A 1NF temporal relational model and algebra coping with valid-time temporal indeterminacy

Anselma, Luca; Piovesan, Luca; Terenziani, Paolo. A 1NF temporal relational model and algebra coping with valid-time temporal indeterminacy. JOURNAL OF INTELLIGENT INFORMATION SYSTEMS. 47 (3) pp: 345-374.
DOI: 10.1007/s10844-015-0367-2

Journal of Intelligent Information Systems manuscript No.
(will be inserted by the editor)

A 1NF temporal relational model and algebra coping with valid-time temporal indeterminacy

Luca Anselma · Luca Piovesan · Paolo Terenziani

Received: date / Accepted: date

Abstract In the real world, many phenomena are time related and in the last three decades the database community has devoted much work in dealing with "time of facts" in databases. While many approaches incorporating time in the relational model have been already devised, most of them assume that the exact time of facts is known. However, this assumption does not hold in many practical domains, in which temporal indeterminacy of facts occurs. The treatment of valid-time indeterminacy requires in-depth extensions to the current relational approaches. In this paper, we propose a theoretically grounded approach to cope with this issue, overcoming the limitations of related approaches in the literature. In particular, we present a 1NF temporal relational model and propose a new temporal algebra to query it. We also formally study the properties of the new data model and algebra, thus granting that our approach is interoperable with pre-existent temporal and non-temporal relational approaches, and is implementable on top of them. Finally, we consider computational complexity, showing that only a limited overhead is added, when moving from the determinate to the indeterminate case.

Keywords Relational Databases · Temporal data · Temporal Indeterminacy

1 Introduction

Given the relevance and diffusion of time-related issues in real-world phenomena, there has been much work over the last three decades in incorporating time into data models, query languages, and database management system (DBMS) implementations. In particular, research about relational temporal databases (henceforth TDBs) demonstrates that time is a peculiar aspect that deeply affects data

Luca Anselma · Luca Piovesan
Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy
E-mail: anselma|piovesan@di.unito.it

Paolo Terenziani
DISIT, Università del Piemonte Orientale "Amedeo Avogadro", Alessandria, Italy
E-mail: paolo.terenziani@mfn.unipmn.it

semantics, so that the addition of some attributes (e.g., START and END times) to relational tables is not enough, since many complex problems need to be tackled in designing, querying and modifying time-varying tables.

> "*Two decades of research into temporal databases have unequivocally shown that a time-varying table, containing certain kinds of DATE columns, is a completely different animal than its cousin, the table without such columns. Effectively designing, querying, and modifying time-varying tables requires a different set of approaches and techniques than the traditional ones taught in database courses and training seminars. ...*" (Snodgrass, 1999).

The TDB community has studied such techniques in over twenty years (Özsoyoglu and Snodgrass, 1995; Wu et al, 1997). Although TDB is still an open area of research, many researchers have already consolidated a "basic core" of results, by defining the TSQL2 consensus approach (Snodgrass, 1995). Nonetheless, the research in this area is still very active, since several challenging open problems need to be faced. One of them is temporal indeterminacy (i.e., "don't know exactly when" indeterminacy (Dyreson and Snodgrass, 1998)). While conventional TDB approaches assume that the exact time of occurrence of facts is known, this is not true in many application areas.

We illustrate some issues in dealing with temporal indeterminacy by drawing examples from the medical domain. For instance, temporal indeterminacy naturally involves all those facts that are subjective such as, for example, the onset of a symptom. Furthermore, it can also concern both those facts that, although objective, cannot be directly observed (see Example 1) and, in certain cases, also observable facts (see Example 2).

Example 1 Bill suffered from pneumonia between March 17^{th} (included) and April 2^{nd} (excluded), but, considering also the possible incubation and remission periods, he was possibly infected between February 26^{th} (included) and April 30^{th} (excluded)[1].

Example 2 In a clinical trial, the patients treated with the drug A manifested severe nausea on day 1 after the drug administration, and possibly also on days 2

[1] In Example 1, the temporal indeterminacy stems from the fact that the history of an infectious disease can be described under two different points of view: a clinical history and a bacteriological one. Roughly speaking, the clinical history corresponds to the evolution of the symptoms in the patient, and can be reasonably determined observing both subjective and objective parameters. On the other hand, the bacteriological history describes the entire life cycle of the presence of the pathogen in the host organism, starting from contagion and ending with the complete elimination of the agent. Taking into account the bacteriological history is of fundamental importance because, for example, patients can be infectious also in the absence of symptoms, or because they run the risk of suffering from a relapse also after the disappearance of the symptoms. However, unlike the clinical history, the bacteriological one often is not completely observable. In this case the physician can only make reasonable assumptions about it. For instance, at the time of diagnosis the physician can determine (observing, e.g., a pulmonary infiltrate) that Bill suffered from pneumonia from the appearance of the symptoms, occurred on March 17^{th}, until the disappearance of the symptoms on April 2^{nd}. These times represent the clinical history of the disease. Considering the incubation period of pneumonia and its remission period, the physician can also reasonably assume that the contagion started from 1 to 20 days before the symptom appearance and that Bill's body will completely eliminate the pneumonia pathogen in a time that may vary from the symptom disappearance to four weeks later.

and 3. In another group of patients, drug B caused severe nausea from day 1 to day 4 after the treatment and possibly on days 5 and 6[2].

Since temporal indeterminacy is pervasive in many application domains, many approaches have been devised to cope with it, e.g., within the Artificial Intelligence field (Vila, 1994; Allen, 1991; Emerson, 1990). However, in the area of relational databases, the number of approaches coping with temporal indeterminacy is more restricted (see the survey in (Jensen and Snodgrass, 2008)) and current approaches have several limitations, as widely discussed in Section 5 of this paper. In particular, to the best of our knowledge, our approach is the first one providing both a 1NF temporal relational model and an algebra to query it satisfying all the TDB desiderata discussed in Section 2.2.

We aim at overcoming the limitations of current literature, extending the relational approach to cope with temporal indeterminacy. We follow a methodology that is commonly used in the area of TDBs: we propose (i) a new data model to model temporally indeterminate relational data, and (ii) a new relational algebra to query it. However, our goal is not simply to devise an ad-hoc solution to the treatment of temporal indeterminacy, but also to propose a theoretically sound framework which is *implementable* on top of current technologies (in particular, on top of TSQL2 "consensus" approach) and *interoperable* with current frameworks (specifically, TSQL2 and standard non-temporal relational DBMSs), to grant that previous data can be maintained in the new framework. In order to guarantee the achievement of such objectives, we prove that our data model is a *consistent extension* of TSQL2 one (which, in turn, is a consistent extension of SQL), and our algebra is *reducible* to the standard relational one.

It is worth noticing that, although this paper uses examples of temporal indeterminacy derived from the medical field, the approach we propose is general and domain-independent. Indeed, this contribution is in the line of research that we have been following for several years, aiming at extending temporal relational databases to cope with new temporal phenomena, such as the telic/atelic distinction (Terenziani et al, 2007; Terenziani and Snodgrass, 2004), periodically repeated data (Terenziani, 2003; Stantic et al, 2012), proposal vetting (Anselma et al, 2013a), instantaneous events (Terenziani, 2013) and (semantics of) temporal indeterminacy (Anselma et al, 2013c), proposing general (domain-independent) and theoretically grounded solutions.

In order to cope with temporal indeterminacy, we first extend the data model, and then propose a new relational algebra to query it. It is worth stressing that, for the sake of generality, in our approach we have chosen to define our query language operating at the algebraic level. Indeed, SQL-like query languages are based on an underlying relational algebra. The definition of a new temporal SQL-like query language coping with temporal indeterminacy based on the temporal relational algebra we are going to define is not difficult, and will be one of the tasks of our future work.

Notably, the treatment of complex temporal phenomena in the relational model involves a major departure from traditional relational database techniques, and a

[2] In Example 2, a homogeneous group of patients is given a chemotherapeutic drug A and they exhibit severe (debilitating) nausea. In order to demonstrate the improvements that the drug A brings to the quality of life of patients, the analyst considers the presence of the nausea side effect also in a comparable group of patients treated with a different drug, drug B.

switch towards Artificial Intelligence (henceforth AI) techniques. The reason is that (relational) database techniques are geared towards the treatment of explicit data. On the other hand, when time is involved, a wide degree of implicit knowledge is usually involved. The treatment of implicit knowledge is outside the scope of traditional relational methodologies, while it is central for typical AI techniques. This is also the case of temporal indeterminacy. For instance, consider Example 1 above: at the level of granularity of days, it indeed implicitly represents the disjunctive data:
Bill was infected by pneumonia ***either*** *starting on February* 26^{th} *and ending on April* 2^{nd}, ***or*** *starting on February* 27^{th} *and ending on April* 2^{nd}, ***or*** *...,* ***or*** *starting on February* 26^{th} *and ending on April* 3^{rd}, *...,* ***or*** *starting on March* 17^{th} *and ending on April* 30^{th}.
(the disjunction would involve 580 terms). In this paper, we propose a compact (and implicit) representation of such disjunctive data, and this effort involves the adoption of AI symbolic manipulation techniques (see, e.g., the definition of union, intersection and difference operators on the valid times of tuples in Section 4 and the algorithms for computing the difference on the valid times in Appendix B).

Notably, our group has a tradition in the application of AI techniques to cope with complex temporal phenomena in relational databases (Terenziani, 2012; Anselma et al, 2013a,b; Terenziani, 2003; Terenziani and Snodgrass, 2004; Terenziani, 2013).

1.1 Organization of the paper

While the introduction just mentions the relational model and algebra and their extension to cope with valid-time indeterminacy, Section 2 is a brief introduction to such issues, to provide readers who are not familiar with such topics with the necessary background (of course, this section can be ignored by experts in the relational approach and its temporal extensions). Sections 3 and 4 constitute the core of our paper. Section 3 introduces our temporal data model, which extends TSQL2 "consensus" one to deal with temporal indeterminacy, and analyzes its properties. In Section 4 we define a temporal relational algebra to query our new data model and we study its properties. Section 5 discusses related works and Section 6 contains conclusions. The formal proofs of the properties in Sections 3 and 4 are reported in Appendix A and an algorithm for computing the difference between indeterminate valid times is in Appendix B. Finally, in Appendix C, we present a motivating comparison between our approach and an existing one.

2 Preliminaries

2.1 The relational temporal model

In many real-world domains, time has a pervasive nature and it may be represented as a particular kind of information. However, since the early 1980s, it is clear that the treatment of time in the relational model cannot be simply reduced to the addition of two time attributes (i.e., start time and end time) to the usual relational schema. This solution could suffice if only data representation is considered.

Relational databases also involve querying data and, considering querying, time behaves differently from the other attributes. In fact, over twenty years of research in temporal relational databases have clarified that the treatment of time in the relational approach involves the solution of difficult problems, and the adoption of advanced dedicated techniques (Snodgrass, 1999). In this spirit, many extensions to the standard relational model were devised (Özsoyoglu and Snodgrass, 1995; Jensen and Snodgrass, 2008; McKenzie and Snodgrass, 1991; Tansel et al, 1993; Jensen and Snodgrass, 1999), and more than 2000 papers on TDBs were published over two decades (Wu et al, 1997).

In the following, we use TSQL2 as a basis to introduce some of the basic issues relevant to the approach we propose in this paper. TSQL2 (Snodgrass, 1995) is probably the most known temporal database approach. It has been defined as a consensus of many researchers all over the world. Recently, several commercial tools have implemented (at least in part) the TSQL2 approach, and its SQL/Temporal evolution (see `http://www.cs.arizona.edu/~rts/sql3.html`).

