Temporal versus First-Order Logic to Query Temporal Databases

Serge Abiteboul, Stanford University*
Laurent Herr, INRIA-Rocquencourt
Jan Van den Bussche, University of Antwerp†

Abstract

A database history can be modeled as a (finite) sequence of instances discretely ordered by time. Similarly, the behavior of a system such as an operating system or a reactive system can be modeled by an infinite such sequence. One can view the sequence as one single database where each relation has an additional column holding the time instant of validity of each tuple. The temporal database can then be queried using standard relational calculus (first-order logic) on this "timestamp" representation. One may alternatively use an implicit access to time and express queries in *temporal logic*. It is known that these two approaches yield the same expressive power in the propositional case. Their comparison in the predicate/database context remained open. We prove here that there are first-order logic queries on the timestamp representation that are *not* expressible in (extended) temporal logic. The proof technique is novel and is based on communication complexity.

*On leave from INRIA-Rocquencourt

†Work performed while on leave at INRIA-Rocquencourt. Post-doctoral research fellow of the Belgian National Fund for Scientific Research. Contact address: Jan Van den Bussche, Informatica , UIA, Universiteitsplein 1, B-2610 Antwerpen, Belgium. E-mail: vdbuss@uia.ua.ac.be.

1 Introduction

A database history can be modeled as a (finite) sequence of instances discretely ordered by time. Similarly, the behavior of a system such as an operating system or a reactive system can be modeled by an infinite such sequence. We are concerned here with querying sequences of database instances, also called (discrete-time) *temporal databases*. As discussed in Chomicki's excellent survey on the fundamental aspects of temporal query languages [3], there are two different approaches to query a temporal database using a first-order language:

1. One can view the sequence as one single relational database of an augmented schema where a "timestamp" column is added to each relation. The new column holds the time instants of validity of each tuple. This timestamp representation can then be queried using relational calculus with variables ranging either over data elements or over timestamps. The linear order on timestamps is given as a built-in relation. We denote relational calculus with explicit access to time by TS-FO (for timestamp-first-order logic).

2. Alternatively, one can use *temporal logic* [4] which provides a more "implicit" access to time. Standard temporal logic is an extension of classical logic with the temporal operators *since*, *until*, *next*, and *previous*. Observing that these operators can be viewed as searching for regular events, one can be more general and supply a temporal operator for each regular language [15]. We denote standard temporal logic by TL, and extended temporal logic (with general regular events) by ETL.

It is natural to compare these potential query languages. For *propositional* temporal databases (corresponding to the case where the schema consists of relations of arity zero, i.e., Boolean flags, only), this issue is well understood. Since the temporal operators of TL can easily be expressed in TS-FO, TL is trivially contained in TS-FO. A classical result ([10], see also [7]) states that the converse containment holds as well: TL and TS-FO are equivalent for propositional temporal databases. This is often referred to as the "expressive completeness" of propositional TL. It is also known that ETL is strictly stronger than these two languages in the propositional case: in ETL, one can express that a certain property is periodically true which is impossible in TS-FO.

How the languages compare in the general case remained open. In this paper, we show that some TS-FO queries are not expressible in ETL. Thus, TL is not expressively complete, and TS-FO and ETL are incomparable. We prove these results both for finite temporal databases and for infinite ones.

Evidences for this result already existed since 1971 [9]. Indeed, Kamp obtained results implying that TL is strictly weaker than TS-FO on the class of *densely* ordered temporal structures, an essentially different context. Moreover, Toman and Niwinski [13] (still in the densely ordered case) showed that no finite set of first-order temporal operators can be added to TL so as to achieve expressive completeness.

Also, in a previous paper, we established another failure in the discrete first-order case of a classical propositional result. Propositional TL with *next* and *until* only can simulate full propositional TL. We showed in [2] that this is not true in the predicate case. What was not pointed out there is that this failure implies, in turn, the failure in the discrete first-order case of the *separation property* [5], a most fundamental property of propositional TL known to be intimately related to its expressive completeness.

An example of a query separating ETL and TS-FO is *"are there two distinct time instants at which the staff consists of exactly the same employees?"* (on a temporal database recording the evolution of a company's staff). We prove our result by developing a fine analysis of TL computations on a class of temporal databases called "split" and using a proof technique based on *communication complexity* [16, 11]. To our knowledge, this is the first time this tool is employed to analyze the expressive power of query languages.

