Point vs. Interval-based Query Languages for Temporal Databases

(Extended Abstract)

David Toman

Department of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 1A4
david@cs.toronto.edu

Abstract

In this paper we establish a correspondence between two major views of temporal databases and the corresponding first-order temporal query languages: the point-based view of temporal databases vs. the interval-based view of temporal databases. We show that all first-order queries can be conveniently asked using a point-based first-order query languages in a much more declarative and natural way and then mechanically translated into an interval-based query language, e.g., TSQL2. Such an approach combines the ease of formulating queries in first-order logic (temporal relational calculus) with the efficient query evaluation algorithms developed for the interval-based temporal databases.

1 Introduction

In this paper we try to fill the gap between two main directions of research in the area of temporal databases and temporal query languages:

The first direction studies the temporal databases as two-sorted first-order structures and temporal query languages as suitable languages of logic over such structures [5]. This approach provides a clean and declarative semantics of queries. The associated query languages are usually based on first-order logic (relational calculus [14]) or some of its sublogics (e.g., temporal logic [3]). A natural assumption in these languages is that the temporal variables in the queries refer to *individual time instants*. However, the relational structures that capture the temporal information are usually infinite (as the domain of time instants is infinite in this model) and thus not immediately suitable as practical temporal databases. *Finite encodings* of such structures are needed, e.g., [7], to make this approach practically usable.

The second approach is connected with the more practical side of research in the area of temporal databases. Here special care is given to the development of space-efficient storage techniques of large and often infinite sets of time instants [12]. A classical solution of this problem uses encoding of such sets of time instants using intervals [9, 11]. However, a side-effect of such a solution is that the query languages over the interval-based temporal databases are based on the syntax of the *actual encoding* of sets of time instants, rather than on the underlying meaning of the encoding. This is especially true for many of the *interval-based* temporal query languages, where the variables in the queries range over *intervals*. This fact leads to several problems:

- The query languages are complicated as they explicitly refer to the particular encoding of sets of time instants, rather than to the individual time instants themselves (cf. Appendix A). Giving a clear semantics to such temporal query languages is quite difficult [12]: the intervals serve merely as a tool that allows finite encoding of infinite point-based temporal databases, while the actual semantics of queries is defined with respect to the point-based temporal databases themselves.
- The formulation of queries in the interval-based query language is not natural: the declarative nature of asking queries in a logic-based query language is lost by the need for explicit handling of the peculiarities of the chosen encoding of sets of time instants (in our case: intervals).

The advantage of utilizing the second solution is the ease of implementation—we can reuse virtually all of the standard relational technology, which leads to very efficient query evaluation.

To bridge this gap, we propose an approach that allows us to ask declarative point-based first-order queries over the space-efficient interval-based encodings of temporal databases. The approach is based on a translation of the point-based queries to interval-based queries. Such a translation allows us to use the existing interval-based temporal database systems (based on TQUEL [9] or TSQL [11]) as a back-end over which the declarative point-based queries can be asked. The translation has to satisfy the following requirements:

Preservation of answers: a tuple of values is in the answer to a given point-based temporal query evaluated over a given temporal database at time t if and only if the same tuple is related to an interval I in the answer to the translated query over the encoding of the given temporal database and I contains t.

Closure: the set of time instants associated with every tuple in the answer to the transformed query is finitely representable using the same encoding that is used to encode the (relations of the) temporal database itself.

Such a translation can be immediately used as an algorithm that allows high-level point-based queries to be compiled into an interval-based query language, e.g., TQUEL or TSQL2[1]. Note that this translation addresses completely different issues than the first-order logic to SQL translations[2] (e.g., [1, 15]). Note also that our goal is different than the goal pursued in the area of *constraint databases* [7]. Our transformation allows us to translate the point-based queries to the *existing* interval-based query languages. This approach can serve as a convenient front-end to the existing temporal database management systems [5, 10].

The contributions of this paper can be summarized as follows: (1) we show that over a single-dimensional linearly ordered model of time the point-based formalism is *equally* powerful as the interval-based one while the point-based queries are much easier to formulate, (2) we provide a translation procedure that automatically maps the point-based queries to the interval-based queries, and (3) we provide a syntactic criterion that guarantees safety of the translated queries with respect to the underlying interval-based encoding of temporal databases.

The rest of the paper is organized as follows: Section 2 gives precise definitions of point-based and interval-based temporal databases and the corresponding first-order query languages. Section 3 introduces the translation of point-based queries to interval-based ones and proves its correctness. Section 4 discusses safety of temporal queries. Section 5 shows that the class of point-based queries allows to express all generic interval-based queries. Section 6 concludes the paper with directions of further research, especially the possibilities of adopting the method from Section 3 to different encodings of temporal databases. We also show examples of a few point-based temporal queries and their translations in Appendix A.

[1] In the scope of this paper we abstract from any particular query language and use the first-order logic as the basis for both the point-based and the interval-based queries.

[2] We study the differences between the point- and interval-based query languages, rather than the evaluability and safety issues.

2 Temporal Databases and Query Languages

This section gives precise definitions of temporal databases and the corresponding first-order query languages.

2.1 Temporal Databases

We define two classes of two-sorted first-order structures extended with *user-defined* relations (following the terminology introduced in [5]):

1. the class of *Abstract temporal databases* (point-based), elements of which serve as the actual representation-independent meanings of the temporal databases, and
2. the class of *Concrete temporal databases* (interval-based) that contains the space-efficient encodings of the abstract temporal databases.

First, we define the *data objects* that are stored in the database. For this purpose we use a domain (sort) that models the usual uninterpreted constants:

Definition 2.1 (Data Domain) *Let D be a first order structure $(D; =, c_1, c_2, \ldots)$ with a countable carrier, $|D|$ equipped with equality relation and constant symbols c_i for every element of the carrier. We call D a* Data Domain (or a domain of uninterpreted constants).

This sort is used as the representation of the *data* values in both the point- and the interval-based approaches. To define the domain that models *time* in the database consider the following definitions[3]:

Definition 2.2 (Point-based Domain $\mathbf{T_P}$) *Let T be a set and $<$ a discrete linear order without endpoints on T. Then the structure $T_P = (T, <)$ is the* Point-based Temporal Domain.

