On the Feasibility of Checking Temporal Integrity Constraints *

Jan Chomicki† Damian Niwiński‡
Kansas State University Warsaw University

May 22, 1995

Keywords: integrity constraints, temporal logic, decidability, computational complexity, temporal databases.

1989 CR Classification: F.2.2, F.4.1, H.2.3.

Abstract

We analyze the computational feasibility of checking temporal integrity constraints formulated in some sublanguages of first-order temporal logic. Our results illustrate the impact of the quantifier pattern on the complexity of this problem. The presence of a single quantifier in the scope of a temporal operator makes the problem undecidable. On the other hand, if no quantifiers are in the scope of a temporal operator and all the quantifiers are universal, temporal integrity checking can be done in exponential time.

1 Introduction

As temporal databases become more widely used in practice [27, 28], the need arises to address database integrity issues that are specific to such databases. In particular, it is necessary to generalize the standard notion of *static integrity* (involving single database states) to *temporal integrity* (involving sequences of database states).

This work is the first attempt to date to analyze the *computational feasibility* of checking temporal integrity constraints. We consider various sublanguages of first-order temporal logic (*FOTL*). The starting point is the class of *biquantified formulas* proposed by Lipeck, Saake, and their students [14, 17, 19, 20]. Biquantified formulas allow only future tense

*To appear in *Journal of Computer and System Sciences.* A preliminary version of this paper was presented at the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 1993, Washington, D.C. Supported by NSF grant IRI-9110581, KSU Faculty Development Award, and Polish KBN grants 2 1192 9101 and 2 P301 009 06.

†Address: Computing and Information Sciences, 234 Nichols Hall, Kansas State University, Manhattan, KS 66506-2302. E-mail: chomicki@cis.ksu.edu.

‡Work performed mainly while visiting Kansas State University. Address: Institute of Informatics, Warsaw University, Banacha 2, 00-950 Warsaw, Poland. E-mail: niwinski@mimuw.edu.pl.

temporal operators and restricted quantification in the following sense: the quantifiers can be either *external* (not in the scope of any temporal operators) or *internal* (no temporal operator in their scope). Moreover, all external quantifiers are universal. Biquantified formulas arise as the result of *composing* propositional temporal logic with first-order predicate logic [9], i.e., taking a version of propositional temporal logic in which propositional atoms are replaced by first-order predicate formulas.

The main complexity results are as follows:

- for biquantified formulas with a single internal quantifier (existential or universal), temporal integrity checking is undecidable (Π_2^0-hard),
- for biquantified formulas with no internal quantifiers (called *universal*), temporal integrity checking is decidable (in exponential time).

Although the time in our framework is infinite, integrity constraints impose restrictions on finite sequences of database states. Consequently, temporal integrity violations should be detectable using finite databases. This intuition is formalized by the requirement that integrity constraints define *safety properties* [1, 21]. We are going to adopt this point of view, thus we consider only formulas that define safety properties.

As a side result we obtain a biquantified formula (defining a *safety property*) which is not equivalent to any formula $\Box\,\Phi$ where Φ is a past formula. It is well known [21] that every *propositional* temporal logic formula defining a safety property is equivalent to a formula in the latter form. Thus, our result highlights the difference between propositional and first-order temporal logic.

Our results are applicable both to *temporal integrity checking* and to *temporal triggers* because we show that trigger firing is a notion dual to constraint satisfaction.

The plan of the paper is as follows. In section 2, we introduce the basic concepts of *FOTL*, discuss our framework of temporal integrity checking, and define the class of biquantified formulas and its subclasses. In section 3, we show that for biquantified formulas with a single internal quantifier temporal integrity checking is undecidable. In section 4, we show that for universal biquantified formulas (no internal quantifiers), temporal integrity checking is decidable in exponential time. In section 5, we briefly discuss related work. In section 6, we conclude and outline directions for future research.

2 Basic Notions

In this section, we introduce the basic concepts of *FOTL*, discuss our framework of temporal integrity checking, and define the class of biquantified formulas and its subclasses. A comprehensive recent reference for temporal logic is [10].

Syntax

The expressions of *FOTL* are built from the following sets of symbols: a finite set of predicate symbols, a finite set of constant symbols, a countably infinite set of variables, the equality connective $=$, the logical connectives: $\vee, \wedge, \neg, \Rightarrow$, quantifiers: $\exists, \forall$, future tense temporal connectives: $\bigcirc$ and **until** , and past tense temporal connectives: $\bullet$ and **since** . Each predicate symbol is given with an arity $r \geq 1$. We refer to the sets of predicate and constant symbols as the *vocabulary* of the language. This part is not fixed; we will consider languages over different vocabularies. A *term* is a constant or a variable. An *atomic formula* is of the form $t_1 = t_2$ or $p(t_1, \ldots, t_r)$, where the t_i's are terms and p is a predicate symbol of arity r. Other formulas are defined by the following rules: if A, B are formulas and x is a variable, the following are also formulas: $A \vee B$, $A \wedge B$, $\neg A$, $A \Rightarrow B$, $\exists x A$, $\forall x A$, $\bigcirc A$ (read as "next time A"), A **until** B, $\bullet A$ ("previous time A"), and A **since** B. The formulas that do not use temporal connectives at all are referred to as *pure first-order formulas.* The formulas that use past tense temporal connectives but not future tense ones are called *past temporal formulas.* The formulas that use future tense connectives but not past tense ones are called *future temporal formulas.*

Semantics

A *first-order structure* M over a given vocabulary consists of a *universe* $|M|$, an *interpretation* $c^M \in |M|$ for each constant symbol c and an interpretation $p^M \subseteq |M|^r$ for each predicate symbol p of arity r. We say that a *predicate p of M is true about* a tuple $(m_1, \ldots, m_r)$ of elements of $|M|$ if $(m_1, \ldots, m_r) \in p^M$.

In order to interpret temporal formulas, we assume that time is isomorphic to the natural numbers, i.e., *time instants* form an infinite sequence $t = 0, 1, 2, \ldots$ An *infinite-time temporal database* (usually called shorter: *temporal database* or just *database*) is a sequence

$$D = (D_0, D_1, D_2, \ldots)$$

of first-order structures over the same vocabulary and with the same universe, which we will call the *universe of the database* and denote $|D|$ (so $|D| = |D_0| = |D_1| = \ldots$). We refer to the structure D_t as the *database state* at time instant t. We assume that a constant symbol c has the same interpretation in each database state (i.e. $c^{D_t} = c^{D_0}$, for each t) which we shall denote simply by c^D. The interpretation of a predicate symbol may vary from one time instant to another.

We make the following assumptions about database states: the universe is *infinite* and *countable*, and the interpretation of any predicate symbol is a *finite* relation. These are rather standard assumptions in database theory. Notice that the equality is not considered as one of database predicates (actually it is an infinite relation). For notational convenience only, we shall also assume that, unless otherwise stated, the universe of each database state equals to the set of natural numbers N.

Let $D = (D_0, D_1, D_2, \ldots)$ be a temporal database. A *valuation* is a mapping v that associates a value $v(x) \in N$ with each variable x. Notice that the value of a variable does not change with time (in other words, all variables we consider are *global* or *rigid*). It is convenient to extend the valuation to terms by setting $v(c) = c^D$, for a constant symbol c. Now we define the truth value of a formula φ at a time instant t under a valuation v

by means of the *satisfaction* relation, "φ is satisfied by D at the time instant t under the valuation v", in symbols $D, v, t \models \varphi$, defined inductively as follows:

- $D, v, t \models t_1 = t_2$, whenever $v(t_1) = v(t_2)$,
- $D, v, t \models p(t_1, \ldots, t_r)$, whenever the predicate p of D_t is true about $(v(t_1), \ldots, v(t_r))$,
- $D, v, t \models \neg A$, iff not $D, v, t \models A$,
- $D, v, t \models A \vee B$, iff $D, v, t \models A$ or $D, v, t \models B$, similarly for other logical connectives,
- $D, v, t \models \exists x A$ iff there exists $i \in N$ such that $D, v[x \leftarrow i], t \models A$, where $v[x \leftarrow i]$ is a valuation that maps x to i and otherwise coincides with v,
- $D, v, t \models \forall x A$ iff for all $i \in N$, $D, v[x \leftarrow i], t \models A$,
- $D, v, t \models \bigcirc A$ iff $D, v, t+1 \models A$,
- $D, v, t \models A$ **until** B iff for some $s \geq t$, $D, v, s \models B$ and for all u, $t \leq u < s$, $D, v, u \models A$,
- $D, v, t \models \bullet A$ iff $t > 0$ and $D, v, t-1 \models A$,
- $D, v, t \models A$ **since** B iff for some s, $0 \leq s \leq t$, $D, v, s \models B$ and for all u, $s < u \leq t$, $D, v, u \models A$.