The paper is organized as follows. In Section 2, we provide the necessary background. In Section 3, we exhibit queries on finite temporal databases that are expressible in TS-FO but not in ETL. In Section 4 we show that the techniques also apply to the infinite case. Finally, in Section 5, we mention conditions on the database that bring TL up to the power of TS-FO. We also discuss alternative approaches that may provide different proofs of our result.

2 Preliminaries

We assume some familiarity with relational databases (see, e.g., [1]). A *database schema* is a finite set of relation names, where each relation name has an associated arity. An *instance* of a schema assigns to each relation name a *finite* relation of appropriate arity over a fixed countably infinite domain of data elements. The *active domain* of an instance is the set of all data elements appearing in some of its relations.

A *temporal database* over a database schema $\mathcal{S}$ is a non-empty finite sequence $\mathbf{I} = I_1, \ldots, I_n$ $(n \geq 1)$ of instances of $\mathcal{S}$. Every $j \in \{1, \ldots, n\}$ is called a *state* of $\mathbf{I}$. The *active domain* of a temporal database is the union of the active domains of its instances.

The timestamp representation and the language TS-FO**.** We can identify a temporal database $\mathbf{I}$ with a two-sorted relational structure called the *timestamp representation* of $\mathbf{I}$. *Data elements* are taken from the active domain of $\mathbf{I}$, whereas *timestamps* are from the set of states $\{1, \ldots, n\}$.[1] The timestamp representation also contains the linear order on the states as an explicit binary relation $<$. Furthermore, it contains, for each relation R of arity k in the database schema, an extended relation $\bar{R}$ of arity $k+1$. The first k columns of this relation hold data elements; the last column holds timestamps. The contents of this relation, denoted $\mathbf{I}(\bar{R})$, is

$$\bigcup_{j=1}^{n} (I_j(R) \times \{j\}).$$

Example 2.1 A temporal database over a schema consisting of a single unary relation S, together with its timestamp representation are shown in Figure 1. ■

Using (two-sorted) first-order logic on the timestamp representation of a temporal database, we obtain a query language that is denoted by TS-FO. (The *data-variables* range over data elements and *time-variables* over states.) The sorts of TS-FO variables will be clear from the context.

[1] For clarity, we assume without loss of generality that the domain of data elements is disjoint from the natural numbers. However, it is sometimes possible (and interesting) to simulate timestamps using data elements; we come back to this issue in Section 5.

$$\mathbf{I} = \begin{array}{ccc} I_1(R) & I_2(R) & I_3(R) \\ \boxed{\begin{array}{c} a \\ b \\ c \end{array}} & \boxed{\begin{array}{c} b \\ d \end{array}} & \boxed{\begin{array}{c} a \\ c \end{array}} \end{array} \qquad \begin{array}{c} I(\bar{R}) \\ \boxed{\begin{array}{cc} a & 1 \\ b & 1 \\ c & 1 \\ b & 2 \\ d & 2 \\ a & 3 \\ c & 3 \end{array}} \end{array}$$

Figure 1: A temporal database and its timestamp representation.

Example 2.2 If S is a unary relation holding employees of some company, the following TS-FO formula expresses that x is an employee who has been hired, later fired, and still later re-hired:

$$(\exists t_1)(\exists t_2)(\exists t_3)(t_1 < t_2 < t_3 \wedge \bar{S}(x, t_1) \wedge \neg\bar{S}(x, t_2) \wedge \bar{S}(x, t_3)). \qquad \blacksquare$$

Extended temporal logic. An alternative way of providing a temporal query language is to extend first-order logic with temporal operators rather than explicit time-variables. We will use temporal operators based on regular events, leading to *extended temporal logic*, denoted by ETL [15]. The syntax of ETL over some database schema $\mathcal{S}$ is obtained by using the formation rules for standard first-order logic over $\mathcal{S}$ together with one additional formation rule:

> Let L be a regular language over the finite alphabet $(v_1, \dots, v_p)$, and let $\varphi_1, \dots, \varphi_p$ be formulas. Then
>
> $$L^+(\varphi_1, \dots, \varphi_p) \qquad \text{and} \qquad L^-(\varphi_1, \dots, \varphi_p)$$
>
> are also formulas.

