neq db_{FP}$. Consider any relational fact f_r in any snapshot db_{FP+i}, $i \geq 0$. Because $db_{FP} = db_{FP+1} = \ldots$ there should be a concrete fact f_c in I_c such that $f_r = f_c[\mathbf{D}]$. We will show $f_c[T] = [s, \infty)$ for some $s \leq FP$. Because I_c is coalesced, it is not possible to have a pair of concrete facts f'_c and f''_c in I_c such that $f'_c[\mathbf{D}] = f''_c[\mathbf{D}]$ and $f'_c[T] = [s', e')$ and $f''_c[T] = [s'', e'')$ where $e' = s''$. Therefore, in order for $f_r = f_c[\mathbf{D}]$ to be in the snapshot db_{FP} and any snapshot after the temporal attribute value of f_c, that is $f_c[T]$, must be equal to $[s, \infty)$ for some $s \leq FP$. This shows if the temporal attribute value of a concrete fact has infinity as the end point, then the start point is less than or equal to the switch point of the instance that contains the fact.

Since $db_{FP-1} \neq db_{FP}$, one can conclude there is a fact $f'_r \in db_{FP-1}$ that is not in db_{FP} or there is a fact $f''_r \in db_{FP}$ that is not in db_{FP-1}. In the former case there must be a fact f'_c in I_c such that $f'_r = f'_c[\mathbf{D}]$ and $f'_c[T] = [s', FP)$ for some $s' < FP$. In the latter case there must be a fact $f''_c \in I_c$ such that $f''_r = f''_c[\mathbf{D}]$ and $f''_c[T] = [FP, e)$, for some $e > FP$. Both cases show that the time point FP is either the start point or the endpoint of the temporal attribute value of a concrete fact in I_c. The end point of $f''_c[T]$ must be $e = \infty$ otherwise the snapshot db_{e-1} contains $f''_c[\mathbf{D}]$ while $f''_c[\mathbf{D}] \notin db_e$ which contradicts with FP being the switch point of $[\![I_c]\!]$. Therefore, $f''_c[T] = [FP, \infty)$. Hence, if the end point of a temporal attribute value of a fact is other than infinity, then the endpoint is less than or equal to FP. The maximum of all start points and end points in I_c is FP.

□

Theorem 4.11. *Let Σ be a set of tFDs. Let FP be the switch point of a normalized concrete instance I_c w.r.t. Σ. Any e^*-reduction of I_c with $e^* > FP$ is lossless.*

Corollary 4.12 shows that if there is a conflict in the reduced instance, then it is because there is a conflict in the original instance. In other words, no conflict w.r.t. tFDs emerges by just limiting the end point of an instance.

Corollary 4.12. *Let I_c be a normalized concrete instance and FP be the switch point of I_c. Consider an instance I_{e^*} which is obtained by e^*-reduction of I_c where $e^* > FP$. Let f'_1 and f'_2 in I_{e^*} be the reduced facts of f_1 and f_2 in I_c. If f'_1 and f'_2 are in a conflict w.r.t. a tFD, then the two facts f_1 and f_2 are in a conflict as well.*

Proof. This is because $[\![I_{e^*}]\!] \subseteq [\![I_c]\!]$ and the tFDs are monotone. □

We call a reduction that is not lossless, a *lossy reduction*. As an example, consider the instance I_c in the Example 14. The facts in I_c are in a conflict w.r.t. the temporal key constraint. The reduction of I_c with time point 7 is lossy because the reduced facts are not defined.

We say I' *is a maximal subset of* I *satisfying* Σ *(or* I' *is maximal for short)* if $[\![I']\!]$ is a maximal subset of $[\![I]\!]$ satisfying Σ. Lemma 4.13 shows that I' is a maximal subset of I satisfying Σ, if and only if I'_{e^*} (the reduced database of I') is a maximal subset of I_{e^*} (the reduced database of I) with $e^* = Max(FP, FP') + 1$, where FP and FP' are switch points of I and I' respectively.

Lemma 4.13. *Let* I *and* I' *be two concrete instances such that* $[\![I']\!] \subseteq [\![I]\!]$. *Let* Σ *be a set of tFDs. Suppose* $[\![I']\!]$ *satisfies* Σ. *Let* $FP_m = Max(FP, FP')$ *where* FP *and* FP' *are the switch points of* I *and* I' *respectively. Let* I_{e^*} *and* I'_{e^*} *be the reduced instances of* I *and respectively* I' *with* $e^* = FP_m + 1$. I' *is a maximal subset of* I *that satisfies* Σ *(that is,* $\nexists f_a \in [\![I]\!] - [\![I']\!]$ *such that* $[\![I']\!] \cup \{f_a\} \models \Sigma$*) if and only if* I'_{e^*} *is a maximal subset of* I_{e^*} *satisfying* Σ.

Theorem 4.14. *Let* I *and* I' *be two concrete databases such that* $[\![I']\!] \subseteq [\![I]\!]$. *Let* Σ *be the set of tFDs defined on the schema of* I. *Let* $FP_m = Max(FP, FP')$, *where* FP *and* FP' *are the switch points of* I *and* I' *respectively. Let* I_{e*} *and* I'_{e*} *be the* e^**-reduction of* I *and* I' *respectively where* $e^* = FP_m + 1$. *Then*

$$TRepChk(I, I', \Sigma) \text{ iff } TRepChk(I_{e*}, I'_{e*}, \Sigma)$$

Proof. First we prove if $TRepChk(I_{e*}, I'_{e*}, \Sigma)$ is true, then $TRepChk(I, I', \Sigma)$ is true. Assume by contrary that $TRepChk(I, I', \Sigma)$ is false. Hence, either $I' \not\models \Sigma$ or I' is not a maximal subset satisfying Σ. We consider each case separately.

- $[\![I']\!] \not\models \Sigma$: If $[\![I']\!] \not\models \Sigma$ that means there is a conflict in the database I'. Since the reduction of I' to I'_{e^*} with $FP_m + 1$ is lossless (according to Theorem 4.11) there is a conflict in I'_{e*} as well which means $[\![I'_{e^*}]\!]$ is inconsistent; therefore, I'_{e^*} is not a concrete repair.

- I' is not maximal: By Lemma 4.13, I'_{e^*} is not maximal either.

In both cases, it is concluded that I'_{e^*} is not a repair of I_{e^*} which is a contradiction.

