Periodicity Based Decidable Classes in a First Order Timed Logic

Danièle Beauquier, Anatol Slissenko

Periodicity Based Decidable Classes in a First Order Timed Logic

D. Beauquier [2,1] A. Slissenko [2,3]

Laboratory for Algorithmics, Complexity and Logic, CNRS FRE 2673
University Paris-12, France

Abstract

We describe a decidable class of formulas in a first order timed logic that covers a good amount of properties of real-time distributed systems. Earlier we described a decidable class based on some finiteness properties, and sketched a decidable class in a weaker logic that captures periodicity properties, though without complete proof. The new feature of the decidable class presented here is to be able to treat parametric properties, in particular, properties that concern an arbitrary number of processes as compared to the just mentioned class that could express only properties for a fixed, explicitly given number of processes. As before the properties may use non-trivial arithmetics. We also state another important feature of the class: if a verification formula is not true then our algorithm gives a quantifier-free description of all its counter-models of the complexity involved in the definition of the decidable class and that is quite representative. In conclusion we discuss theoretical and practical issues of the complexity and some open questions.

Key words: verification, decidability, first order timed logic, periodic models.
1991 MSC: 68Q60, 03B70

1 Introduction

There are two main approaches to treat the verification problem: model-checking and theorem-proving.

[1] Corresponding author.
[2] Address: Dept. of Informatics, University Paris-12, 61 Av. du Gén. de Gaulle, 94010, Créteil, France. Tel: 33 (0)1 45 17 16 47 Fax: 33 (0)1 45 17 66 01.
E-mail: {beauquier,slissenko@univ-paris12.fr}
[3] Member of St-Petersburg Institute for Informatics and Automation, Academy of Sciences of Russia.

The model-checking approach replaces the initial problem by its model that can be very precise for systems with simple structure and without parameters, but can be far enough from the realistic system and laborious to construct for more complicated systems. The advantage of model-checking is that many computer tools, that can treat the constructed models automatically, were and are being developed. The model-checking approach uses temporal logics (in fact, in a very limited way) to represent requirements and finite transition systems to model the program to verify. Many known benchmark problems like clock synchronization (even for a fixed number of clocks) [Tel00], 15.3, are out of the scope of presently known model-checking methods.

The other pole of verification uses theorem provers that permit to represent the verification problem in a direct way, but the proof search is mainly by hand. There are many interesting efforts to automate the proof search but nowadays they are not sufficient to facilitate the proof search essentially. The existing general theorem provers do not have sufficiently developed tools, say, to represent human heuristics in a user friendly way.

The approach we develop [BS02b] is aimed, in particular, to fill the gap between the efficiency of model-checking and the generality and expressibility of theorem proving. This gap can be filled by decision algorithms for classes of verification problems [BS02b,BS02a] and by convenient methods of representation of heuristics [Sli99,Sli03]. Here we present results on decidable classes.

From the point of view of logic the verification problem is a problem of proving a formula of the form $\Big(\ \Phi_{Runs} \to \Phi_{Rqrm}\ \Big)$, where Φ_{Runs} is a formula representing all the runs of the program to verify and Φ_{Rqrm} is a formula representing the requirements of the functioning of the program. In fact, the formula Φ_{Runs} is often a conjunction of two parts: one part describes the runs themselves and another part describes the environment of the program, for example the properties of input signals like delays of communications, reaction of controlled devices etc.

Example 1. Suppose we have a distributed algorithm with N processes, and property $R(t,p)$ says that at moment t there is a certain event in the process p. We can express that "an R-event cannot be absent in the same process for a duration greater than d" by the formula

$$\forall\, p \neg \exists\, t \exists\, t' \ \Big(\ (t'-t) > d \wedge \forall\, \tau \in [t,t') \neg\, R(\tau,p)\ \Big)\,. \tag{1}$$

One may think that this event R as well as its absence must be communicated to other processes to make some consent decisions (a distributed system without interaction of processes is not a genuine distributed system). □

Describing runs of a program is a more laborious procedure, however, for a given specification language this can be automated, at least to get some basic complete representation of runs. The word "basic" refers to the fact that though such a representation is complete, it could be not sufficiently good for efficient practical application. To make such a representation practically efficient one may add some simplifications for particular types of programs, but again such simplifications can be automated, see [BCS00].

The both sets of models mentioned above, namely the models describing requirements and the models describing runs, can be expressed in the logic that we study — First Order Timed Logic (FOTL) with continuous time. We can take also a logic with discrete time. However, there are many situations when continuous time is more adequate, for example, controllers are often specified using continuous time; one can find continuous time in network protocol specifications. In our intuitive arguments we often use continuous time. Moreover, algorithmics of continuous time in our framework is simpler because of the choice of underlying logical theories treating arithmetic operations over time. The known worst-case complexity bounds for the theory of real addition are exponentially better than that for the theory of integer addition (Presburger arithmetic). For the theory of real addition and multiplication (Tarski algebra) these bounds, that are the same as for the theory of real addition, are even 'infinitely' better than that for the theory of integer addition and multiplication (formal arithmetics) that is undecidable.

What is also important is that quantifier elimination for the theory of real addition is supported by tools that are rathe often efficient in practice; as for Presburger arithmetic the existing tools are much less efficient – though our experience shows that for the both theories the efficiency of tools can be considerably improved for the verification. However, the method of the present paper, even if we start with the theory of real addition for specifications, brings us to Presburger arithmetics to solve the verification problem.

The decidable class of verification problems we consider here is based on the following observations. The properties related to the functioning of a program are usually finitely refutable, that is if there is a counter-model for such a property then the contradiction is concentrated on a small piece of this counter-model. In Example 1, if the property (1) is false then there is a process p_0 and 2 time moments t_0 and t_1 such that

$$\Big((t_1 - t_0) > d \wedge \forall\, \tau \in [t_0, t_1)\, \neg\, R(\tau, p_0) \Big)\,. \tag{2}$$

So whatever be the behavior of the processes different from p_0 or whatever

be the behavior of p_0 at other time moments, the property will remain false. Hence, the 'core' of the counter-model is concentrated on a piece of interpretation of $O(1)$ complexity.