The elements of the carrier of T model the individual time instants and the linear order $<$ models the succession of time. To define the interval-based temporal domain corresponding to the point-based temporal domain T_P, consider the following definition:

Definition 2.3 (Interval-based Domain $\mathbf{T_I}$) *Let $T_P = (T, <)$ be a point-based temporal domain. We define the set*

$$I(T) = \{(a, b) : a \leq b \wedge a, b \in T \cup \{-\infty, \infty\}\}$$

[3] In this paper we consider only discrete linearly ordered models of time. This is a quite a common assumption, made in virtually all practical implementations. Discrete model of time also allows us to work only with closed intervals. However, a dense model can be adopted by minor modifications that distinguish between open and closed intervals. We briefly discuss this and other scenarios in Section 6.

where $\leq$ is the ordering of the structure $(T, <)$ extended with $\{(a,a), (-\infty, a), (a, \infty), (-\infty, \infty) : a \in T\}$. We denote elements of $I(T)$ by $[a,b]$ (which is the usual notation for intervals). We also define four relations on the elements of $I(T)$. Let $[a,b], [a',b'] \in I(T)$ and

$$\begin{array}{l}([a,b] <_{--} [a',b']) \Longleftrightarrow a < a' \\ ([a,b] <_{+-} [a',b']) \Longleftrightarrow b < a' \\ ([a,b] <_{-+} [a',b']) \Longleftrightarrow a < b' \\ ([a,b] <_{++} [a',b']) \Longleftrightarrow b < b'\end{array}$$

where $<$ is the extended order of $(T, <)$. Then the structure $T_I = (I(T), <_{--}, <_{+-}, <_{-+}, <_{++})$ is the Interval-based Temporal Domain *(corresponding to T_P).*

Note that both T_P and T_I model *the same* structure of time instants—a single-dimensional linearly ordered time line. This observation is crucial as it allows us to define the semantics of the concrete temporal databases by mapping them into the class of abstract temporal databases (which *represent the actual meanings*). Note that we could have chosen a two-dimensional model of time: in that case the intervals (or pairs in general) would be the elements of T_P and we would have to look for encodings of potentially infinite *sets of intervals*, similarly to Definition 2.3.

We defined the domains for both the data elements and the time instants. A temporal database captures the relationships between these individuals in a set of *user-defined* relations:

Definition 2.4 (Database Schema) *Let ρ be a relational database schema $(r_1, \ldots, r_k)$ (a set of single sorted predicate symbols). We define a* Temporal Database Schema *as a set of two-sorted predicate symbols $(R_1, \ldots, R_k)$ where the sort of R_i is $T \times D^{\mathrm{arity}(r_i)}$.*

Let T be a temporal domain and D a data domain. Then an instance of the database schema ρ with respect to T and D *is a set of relations $\mathbf{R}_i \subseteq |T| \times |D|^{\mathrm{arity}(r_i)}$.*

An instance of the database schema is finite *if all the relations $\mathbf{R}_i$ are finite.*

The previous definitions provide all the components, needed to define a Temporal Database:

Definition 2.5 (Temporal Database) *Let T be a temporal domain, D a data domain, and $\mathbf{R}_1 \ldots, \mathbf{R}_k$ an instance of the database schema ρ. Then the two-sorted first-order structure $\mathrm{D} = T \cup D \cup \{\mathbf{R}_1 \ldots, \mathbf{R}_k\}$ is a* Temporal Database over T, D, and ρ.

We define $\mathrm{card}(\mathrm{D})$ *to be the sum of the cardinalities of the relations $\mathbf{R}_1 \ldots, \mathbf{R}_k$.*

It is easy to see that Definition 2.5 can be used to define both the class of abstract temporal databases $\mathcal{D}_P$ as well as the class of the concrete temporal databases $\mathcal{D}_I$. The only difference is in the choice of the temporal domains: in the first case we use T_P and in the second case we use T_I (which in turn is built out of T_P). However, both $\mathcal{D}_P$ and $\mathcal{D}_I$ represent *the same class of single-dimensional temporal databases*.

Of course, in a real database system only *finite* temporal databases can be stored in computer storage. This forces us to restrict our attention to finite instances of the appropriate database schema. Note that the relations $=$ or $<$ have a fixed *built-in* interpretation and thus do not need to be stored in the computer storage. On the other hand, the interpretation of the *user-defined relations* is determined from the stored image of the schema instance. This is the main motivation behind using the finite encodings of potentially infinite sets of time instants—finite encodings allow us to store and query larger class of temporal databases.

The actual meaning of both the point-based and the interval-based temporal databases defines relationships between elements of the data domain that hold at individual time instants. More precisely if $\mathrm{D}_1 \in \mathcal{D}_P$ is a point based temporal database over a database schema containing the relational symbol r_i, t a time instant, and $(a_1, \ldots, a_{\mathrm{arity}(r_i)}) \in |D|^{\mathrm{arity}(r_i)}$ then

$$(t, a_1, \ldots, a_{\mathrm{arity}(r_i)}) \in \mathbf{R}_i$$

means that in the temporal database D_1 the elements $a_1, \ldots, a_{\mathrm{arity}(r_i)}$ are in relation r_i at time t. Similarly, in the case of an interval-based temporal database $\mathrm{D}_2 \in \mathcal{D}_I$ over the same database schema

$$(I, a_1, \ldots, a_{\mathrm{arity}(r_i)}) \in \mathbf{R}_i$$

means that in D_2 the elements $a_1, \ldots, a_{\mathrm{arity}(r_i)}$ are in relation r_i at *all* time instants $t \in I$. This observation allows us to compare the meanings of the point- and interval-based temporal databases:

Definition 2.6 *Let $\mathrm{D}_1 \in \mathcal{D}_P$ and $\mathrm{D}_2 \in \mathcal{D}_I$ be two temporal databases over the same schema ρ. We say that* D_2 encodes D_1 *if*

$$\mathbf{R}_i^{\mathrm{D}_1}(t, \mathbf{x}) \iff \exists I \in T_I . \mathbf{R}_i^{\mathrm{D}_2}(I, \mathbf{x}) \wedge t \in I$$

for all r_i in the database schema ρ, $t \in |T|$, and $\mathbf{x} \in |D|^{\mathrm{arity}(r_i)}$, where $\mathbf{R}_i^{\mathrm{D}}$ is the interpretation of the relation symbol R_i in the database D.

This correspondence defines a map $\|.\| : \mathcal{D}_I \to \mathcal{D}_P$ as an extension of the mapping of relations in $\mathrm{D} \in \mathcal{D}_I$ to relations in $\|\mathrm{D}\| \in \mathcal{D}_P$:

$$\mathbf{R}_i^{\|\mathrm{D}\|} = \|\mathbf{R}_i^{\mathrm{D}}\| = \{(t, \mathbf{a}) : \exists I \in T_I . (I, \mathbf{a}) \in \mathbf{R}_i^{\mathrm{D}} \wedge t \in I\}$$

for all $r_i \in \rho$ and $\mathbf{a} = (a_1, \ldots, a_{\mathrm{arity}(r_i)}) \in |D|^{\mathrm{arity}(r_i)}$. $\|.\|$ preserves the intuitive meaning of the databases for all $\mathrm{D} \in \mathcal{D}_I$. Note that this mapping is not injective.