The order of the letters in the alphabet $(v_1, \dots, v_p)$ is relevant since it allows to relate these letters to the arguments $(\varphi_1, \dots, \varphi_p)$.

The semantics of ETL is as follows. Let $\mathbf{I} = I_1, \dots, I_n$ be a temporal database over $\mathcal{S}$. Let $\varphi(\bar{x})$ be an ETL formula with free variables $\bar{x} = x_1, \dots, x_k$, let $\bar{a} = a_1, \dots, a_k$ be data elements in the active domain of $\mathbf{I}$, and let $j \in \{1, \dots, n\}$ be a state. The *truth* of $\varphi[\bar{a}]$ in $\mathbf{I}$ at time j, denoted by $\mathbf{I}, j \models \varphi[\bar{a}]$, is defined as follows:

1. If φ is an atomic formula, a conjunction, a negation, or a quantification, the definition is as usual. Quantification is always on the active domain.

2. If φ is of the form $L^+(\varphi_1, \dots, \varphi_p)$, with L a regular language over the alphabet $(v_1, \dots, v_p)$, then $\mathbf{I}, j \models \varphi[\bar{a}]$ if there exists a word $w = v_{w_j} \dots v_{w_n}$ of length $(n - j + 1)$ in L such that

   $$\mathbf{I}, j \models \varphi_{w_j}[\bar{a}] \quad \text{and} \ \dots \ \text{and} \quad \mathbf{I}, n \models \varphi_{w_n}[\bar{a}].$$

3. Symmetrically, if φ is of the form $L^-(\varphi_1, \ldots, \varphi_p)$, then $\mathbf{I}, j \models \varphi[\bar{a}]$ if there exists a word $w = v_{w_j} \ldots v_{w_1}$ of length j in L such that

$$\mathbf{I}, j \models \varphi_{w_j}[\bar{a}] \quad \text{and} \ \ldots \ \text{and} \quad \mathbf{I}, 1 \models \varphi_{w_1}[\bar{a}].$$

Example 2.3 The ETL formula $L_1^+(\mathbf{true}, \varphi)$, where L_1 is the language a^*ba^* over the alphabet (a, b), is true at time j iff there is some time in the future of j (including j itself) where φ is true. Similarly, $L_1^-(\mathbf{true}, \varphi)$ expresses that φ holds sometime in the past.

Now recall Example 2.2. The following ETL formula is true of x at some time iff x is not a staff member now, has been one in the past, and will again be one in the future:

$$\neg S(x) \wedge L_1^-(S(x)) \wedge L_1^+(S(x)).$$

For another example, a formula which is true only in the last (or first) state is $L_2^+(\mathbf{true})$ (or $L_2^-(\mathbf{true})$), where L_2 is the singleton language $\{a\}$.

Finally, the formula $L_3^+(\mathbf{true})$, where L_3 is the language $(aa)^*$, is true in the first state iff the length of the temporal database is even. ■

Example 2.4 The previous example already showed how the familiar temporal operators "sometimes in the future" and "sometimes in the past" of standard temporal logic [4] can be expressed in ETL. We next show how the other temporal operators of standard temporal logic can be expressed.

The temporal connectives **since** and **until** can be expressed in ETL as follows:

$$\varphi \ \mathsf{since}\ \psi \equiv L_4^-(\varphi, \psi, \mathbf{true}) \qquad \text{and} \qquad \varphi \ \mathsf{until}\ \psi \equiv L_4^+(\varphi, \psi, \mathbf{true}),$$

where L_4 is the language a^*bc^* over the alphabet (a, b, c). The connectives **next** and **previous** are expressed as

$$\mathsf{next}\ \varphi \equiv L_5^+(\mathbf{true}, \varphi) \qquad \text{and} \qquad \mathsf{previous}\ \varphi \equiv L_5^-(\mathbf{true}, \varphi),$$

where L_5 is the language aba^* over the alphabet (a, b). ■

Temporal logic, i.e., the fragment of ETL having as only temporal operators **since**, **until**, **next**, and **previous**, is denoted by TL.