A more involved finiteness property concerns the behavior of a program. Roughly, the property we introduce is a far going generalization of the reducibility of runs of finite automata: if such a run contains sufficiently many states there is a shorter run with the same end states (pumping lemma). In general this property is false even for rather simple timed systems, for example for timed automata [AD94] as shown in [BS02a]. However, for practical systems we often have some kind of reducibility of runs. Here we introduce finite satisfiability property that deals with runs that have a finite description involving infinitely many time intervals as compared to [BS02b] where finiteness was direct – a finite model was a model described by a finite set of intervals such that on each interval the behavior of each function of the vocabulary was represented by a fixed number of parameters. Here this property of finite satisfiability says that if we take a run and some finite partial subrun in it then we can extend it to a run consisting of ultimately periodic pieces (even more general ones — ultimately repetitive pieces) with a controlled augmentation of complexity. This is the case for our Example 1 above and Example 3 in section 3.2 if to write directly a program that verifies the demanded properties. For these examples we cannot replace "ultimately periodic" by "finite". All these examples concern systems that change their states 'frequently', in other words there is an upper bound on the length of time intervals where the parameters defining the functions remain unchanged.

Combining the both properties, namely finite refutability and finite satisfiability, we define a decidable class of implications $\Big(\Phi_{Runs} \rightarrow \Phi_{Rqrm} \Big)$, where Φ_{Runs} is finitely satisfiable and Φ_{Rqrm} is finitely refutable.

This class has also the following property of a high practical importance: if the formula is false, the algorithm can extract a description of all its counter-models (of a given complexity) as a quantifier-free formula. For requirements with parameters, for example with abstract time constants, the algorithm gives the constraints on parameters describing forbidden sets.

The structure of the paper is as follows. In section 2 we describe FOTL logic. Section 3 contains the definitions of finiteness properties. In section 4 we prove that the existence of an ultimately repetitive model or of chains of such models of a given complexity is decidable. This proof, though it concerns a more general case than we studied in [BS99] is simpler than the proof that was sketched in [BS99]. This decidability result gives a decidable class of verification problems. In conclusion we discus s some open questions.

2 First Order Timed Logic (FOTL)

The starting idea of FOTL, if to think about decidable classes, is to choose a decidable theory to treat arithmetics or other concrete mathematical functions, and then extend it by abstract functions of time that are needed to specify problems under consideration. In some way the theory must be minimal to be sufficient for the purposes of good expressibility. For the purposes of the present paper we can take as such an underlying theory of arithmetical operations the theory of mixed real/integer addition with rational constants and unary multiplications by rational numbers. This theory is known to be decidable [Wei99].

Though we can consider either discrete time as non negative integers or continuous time as non negative reals, we take for concreteness the case of continuous time. Thus we can choose as an underlying arithmetical theory either the theory of real addition or the mentioned above theory of mixed addition that might be used to represent, for example, some particular properties not related to time.

Notations 1

- $\mathbb{R}$ *is the set of reals,* $\mathbb{Z}$ *the set of integers and* $\mathbb{N}$ *the set of natural numbers.*
- $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ *time treated as a subsort of* $\mathbb{R}$.
- $Bool = \{\boldsymbol{true}, \boldsymbol{false}\}$ *are Boolean values;* $undef$ *will be used for undefined.*

2.1 Syntax and semantics of FOTL.

Syntax of FOTL.

The vocabulary W of a FOTL consists of a finite set of *sorts*, a finite set of *function symbols* and a finite set of *predicate symbols*. A set of variables is attributed to each sort. Some sorts are predefined, i. e. have fixed interpretations. Here the predefined sorts are the real numbers $\mathbb{R}$ and time $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ treated as a subsort of $\mathbb{R}$. The other sorts are finite. If a finite sort has a fixed cardinality it can be considered as predefined. The interesting finite sorts are those whose cardinality is not specified, for example the set of processes in a distributed algorithm. Natural numbers with order or even with addition are often useful to represent such sorts; for this reason we can add this predefined sort.

Remark 1. Some kind of infinite discrete sorts can also be treated in the same

framework; such sorts appear, for example, in specifications of cryptographic protocols. We will not introduce such sorts for technical simplicity. We will neither admit boolean combinations of finite sorts (they can be eliminated — see [BS02b]). □

Some functions and predicates are also predefined. As predefined constants we take $Bool$ for boolean values, $undef$ and $\mathbb{Q}$ for rational numbers. Addition $+$, subtraction $-$ and scalar multiplications of reals by rational numbers are predefined functions of the vocabulary. The predicates $=, \leq, <$ over reals are predefined predicates of W. The vocabulary contains $=$ for all types of objects, and the identity function id of the type $\mathcal{T} \to \mathcal{T}$ to represent the current time.

An *abstract function* (i. e. without any a priori fixed interpretation) is of the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$, and an *abstract predicate* is of the type $\mathcal{T} \times \mathcal{X} \to Bool$, where $\mathcal{X}$ is a direct product of finite sorts and $\mathcal{Z}$ is an arbitrary sort. The (sub)vocabulary of abstract functions and predicates will be denoted V.

A vocabulary W being fixed, the notion of *term* and that of *formula* over W are defined in a usual way.

Semantics of FOTL.

A priori we impose no constraints on the admissible interpretations. Thus, the notions of interpretation, model, satisfiability and validity are treated as in first order predicate logic modulo the preinterpreted part of the vocabulary. Thus $\mathcal{M} \models F$, $\mathcal{M} \not\models F$ and $\models F$ where $\mathcal{M}$ is an interpretation and F is a formula, denote respectively that $\mathcal{M}$ is a model of F, $\mathcal{M}$ is a counter-model of F and F is valid.

Remark that an interpretation f^* of a function f of type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ describes a family of temporal processes with value in $\mathcal{Z}$ parametrized by the elements of the interpretation $\mathcal{X}^*$ of $\mathcal{X}$.

Notations 2

• *f_x, where $f : \mathcal{T} \times \mathcal{X} \to \mathcal{Z}$, stands for $\lambda t f(t, x)$, i. e. for the function obtained from f for a fixed x.*

• *σ^- and σ^+ denote respectively the left and the right ends of interval σ.*

2.2 Vocabulary of algorithm versus verification vocabulary.

Though we do not consider here how to represent runs in FOTL (see [BS02b] on this subject), the following Example 2 gives some hints on the relation of vocabularies used to specify algorithms and vocabularies used for requirements and verification.

Example 2. Consider the following algorithm in a self-explanatory notation (in fact, it is a 'basic' Gurevich ASM [Gur95,Gur00]). This algorithm consists of a set $\mathbb{P}$ of non interacting processes. Each process outputs time moments at which it detects a non zero input.

Vocabulary of the algorithm.

Sorts: $\mathcal{T}$, $\mathbb{R}$, $Bool$, $\mathbb{P}$.

Input functions: $Inp : \mathbb{P} \to \mathbb{R}$, $CT :\to \mathcal{T}$, (current time) a predefined function.

Output function: $Out : \mathbb{P} \to \mathcal{T}$.