Now we show if $TRepChk(I_{e*}, I'_{e^*}, \Sigma)$ is false, then $TRepChk(I, I', \Sigma)$ is false. Since I'_{e^*} is not a repair of I_{e^*}, it means either $I'_{e^*} \not\models \Sigma$ or I'_{e^*} is not maximal. If $I'_{e^*} \not\models \Sigma$, then I' is not a repair of I because based on Corollary 4.12 $I'_{e^*} \not\models \Sigma$. If I'_{e^*} is not maximal, then I' is also not maximal (Lemma 4.13).

Note that if $TRepChk(I, I', \Sigma)$ is true, then $TRepChk(I_{e*}, I'_{e*}, \Sigma)$ is true as well because if $TRepChk(I_{e*}, I'_{e*}, \Sigma)$ is false, it contradicts with $TRepChk(I, I', \Sigma)$ being true as we proved above. Also, if $TRepChk(I, I', \Sigma)$ is false, then $TRepChk(I_{e*}, I'_{e*}, \Sigma)$ is false as well, otherwise, it contradicts with $TRepChk(I, I', \Sigma)$ being false as proved above.

□

Algorithm 2 is the pseudo code of the implementation of Theorem 4.14.

The complexity of a repair checking algorithm depends on the complexity of checking the consistency of an database (w.r.t. a class of constraints) and the complexity of checking the minimality of the difference between the repair and the original database [6]. The complexity of *TRepChk* algorithm is same as the complexity of relational repair checking for functional dependencies, which is in PTIME.

Algorithm 2: Temporal Repair Checking $TRepChk(I, I', \Sigma)$

Input : Two coalesced concrete databases I and I' such that $[\![I']\!] \subset [\![I]\!]$, a set of tFDs Σ

Output: **True** if I' is a temporal repair of I w.r.t. Σ, **False** otherwise.

```
1  FP_m ← maximum of all start points s and end points e in I and I';
2  e* ← FP + 1;
3  I_e* ← e*-reduction(I);
4  I'_e* ← e*-reduction(I');
5  if I'_e* ⊭ Σ then
6  |  return False;
7  end
8  while there is a fact f in [[I_e*]] − [[I'_e*]] do
9  |  if I'_e* ∪ {f} ⊨ Σ then
10 |  |  return False;
11 |  end
12 |  [[I_e*]] ← [[I_e*]] − {f};
13 end
14 return True;
```

4.3 Temporal Repair Construction

In this section, we discuss if it is possible to construct a repair for a concrete database I_c by constructing a repair for its reduced database obtained by $e^* = FP+1$, where FP is the switch point of I_c.

Definition 4.15. *e^*-expansion:* An e^*-expansion with a time point e^* (or *expansion* if e^* is known) is a function from $\mathcal{F}_c \mapsto \mathcal{F}_c$, where $\mathcal{F}_c$ is a set of concrete facts. The result of applying e^*-expansion on a concrete fact f_c is another concrete fact f'_c such that:

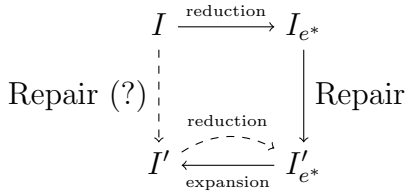

Figure 4.1: Repair construction for temporal concrete database I where $e^* = FP + 1$

1. $f'_c = f_c$ if $f_c[T] = [s, e)$ for some ($e < e^*$ or $e > e^*$).

2. $f'_c[\mathbf{D}] = f_c[\mathbf{D}]$ and $f'_c[T] = [s, \infty)$ if $f_c[T] = [s, e^*)$.

Similar to the e^*-reduction, an e^*-expansion can be lifted to concrete databases as well:

$$e^*\text{-expansion}(I_c) = \bigcup_{f_c \in I_c} (e^*\text{-expansion}(f_c))$$

Observe that if all the concrete facts in I_c have end points less than e^*, then the e^*-expansion of I_c is identity on I_c. The reduced concrete database obtained by an e^*-expansion of an database is a super-set of the original database (considering the abstract viewsv).

Theorem 4.16 is the main result of this section. Figure 4.1 depicts this result.

Theorem 4.16. *Let I be a concrete coalesced database with FP as the switch point. Let $e^* = FP + 1$. Let I_{e^*} be the reduction of I. Let I'_{e^*} be a repair of I_{e^*} w.r.t. a set of tFDs Σ. Then $I' = e^*$-expansion(I_{e^*}) is a repair for I.*

Algorithm 2 ($TRepChk(I, I', \Sigma)$) can be modified to build a temporal repair. First the database I is reduced to I_{e^*} with $FP + 1$, where FP is the switch point of I. Then by setting I'_{e^*} to $\emptyset$ and iteratively adding a fact $f \in [\![I_{e^*}]\!] - [\![I'_{e^*}]\!]$ to I'_{e^*} while $I'_{e^*} \cup \{f\}$ still satisfies Σ a repair is constructed for I_{e^*}. At the end, by using an expansion on I'_{e^*} with $FP + 1$ a repair for I is obtained. We call this point based repair construction algorithm *PntRCon*. Observe that, depending on the order of adding facts, a different repair is obtained at each run of the algorithm.

Algorithm 3: $PntRCon(I)$ Point-based repair construction algorithm

Input : A coalesced concrete databases I, a set of tFDs Σ

Output: A temporal Repair of I w.r.t. Σ

1 $FP \leftarrow$ switch point of I;

2 $e^* \leftarrow FP + 1$;

3 $I_{e^*} \leftarrow e^*$-reduction (I);

4 $I'_{fp} \leftarrow \emptyset$;

5 **while** *there is an abstract fact* f_a *in* $[\![I_{e^*}]\!] - [\![I'_{e^*}]\!]$ **do**

6 | **if** $[\![I'_{e^*}]\!] \cup \{f_a\} \models \Sigma$ **then**

7 | | $f_c[\mathbf{D}] = f_a[\mathbf{D}]$;

8 | | $f_c[T] = [f_a[T], f_a[T] + 1)$;

9 | | $I'_{e^*} \leftarrow I'_{e^*} \cup \{f_c\}$;

10 | **end**

11 | $[\![I_{e^*}]\!] \leftarrow [\![I_{e^*}]\!] - f_a$;

12 **end**

13 coalesce the facts in I_{e^*};

14 $I' \leftarrow$ expansion (I_{e^*});

15 return I';

In lines 4-9 we are finding a repair for I_{e^*} based on relational repair construction [11, 34]. Theorem 4.17 shows the switch point of a repair is the same as the switch point of the original database.