The above examples also illustrate a subtle feature of our definition. When searching for a regular event in the future (using the L^+ connective), we require that a word in L can be found which reaches precisely the last state of the temporal database. Similarly, when searching in the past, a word must be found which reaches precisely the first state. We refer to this as *full search*, as opposed to *partial search* which does not require the match to reach the beginning or end. As illustrated in some of the above examples, it is easy to simulate partial search using full search: it suffices to continue testing for **true** after the desired match has been found. Conversely, it can be shown (proof omitted) that full search can be simulated using partial search.

Boolean queries. A *Boolean query* on a class of temporal databases over some fixed schema is a mapping assigning true or false to each database in the class. Every TS-FO sentence (formula without free variables) defines a Boolean query in the obvious way. Also every ETL sentence φ defines a Boolean query Q in a natural way:

For temporal database $\mathbf{I}$, $Q(\mathbf{I}) = $ true iff φ is true on $\mathbf{I}$ at every state.

An alternative would be to require φ to be true only at the *first* state, but this leads to the same class of queries.

TL is obviously expressible in TS-FO. For example, to express that φ **until** ψ holds at t, one states that there exists $t' > t$ such that ψ holds at t' and φ holds at each t'' between t and t'. As shown in Example 2.3, the query "the length of the temporal database is even" is expressible in ETL. It is not expressible TS-FO, since parity of a linear order is well-known not to be first-order definable. In the next section, we will show that conversely, there are queries expressible in TS-FO but not in ETL. Since TL is a sublanguage of ETL, we will thus establish that TL is strictly less expressive than TS-FO.

3 TS-FO queries inexpressible in ETL

In this section, we first introduce a variant of the communication protocols of [16] (see also [11]), and introduce the notion of "constant communication complexity" of binary predicates on sets of sets (of data elements). We also introduce the class of *split* temporal databases. Each binary predicate on sets of sets gives rise to a query on split databases. We then prove that if the communication complexity of a predicate is not constant, then the corresponding query is not expressible in ETL. However, natural predicates of non-constant communication complexity exist whose corresponding queries *are* expressible in TS-FO.

3.1 Communication protocols

Let P be a binary predicate on sets of sets of data elements. We say that P has *constant communication complexity* if there exist fixed natural numbers k and r and a communication protocol between two parties (denoted by A and B) that, for each finite set D of data elements, can evaluate $P(X,Y)$ on any sets X and Y of non-empty subsets of D as follows:

1. A gets X and B gets Y. Both parties also know D.
2. A sends a message $a_1 = a_1(D,X)$ to B, and B replies with a message $b_1 = b_1(D,Y,a_1)$ to A. Each message is a k-ary relation on D.
3. A again sends a message $a_2 = a_2(D,X,b_1)$ to B, and B again replies with a message $b_2 = b_2(D,Y,a_1,a_2)$.
4. After r such message exchanges, both A and B have enough information to evaluate $P(X,Y)$ correctly. Formally, they apply a Boolean function $a_{r+1}(D,X,b_1,\ldots,b_r)$ (for A) or $b_{r+1}(D,Y,a_1,\ldots,a_r)$ (for B) which evaluates to true iff $P(X,Y)$ is true.

So, formally, a protocol consists of the functions $a_1, \ldots, a_r, a_{r+1}$ and $b_1, \ldots, b_r, b_{r+1}$. Note that the computing power of A and B is unlimited; the functions defining the protocol can be completely arbitrary.

Example 3.1 As a very simple example, let $P(X, Y)$ be true if the maximal cardinality of an element in X is larger than the maximal cardinality of an element in Y. Then P has constant communication complexity with $k = 1$ and $r = 1$. Indeed, A sends to B an element of X with maximal cardinality, and B replies with an analogous element for Y. Both A and B can then evaluate $P(X, Y)$ on their own, by a simple comparison of cardinalities.