TSQL2 deals with both valid time and transaction time. For the sake of brevity, in the following we focus on valid time only[3]. In TSQL2 tuples are associated with *valid time*. For valid time, a limited precision is assumed and the *chronon* is the basic time unit. The time domain is totally ordered and isomorphic to the subsets of the domain of natural numbers. The domain of valid times D_{VT} is given as a set $D_{VT} = \{t_1, \ldots, t_k\}$ of chronons. The schema of a valid-time relation $R = (A_1, \ldots, A_n | VT_s, VT_e)$ consists of an arbitrary number of non-timestamp attributes $A_1, \ldots, A_n$ encoding some fact, and of two timestamp attributes VT_s and VT_e with domain D_{VT}. Thus, a tuple $x = (a_1, \ldots, a_n | t_s, t_e)$ in a valid-time relation $r(R)$ on the schema R (henceforth called a (TSQL2) temporal tuple) consists of a number of attribute values associated with two chronons $t_s, t_e \in D_{VT}$. The intended meaning of a temporal TSQL2 tuple is that the recorded fact is *true in the modeled reality* during each valid-time chronon c, $t_s \leq c \leq t_e$.

For instance, Example 3 (in the following) can be represented into the TSQL2 formalism because, at the granularity of days, the presence of symptoms may be considered temporally determinate. Its tabular representation is shown in Table 1.

Example 3 Bill manifested high fever from April 10^{th} (included) until April 15^{th} (excluded).

Notation and terminology. Given a tuple x defined on the schema $R = (A_1, \ldots, A_n | VT_s, VT_e)$, we denote with A the set of attributes $\{A_1, \ldots, A_n\}$. Then $x[A]$ denotes the values in x of the attributes in A. $x[VT_s]$ and $x[VT_e]$ denote the starting and ending time respectively, and $x[VT]$ denotes the pair of starting

[3] In many TDB approaches two independent time dimensions have been identified, namely transaction time and valid time. *Valid time* represents the time when the fact described by a tuple holds in the modeled world. *Transaction time* represents the time when a tuple is present in the database. Temporal indeterminacy may only concern valid time, since transaction time (i.e., the database insertion/deletion time) is always known in an exact way. As a consequence, in this paper, we just focus on valid time. Extensions to cope also with transaction time are easy since transaction time can be coped with as in the other TDB approaches, e.g., as in TSQL2.

Table 1 Relation $SYMPTOMS\_DET$ (TSQL2 representation of Example 3).

Patient	**Symptom**	$\mathbf{VT_s}$	$\mathbf{VT_e}$
Bill	High Fever	Apr 10^{th}	Apr 14^{th}

and ending time. As in TSQL2, we call $A_1, \ldots, A_n$ ***explicit*** attributes, and VT_s and VT_e ***implicit*** attributes. We call ***value-equivalent*** two or more tuples having the same values for the explicit attributes.

Notice that TSQL2 has a **First Normal Form** (1NF) representation of temporal data. A relation is in 1NF if it has the property that none of its domains has elements which are themselves sets (Codd, 1971). The 1NF grants that the values of each attribute can be stored in a fixed and predefined amount of space. The main issue, however, is not the definition of the relational representation, but the definition of the query language. Both an extension of SQL and of Codd's algebra have been proposed (in the following, we just focus on the temporal algebra). *Algebraic* operators have been defined on the temporal model as a temporal extension of Codd's operators. However, in TSQL2, the representation formalism has been designed in such a way that it was possible to define a temporal algebra satisfying several fundamental properties. In particular, two of them are of paramount importance.

The first property is **closure**. Informally, a data model is closed under an operator if the result of the application of the operator to the elements of the model is still an element belonging to the model. In the context of determinate temporal databases, closure entails that algebraic operators work on determinate temporal relations and provide as output *determinate temporal* relations as well. Supporting such a property involves defining, for each algebraic operator, how the (start and end of the) valid time of the output tuples is obtained, on the basis of the valid time of the input tuples.

A much easier solution could have been to provide users with some operator to remove time (e.g., an operator selecting all and only the tuples holding at a given time point), and then to provide them with standard non-temporal Codd's operators. However, such a solution could not have granted the closure property (in fact, the output is not a temporal relation, but a non-temporal one), and it is strictly less expressive than the temporal algebra. For instance, Queries 1 and 2 in the following **cannot** be properly answered using such a simplified approach.

Let us consider, for instance, the situation described by Examples 1 and 3. Bill suffered from high fever after the disappearance of the pneumonia symptoms and fever is a symptom of many infections, including pneumonia. Thus, the physician might wonder whether the fever could have been caused by a pneumonia relapse and (s)he asks to the system Query 1.

Query 1 *Might Bill have had high fever not during the pneumonia infection? If so, when?*

Considering Example 2, to demonstrate the improvements that the drug A brings to the quality of life of patients with respect to drug B, the analyst can express Query 2.

Query 2 *When (i.e., in which days) did nausea occur in patients treated with drug B and not in patients treated with drug A?*

A second fundamental property which must be supported by any temporal relational approach is **reducibility**. Such a property is essential in order to grant that, when time is disregarded, TSQL2 algebraic operators behave like standard SQL ones. This property grants "continuity" for users, facilitating a cost-effective migration to a temporal model:

> *"Reducibility aims to protect the investment in programmer training and to ensure continued efficient, cost-effective application development upon migration to a temporal model"* (Jensen and Snodgrass, 2008).

As a remarkable side-effect, reducibility also grants for the fact that TSQL2 can be implemented as an additional layer on top of the SQL model and, indeed, several prototypical implementations have been already devised following such a strategy.

Additionally, in TSQL2, as well as in most temporal models, a temporal relation is modeled as a set of temporal tuples, each one consisting of a "conventional" non-temporal component paired with a temporal component, modeling its valid and/or transaction time. Thus, a "conventional" non-temporal relation can be seen as a degenerate temporal one, in which the temporal component is empty, and reducibility grants that query operators behave in the "conventional" way on them, thus making interoperability with non-temporal approaches feasible.

To achieve both closure and reducibility, as in most approaches in the TDB literature (see, e.g., the survey in (McKenzie and Snodgrass, 1991)), in TSQL2 temporal algebraic operators behave like standard non-temporal ones on the explicit attributes, and involve the application of set operators on the implicit attributes. More precisely, temporal relational difference performs difference on the temporal attributes of value-equivalent tuples and temporal natural join performs the intersection of temporal attributes. This definition can be also motivated by the *sequenced* semantics (Dunn et al, 2002): the results of algebraic operations should be valid independently at each point of time. For instance in TSQL2, given two relations r and s with schema $(A_1, \ldots, A_n | VT_s, VT_e)$, the temporal difference between r and s is defined as follows:

$$r -^V s = \{z \backslash \exists x \in r(z[A] = x[A] \wedge \exists t \in cover^V(chr(x[VT]) - \{chr(y[VT]) \setminus y \in s \wedge y[A] = x[A]\}) \wedge z[VT_s] = min(t) \wedge z[VT_e] = max(t))\}$$

In the TSQL2 definition of difference, each tuple x in the minuend is compared with all its value-equivalent tuples y in the subtrahend $(\cdots \setminus y \in s \wedge y[A] = x[A])$. The representations into sets of chronons of their valid times are obtained through the *chr* function, then a set difference is performed between the set representing x and those derived from all the tuples y $(chr(x[VT]) - \{chr(y[VT]) \setminus \ldots\})$. Finally, the $cover^V$ function partitions the resulting set in such a way that each element of the partition is convex (i.e., it contains all the chronons included between its minimum and maximum ones), and for each element of the partition a tuple value-equivalent to x and with VT_s and VT_e corresponding to the minimum and maximum chronons of the partition element is included in the result.

For example, assuming to represent data about Bill's pneumonia in a TSQL2 relation named $DISEASES\_DET$[4] with schema $(Patient, Disease | VT_s, VT_e)$,

[4] In the relation $DISEASES\_DET$ we consider determinate time only. Thus, the valid time of the tuple regarding Bill's pneumonia will be referred only to the clinical history of the disease.

data regarding Example 1 can be expressed with the tuple $(Bill, Pneumonia|Mar17^{th}, Apr1^{st})$ and Query 1 can be expressed in TSQL2 algebra as follows:

Query 1 (TSQL2 algebra).

$$\pi^{V}{}_{Patient}(\sigma^{V}{}_{\substack{Patient=Bill \wedge \\ Symptom=High\,Fever}}(SYMPTOMS\_DET))$$
$$-^{V}$$
$$\pi^{V}{}_{Patient}(\sigma^{V}{}_{\substack{Patient=Bill \wedge \\ Disease=Pneumonia}}(DISEASES\_DET))$$

The first selection operation selects tuples concerning Bill's high fever and the second finds out tuples concerning Bill's pneumonia infection. Projection is applied to both results to retain only the *Patient* attribute. Finally, the temporal difference can be applied, determining when Bill had high fever but not pneumonia (i.e., on days 10^{th}, 11^{th}, 12^{th}, 13^{th} and 14^{th} of April). Notice that, since TSQL2 data model is closed under the relational algebraic operators, the time of the results of the above operations is retained.

2.2 Implications on our approach

In this paper, following the above discussion and in line with the mainstream of the research in relational TDB, we aim at:

1. devising a **1NF relational data** (representation) **model** and
2. defining a **relational algebraic query language** operating on it
3. for which the property of **closure** holds
4. and the property of **reducibility** holds.

It is worth noticing that our approach is clearly different from existent AI approaches (see, e.g., the surveys (Vila, 1994; Allen, 1991; Emerson, 1990)), since it aims at very different objectives. In particular, differently from AI approaches, our goal is to provide a relational model and to extend Codd's relational algebra operators to cope with temporal indeterminacy. This entails some issues, including representing data in 1NF and devising an algebra that grants closure and reducibility. In addition, our approach is different also from other existing ones in the field of TDBs, since many of them focus on the model and they do not provide a closed algebra to query it (e.g., (Snodgrass, 1995; Das and Musen, 1994)), or they do not provide a model suitable for the implementation on a relational database (e.g., the semantic, abstract, not in 1NF representations in (Anselma et al, 2013c)).

3 Relational Data Model to deal with temporal indeterminacy

We adopt a temporal ontology commonly used in several TDBs (Snodgrass, 1995; Jensen and Snodgrass, 1996, 2008), and, specifically, in TSQL2: time is discrete, linearly ordered and isomorphic to the integers.

Definition 1 (Chronon) The chronon is the basic time unit. The chronon domain T^C, also called timeline, is the totally ordered set of chronons $\{\dots, c_i, \dots, c_j, \dots\}$, with $c_i < c_j$ as $i < j$.

We propose a representation for temporally indeterminate valid times where one can specify a (possibly empty) time interval in which the fact certainly holds, and a time interval in which it may hold.

Definition 2 (Indeterminate Temporal Element (ITE)) An ITE is a pair of time intervals[5] $\langle [d_s, d_e), [i_s, i_e) \rangle$. d_s, d_e, i_s and i_e are chronons such that $i_s \leq d_s \leq d_e \leq i_e$ and $i_s < i_e$. Thus, notice that $[d_s, d_e)$ may be empty (when $d_s = d_e$), while $[i_s, i_e)$ cannot (in fact $i_s < i_e$). Moreover, $[d_s, d_e)$ must be contained in $[i_s, i_e)$.