Proper internal function: $Flag : \mathbb{P} \to Bool$

Initial values (at time moment 0): $Flag(p) = \mathbf{true}$, $Inp(p) = 0$, $Out(p) = 0$.

Repeat
 ForAll $p \in \mathbb{P}$ **InParallelDo**
  If $Flag(p) \wedge Inp(p) = 0$ **Then** $Flag(p) := \mathbf{false} \parallel Out(p) := 0$ **EndIf** $\parallel$
  If $\neg\, Flag(p) \wedge Inp(p) \neq 0$ **Then** $Flag(p) := \mathbf{true} \parallel Out(p) := CT$ **EndIf**
 EndForAll
EndRepeat

To describe the functioning of this algorithm in a FOTL we introduce timed versions of functions. For a function f of a type $\mathbb{P} \to \mathcal{Z}$ we introduce a function f° of type $\mathcal{T} \times \mathbb{P} \to \mathcal{Z}$. And in terms of these functions one can describe the runs of the algorithm. □

2.3 Interpretations related to the verification problem.

Here we introduce specific classes of interpretations of a finite complexity. These interpretations (representing particular runs) play a key role in our application to verification problems.

In the verification setting we distinguish three kinds of dynamic functions: external, internal and some auxiliary functions. The requirements should be finally expressed in terms of inputs/outputs and parameters (that can be treated as static inputs). Auxiliary functions may appear in the algorithm that meets the requirements, or be imposed by the user to express directly the user's vision of the system. Internal functions are computed by the algorithm and thus, strictly speaking, are described in a piecewise constant way. However, their 'physical' interpretation may be of other nature. For example, to represent a piece of linear function $a \cdot t + b$ on an interval σ we give two values a and b for the function and two values σ^- and σ^+ for the interval. And these values remain constant up to the moment when the algorithm calculates the next piece. But the 'physical' interpretation of this function that may be used in guards of the algorithm is not constant — however, it is described as a term of the vocabulary.

We incorporate these considerations in the following system of notions.

We assume that for every abstract function f of type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ there is fixed a term U_f with values of type $\mathcal{Z}$ constructed only from constants, variables and predefined functions. We may admit a fixed number of such terms U_f for a given function (as we do in [BS02b]) — the reasoning remains the same with minor technical changes.

The vocabulary of FOTL does not give many possibilities to construct U_f. We will consider the following types of terms: first, those of the form z with z being a variable for an abstract sort (representing abstract constants of the type $\mathcal{Z}$) if $\mathcal{Z}$ is an abstract sort, and second, the terms of the form $\xi_0 t + \xi_1 a + z$, where $\xi_0, \xi_1 \in \mathbb{Q}$ and t, a and z are real variables whose role is defined as follows: t is the time variable standing for the time argument, a is the left end of the interval on which we consider our function, and z is a real parameter. We cannot make ξ_0 and ξ_1 variables because the inclusion of this sort $\mathbb{Q}$ to our vocabulary destroys the decidability we wish to ensure.

In a more general way we can permit that the parameters (ξ_0, ξ_1) can be chosen from a finite set $\Xi_f \subset \mathbb{Q}^2$, see [BS99,BS02b], or add the second end of the interval in the expression for U_f; in the former case, U_f is a finite set of terms instead of just one term.

We will write U_f also as $U_f(t, a, z)$ to make explicit the parameters. Let ζ be an interval. We say that *f_x is U_f-defined on ζ with parameter z_0*, if for $t \in \zeta$ the value $f_x(t)$ is defined as $f_x(t) = U_f(t, \zeta^-, z_0)$ (we use Notations 2).

Define also U_{id} as t, $U_P(t)$ for $P \in V_{Pred}$ as b, where b is a Boolean variable,

and thus, U_f is attributed to every $f \in V$.

A *partition* of $\mathcal{T}$ is a sequence $\pi = (\zeta_i)_{i\in\overline{N}}$ of non empty disjoint intervals where: (1) $\overline{N}$ is a prefix of $\mathbb{N}$, (2) $\bigcup_{i\in\overline{N}} \zeta_i = \mathcal{T}$, (3) $\zeta_i^+ = \zeta_{i+1}^-$ for $0 \leq i \leq |\overline{N}| - 1$, (4) $\zeta_0^- = 0$, $\zeta_k^+ = \infty$ if $\overline{N}$ is finite and k is its last element.

In the logic introduced above one can describe rather directly (see [BS02b,BCS00]) the runs of basic Gurevich Abstract State Machines [BS02b] or while-programs, transforming the latter into basic Abstract State Machines and applying the transformation from [BS02b,BCS00].

3 Finiteness Properties

We consider a First Order Timed Logic (FOTL) which is an extension of the theory of reals $\mathbb{R}$ with order, addition and unary multiplications by rationals $\mathbb{Q}$ that are considered as constants in this theory. Such an extension is defined by abstract sorts and abstract functions f of the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$, where $\mathcal{T} =_{df} \mathbb{R}_{\geq 0}$ is a sort of time treated as a subsort of $\mathbb{R}$, and $\mathcal{Z}$ is either $\mathcal{T}$ or an abstract sort. This abstract sort is finite but of unknown cardinality that can be arbitrary. We assume that there can be also predefined finite sorts (thus of known cardinality).

For technical simplicity we *assume* that in the type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ of an abstract function (more general case was considered in [BS02b])

> *$\mathcal{X}$ is one sort, not a direct product,*
> *though this sort may depend on the function.*

3.1 Repetitive interpretations

Let f be an abstract function of type $\mathcal{T} \times \mathcal{X} \to \mathcal{Z}$ and $\mathcal{X}^*$ be an interpretation of $\mathcal{X}$. For $x^* \in \mathcal{X}^*$ a *(finite) partial interpretation ((F)PI)* $f^*_{x^*}$ of f_{x^*} is given by a (finite) set of disjoint intervals, that will be called the *support* of the (F)PI, and by the values of parameters that are to be put into U_f to define f_{x^*} on each such interval.

A FPI has *complexity* k if the number of intervals is k.

A *partial interpretation (PI)* of f is a subset $\mathcal{Y}^*$ of $\mathcal{X}^*$ and a collection of PIs, one for each f_{y^*}, $y^* \in \mathcal{Y}^*$.

Let $\mathcal{X}^*$ be an interpretation of $\mathcal{X}$ for each abstract sort $\mathcal{X}$. A *partial interpretation (PI)* of V is a collection of PIs, one for each abstract function.