Thus there is no unique concrete temporal database that encodes a given abstract temporal database. In the last section we show how to choose a representative for every equivalence class of $\mathcal{D}_I$ generated by $\|.\|$ that has amenable properties.

2.2 Temporal Query Languages

In this section we define the first-order query languages over the classes $\mathcal{D}_P$ and $\mathcal{D}_I$.

Definition 2.7 (Point-based Language $\mathbf{L_P}$)
Let ρ be a database schema and

$$L ::= R_i(t, \mathbf{x}) \mid L \wedge L \mid \neg L \mid \exists x.L \mid \exists t.L \mid x_1 = x_2 \mid t_1 < t_2$$

where R_i is the extension of the predicate symbol $r_i \in \rho$. The language L_P is defined as a set of formulas $\varphi \in L$, such that $FV(\varphi)$[4] contains exactly one temporal variable t. An answer *to a query $\varphi \in L_P$ over $\mathrm{D} \in \mathcal{D}_P$ is the set $\{t, \mathbf{x} : \mathrm{D} \models \varphi(t, \mathbf{x})\}$.*

The condition on the free variables of the queries in L_P enforces closure of the language with respect to the temporal database schema. Similarly, in the interval-based case:

Definition 2.8 (Interval-based Language $\mathbf{L_I}$)
(interval-based language with explicit endpoints). Let ρ be a database schema and

$$L ::= R_i(I, \mathbf{x}) \mid L \wedge L \mid \neg L \mid \exists x.L \mid \exists I.L \mid x_1 = x_2 \mid I_1^* < I_2^*$$

where R_i is the extension of the predicate symbol $r_i \in \rho$ and $I^* \in \{I^+, I^-\}$. The language L_I is defined as a set of formulas $\varphi \in L$, such that $FV(\varphi)$ contains exactly one interval variable I. An answer *to a query $\varphi \in L_I$ over $\mathrm{D} \in \mathcal{D}_I$ is the set $\{I, \mathbf{x} : \mathrm{D} \models \varphi(I, \mathbf{x})\}$.*

Note that L_I uses the construction $I_1^* < I_2^*$ instead of the relation symbols $<_{--}, <_{+-}, <_{-+}, <_{++}$ that refer to the *actual* structure of the interval-based temporal domain T_I. However, it is easy to see that, e.g., $I <_{--} J \iff I^- < J^-$, where the $<$ on the right-hand side is interpreted in $(T, <)$ (and similarly for the other cases) and the new notation is merely a syntactic sugar. We assume the usual Tarskian semantics for formulas in both of the temporal query languages [1] (where $\models$ denotes the satisfaction relation for first-order logic). The usual interval-based query languages (e.g., TQUEL or TSQL2) can be defined in terms of the language L_I introduced in Definition 2.8[5]. We illustrate this fact by defining a query language that uses Allen's interval algebra operators (the most common choice among the interval-based query languages):

[4] The set of free variables in φ.

[5] Of course, only the first-order fragments of those languages. No aggregation or other non first-order features.

Remark 2.9 (Interval-based Language $\mathbf{L_{Allen}}$)
(interval-based language based on the Allen's Algebra Operators [2]). Let ρ be a database schema.

$$L ::= R_i(I, \mathbf{x}) \mid L \wedge L \mid \neg L \mid \exists x.L \mid \exists I.L \mid x_1 = x_2 \mid I_1 \rho I_2$$

where $\rho \in \{$precedes, meets, overlaps, equals, contains$\}$.

Clearly, all the queries in this language are definable in L_I (follows immediately from the definition of Allen's Algebra operators [2]). The opposite direction follows from the observation that, e.g.:

$$\begin{aligned} I^- \leq J^- \quad \iff \quad & (I \text{ contains } J) \vee (I \text{ overlaps } J) \vee \\ & (I \text{ meets } J) \vee (I \text{ precedes } J) \end{aligned}$$

(and similarly for all the remaining cases). Thus the language that uses Allen's operators L_{Allen} is equivalent to L_I. Henceforth we only use the language L_I.

3 Mapping Point-based Queries to Interval-based Queries

In this section we develop a mapping F that allows us to ask point-based queries over interval-based temporal databases. The mapping has to satisfy the following two requirements for every finite concrete database $\mathrm{D} \in \mathcal{D}_I$, tuple $\mathbf{x} \in |D|^k$, and query $\varphi \in L_P$:

1. $\|\mathrm{D}\| \models \varphi(t, \mathbf{x})$ if and only if $\exists I.\mathrm{D} \models F(\varphi)(I, \mathbf{x})$ and $t \in I$, and
2. $\{I : \mathrm{D} \models F(\varphi)(I, \mathbf{x})\}$ is finite.

Clearly (1) could be easily met by using unit intervals in the place of point variables through the original query, e.g., $t \mapsto I_t$ and $\forall t.\varphi \mapsto \forall I.(I^- = I^+) \supset \varphi[t/I]$ [8]. However, this solution (also clearly) violates the second requirement.

The main idea behind our transformation lies in the separation of the bindings of data variables $\mathbf{x}$ from the bindings of the temporal variables $\mathbf{t}$ in a given formula:

$$\varphi(\mathbf{t}, \mathbf{x}) \mapsto \exists \mathbf{I}.\varphi^I(\mathbf{I}, \mathbf{x}) \wedge \varphi^<(\mathbf{I}, \mathbf{t})$$

where $\varphi^I \in L_I$ and $\varphi^< \in L_<$ (the first-order language of linear order, where the endpoints of intervals I^+, I^- can be thought of as constants). This separation forms an invariant in an inductive transformation based on the structure of the input formula (in general we will need a finite disjunction of formulas on the RHS).

The reason for this separation is that in the original formula the data quantifiers ($\exists x$) apply only to φ^I (and can be dealt with using projection in the translation of the formula), while the temporal quantifiers ($\exists t$) are eliminated during the translation process using a *quantifier elimination procedure* for $Th(<_{Lin})$ on the formula $\varphi^<$. Note that while L_I does not contain any quantifiers over the point temporal variables, it still contains quantifiers over the interval variables. As a side-effect, no comparisons of the form $t_1 < t_2$ remain in the resulting formula. The single remaining temporal variable is used to describe the intervals in the answer of the query (cf. Definition 2.8).

Notation 3.1 In the sequel we use $x, y, \ldots$ to denote variables of the sort D; $t, t_1, t_2, s, \ldots$ to denote variables of the sort T_P, and $I, J, I_1, \ldots$ to denote variables of the sort T_I. We use bold letters to denote finite sets of variables, e.g., $\mathbf{x}$ is a shorthand for $x_1, \ldots, x_k$ for an appropriate k, etc. We also use $t \in I$ as a shorthand for $I^- \leq t \leq I^+$.