We have a first lemma:

Lemma 3.2 *The equality, inclusion and disjointness predicates do not have constant communication complexity.*

Proof. (Sketch) Suppose there is a communication protocol for the equality predicate with r exchanges of messages of arity k. Call any such sequence $a_1 b_1 \ldots a_r b_r$ of messages a *dialogue*. Since k is fixed, for large enough D there are less dialogues than sets of non-empty subsets of D. Hence, there are two different such sets X and Y such that the protocol yields the same dialogue when evaluating $P(X, X)$ and $P(Y, Y)$. But then this same dialogue will also be used for evaluating $P(X, Y)$; a contradiction.

It follows that the inclusion and disjointness predicates are not of constant communication complexity either. Indeed, communication protocols for these predicates can be easily transformed into a communication protocol for equality. It suffices to observe that $X = Y$ iff X is included in Y and vice versa, and that $X \subseteq Y$ iff X and Y^c are disjoint. ■

Our notion of communication protocols is a "set-based" variant of the original bit-based one, where the predicate to be evaluated is a predicate on bit-strings, and the exchanged messages are individual bits. Yao [16] showed in this setting that the equality predicate on strings of length n requires a number of bit exchanges that is linear in n. Lemma 3.2 can also be proven from this fact.

3.2 Split databases

We now fix the database schema to consist of one single unary relation S. A temporal database is then a sequence of finite sets of data elements. A temporal database is called *split* if there is exactly one state whose instance is empty. This state is called the *middle* state of the split database. If $\mathbf{I} = I_1, \ldots, I_n$ is a split database with middle state m then its *right part* $I_m, \ldots, I_n$ is denoted by $\mathbf{I}_{right}$ and its *left part* $I_1, \ldots, I_m$ by $\mathbf{I}_{left}$. Observe that one can test in TL whether a temporal database is split.

We next define an auxiliary language *split*-ETL whose semantics is only defined on split databases. Syntactically, split-ETL differs from ETL only in that each temporal operator L^+ (L^-) is split into a "left" and a "right" version L^+_{left} and L^+_{right} (L^-_{left} and L^-_{right}).

Informally, the left (right) version of a temporal operator behaves roughly the same as the operator itself, except that only the left (right) part of the split database is taken into consideration. Formally, let $\mathbf{I}$ be a split database of length n with middle state m. For each state j of $\mathbf{I}$ we define

$$\mathit{left}(j) := \begin{cases} j & \text{if } j \leq m \\ m & \text{if } j \geq m \end{cases} \qquad \text{and} \qquad \mathit{right}(j) := \begin{cases} 1 & \text{if } j \leq m \\ j - m + 1 & \text{if } j \geq m \end{cases}$$

So, $\mathit{left}(j)$ ($\mathit{right}(j)$) is the state in the left (right) part of $\mathbf{I}$ corresponding to j, if j is indeed contained in that part; if not, the default values m and 1, respectively, are used.

The semantics of the split temporal operators is then defined as follows. For $\star$ being either $-$ or $+$, $\mathbf{I}, j \models L^\star_{\mathit{left}}$ if $\mathbf{I}_{\mathit{left}}, \mathit{left}(j) \models L^\star$, and $\mathbf{I}, j \models L^\star_{\mathit{right}}$ if $\mathbf{I}_{\mathit{right}}, \mathit{right}(j) \models L^\star$.

We now have our second lemma.

Lemma 3.3 *On split databases, each* ETL *formula is equivalent to a split-*ETL *formula.*

Proof. (Sketch) Consider a temporal operator L^+ of ETL, with L a regular language over the alphabet $(v_1, \ldots, v_p)$. Then L is defined by some finite automaton M. Let the states of M be numbered $1, \ldots, q$, with 1 the initial state, and let F be the set of final states. For $z \in \{1, \ldots, q\}$ and $Z \subseteq \{1, \ldots, q\}$, let M_{zZ} be the automaton obtained from M by changing the initial state to z and the set of final states to Z, and denote by L_{zZ} the language defined by M_{zZ}. Let v_0 a symbol not in the alphabet $\{v_1, \ldots, v_p\}$. Then the ETL formula $L^+(\varphi_1, \ldots, \varphi_p)$ can be expressed in split-ETL as

$$((\mathit{at\_right} \vee \mathit{at\_middle})) \wedge L^+_{\mathit{right}}(\varphi_1, \ldots, \varphi_p) \vee$$
$$\Big(\mathit{at\_left} \wedge \bigvee_{z=1}^{q} ((L_{1\{z\}})^+_{\mathit{left}}(\varphi_1, \ldots, \varphi_p) \wedge (v_0 L_{zF})^+_{\mathit{right}}(\mathbf{true}, \varphi_1, \ldots, \varphi_p))\Big).$$