A partial interpretation $\mathcal{M}'$ of a function f_{x^*} is an *extension* of a partial interpretation $\mathcal{M}$ of f_{x^*} if every interval of $\mathcal{M}$ is contained in an interval of $\mathcal{M}'$, and the restriction of $\mathcal{M}'$ on intervals of $\mathcal{M}$ gives $\mathcal{M}$. In a similar way we define an *extension* of a PI of V or W.

In addition to usual finite interpretations we consider ultimately repetitive interpretations and some more general finitely definable interpretations.

An interpretation $\mathcal{M}$ of f_{x^*} is *ultimately repetitive with complexity c* and *period* h if it is a finite interpretation with complexity c or is a concatenation of a finite interpretation of complexity c on some interval, say $[0, h_0)$, followed by an interpretation of the following 'almost periodic' structure: any interval $I_i = [h_0 + i \cdot h, h_0 + (i+1) \cdot h)$, $i \geq 0$, is partitioned into c consecutive intervals $\zeta_{i,j}$, $0 \leq j \leq (c-1)$, with periodic structure of lengths: $|\zeta_{i,j}| = |\zeta_{i+1,j}|$ and such that on each $\zeta_{i,j}$ the function f_{x^*} is defined by $U_f(t, \bar{\zeta_{i,j}}, z_j)$ with the same z_j for a fixed j and for all i. The intervals $\zeta_{i,j}$ will be called *defining intervals* of this ultimately repetitive interpretation.

Chains of Repetitive Interpretations.

A finite prefix of an ultimately repetitive interpretation is *exact* if its right end coincides with the right end of one of its defining intervals I_i. We say that an interpretation is a *chain of ultimately repetitive interpretations with complexity* (L, c) if it is a concatenation of at most $(L-1)$ finite exact prefixes of repetitive interpretations and of one infinite ultimately repetitive interpretation, each of complexity c.

Now the complexity of a chain of ultimately repetitive interpretations is a pair of numbers (L, c). We are going to add more components to the complexity measure.

Remark 2. The periods of repetitive interpretations that constitute a chain may be different. However, in the context of decidability they must be given modulo a same, maybe unknown, multiplicative parameter. For example, the periods may be said to be $2 \cdot h$, $\frac{17}{5} \cdot h$ etc. with h being an unknown parameter (implicitly bound by existential quantifier). This information is presumed to be extractable from the specifications under consideration, and may look as "if there is a counter-model then it is a chain of repetitive interpretations with periods ...". A particular case is one unknown period.

Equivalence.

To reduce the complexity of interpretations in spite of a possibly large amount of elements in abstract sorts we introduce a notio n of equivalence of interpretations, and on its basis will generalize the complexity measures for PI of individual f_{x^*}. Such an equivalence is defined over elements of the interpretation of abstract sorts for each f.

Without loss of generality, an abstract sort $\mathcal{X}$ is interpreted as an initial segment $\mathcal{X}^*$ on natural numbers. Let $\mathcal{Y}^* \subset \mathcal{X}^*$. An equivalence E over $\mathcal{Y}^*$ is *interval-wise* if its classes are intervals. An equivalence E over $\mathcal{Y}^*$ is *f-compatible* if $y_1 E y_2$ implies that the functions $\lambda t f^*(t, y_1^*)$ and $\lambda t f^*(t, y_2^*)$ are equal, i. e. have the same support and have the same values on each interval of the support.

Complexity of finite partial interpretations

A partial interpretation of f over $\mathcal{Y}^* \subset \mathcal{X}^*$ is a finite partial interpretation (FPI) of *complexity* (m, c) if there is an interval-wise equivalence E on $\mathcal{Y}^*$ with at most m classes which is f-compatible and such that each $f_{y^*}^*$ with $y^* \in \mathcal{Y}^*$ has complexity c. (If a function depends on a direct product of sorts,that we excluded, then we can use the same framework for rectangular-wise equivalences like we do in [BS02b].)

A finite partial interpretation (FPI) of V of complexity (m, c) is a collection of FPIs with complexity (m, c), one for each abstract function. A FPI of complexity (m, c) will be called a (m, c)-PI.

Complexity of interpretations

An interpretation of f over $\mathcal{X}^*$ is a *finite interpretation with complexity* (m, c) if there is an interval-wise equivalence E on $\mathcal{X}^*$ with at most m classes $X_1, \ldots, X_m$ such that

- E is f-compatible,

- for each class X_i, there is a partition of time $\mathcal{T}$ into c intervals $\xi_1^i, \ldots, \xi_c^i$ such that for each $x^* \in X_i$ and $t \in \xi_j^i$ we have $f_{x^*}^*(t) = U_f(t, \xi_j^{i-}, z_j^i)$.

An interpretation of f over $\mathcal{X}^*$ is *ultimately repetitive with complexity* (m, c) if there is an interval-wise equivalence E on $\mathcal{X}^*$ with at most m classes which

is f-compatible and such that for each class, all the $f^*_{x^*}$ for x^* in this class are ultimately repetitive with complexity c, with the same partition of time and the same parameters of U_f.

An interpretation of f over $\mathcal{X}^*$ is *a chain of ultimately repetitive interpretations with complexity* (m, L, c) if there is an interval-wise equivalence E on $\mathcal{X}^*$ with at most m classes, which is f-compatible and such that for any class, each $f^*_{x^*}$ for x^* in the class is a chain of ultimately repetitive interpretations with complexity (L, c)), with the same partition of time and the same parameters of U_f.

We will define the *finiteness* properties in terms of FPI contained in models or counter-models $\mathcal{M}$ of the formulas under consideration.

As we are interested only in interpretations with a finite complexity we introduce here the appropriate class of interpretations.

A PI with complexity $\mathcal{K}$ will be called a $\mathcal{K}$-PI.

Notations 3.

• For a class $\mathcal{C}$ of interpretations we denote by $\mathcal{C}(\mathcal{K})$ the set of interpretations in the class $\mathcal{C}$ with complexity $\mathcal{K}$.

• $\mathcal{UR}$ is the class of ultimately repetitive interpretations.

• $\mathcal{UR}^*$ is the class of chains of ultimately repetitive interpretations.

• $\mathcal{UR}^*(\mathcal{K}, \Lambda)$, where $\Lambda \subset \mathbb{Q}_{>0}$, is the set of interpretations from $\mathcal{UR}^*$ with complexity $\mathcal{K}$ (of the form (m, L, c)) whose period lengths are from Λ. (Recall that for a given ultimately repetitive interpretation $f^*_{x^*}$, the period length is fixed, so the set Λ specifies possible period lengths for interpretation of different functions f_x.)

• $\mathcal{UR}^*(\Lambda)$ is the union of all $\mathcal{UR}^*(\mathcal{K}, \Lambda)$ over $\mathcal{K}$.