To proceed with the technical development, consider the following definition:

Definition 3.2 *A query is in* simple normal form *if it is of the form $\exists\mathbf{I}(\varphi^I \wedge \varphi^<)$ where $\varphi^I \in L_I$ and $\varphi^< \in L_<$. A query is in* normal form *if it is a finite disjunction of formulas in simple normal form.*

Note that a formula in normal form is not necessarily an element of the target interval-based temporal query language L_I. This is due to the presence of the subformula $\varphi^< \in L_<$. However, it always can be *interpreted* over an interval-based database (the φ^I part can be evaluated over the interval based temporal database immediately; the $\varphi^<$ is then used to restrict the (meaning of the) result obtained by evaluating the φ^I formula). The subformula $\varphi^< \in L_<$ is always quantifier free, and thus the whole formula contains only quantifiers over intervals and data variables. Moreover:

Lemma 3.3 *For every positive boolean combination ψ of formulas in normal form there is a formula in normal form equivalent to ψ.*

P r o o f: Follows from DeMorgan's laws by induction on structure of ψ.

Lemma 3.4 *Let φ be a formula in simple normal form. Then there is an equivalent formula of the form*

$$\exists\mathbf{I}(\varphi^I \wedge \mathbf{t} \in \mathbf{I}) \wedge \varphi^<$$

where $\mathbf{t}$ is the set of free temporal variables of φ, $\varphi^I \in L_I$, and $\varphi^<$ is a finite conjunction of formulas of the form $t_i < t_j$.

P r o o f: For every $t \in \mathbf{t}$ we find all upper and all lower bounds I^* in φ. This way we form the conjunct $\mathbf{t} \in \mathbf{I}$ (as the max and min functions are first-order definable). The remaining formulas are of the form $t_i < t_j$ and are left as the $\varphi^<$ conjunct. These conjuncts do not contain any references to the interval variables I and thus can be pulled out of the scope of the quantifier $(\exists\mathbf{I})$.

Lemma 3.5 *Let φ be a formula of the form*

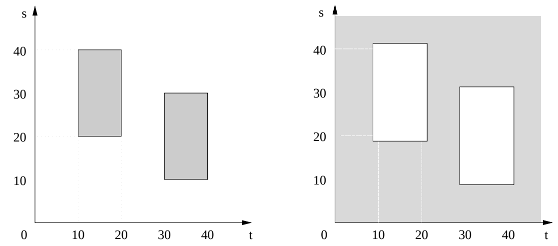
$$\exists\mathbf{I}(\varphi^I \wedge \mathbf{t} \in \mathbf{I}).$$

Then there is a formula ψ of the form $\exists\mathbf{I}(\psi^I \wedge \mathbf{t} \in \mathbf{I})$ such that $(\neg\varphi) \equiv \psi$ in $Th(<_{Lin})$.

P r o o f: Let $\varphi^I = \varphi(\mathbf{I}, \mathbf{x})$ and

$$\begin{aligned}\bar{\varphi}(\mathbf{I}, \mathbf{x}) :=& \\ &\textstyle\bigwedge_{I \in \mathbf{I}} \big((\exists\mathbf{I}'.\varphi(\mathbf{I}', \mathbf{x}) \wedge I^+ = I'^- - 1 \vee I^+ = \infty) \wedge \\ &\qquad (\exists\mathbf{I}'.\varphi(\mathbf{I}', \mathbf{x}) \wedge I^- = I'^+ + 1 \vee I^- = -\infty) \big)\end{aligned}$$

Then $\neg\varphi \equiv \exists\mathbf{I}(\bar{\varphi}(\mathbf{I}, \mathbf{x}) \wedge \forall\mathbf{I}'(\varphi(\mathbf{I}', \mathbf{x}) \supset \mathbf{I} \cap \mathbf{I}' = \emptyset) \wedge \mathbf{t} \in \mathbf{I})$ with respect to $Th(<_{Lin})$ assuming the standard rules for $+1$ and -1 over $Z \cup \{-\infty, \infty\}$ and $\mathbf{I} \cap \mathbf{I}' = \emptyset$ is a shorthand for $\bigwedge_{I \in \mathbf{I}} (I^+ < I'^-) \vee (I'^- < I^+)$. (cf. Figure 1).



$\varphi(t, s, a)$ $\qquad$ $\neg\varphi(t, s, a)$

Let $a \in |D|$ such that the formula $\varphi(t, s, a)$ holds for all $(t, s) \in [10, 20] \times [20, 40] \cup [30, 40] \times [10, 30]$. Then $\neg\varphi(t, s, a)$ holds for all pairs of values (s, t) that belong to any interval (rectangle) with end-points in $\{-\infty, 9, 21, 29, 41, \infty\}$ for the coordinate t and $\{-\infty, 9, 19, 31, 41, \infty\}$ for s that does not intersect any of the rectangles on the left hand side on the figure. The result is shown on the right hand side of the figure.

Figure 1: Complementation of φ for a fixed data constant a (and a discrete temporal domain).

Note that while the relation symbols in the database schema always contain exactly one temporal variable. This is no longer true for arbitrary queries (cf. Appendix A, query 4). This situation can not be avoided in general [13].

Theorem 3.6 *Let $\varphi \in L_P$. Then there exists an equivalent formula in normal form.*

P r o o f: Essentially we are trying to show that $\varphi\|\mathrm{D}\| = \|F(\varphi)(\mathrm{D})\|$ for all $\mathrm{D} \in \mathcal{D}_I$ and $\varphi \in L_P$. By induction on the structure of φ. Base case (atomic formulas):

$$\begin{array}{lcl} R(t, \mathbf{x}) & \mapsto & \exists I_t . R(I_t, \mathbf{x}) \wedge (t \in I_t) \\ t_i < t_j & \mapsto & \quad\text{true} \quad \wedge\ t_i < t_j \\ x_i = x_j & \mapsto & \quad x_i = x_j \ \wedge\ \text{true}. \end{array}$$

Moreover, we assume that the variable name I_t is unique during the whole translation.

$$
\begin{array}{rcll}
\neg F(\varphi) & \equiv & \neg \bigvee \exists \mathbf{I}(\varphi^I \wedge \varphi^<) & \text{by IH} \\
& \equiv & \neg \bigvee (\exists \mathbf{I}(\varphi^I \wedge \mathbf{t} \in \mathbf{I}) \wedge \varphi'^<) & \text{by Lemma 3.4 applied on every disjunct} \\
& \equiv & \bigwedge (\neg \exists \mathbf{I}(\varphi^I \wedge \mathbf{t} \in \mathbf{I}) \vee \neg \varphi'^<) & \text{by DeMorgan laws} \\
& \equiv & \bigwedge (\exists \mathbf{I}(\varphi'^I \wedge \mathbf{t} \in \mathbf{I}) \vee \neg \varphi'^<) & \text{by Lemma 3.5 applied on every disjunct} \\
& \equiv & \bigvee \exists \mathbf{I}(\varphi''^I \wedge \varphi''^<) & \text{by Lemma 3.3 and closure of } L_< \text{ under } \neg
\end{array}
$$

Figure 2: Translation of $\neg\varphi$ to normal form.

