# Automated Verification of Dense-Time MTL Specifications via Discrete-Time Approximation[⋆]

Carlo A. Furia[1], Matteo Pradella[2], and Matteo Rossi[1]

[1] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
[2] CNR IEIIT-MI, Milano, Italy
{furia, pradella, rossi}@elet.polimi.it
http://home.dei.polimi.it/lastname/

**Abstract.** This paper presents a verification technique for dense-time MTL based on discretization. The technique reduces the validity problem of MTL formulas from dense to discrete time, through the notion of *sampling invariance*, introduced in previous work [13]. Since the reduction is from an undecidable problem to a decidable one, the technique is necessarily incomplete, so it fails to provide conclusive answers for some formulas. The paper discusses this shortcoming and hints at how it can be mitigated in practice. The verification technique has been implemented on top of the $\mathbb{Z}$ot tool [19] for discrete-time bounded validity checking; the paper also reports on in-the-small experiments with the tool, which show some results that are promising in terms of performance.

**Keywords:** real-time, metric temporal logic, discretization, dense time, verification techniques, sampling

## 1 Introduction

Metric temporal logics such as MTL [18] and TRIO [5] are effective and flexible notations to model and reason about a wide range of systems — real-time, in particular — with varying level of detail. Both MTL and TRIO are parametric with respect to the temporal domain, and permit to describe systems either with a dense or a discrete notion of time [11].

Indeed, when modeling the behavior of real-time systems, the nature of the time domain plays a prominent role, and it must be carefully chosen. From a modeling viewpoint, dense time offers advantages in terms of naturalness and completeness of description, being of the same quality as "physical time", in particular when describing the composition of purely asynchronous processes (which can occur at any instant in time); also, it is usually strictly more expressive than discrete time [2]. Conversely, in practice, discrete-time models are generally more amenable to (automated) verification than dense-time ones. In fact, dense-time formalisms are often undecidable or with highly complex decidability problems [2]; in addition, while verification methods for discrete-time models can often

[⋆] Work partially supported by MIUR under the FIRB ArtDeco project.

be built upon existing techniques (e.g., for LTL, automata, and untimed formalisms), the native treatment of dense time requires novel, more ingenuous, solutions. In the literature, various techniques have been proposed to mitigate this problem. A significant category of such approaches rely on some notion of *discretization*, which consists in reducing the verification problem from dense to discrete time. Therefore, discretization techniques permit the re-use of existing techniques (and tools), but, for formalisms that are strictly more expressive in their dense-time variant, they are also necessarily *incomplete*, i.e., they fail to give conclusive results on some instances of the verification problem.

In [13], we introduced the notion of *sampling* for temporal logic formulas, an idealization of the physical sampling process. We defined ${}^{\mathbb{R}}_{\mathbb{Z}}$TRIO, a subset of the TRIO metric temporal logic interpreted over *behaviors* (i.e., total functions of time), and we identified a sufficient condition under which ${}^{\mathbb{R}}_{\mathbb{Z}}$TRIO formulas are *sampling invariant* (i.e., such that they can be interpreted consistently over dense-time behaviors and over discrete-time samplings thereof). While the results of [13] were derived for ${}^{\mathbb{R}}_{\mathbb{Z}}$TRIO, it is immediate to translate them for MTL, the reference language in this paper. Hence, in the following we always refer to MTL rather than ${}^{\mathbb{R}}_{\mathbb{Z}}$TRIO, also when citing results from [13].

In the field of formal verification, automata-based techniques have been extensively studied [6]. However, in the last few years, the increased practical efficiency of SAT solvers has rendered SAT-based verification techniques an interesting and viable alternative [3]. These are particularly well-suited in purely logical/descriptive approaches, where both the system to be analyzed and its desired properties (i.e., the entire verification problem) are expressed as temporal logic formulas. In [20], we introduced $\mathbb{Z}$ot, a SAT-based verification tool for discrete-time metric temporal logics with past operators (e.g., TRIO and MTL).

In this paper, we build upon the results of [13] and [20] to provide an effective, fully automated technique and tool for the verification of specifications written in dense-time metric temporal logic. Our contribution is twofold. First, the verification technique is introduced and proved sound. The technique relies on *two approximations* ($\phi^+$ and $\phi^-$) of the formula representing the instance of the verification problem. These approximations represent a mapping of the problem to the discrete-time domain; in other words, they encode information about the samplings of the original dense-time behaviors. Approximations are built parametrically with respect to a chosen length of the sampling period for these samplings. Then, the validity of $\phi^+$ over discrete time implies the validity of the original formula over dense time; conversely, the non-validity of $\phi^-$ over discrete time implies the non-validity of the original formula over dense time. As mentioned above, the technique must be incomplete, i.e., it may happen that the validity check of the approximations yields inconclusive results. This paper discusses how this can be mitigated in practice.