3.2 Finite Refutability and Finite Satisfiability

Let α be a total computable function transforming a complexity value of the form (m, c) into a complexity value of the form (m, c) or of the form (m, L, c). Below, $\mathcal{K}$ is a complexity of the form (m, c).

A formula G is $\mathcal{K}$-*refutable* if for every its counter-model $\mathcal{M}$ there exists a $\mathcal{K}$-FPI $\mathcal{M}'$ such that $\mathcal{M}$ is an extension of $\mathcal{M}'$ and any extension of $\mathcal{M}'$ to a

total interpretation is a counter-model of G.

Speaking informally, finite refutability (with a given complexity) of a formula means that any counter-model of this formula contains a piece of this complexity that concentrates all the contradictions that determine the fact that the interpretation is a counter-model; so any extension of this piece will be again a counter-model.

Finite satisfiability, defined just below, is a notion that is, in some way, dual to finite refutability. If in a *model* we take any piece of a given complexity (imagine that this piece is defined on some amount of separated intervals) then we can fill the gaps between defined parts by pieces of total complexity that is bounded as function of the complexity of the given initial piece. This bound is determined by the augmentation function.

A formula G is $(\mathcal{C}, \mathcal{K})$-*satisfiable with augmentation* α if for every $\mathcal{K}$-FPI $\mathcal{M}$ extendable to a model of G there is an extension $\mathcal{M}'$ of $\mathcal{M}$ from $\mathcal{C}(\alpha(\mathcal{K}))$ that is a model of G.

A formula G is $\mathcal{C}$-*satisfiable with augmentation* α iff for every $\mathcal{K}$, for every $\mathcal{K}$-FPI $\mathcal{M}$ extendable to a model of G, there is an extension $\mathcal{M}'$ of $\mathcal{M}$ from $\mathcal{C}(\alpha(\mathcal{K}))$ that is a model of G.

Remark that we can speak not about finite refutability or satisfiability of formulas but about that of sets of interpretations: for refutability about of a set that corresponds to the set of counter-models, and for satisfiability about a set that corresponds to the set of models.

Example 3. $\mathcal{UR}^*$-satisfiable but not $\mathcal{UR}$-satisfiable formula.
Consider the set of runs of the timed automaton in Figure 1. The state of the automaton at time t is denoted by $loc(t)$. The clocks are x and y. The condition $y = 2$, $\{y\}$ on the edge from s_2 to s_1 means that this transition is fired when the clock y arrives at 2, and after that it is reset to 0. The other conditions are understood in a similar way.

Fig. 1. An automaton whose some runs are chains of repetitive interpretations.

The set of runs of this automaton is described by some formula, but we will speak about the set itself. Remark that we do not have abstract sorts of unknown cardinality, so there is no need to care about equivalences over abstract sorts. So we will measure complexity by one value c for $\mathcal{UR}$ and by two values (L, c) for $\mathcal{UR}^*$.

We claim that this set of runs is not $\mathcal{UR}$-satisfiable by ultimately repetitive

interpretations. Suppose that it is $\mathcal{UR}$-satisfiable with augmentation α for some complexity c_0, and $\alpha(c_0) = N_0$. Take a model $\mathcal{M}$ such that at moment $t_0 = N_0 + \frac{1}{2}$ we have $loc(t_0) = s_1$, $x(t_0) = \frac{1}{2}$ and $y(t_0) = 0$. The 1-PI with support t_0 can be extended to a model, but any such extension has complexity at least $N_0 + 1$.

In return, one can prove that this set of runs is $\mathcal{UR}^*$-satisfiable with identical augmentation (one can say 'without augmentation') and with $\Lambda = \{1, 2\}$. (Recall that Λ is the set of possible values of the periods length.) □

Remark 3. In [BS02a] we used more general notions of finite refutability and finite satisfiability where the involved models were considered up to some equivalence over values of interpretation at fixed time moments. A timed automaton is reducible with a threshold L if any its run having more than L changes of states can be replaced by an equivalent run having not more than L changes. In [BS02a] we proved that the formula representing the runs of a reducible timed automata is $\mathcal{F}$-satisfiable, and the reducibility is decidable. The notions of this paper can be also extended in this way.

Remark 4. Finite refutability of properties of functioning of practical real time systems often (maybe almost always) takes place. For example, the safety property is usually finitely refutable. As for liveness, too general formulations can be not finitely refutable. For example, if we consider liveness for the mutual exclusion with unbounded waiting time, it will not be finitely refutable. But in any practical system the waiting time is always bounded. If we add such a bound the liveness becomes finitely refutable. And this puts to the fore a general principle: adding practical bounds we arrive at finiteness properties. See [BS02b] for detailed examples.

Remark 5. Intuitively, finite satisfiability of an algorithm means that every its run is reducible in the following sense: every interval of the run can be replaced by a piece with the same end states and of bounded complexity with respect to the class $\mathcal{C}$ under consideration. Many control algorithms possess this property.

4 Decidable Class of the Verification Problem

Class $VERIF(\Lambda, \mathcal{K}, \alpha)$ of verification problems.

We will use the notation $\mathcal{C}_h =_{df} \mathcal{UR}^*(h \cdot \Lambda)$ in our description of the class of decidable verification problems . Here $h \cdot \Lambda$, where h is a real and Λ is a *finite* set of rational numbers, is the set of reals of the form $h \cdot \lambda$ with $\lambda \in \Lambda$.

Denote by $VERIF(\Lambda, \mathcal{K}, \alpha)$ the class of FOTL-formulas of the form $(\Phi \to \Psi)$ such that for some $h \in \mathbb{R}_{>0}$ the formula Ψ is $\mathcal{K}$-refutable and Φ is $(\mathcal{C}_h, \mathcal{K})$-satisfiable with augmentation α. So in our description of the decidable class there is an unknown parameter h.

Recall that U_f is fixed.

Our main goal is to prove a theorem about decidability of the class $VERIF(\Lambda, \mathcal{K}, \alpha)$. The following result by V. Weispfenning is needed not only for the proof but also for a enhanced form of the decidability.