Intuitively, the interval $[d_s, d_e)$ (henceforth *determinate time interval*) represents the times in which a fact *certainly* holds; the interval $[i_s, i_e)$ (henceforth *indeterminate time interval*) represents the times when the represented fact *may* hold. Since all the times in which a fact certainly holds are also times when it possibly holds, in our representation the indeterminate time interval of an ITE always contains the determinate time interval of that ITE.

Since each interval can be represented by a pair of temporal attributes corresponding to its endpoints (consider, e.g., (Snodgrass, 1995)), temporal indeterminacy is represented by a quadruple of temporal attributes over T^C.

Considering a granularity of days, the ITE corresponding to Example 1 is $\langle [Mar\,17^{th}, Apr\,2^{nd}), [Feb\,26^{th}, Apr\,30^{th}) \rangle$ and the ITEs corresponding to Example 2 are $\langle [1, 2), [1, 4) \rangle$ for drug A and $\langle [1, 5), [1, 7) \rangle$ for drug B.

Definition 3 (Temporally indeterminate tuples and relations) Given a schema $(A_1, \dots, A_n)$ where each A_i represents a non-temporal attribute on the domain Dom_i, a (valid-time) indeterminate relation r is an instance of the schema $(A_1, \dots, A_n | D_s, D_e, I_s, I_e)$ over the domain $Dom_1 \times \cdots \times Dom_n \times T^C \times T^C \times T^C \times T^C$. Each tuple $x = (v_1, \dots, v_n | d_s, d_e, i_s, i_e) \in r$ is termed a (valid-time) temporally indeterminate tuple. The temporal component (d_s, d_e, i_s, i_e) of tuple x represents the ITE $\langle [d_s, d_e), [i_s, i_e) \rangle$ corresponding to the fact that the tuple x *certainly holds* in the interval $[d_s, d_e)$ and *possibly holds* in the interval $[i_s, i_e)$.

Tables 2 and 3 represent Examples 1 and 2, assuming in both cases the granularity of days.

Table 2 Relation $DISEASES^{TI}$ representing Example 1.

Patient	Disease	$\mathbf{D_s}$	$\mathbf{D_e}$	$\mathbf{I_s}$	$\mathbf{I_e}$
Bill	Pneumonia	Mar 17^{th}	Apr 2^{nd}	Feb 26^{th}	Apr 30^{th}

[5] As in many TDB approaches, for the sake of convenience time intervals $[c_s, c_e)$ are closed on the left and open on the right (i.e., their left bound is included and their right bound is excluded). Notice, however, that our approach is not dependent on such a choice.

Table 3 Relation $SIDE\_EFFECTS^{TI}$ representing Example 2.

Drug	Side_Effect	D_s	D_e	I_s	I_e
A	Severe Nausea	1	2	1	4
B	Severe Nausea	1	5	1	7

3.1 Properties of the data model

The data model we propose is a *consistent extension* of the data model of TSQL2 (which, in turn, is a consistent extension of the standard non-temporal relational data model (Snodgrass, 1995)). Such a property depends on the fact that ITEs can represent determinate time (i.e., the time represented in TSQL2) as a special case[6].

Property 1 (Consistent extension (ITEs)) Determinate temporal elements can be modeled by ITEs of the form $\langle [d_s, d_e), [d_s, d_e) \rangle$, i.e., an ITE having the same determinate and indeterminate interval. Determinate temporal relations can be modeled by temporally indeterminate relations in which each tuple is associated with an ITE of the form $\langle [d_s, d_e), [d_s, d_e) \rangle$.

Table 4, for instance, represents the determinate-time interval of Example 3 (see also the TSQL2 representation in Table 1).

Table 4 Relation $SYMPTOMS^{TI}$ representing Example 3.

Patient	Symptom	D_s	D_e	I_s	I_e
Bill	High Fever	Apr 10^{th}	Apr 15^{th}	Apr 10^{th}	Apr 15^{th}

The above property grants that temporally determinate data can be still modeled in our data model (thus granting interoperability with previous data).

4 Relational algebra to deal with temporal indeterminacy

We propose a query language for our data model. For the sake of clarity and generality, we have chosen to operate at the algebraic level. Codd (1972) designated as complete any query language that was as expressive as a set of relational algebraic operators: relational union ($\cup$), relational difference ($-$), selection (σ_P), projection (π_X), and Cartesian product ($\times$). We generalize these operators to cover (valid-time) temporally indeterminate data. We define our relational algebraic operators as an extension as close as possible to TSQL2 and current TDB models, to grant interoperability.

For the sake of clarity, we adopt the following concise notation for ITEs.

Notation and terminology. Given a tuple $x = (v_1, \ldots, v_n | d_s, d_e, i_s, i_e)$, $\langle d, i \rangle$ represents its temporal component, where d stands for the determinate time interval $[d_s, d_e)$ and i stands for the indeterminate time interval $[i_s, i_e)$.

[6] Of course, more efficient implementations of determinate-time relations (with just two temporal attributes) can be easily provided.

As in several TDB models, our temporal operators behave as standard non-temporal operators on the non-temporal attributes (this is important to achieve the property of reducibility, discussed henceforth), and apply set operators on the temporal component of tuples (consider, e.g., (Snodgrass, 1995)) in the case of difference and Cartesian product. In Figure 1 we define the relational operators of union ($\cup^{TI}$), difference ($-^{TI}$), projection (π^{TI}_X), selection (σ^{TI}_P) and Cartesian product ($\times^{TI}$) between temporally indeterminate relations. Relational union $r \cup^{TI} s$ reports in output all the tuples belonging to r or to s. Analogously, projection maintains only the values for the specified attributes X, leaving the rest of the tuple (including the temporal component) unchanged. Non-temporal selection applies the selection predicate to the non-temporal component of each tuple. On the other hand, Cartesian product involves the intersection of the temporal components (ITEs), and difference involves the difference of temporal components (ITEs). From a theoretical point of view, such a choice is motivated by the *sequenced* semantics (Dunn et al, 2002): results should be valid independently at each point of time. Supporting the sequenced semantics is a crucial step to grant the reducibility property.

In Figure 2 we define the intersection and difference operators between ITEs. For the intersection, the determinate part of the result is obtained by the intersection between the determinate parts of the considered ITEs ($d \cap d'$), while the indeterminate one is the intersection of the indeterminate parts ($i \cap i'$).

The definition of difference deserves a deeper analysis, since it is, as in many TDB approaches, the most complex and challenging operator. In the difference, there are two possible cases. First, a tuple $(v|\langle d, i\rangle)$ belongs to r, and there are no tuples in s value-equivalent to $(v|\langle d, i\rangle)$ (i.e., no tuple in s has the same non-temporal component v as $(v|\langle d, i\rangle)$). In such a case, the tuple $(v|\langle d, i\rangle)$ is reported in the output. Second, given a tuple $(v|\langle d, i\rangle)$ belonging to r, there are some tuples $(v|\langle d_1, i_1\rangle), \ldots, (v|\langle d_k, i_k\rangle)$ in s value-equivalent to it. In such a case, the output must have as valid time those chronons that are in $\langle d, i\rangle$ but not in $\langle d_1, i_1\rangle, \ldots, \langle d_k, i_k\rangle$, i.e., the difference between the ITE $\langle d, i\rangle$ and the set of ITEs $\{\langle d_1, i_1\rangle, \ldots, \langle d_k, i_k\rangle\}$ must be computed (see Figure 2). As we have already men-

$$r \cup^{TI} s = \{(v|\langle d, i\rangle) \setminus (v|\langle d, i\rangle) \in r \vee (v|\langle d, i\rangle) \in s\}$$

$$\pi^{TI}_X(r) = \{(v|\langle d, i\rangle) \setminus \exists (v_1|\langle d_1, i_1\rangle) \in r \wedge v = v_1[X] \wedge \langle d, i\rangle = \langle d_1, i_1\rangle\}$$

$$\sigma^{TI}_P(r) = \{(v|\langle d, i\rangle) \setminus (v|\langle d, i\rangle) \in r \wedge P(v)\}$$

$$r \times^{TI} s = \{(v_r \cdot v_s|\langle d, i\rangle) \setminus \exists \langle d_r, i_r\rangle, \langle d_s, i_s\rangle((v_r|\langle d_r, i_r\rangle) \in r \wedge (v_s|\langle d_s, i_s\rangle) \in s \wedge \langle d, i\rangle = \langle d_r, i_r\rangle \cap^{ITE} \langle d_s, i_s\rangle \wedge i \neq \emptyset)\}$$

$$\begin{aligned} r -^{TI} s = \{&(v|\langle d, i\rangle) \setminus (\exists \langle d_r, i_r\rangle((v|\langle d_r, i_r\rangle) \in r \wedge \nexists \langle d_s, i_s\rangle((v|\langle d_s, i_s\rangle) \in s \wedge \langle d, i\rangle = \langle d_r, i_r\rangle)))\vee \\ &(\exists \langle d_r, i_r\rangle((v|\langle d_r, i_r\rangle) \in r \wedge \exists!(v|\langle d_1, i_1\rangle), \ldots, (v|\langle d_k, i_k\rangle))((v|\langle d_1, i_1\rangle) \in s, \ldots, (v|\langle d_k, i_k\rangle) \in s \wedge \\ &\langle d, i\rangle = \langle d_r, i_r\rangle -^{ITE} \{\langle d_1, i_1\rangle, \ldots, \langle d_k, i_k\rangle\} \wedge i \neq \emptyset)))\} \end{aligned}$$

$$\sigma^{TI}_{CERT_\phi}(r) = \{(v|\langle d, i\rangle) \setminus (v|\langle d, i\rangle) \in r \wedge \phi(d)\}$$

$$\sigma^{TI}_{POSS_\phi}(r) = \{(v|\langle d, i\rangle) \setminus (v|\langle d, i\rangle) \in r \wedge \phi(i)\}$$

Fig. 1 Definition of temporally indeterminate relational operators. In $(v|\langle d, i\rangle)$, v and $\langle d, i\rangle$ stand for the non-temporal component of a tuple and its associated ITE, respectively. The uniqueness quantifier $\exists!$ is used to identify the tuples value-equivalent to $(v|\langle d_r, i_r\rangle)$.