As a second contribution, we demonstrated the practical applicability of the technique by implementing it on top of the $\mathbb{Z}$ot validity checker [19], and by performing some experiments. Although limited to a small set of examples, our tests show interesting results; in particular, incompleteness is shown to be not often a practical hurdle (usually because other limitations are more significant, such as the inherent scalability even of discrete-time methods). The tests are thus a first assessment of the feasibility of our discretization techniques.

The paper is organized as follows: Section 1.1 surveys some related works on discretization techniques; Section 2 introduces the MTL subset considered in this paper and recalls the notions of sampling (and sampling invariance) from [13]; Section 3 presents the discretization technique itself; Section 4 briefly describes the implementation and reports on the experiments carried out; finally, Section 5 concludes. For lack of space, we omit some proofs, a few technical details, and several experimental results; we refer the interested reader to [12].

## 1.1 Related Works

The problem of reducing the dense-time verification problem to the discrete-time one was first explicitly studied in the seminal paper by Henzinger, Manna, and Pnueli [16]. Their discretization techniques are based on the notion of *digitization*; a (semantic) property (i.e., a set of timed state sequences) is digitizable if it is both closed under digitization and closed under inverse digitization. Basically, a property is closed under digitization if all the timed state sequences obtained by digitizing the real-timed state sequences are also integer-timed state sequences of the property; conversely, a property is closed under inverse digitization if all its integer-timed state sequences can be obtained by digitizing some real-timed state sequences of the property. The digitization of a timed state sequence is built by considering all possible roundings, with respect to any threshold $0 \leq \epsilon < 1$, of the timestamps in the timed state sequence. Note that the timestamps are *weakly monotonic*, so that more than one state value can share the same timestamp.

The comparison between the notion of digitization and the notion of sampling invariance — to be recalled in Section 2.2 — shows three main differences (see [10] for details). First, digitization assumes weakly monotonic timed words as semantic models, whereas sampling invariance considers (strongly monotonic) interval-based behaviors; each of these models has its own advantages and disadvantages [17]. Second, it has been shown [10] that the sets of MTL properties that are digitizable and sampling invariant are incomparable, i.e., there are digitizable properties that are not sampling invariant and sampling invariant properties that are not digitizable; this suggests that discretization techniques based on these two notions are likely to have different domains of applicability. Third, whereas sampling invariance is a syntactic property (i.e., it is defined for MTL formulas), digitizability is a semantic notion (i.e., it is defined for sets of timed words); as a consequence it is straightforward to characterize a significant subset of the MTL language whose formulas are sampling invariant, whereas doing the same with respect to digitizability is considerably more complicated [4].

Many subsequent works have applied the notion of digitization of [16], or other notions of discretization, to specific formalisms. In the remainder of this section we briefly report on a few of them, referring to [12] for more examples.

Chakravorty and Pandya [4] apply the notion of digitization to Interval Duration Logic (IDL), a variant of duration calculus where formulas are interpreted over timed state sequences. Overall, they introduce a technique to reduce the validity problem for dense-time IDL formulas to that of discrete-time IDL; this is possible for all IDL formulas that are closed under inverse digitization. However, it is hard to characterize closure under inverse digitization for IDL formulas; to lessen the problem, a new notion of *strong closure under inverse digitization* (SCID) is introduced. It is much simpler to determine if a formula is SCID, and SCID formulas are also closed under inverse digitization. For formulas that are not SCID, they give approximations to stronger and weaker formulas that are SCID. Finally, the validity problem for discrete-time IDL is decidable. Using these techniques, Sharma, Pandya, and Chakravorty [21] experiment with a variety of discrete-time verification tools.