V. Weispfenning's Quantifier Elimination Theorem.

In [Wei99] V. Weispfenning gives a quantifier elimination for theory L'' with *mixed* variables, namely variables over reals and variables over integers. The vocabulary of L'' consists of two just mentioned sorts: reals $\mathbb{R}$ and integers $\mathbb{Z} \subset \mathbb{R}$, rational numbers $\mathbb{Q}$ as constants, (binary) addition, scalar (unary) multiplication by rational numbers, integer part $\lfloor\ \rfloor$, congruences $\equiv_n$ modulo concrete natural numbers n. We consider the vocabulary without congruences as the latter can be eliminated, see [Wei99]. The first part of the Corollary 3.4 of [Wei99] says the following:

Lemma 4 *There is an algorithm assigning to a given L''-formula Φ a quantifier-free L''-formula that is equivalent to Φ.*

The theory L'' is used to formulate **our main result**:

Theorem 5 *Given a complexity $\mathcal{K} = (m, c)$, a computable augmentation function α and a finite set $\Lambda \subset \mathbb{Q}_{>0}$, the validity of formulas from $VERIF(\Lambda, \mathcal{K}, \alpha)$ is decidable. Moreover, if a formula of this class is false then its counter-models of complexity $\alpha(\mathcal{K})$ can be described by a quantifier-free L''-formula (see also Theorem 9 for some precisions concerning the role of h in this description).*

This Theorem 5 will be proved in this section, and the proof uses Lemma 4.

4.1 Some reductions

The initial observation to describe the decidable classes is the following one:

Proposition 6 *Suppose that the existence of h and of a counter-model of a fixed complexity $\mathcal{K}$ from $\mathcal{C}_h$ is decidable for closed FOTL-formulas. Given $\mathcal{K} = (m, c)$, the validity of formulas from $VERIF(\Lambda, \mathcal{K}, \alpha)$ is decidable: such a*

formula has a counter-model if and only if it has a counter-model of complexity $\alpha(\mathcal{K})$ in $\mathcal{C}_h$ for some h.

Proof. Suppose that a formula $F = (\Phi \rightarrow \Psi)$ from $VERIF(\Lambda, \mathcal{K}, \alpha)$ has a counter model $\mathcal{M}$. This $\mathcal{M}$ is a counter-model of Ψ and a model of Φ. $\mathcal{K}$-refutability of Ψ means that there is a restriction $\mathcal{M}_1$ of complexity $\mathcal{K}$ of $\mathcal{M}$ whose all extensions are counter-models of Ψ. The premise Φ is $(\mathcal{C}_h, \mathcal{K})$-satisfiable with augmentation α for some h. Thus, $\mathcal{M}_1$ can be extended to a model of Φ with complexity $\alpha(\mathcal{K})$. This extension remains a counter-model of Ψ, and hence it is a counter-model of F. ■

Thus the problem is reduced to the decidability of the existence of a (counter-)model of a given complexity in our class of models, that can be stated as

Proposition 7 *Given a finite set Λ of lengths of periods and a complexity $\mathcal{K}$, the existence of h and of a (counter-)model from $\mathcal{C}_h$ of complexity $\mathcal{K}$ is decidable for FOTL-formulas.*

4.2 Elimination of abstract functions

To avoid minor but tedious technical difficulties we will prove Proposition 7 for $\mathcal{UR}$-models. It will be clear that the case of $\mathcal{UR}^*$-models needs only one more index for existential variables in order to represent the number L of enchained ultimately repetitive models.

Let G be a closed FOTL-formula and $\mathcal{K}$ be a bound on the complexity of ultimately repetitive models. We wish to decide whether there exists such a model of G with the given bound on complexity. We can decide it for finite models — this was done in [BS02b]. If this procedure gives a negative answer for finite models we try the procedure to check the existence of ultimately repetitive models with a non trivial repetitive part.

Simplification of atomic formulas.

We make some trivial simplifications to make easier the presentation of the decision procedure.

Predicates will be treated as functions with Boolean values.

We replace time variables by real variables.

By adding additional (quantified) variables we can reduce all atomic formulas to arithmetic (in)equalities or to equalities of the forms respectively

• $a_1 \cdot t_1 + \cdots + a_n \cdot t_n \ \omega \ 0$, where $a_1, \ldots, a_n \in \mathbb{Q}$, $t_1, \ldots, t_n$ are real variables and ω is an arithmetic relation $(=, <, \leq, \ldots)$;

• $u = f(x, v)$ where f is an abstract function and x, u and v are variables or constants of respective types.

The transformations to apply to achieve these forms are standard ones, for example,

$$\tau = a_1 \cdot \eta_1 + \cdots + a_m \cdot \eta_m \ \leftrightarrow$$

$$\forall \tau'_1 \ldots \forall \tau'_m \, (\textstyle\bigwedge_{i=1}^{m} \tau'_i = \eta_i \ \rightarrow \ \tau = a_1 \cdot \tau'_1 + \cdots + a_m \cdot \tau'_m),$$

where η_i are terms, t_i are variables and $a_1, \ldots, a_n \in \mathbb{Q}$.

Description of partitions.

The given complexity $\mathcal{K}$ has the form (m, c), where m is the number of equivalence classes of $\mathcal{X}$, c is the number of time intervals to consider. These intervals can be different for different functions and even for different classes of equivalence of the same function. Concerning the lengths of repetitive parts we know that they are of the form $h \cdot \lambda$ with $\lambda \in \Lambda$.

Remark 6. Our method works also for the case when all the lengths are known rational numbers (corresponds to $h = 1$) — this case is simpler, but the method does not work for the both possibilities, that is for the case when some lengths are known and the others are rational multiples of h. Actually, in this last case, our transformation leads to a formula which contains a product of a real variable and an integer one. □

We describe the existence of a repetitive model with the mentioned complexity, defined now by two numbers m (the number of equivalence classes) and c (the number of intervals where the function has a fixed definition), in a theory that extends a theory L'' of mixed addition mentioned above. This existence is described by a formula that states the existence of certain intervals and partitions and expresses atomic formulas with abstract symbols in arithmetical terms. For technical simplicity we assume below that all the intervals we consider are of the form $[a, b)$ with $a < b$. The general case obliges to introduce Boolean variables to make precise for each end whether it belongs to the interval or not.

The formula that states the existence of a repetitive model of a given complexity follows the following lines ("ERM" comes from "Existence of a Repetitive Model"):

(ERM1) There exists a positive real h (that defines $\mathcal{C}_h$).

(ERM2) For each sort $\mathcal{X}$ there exists its interpretation as an initial interval $[0, M_{\mathcal{X}} - 1]$ of $\mathbb{N}$, $M_{\mathcal{X}} \geq 1$.