Induction: Let $F(\varphi_i) = \bigvee \exists \mathbf{I}_i(\varphi_i^I \wedge \varphi_i^<)$ be formulas in normal form equivalent to φ_i for $i = 1, 2$ with no common interval variables. Then by case analysis:

$\varphi_1 \vee \varphi_2$**:** In this case we simply merge the finite disjunctions representing both subformulas into a single disjunction $\bigvee \psi_i$ where ψ_i is a disjunct in $F(\varphi_1)$ or $F(\varphi_2)$.

$\neg\varphi$**:** This is the most complicated case. Figure 2 shows the detailed steps needed to convert the negated subformulas.

$\exists x.\varphi$**:** Here we use the properties of the normal form: $\exists x$ commutes freely with disjunction and other existential quantifiers. Also x does not appear in $\varphi^<$ and thus the existential quantifier can be pushed into the inner conjunction. The resulting formula is $\bigvee \exists \mathbf{I}.(\exists x.\psi_j^I) \wedge \psi_j^<$

$\exists t.\varphi$**:** Similarly to the previous case, the quantifier commutes with disjunction, existential quantification, and can be pushed into the conjunction as this time t does not appear in φ^I. However, in this case we need to eliminate the quantifier as L_I does not allow quantifiers ($\exists t$) over T_P. For this purpose we use the quantifier elimination procedure of $Th(<_{Lin})$ applied on $\exists t.\varphi^<$. The resulting formula is $\bigvee \exists \mathbf{I}.\psi_j^I \wedge \mathrm{QE}_t(\psi_j^<)$ where $\mathrm{QE}_t(\psi_j^<)$ is the result of applying the quantifier elimination on $\exists t.\varphi_j^<$.

Corollary 3.7 *For every query in L_P there is an equivalent query in L_I.*

P r o o f: Immediate from Theorem 3.6 by observation that queries in L_P can have at most one free temporal variable t. Thus by application of Lemma 3.4 we obtain a query in L_I as there are no inequalities of the form $t_i < t_j$ remaining in the normal form.

Theorem 3.6 is proven for a minimal complete set of first-order connectives. However, for practical purposes we introduce a direct conversion for conjunction:

$$F(\varphi_1 \wedge \varphi_2) = (\bigvee \varphi_{1,i}) \wedge (\bigvee \varphi_{2,i}) = \bigvee \psi_i$$

using Lemma 3.3. This is especially important from the implementation point of view. The conjunction rule is needed to preserve evaluability of the resulting queries.

4 Range-restricted Temporal Queries

In Section 3 we have introduced a transformation that allows point-based queries to be asked over interval-based temporal databases. However, to make the transformation a practical tool for querying temporal databases, we need to address the issue of evaluability (or safety): the only practical queries are those which can be evaluated by an algorithm. This leads to the requirement that the answers and the intermediate results to the queries are finite. In relational calculus this leads to the requirement that the set of all possible valuations for all the individual variables in the query has to be finite (range-restricted)[6]. We generalize this notion to the temporal databases relatively to the chosen encoding of sets of time instants:

Definition 4.1 (Range-restricted Query) *Let $\varphi \in L_P$, $F : L_P \to L_I$ a mapping, and* rr *be a syntactic criterion that decides if a query in L_I is range-restricted. Then φ is a* range-restricted query with respect to F *if* $\mathrm{rr}(F(\varphi))$.

This definition allows us to transfer the issue of range-restricted queries from the target (interval-based) language to the point-based language L_P. Our goal is to isolate a subclass of L_P that is guaranteed to map onto the range-restricted queries in L_I.

The use of the criteria for relational calculus to test range-restrictions on the individual variables in a L_I query is quite natural: the queries in L_I are merely two-sorted queries over finite relational databases. The underlying meaning of the representation is irrelevant in this case.

We show that the following two claims holds for the language L_P:

1. the range-restriction of data variables can be tested identically as in the case of relational calculus and
2. the range-restriction of temporal variables is not an issue with respect to the mapping F.

Before we isolate the class of range-restricted formulas of L_P we need to define a subclass of the abstract temporal databases that can be finitely represented by concrete temporal databases:

[6] For discussion of topics related to evaluability, safety, and domain independence see, e.g., [1, 14, 15].

Definition 4.2 *An abstract temporal database is* finitary *if it is an image of a finite concrete temporal database under* $\|.\|$.

We can equivalently describe the class of finitary temporal databases by looking on the sets of time instants associated with the individual (data) tuples stored in the database relations. When using the interval-based temporal databases as the encodings of the abstract temporal databases we can describe the finitary property using the following observation:

Definition 4.3 *Let* D *be an abstract temporal database. A* support *of a formula* φ *under a valuation* $\mathbf{a}$ *is the set* $\{t : \mathrm{D} \models \varphi(t, \mathbf{a})\}$.

This leads immediately to an equivalent formulation of the finitary property of abstract temporal databases with respect to the class of concrete temporal databases based on *intervals*:

Lemma 4.4 *An abstract temporal database is* finitary *if and only if*

1. *it contains a finite number of facts and*
2. *the support of every fact can be represented as a finite union of intervals.*

Clearly, the second condition is preserved under arbitrary boolean operations on formulas (including $\neg$) and quantification over the temporal domain. The only problems may be caused by quantification over the data domain:

Theorem 4.5 *Let all data variables in* $\varphi \in L_P$ *be range restricted. Then* $F(\varphi)$ *is range restricted in the class of finite interval-based temporal databases.*

Proof: If φ (or some of its subformulas) has the form $\exists x.\psi$, then by the assumption that x is range-restricted, the support of $\exists x.\psi$ is a finite union of intervals if and only if the support of $\psi(x)$ is a finite union of intervals for every value in the range of x. The rest is an induction on the structure of φ.

This theorem also shows that there are queries that are *not* range-restricted over the point-based temporal databases but can be transformed to equivalent range-restricted queries over the interval-based encodings of such databases (cf. Appendix A, query 2). This fact is due the closure under complementation over the interval-based temporal domain.