In the above, the language $v_0 L_{zF}$ is interpreted over the alphabet $(v_0, v_1, \ldots, v_p)$, and we have used the abbreviations

$$\begin{aligned} \mathit{at\_middle} &= \neg(\exists x)S(x); \\ \mathit{at\_left} &= K^+_{\mathit{left}}(\mathbf{true}, \mathit{at\_middle}); \\ \mathit{at\_right} &= K^-_{\mathit{right}}(\mathbf{true}, \mathit{at\_middle}), \end{aligned}$$

where K is the language a^+b over the alphabet (a, b).

The case L^- is treated similarly. ■

3.3 Inexpressibility

Let P be a binary predicate on sets of sets, as in Subsection 3.1. Consider the Boolean query Q_P on split databases defined as follows. For a split database $\mathbf{I} = I_1, \ldots, I_n$ with middle state m, $Q_P(\mathbf{I}) = \text{true}$ if $P(L, R)$ holds, where $L = \{I_j \mid 1 \leq j < m\}$ and $R = \{I_j \mid m < j \leq n\}$.

Our third lemma connects temporal queries to communication protocols:

Lemma 3.4 *If Q_P is expressible in* ETL*, then P has constant communication complexity.*

Proof. (Sketch) Assume Q_P is expressible in ETL. By Lemma 3.3, Q_P is expressible by a split-ETL formula θ. Consider all subformulas of θ of the form $L^\star_\delta(\ldots)$, where $\star$ is $+$ or

– and δ is *left* or *right*, and let $\pi_1, \ldots, \pi_r$ be a listing of these such that each subformula occurs after its own subformulas. Let k be the maximal number of free variables of any of these subformulas. We show that θ yields a communication protocol for P with r exchanges of messages of arity k.

Let X and Y be two sets of non-empty subsets of a finite set D of data elements, and consider any split temporal database $\mathbf{I}$ with middle state m, such that $X = \{I_j \mid 1 \leq j < m\}$ and $Y = \{I_j \mid m < j \leq n\}$. In order to evaluate $P(X, Y)$, it suffices to evaluate $Q_P(\mathbf{I})$, for which in turn it suffices to evaluate θ at some state of $\mathbf{I}$. To do the latter, the parties evaluate, in succession, each subformula π_i on every k-tuple of active domain elements, at the middle state. If the temporal operator of π_i is a left (right) version, then A (B) knows how to do this and he sends the resulting k-ary relation to B (A). (Note in this respect that both parties can be assumed, without loss of generality, to know the active domain of $\mathbf{I}$; if not, they can send the set of elements of D appearing in their set of sets to each other in a single exchange of messages.) When the values of all the π_i are known to both parties, they have enough information to evaluate θ. ■

Putting everything together, we obtain our main result:

Theorem 3.5 *Over schemas containing at least one relation of non-zero arity, there are queries expressible in* TS-FO *but not in* ETL. *In particular, query Q "are there two different states with the same instance?" is expressible in* TS-FO *but not in* ETL.

Proof. (Sketch) For simplicity we assume the schema consists of a single unary relation S. Query Q is obviously expressible in TS-FO:

$$(\exists t)(\exists t')(t \neq t' \wedge (\forall x)(\bar{S}(x,t) \leftrightarrow \bar{S}(x,t'))).$$

On the class of split databases whose left and right parts do not contain repetitions, Q corresponds to Q_P, where P is the non-disjointness predicate. By Lemma 3.2, the complement of P (so also P itself) does not have constant communication complexity. Hence, by Lemma 3.4, Q is not expressible in ETL. ■

Corollary 3.6 *Over schemas containing at least one relation of non-zero arity,* TL *is strictly less expressive than* TS-FO.

Note that our result remains valid under the assumption that the data elements are totally ordered. Indeed, the proof of Lemma 3.2 holds regardless of any knowledge (e.g., total order) the parties may have of the set D.