(ERM3) For any abstract function $f : \mathcal{T} \times \mathcal{X}_f \to \mathcal{Z}_f$ there exist natural numbers

$$M_{f,0} = 0 \leq M_{f,1} \leq \cdots \leq M_{f,m-1} \leq M_{f,m} = M_{\mathcal{X}_f} - 1$$

that give a partition of the interpretation of $\mathcal{X}_f$ into m classes. The k-th class (maybe empty) is the set of naturals $n \in [M_{f,k-1}, M_{f,k})$, for $k = 1, \ldots, m-1$, and the last class is the set of naturals $n \in [M_{f,m-1}, M_{f,m}]$(remark that this partition may say that the number of classes is not greater than m, i. e. the classes are not necessarily non empty).

(ERM4) The length $h_{f,k}$ of repetitive intervals for f and for the k-th class of its equivalence is of the form $h \cdot \lambda$ for $\lambda \in \Lambda$. This can be expressed in terms of the existence of $h_{f,k}$ and of an appropriate disjunction over Λ.

(ERM5) For any abstract function f and for each class of equivalence (this f and the index of the equivalence class are just indices for the variables: one index is f and the other is the index $k \in \{0, 1, \ldots, m-1\}$ of the equivalence class)

(ERM5.1) there exists an initial interval $[0, H_{f,k})$, there exists its partition into c subintervals defined by points

$$H_{f,k,0} = 0 < H_{f,k,1} < \cdots < H_{f,k,c-1} < H_{f,k,c} = H_{f,k}$$

and there exists a list $(Z_{f,k,j})_{0 \leq j < c}$ of parameters of U_f, different from the beginning of intervals that define f_x, where x is in the k-th equivalence class, on each of these intervals.

(ERM5.2) There exists a partition of $[0, h_{f,k})$ into c intervals defined by points

$$0 < h_{f,k,1} < \cdots < h_{f,k,c-1} < h_{f,k,c} = h_{f,k}$$

and there exists a list $z_{f,k,j}$ of parameters of U_f, different from the beginning of intervals such that U_f with the respective parameters and the beginning of the intervals

$$\alpha_{f,k,i,j} =_{df} [H_{f,k} + i \cdot h_{f,k} + h_{f,k,j}, H_{f,k} + i \cdot h_{f,k} + h_{f,k,j+1})$$

defines f_x, where x is in the respective equivalence class, on this interval for all $i \geq 0$.

(ERM6) The variables introduced above permit to represent abstract functions in terms of U_f and thus, to eliminate them — see below.

Remark 7. To have different forms of functions for different equivalence classes and for different intervals one can consider not a single pair (ξ_0, ξ_1) to define U_f but a finite set of such pairs. This generalization is easily treatable in the same framework by adding appropriate disjunctions (that are 'finite existential quantifiers') over these pairs. □

Arithmetical description of abstract functions.

Recall that the atomic formulas are of the form $f(x,t) = v$, where x is a variable for $\mathcal{X}$, t is a variable for $\mathbb{R}$ (time was eliminated) and v is a variable for $\mathbb{R}$, $\mathcal{X}$ or pre-interpreted abstract sort, like *Bool*. Remark that any of x or t can be dummy. On the other hand, there is no need to consider formulas where x is a pre-defined abstract sort. Actually if the latter is the case then this sort is of known cardinality, say, κ, and we can replace $f(x,.)$ by κ functions. However, v of a pre-defined sort is necessary to treat predicates. For predicates, *Bool* will be represented, as any other abstract sort, by an initial segment of natural numbers, namely, *Bool* becomes $\{0,1\}$.

The elimination of abstract functions is done as follows. Replace in the given formula G each occurrence of atomic formula $f(t,x) = v$ by (though the symbols with indices used below are just variables, their role was explained above in (ERM2)–(ERM5), in particular intervals α were mentioned in (ERM5.2))

$$\bigwedge_{0\le k<m}\ \bigwedge_{0\le j<s}\Big[\ \Big(\ M_{f,k} \le x < M_{f,k+1}\ \wedge\ t \in [H_{f,k,j}, H_{f,k,j+1})\ \Big)$$

$$\rightarrow\ U_f(t, H_{f,k,j}, Z_{f,k,j}) = v\ \Big]$$

$$\wedge\ \bigwedge_{0\le k<m}\ \bigwedge_{0\le j<s}\ \forall\, i\ \Big[\ \Big(\ M_{f,k} \le x < M_{f,k+1}\ \wedge\ t \in \alpha_{f,k,i,j}\ \Big)$$

$$\rightarrow\ U_f(t, \alpha^{-}_{f,k,i,j}, z_{f,k,j}) = v\ \Big] \qquad (3)$$

Notations 4

• $\widetilde{G}$ is the formula obtained from G after the transformations of atomic formulas according to (3);

• Π is the list of all variables mentioned above in (ERM2)–(ERM5) except h, i. e. $M_{f,k}$, $H_{f,k,j}$, $h_{f,k,j}$, $Z_{f,k,j}$ and $z_{f,k,j}$ for $0 \le k < m$, $0 \le j < s$ and $f \in V$;

• B is a conjunction of the inequalities mentioned above in (ERM2)–(ERM5);

• G_0 is the formula $\exists\, h\, \exists\, \Pi\, \widetilde{G}_0$ where $\widetilde{G}_0 =_{df} (B\ \wedge\ \widetilde{G})$.

Proposition 8 *A closed FOTL formula G has an ultimately repetitive model of complexity $K = (m,s)$ if and only if G_0 is valid (interpreted over reals and integers with their usual relations, addition and multiplication).*

Proof. The proof is just a verification that the elimination of abstract sorts and functions is correct. Intuitively it is clear; a detailed proof may follow the lines of the proof of Lemma 3 from [BS02b]. ■

The formula G_0 is not a L''-formula because of subformulas $t \in \alpha_{f,k,i,j}$ and some of subformulas $U_f(t, \alpha^-_{f,k,i,j}, z_{f,k,j}) = v$ in (3); these subformulas have mixed binary multiplications:

$$H_{f,k} + i \cdot h_{f,k} \cdot h + h_{f,k,j} \leq t < H_{f,k} + i \cdot h_{f,k} \cdot h + h_{f,k,j+1},$$
$$\xi_0 \cdot t + \xi_1 \cdot (H_{f,k} + i \cdot h_{f,k} \cdot h + h_{f,k,j}) + z = v \tag{4}$$

where i is an integer variable and h is a real one (recall that $a_{f,k}$, ξ_0, ξ_1 are concrete rational numbers, symbols $H_{f,k}$, $h_{f,k,j}$ and t stand for real variables and z and v may be real variables or rational constants).