5 Generic Queries and Expressive Power

In Section 2 we have observed that the mapping $\|.\|$ is not injective. This fact allows a single abstract temporal database to be represented by several different concrete temporal databases. The mapping $\|.\|$ naturally factors the class of concrete temporal databases $\mathcal{D}_I$ into equivalence classes. Every concrete temporal database in the same equivalence class generated by $\|.\|$ has the same *meaning*. This observation raises important questions:

1. is there a *canonical* representative of every equivalence class generated by $\|.\|$?
2. are there formulas in L_I that distinguish elements in the same equivalence class generated by $\|.\|$; if they exist, what do they mean?
3. Can all L_I queries be expressed equivalently in the language L_P?

The answer to the first question is positive:

Definition 5.1 (Flat Interval Database) *Let* $\mathrm{D} \in \mathcal{D}_I$. *We call* D a flat interval database *if*

$$((I, \mathbf{x}) \in \mathbf{R}_i \wedge (J, \mathbf{y}) \in \mathbf{R}_i) \supset (\mathbf{x} \neq \mathbf{y} \vee I \cap J = \emptyset)$$

for all relations $\mathbf{R}_i$ *such that* $r_i \in \rho(\mathrm{D})$, $I, J \in T_I$, *and* $\mathbf{x}, \mathbf{y} \in |D|^{\mathrm{arity}(r_i)}$.

The definition essentially says that the intervals in tuples with the same data values may not overlap. The Flat Interval Databases are also the smallest (with respect to cardinality) and thus occupy the least amount of computer storage.

Lemma 5.2 *For every* $\mathrm{D} \in \mathcal{D}_I$ *there is a flat* $\mathrm{D}' \in \mathcal{D}_I$ *such that* $\|\mathrm{D}\| = \|\mathrm{D}'\|$. *Moreover,* $\mathrm{card}(\mathrm{D}) \geq \mathrm{card}(\mathrm{D}')$.

This is important from the implementation point of view as space-efficiency is a major concern.

The answer to the second question is (unfortunately) positive as well. There are queries in L_I whose semantics is not well defined as their answers depend on the particular representation of a given abstract temporal database:

Example 5.3 *Let* $\mathrm{D}_1, \mathrm{D}_2 \in \mathcal{D}_I$ *be two databases over the database schema* $(r(x))$ *defined by*

$$\mathbf{R}^{\mathrm{D}_1} = \{([0,3], a)\} \text{ and } \mathbf{R}^{\mathrm{D}_2} = \{([0,2], a), ([1,3], a)\}.$$

Then the formula $\exists I, J. \exists x (R(I, x) \wedge R(J, x) \wedge I \neq J)$ *is true in* D_2 *but false in* D_1.

This is obviously a problem, as both D_1 and D_2 represent the same abstract temporal database D where $\mathbf{R}^{\mathrm{D}} = \{(1, a), (2, a), (3, a)\}$. We introduce the notion of *generic queries*—queries that are independent of the actual representation of the abstract temporal database:

Definition 5.4 (Generic Queries) *Let* $\|.\|$ *be a mapping* $\mathcal{D}_I \to \mathcal{D}_P$ *and* $\varphi \in L_I$. *Then we say that* φ is generic with respect to $\|.\|$ *if*

$$\|\mathrm{D}_1\| = \|\mathrm{D}_2\| \supset \|\varphi \mathrm{D}_1\| = \|\varphi \mathrm{D}_2\|$$

for all $\mathrm{D}_1, \mathrm{D}_2 \in \mathcal{D}_I$.

This notion is similar to the genericity notion for standard relational queries [4]; the definition allows only queries that are representation-independent.

We can show that all generic queries of L_I can be equivalently asked using the language L_P:

Theorem 5.5 *Let $\varphi \in L_I$ be a generic query with respect to $||.||$ and $\mathrm{D} \in \mathcal{D}_I$ an interval database. Then there is $\psi \in L_P$ such that $\psi||\mathrm{D}|| = ||\varphi\mathrm{D}||$.*

Proof: We just make the endpoints of the intervals explicit:

$$R(T, \mathbf{x}) \mapsto \neg R(t^- - 1, \mathbf{x}) \wedge \neg R(t^+ + 1, \mathbf{x}) \\ \wedge \forall t.(t^- \leq t \leq t^+ \supset R(t, \mathbf{x}))$$

The rest of the query remains unchanged (after replacing T^+ by t^+ and T^- by t^-). Because φ is generic, it gives the same answer on all $||.||$-equivalent concrete temporal databases, especially on the appropriate flat temporal database. The conclusion follows from the observation that a flat temporal database exists in all the $||.||$-equivalence classes by Lemma 5.2.

6 Conclusion and Future Work

We have established a correspondence between the point-based and the interval-based views of temporal databases and studied properties of this correspondence.

In [3] a transformation mapping first-order temporal logic formulas to TSQL2 was proposed. The goal of that paper is similar to our goal. However, the presented transformation can not cope with the full first-order logic over temporal databases as it depends on the *single-dimensional* nature of temporal logic[7] [13]. On the other hand, it can use this property of temporal logic to make the transformation simpler.

Future research includes the following topics:

Other classes of concrete temporal domains. When we carefully analyze the Definition 2.3, we can see that the intervals are essentially quantifier free formulas in $Th(<_{Lin})$ with exactly one free variable. This idea can be generalized to other theories: Let (T, σ) be a point-based temporal domain over $Th(\sigma)$. Then we can define the set $C_\sigma = \{\phi(t) : \phi \in L_\sigma \wedge FV(\phi) = \{t\}\}$ where L_σ is the set of finite conjunctions of atomic formulas in the language of σ. This set can serve as the carrier of the temporal domain for the class of concrete temporal databases. However, $Th(<_{Lin})$ is a very well behaved theory. To use $Th(\sigma)$ instead, we need to answer following questions:

- Can the elements of C_σ be uniformly represented by tuples of elements of T (or $T \cup \{\infty\}$)?
- What are the appropriate relations on C_σ (i.e., the counterparts of the $<_{--}, <_{-+}, <_{+-}$, and $<_{++}$ relations of T_I)?
- Does $Th(\sigma)$ admit quantifier elimination? If so, is the quantifier elimination *value independent* (i.e., does the shape of the result of the quantifier elimination depend on the actual values present in the input formula or only on the structure of the input formula)? This is an important issue, as the quantifier elimination is a part of the *translation* process where the actual values are not known.
- Is complementation in $Th(\sigma)$ first-order definable (as in Lemma 3.5) in the first-order query language over the concrete temporal databases?

Note that a general solution may require a limited second-order quantification [12].

Optimization. The formulas produced by the transformation F can be exponential in the size of the input formula. This blowup comes from two sources:

1. The use of Lemma 3.5 on the negated subformulas of the original query. However, the exponential blowup can be avoided by introducing views in place of the replicated subformulas (then the remaining cumulative size of the views and the remaining formula is at most quadratic), and
2. The use of Lemma 3.3 that converts boolean combinations to normal form. This blowup can not be avoided in general[8].