A formula with no occurrences of free variables is said to be *closed* or a *sentence*. Clearly, the truth value of a closed formula does not depend on the valuation, so if A is a sentence, we simply write $D, t \models A$. We also write $D, v \models A$ or $D \models A$ to mean $D, v, 0 \models A$ or $D, 0 \models A$, respectively. In this last case, we say that D is a *model* of A.

Other well known temporal connectives $\Diamond$ (*"sometime in the future"*), $\Box$ (*"always in the future"*), $\blacklozenge$ (*"sometime in the past"*), and $\blacksquare$ (*"always in the past"*) can be defined using **until** or **since**:

$$\Diamond A \equiv \mathit{True} \textbf{ until } A$$

$$\Box A \equiv \neg \Diamond \neg A$$

$$\blacklozenge A \equiv \mathit{True} \textbf{ since } A$$

$$\blacksquare A \equiv \neg \blacklozenge \neg A.$$

For a past formula A, the truth of A at instant t is determined only by the database states $D_0, \ldots, D_t$. Therefore, we shall also consider *finite-time temporal databases*, of the form $D = (D_0, \ldots, D_t)$, with the satisfaction of past formulas defined and denoted similarly as in the infinite case.

If $D = (D_0, D_1, D_2, \ldots)$ is an infinite-time database, then any finite-time database $(D_0, \ldots, D_t)$ is called a *prefix of* D. In this case D is an *extension* of $(D_0, \ldots, D_t)$.

Examples

Consider a temporal database $(D_0, \ldots, D_{t_0})$ whose every state D_t, $0 \leq t \leq t_0$, contains information about customer orders submitted or filled at instant t. For example, $Sub(a)$ is satisfied at instant t if a is an order submitted at t.

The constraint *"an order can be submitted only once"* (implementing a kind of object identity) can be specified as:

$$\forall x \ \Box \, (Sub(x) \Rightarrow \ \bigcirc \ \Box \, \neg Sub(x)).$$

The constraint *"orders should be filled in the order that they are submitted"* (in fact enforcing a queue-like discipline) can be written as:

$$\forall x \ \forall y \qquad \Box \, \neg(x \neq y \wedge Sub(x) \wedge \ \bigcirc \, ((\neg Fill(x)) \ \mathbf{until} \ (Sub(y) \wedge ((\neg Fill(x)) \ \mathbf{until} \ (Fill(y) \wedge \neg Fill(x)))))).$$

Intuitively, the above formula says that there cannot be two different orders x and y such that x is submitted before y and x is not filled before y is filled.

Extended vocabulary

It is sometimes convenient to extend the above language by symbols that refer explicitly to the ordering of natural numbers: the binary relation symbols $\leq$ and $succ$, and the monadic relation symbol $Zero$. These symbols are assumed to be interpreted in the same way in any database state D, namely, $\leq^D$ is the standard ordering relation on N, $succ^D$ is the standard successor relation (i.e., $succ^D(a,b)$ is true iff $b = a+1$) and $Zero(a)$ is true iff $a = 0$. Notice that, like the equality, $\leq$ and $succ$ differ from other predicate symbols in that they define infinite relations.

Classification of formulas

Let us recall in this context the standard classification of first-order formulas in prefix normal form [2]. It is well known that every first-order formula can be transformed to this form. We start with the class of Σ_0 formulas (or equivalently, Π_0 formulas) consisting of first-order formulas that do not contain quantifiers or temporal connectives. Note that Σ_0 formulas are just boolean combinations of atomic formulas of the form $t_1 = t_2$ or $p(t_1, \ldots, t_r)$ where the t_i's are terms and p is a predicate symbol of arity r. The classes of Σ_n and Π_n formulas are defined inductively as follows. If A is a Σ_n formula then any formula of the form $\forall x_1 \ldots \forall x_k A$ is Π_{n+1} formula. If A is a Π_n formula then any formula of the form $\exists x_1 \ldots \exists x_k A$ is Σ_{n+1} formula. Denote by Σ_* the union of all Σ_n for $n \geq 0$.

For a class $\mathcal{C}$ of formulas, let $tense(\mathcal{C})$ be the class of temporal formulas obtained from $\mathcal{C}$ using future temporal and propositional connectives but no quantifiers (i.e., using the rules of the definition of formulas). If $Q \in \{\exists, \forall\}$, let $Q\mathcal{C}$ be the class of all formulas of the form QxA, where $A \in \mathcal{C}$ and let $Q^*\mathcal{C}$ be the union of all classes $QQ \ldots Q\mathcal{C}$. If we take a class of formulas of the form $Q^* tense(\mathcal{C})$, the quantifiers in Q^* are called *external* and the quantifiers in $\mathcal{C}$ *internal*.

We will be mainly concerned with the following classes of formulas:

- *biquantified* formulas, in symbols $\forall^* tense(\Sigma_*)$,
- *universal* formulas, in symbols $\forall^* tense(\Sigma_0)$,
- biquantified formulas with a *single internal quantifier*, in symbols $\forall^* tense(\Sigma_1)$.

Note that the Π_1 formulas are equivalent to the negations of the Σ_1 formulas. Then, since $tense(\mathcal{C})$ is closed under negation, the $\forall^* tense(\Sigma_1)$ formulas are equivalent to $\forall^* tense(\Pi_1)$ formulas.

Both the examples given earlier in this section are universal (i.e., $\forall^* tense(\Sigma_0)$) formulas.

Temporal integrity checking and triggers

Temporal integrity constraints are imposed on the *current history* of a database, i.e., the sequence of states up to the current one. Such a history is of course a finite-time temporal database. However, the semantics of a temporal integrity constraint is defined with respect to an infinite structure representing a possible future evolution of the database. The notion of *potential constraint satisfaction* (called *potential validity* in [20]) reconciles those two views.

For a closed *FOTL* formula C, the set $Pref(C)$ is defined as follows:

$$\eta \in Pref(C) \text{ if there is a model } D = (D_0, D_1, \ldots) \text{ of } C \text{ such that } \eta = (D_0, \ldots, D_t).$$

A constraint C is *potentially satisfied* at instant t if the current history $(D_0, \ldots, D_t) \in Pref(C)$. In other words, a constraint is potentially satisfied after an update if the history ending in the state resulting from the update has an (infinite) extension to a model of the constraint.

In the database context, considering infinite sequences of states means that the database is infinitely updatable. This seems like a desirable characteristic, even though in most applications databases have only a finite lifetime. Similarly, in concurrent programming one often studies infinite behaviors, although every practical program runs only for a finite time. In both cases infinity provides a convenient mathematical abstraction. Moreover, the paper [20] that originated the study of temporal integrity constraints considered constraints over infinite sequences. As we study classes of constraints proposed in that very paper, we want to stay as close to the original motivation as possible. In the future, it may prove wortwhile to study integrity checking over finite histories, as well as different notions of constraint satisfaction.

A Condition-Action trigger [6] "**if** C **then** A" *fires* at instant t for a (ground) substitution θ to the free variables of C if $\neg C\theta$ (the result of applying the substitution θ to $\neg C$) is not potentially satisfied at t. The action executed is $A\theta$. Intuitively, a trigger fires after an update if no extension of the history ending in the state resulting from the update can make the trigger condition false. So we can see that the notion of trigger firing is dual to potential constraint satisfaction. That corresponds to the intuition that integrity checking triggers should fire when the integrity is violated.

Safety properties

Any set of temporal databases over a fixed vocabulary can be considered a *property*. We say that a property $\mathcal{P}$ is *defined* by a (closed) formula A if it consists of all models of that sentence, i.e. $\mathcal{P} = \{D : D \models A\}$.

A property $\mathcal{P}$ is called a *safety* property [1] if it satisfies the following condition:

> Let $D = (D_0, D_1, D_2, \ldots)$ be a temporal database. If any prefix $(D_0, \ldots, D_t)$ of D can be extended to some D' in $\mathcal{P}$, then D itself belongs to $\mathcal{P}$.

A formula defining a safety property is called a *safety formula.*

We note the following:

Proposition 2.1 *Any property defined by a formula of the form* $\Box\, A$*, where* A *is a past formula, is a safety property.*

A property $\mathcal{P}$ is a *liveness* property [1] if it satisfies the following condition:

> Any finite-time temporal database $D = (D_0, D_1, D_2, \ldots, D_t)$ can be extended to an element of $\mathcal{P}$.