Theorem 4.17. *Let I be a concrete normalized database and FP be its switch point. Let Σ be a set of tFDs. The switch point of the repair constructed by algorithm $PntRCon$ is also FP.*

In the next proposition we will show Algorithm $PntRCon$ is *sound* and *complete*, that is it produces a repair for I with switch point of FP (soundness) and produces all the repairs of I that have FP as their switch point (completeness)

Proposition 4.18. *Algorithm PntRCon is sound and complete.*

Algorithm *PntRCon* is not efficient when the majority of the concrete facts are not in a conflict. Let f_c be a concrete fact in an inconsistent concrete database I_c. Suppose f_c is not in any conflicts in I_c. If we use the algorithm *PntRCon* to construct a repair for I_c, for every abstract fact in $[\![f_c]\!]$, lines 4-8 will be executed. Therefore, we propose another algorithm for temporal repair construction that uses factorization and leverages the time intervals (Algorithm $IntRCon$). In Algorithm $IntRCon$, a fact such as f_c, which is not in any conflicts, will be in any repair without a need to loop over every abstract fact in $[\![f_c]\!]$ and check if it can be added to a repair. In algorithm $IntRCon(I, \Sigma)$ we first find all the subsets of the facts that are violating a tFD in Σ.

$$\mathbf{cs} \leftarrow \{\Delta | \Delta = \{f_1, f_2\}\ f_1 \in I_c, f_2 \in I_c \text{ and } \exists \sigma \in \Sigma \text{ such that } \Delta \not\models \sigma\}$$

Since I_c is factorized, satisfaction of a temporal functional dependency $\sigma \in \Sigma$ can be checked by treating time intervals as constants. Also, observe that since I_c is normalized the facts that are in a set $\Delta \in \mathbf{cs}$ have the same time interval.

Some of the sets in **cs** have a common concrete fact. Algorithm $IntRCon(I, \Sigma)$ merges the sets in **cs** with a common fact. After merging the algorithm continues by selecting each set in **cs**.

For better understanding of the steps in the algorithm, suppose we have a schema E'(Name, WorksFor, Salary, Time) where $Name, Time \rightarrow WorksFor, Salary$ is the tFD. Consider the temporal database in Figure 4.2**(a)**. It is inconsistent w.r.t. the tFD. The factorized database is shown in 4.2**(b)**.

$\mathbf{cs} = \{\{f_2, f_3\}, \{f_4, f_5\}, \{f_4, f_6\}, \{f_5, f_6\}\}$. The set $\mathbf{cs}_\cap$ is a subset of **cs** containing the sets that have a fact in common: $\mathbf{cs}_\cap = \{\{f_4, f_5\}, \{f_4, f_6\}, \{f_5, f_6\}\}$. After merging we have:

$$\mathbf{cs} = \{\{f_2, f_3\}, \{f_4, f_5, f_6\}\}$$

Name	WorksFor	Salary	Time
Ed	IBM	60k	[0,10)
Ed	Google	90k	[5,∞)
Ed	IBM	70k	[2,10)
Jo	Google	75k	[8,∞)

(a)

	Name	WorksFor	Salary	Time
f_1	Ed	IBM	60k	[0,2)
f_2	Ed	IBM	60k	[2,5)
f_3	Ed	IBM	70k	[2,5)
f_4	Ed	IBM	60k	[5,10)
f_5	Ed	IBM	70k	[5,10)
f_6	Ed	Google	90k	[5,10)
f_7	Ed	Google	90k	[10,∞)
f_8	Jo	Google	75k	[8,∞)

(b)

Figure 4.2: **(a)** An inconsistent temporal database;**(b)** After normalization w.r.t. Σ

conflicts	Name	WorksFor	Salary	Time
Δ_1	Ed	IBM	60k	[2,5)
	Ed	IBM	70k	[2,5)
Δ_2	Ed	IBM	60k	[5,10)
	Ed	IBM	70k	[5,10)
	Ed	Google	90k	[5,10)

Figure 4.3: The conflicts for database shown in Figure 4.2

which is shown in Figure 4.3. The facts that are not in **cs** will be in every repair. The algorithm iterates over each set in **cs**; chooses a fact and a subset of its time interval and adds it to the set A which is initially empty. The algorithm continues adding facts until no more facts can be added and A is still consistent. Algorithm $IntRCon$ works as follows: After reducing the original instance I into the reduced instance I_{e^*}. the algorithm copies all the concrete facts that are not in any conflict set in I_{e^*} into the repair I'_{e^*}. Then the algorithm iteratively picks a concrete fact f from a conflict set and construct a subset of f (called f') and tries to add the constructed fact f' to A which is initially empty. If after factorizing $A \cup \{f'\}$ a conflict emerges then the subset of f' that does not cause the conflict is added to A.

Algorithm $IntRCon$ finds all the subsets of size two in I that are in a conflict w.r.t. a tFD. It checks the satisfaction of tFDs and uses naïve normalization (needs sorting of the start points and end points in $A \cup \{f'\}$). In the worst case all the the subsets of size two are in **cs**. If m is the number of tFDs. We assume m is very

Algorithm 4: *IntRCon* Interval-based repair construction algorithm

Input : A normalized concrete database I w.r.t. a set of tFDs Σ
Output: A repair of I w.r.t. Σ

$I'_{e^*} \leftarrow \emptyset$;
$I' \leftarrow \emptyset$;
$e^* \leftarrow FP + 1$;
$I_{e*} \leftarrow$ reduction (I);
$\mathbf{cs} \leftarrow \{\Delta | \Delta = \{f_1, f_2\}\ f_1 \in I_c, f_2 \in I_c$ and $\exists \sigma \in \Sigma$ such that $\Delta \not\models \sigma\}$;
$I' \leftarrow I \backslash (\bigcup_{\Delta \in \mathbf{cs}} \Delta)$;
$\mathbf{cs}_\cap \leftarrow \{\Delta \in \mathbf{cs} \mid \exists \Delta' \in \mathbf{cs}. \exists f$ such that $f \in (\Delta \cap \Delta')\}$;
$\mathbf{cs} = \mathbf{cs} \backslash \mathbf{cs}_\cap$;
while $\exists \Delta_1, \Delta_2 \in \boldsymbol{cs}_\cap$ *such that* $(\Delta_1 \neq \Delta_2$ *and* $\Delta_1 \cap \Delta_2 \neq \emptyset)$ **do**
    $\Delta' = \Delta_1 \cup \Delta_2$;
    $\mathbf{cs}_\cap = (\mathbf{cs}_\cap \backslash \Delta_1, \Delta_2) \cup \{\Delta'\}$;
end
$\mathbf{cs} = \mathbf{cs} \cup \mathbf{cs}_\cap$;
for *each* $\Delta \in \boldsymbol{cs}$ **do**
    $A \leftarrow \emptyset$;
    while $\Delta \neq \emptyset$ **do**
        choose a fact f in Δ ;
        choose a new interval $[s_1, e_1)$ s.t. $[s_1, e_1)$ is a subset of $f[T]$;
        $f'[\mathbf{D}] \leftarrow f[\mathbf{D}]$;
        $f'[T] \leftarrow [s_1, e_1)$;
        $B \leftarrow$ normalize $A \cup \{f'\}$;
        for *each fragment* f'' *of* f' *that causes conflict in* B **do**
            $B \leftarrow B \backslash \{f''\}$;
        end
        $A \leftarrow B$;
    end
    $I'_{e*} \leftarrow I'_{e*} \cup A$;
end
Coalesce the facts in I'_{e*};
$I' \leftarrow$ expansion(I'_{e*});
output I';