The other possibility for optimization is to utilize additional constructs in the language L_I: The complementation in $Th(<_{Lin})$ is first-order expressible (cf. Lemma 3.5). However, for a linearly ordered universe it could be easier to "scan" the values stored in the temporal relations after sorting them in, e.g., the lexicographical order of temporal dimensions. What are the appropriate extensions to L_I that exploit this feature of linearly ordered sets? A canonical representation of products of sets of intervals similar to Definition 5.1[9] needs to be developed in the case of dimension higher than one to facilitate efficient query evaluation.

Spatial and other higher-dimensional extensions. The current version of the proposed transformation F depends on the restriction of queries to a single-dimensional model of time. However, the only problem with adopting a multi-dimensional models of time (or

[7] The number of free temporal variables is exactly one in every subformula of a given temporal logic formula.

[8] This is similar to the problem in the first-order logic to relational algebra translations, e.g., [1, 15], where the conjunctions need to be distributed into disjunctions (in the worst case) to guarantee evaluability of the resulting query.

[9] For the purpose of our translation, the enforcement of maximal intervals is not essential, except for performance reasons.

space) is the closure with respect to the target interval-based query language: if we allow, e.g., two-dimensional queries, then the formula $\varphi^<$ in the result of the translation defined in Theorem 3.6 may contain a subformula of the form $t_i < t_j$. The semantics of such formula can not be captured by (products) of single-dimensional intervals and thus can not be expressed in L_I. There are two possible solutions to this problem:

1. we can enrich the two-dimensional temporal domain with additional elements and constructs (built-in relations) that allow us to capture the semantics of such formulas or

2. we can restrict the two-dimensional queries to those, where such dependency between the two dimensions does not exist (similarly to [6]).

The extension to dimension higher than two is then straightforward.

References

[1] Abiteboul, S., Hull, R., Viannu, V. Foundations of Databases. Addison Wesley, 1995.

[2] Allen, J. F. Maintaining Knowledge about Temporal Intervals. *Comm. ACM*, 16(11):832–843, 1983.

[3] Bohlen, M. H., Chomicki, J., Snodgrass, R. T., Toman, D. Querying TSQL2 Databases with Temporal Logic. To appear in Proc. EDBT'96.

[4] Chandra, A. K., Harrel, D. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences,* 21:156–178, 1980.

[5] Chomicki J. Temporal Query Languages: a Survey. Proc. *International Conference on Temporal Logic,* July 1994, Bonn, Germany, Springer-Verlag (LNAI 827), pp. 506–534.

[6] Chomicki, J., Goldin, D. Q., Kuper, G. Variable Independence and Aggregation Closure. To appear in Proc. *ACM Symposium on Principles of Database Systems,*, 1996.

[7] Kanellakis, P. C., Kuper, G. M., Revesz, P. Z. Constraint Query Languages, Proc. *9th ACM Symposium on Principles of Database Systems,* pp 299–313, 1990.

[8] Ladkin, P. Satisfying FO Constraints About Time Intervals. Proc. *AAAI'88*, pp. 512–517, 1988.

[9] Snodgrass R. T. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[10] Snodgrass R. T. Temporal Databases: status and research directions. *ACM SIGMOD Record*, 19(4):83–89, 1990.

[11] Snodgrass R.T. (editor). *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 674+xxiv pages, 1995.

[12] Tansel, A. et al. (editors). Temporal Databases: Theory, Design, and Implementation, Benjamin Cummings 1993.

[13] Toman, D., Niwinski, D. First-Order Temporal Queries Inexpressible in Temporal Logic. To appear in Proc. EDBT'96.

[14] Ullman J. D. Principles of Database and Knowledge-base Systems, Vol. 1, 2. Computer Science Systems, 1989.

[15] Van Gelder A., Topor R. W. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(2):235–278, June 1991.

A Examples of Queries

Example 1. Let Salary be a relation holding names and history of employee salaries. Query: *Find names of everyone whose salary has decreased*:

$$\{t, x : \exists t_1, t_2 . t_1 > t_2 \wedge \exists s_1, s_2 . s_1 < s_2 \wedge \\ \text{Salary}(t_1, x, s_1) \wedge \text{Salary}(t_2, x, s_2)\}$$

Note that the variable t does not appear in the query. This is not a problem; the answer to the query is supposed to be *true at all time instants*.

This query is easy enough to be formulated in L_I (by hand):

$$\{[-\infty, \infty], x : \exists I_1, I_2 . I_1^+ > I_2^- \wedge \exists s_1, s_2 . s_1 < s_2 \wedge \\ \text{Salary}(I_1, x, s_1) \wedge \text{Salary}(I_2, x, s_2)\}$$

It is enough to realize that $I_1^+ > I_2^-$ implies the existence of time instants t_1, t_2 with the required properties. Our transformation F finds exactly this solution.

Example 2. Let Indep be a relation that contains names and periods of independence of countries. Query: *Find all periods when no country was independent*:

$$\{t : \neg \exists x . \text{Indep}(t, x)\}$$

This is slightly tricky as the variable t appears only under negation and thus appears not to be range-restricted. Our transformation produces the following result (essentially by applying Lemma 3.5):

$$\{I : \exists I' (\psi(I') \wedge I^+ = I'^- - 1 \vee I^+ = \infty) \wedge \\ \exists I' (\psi(I') \wedge I^- = I'^+ + 1 \vee I^- = -\infty) \wedge \\ \forall I' (\psi(I') \supset I \cap I' = \emptyset)\}$$

where $\psi(I)$ is a shorthand for $\exists x . \text{Indep}(I, x)$.

Example 3. A temporal query from [5] over the same database as in (2) is: *Find all the countries that lost and regained their independence (list names and periods between two independence periods)*:

$$\{t, x : \exists t_1, t_2 . t_1 < t < t_2 \wedge \\ \mathrm{Indep}(t_1, x) \wedge \mathrm{Indep}(t_2, x) \wedge \neg \mathrm{Indep}(t, x)\}$$

The interval solution for this query is not easy: a *naive* attempt

$$\{I, x : \exists I_1, I_2 . I_1^+ < I^- \wedge I^+ < I_2^- \wedge \\ \mathrm{Indep}(I_1, x) \wedge \mathrm{Indep}(I_2, x) \wedge \neg \mathrm{Indep}(I, x)\}$$

fails for the non-flat temporal databases. Moreover, the variable I is not range-restricted.