A formula defining a liveness property is called a *liveness formula.*

It should be clear that a liveness formula is useless for integrity checking because it does not produce any integrity violations (it is always potentially satisfied). For an example, consider the formula $\Diamond\, \exists x\ p(x)$. [1] shows that every property is an intersection of a safety property and a liveness property. Only the safety property (defined by a temporal formula) is useful for temporal integrity. Intuitively, this can be explained as follows: Integrity checking is always done using a finite history. Therefore, if an infinite sequence of states violates a constraint, the integrity checking mechanism should be able to discover that after looking at some finite prefix of this sequence. The behavior "at infinity" is not known at the time of checking the constraint because future updates may be arbitrary. This should be contrasted with the situation in the area of concurrent programming [21] where formulas that are not safety formulas are very useful. Such formulas usually describe properties of a given concurrent program whose behavior "at infinity" is known. Following this line of thought, liveness formulas may be useful as integrity constraints if the database changes in a controlled manner, for example when all the updates are performed by predefined procedures. Then a priori integrity verification may become possible [7, 18].

The restriction to safety formulas is essential for the technical results presented in this paper, particularly for the positive results in section 4 which do not hold for non-safety formulas.

Propositional temporal logic

We shall also consider a propositional version of the logic presented above. The vocabulary of the propositional temporal logic of linear time (propositional TL) consists of a set of propositional letters that are used as atomic formulas of the logic. Other formulas are formed from atomic formulas using the Boolean operators and temporal connectives. The semantics is given in terms of infinite sequences of propositional *states*, where a state is a mapping from the set of propositional letters to the set $\{true, false\}$. The interpretation of the temporal connectives is similar as in the first order case presented above [8].

Propositional temporal logic has attracted much attention as a formalism for reasoning about concurrent programs, since the basic properties of this logic (satisfiability, validity, equivalence of formulas) are elementarily decidable (see [8] for a survey). We shall see in section 4 that in some special cases decision problems for $FOTL$ can be reduced to the propositional case.

3 Internal quantification

In this section, we study the complexity of the extension problem, i.e., of $Pref(\alpha)$, for a biquantified formula α with a single internal quantifier (a $\forall^* tense(\Sigma_1)$ formula). It is easy to see that it does not matter whether this quantifier is universal or existential because a universal quantifier can be replaced by an existential one and vice versa. For example, $\Box \, \forall x \, \gamma \equiv \neg \Diamond \, \exists x \, \neg\gamma$. We assume that the internal quantifier is existential. We show that there is a $\forall^3 tense(\Sigma_1)$ formula $\tilde{\varphi}$ for which the extension problem is Π^2_0-complete. We also show that $\tilde{\varphi}$ is a formula that defines a safety property but cannot be transformed to the form $\Box \, A$, where A is a past formula.

The vocabulary of $\tilde{\varphi}$ consists of monadic predicate symbols only. As the first step we construct another formula φ that is universal but over the extended vocabulary (c.f., section 2). The argument uses an encoding of the computations of a deterministic Turing machine for which some temporal property is shown Π^0_2-complete.

Repeating computations

We now describe a problem about Turing machines and prove its Π^0_2-completeness.

We consider deterministic Turing machines with a single tape infinite to the right over an alphabet Σ including $B \in \Sigma$ as a blank symbol and $\{0, 1\} \subset \Sigma$ as an input alphabet.

We say that a word $w \in \{0, 1\}^*$ *induces a repeating behaviour* of a machine M if the computation of M starting with w as an input is infinite and also the machine's head visits the leftmost cell of the tape infinitely often.

Lemma 3.1 *There exists a Turing machine M such that the set of words w which induce a repeating behaviour of M is Π^0_2-complete.*

Proof: Let us fix a Π^0_2-complete language $A \subseteq \{0,1\}^*$. By a well–known characterization of the Π^0_2 sets (c.f. Rogers [23], Chapter 14), there exists a recursive relation $R(x,y,z)$, such that

$$x \in A \iff \forall y \exists z\, R(x,y,z)$$

Clearly, we can have a Turing machine, say M_0, over an alphabet $\{0,1,\$\}$, which halts for every input and accepts a word $w\$v\$u$ iff w, v, and u are words in $\{0,1\}^*$ satisfying the relation R.

We shall construct a machine M with the input alphabet $\{0,1\}$ such that a word $w \in \{0,1\}^*$ will induce a repeating behaviour of M iff $w \in A$.

Given an input w, the machine first shifts it one cell to the right and marks the *second* cell of the tape in some special way; this is in order to prevent visiting the origin of the tape except in the situations to be described below.

Now M enters into an infinite loop in which it will examine all the words $v \in \{0,1\}^*$ in some fixed linear order. For each such v, the machine M starts another loop in which it enumerates all the pairs (u,m), where $u \in \{0,1\}^*$ and m is (an encoding) of a natural number. For each such (u,m), M simulates m steps of the computation of the machine M_0 on the input $w\$v\$u$ (remember that w is the original input of the machine M). If it happens that the simulated machine M_0 stops and accepts within m steps, then the machine M exits from the loop enumerating the pairs (u,m) for this given v and goes to the origin of its own tape. Next, it passes to an examination of the succesor of v. Note that if, for some v, no attempt of simulating M_0 is successful, then M will loop forever for that v and will not visit the origin of the tape any more, neither will it examine any succesor of v. This may happen, however, only if there is no u such that $R(w,v,u)$ holds.

Then, it is easy to see that an input w induces a repeating behaviour of M iff, for all v, there is some u such that the relation $R(w,v,u)$ holds, that is, iff $w \in A$. $\Box$

Remark: For some nondeterministic Turing machines, the problem of deciding whether a word induces a repeating behaviour is not arithmetical. This fact has been used in the proof that the satisfiability problem for first–order temporal logic is Σ^1_1-complete [15].

Formula φ

We are now going to show that the repeating behaviour of a Turing machine can be expressed in the framework of first-order temporal logic over a vocabulary extended by the symbols $\leq$, *succ* and *Zero*.

Let us fix a Turing machine M satisfying the property of the preceding lemma. Recall that the alphabet Σ of the tape symbols used by M includes the input alphabet $\{0,1\}$ and the special blank symbol B. Let Q be the set of states of M including the initial state q_0. We encode configurations of M (i.e., sequences of the form $\alpha q \beta B^\omega$ where $\alpha, \beta \in \Sigma^*$ and $q \in Q$) by database states in the following way: the vocabulary will include a monadic symbol P_q for each state $q \in Q$ and a monadic predicate symbol P_σ for each $\sigma \in \Sigma$ except for B. The intuitive meaning of a formula $P_\sigma(x)$ (resp. $P_q(x)$) is: the xth symbol of the actual configuration is σ (resp. q). It will be further convenient to use the notation $P_B(x)$ as an abbreviation for $\bigwedge_{z\in Q\cup\Sigma-\{B\}}\neg P_z(x)$. The intuitive meaning of this is: the xth symbol of the actual configuration is the blank symbol.

The formula which forces a temporal database to encode a repeating computation of M will express the following database properties. First, for any element x of the universe, in any database state (i.e., *always* in the temporal sense), at most one of the monadic predicates P_z, where $z \in Q \cup \Sigma - \{B\}$, is true about x. Second, the initial database state encodes some initial configuration of M. Third, subsequent database states encode subsequent configurations. Finally, there are infinitely many database states which encode configurations with the machine's head scanning the leftmost tape cell.

We sketch the construction here. It is given in detail in the Appendix. Each of the above conditions can be expressed by a temporal formula of the form $\forall x_1 \ldots \forall x_k \psi$, where ψ is quantifier-free. For example, the last statement (concerning the repeating behaviour of the machine) can be formalized as

$$\forall x\ Zero(x) \Rightarrow \Box\ \Diamond \bigvee_{q\in Q} P_q(x).$$

Indeed, this formula forces that, for the least element of the database universe, infinitely often some predicate P_q is true. This corresponds to the fact that, in the encoded computation, the first symbol of a configuration is infinitely often a state, i.e. the machine's head visits the origin of the tape infinitely often. Moreover, for all these formulas $k \leq 3$. Then, using the standard transformations of predicate calculus, we can write the entire formula in the form $\forall x_1 \forall x_2 \forall x_3 \psi$, where ψ is quantifier-free.

We can summarize the above considerations in the following.