small compared to the size of the input instance. Therefore the time complexity is in PTIME in the cardinality of the input instance.

Theorem 4.19 shows Algorithm *IntRCon* (Algorithm 4) is sound and complete. Note that the switch point of the repair is also FP and the proof is same as the proof for Theorem 4.17.

Theorem 4.19. *Algorithm IntRCon is sound (i.e. always produces a repair) and complete (i.e. all repairs can be produced by the algorithm).*

Proof. In order to show that the algorithm is sound, one needs to show the produced result is always a repair, meaning that the output instance I' not only satisfies Σ but also is maximal. By definition of the set **cs**, the facts that are not in any set in **cs** satisfy Σ. When the set of facts in A is added to the database I'_{e^*} (that is, $I'_{e*} \leftarrow I'_{e*} \cup A$), the database I'_{e^*} still satisfies Σ because if there is a fact f_c in I'_{e^*} that is in conflict with a fact f'_c in A w.r.t. a tFD, then f_c should have been in a set in **cs** in the first place. Therefore, set A not only does not contain any conflicts, but also does not introduce any conflicts after being added to I_{e^*}.

In order to show that the resulting database is minimally different from the original one, we need to show when a fact f'_c with the interval $[s_1, e_1)$ is discarded in Algorithm *IntRCon*, no subset of it can be added to A and still satisfies Σ. Since Algorithm *IntRCon*, factorizes $A \cup \{f'\}$ and checks if a subset of f' (denoted by f'_2 in the algorithm) or f' itself can be added to A. Only the time points in $f'[T]$ that cause conflicts in A are discarded.

For completeness proof, consider any repair I' of a database I such that FP is the switch point of both I and I'. Let I' be a coalesced database and I (the input of the algorithm) a factorized database. Denote by I'_{e^*} and I_{e^*} the reduced databases of I' and respectively I with $e^* = FP + 1$. Note that since Σ is a set of tFDs, $[\![I']\!] \subseteq [\![I]\!]$ (and $[\![I'_{e^*}]\!] \subseteq [\![I_{e^*}]\!]$). We show Algorithm *IntRCon* can generate I' which is obtained by expansion of I'_{e^*}. We need to fragment the instance I'_{e^*} into an instance I'' such

that:

$$\forall f \in I'' \; \exists f' \in I_{e^*} \text{ such that either } f[T] = f'[T] \text{ or } [\![f]\!][T] \subset [\![f']\!][T].$$

The reason to fragment the facts in I'_{e^*} is to distinguish the concrete facts that are in a conflict set in **cs**. For each $\Delta \in \mathbf{cs}$, define the set $\mathbf{rs}_\Delta$ in the following way:

$$\mathbf{rs}_\Delta = \{f'' | f'' \in I'' \wedge \exists f \in \Delta \text{ such that } [\![f'']\!][T] \subseteq [\![f]\!][T]\}.$$

Since I' is a repair and Σ is a set of tFDs, we have:

$$\mathbf{rs}_\Delta \models \Sigma.$$

Algorithm *IntRCon* first transfers all the facts in I_{e^*} that are not in any conflict set to the repair. Database I'' contains these facts because I'_{e^*} is a repair and I'_{e^*} contains them.

When the algorithm constructs the set A for each Δ, one can deterministically choose the facts in $\mathbf{rs}_\Delta$. Therefore, the repair obtained by the algorithm is the same as I''. By coalescing the concrete facts in I'', the database I'_{e^*} is obtained and the expansion of I'_{e^*} with e^* results in I'.

□

4.4 Related Work

We first survey the papers on relational repair checking and then discuss the paper [14] which is more related to our approach.

There are a few papers that investigate the repair checking problem such as [11, 3]. Chomicki and Marcinkowski [11] investigated subset-repair checking problem for different classes of integrity constraints such as functional dependencies and inclusion

dependencies. They showed that if Σ (i.e. a set of integrity constraints) is a union of acyclic set of inclusion dependencies and a set of functional dependencies, then repair checking w.r.t. Σ is in PTIME. However, the repair checking problem for arbitrary FDs and INDs is coNP-complete [11]. Later, Afrati and Kolaitis [3] proved that if Σ is weakly acyclic sets of LAV tgds (which have only one atom on the left hand side of the tgd) with a set of egds, the (subset) repair checking is in PTIME. Arming et al. provide a thorough analysis of the complexity of the repair checking problem for different kinds of constraints and different types of complexity (such as data and combined) [6].

Chomicki and Wisjen in the paper [14] discuss consistent query answering in temporal databases where the integrity constraints are non-temporal FDs to held in every snapshot. This is equivalent to tFDs holding in the entire temporal database. The authors tried to find a condition on repair functions such that each snapshot can be repaired individually and independently of past and future snapshots. If a repair function has this condition, then some query re-writings can be used for CQA for a particular class of temporal queries. The authors also introduced persistent repairs. A persistent repair is a maximal (w.r.t. cardinality) consistent subset of the *minimal factorized concrete temporal database* instance [1]. For example, in the Example 13, repairs I'_1 and I'_2 are persistent repairs. Persistent repairs are obtained from the *factorized* inconsistent concrete database by choosing only one of the concrete facts in a conflict set. A persistent repair is a concrete repair but not vice versa because concrete repairs choose *subsets of concrete facts* in a conflict set. Persistent repairs have the smallest cardinality among all concrete repairs, however as shown in the Example 13 that there are other concrete repairs (e.g. repair I'_3) with the same cardinality as persistent repairs.