ITE intersection. $\langle d, i\rangle \cap^{ITE} \langle d', i'\rangle = \langle d \cap d', i \cap i'\rangle$
ITE difference. $\langle d, i\rangle -^{ITE} \{\langle d'_1, i'_1\rangle, \ldots, \langle d'_k, i'_k\rangle\} = cover(chr(d) - (chr(i'_1) \cup \cdots \cup chr(i'_k)), chr(i) - (chr(d'_1) \cup \cdots \cup chr(d'_k)))$.

$$
\begin{aligned}
&chr([c_s, c_e)) = \{c \in T^C \setminus c_s \leq c < c_e\}\\
&isConvex(s) \iff \nexists c \in T^C (min(s) \leq c \leq max(s) \wedge c \notin s)\\
&maximal(S) = \{s \setminus s \subseteq S \wedge isConvex(s) \wedge \nexists s' \subseteq S (isConvex(s') \wedge s \subset s')\}\\
&partition(i, \{d_1, \ldots, d_k\}) = \{\langle d_j, i_j\rangle \setminus\\
&\qquad d_j \in \{d_1, \ldots, d_k\} \wedge d_j \subseteq i_j \wedge \nexists d_h \in \{d_1, \ldots, d_k\} (d_h \neq d_j \wedge d_h \cap i_j \neq \emptyset) \wedge i_1 \cup \cdots \cup i_k = i \wedge\\
&\qquad i_i \cap i_2 = \emptyset \wedge \cdots \wedge i_i \cap i_k = \emptyset \wedge \cdots \wedge i_{k-1} \cap i_k = \emptyset \wedge isConvex(i_1) \wedge \cdots \wedge isConvex(i_k)\}\\
&cover(D, I) = \{\langle \emptyset, [min(i), max(i) + 1)\rangle \setminus i \in maximal(I) \wedge \nexists c \in D (c \in i)\} \cup\\
&\{\langle [min(d'), max(d') + 1), [min(i'), max(i') + 1)\rangle \setminus\\
&\exists i \in maximal(I) (\langle d', i'\rangle \in partition(i, \{d \setminus d \in maximal(D) \wedge d \subseteq i \wedge d \neq \emptyset\}))\}
\end{aligned}
$$

Fig. 2 Definition of intersection and difference between ITEs.

tioned in Section 2, concerning TSQL2 and determinate time, in general the result of difference can be expressed in alternative (but snapshot equivalent Snodgrass (1995)) ways, depending on the covering function (abstractly indicated by "cover" in TSQL2 definition). This, is, a fortiori, true also in case indeterminate time is considered. Thus, in this paper, we operate in two steps. First, for the sake of generality, we propose an abstract definition of difference, generalizing TSQL2 one to cope with indeterminate time. It describes in an abstract and formal way all possible solutions to temporal difference. To do so, as in TSQL2 definition, it resorts to the conversion of temporal intervals to\from set of chronons, and is parametric over a covering function. Additionally, the conditions that the covering functions must respect are explicitly detailed. As a second step, in Appendix B, we propose a concrete and efficient implementation of such a definition. To do so, we choose a specific covering policy, and we directly operate on time intervals (with no conversions from\to chronons).

More in detail, the *chr* function computes the set of chronons belonging to a temporal interval expressed by its starting and ending chronon. The *cover* function computes a set of ITEs from a set D of determinate chronons and a set I of indeterminate chronons. It uses some auxiliary functions. The *isConvex* function is true if and only if the chronons in a set are contiguous. The *maximal* function, given a set S of chronons, produces the set of the maximal convex sets s of chronons. The *partition* function, given a convex set i of indeterminate chronons and given a set of convex non-intersecting sets $d_1, \ldots, d_k$ of determinate chronons such that $d_1, \ldots, d_k$ are contained in i (see the *cover* function), produces a set $\{\langle i_1, d_1\rangle, \ldots, \langle i_k, d_k\rangle\}$ of pairs $\langle i_j, d_j\rangle$, where i_j and d_j are sets of chronons, such that:

1. $\{i_1, \ldots, i_k\}$ is a partition of i;
2. each set of chronons $i_j (1 \leq j \leq k)$ is convex;
3. each set of chronons i_j contains all chronons in d_j, and no other chronon in $d_h, h \neq j, d_h \in \{d_1, \ldots, d_k\}$.

It is worth noticing that, in general, there are different ways of determining the sets i_j above. As a consequence, different policies can be adopted, and different imple-

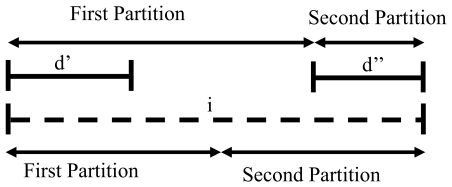

Fig. 3 Different types of partition policies

mentations can be devised for partitioning, each one providing a different correct result. Our approach is independent of the specific partitioning policy, provided that partitioning satisfies the specifications above. Two different partitioning policies are shown in Figure 3. In the figure we represent an indeterminate convex set of chronons i and two determinate ones d' and d''. For the sake of simplicity, we represent convex sets as intervals. Following the definition above, the only parts of i which are restricted to belong to two different partitions are $i \cap d'$ and $i \cap d''$, while the remaining part has no restrictions. For example, in the partitioning policy in the upper part of the figure all the remaining part of i is added to the first partition, while the partitioning policy in the lower part of the figure equally splits such interval between the two partitions. The first policy is the one adopted by the algorithm of difference in B.

The *cover* function provides as output the union of two sets of ITEs. The first set corresponds to the indeterminate intervals which do not contain determinate chronons (thus, the resulting ITEs have an empty determinate component). The indeterminate components are the maximal convex intervals covering such indeterminate chronons. Notice that, since the intervals are represented as closed on the left and open on the right, we convert the representation from a convex set of chronons to an interval by means of the functions *min*, *max* and *increment*[7]. The second set corresponds to the indeterminate intervals that contain determinate chronons. Given the maximal convex sets i of chronons in I, each i is partitioned by means of the *partition* function in such a way that each element i' of the partition contains exactly one maximal convex set of determinate chronons (i.e., a determinate component).

Finally, we can define the difference between an ITE and a set of ITEs. The determinate chronons of the difference (i.e., the chronons when the fact certainly holds) are the determinate chronons of the minuend that are not chronons (either determinate or indeterminate) of the subtrahend. The indeterminate chronons of the difference (i.e., the chronons when the fact may hold) are the indeterminate chronons of the minuend except the determinate chronons of the subtrahend.

Indeed, a temporal chronon must be present in the determinate interval of the difference only if it is determinate in the minuend, and it is not present in the determinate or indeterminate intervals of the subtrahend. On the other hand, a chronon must be present in the indeterminate intervals of the difference only if it is determinate or indeterminate in the minuend, and is not present in the determinate intervals of the subtrahend. Notice that, if a chronon is determinate or indeterminate in the minuend, and is indeterminate in the subtrahend, it must be

[7] The *min* and *max* functions have the obvious meanings. The increment function can be defined as $c + 1 = c' \in T^C \setminus c' > c \wedge \nexists c'' \in T^C (c < c'' < c')$.

indeterminate in the difference: indeed, since it is only possible in the subtrahend, it may be the case that it is present as a chronon in the subtrahend, thus it must be included in the indeterminate interval of the difference.

Our definition above of the temporal extension of Codds operators is equal to the one adopted by several TDB models (including TSQL2), except for the fact that we operate on ITEs instead than on determinate time intervals. In particular, our temporal operators behave as standard nontemporal operators on the non-temporal attributes. As in TSQL2, only Cartesian product and difference operate on the temporal part of tuples. As a consequence, the complexity of the other algebraic operators (selection, projection, and union) is the same in our approach to indeterminate time, in such approaches to determinate time (including TSQL2), and in the standard nontemporal (i.e., Codds) operators. As in most TDB approaches (including TSQL2), our Cartesian product and difference operate on the temporal part of tuples, performing intersection (between the times of the paired tuples) and difference (between the time of each tuple in the first relation, and the times of all the tuples in the second relation that are value-equivalent to it). Thus, there is no difference regarding accesses to the secondary memory (which is, obviously the most important aspect to be considered) between our approach to indeterminate time and most approaches (including TSQL2) to determinate time. The only difference regards the operations, performed in main memory, on the temporal parts of the tuple. However, the same operations (intersection or difference) between the temporal parts [illegible] have to be performed, both in the determinate and in the indeterminate case. The only difference regards what the temporal parts [illegible] are. In the determinate case, and considering only valid time, the temporal part is usually a time interval, represented by a starting and an ending time (consider, e.g., TSQL2). In our approach to temporal indeterminacy, the temporal part is a pair of time intervals: an ITE consists of an interval identifying certain chronons, and an interval identifying possible ones. As a trivial consequence, the cost of the intersection operation on temporal parts (executed in main memory) is, in our approach, the double of the similar operation in the determinate case. Abstractly speaking, the same consideration could be made concerning temporal difference. However, in the case of difference, concrete implementations (including the one we have proposed in Appendix B) exploit the ordering of intervals (induced by the ordering of their endpoints). In such a context, an ITE $\langle [d_s, d_e), [i_s, i_e) \rangle$ can be regarded as an ordered triple of intervals $\langle [i_s, d_s), [d_s, d_e), [d_e, i_e) \rangle$. As a consequence, the cost of difference on temporal parts (executed in main memory) is, in our approach, the triple of the similar operation in the determinate case. (A detailed complexity analysis of our implementation of difference between set of ITEs is proposed in Appendix B). Query 1 (Temporally Indeterminate algebra) uses the same notation of Query 1 (TSQL2 algebra). However, there is a substantial difference between the two queries. TSQL2 algebraic operators σ^V, π^V and $-^V$ only operate on determinate temporal relations, providing determinate temporal relations in output. On the other hand, our operators σ^{TI}, π^{TI} and $-^{TI}$ operate on indeterminate temporal relations.

Query 1 (Temporally Indeterminate algebra).

$$\pi^{TI}{}_{Patient}(\sigma^{TI}{}_{\substack{Patient=Bill \wedge \\ Symptom=High\ Fever}}(SYMPTOMS^{TI}))$$
$$\_^{TI}$$
$$\pi^{TI}{}_{Patient}(\sigma^{TI}{}_{\substack{Patient=Bill \wedge \\ Disease=Pneumonia}}(DISEASES^{TI}))$$

The result of Query 1 (Temporally Indeterminate algebra) obtained using the approach described in this paper is shown in Table 5. The table shows that there is no day in which, certainly, high fever occurred out of pneumonia (this fact is represented by an empty determinate interval, conventionally expressed by the same start and end times). On the other hand, it is possible, from April 10^{th} (included) to April 15^{th} (excluded), that high fever occurred after the end of pneumonia (this fact, e.g., might signal the possibility of another infection, explaining the high fever).

Table 5 Result of Query 1 (Temporally Indeterminate algebra) applied to Tables 2 and 4.

Patient	$\mathbf{D_s}$	$\mathbf{D_e}$	$\mathbf{I_s}$	$\mathbf{I_e}$
Bill	Apr 10^{th}	Apr 10^{th}	Apr 10^{th}	Apr 15^{th}

Also Query 2 can be expressed through the temporally indeterminate algebra, as:

Query 2 (Temporally Indeterminate algebra).

$$\pi^{TI}_{Side\_Effect}(\sigma^{TI}{}_{\substack{Drug=B \wedge \\ Side\_Effect=SevereNausea}}(SIDE\_EFFECTS^{TI}))$$
$$\_^{TI}$$
$$\pi^{TI}_{Side\_Effect}(\sigma^{TI}{}_{\substack{Drug=A \wedge \\ Side\_Effect=SevereNausea}}(SIDE\_EFFECTS^{TI}))$$

Table 6 represents the result of Query 2 expressed with our formalism.

Table 6 Result of Query 2 (Temporally Indeterminate algebra).

Side Effect	$\mathbf{D_s}$	$\mathbf{D_e}$	$\mathbf{I_s}$	$\mathbf{I_e}$
Severe Nausea	4	5	2	7

It shows that there is certainly a day in which, in all cases, B caused severe nausea and A did not (day 4). This day represents a certain improvement of the treatment with A with respect to treatment with B. Then, there are four days in which, in some cases, B causes nausea and A did not (i.e., days 2, 3, 5, 6). These days represent a period of possible improvement (remember that our time intervals are open to the right).

Finally, we introduce temporal selection operators. Since we support indeterminate valid time, one may want to ask for tuples that *certainly* satisfy the temporal

selection predicate ϕ (i.e., such that the determinate component of the valid time satisfies the selection predicate ϕ) or tuples that *possibly* satisfy it (i.e., such that the indeterminate component of the valid time satisfies the selection predicate ϕ). We thus define two temporal selection operators, $\sigma^{TI}_{CERT_\phi}(r)$ and $\sigma^{TI}_{POSS_\phi}(r)$.