Proposition 3.1 *There is a temporal formula φ over the vocabulary described above, and of the form $\varphi \equiv \forall x_1 \forall x_2 \forall x_3 \psi$, where ψ is quantifier-free, such that any temporal database over this vocabulary satisfies φ if and only if it is an encoding of a repeating computation of the machine M.*

Proposition 3.2 *The property defined by a formula of Proposition 3.1 is a safety property.*

Proof: Since the machine M is deterministic, any finite sequence of database states of length ≥ 1 has at most one extension to a temporal database. Thus the safety condition holds obviously. $\Box$

But the main feature of the above-defined formula φ is the following.

Theorem 3.1 *The problem of deciding whether a given finite-time temporal database over the vocabulary considered above can be extended to a (infinite-time) temporal model of φ is Π^0_2-complete.*

Proof: We show a Turing reduction of the problem discussed in Lemma 3.1. Let $w \in \{0,1\}^*$. Consider a database state which encodes the initial configuration of the machine M with the input w. Now the one element sequence consisting of this database state can be extended to a temporal model of φ iff w induces a repeating behaviour of M. Thus the problem is Π^0_2-hard.

In order to see that the problem is actually in the class Π^0_2, we can give it an alternative formulation as follows. Given a finite sequence of database states, determine if for each n there is a finite prolongation of this sequence which encodes an initial segment of some computation of M in which the machine's head visits the leftmost cell of the tape at least n times. Since finite sequences of database states can be encoded by integers and it is clearly decidable if a finite sequence of database states encodes some initial segment of a computation of M then the latter problem is clearly in Π^0_2. In order to see that it is equivalent to the original one, observe that a finite sequence of database states has at most one prolongation of a given length to a sequence which actually encodes an initial segment of some computation of M. $\Box$

Corollary 3.1 *The formula φ is not equivalent to any formula of the form $\Box\,\theta$, with θ being a past formula.*

Proof: Suppose to the contrary that φ is equivalent to $\Box\,\theta$. The set of finite sequences of database states that satisfy θ cannot coincide with the set of prefixes of temporal models of φ since the former set is clearly decidable while the latter was shown to be Π^0_2-complete. This, however, is not yet a contradiction, since the mere satisfaction of θ by a sequence of database states does not imply that the sequence can be extended to a temporal model of $\Box\,\theta$. We have to proceed more subtly.

Let $w \in \{0,1\}^*$ and let C_0 be the initial configuration of the machine M with the input w. We claim that the following condition is equivalent to the statement that w induces a repeating behavior of M:

> For each n, the computation of M on the input w does not terminate within n steps and, *for each* finite-time database $D = (D_0, D_1, \ldots, D_n)$ such that D_i is the encoding of the ith configuration of the computation of M on w (the quantifier "for each" stands here for formal reasons, in fact there may be at most one such database), D satisfies the past formula θ.

One implication is plain and the other follows from our hypothesis and the fact that the initial configuration has at most one prolongation to a computation of M of length n.

Now, using appropriate encodings of finite sequences of database states by natural numbers, one can easily see that the last problem is in the class Π^0_1 which contradicts Lemma 3.1. $\Box$

Formula $\tilde{\varphi}$

The formula $\tilde{\varphi}$ will force that a database encodes a repeating computation if considered with an ordering of type ω on the universe which is not explicitly present but can be defined by means of temporal formulas. Definability of such an ordering in temporal logic is a standard construction used in the proofs of the incompleteness of *FOTL* [11].

Thus, we transform the above formula φ into a $\forall^3 tense(\Sigma_1)$ formula $\tilde{\varphi}$ written over a vocabulary consisting of monadic symbols only.

Let W be a new monadic symbol. Consider the following temporal formulas:

W1 $\forall x \forall y \Box ((W(x) \wedge W(y)) \Rightarrow x = y)$

W2 $\Box \exists x W(x)$

W3 $\forall x \Box (W(x) \Rightarrow \bigcirc \Box \neg W(x))$.

Notice that **W1** and **W3** are universal formulas, while **W2** is a $tense(\Sigma_1)$ formula. The formula **W1** forces that for each database state there is at most one element of the universe satisfying the predicate W; **W1** and **W2** induce that, for each database state, there is *exactly one* such element. **W3** forces that for each element of the universe, the predicate W is true in *at most one* database state. Whenever **W1** and **W3** are true in a temporal database D, they induce a total ordering of type ω on the set W^D of those elements of the universe that satisfy the predicate W in some database state, defined as follows: $i \preceq_W j$ iff $W(i)$ happens not later than $W(j)$. This ordering can be expressed by a temporal quantifier-free formula:

$$x \preceq_W y \equiv_{df} \Diamond (W(x) \wedge \Diamond W(y)).$$

Moreover, if D satisfies **W2** then the set W^D is infinite, and the successor relation corresponding to $\preceq_W$ can be defined by the formula

$$S_W(x, y) \equiv_{df} \Diamond (W(x) \wedge \bigcirc W(y))$$

while the initial ("zero") element is clearly distinguished by

$$Z_W(x) \equiv_{df} W(x).$$

We shall consider the vocabulary of monadic symbols of the previous subsection (induced by the Turing machine M) extended by the symbol W. Now, if we restrict our attention to models of $\mathbf{W1} \wedge \mathbf{W2} \wedge \mathbf{W3}$, we can define the encoding of configurations of M by database states in the similar way as before, but now with respect to the new ordering $\preceq_W$. Also, the property "a database encodes a repeating computation of M w.r.t. $\preceq_W$" can be defined in a similar way.

Let the formula $\varphi \equiv \forall x_1 \forall x_2 \forall x_3 \psi$ be as in the previous subsection. Let φ_W be

$$(\forall x_1 \forall x_2 \forall x_3) \Diamond W(x_1) \wedge \Diamond W(x_2) \wedge \Diamond W(x_3) \Rightarrow \psi_W$$

where ψ_W is a formula obtained from ψ by replacing each subformula $x \leq y$ by $x \preceq_W y$, $succ(x, y)$ by $S_W(x, y)$, and $Zero(x)$ by $Z_W(x)$. Consider the formula $\varphi_W \wedge \mathbf{W1} \wedge \mathbf{W2} \wedge \mathbf{W3}$. This formula can readily be transformed to the form $\tilde{\varphi} \equiv \forall x_1 \forall x_2 \forall x_3 \tilde{\psi}$, where $\tilde{\psi}$ is $tense(\Sigma_1)$. Notice that $\tilde{\varphi}$ uses only monadic predicate symbols of the database vocabulary.

Consequently, we have new versions of Propositions 3.1 and 3.2, Theorem 3.1 and Corollary 3.1.

Proposition 3.3 *A database D satisfies $\tilde{\varphi}$ iff D encodes a repeating computation of the machine M, w.r.t. the ordering $\preceq_W$.*

Proposition 3.4 *The property defined by the formula $\tilde{\varphi}$ is a safety property.*

Proof: Let $D = (D_0, D_1, D_2, \ldots)$ be a temporal database such that any prefix $(D_0, \ldots, D_t)$ of D can be extended to a model of $\tilde{\varphi}$. It is easy to see, that D must satisfy $\mathbf{W1} \wedge \mathbf{W2} \wedge \mathbf{W3}$. We have to show that $D \models \varphi_W$. By Proposition 3.3, any prefix of D can be extended to a temporal database encoding some repeating computation of M. In all these encodings, the database state D_0 encodes an initial configuration of the corresponding computation. Notice that this configuration is not entirely determined by D_0 itself and not even by the whole D, but it depends also on the interpretation of the predicate letter W in the actual extension. We show, however, that if we consider sufficiently large prefixes then the initial configurations in question are equal. Indeed, consider the first time instant t such that, for all a, if some $P_z(a)$, $z \in Q \cup \Sigma$, holds in D_0 then $W(a)$ holds in some database state D_i with $i < t$ or it holds in no database state of D at all. Let b be the element such that $W(b)$ holds in D_t (there is such an element since D satisfies $\mathbf{W2}$). Observe that, for any extension of the prefix $(D_0, \ldots, D_t)$ that encodes a computation of the machine M, if we consider the initial configuration of this computation then b "encodes" the first position of the machine's tape holding the symbol blank. (Intuitively, the information about the initial configuration encoded by D_0 has been completed by t.) Then it is not difficult to see that all the computations encoded by the extensions of the prefix $(D_0, \ldots, D_t)$ start with the same initial configuration and consequently are equal. Using a similar argument for subsequent configurations, one can further see that D itself encodes this unique computation in question. Thus, by Proposition 3.3, we are done. $\Box$

Theorem 3.2 *The problem of deciding whether a given finite-time temporal database over the vocabulary considered above can be extended to a temporal model of $\tilde{\varphi}$ is Π^0_2-complete.*

Proof: The argument is analogous as in the proof of Theorem 3.1. However, in the reduction, given $w \in \{0,1\}^*$, we have now to construct a finite-time temporal database, say $D_w = (D_0, D_1, \ldots, D_{t_w})$, that usually has more than one state. It is easy to construct D_w in such a way that D_0 is the encoding of the initial configuration of M on the input w, in any possible extension of D_w satisfying $\tilde{\varphi}$ (due to appropriate choice of the predicate W in the database states $D_0, D_1, \ldots, D_{t_w}$). $\Box$

The following is obtained by a slight modification of the proof of Corollary 3.1.