[1] See the sub-section 4.4.1 for definition of factorization.

Name	WorksFor	Zipcode	Country	Time
Ada	IBM	14221	USA	[2,5)
Ada	Intel	98345	USA	[2,5)
Ada	IBM	98345	Canada	[2,5)
Bob	Google	888 HQ	USA	[5,10)
Bob	Oracle	78987	USA	[5,10)

Figure 4.4: An inconsistent concrete instance w.r.t. σ_1 and σ_2

4.4.1 Detailed comparison with paper [14]

Factorization vs Normalization Let I_c be a concrete instance. Let Σ be a set of denial constraints [18]. A *conflict set* of I_c w.r.t. Σ is a minimal (with respect to set inclusion) subset **cs** of I_c that is inconsistent with respect to Σ. The concrete database I_c is *factorized* with respect to Σ if for every conflict set **cs** of I_c, all temporal facts of **cs** have identical time intervals [14]. We have not used any of these definitions because of the ambiguity in the definition of a conflict set. For example, minimality is defined based on set inclusion. Consider the instance shown in Figure 4.4 and the following tFDs:

$$\sigma_1 : Name, Time \rightarrow WorksFor$$

$$\sigma_2 : Zipcode, Time \rightarrow Country$$

Are the first two facts in a conflict set, or the first three, or all of the facts in one conflict set?

Persistent Repair vs Concrete Repair Persistent Repair are defined based on the minimal factorized instance. Though the authors in [14] have shown such a unique minimal factorized instance exists there is no discussion of the time complexity of obtaining such an instance. A persistent repair is a concrete repair, the reverse does not hold. The advantage of persistent repair is that it has the minimum cardinality.

4.5 Conclusion

In this chapter, we showed by using the switch point of a temporal database we can reduce the problem of temporal repair checking on infinite temporal database to temporal repair checking on finite temporal databases without losing the conflicts that are necessary for repair checking and even for building a repair. We showed in case of temporal functional dependencies, a reduction based on the switch point is lossless. In case of repairing a temporal database, we introduce an expansion function to get an infinite temporal repair. We also proposed two algorithms for temporal repair construction: one is based on time points and the other leverages time intervals.

CHAPTER 5

Experiments

The goal of this chapter is to show the performance of the normalization algorithms as well as the repair algorithms PntRCon and IntRCon on real-life and synthetic data. First we present the datasets we are using, then we compare the naïve normalization and $norm(I, \Phi)$. Finally we compare the repair algorithms. Note that we do not compare the repairs obtained by each algorithm because in our research we did not address any criteria to measure goodness of a repair in temporal databases.

5.1 Datasets

We have used both real-life and synthetic data. The following sub-sections describe the data and the tFD(s) we are using in the experiments.

5.1.1 Real-life Data

MIMIC-III (Medical Information Mart for Intensive Care III) [23] is a large, freely-available (upon request) database comprising de-identified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012. We use four tables that contain temporal information from this data : *TRANSFERS*, *ADMISSIONS*, *ICUSTAYS* and *PRESCRIPTIONS*. These data are available as csv files.

- TRANSFERS: Patient movement from bed to bed within the hospital, including ICU admission and discharge. Original cardinality = 261, 897 records.
- ADMISSIONS: Every unique hospitalization for each patient in the hospital. Original cardinality = 58,976 records.
- ICUSTAYS: Every unique ICU stay in the hospital in the database. Original cardinality: 61,532 records.
- PRESCRIPTIONS: Medications ordered for a given patient. Original cardinality: 4,156,450 records.

5.1.2 Pre-processing

TRANSFERS. We have removed the records with the same date as the start point and end point. We have also removed the records about discharging a patient because they do not have an end-point and replacing the endpoint with ∞ for discharge patients is not meaningful in this table. The cardinality of the table after pre-processing is 145, 277 records. We define the following temporal key constraint on this table:

$$\sigma_{transfers} = SUBJECT\_ID, INTIME, OUTTIME \rightarrow TRANSFERS$$

ADMISSIONS. We have removed the records of patients who are admitted and discharged on the same day. Each row of this table is given a unique number called *HADM_ID*. We ignore the uniqueness of *HADM_ID* and define the following temporal key constraint on this table:

$$\sigma_{admission} = SUBJECT\_ID, ADMITTIME, DISCHTIME \rightarrow ADMISSIONS$$

According to our result, there is a conflict in this table. There are two records of one patient that their time intervals overlap! **ICUSTAYS.** Each row of ICUSTAYS table is given a unique number called *ICUSTAY_ID*. We ignore this key and define the following temporal key constraint on this table:

$$\sigma_{icu} = SUBJECT\_ID, INTIME, OUTTIME \rightarrow ICUSTAYS$$

PRESCRIPTIONS. We have used this table to check the scalability of Algorithm IntRCon. The original table has more than 4 million records. We have removed the records without a start point (3182 records) and the records with the same date as the start point and endpoint. The records of patients in which the value of the start point is greater than the value of the end point are eliminated as well (15922 records). We have replaced the unknown end points with ∞. We define the following temporal functional dependency over this table: a patient cannot receive the same dose of a drug on the same day. That is:

$$\begin{aligned}\sigma_{pres} =& SUBJECT\_ID, FORMULARY\_DRUG\_CD, \\ & NDC, DOSE\_VAL\_RX, STARTDATE, ENDDATE \\ & \rightarrow ROUTE\end{aligned} \tag{5.1}$$

The attributes FORMULARY_DRUG_CD and NDC are related to drug scientific name and unique number respectively. The attribute ROUTE shows the route that the drug was given to the patient.

5.1.3 Synthetic Data

The relation schema is Emp(Name, Company, ZipCode, Country, Time). We assume in our domain that we have 10 names, 6 companies, and 5 countries and 22 zip codes. The start point of each tuple is considered to be a year between the years 1960-2020 and the endpoint is either a year greater than the start point and less than 2020 or ∞. The tables we generate are not normalized. The following tFDs (Σ_{emp}) are defined on this schema:

1. $Name,\ Time \rightarrow Company$
2. $ZipCode,\ Time \rightarrow Country$

5.2 Results

5.2.1 Implementation

The algorithms have been implemented in Java. Both the synthetic data and the algorithms are available online at github[1]. We have run each algorithm 20 times on an Intel Core i5-6200U CPU @ 2.30 GHZ $\times$ 4 system with 8GB of memory. The Java heap size is set to 8GB as well.