Consider, for example, the following query:

Query 3 *In the situation described by Example 2, the analyst may want to consider only the case where the patients suffered from nausea for at least two days.*

This query can be "pessimistically" expressed considering only certain results (i.e., only periods in which all the patients suffered from severe nausea for at least two days) as:

Query 3 CERT (Temporally Indeterminate algebra).

$$\sigma^{TI}_{CERT_{DURATION\geq 2}}(\sigma^{TI}_{Side\_Effect=SevereNausea}(SIDE\_EFFECTS^{TI}))$$

Or "optimistically", considering also possible results in which some patients could have suffered in a period at least two days long:

Query 3 POSS (Temporally Indeterminate algebra).

$$\sigma^{TI}_{POSS_{DURATION\geq 2}}(\sigma^{TI}_{Side\_Effect=SevereNausea}(SIDE\_EFFECTS^{TI}))$$

In the first case (Query 3 CERT), the only tuple $(B, Severe\ Nausea|1, 5, 1, 7)$ belongs to the result, because its determinate time interval $[1, 5)$ is four days long. On the other hand, the answer to the same query expressed requiring also possible results (Query 3 POSS) will contain both the tuples of Table 3.

4.1 Properties of the algebra

We have defined our algebraic operators in a principled way in order to ensure that several "core" properties hold for it. A first essential property is closure, to grant that the result of the application of our algebraic operators on the new data model can still be expressed in our data model (i.e., it is expressive enough to support our algebraic operators).

Property 2 (Closure under ITE set operators) The representation language of ITEs is closed under the operations of $\cap^{ITE}$ and $-^{ITE}$.

The property above grants that the output of intersection and difference on sets of ITEs is still a set of ITEs. This implies that the result of our temporal algebraic operators can still be expressed as a relation in our data model, as stated by the following property.

Property 3 (Closure under temporally indeterminate algebraic operators) Our data model is closed under the temporally indeterminate relational algebraic operators.

The consistent extension property grants that, if only temporally determinate data are used, our algebraic operators behave as TSQL2 ones.

Property 4 (Consistent extension (temporally indeterminate relational algebraic operators)) If only determinate ITEs of the form $\langle d, d \rangle$ are used as valid time associated with tuples, our relational operators $\cup^{TI}$, $-^{TI}$, σ_P^{TI}, π_X^{TI} and $\times^{TI}$ are equivalent to the standard TSQL2 valid-time relational operators $\cup^T$, $-^T$, σ_P^T, π_X^T and $\times^T$.

Finally, the so-called "reducibility" property is a fundamental one, granting that, if we prune our approach removing the treatment of time (i.e., if we reduce our approach to the treatment of non-temporal attributes only), the temporally indeterminate relational algebraic operators behave exactly as non-temporal relational algebraic ones. To reduce a temporally indeterminate relation to a standard non-temporal relation, we define the timeslice operator.

Definition 4 (Timeslice operator) Let r be a relation defined over the schema $(A_1, \ldots, A_n | D_s, D_e, I_s, I_e)$ and c an arbitrary time value (i.e., a chronon), the result of the timeslice operator $\rho_c^{TI}(r)$ is a standard non-temporal relation over the schema $(A_1, \ldots, A_n)$ defined as follows:

$$\rho_c^{TI}(r) = \{(v) \setminus \exists (v | \langle d, i \rangle) \in r(c \in d)\}.$$

Property 5 (Reducibility of temporally indeterminate relational algebra to non-temporal relational algebra) Temporally indeterminate algebraic operators are reducible to non-temporal algebraic operators, i.e., for each algebraic operator Op^{TI} in our model, and indicating with Op the corresponding non-temporal relational operator, for each temporally indeterminate relation r and for an arbitrary time value t the following holds (the analogous holds for binary operators):

$$\rho_c^{TI}(Op^{TI}(r)) = Op(\rho_c^{TI}(r)).$$

5 Related Work

In many applications, the exact temporal location of facts cannot be determined, so that some form of temporal indeterminacy must be managed. As a consequence, many approaches to temporal indeterminacy have been devised in different research areas. For instance, in the area of AI many different forms of temporal indeterminacy have been considered, including qualitative and quantitative constraints between events (see, e.g., the survey (Allen, 1991)). In the area of Object Oriented TDBs, a recent approach by Combi et al. (Combi et al, 1997) copes with temporal indeterminacy.

On the other hand, when moving to the area of relational databases, the number of approaches coping with temporal indeterminacy becomes more restricted. A survey of relational TDB approaches to temporal indeterminacy has recently been provided in (Dyreson, 2009). In the earliest TDB work on temporal indeterminacy, an indeterminate instant was modeled with a set of possible chronons (Snodgrass, 1982). Dutta (1989) introduced a fuzzy set approach. Gadia et al. (1992) proposed a model to support value and temporal incompleteness.

Interestingly, the relevance of temporal indeterminacy is stressed in the TSQL2 "consensus" book (Snodgrass, 1995), where Chapter 18 is dedicated to such a challenging topic. However, Chapter 18 presents only a 1NF model and an extension of SQL, while it does not provide a relational algebra.

Dyreson and Snodgrass (1998) and Dekhtyar et al. (2001) have proposed probabilistic approaches coping with different forms of temporal indeterminacy. Dyreson and Snodgrass (1998) cope with valid-time indeterminacy by associating a period of indeterminacy with a tuple. A period of indeterminacy is a period between two indeterminate instants, each one consisting of a range of granules and of a probability distribution over it. Additionally, they impose the constraint that the ranges of granules defining the starting and ending points of a period cannot overlap, so that each tuple has a "necessary" period of existence. It is worth noticing that in (Dyreson and Snodgrass, 1998) no relational algebra is proposed (and, indeed, it is easy to show that their "periods of indeterminacy" formalism is not closed under the relational operator of difference). Dekhtyar et al. introduce temporal probabilistic tuples to cope with data such as "*data tuple d is in relation r at some point of time in the interval $[t_i, t_j]$ with probability between p and p'* ". They also provide algebraic relational operators for their data model. However, they restrict their attention to facts that are instantaneous, while our approach also considers facts with duration.

On the other hand, Brusoni et al. (1999) have faced indeterminacy in the context of dealing with temporal constraints between tuples. In (Brusoni et al, 1999), bounds on differences are used in order to represent temporal constraints between tuples. The notion of conditional interval is introduced, to cope with the indeterminacy involved by temporal constraints in the relational context. Also, a relational algebra has been devised to cope with conditional intervals.

In (Anselma et al, 2013c), we already proposed a semantic model for the temporal indeterminacy in TDBs and a family of achievable representational models. However, such models are semantic-oriented, abstract and not in 1NF (thus inefficient and not suitable for a direct implementation). On the contrary, the representational model presented in this paper is concrete and in 1NF. Moving towards a 1NF relational model involves new challenging issues, which we faced in our approach. In particular, we devised a closed algebra for our 1NF model (see Section 4) and a concrete implementation of the algorithms performing the operations (in Appendix B), which represent a significant step forward in the state of the art.

The approach by Das and Musen (1994) is also relevant, and is useful to further stress the importance of devising a closed algebra. Das and Musen propose a 1NF temporal relational model in which a starting and an ending time are associated with tuples (in a state table) to model their valid times, as well as a new temporal algebra operating on them. In the final section of their paper, the model is extended in order to cope with temporal indeterminacy. In such a case, both the starting and ending times are represented as two intervals of uncertainty (IOU). Thus, a durative fact is represented as a fact having an IOU as starting time and an IOU as ending time; the interval between the upper bound of the starting time and the lower bound of the ending time represents an interval of certainty (IOC; i.e., an interval of time in which the fact necessarily holds). However, Das and Musen did not extend their temporal algebra to cope with temporal indeterminacy. They cope with temporal indeterminacy in the query by first removing temporal indeterminacy from input data (by taking either the minimum or the maximum valid-time interval for indeterminate time), and then they apply their temporal algebra for determinate time to the result. However, this a-priori removal is a severe limitation: (i) from the theoretical point of view, it means that their extended formalism coping with indeterminacy is not closed under their algebra (since their

algebraic operators cannot operate on indeterminate time and the output of their queries cannot be an indeterminate temporal relation); (ii) from the practical point of view, it reduces the expressiveness of the query language. A concrete example showing such limitations is discussed in Appendix C.

6 Conclusions

Temporal indeterminacy occurs when the **exact** temporal location of facts and properties is not known. This phenomenon is common in many real-world application domains. However, despite the popularity and the wide adoption of the relational model, only few approaches have proposed complete methodologies to cope with temporal indeterminacy in the relational context, considering both representation and query language (algebra). In this paper, we propose a comprehensive methodology overcoming the limitations of current approaches in the literature. In particular, to achieve such a goal, we widely resort to AI symbolic manipulation techniques (consider, e.g., the algorithms in Appendix B). Our goal is not just to devise an ad-hoc extension to current relational approaches, but to provide a general and theoretically grounded methodology, granting the main properties that are usually requested for (theoretically sound) relational TDB approaches. We have thus proposed a new relational data model, showing that it is a consistent extension of the "consensus" temporal model in TSQL2. In order to query temporal data, we have also proposed a temporal algebra, which is reducible to the standard (non-temporal) algebra and such that our representation formalism is closed under our algebra. We also considered computational complexity and shown that the move from determinate to indeterminate time only adds a limited overhead, while the I/O operations to retrieve/store tuples from/to secondary memory (which is by far the most important aspect to be considered) are the same in the determinate and in the indeterminate case. Despite several TDB works have already considered some issues related to valid-time temporal indeterminacy, our approach is, to the best of our knowledge, the only 1NF relational one which defines both the model and the algebra to query it, granting the above general properties.

Despite the fact that the treatment of temporal indeterminacy greatly enhances the expressiveness of TDBs, extending their practical applicability to new application domains, only limited extensions to temporal SQL languages would be required to cope with it. In particular, (i) the keyword "`INDETERMINATE`" can be added in the creation of temporally indeterminate relation; (ii) the insertion of tuples into such relation should be extended to support the definition of both the certain and possible time intervals for valid time; (iii) the query language must be extended to support the possibility of prefixing temporal selection predicates (in the "`WHERE`" part of `SELECT`) with the "`CERT`" or "`POSS`" keywords. On the other hand, the development of a calculus for our approach is a challenging goal, that we aim to address in our future work. We are also planning to extend our approach to consider the telic/atelic distinction (Terenziani and Snodgrass, 2004), and to allow for the introduction of probability distributions, in case they are known.

Acknowledgments

The authors are very much indebted to R.T. Snodgrass, for many enlightening suggestions and invaluable support he gave us in the preliminary stages of this work.

The work described in this paper was partially supported by Compagnia di San Paolo, in the Ginseng project.