Corollary 3.2 *The formula $\tilde{\varphi}$ is not equivalent to any formula of the form $\Box\, \theta$, with θ being a past formula.*

4 Universal formulas

We now fix an arbitrary finite vocabulary L. Recall that, by definition, the universe of a database is infinite and countable and, by convention, it coincides with N. In what follows, we will sometimes relax this last convention. Let A be an infinite subset of the universe of a database D containing the interpretations of all the constant symbols of L. The restriction of D to A, in symbols $D|A$, is a database with the universe A in which the interpretation of a predicate symbol p of arity r at a time instant t is given by $p^{D_t} \cap A^r$. Clearly, one could find a database with the universe N isomorphic to $D|A$. The following property will be useful : if D is a model of a *universal* temporal sentence, so is $D|A$.

Let $D = (D_0, D_1, \ldots, D_t, \ldots)$ be a finite- or infinite-time temporal database with the universe N. We call an element $m \in N$ *relevant to* D if m is an interpretation of some constant symbol or m is in the domain of the interpretation of some relation symbol in some state of D. Otherwise, an element m is *irrelevant to* D. Let R_D be the set of all elements relevant to D. Let $I_D = N - R_D$. Clearly, for a finite-time temporal database D, R_D is finite and I_D is infinite.

Lemma 4.1 *Let φ be a universal safety sentence. Suppose that a finite-time temporal database $D = (D_0, D_1, \ldots, D_t)$ has an extension to an infinite-time temporal database satisfying φ. Then D can be also extended to some D' satisfying φ, such that $R_{D'} = R_D$.*

Proof:
Let D'' be an extension of D satisfying φ. We define D' by setting $D'_i = D_i$ for $i = 0, 1, \ldots t$, and for each $n > t$, each predicate letter p of arity r, and $a_1, \ldots, a_r \in N$:

$$p^{D'_n}(a_1, \ldots, a_r) \text{ iff } p^{D''_n}(a_1, \ldots, a_r) \text{ and } a_1, \ldots, a_r \in R_D.$$

Clearly $R_{D'} = R_D$. In order to verify that D$' \models \varphi$, it is enough to show that for each $n \in N$, the prefix $(D'_0, \ldots, D'_n)$ of D$'$ has an extension satisfying φ. We can assume $n > t$. Let $I_{D''_n}$ be the set of those $n \in N$ that are not relevant to D'' in the prefix $(D''_0, \ldots, D''_n)$. That is, $m \in I_{D''_n}$ iff m is neither an interpretation of a constant symbol nor is it in the domain of the interpretation of a relation symbol in some state D''_m, for $m \leq n$. Consider the temporal database E obtained by the restriction of D$''$ to the set $R_D \cup I_{D''_n}$. Because φ is a universal formula, E is also a model of φ. But the prefix $(E_0, \ldots, E_n)$ is clearly isomorphic to $(D'_0, \ldots, D'_n)$. Then, we can infer that $(D'_0, \ldots, D'_n)$ itself has the desired extension. $\Box$

The important point in the above lemma is that the universe of the model D' in consideration must be infinite, according to our usual requirement about databases. (In particular, the restriction of the above D'' to R_D, although it is obviously a model of φ, cannot be considered a temporal database unless we allow temporal databases with finite universes.)

To illustrate the importance of the above point, we show an example of a universal temporal formula that has models with arbitrary large finite universes but does not have a model with an infinite universe. Let the formulas **W1** and $x \preceq_W y$ be as in section 3. Define:

W4 $(\forall x)((\neg W(x))$ **until** $(W(x) \wedge \bigcirc \Box \neg W(x)))$.

This formula forces that any element of the universe satisfies the relation W in exactly one database state. Let Q be a fresh monadic predicate letter and let **Q1**, **Q4**, and $\preceq_Q$ be defined like **W1**, **W4**, and $\preceq_W$ resp., except that all the occurrences of W are replaced by Q. Take the conjunction of **W1**, **W4**, **Q1**, **Q4**, and the formula

$$(\forall x, y)(x \preceq_Q y \Rightarrow y \preceq_W x).$$

The last formula asserts that the ordering of the universe induced by Q is the inverse of the ordering induced by W. Together with **Q4** it forces the existence of a $\preceq_W$-decreasing chain formed by all elements of the universe.

Theorem 4.1 *Given a finite-time temporal database $D = (D_0, D_1, \ldots, D_t)$ and a universal safety sentence $\varphi \equiv \forall x_1 \ldots \forall x_k \psi$, one can construct a finite sequence of propositional states $w^D = (w_0, w_1, \ldots, w_t)$ and a formula φ_D of propositional TL such that D can be extended to an infinite-time temporal database satisfying φ iff w^D can be extended to an infinite sequence of propositional states satisfying φ_D. The formula φ_D can be chosen of the size $O((|\varphi| \cdot |R_D|)^{\max(k,l)})$ where l is the maximum arity of database relations.*

Proof: Let $\varphi \equiv \forall x_1 \ldots \forall x_k \psi$. Without loss of generality, we can assume that $R_D = \{0, 1, 2, \ldots, m_D\}$ for some $m_D \in N$. Let C_L be the (finite) set of constants symbols of the vocabulary L. We also fix some k symbols, say $z_1, \ldots, z_k$, disjoint from R_D and C_L (intuitively, they will represent the elements of the universe outside R_D). Let $M = R_D \cup \{z_1, \ldots, z_k\}$. We define a vocabulary of the propositional temporal logic as a set L_D consisting of the following propositional symbols:

- $(a = b)$ for each $a, b \in M \cup C_L$,
- $p(a_1, \ldots, a_{ar(p)})$ for each predicate p of L and each sequence $a_1, \ldots, a_{ar(p)} \in M \cup C_L$.

(Note that we have intentionally chosen some well formed first-order formulas to denote the propositional letters in L_D.)

Now let $f : \{x_1, \ldots, x_k\} \longrightarrow M$ be any mapping. For each atomic formula α over L with variables among $\{x_1, \ldots, x_k\}$, let $\alpha[f]$ be the result of the substitution of $f(x_i)$ for x_i. For example, if $f(x_1) = 5$ and $f(x_2) = z_3$ then $(x_2 = x_1)[f] \equiv (z_3 = 5)$. Notice that each $\alpha[f]$ can be identified with a propositional symbol in L_D. The $\alpha[f]$ operation can be extended in a natural way to all quantifier-free first-order temporal formulas over L, with variables among $\{x_1, \ldots, x_k\}$; $\alpha[f]$ is then a formula of the propositional temporal logic over L_D. Let

$$\psi_D \equiv \bigwedge_f \psi[f]$$

where f ranges over all mappings from the set $\{x_1, \ldots, x_k\}$ to M.

We also define the propositional temporal formula $Axiom_D$, as the conjunction of the following:

- $(a = a)$ for each $a \in M \cup C_L$,
- $(a = b) \Leftrightarrow (b = a)$ for each $a, b \in M \cup C_L$,

- $(a = b) \wedge (b = c) \Rightarrow (a = c)$ for each $a, b, c \in M \cup C_L$,
- $(a_1 = b_1) \wedge \ldots \wedge (a_{ar(p)} = b_{ar(p)}) \Rightarrow p(a_1, \ldots, a_{ar(p)}) \Leftrightarrow p(b_1, \ldots, b_{ar(p)})$ for each predicate p of L and each $a_1, \ldots, a_{ar(p)}$, $b_1, \ldots, b_{ar(p)} \in M \cup C_L$,
- $(i = c)$ for each $i \leq m_D$ and $c \in C_L$, such that $i = c_0^D$,
- $(c = d)$ for each $c, d \in C_L$, s.t. $c^{D_0} = d^{D_0}$,
- $\neg(i = j)$ for each $i, j \leq m_D$, $i \neq j$,
- $\neg(i = c)$ for each $i \leq m_D$ and $c \in C_L$ s.t. $i \neq c^{D_0}$,
- $\neg(c = d)$ for each $c, d \in C_L$, s.t. $c^{D_0} \neq d^{D_0}$,
- $\neg(z_i = j)$ for each z_i and each $j \leq m_D$,
- $\neg(z_i = z_j)$ for each z_i, z_j, such that $i \neq j$,
- $\neg p(a_1, \ldots, a_{ar(p)})$ for each predicate p of L and each $a_1, \ldots, a_{ar(p)} \in M \cup C_L$ s.t. at least one a_i is among $\{z_1, \ldots, z_k\}$.