[1] `https://github.com/ladangol/IntRCon`

5.2.2 Naïve normalization vs. $norm(I, \Phi)$

Table 5.1 compares the run time and the size of the normalized instance obtained by running naïve normalization and $norm(I, \Phi)$ on synthetic and real datasets. Each algorithm is run for 20 times. For the synthetic data since the number of conflicts is high naïve normalization performs better. The size of the normalized instance obtained by $norm(I, \Phi)$ is the same as the the one obtained by naïve normalization (except for Emp100 which has lower number of conflicts) in the case of the synthetic data. However there is a huge difference in the size of the normalized instances obtained by these two algorithms when considering the real-life data with a lower number of conflicts.

5.2.3 Real-life data repair

We use the $norm(I, \Phi)$ algorithm for real-life data whenever necessary. Algorithm PntRCon cannot be run on any of the real-life datasets that we are using because it ran out of the heap space. Therefore, we select the Transfer table (because it has more conflicts than ADMISSIONS and ICUSSTAYS tables) and consider the first 10k, 20k, 30k and 40k records of this table. Table 5.2 shows the the run time of IntRCon on Admission and ICUStays table.

TRANSFERS table. The TRANSFERS table is already normalized w.r.t the above tkc because all the conflicts are between the records that have the same time interval (no overlaps). Also, there is no need to run the reduction and expansion functions on this table because there is no record with ∞ as time point. We have compared PntRCon and IntRCon on this table. The x-axis shows the different number of records of the TRANSFERS table, and y-axis show how long it took for the algorithms to find a repair (in milliseconds). Each algorithm has been run 20 times and the average of running times has been taken. The algorithm PntRCon runs out of memory when the number of records of the table is more than 40,000.

PRESCRIPTIONS table. Since this table has a large number of records, we used it to study the scalability of Algorithm IntRCon. As shown in Figure 5.2, the algorithm runs out of space when considering 1.5 million records.

5.2.4 Synthetic data repair

As shown in Figure 5.3, Algorithm PntRCon outperforms when the number of conflicts are high. We have used naïve normalization to normalize the tables before running Algorithm IntRCon.

db	Φ	norm(db, Φ)		Naïve Normalization		#of conflicts	# of tuples in conflicts
		size (records)	Time (ms)	size (records)	Time (ms)		
Emp100	Σ_{emp}	1479	14	3009	7.8	443	100
Emp500	Σ_{emp}	7427	31	7427	18.2	11190	500
Emp1k	Σ_{emp}	14846	50	14846	21.6	1118106	5000
Emp10k	Σ_{emp}	145,768	201.4	145,768	68.15	4,427,796	10000
Admission (Size = 57,879)	$\sigma_{admission}$	57,880	27,664	565,198	204	1	2
ICUStays (Size = 61,531)	σ_{icu}	61531	40,087	522,789	198	0	0
Transfer (Size = 145,277)	σ_{trans}	145,277	276,619	591,973	375	25	50
Prescription (Size=500k)	σ_{pres}	514,328	515,026	9,041,001	1369	6450	8472

Table 5.1: Naïve Normalization vs. $norm(I, \Phi)$

db	Size (# of records)	Time (sec)
ADMISSIONS	57,880	31.95
ICUSTAYS	61,531	35.61

Table 5.2: Run time of Algorithm IntRCon on ICUSTAYS and ADMISSIONS tables

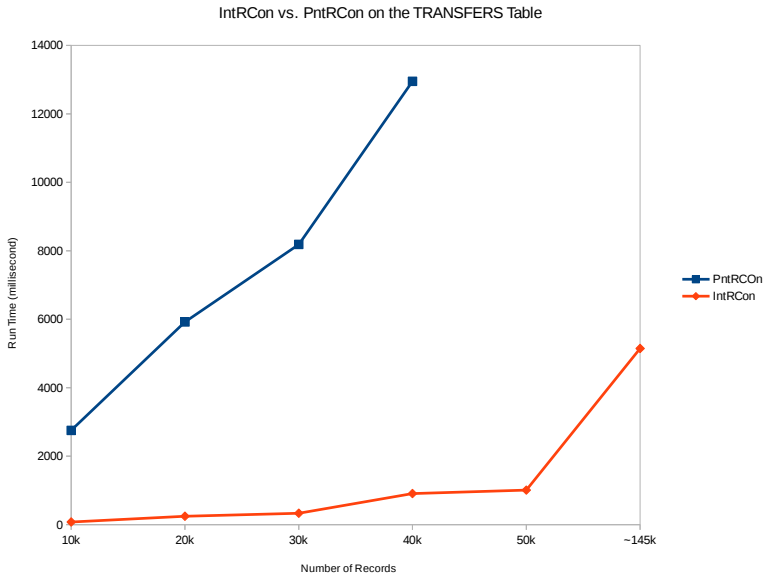

Figure 5.1: IntRCon vs PntRCon on the TRANSFERS table

Table	# of conflicts	# of tuple in the conflict
Emp50	93	47
Emp100	443	100
Emp500	11190	500
Emp1k	44335	1000
Emp5k	1118106	5000
Emp10k	4427796	10000