References

Allen JF (1991) Time and time again: the many ways to represent time. International Journal of Intelligent Systems 6(4):341–355

Anselma L, Bottrighi A, Montani S, Terenziani P (2013a) Extending BCDM to cope with proposals and evaluations of updates. IEEE Trans Knowl Data Eng 25(3):556–570, DOI 10.1109/TKDE.2011.170, URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.170

Anselma L, Stantic B, Terenziani P, Sattar A (2013b) Querying now-relative data. J Intell Inf Syst 41(2):285–311, DOI 10.1007/s10844-013-0245-8, URL http://dx.doi.org/10.1007/s10844-013-0245-8

Anselma L, Terenziani P, Snodgrass RT (2013c) Valid-time indeterminacy in temporal relational databases: Semantics and representations. IEEE Trans Knowl Data Eng 25(12):2880–2894, DOI 10.1109/TKDE.2012.199, URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2012.199

Brusoni V, Console L, Terenziani P, Pernici B (1999) Qualitative and quantitative temporal constraints and relational databases: Theory, architecture, and applications. IEEE Trans Knowl Data Eng 11(6):948–968

Codd EF (1971) Further normalization of the data base relational model. IBM Research Report, San Jose, California RJ909

Codd EF (1972) Relational completeness of data base sublanguages. In: R Rustin (ed): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California

Combi C, Cucchi G, Pinciroli F (1997) Applying object-oriented technologies in modeling and querying temporally oriented clinical databases dealing with temporal granularity and indeterminacy. IEEE Transactions on Information Technology in Biomedicine 1(2):100–127

Das AK, Musen MA (1994) A temporal query system for protocol-directed decision support. Methods Inf Med 33(4):358–370, PMID: 7799812

Dekhtyar A, Ross RB, Subrahmanian VS (2001) Probabilistic temporal databases, i: algebra. ACM Trans Database Syst 26(1):41–95

Dunn J, Davey S, Descour A, Snodgrass RT (2002) Sequenced subset operators: Definition and implementation. In: Agrawal R, Dittrich KR (eds) ICDE, IEEE Computer Society, pp 81–92

Dutta S (1989) Generalized events in temporal databases. In: Data Engineering, 1989. Proceedings. Fifth International Conference on, pp 118–125, DOI 10.1109/ICDE.1989.47207

Dyreson CE (2009) Temporal indeterminacy. In: Liu L, Özsu MT (eds) Encyclopedia of Database Systems, Springer US, pp 2973–2976

Dyreson CE, Snodgrass RT (1998) Supporting valid-time indeterminacy. ACM Trans Database Syst 23(1):1–57

Emerson EA (1990) Temporal and modal logic. In: van Leeuwen J (ed) Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), The MIT Press, pp 995–1072

Gadia SK, Nair SS, Poon YC (1992) Incomplete information in relational temporal databases. In: Yuan LY (ed) VLDB, Morgan Kaufmann, pp 395–406

Jensen C, Snodgrass R (2008) Temporal Database Entries for the Springer Encyclopedia of Database Systems. TimeCenter Technical Report, Timecenter

Jensen CS, Snodgrass RT (1996) Semantics of time-varying information. Inf Syst 21(4):311–352

Jensen CS, Snodgrass RT (1999) Temporal data management. IEEE Trans Knowl Data Eng 11(1):36–44

McKenzie LE, Snodgrass RT (1991) Evaluation of relational algebras incorporating the time dimension in databases. ACM Comput Surv 23(4):501–543

Özsoyoglu G, Snodgrass RT (1995) Temporal and real-time databases: A survey. IEEE Trans Knowl Data Eng 7(4):513–532

Snodgrass RT (1982) Monitoring distributed systems: A relational approach. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA

Snodgrass RT (ed) (1995) The TSQL2 Temporal Query Language. Kluwer

Snodgrass RT (1999) Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann

Stantic B, Terenziani P, Governatori G, Bottrighi A, Sattar A (2012) An implicit approach to deal with periodically repeated medical data. Artificial Intelligence in Medicine 55(3):149–162

Tansel AU, Clifford J, Gadia SK, Jajodia S, Segev A, Snodgrass RT (eds) (1993) Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings

Terenziani P (2003) Symbolic user-defined periodicity in temporal relational databases. IEEE Trans Knowl Data Eng 15(2):489–509, DOI 10.1109/TKDE.2003.1185847, URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2003.1185847

Terenziani P (2012) Temporal aggregation on user-defined granularities. J Intell Inf Syst 38(3):785–813, DOI 10.1007/s10844-011-0179-y, URL http://dx.doi.org/10.1007/s10844-011-0179-y

Terenziani P (2013) Coping with events in temporal relational databases. IEEE Trans Knowl Data Eng 25(5):1181–1185, DOI 10.1109/TKDE.2011.265, URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.265

Terenziani P, Snodgrass R (2004) Reconciling point-based and interval-based semantics in temporal relational databases: a treatment of the telic/atelic distinction. Knowledge and Data Engineering, IEEE Transactions on 16(5):540–551, DOI 10.1109/TKDE.2004.1277816

Terenziani P, Snodgrass RT, Bottrighi A, Torchio M, Molino G (2007) Extending temporal databases to deal with telic/atelic medical data. Artificial Intelligence in Medicine 39(2):113–126

Vila L (1994) A survey on temporal reasoning in artificial intelligence. AI Commun 7(1):4–28

Wu Y, Jajodia S, Wang XS (1997) Temporal database bibliography update. In: Temporal Databases, Dagstuhl, pp 338–366

Appendices

A Proofs

Remember from text that
Notation. We adopt the following notation: given a tuple $t = (v_1, \ldots, v_n | d_s, d_e, i_s, i_e)$, $\langle d, i \rangle$ represents its temporal component, where d stands for the determinate time interval $[d_s, d_e)$, and i stands for the indeterminate time interval $[i_s, i_e)$.

Proof (Proof of Property 4) We consider the relational operators of Cartesian product and of difference. The proof for the other operators is easy.
Cartesian Product
In the case of determinate ITEs, the ITE intersection results in

$$\langle d, d \rangle \cap^{ITE} \langle d', d' \rangle = \langle d \cap d', d \cap d' \rangle$$

which, for the property of consistent extension on ITEs, is equivalent to the determinate temporal element

$$d \cap d'$$

Therefore, the definition of temporally indeterminate Cartesian product

$$\begin{aligned} r \times^{TI} s = \{(v_r v_s | \langle d, i \rangle) \backslash \\ \exists \langle d_r, i_r \rangle, \langle d_s, i_s \rangle ((v_r | \langle d_r, i_r \rangle) \in r \wedge (v_s | \langle d_s, i_s \rangle) \in s \wedge \\ \langle d, i \rangle = \langle d_r, i_r \rangle \cap^{ITE} \langle d_s, i_s \rangle \wedge i \neq \emptyset)\} \end{aligned}$$

is equivalent to the definition of TSQL2 Cartesian product considering valid time only:

$$\begin{aligned} r \times^{T} s = \{(v_r v_s | t) \backslash \\ \exists t_r, t_s ((v_r | t_r) \in r \wedge (v_s | t_s) \in s \wedge t = t_r \cap t_s \wedge t \neq \emptyset)\} \end{aligned}$$

Difference
Now we examine the relational difference in our approach, taking in consideration the case where only determinate time is dealt with. The definition of temporally indeterminate relational difference, in case we deal with determinate times (represented with a same determinate and indeterminate interval) can be written as:

$$\begin{aligned} r -^{TI} s = \{(v | \langle d, d \rangle) \setminus (\exists \langle d_r, d_r \rangle ((v | \langle d_r, d_r \rangle) \in r \wedge \\ \nexists \langle d_s, d_s \rangle ((v | \langle d_s, d_s \rangle) \in s \wedge \langle d, d \rangle = \langle d_r, d_r \rangle))) \vee \\ (\exists \langle d_r, d_r \rangle ((v | \langle d_r, d_r \rangle) \in r \wedge \exists ! (v | \langle d_1, d_1 \rangle), \ldots, \\ (v | \langle d_k, d_k \rangle)) ((v | \langle d_1, d1 \rangle) \in s, \ldots, (v | \langle d_k, d_k \rangle) \in s \wedge \\ \langle d, d \rangle = \langle d_r, d_r \rangle -^{ITE} \{\langle d_1, d_1 \rangle, \ldots, \langle d_k, d_k \rangle\} \wedge \\ d \neq \emptyset)))\} \end{aligned}$$

In the case of determinate ITEs, the ITE difference can be written as:

$$\begin{aligned} \langle d, d \rangle -^{ITE} \{\langle d'_1, d'_1 \rangle, \ldots, \langle d'_k, d'_k \rangle\} = \\ cover(chr(d) - (chr(d'_1) \cup \cdots \cup chr(d'_k)), \\ chr(d) - (chr(d'_1) \cup \cdots \cup chr(d'_k))) \end{aligned}$$

The cover function returns the ITEs $\langle d''_j, d''_j \rangle$ where each d''_j corresponds to the maximal convex set of chronons in $chr(d) - (chr(d'_1) \cup \cdots \cup chr(d'_k))$.

Substituting in the difference, we have:

$$\begin{aligned} r -^{TI} s = \{(v|\langle d, d\rangle) \setminus (\exists \langle d_r, d_r\rangle((v|\langle d_r, d_r\rangle) \in r \wedge \\ \nexists \langle d_s, d_s\rangle((v|\langle d_s, d_s\rangle) \in s \wedge \langle d, d\rangle = \langle d_r, d_r\rangle))) \vee \\ (\exists \langle d_r, d_r\rangle((v|\langle d_r, d_r\rangle) \in r \wedge \exists !(v|\langle d_1, d_1\rangle), \ldots, \\ (v|\langle d_k, d_k\rangle))((v|\langle d_1, d_1\rangle) \in s, \ldots, (v|\langle d_k, d_k\rangle) \in s \wedge \\ \langle d, d\rangle = cover(chr(d_r) - (chr(d_1) \cup \cdots \cup chr(d_k)), \\ chr(d_r) - (chr(d_1) \cup \cdots \cup chr(d_k))) \wedge d \neq \emptyset))). \end{aligned}$$

Now we report the definition of relational difference of TSQL2.

$$\begin{aligned} r -^{B} s = \{z \setminus \exists x \in r(z[A] = x[A] \wedge \\ \exists t \in cover^B(bi\_chr(x[TT], x[VT]) - \\ \{bi\_chr(y[TT], y[VT]) \setminus y \in s \wedge y[A] = x[A]\}) \wedge \\ z[TT_s] = min\_1(t) \wedge z[TT_e] = max\_1(t) \wedge \\ z[VT_s] = min\_2(t) \wedge z[VT_e] = max\_2(t))\} \end{aligned}$$

where A, TT, VT represent the non-temporal, transaction-time and valid-time attributes, respectively, and the subscripts s and e represent the starting and ending chronons of the interval.

Considering relations with valid time only, the definition may be simplified as:

$$\begin{aligned} r -^{V} s = \{z \setminus \exists x \in r(z[A] = x[A] \wedge \exists t \in cover^V(chr(x[VT]) - \\ \{chr(y[VT]) \setminus y \in s \wedge y[A] = x[A]\}) \wedge \\ z[VT_s] = min(t) \wedge z[VT_e] = max(t))\} \end{aligned}$$

Now we prove that $-^{TI}$ and $-^{V}$ are equivalent. In the definition of $-^{TI}$, we provide for two cases.

The first disjunct of $-^{TI}$ corresponds to the case where there is no value-equivalent tuple in s as the tuple $(v|\langle d_r, d_r\rangle)$ in r; in this case the tuple $(v|\langle d_r, d_r\rangle)$ is included in the result. Also $-^{V}$, since in this case the set $\{y[VT] \setminus y \in s \wedge y[A] = x[A]\}$ is empty, includes in the result the tuples in r with no value-equivalent tuples in s.