All the other atoms are of the form

$$u = v \tag{5}$$

with u and v being natural numbers or constants representing elements of abstract sorts , or of the form

$$a_1 \cdot z_1 + \cdots + a_n \cdot z_n \; \omega \; c \tag{6}$$

with $a_i, c \in \mathbb{Q}$ and z_j being real variables and $\omega \in \{=, <, >, \leq, \geq\}$. Remark that t from (4), as well as h, may be among z_j of (6). On the other hand, the natural number variables from (5) are not involved in any arithmetical terms and do not mix with variables for reals or with i from (4).

Divide all terms in inequalities (4) and (6) by h. Underline that $h > 0$ and is common for the whole formula and quantified by the most exterior existential quantifier. The bijection $z \leftrightarrow \frac{z}{h}$ preserves the order relations and commutes with the operations over reals that we use.

Replace expressions $\frac{z}{h}$, where z is a variable, by new variables to get a formula $G_1 =_{df} \exists\, h \exists\, \Pi\, \widetilde{G}_1$ (we may, in fact, use the old real variable names, that is we may replace $\frac{z}{h}$ by z). This formula G_1 is valid iff G_0 is valid, and G_1 has atoms of the form (5) and of the form (we use the old notations for real variables divided by h):

$$a \cdot i + a_1 \cdot z_1 + a_2 \cdot z_2 + \cdots + \frac{c}{h} \; \omega \; 0, \tag{7}$$

where a, a_1, $a_2, \ldots, c \in \mathbb{Q}$, i is an integer variable and the other symbols stand for real variables.

Now replace in $\widetilde{G}_1$ each $\frac{1}{h}$ by a new variable that we will, however denote by

the same letter h. Denote the obtained formula by $\widetilde{G}_2$. This $\widetilde{G}_2$ is a L''-formula. The whole formula G_1 is equivalent to formula G_2 that can be represented as

$$\exists\, h\, \exists \Pi\ \widetilde{G}_2. \tag{8}$$

If to eliminate quantifiers in $\widetilde{G}_2$ (Lemma 4), we get a quantifier-free L''-formula describing all models of bounded complexity (s, m) and with fixed relations between period lengths of the initial formula G (we are not to forget that we have changed the 'meaning' of initial variables in the meantime). If to eliminate all quantifiers in (8) we answer the question about the existence of such a model.

Hence, we have proven the following theorem that is even stronger than Proposition 7:

Theorem 9 *Given a closed FOTL-formula G, all its repetitive models (we mean chains of ultimately repetitive interpretations) of a given complexity in $\mathcal{C}_h$ and all respective h can be represented by a quantifier-free L''-formula. In particular the existence of such models is decidable.*

Together with Proposition 6 it gives Theorem 5.

Complexity of the decision procedure.

The *complexity* of the decision procedure is determined by the complexity of Weispfenning's Quantifier Elimination and by the complexity of our reductions. The worst case complexity of Weispfenning's Quantifier Elimination is that of the decision procedure for Presburger arithmetics, i. e. $2^{l^{n^{O(a)}}}$, where l is the length of the formula (presumed to be in a prenex form), n is the number of variables and a is the number of blocks of alternating quantifiers. Our reductions add $O(\alpha(k)|V|)$ variables and (together with transforming the formula into a prenex form) augment the size of the initial formula exponentially in the general case. However, the formulas that are used to prove the worst case complexity do not appear in practice.

Conclusion

Though logics that are expressible, and thus relatively easy to use to represent verification problems directly and completely, are usually undecidable (even deductively incomplete), practical verification problems seem to be not only decidable but feasible, that is to be decidable practically efficiently. The reason

is that practical algorithms are very far from diagonal Turing machines used to prove undecidability or high lower bounds, and the requirements that we wish to prove are also rather particular and simple. So the problem is to describe their properties that ensure the decidability of the verification problem.

A general source of properties that may help to describe decidable or feasible classes are hand-made proofs. In such proofs, if to speak about practical algorithms, we implicitly give a finite description of the algorithm under consideration.

In this paper we are closer to finite automata viewpoint; this viewpoint may give some other decidable classes. To outline the limits of decidable classes we cannot avoid proving what classes that seem to be close to a decidable one are already undecidable. Such a class related to the result of this paper is the class with several unknown periods.

A more difficult and much more laborious problem is to pass from decidable classes to feasible ones. One question to elaborate is the representation of runs in logic. There are many considerations that can improve the efficiency (we started such kind of analysis in [BCS00]). Another source of improving the efficiency lies in improving quantifier elimination or, more generally, decidability procedures, taking into consideration the peculiarities of practical algorithms and properties.

The description of our decidable class is semantical. The problem of recognizing the class is undecidable (see [BS02a]). An open question is to give sufficient syntactical conditions to ensure finiteness properties. This can be done for some properties like safety (finite refutability is, in a way, a general definition of safety properties) or liveness, for the latter one with a certain care.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BCS00] D. Beauquier, T. Crolard, and A. Slissenko. A predicate logic framework for mechanical verification of real-time Gurevich Abstract State Machines: A case study with PVS. Technical Report 00–25, University Paris 12, Department of Informatics, 2000. Available at http://www.univ-paris12.fr/lacl/.

[BS99] D. Beauquier and A. Slissenko. Decidable classes of the verification problem in a timed predicate logic. In *Proc. of the 12th Intern. Symp. on*

Fundamentals of Computation Theory (FCT'99), Iasi, Rumania, August 30 –September 3, 1999, Lect. Notes in Comput. Sci, vol. 1684, pages 100–111. Springer-Verlag, 1999.

[BS02a] D. Beauquier and A. Slissenko. Decidable verification for reducible timed automata specified in a first order logic with time. *Theoretical Computer Science*, 275(1–2):347–388, 2002.

[BS02b] D. Beauquier and A. Slissenko. A first order logic for specification of timed algorithms: Basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113(1–3):13–52, 2002.

[Gur95] Y. Gurevich. Evolving algebra 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–93. Oxford University Press, 1995.

[Gur00] Y. Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.

[Sli99] A. Slissenko. Minimizing entropy of knowledge representaion. In *Proc. of the 2nd International Conf. on Computer Science and Information Technologies, August 17–22, 1999, Yerevan, Armenia*, pages 2–6. National Academy of Sciences of Armenia, 1999.

[Sli03] A. Slissenko. A logic framework for verification of timed distributed algorithms. Version of April 2001. Technical Report TR 2003–04, University Paris 12, Laboratory for Algorithmics, Complexity and Logic (LACL), 2003. Available at http://www.univ-paris12.fr/lacl/.

[Tel00] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 2nd edition, 2000.

[Wei99] V. Weispfenning. Mixed real-integer linear quantifier elimination. In *Proc. of the 1999 Int. Symp. on Symbolic and Algebraic Computations (ISSAC'99)*, pages 129–136. ACM Press, 1999.