Table 5.3: Number of conflicts w.r.t. Σ_{emp} in the synthetic data

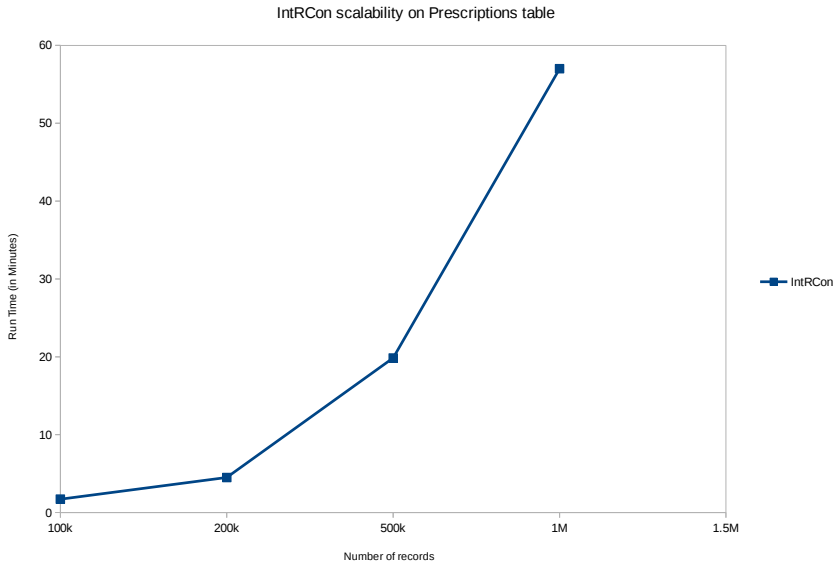

Figure 5.2: IntRCon on the PRESCRIPTIONS table

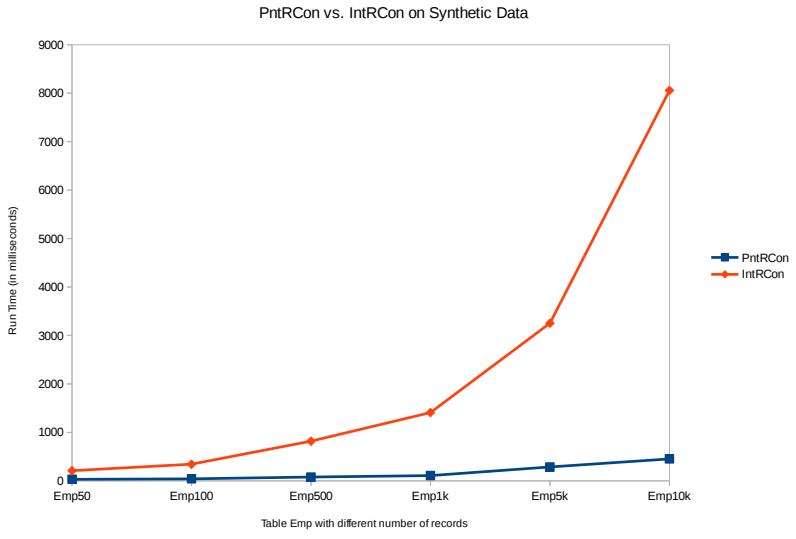

Figure 5.3: IntRCon vs PntRCon on the synthetic data with high number of conflicts

5.3 Discussion

Regarding the normalization algorithms, in the MIMIC dataset less than 1% of data are in conflicts. As Table 5.1 shows, if $norm(db, \Phi)$ is used, the size of the normalized instance is one order of magnitude less than an instance obtained by the naïïve normalization. The size of the normalized instance matters (as shown in Figure 5.2) because it is the input of the repair algorithms. On the other hand, the run time of the naïve normalization is orders of magnitude faster. The generated synthetic data intentionally had more conflicts. In such cases using naïve normalization is better because the size of normalized instance is the same (or almost the same) while the naïve normalization is faster.

Observe that the normalized output of the normalization algorithms is used as input for the repair algorithms. As shown in our experiments (Figure 5.1) and Figure 5.2), Algorithm PntRCon is not scaling well on big datasets because it stores a hash-map of the values of the lhs and rhs attributes of tFDs for each time point in memory. Depending on the tFD, some parallelization in the implementation might help which we left for future work. Algorithm PntRCon outperforms IntRCon when the number of conflicts are high (and the size of the dataset is small). The advantage of algorithm IntRCon is on datasets with less number of conflicts. We think most of the real-life data sets have lower number of mistakes compared to our synthetic data. For example in the PRESCRIPTIONS table, less than 1% of data has their start point bigger than their endpoint. As shown in Figure 5.2 on the prescription table Algorithm IntRCon cannot handle more than 1.5 million records. The algorithm uses a hash-map that stores the lhs and rhs attribute values of a tFD for each interval in a normalized instance. A parallel implementation of this algorithm might be possible depending on the tFDs.

CHAPTER 6

Future Work

Both temporal data exchange and temporal data repair can have their own extension which we discuss below. However, managing inconsistency in the context of data exchange has been explored in relational databases [36]. The classic data exchange does not handle the situations where there is no solution (that is result of the chase procedure is a failure). In [36] a new semantics for data exchange is suggested which is known as *exchange-repair semantics*. In the *exchange-repair semantics* all the source instances that are minimally different from the original source instance and have a solution are considered. One future direction is to adopt exchange-repair semantics in the context of temporal data with the building blocks that we introduced in this dissertation for temporal data exchange and temporal data repair.

6.1 Temporal Data Exchange

In the context of temporal data exchange a natural extension is to enrich the schema mappings such that they can express temporal phenomena. For example, temporal modal operators such as $\Diamond$ (*sometime in the future*), $\Box$ (*always in the future*), $\blacklozenge$ (sometime in the past) and $\blacksquare$ (*always in the past*) can be added to the language. For example, consider the following constraint which says every PhD graduate was a PhD candidate at some point before they graduate and they had a topic and an adviser.

$$\Box(\forall n\ PhDgrad(n) \rightarrow \blacklozenge \exists adv, top\ PhDCan(n, adv, top))$$

This constraint is equivalent to the following constraint in *two-sorted FOL (2-FOL)*[13]:

$$\forall n, t\ PhDgrad(n, t) \rightarrow \exists\ adv,\ top,\ t'\ PhDCan(n, adv, top, t') \wedge\ t' < t$$

At any snapshot db_ℓ if there is a fact about a PhD graduate in $PhDgrad$, then a snapshot db_i, $i < \ell$, should contain a fact about the PhD graduate in $PhDCan$ with a topic and an adviser. Considering these schema mappings, the notions of chase steps, solutions and universal solutions should be redefined. As an example if $\blacklozenge$ is used in the rhs of a dependency (such as the above dependency), is it enough to choose an arbitrary snapshot and generate facts according to the rhs of the dependency in that snapshot? What will be a universal solution in this case?

The schema mappings can also be enriched with linear order $<$ as in [2] and with arithmetic operations as in [37]. The linear order in [2] is interpreted over an arbitrary countable dense linear order without endpoints while the domain of time points that we consider in this dissertation is discrete linear order. Afrati et al. [2] conjecture that the results they obtained, regarding data exchange and query answering in presence of arithmetic operations, would change significantly in discrete ordered domains.

Another direction of research is to revisit the classical data exchange problems in the context of temporal databases such as the notion of *core* [17] and comparing open world assumption and closed world assumption [21, 31].