The second disjunct of $-^{TI}$ corresponds to the case where a tuple $(v|\langle d_r, d_r\rangle)$ in r has the value-equivalent tuples $(v|\langle d_1, d_1\rangle), \ldots, (v|\langle d_k, d_k\rangle)$ in s. In the definition of $-^{V}$, the set $\{y[VT] \setminus y \in s \wedge y[A] = x[A]\}$ corresponds to the same value-equivalent tuples $(v|\langle d_1, d_1\rangle), \ldots, (v|\langle d_k, d_k\rangle)$ in s. Thus, both $-^{TI}$ and $-^{V}$ perform set difference between the same sets of chronons. In $-^{TI}$ the *cover* function returns the minimum and maximum chronons in the convex sets of the set difference, whereas in $-^{V}$ the $cover^V$ function returns only the convex sets and the minimum and maximum chronons are determined in the definition of $-^{V}$. For the consistent extension property on ITEs, an ITE $\langle d, d\rangle$ is equivalent to a determinate temporal element d. It is worth noticing that neither $-^{TI}$ nor $-^{V}$ return tuples with empty temporal elements because of the clause $d \neq \emptyset$ for $-^{TI}$ and because of the existential quantification $\exists t$ for $-^{V}$.

Proof (Proof of Property 5) For the sake of brevity, we prove the property considering the Cartesian product operator. The proofs for the other operators are similar. Let r and s be ITE relations with schemas $(A|T)$ and $(B|T)$ respectively, where A, B and T stand for the attributes $\{A_1, \ldots, A_l\}$, $\{B_1, \ldots, B_m\}$ and $\{D_s, D_e, I_s, I_e\}$ respectively, then

$$\rho_t^{TI}(r \times^{TI} s) = \rho_t^{TI}(r) \times \rho_t^{TI}(s)$$

where $\times^{TI}$ is the ITE Cartesian product, $\times$ is the standard non-temporal Cartesian product and ρ_t^{TI} is the timeslice operator. We show the equivalence by proving the two inclusions separately, i.e., we prove that the left-hand side of the formula (henceforth lhs) implies the right-hand side (henceforth rhs) and that the rhs implies the lhs.

($\mathbf{x''} \in \mathbf{lhs} \Rightarrow \mathbf{x''} \in \mathbf{rhs}$)
Let $x'' \in lhs$. Then, by the definition of ρ_t^{TI}, there exists a tuple $x' \in (r \times^{TI} s)$ such that $x'[A,B] = x''[A,B]$
and $t \in x'[D]$.

By the definition of $\times^{TI}$, there exist tuples
$x_1 \in r$ and $x_2 \in s$ such that $x_1[A] = x'[A], x_2[B] = x'[B]$ and $x_1[T] \cap x_2[T] = x'[T]$.

Then, by the definition of ρ_t^{TI}, there exists a tuple
$x'_1 \in \rho_t^{TI}(r)$ such that $x'_1[A] = x_1[A] = x'[A]$, and there exists a tuple $x'_2 \in \rho_t^{TI}(s)$ such that $x'_2[B] = x_2[B] = x'[B]$.

Therefore, by the definition of $\times$, there exists $x''_{12} \in rhs$ such that $x''_{12}[A] = x'_1[A]$ and $x''_{12}[B] = x'_2[B]$.

By construction, $x''_{12} = x''$.

($\mathbf{x''} \in \mathbf{rhs} \Rightarrow \mathbf{x''} \in \mathbf{lhs}$)
Now assume $x'' \in rhs$. Then, by definition of $\times$, there exist tuples $x'_1 \in \rho_t^{TI}(r)$ and $x'_2 \in \rho_t^{TI}(s)$ such that
$x'_1[A] = x''[A]$ and $x'_2[B] = x''[B]$.

By the definition of ρ_t^{TI}, there exists a tuple $x_1 \in r$ such that $x_1[A] = x'_1$ and $t \in x'_1[D]$ and there exists a tuple $x_2 \in r$ such that $x_2[B] = x'_2$ and $t \in x'_2[D]$.

Then by definition of $\times^{TI}$ there must exist a tuple
$x' \in (r \times^{TI} s)$ such that $x'[A] = x_1[A]$, $x'[B] = x_2[B]$, $x'[T] = x_1[T] \cap x_2[T]$ and $t \in x'[D]$.

Then, by definition of ρ_t^{TI}, there exists a tuple
$x''_{12} \in lhs$ such that $x''_{12}[A,B] = x'[A,B]$.

By construction, $x''_{12} = x''$.

B Algorithms

In Section 4 we have proposed a definition of the relational difference between two temporally indeterminate relations based on an abstract definition of the difference between an ITE (henceforth minuend) and a set of ITEs (henceforth subtrahends). For the sake of clarity, the ITE difference in Section 4 was based on a conversion from ITEs to sets of chronons and back. The definition is very general, covering all the possible alternative solutions (since there are, in general, multiple equivalent ways of converting the chronons in the result into a set of ITEs).

On the other hand, in this Appendix we propose an actual algorithm to perform ITE difference. The algorithm is based on the abstract definition of Section 4, but it is more efficient, since it directly operates on time intervals instead of sets of chronons. Also, it is based on a specific partitioning policy. Before detailing the algorithm, we need to introduce some useful concepts.

The basic problem with temporal relational difference (independently of whether determinate or indeterminate time intervals are adopted) is that interval difference must, in general, be performed between sets of intervals. Even in the determinate case, the difference between an interval $[s_1, e_1)$ and an interval $[s_2, e_2)$ contained into it (i.e., such that $s_1 < s_2 < e_2 < e_1$) results in two intervals $[s_1, s_2)$ and $[e_1, e_2)$. Thus, even if the operation starts with the difference between one interval and a set of intervals (one for each one of the value-equivalent tuples), intermediate computational steps must consider difference between two sets of intervals. In general, such an operation would require quadratic time. However, such a complexity can be reduced by exploiting ordering (the ordering between intervals can be trivially defined on the basis of the temporal ordering of their endpoints). We exploit such an idea also in our case, in which ITEs are considered (instead of determinate intervals). To do so, we introduce the notion of (ordered) list of "Typed Intervals".

A ***Typed (Temporal) Interval*** (henceforth TY) represents a convex set of chronons, which are all "labeled" either as determinate (DET) or as indeterminate ($INDET$). A TY is completely described by the triple $\langle start, end, type \rangle$, where $start, end \in T^C$ are the starting and ending points of the TY (as in the ITE representation, a TY interval $[start, end)$ includes the starting chronon and excludes the ending one) and $type \in \{DET, INDET\}$. Hereinafter, for the sake of brevity, we will use the dot notation for TY (e.g., if ty is a TY, $ty.start$ is the starting point of ty).

We use three relations between TYs. In particular, given two TYs ty_1 and ty_2,

- before(ty_1, ty_2) stands for $ty_1.end \leq ty_2.start$
- meets(ty_1, ty_2) stands for $ty_1.end = ty_2.start$
- overlaps(ty_1, ty_2) stands for $ty_1 \cap ty_2 \neq \emptyset$

In the algorithms below, we will also use the notion of *List of TYs*. A List of TYs is a collection of TYs such that it is:

- *maximal* in the sense that $ty_1, ty_2 \in l \wedge meets(ty_1, ty_2)$ $\Rightarrow ty_1.type \neq ty_2.type$;
- *without intersections*, i.e., $ty_1, ty_2 \in l$ $\Rightarrow \neg overlaps(ty_1\, ty_2)$;
- *ordered*, i.e., in a List of TYs $(ty_1, \ldots, ty_i, \ldots, ty_j, \ldots, ty_n)$ $i < j$ $\Rightarrow before(ty_i, ty_j)$.

Given $l : list\_of\_ty$, we denote with $l.size$ the number of elements contained in l. $l[i]$ is the i-th element of the list l (with $1 \leq i \leq l.size$). We also use the notation "insert el into l in position i", "append el to l" and "remove from l element in position i" to denote, respectively, insertion, insertion in the last position and the classical deletion of an element from the list l. In order to grant maximality, if an element is added to a List of TYs and it meets the subsequent element or the previous one meets it and their types are equal, they are automatically merged.

In addition, two transform operations are defined: the $toTY$ operation transforms a set of ITEs into a List of TYs, and the $toITE$ operation transforms a List of TYs into a set of ITEs. Given the previous notions, we now describe the algorithm for difference between ITEs (see Algorithm 6). It is basically divided into three phases:

1. **Both the minuend and the subtrahends are transformed into Lists of TYs.** This operation is performed by using the $toTY$ function (see Algorithm 1) that, given as input a set of ITEs *set*, returns a List of TYs. In the basic case, in which *set* contains only an element, called *ite*, the number of returned elements depends on the structure of *ite*: in case its determinate interval $[d_s, d_e)$ is empty, a single INDET TY is returned, otherwise a list containing a INDET TY representing $[i_s, d_s)$ (if not empty), a DET one representing $[d_s, d_e)$ and a INDET one representing $[d_e, i_e)$ (if not empty) is returned.

Algorithm 1: toTY

```
Input  : set : set_of_ITE
Output: list_of_TY
result : list_of_TY;
if set contains a single element "ite" then
    result ← empty;
    if ite.ds ≥ 0 and ite.ds < ite.de then
        if ite.is < ite.ds then
            append ⟨is, ds, INDET⟩ to result;
        end
        append ⟨ds, de, DET⟩ to result;
        if ite.ie > ite.de then
            append ⟨de, ie, INDET⟩ to result;
        end
    else
        append ⟨is, ie, INDET⟩ to result
    end
else
    set1, set2 : set_of_ITE;
    set1, set2 ← empty;
    add half of the elements in set to set1;
    add the other half of set to set2;
    list1, list2 : list_of_TY;
    list1 ← toTY(set1);
    list2 ← toTY(set2);
    result ← merge(list1, list2);
end
return result;
```

Algorithm 2: merge

```
Input  : list1, list2 : list_of_TY
Output: list_of_TY
i, j ← 1;                        /* i is the index for list1, j for list2 */
toinsert : TY;
toinsert ← empty;
while i ≤ list1.size or toinsert ≠ empty do
    if toinsert = empty then
        toinsert = list1[i];
        i ← i+1;
    end
    while j ≤ list2.size and before(list2[j], toinsert) do
        j ← j+1;
    end
    if j > list2.size then
        append toinsert to list2;
        toinsert ← empty;
    else
        if ¬overlaps(toinsert, list2[j]) then
            add toinsert to list2 in position j;
            toinsert ← empty;
        else
            union : list_of_TY;
            union ← unify(list2[j], toinsert);
            remove from list2 element in position j;
            for k = 1, . . . , union.size-1 do
                add union[k] to list2 in position j;
                j ← j+1;
            end
            toinsert ← union[union.size];
        end
    end
end
return list2;
```

Algorithm 3: unify

```
Input  : ty1, ty2 : TY
Output: list_of_TY
result : list_of_TY;
result ← empty;
if ty1.type = ty2.type and (meets(ty1, ty2) or meets(ty2, ty1) or overlaps(ty1, ty2))
then
    append ⟨ min(ty1.start, ty2.start), max(ty1.end, ty2.end), ty1.type)⟩ to result ;
    return result;
end
if ¬overlaps(ty1, ty2) then
    if before(ty1, ty2) then
        add ty1 to result in position 1;
        append ty2 to result;
    else
        add ty2 to result in position 1;
        append ty1 to result;
        ;
    end
else
    n : TY;
    p : TY;
    if ty1.type = DET then
        n ← ty1;
        p ← ty2;
    else
        n ← ty2;
        p ← ty1;
    end
    if p.start < n.start then
        append ⟨p.start, n.start, INDET⟩ to result;
    end
    append n to result;
    if p.end > n.end then
        append ⟨n.end, p.end, INDET⟩ to result;
    end
end
return result;
```

In the case in which *set* contains more than one element (e. g., for the subtrahends), *set* is partitioned into two subsets, then, for both of them, a List of TYs is obtained separately. Finally, the two lists are combined by the merge algorithm (see Algorithm 2) that grants that the list in the result respects the properties previously mentioned (i.e., it is *maximal*, *without intersections* and *ordered*).