We set

$$\varphi_D \equiv_{df} \psi_D \wedge \Box\, Axiom_D.$$

We are now going to define the sequence of propositional states $w^D = (w_0, w_1, \ldots, w_t)$. Recall that a propositional state is a mapping from the set of propositional letters to the set $\{\text{true}, \text{false}\}$. Intuitively, w_ℓ is a description of the database state D_ℓ. Recall that the interpretation of a constant symbol c is the same in each database state, in symbols c^D.

We set

- $w_\ell(c = d) = \text{true}$ iff $c^D = d^D$ for all constant symbols $c, d \in C_L$,
- $w_\ell(c = i) = w_\ell(i = c) = \text{true}$ iff $c^D = i$ for $c \in C_L$, $i \leq m_D$,
- $w_\ell(i = i) = \text{true}$,
- $w_\ell(i = j) = \text{false}$ for $i \neq j$,
- $w_\ell(z_i = z_i) = \text{true}$,
- $w_\ell(z_i = z_j) = \text{false}$ for $i \neq j$,
- $w_\ell(z_i = c) = w_\ell(c = z_i) = \text{false}$ for $c \in C_L$,
- $w_\ell(z_i = j) = w_\ell(j = z_i) = \text{false}$,
- $w_\ell(p(a_1, \ldots, a_{ar(p)})) = \text{true}$ for $a_1, \ldots, a_{ar(p)} \in \{0, 1, \ldots, m_D\} \cup C_L$ iff the predicate p is true for the vector $(a_1, \ldots, a_{ar(p)})$ in the database state D_ℓ,

- $w_\ell(p(a_1,\ldots,a_{ar(p)}) =$ false whenever at least one of the a_i's is in among $\{z_1,\ldots,z_k\}$.

Suppose now that if $D = (D_0, D_1, \ldots, D_t)$ has an extension to an infinite-time temporal database $D' = (D_0, D_1, \ldots, D_t, \ldots)$ satisfying φ. By Lemma 4.1, we can assume that $R_{D'} = R_D$. Let

$$w_\infty^D = (w_0, w_1, \ldots, w_t, w_{t+1}, w_{t+2}, \ldots)$$

where for $\ell > t$, w_ℓ is defined in the same way as for $\ell \leq t$ (see above). We will verify that w_∞^D satisfies φ_D.

Clearly, w_∞^D satisfies $\Box\, Axiom_D$. Now let $f : \{x_1,\ldots,x_k\} \longrightarrow M$. Fix some distinct k elements $b_1,\ldots,b_k \in N - R_D$ and define a valuation v, by

$$v(x_i) \quad = \begin{cases} f(x_i), & \text{if } f(x_i) \in R_D \\ b_j, & \text{if } f(x_i) = z_j. \end{cases}$$

Now it is easy to verify by structural induction that for any quantifier-free temporal formula $\alpha(x_1,\ldots,x_k)$, and for any time instant s

$$D', v, s \models \alpha \text{ iff } w_\infty^D, s \models \alpha[f].$$

Since by our assumption $D', v, 0 \models \psi$, we can conclude that $w_\infty^D \models \psi[f]$. Because f was arbitrarily chosen, we have $w_\infty^D \models \psi_D$, and hence also $w_\infty^D \models \varphi_D$, as required.

To show the other direction, suppose w^D can be extended to to an infinite sequence that is a model of φ_D. Then in order to obtain an extension of D satisfying φ, we need to "decode" propositional states into relations. Suppose that the sequence $w^D = (w_0, w_1, \ldots, w_t)$ can be extended to an infinite sequence of propositional states, say

$$w_\infty^D = (w_0, w_1, \ldots, w_t, w_{t+1}, w_{t+2}, \ldots)$$

which satisfies φ_D. For each $i > t$, we define a database state D_i as follows. The interpretation of the constant symbols is the same as in D. For a predicate symbol p and $a_1,\ldots,a_{ar(p)} \in N$, $p^{D_i}(a_1,\ldots,a_{ar(p)})$ holds iff $a_1,\ldots,a_{ar(p)} \in R_D$ and the propositional symbol $p(a_1,\ldots,a_{ar(p)})$ evaluates to true in the propositional state w_i, otherwise $p^{D_i}(a_1,\ldots,a_{ar(p)})$ does not hold. We will show that the resulting infinite-time database $D_\infty = (D_0, D_1, \ldots, D_t, D_{t+1}, D_{t+2}, \ldots)$ satisfies φ. Let v be a valuation. Suppose that the values $v(x_i)$, $i = 1,\ldots,k$, which are not in R_D form a set $\{b_1,\ldots,b_p\}$ (this set can be empty). Fix some embedding e of this set into $\{z_1,\ldots,z_k\}$. Define $f : \{x_1,\ldots,x_k\} \longrightarrow M$ as

$$f(x_i) \quad = \begin{cases} v(x_i), & \text{if } v(x_i) \in R_D \\ e(v(x_i)), & \text{otherwise.} \end{cases}$$

Now, using the fact that $w_\infty^D \models \Box\, Axiom_D$, it is easy to verify by structural induction that, for any quantifier-free temporal formula $\alpha(x_1,\ldots,x_k)$, and for any time instant s,

$$D_\infty, v, s \models \alpha \text{ iff } w_\infty^D, s \models \alpha[f].$$

Since, by our assumption $w^D_\infty, 0 \models \psi[f]$ for any f, we have $D_\infty, v, 0 \models \psi$ for any v, and consequently $D_\infty \models \varphi$, as required.

The polynomial upper bound on the size of φ_D follows directly from the proof. The exponent is due to the fact that the size of $Axiom_D$ is $O(|R_D|^{\max(k,l)})$. $\Box$

The above theorem will allow us to reduce the question if a finite-time temporal database can be extended to a model of a universal safety sentence to the question of satisfiability of some propositional temporal formula. By the results on propositional temporal logic, this will yield decidability of the extension problem.

Lemma 4.2 *Given a sequence of propositional states $w = (w_0, w_1, \ldots, w_t)$ and a formula α of propositional TL, one can decide if w can be extended to a model of α in time $O(t \cdot |\alpha|) + 2^{O(|\alpha|)}$ and space $t \cdot |\alpha|^{O(1)}$.*

Proof: The algorithm consists of two phases. The first phase is deterministic: we check whether the propositional states $w_0, w_1, \ldots, w_t$ are consistent with α, i.e., whether they can form a prefix of a model of α. This can be done using the approach of Sistla and Wolfson [26]. The essence of this approach consists of building a formula $\alpha_{t'}$ on the basis of α and $w = (w_0, w_1, \ldots, w_t)$ for $t' = 0, 1, \ldots, t$. This formula will be tested for satisfiability in the second phase.

We show how to build the formula α_0. Start with the expression $[\alpha]_0$. We will push the subscript (denoting the state index in the prefix w) inside the formula. If $\alpha \equiv \beta$ **until** γ, then it will be rewritten to $[\gamma]_0 \vee [\beta]_0 \wedge [\alpha]_1$. If $\alpha \equiv \beta \wedge \gamma$, then it will be rewritten to $[\beta]_0 \wedge [\gamma]_0$. If $\alpha \equiv \neg\beta$, then several cases are distinguished. If $\beta \equiv \neg\gamma$, then $[\alpha]_0$ is rewritten to $[\gamma]_0$. If $\beta \equiv \gamma \wedge \delta$, then $[\alpha]_0$ is rewritten to $[\neg\gamma]_0 \vee [\neg\delta]_0$. If $\beta \equiv \gamma$ **until** δ, then the result of the rewriting is $[\neg\delta]_0 \wedge ([\neg\gamma]_0 \vee [\alpha]_1)$. If $\beta \equiv \bigcirc\,\gamma$, then $[\alpha]_0$ is rewritten to $[\neg\gamma]_1$. Finally, if $\alpha \equiv \bigcirc\,\beta$, then $[\alpha]_0$ is rewritten to $[\beta]_1$. Subsequently, the expressions with subscript 0 are further rewritten using the same rules. It should be clear that as the result of the rewriting, we obtain a boolean combination of propositional atoms with subscript 0 and expressions of the form $[\beta]_1$ where β is a subformula of the original formula or its negation. Now atoms of the first kind can be replaced by **true** or **false** depending on the propositional state w_0, and the resulting formula simplified. In this way we obtain the formula α_0. In the next step expressions with subscript 1 are rewritten and the resulting formula simplified to obtain α_1 etc. Finally, the formula α_t is obtained.