6.2 Temporal Data Repair

In the context of repair, the immediate extension is consideration of *denial constraints* [18]. A denial constraint is a first-order logic formula in the form

$$\forall \mathbf{x}, t \quad \neg(\phi(\mathbf{x}, t) \wedge \beta(\mathbf{x}, \mathbf{t}))$$

where β is a conjunction of atoms referring to built-in, arithmetic or comparison, predicates. Denial constraints are more expressive than functional dependencies. For example with denial constraints one can express the property that the salary of an employee cannot be reduced. Similarly to functional dependencies, a repair of a database that is inconsistent w.r.t. a set of denial constraints is obtained only by tuple deletion.

A lot of assumptions we made throughout Chapter 4 are not applicable for denial constraints. For example, consider the schema $E(name, worksfor, salary, time)$ and the following denial constraint that checks the monotonicity of the salary attribute over time (a temporal denial constraint):

$$\forall n, w, s, s', t, t' \ \neg(E(n, w, s, t) \wedge E(n, w, s', t') \wedge s < s' \wedge t > t')$$

Consider the database I_c with two facts that are in a conflict w.r.t. the denial constraint:

$$I_c : E(Ada, IBM, 70k, [7, \infty)), E(Ada, IBM, 60k, [7, \infty))$$

A reduction with $e^* = FP + 1 = 8$ is lossy in this case because for checking monotonicity of salary two distinct time points are needed and the reduced database has only one time point (i.e., time point 7). It seems that in order to find a lossless reduction for denial constraints not only the switch point but also the type of order constraint on temporal attributes in the constraint should be considered as well. Another difference between tFDs and denial constraints is that a single fact can be in a conflict w.r.t. a denial constraint, so it is not guaranteed that all intervals of the facts in the source database are in the repair as well (therefore, the switch point might change after a repair).

Another direction, is to consider consistent query answering [5] for temporal and non-temporal queries [14].

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Foto N. Afrati, Sara Cohen, and Gabriel M. Kuper. On the complexity of tree pattern containment with arithmetic comparisons. *Inf. Process. Lett.*, 111(15):754–760, 2011.

[3] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 31–41. ACM, 2009.

[4] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, New York, NY, USA, 2014.

[5] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

[6] Sebastian Arming, Reinhard Pichler, and Emanuel Sallinger. Complexity of repair checking and consistent query answering. In *ICDT*, volume 48 of *LIPIcs*, pages 21:1–21:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[7] Michael H. Böhlen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in temporal databases. In *VLDB*, pages 180–191. Morgan Kaufmann, 1996.

[8] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F. Naughton. Modeling entity evolution for temporal record matching. In *SIGMOD Conference*, pages 1175–1186. ACM, 2014.

[9] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F. Naughton. Tracking entities in the dynamic world: A fast algorithm for matching temporal records. *PVLDB*, 7(6):469–480, 2014.

[10] Jan Chomicki. Temporal query languages: A survey. In *ICTL*, volume 827 of *Lecture Notes in Computer Science*, pages 506–534. Springer, 1994.

[11] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[12] Jan Chomicki and David Toman. Temporal logic in information systems. In *Logics for Databases and Information Systems*, pages 31–70. Kluwer, 1998.

[13] Jan Chomicki and David Toman. Temporal databases. In *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 429–467. Elsevier, 2005.

[14] Jan Chomicki and Jef Wijsen. Consistent query answering for atemporal constraints over temporal databases. In *TIME*, pages 149–156. IEEE Computer Society, 2016.

[15] Xin Luna Dong and Wang-Chiew Tan. A time machine for information: Looking back to look forward. *PVLDB*, 8(12):2044–2045, 2015.

[16] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[17] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

[18] Terry Gaasterland, Parke Godfrey, and Jack Minker. An overview of cooperative answering. *J. Intell. Inf. Syst.*, 1(2):123–157, 1992.

[19] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[20] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Trans. Database Syst.*, 39(4):31:1–31:42, 2014.

[21] André Hernich, Leonid Libkin, and Nicole Schweikardt. Closed world data exchange. *ACM Trans. Database Syst.*, 36(2):14:1–14:40, 2011.

[22] Tomasz Imielinski and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[23] Pollard Tom J. Shen Lu Lehman Li-wei H. Feng Mengling Ghassemi Mohammad Moody Benjamin Szolovits Peter Anthony Celi Leo Mark Roger G. Johnson, Alistair E. W. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 2016. Data Descriptor.

[24] Manolis Koubarakis. Database models for infinite and indefinite temporal information. *Inf. Syst.*, 19(2):141–173, March 1994.

[25] Manolis Koubarakis. Foundations of indefinite constraint databases. In *Principles and Practice of Constraint Programming, Second International Workshop*, pages 266–280. Springer, 1994.

[26] Krishna Kulkarni and Jan-Eike Michels. Temporal features in SQL:2011. *SIGMOD Rec.*, 41(3):34–43, October 2012.

[27] Demian Lessa, Bharat Jayaraman, and Jan Chomicki. Temporal data model for program debugging. In *DBPL*, 2011.

[28] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. Linking temporal records. *Frontiers Comput. Sci.*, 6(3):293–312, 2012.

[29] Leonid Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69. ACM, 2006.

[30] Leonid Libkin. Incomplete data: what went wrong, and how to fix it. In *PODS*, pages 1–13. ACM, 2014.

[31] Leonid Libkin and Cristina Sirangelo. Open and closed world assumptions in data exchange. In *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[32] Xiao Ling and Daniel S. Weld. Temporal information extraction. In *AAAI*. AAAI Press, 2010.

[33] A matter of time: Temporal data management in DB2 10, 2012. `http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldata`.

[34] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.

[35] Partha Pratim Talukdar, Derry Wijaya, and Tom M. Mitchell. Coupled temporal scoping of relational facts. In *WSDM*, pages 73–82. ACM, 2012.

[36] Balder ten Cate, Richard L. Halpert, and Phokion G. Kolaitis. Exchange-repairs - managing inconsistency in data exchange. *J. Data Semantics*, 5(2):77–97, 2016.

[37] Balder ten Cate, Phokion G. Kolaitis, and Walied Othman. Data exchange with arithmetic operations. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 537–548, 2013.

[38] David Toman. Point vs. interval-based query languages for temporal databases. In *PODS*, pages 58–67. ACM Press, 1996.

[39] David Toman. Point-based temporal extension of temporal SQL. In *DOOD*, volume 1341 of *Lecture Notes in Computer Science*, pages 103–121. Springer, 1997.

[40] Yafang Wang, Maximilian Dylla, Marc Spaniol, and Gerhard Weikum. Coupling label propagation and constraints for temporal fact extraction. In *ACL (2)*, pages 233–237. The Association for Computer Linguistics, 2012.