2. **Each TY obtained from the minuend (*minuend_el*) is compared with the elements in the subtrahend_list in order to remove overlaps.** A many-to-many difference between the List of TYs deriving from minuends and those obtained from subtrahends needs to be performed. However, exploiting the ordering of both the lists, they are visited only once. In particular, if an element of the subtrahend_list is in relation of *before* with the one actually considered for the minuend, there is no need to compare it with the following elements of the minuend. On the other hand, if the actual minuend_el is before the i-th element of subtrahend_ list, there is no need to compare minuend_el with the following elements of subtrahend_ list. Thus, an iterator is initialized to the first element of the subtrahend_list (in the algorithm it is represented by the variable j) and for each minuend_el, j is increased until a subtrahend_list[j] is found, such that subtrahend_list[i] is not *before* minuend_el. There are three cases:

   - **The end of subtrahend_list is reached.** If $j > subtrahend\_list.size$, minuend_el and next elements of minuend_list intersect with no element in the subtrahend. Thus, they are added to the result.
   - **An element subtrahend_list[j] is found, such that minuend_el is *before* subtrahend_list[j]** In this case, the current minuend_el can be added to the result as it is. The algorithm restarts from the next TY of minuend_list, keeping the value reached by the iterator j.
   - **The last case is that in which subtrahend_list[j] *overlaps* the designed element minuend_el.** In such a case, a difference operation is performed between minuend_el and subtrahend_list[i]. The result of difference between TYs is a List of TYs composed by up to three TYs.

Algorithm 4: minus

```
Input  : minuend, subtrahend : TY
Output: list_of_TY
result : list_of_TY;
result ← empty;
if subtrahend.type = DET then
    if minuend.start < subtrahend.start then
        append ⟨minuend.start, subtrahend.start, minuend.type⟩ to result;
    if minuend.end > subtrahend.end then
        append ⟨subtrahend.end, minuend.end, minuend.type⟩ to result;
else
    if minuend.type = INDET then
        append minuend to result;
    else
        if minuend.start < subtrahend.start then
            append ⟨minuend.start, subtrahend.start, DET⟩ to result;
        if overlaps(minuend, subtrahend) then
            append ⟨max(minuend.start, subtrahend.start), min(minuend.end,
            subtrahend.end), INDET⟩ to result;
        if minuend.end > subtrahend.end then
            append ⟨subtrahend.end, minuend.end, DET⟩ to result;
    end
end
return result;
```

All the elements of this list, except the last, can be inserted in the result, while the last one (if the list size is at least one) takes the place of minuend_el in the algorithm and it is compared with next elements of the subtrahend_list.

3. **A set of ITEs representing the result of the difference between TYs is obtained.** This operation is performed by the function *toITE* (see Algorithm 5), which accomplishes, for the domain of TYs, the same task of cover function of Figure 2. Given the particular structures used in this implementation, the partition policy of our algorithm tends to create ITEs similar to the ones shown in the upper part of Figure 3.

Algorithm 5: toITE

```
Input  : intervals : list_of_ty
Output: set_of_ITE
result : set_of_ITE;
result ← empty list;
ds, de, is, ie : Chronons;
i ← 1;
detins : Boolean;
el : TY;
detins ← false;
while i ≤ intervals.size do
    el ← intervals[i];
    ds, is ← el.start;
    ie ← el.end;
    if el.type = DET then
        de ← el.end;
        detins ← true;
    else
        de ← ds;
    end
    i ← i+1;
    while i ≤ intervals.size and meets(el, intervals[i]) and (intervals[i].type =
    INDET or ¬detins) do
        el ← intervals[i];
        ie ← el.end;
        if el.type = DET then
            ds ← el.start;
            de ← el.end;
            detins ← true;
        end
        i ← i+1;
    end
    add ⟨ds, de, is, ie⟩ to result;
end
return result
```

Algorithm 6: Difference

```
Input  : minuend : ITE and
         subtrahends : set_of_ITE
Output: set_of_ITE

minuend_list : list_of_TY;
minuend_list ← toTY(minuend);
subtrahend_list : list_of_TY;
subtrahend_list ← toTY(subtrahends);
result : list_of_TY;
result ← empty list;
i, j ← 1;
minuend_el : TY;
minuend_el ← empty;
while i ≤ minuend_list.size or minuend_el ≠ empty do
    if minuend_el = empty then
        minuend_el ← minuend_list[i];
        i ← i+1;
    end
    while j ≤ subtrahend_list.size and before(subtrahend_list[j], minuend_el) do
        j ← j+1;
    end
    if j > subtrahend_list.size or ¬overlaps(minuend_el, subtrahend_list[j]) then
        append minuend_el to result;
        minuend_el ← empty;
    else
        difference : list_of_TY;
        difference ← minus(minuend_el, subtrahend_list[j]);
        for k = 1, . . . , difference.size-1 do
            append difference[k] to result;
        end
        if difference = empty then
            minuend_el ← empty;
        else
            if difference[difference.size].end > subtrahend_list[j].end then
                minuend_el ← difference[difference.size];
                j ← j+1;
            else
                append difference[difference.size] to result;
                minuend_el ← empty;
            end
        end
    end
end
return toITE(result);
```

B.1 Discussion on Complexity

Complexity (Difference between two sets of ITEs).
Suppose that n is the number of ITEs that have to be subtracted from one ITE. Difference operates in three main steps: (1) pre-processing (toTY), (2) difference evaluation, and (3) post-processing (toITE). The toTY transformation takes in input a set of ITE, and converts it into an ordered list of TYs henceforth. In general, each ITE may correspond to three TYs. toTY basically operates like the classical mergesort algorithm, with a complexity which is $O(m \log_2 m)$, where m is the number of TYs (i.e., m is at most $3 * (n + 1)$). By exploiting the ordering of the list of TYs, in step 2 difference can be executed by visiting each TY at most once, i.e., in a time that is $O(m)$. Finally, toITE reconvert TYs into ITEs, also "coalescing"

indeterminate TY that meet each other. By exploiting the ordering of TYs in the list, also such operation is performed by visiting each TY once, i.e., in linear time $O(m)$. Overall, the complexity is thus dominated by the initial ordering step (step 1), and is $O(m \log_2 m)$.

As discussed in Section 4, the complexity of our relational operator of difference is the same as the one of many TDB approaches (in particular, the same I/O operations are performed in both cases), including TSQL2, except the operation of difference between time intervals. We now compare such complexity, considering the specific implementation described above.

Complexity (Difference between two sets of ITEs vs. difference between sets of determinate time intervals).

Let us now consider the complexity in the determinate case, supposing that n is the cardinality of the set of (determinate) time intervals to be subtracted from a given one. For the sake of efficiency, also Difference between sets of determinate time intervals can exploit a pre-processing step to order them. A slight variation of mergesort can be used, so that the complexity is $O(n \log_2 n)$. After that, difference can be obtained subtracting one interval at a time, in time $O(n)$. No post-processing step is needed in the determinate case (since no conversion is required, and the output of determinate difference is already coalesced). As in the case of indeterminate time, the complexity of difference is thus dominated by the ordering step, which, applied to sets of x elements, requires $O(x \log_2 x)$ time. The main difference is thus a multiplicative constant, due to the fact that an ITE indeed correspond to an ordered list of (at most) three intervals.

C Comparison with a non-closed approach

In this appendix, we present a comparison between our approach with the one proposed by Das and Musen (1994) in the medical field, to exemplified the importance of devising a data model and algebra with the closure property. As discussed in Section 5, Das and Musen (1994) proposed a 1NF temporal relational model coping with temporal indeterminacy through the introduction of two intervals of uncertainty (IOUs), one for the starting time and one for the ending time. The interval between the upper bound of the starting time and the lower bound of the ending time represents an interval of certainty (IOC; i.e., an interval of time in which the fact necessarily holds). Notice that Das and Musen propose an implementation in which (at most) three tuples are used to model a temporally indeterminate fact. A tuple with Type equal to "body" represents the time when the fact certainly holds (IOC), while the other two tuples with Type equal to "start" and "end" represent the time when the fact possibly holds (the IOU of the starting time and the IOU of the ending time, respectively). Considering Example 2, Das and Musen represent data as shown in Table 7.

Table 7 Relation SIDE_EFFECTS_DM (Das and Musen representation of Example 2).

Start_Time	Stop_Time	Type	Drug	Side_Effect
1	1	body	A	Severe Nausea
2	3	end	A	Severe Nausea
1	4	body	B	Severe Nausea
5	6	end	B	Severe Nausea

However, Das and Musen did not extend their temporal algebra to cope with temporal indeterminacy.

> "*To manipulate states, on the other hand, we must choose between the minimum or maximum span representation of the state. . . . Either of these approaches then results in a single pair of endpoints for the state-based data. . .*" (Das and Musen, 1994).

In other words, they cope with temporal indeterminacy in the query by first removing temporal indeterminacy from input data (by taking either the minimum or the maximum valid-time interval for indeterminate time), and then they apply their temporal algebra for determinate time to the result. However, this is restrictive: indeed, their extended formalism coping with indeterminacy is not closed under their algebra, since their algebraic operators cannot operate

on indeterminate time and the output of their queries cannot be an indeterminate temporal relation (in fact, indeterminacy is removed in the first, necessary, step of their queries). This is a major limitation. Indeed, Das and Musen themselves noticed that their algebraic operators, when operating on temporally indeterminate facts, " *. . . may produce anomalous results. . .* " (Das and Musen, 1994). Indeed, in the following, we show a simple example demonstrating that Das and Musen's approach is limited, in that certain queries cannot be properly managed.

Let us consider the situation described by Example 2, and suppose that the user wants to know when drug B and not drug A caused nausea (i.e., Query 2). Considering the information in Example 2, the output should be that B and not A caused nausea certainly on day 4, and possibly on days 2, 3, 5 and 6. Notice, however, that such a result cannot be obtained operating as proposed by Das and Musen. If the minimum valid time (i.e., the "certain" time IOC) is first selected, then the difference between the time intervals $[1, 4]$ and $[1, 1]$ should be performed, obtaining $[2, 4]$ as a result (see Table 8). On the other hand, if the maximum valid time (i.e., the "possible" time, IOU) is first selected, then the difference between the time intervals $[1, 6]$ and $[1, 3]$ should be performed, obtaining $[4, 6]$ as a result (see Table 9). Obviously, none of them is the desired (correct) result to Query 2.

Table 8 Das and Musen's answer to Query 2, considering the *minimum* valid time.

Start_Time	**Stop_Time**	**Side_Effect**
2	4	Severe Nausea

Table 9 Das and Musen's answer to Query 2, considering the *maximum* valid time.

Start_Time	**Stop_Time**	**Side_Effect**
4	6	Severe Nausea

Indeed, although quite simple, the above example demonstrates the necessity of developing a closed temporal algebra for indeterminate time, i.e., an algebra in which temporal indeterminate data (relations) are directly managed (with no need of removing indeterminacy) as first-class entities, which may be input and output of the queries. Despite the diffusion of the relational model, and the relevance of temporally indeterminate data in many real-world contexts, so far there is no temporal relational approach providing both a 1NF data representation formalism and a closed relational algebra operating on it to cope with temporally indeterminate data (see also the discussion in Section 5). Providing such an approach and proving its reducibility to the standard non-temporal algebra are the results we achieved in the work we describe on in this paper.