The second phase verifies the satisfiability of α_t using the nondeterministic polynomial space procedure of Sistla and Clarke [25].

It is easy to see that every formula α_i, $0 \le i \le t$, is of size at most $O(|\alpha|)$. Thus the first phase takes $O(t \cdot |\alpha|)$ time, and the second phase takes $2^{O(|\alpha|)}$ time. $\Box$

From this and Theorem 4.1, we can infer our main result about universal formulas.

Theorem 4.2 *Given a finite-time temporal database $D = (D_0, D_1, \ldots, D_t)$ and a universal safety sentence $\varphi \equiv \forall x_1 \ldots \forall x_k \psi$, one can decide if D can be extended to an infinite-time temporal database satisfying φ in time $O(t \cdot (|\varphi| \cdot |R_D|)^{\max(k,l)}) + 2^{O((|\varphi| \cdot |R_D|)^{\max(k,l)})}$ where l is the maximum arity of database relations.*

In the formulation of Theorem 4.2 it is essential that φ be a safety sentence. For universal sentences that are not *safety* sentences, e.g., $\forall x \diamond p(x)$, Lemma 4.1 fails and the proofs of Theorem4.1 and Theorem 4.2 do not go through. This fact provides an additional justification for considering only safety formulas as integrity constraints.

5 Related work

It seems safe to say that *FOTL* is the preferred language for the specification of temporal integrity constraints as evidenced by the following papers (the list is by no means complete): [3, 4, 5, 7, 14, 17, 18, 19, 20, 22, 26].

Lipeck, Saake, and their students [14, 17, 19, 20] introduced the class of *biquantified* formulas (although they didn't use this term), proposed several constraint checking methods for this class, and were the first to define a notion equivalent to potential constraint satisfaction. It is clear from the lower recursion-theoretic bounds established in the present paper that checking potential constraint satisfaction for the full class of biquantified formulas is not computationally feasible. Therefore, the methods of Lipeck and Saake implemented by necessity a weaker notion of constraint satisfaction, namely one in which constraint violations are always detected but not necessarily at the earliest possible time.

In our earlier work [3] we introduced *Past FOTL* (*FOTL* with past operators only) as the language for specifying temporal integrity constraints and proposed an efficient method to evaluate such constraints (again, under a weaker notion than potential constraint satisfaction). We have also extended *Past FOTL* to *Past Metric FOTL* in order to be able to formulate real-time constraints (constraints that refer to the values of a clock).

Qian and Waldinger [22] also recognized the need for efficient checking of temporal constraints. They presented many compelling examples but no general method. Castilho, Casanova and Furtado [7] and Kung [18] dealt mainly with the issue of consistency of temporal constraints with action specifications and did not study integrity checking in the presence of arbitrary updates.

Sistla and Wolfson [26] discussed triggers whose conditions are temporal formulas. However, instead of standard first-order quantifiers, they used *freeze* quantifiers with non-standard semantics. The expressive power of their trigger language depends on the query language of the underlying DBMS. It is interesting to note that if the latter language is a first-order query language like relational calculus or algebra, the trigger language of [26] is as expressive as the class $\exists^* tense(\Sigma_*)$ of *FOTL* formulas, i.e., the class of negations of biquantified formulas. The integrity checking method of [26] implements, like [20], a notion of constraint satisfaction that is weaker than potential constraint satisfaction.

In the object-oriented context Gehani, Jagadish and Shmueli [12, 13] discussed triggers that are fired by event occurrences. They introduced a language of extended regular expressions for specifying composite events.

6 Conclusions and further work

We have analyzed the computational feasibility of checking temporal integrity constraints formulated in some sublanguages of *FOTL*. Our results illustrate the impact of the quantifier pattern on the complexity of this problem. The presence of a single quantifier in the scope of a temporal operator makes the problem undecidable. On the other hand, if no quantifiers are in the scope of a temporal operator and all the quantifiers are universal, temporal integrity checking can be done in exponential time.

The most immediate question is whether one can remove $|R_D|$ (the size of the set of relevant domain elements) from the exponent in the time bound for universal formulas. Most probably no, as shown by the following argument. Consider a deterministic Turing machine M that decides the problem *SAT* within polynomial space and exponential time. Using techniques similar to those used in section 3, we can encode an initial configuration of this machine by a single database state D_0. We can make this initial configuration as large as the maximal space needed for the computation. Then, it is not difficult to write a universal formula φ, defining a safety property, such that the singleton sequence (D_0) can be extended to a model of φ iff M accepts. (Notice that we are now dealing with a machine operating in a bounded space. It is enough that the successor relation will be correctly defined in D_0; the formula φ can force that this relation remains the same throughout the other database states. Therefore, the problem resulting from the impossibility of axiomatizing the successor relation by a universal formula, that we have encountered in section 3, does not arise here.) Now, if we could answer the extension question within a time polynomial in the size of D_0 then we could solve *SAT* in polynomial time, which is believably not possible.

It is essential that a universal sentence be a safety sentence for the exponential upper bound to hold. Sistla [24] showed that propositional safety formulas can be characterized syntactically. He also proved that recognizing propositional safety formulas is decidable. We conjecture that his results generalize to universal biquantified formulas.

To make temporal integrity checking more practical, weaker notions of constraint satisfaction should be considered and their computational complexity analyzed. An important notion in this area is that of a *history-less* constraint evaluation [3, 4]. Intuitively, the complexity of history-less constraint evaluation does not depend on the length of the database history but only on the number of different attribute values that appear in it. A history-less evaluation method for *Past FOTL* implementing a notion weaker than potential constraint satisfaction is presented in [3]. It remains to be seen whether a history-less method can be devised for universal biquantified formulas.

Acknowledgments

The comments of the anonymous referees are gratefully acknowledged.

References

[1] Alpern, B., and Schneider, F. Defining Liveness. *Inf. Process. Lett. 21* (1985), 181–185.

[2] Chang, C., and Keisler, H. *Model Theory.* North-Holland, 1977.

[3] Chomicki, J. History-less Checking of Dynamic Integrity Constraints. In *IEEE International Conference on Data Engineering* (Phoenix, Arizona, February 1992).

[4] Chomicki, J. Real-Time Integrity Constraints. In *ACM Symposium on Principles of Database Systems* (San Diego, California, June 1992).

[5] Chomicki, J. Temporal Integrity Constraints in Relational Databases. *IEEE Data Engineering Bulletin 17*, 2 (June 1994), 33–37.

[6] Dayal, U., et al. The HiPAC Project: Combining Active Databases and Timing Constraints. *SIGMOD Record 17*, 1 (March 1988), 51–70.

[7] de Castilho, J., Casanova, M., and Furtado, A. A Temporal Framework for Database Specifications. In *International Conference on Very Large Data Bases* (1982), pp. 280–291.

[8] Emerson, E. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B. Elsevier/MIT Press, 1990, ch. 16, pp. 995–1072.

[9] Finger, M., and Gabbay, D. Adding a temporal dimension to a logic system. *Journal of Logic, Language and Information* (1992).

[10] Gabbay, D., Hodkinson, I., and Reynolds, M. *Temporal Logic: Mathematical Foundations and Computational Aspects.* Oxford University Press, 1994.

[11] Garson, J. Quantification in Modal Logic. In *Handbook of Philosophical Logic*, D. Gabbay and F. Guenthner, Eds. D. Reidel, 1984, ch. II.5, pp. 249–307.

[12] Gehani, N., Jagadish, H., and Shmueli, O. Composite Event Specification in Active Databases: Model & Implementation. In *International Conference on Very Large Data Bases* (1992), pp. 327–338.

[13] Gehani, N., Jagadish, H., and Shmueli, O. Event Specification in an Active Object-Oriented Database. In *ACM SIGMOD International Conference on Management of Data* (1992), pp. 81–90.