4 Infinite temporal databases

In this section, we extend our result to the case of infinite (but still discrete-time) temporal databases.

An *infinite temporal database* over a schema $\mathcal{S}$ is an infinite sequence $\mathbf{I} = I_1, I_2, \ldots$ of instances of $\mathcal{S}$. So, the set of states is the set of non-negative natural numbers, and the active domain may be infinite (although every individual instance is, by definition, still finite). In the present discussion, we focus on expressiveness, and not on the issue of finitely representing an infinite temporal database, or effectively computing answers to queries. References on these issues can be found in [3].

The query languages TS-FO and ETL can also be used on infinite temporal databases. For TS-FO, this is clear. For ETL, one uses ω-languages rather than ordinary languages in defining the semantics of the future temporal operators, since the future of every state is now infinite. The past of every state is, on the contrary, still finite. (All our results extend easily to the case of two-way infinite temporal databases.) Recall [8] that an ω-language is a set of infinite, rather than finite, words, and that a regular ω-language can still be defined by a finite automaton; an infinite word is accepted by the automaton if while reading the word it enters an accepting state infinitely often.

We now argue that our techniques of the previous section extend to the infinite case. An infinite temporal database is again called *split* if there is exactly one state whose instance is empty. The *right part* of an infinite split database is itself infinite; the *left part* is finite. Syntax and semantics of *split*-ETL on infinite split databases are defined in terms of ETL exactly as before. The result that split-ETL can simulate ETL on split databases goes through in the infinite case; the only modification to the proof of Lemma 3.3 is that in the large expression for L^+, L_{zF} now becomes an ω-language. Finally, the proof of Lemma 3.4 carries over verbatim, with the condition that instead of a finite $\mathbf{I} = I_1, \ldots, I_n$ we use an infinite $\mathbf{I} = I_1, I_2, \ldots$, and instead of $\{I_j \mid m < j \leq n\}$ we use $\{I_j \mid m < j\}$. Note that this implies that party B of the protocol deals with an infinite object, but this is of no concern since his computing power is unlimited.

We thus have:

Theorem 4.1 *On infinite temporal databases over a scheme containing at least one relation of non-zero arity, there are queries expressible in* TS-FO *but not in* ETL*. As a consequence,* TL *is strictly weaker than* TS-FO *on infinite temporal databases.*

5 Discussion

To conclude, we mention some cases in which TS-FO is not more powerful than TL. We also discuss alternative approaches that may provide different proofs of our result.

Local time. For clarity, we have separated the data elements in a temporal database from the natural numbers used to number its states. If, however, one allows these natural numbers to be stored in the database instances, an interesting special case can be indicated in which TL is expressively complete, i.e., equivalent to TS-FO.

More specifically, assume the database schema contains a unary relation *Time*, and assume the contents of that relation at the i-th state is the singleton $\{i\}$. Temporal databases of this kind are said to have *local time*. The local time assumption is quite realistic in practice, and has been made, e.g., by Gabby and McBrien [6]. It also seems to be implicitly made by Tuzhilin and Clifford [14]. We note:

Proposition 5.1 *On local-time databases,* TL *is equivalent to* TS-FO*.*

Proof. (Sketch) The proposition holds because on the class of local-time temporal databases, both the timestamp representation of the database as the linear order on the timestamps are definable in TL. Indeed, for a relation R, the timestamped relation $\bar{R}$ consists simply of all tuples (u, i), where u is in R at some state and i is the contents of *Time* at that state. This is readily expressible in TL. The linear order $t < t'$ on timestamps then corresponds to saying that t is the contents of *Time* at some state and t' is the contents of *Time* at a later state. Also this is readily expressed in TL. ■

The above proposition provides an a posteriori justification of the, at first sight erroneous, expressive completeness claims on TL made in [6, 14].

Actually, by essentially the same argument, a more general result can be proven. Let $\varphi(\bar{x})$ be an arbitrary fixed TL-formula. For each state j of a temporal database $\mathbf{I}$, φ defines a relation $\varphi(I_j)$ on the j-th instance I_j. If $\varphi(I_j)$ and $\varphi(I_\ell)$ are disjoint for any two different states j and ℓ, $\mathbf{I}$ is called *φ-disjoint.* Observing that $\varphi(I_j)$ can then be used as a simulation of the timestamp j, it can be shown that:

Proposition 5.2 *On the class of φ-disjoint databases,* TL *is equivalent to* TS-FO.