The transformation F converts the query as follows (we show the interesting steps of the transformation). In the first step the atomic formulas are replaced by their interval-based equivalents:

$$\begin{aligned}\{t, x : \exists t_1, t_2 . &\exists I_1 (\mathrm{Indep}(I_1, x) \wedge t_1 \in I_1) \wedge \\ &\exists I_2 (\mathrm{Indep}(I_2, x) \wedge t_2 \in I_2) \wedge \\ &\neg \exists I (\mathrm{Indep}(I, x) \wedge t \in I) \wedge t_1 < t < t_2\}\end{aligned}$$

The second step uses the rule for processing $\neg$:

$$\begin{aligned}\{t, x : &\exists t_1, t_2 . \\ &\exists I_1 (\mathrm{Indep}(I_1, x) \wedge t_1 \in I_1) \wedge \\ &\exists I_2 (\mathrm{Indep}(I_2, x) \wedge t_2 \in I_2) \wedge \\ &\exists I (\exists I' (\mathrm{Indep}(I', x) \wedge I^+ = I'^- - 1 \vee I^+ = \infty) \wedge \\ &\quad \exists I' (\mathrm{Indep}(I', x) \wedge I^- = I'^+ + 1 \vee I^- = -\infty) \wedge \\ &\quad \forall I' (\mathrm{Indep}(I', x) \supset I \cap I' = \emptyset) \wedge t \in I) \wedge \\ &t_1 < t < t_2\}\end{aligned}$$

Now by Lemma 3.3 we obtain a formula in normal form. This step also separates the pieces of the formula that are already in L_I from those pieces which still contain comparisons of the form $t_i < t_j$:

$$\begin{aligned}\{t, x : &\exists t_1, t_2 . \exists I, I_1, I_2 . \mathrm{Indep}(I_1, x) \wedge \mathrm{Indep}(I_2, x) \wedge \\ &\exists I' (\mathrm{Indep}(I', x) \wedge I^+ = I'^- - 1 \vee I^+ = \infty) \wedge \\ &\exists I' (\mathrm{Indep}(I', x) \wedge I^- = I'^+ + 1 \vee I^- = -\infty) \wedge \\ &\forall I' (\mathrm{Indep}(I', x) \supset I \cap I' = \emptyset) \wedge \\ &(I_1^- \leq t_1 \leq I_1^+) \wedge (I^- \leq t \leq I^+) \wedge (I_2^- \leq t_2 \leq I_2^+) \wedge \\ &(t_1 < t < t_2)\}\end{aligned}$$

Now we can push the $\exists t_1, t_2$ into the formula and eliminate both the quantifiers $\exists t_1, t_2$:

$$\begin{aligned}\{t, x : &\exists I, I_1, I_2 . \mathrm{Indep}(I_1, x) \wedge \mathrm{Indep}(I_2, x) \wedge \\ &\exists I' (\mathrm{Indep}(I', x) \wedge I^+ = I'^- - 1 \vee I^+ = \infty) \wedge \\ &\exists I' (\mathrm{Indep}(I', x) \wedge I^- = I'^+ + 1 \vee I^- = -\infty) \wedge \\ &\forall I' (\mathrm{Indep}(I', x) \supset I \cap I' = \emptyset) \\ &\wedge (I_1^- + 1 \leq t) \wedge (I^- \leq t \leq I^+) \wedge (t \leq I_2^+ - 1)\}\end{aligned}$$

This leaves us with a formula in normal form containing exactly one free point-based temporal variable t. The condition on the variable t can be equivalently stated as

$$t \in [\max(I_1^- + 1, I^-), \min(I^+, I_2^+ - 1)].$$

Replacing the last line in the above formula by

$$I = [\max(I_1^- + 1, I^-), \min(I^+, I_2^+ - 1)]$$

gives the sought translation to L_I.

Example 4. The translation of queries containing negation can not be significantly simplified even if we use only flat interval databases. This fact is due to the impossibility of enforcing the "flat" condition on formulas with temporal dimension higher than one. Consider following example:

In the temporal database from Example (2) ask for *All pairs dates, when every country was independent at most at one of them (list the periods enclosed by these dates)*

$$\{t : \exists t_1, t_2 . \neg \exists x (\mathrm{Indep}(t_1, x) \wedge \mathrm{Indep}(t_2, x)) \wedge t_1 \leq t \leq t_2\}$$

To ask this query in the interval-based query language we would have to consider the intervals of independence of the individual countries and their interactions. However, by applying the mapping F we get:

$$\{I : \exists I_1, I_2 . \varphi(I_1, I_2) \wedge I = [I_1^-, I_2^+]\}$$

where

$$\begin{aligned}\varphi(I_1, I_2) = &\exists I_1', I_2' (\psi(I_1', I_2') \wedge I_1^+ = I_1'^- - 1 \vee I_1^+ = \infty) \wedge \\ &\exists I_1', I_2' (\psi(I_1', I_2') \wedge I_1^- = I_1'^+ + 1 \vee I_1^- = -\infty) \wedge \\ &\exists I_1', I_2' (\psi(I_1', I_2') \wedge I_2^+ = I_2'^- - 1 \vee I_2^+ = \infty) \wedge \\ &\exists I_1', I_2' (\psi(I_1', I_2') \wedge I_2^- = I_2'^+ + 1 \vee I_2^- = -\infty) \wedge \\ &\forall I_1', I_2' (\psi(I_1', I_2') \supset (I_1' \cap I_1 = \emptyset \wedge I_2' \cap I_2 = \emptyset))\end{aligned}$$

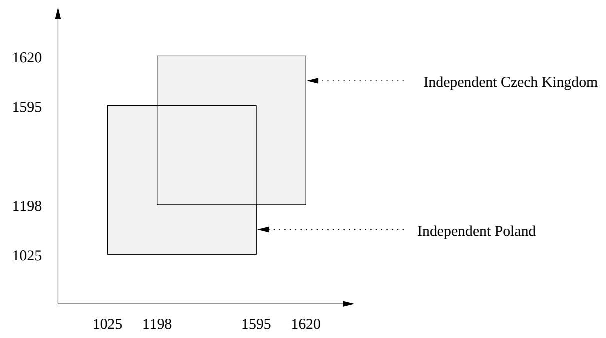
and

$$\psi(I_1, I_2) := \exists x (\mathrm{Indep}(I_1, x) \wedge \mathrm{Indep}(I_2, x))$$

by an application of Lemma 3.5. $\psi(I_1, I_2)$ is a part of the original query, where the occurrences of the symbol Indep have been modified using the rules for atomic formulas. It is easy to see the problem with the canonical form of relations for dimension higher than one. Consider an instance of the Indep relation

$$\mathrm{Indep} = \{\ ([1040, 1595], \mathrm{Poland}), \\ ([1198, 1620], \mathrm{Czech\ Kingdom})\}$$

This temporal database represents a fragment of a flat interval database [5] (immediate from Definition 5.1). However, the subformula $\psi(I_1, I_2)$ from the previous example evaluates to:



which is difficult to uniquely represent as a finite union of disjoint squares (2-dimensional intervals). The higher dimension can not be avoided in general [13].