[14] Gertz, M., and Lipeck, U. Deriving Integrity Maintaining Triggers from Transition Graphs. In *IEEE International Conference on Data Engineering* (1993).

[15] Harel, D. Recurring Dominoes: Making the Highly Undecidable Highly Understandable. *Annals of Discrete Mathematics 24* (1985), 51–71.

[16] Hopcroft, J., and Ullman, J. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[17] Hülsmann, K., and Saake, G. Theoretical Foundations of Handling Large Substitution Sets in Temporal Integrity Monitoring. *Acta Informatica 28*, 4 (1991).

[18] Kung, C. On Verification of Database Temporal Constraints. In *ACM SIGMOD International Conference on Management of Data* (Austin, Texas, 1985), pp. 169–179.

[19] Lipeck, U., Gertz, M., and Saake, G. Transitional Monitoring of Dynamic Integrity Constraints. *IEEE Data Engineering Bulletin* (June 1994).

[20] Lipeck, U., and Saake, G. Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems 12*, 3 (1987), 255–269.

[21] Pnueli, A. Applications of Temporal Logic to the Specification and Verification of Reactive Systems: a Survey of Current Trends. In *Current Trends in Concurrency*. Springer-Verlag, LNCS 224, 1986.

[22] Qian, X., and Waldinger, R. A Transaction Logic for Database Specification. In *ACM SIGMOD International Conference on Management of Data* (1988), pp. 243–250.

[23] Rogers Jr., H. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.

[24] Sistla, A. On Characterization of Safety and Liveness Properties in Temporal Logic. In *ACM Symposium on Principles of Distributed Computing* (1985), pp. 39–48.

[25] Sistla, A., and Clarke, E. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM 32*, 3 (July 1985), 733–749.

[26] Sistla, A., and Wolfson, O. Temporal Triggers in Active Databases. *IEEE Transactions on Knowledge and Data Engineering* (1995). To appear.

[27] Snodgrass, R. Temporal Databases. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space* (1992), Springer-Verlag, LNCS 639, pp. 22–64.

[28] Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R., Eds. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.

Appendix

A temporal formula encoding repeating computations of a Turing machine

We are going to describe a way in which the computations of Turing machines can be encoded by temporal databases, and to present a temporal formula expressing the property that the encoded computation is repeating. This will be the formula φ that is required in Proposition 3.1. Recall that, in this formula, we are allowed to use the *extended* vocabulary (c.f., Section 2). More specifically, our vocabulary will consist of a finite number of unary predicate symbols, and the specific predicate symbols $\leq$ (binary), *succ* (binary) and *Zero* (unary) interpreted over the universe N in the standard way. Note that the interpretation of any unary predicate symbol in any database state is finite.

Let us fix a Turing machine M satisfying the property of Lemma 3.1. We may assume that M has one tape infinite to the right. Recall that the alphabet Σ of the tape symbols used by M includes the input alphabet $\{0, 1\}$ and the special blank symbol B. Let Q be the set of states of M including the initial state q_0.

We assume the reader is familiar with the concept of *configuration* of a Turing machine (also called instantaneous description [16]). For our purpose, it is convenient to slightly modify this concept and to present a configuration as an *infinite*, rather than finite, sequence of elements in $\Sigma \cup Q$. Such a sequence is of the form $\alpha q \beta B^\omega$, where $\alpha, \beta \in \Sigma^*$ and $q \in Q$. Note that there is exactly one occurrence of a machine's state symbol and that all but finitely many symbols in the sequence are B. Thus a configuration corresponds precisely to an actual content of the Turing machine's tape, and the position of q indicates the actual position of the machine's head: it is supposed to be scanning the first symbol of βB^ω. (Notice that we do not forbid B to occur in α or β.) An *initial configuration* is of the form $q_0 w B^\omega$, where $w \in \{0,1\}^*$. Now, for a configuration C, there is at most one configuration C' resulting from C by one move of M. If it is the case and $C = c_0 c_1 \ldots$, $C' = c'_0 c'_1 \ldots$, then each three consequtive letters $c_i c_{i+1} c_{i+2}$ determine uniquely c'_{i+1}, and $c_0 c_1$ determines uniquely c'_0. Now a *computation* of M with an input w can be identified with a sequence of configurations $C_0, C_1, \ldots$, where C_0 is an initial configuration, say $C_0 = q_0 w B^\omega$, and for each i, C_{i+1} results from C_i by one move. Moreover, the input w induces a repeating behaviour of M if the computation sequence is infinite and contains infinitely many configurations of the form $q\alpha B^\omega$. We call such a sequence a *repeating computation* (of M).

Our vocabulary will include a monadic symbol P_q for each state $q \in Q$ and a monadic predicate symbol P_σ for each $\sigma \in \Sigma$ except for B. We say that a database state D *encodes a configuration* $C = c_0 c_1 \ldots$ if for each $i \in N$, at most one monadic predicate is true about i and, for each $z \in Q \cup \Sigma - \{B\}$ the predicate corresponding to the symbol P_z is true about i iff $c_i = z$. Notice that $c_i = B$ if and only if none of the predicates P_z is true about i. Since B is the only symbol that can occur in a configuration infinitely often, each configuration is indeed encoded by a database state. It will be further convenient to use the notation $P_B(x)$ as an abbreviation for $\bigwedge_{z \in Q \cup \Sigma - \{B\}} \neg P_z(x)$.

Now, we say that a temporal structure $\mathbf{D} = (D_0, D_1, \ldots)$ *encodes a computation* $C_0, C_1, \ldots$, if each D_i encodes C_i, for $i = 0, 1, \ldots$

We will construct a formula which will force a temporal database to encode a repeating computation of M. At first, we construct several formulas, each one of them capturing one of the desired properties.

1. For any element i of the universe, in any database state at most one of the monadic predicates $P_q, q \in Q$, $P_\sigma, \sigma \in \Sigma - \{B\}$ is true about i.

$$\forall x \, \Box \bigwedge_{a,b \in \Sigma \setminus \{B\} \cup Q, a \neq b} \neg(P_a(x) \wedge P_b(x))$$

2. The initial database state encodes some initial configuration of M.

$$\forall x, y \, \mathit{Zero}(x) \Rightarrow P_{q_0}(x)$$

$$\wedge (\neg \mathit{Zero}(x) \wedge x \leq y \wedge \neg P_B(y)) \Rightarrow ((P_0(y) \vee P_1(y)) \wedge (P_0(x) \vee P_1(x)))$$

(Note that we do not need to express the property that there *is* some B since it follows from our proviso that the interpretation of any of the predicate symbol P_a, $a \in \Sigma \cup Q$, is finite.)

3. Subsequent database states encode subsequent configurations. For this, we shall use the fact that any three consecutive positions of a configuration determine uniquely the content of the middle position in the next configuration. More specifically, we shall need a formula for each transition of M. Suppose there is a transition of the form $q, \sigma \rightarrow p, \rho, R$ (i.e., if the machine is scanning symbol σ in state q then in the next move it changes state to p, replaces σ by ρ and moves the head right). Then the corresponding formula is

$$\forall x, y, z \bigwedge_{c \in \Sigma} \Box\, (succ(x, y) \wedge succ(y, z) \wedge P_q(x) \wedge P_\sigma(y) \wedge P_c(z)$$

$$\Longrightarrow \bigcirc\, (P_\rho(x) \wedge P_p(y) \wedge P_c(z)))$$

Formulas for other transitions are similar.

Also, we will have a formula for the case where none of the three consecutive positions contains a state. This formula will be

$$\forall x, y, z \bigwedge_{a,b,c \in \Sigma} \Box\, (succ(x, y) \wedge succ(y, z) \wedge P_a(x) \wedge P_b(y) \wedge P_c(z)$$

$$\Longrightarrow \bigcirc\, P_b(y))$$

4. There are infinitely many database states which encode configurations with the machine's head scanning the leftmost tape cell.

$$\forall x\, Zero(x) \;\Rightarrow\; \Box \Diamond \bigvee_{q \in Q} P_q(x)$$

Now, it is easy to see that a temporal database satisfies the *conjunction* of all the above formulas iff it is an encoding of a repeating computation of the machine M, in the sense explained above. Note that any of the above formulas can be written in the form $\varphi \equiv \forall x_1 \forall x_2 \forall x_3 \psi$, where ψ is quantifier-free. Moreover, the conjunction of those formulas can be also written in such a form. Thus, the proof of Proposition 3.1 is completed.