For example, local-time databases are φ-disjoint with φ simply being $Time(x)$. Another case in point are *insert-only* databases where for each j, the instance at state $j+1$ is obtained from the instance at state j by inserting a non-zero number of tuples in some of the relations. Insert-only databases are φ-disjoint with φ being $\bigvee(R(\bar{x}) \wedge \neg\, \textsf{previous}\, R(\bar{x}))$ (where the disjunction is over all relations R in the schema).

In temporal logics with iteration capabilities, even more powerful simulations of timestamps by tuples of data elements are possible [2].

Other possible approaches. An alternative approach to establish our Corollary 3.6 would be to prove that TS-FO^3, the 3 time-variable fragment of TS-FO, is strictly less expressive than full TS-FO. Indeed, it is known and not difficult to verify that every TL query is already expressible by a formula in TS-FO using at most 3 distinct time-variables. Note that our proof of Theorem 3.5 implies that TL is strictly contained in TS-FO^3; actually, the proof shows that even some TS-FO^2 queries are not expressible in TL.

More generally, one might conjecture that there is a strict hierarchy in expressive power among the fragments TS-FO^k for each k. (It is known that $\text{TS-FO}^1 \subset_{\neq} \text{TS-FO}^2 \subset_{\neq} \text{TS-FO}^3$.) A closely related question is whether there is a strict FO^k-hierarchy on the class of ordered finite graphs. Here, FO^k denotes the k variable fragment of standard first-order logic on ordered graphs.

On might also try to separate TL and TS-FO with a proof based on Ehrenfeucht-Fraïssé style games. Segoufin [12] designed a very elegant extension of Ehrenfeucht-Fraïssé games capturing precisely the expressive power of TL. In our experience, however, it is quite hard to explicitly construct families of pairs of temporal databases that are indistinguishable in TL. Our approach based on communication complexity turned out to be more successful. Our proof is robust under built-in relations on data elements, such as total order, and at the same time separates the more powerful ETL from TS-FO.

Acknowledgments

We thank Luc Segoufin for pointing out the relation to communication complexity, and Victor Vianu for raising the TS-FO^k issue. We thank them both for helpful discussions and encouragements.

References

[1] S. Abiteboul, R. Hull and V. Vianu, Foundations of Databases, Addison-Wesley, Reading-Massachusetts. 1994.

[2] S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal connectives versus explicit timestamps in temporal query languages. In J. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, Workshops in Computing, pages 43–57. Springer-Verlag, 1995.

[3] J. Chomicki. Temporal query languages: a survey. In D.M. Gabbay and H.J. Ohlbach, editors, *Temporal Logic: ICTL'94*, volume 827 of *Lecture Notes in Computer Science*, pages 506–534. Springer-Verlag, 1994.

[4] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier, 1990.

[5] D. Gabbay. The declarative past and the imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, B. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.

[6] D. Gabbay and P. McBrien. Temporal logic and historical databases. In *Proceedings 17th International Conference on Very Large Databases*, pages 423–430, 1991.

[7] D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic, Mathematical Foundations and Computational Aspects*, volume 1 of *Oxford Logic Guides*.

[8] D. Perrin. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier, 1990. Oxford University Press, 1994.

[9] H. Kamp. Formal properties of 'now'. *Theoria*, 37:227–273, 1971.

[10] J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[11] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[12] L. Segoufin. Temporal logic and games. INRIA, VERSO, 1995.

[13] D. Toman and D. Niwinski. First-order queries over temporal databases inexpressible in temporal logic. Dept. of Comp. and Info. Science, Kansas State University, 1995. To appear in Proceedings 5th International Conference on Extending Database Technology, Avignon, France, March 1996.

[14] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 13–23. Morgan Kaufmann, 1990.

[15] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–93, 1983.

[16] A. C.-C. Yao. Some complexity questions related to distributive computing. In *Proceedings 11th ACM Symposium on the Theory of Computing*, pages 294–300, 1979.