De Alfaro and Manna [7] approach the problem of discretization with reference to the temporal logic TL, a particular flavor of predicative modal logic, and to the timed trace semantics. The authors first introduce the notion of *sample invariance* (not to be confused with our notion of sampling invariance, see Section 2.2): a temporal logic is sample invariant if the formulas of the logic do not distinguish between any two timed traces for which a (sufficiently fine-grained) trace that refines both exists. Then, the notion of *finite variability* is introduced: roughly speaking, a formula $\phi$ is finitely variable if, for each timed trace, one can find a refinement (called ground trace) such that any subformula of $\phi$ has a constant truth value within any interval of the refined trace. For finitely variable formulas over ground traces, the satisfaction relation of a formula $\phi$ in the continuous semantics corresponds to that of $\Omega(\phi)$ in the discrete semantics (where $\Omega$ is a suitably defined translation function). The paper states some sufficient syntactic condition for a formula to achieve the finite variability requirement. Based on this, a methodology for continuous-time verification is proposed; it is based on refinement of continuous-time formulas to finitely-variable formulas, which can then be verified in discrete time.

Fainekos and Pappas [9] present a technique for testing specifications written in MITL (an MTL subset) against continuous-time signals by analyzing only discrete samplings of the signals. Their technique shares some underlying motivations and ideas with ours, although the two approaches have complementary scopes: our tool-supported technique provides a partial verification procedure for MTL formulas through discrete-time analysis, whereas [9] discusses practical conditions under which the continuous-time behavior of a dynamical system can be analyzed by means of its discrete-time observations.

## 2 Preliminaries

### 2.1 Specification Language: MTL

In this paper, we consider a variant of purely propositional Metric Temporal Logic (MTL, [2]) as the specification language. For brevity, we refer to this variant simply as “MTL”.

Let $\mathcal{P}$ be a finite (non-empty) set of atomic propositions, and $\mathcal{I}$ the set of all (possibly unbounded) intervals of the time domain $\mathbb{T}$ with rational endpoints.

In this paper $\mathbb{T}$ coincides with either the reals $\mathbb{R}$ (dense time) or the integers $\mathbb{Z}$ (discrete time) — or some subset thereof; we call *bi-infinite* the sets $\mathbb{R}$ and $\mathbb{Z}$, and *mono-infinite* their subsets $\mathbb{R}_{\geq 0}$ and $\mathbb{N} = \mathbb{Z}_{\geq 0}$.

*Behaviors* are total mappings $b : \mathbb{T} \to 2^{\mathcal{P}}$ that assign to every instant $t \in \mathbb{T}$ the set of propositions $b(t) \subseteq \mathcal{P}$ that are true at $t$. We denote as $\mathcal{B}_{\mathbb{T}}$ the set of all *behaviors* over $\mathbb{T}$.

**MTL syntax and semantics.**

*Syntax.* The following grammar defines the *syntax* of MTL, where $I \in \mathcal{I}$ and $\beta$ is a Boolean combination of atomic propositions, i.e., $\beta ::= \mathsf{p} \mid \neg\beta \mid \beta_1 \wedge \beta_2$ for $\mathsf{p} \in \mathcal{P}$.

$$\phi ::= \beta \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathsf{U}_I(\beta_1, \beta_2) \mid \mathsf{S}_I(\beta_1, \beta_2) \mid \mathsf{R}_I(\beta_1, \beta_2) \mid \mathsf{T}_I(\beta_1, \beta_2)$$

The basic temporal operator of MTL is the *bounded until* $\mathsf{U}_I$ (and its past counterpart *bounded since* $\mathsf{S}_I$), whose subscript $I$ denotes the interval of time over which the operator predicates. However, the results of sampling invariance, recalled in Section 2.2, as well as the discretization techniques introduced in Section 3, are easier to present when referred to MTL formulas that are in a normal form where negations are pushed down to (Boolean combinations of) atomic propositions, and no temporal operators are nested. Therefore, for the sake of simplicity, we introduced directly the MTL syntax for this normal form; hence, we also have the operators *bounded release* $\mathsf{R}_I$ and *bounded trigger* $\mathsf{T}_I$ — dual of *until* and *since*, respectively — as primitive.

Throughout the paper we omit the explicit treatment of past operators (i.e., $\mathsf{S}_I$ and $\mathsf{T}_I$) as it can be trivially derived from that of the corresponding future operators, as shown in [12]. We also assume a number of abbreviations, such as $\bot, \top, \Rightarrow, \Leftrightarrow$, and the following derived operators: $\Diamond_I(\beta) \equiv \mathsf{U}_I(\top, \beta)$, $\overleftarrow{\Diamond}_I(\beta) \equiv \mathsf{S}_I(\top, \beta)$, $\Box_I(\beta) \equiv \mathsf{R}_I(\bot, \beta)$, and $\overleftarrow{\Box}_I(\beta) \equiv \mathsf{T}_I(\bot, \beta)$.

*Semantics.* MTL *semantics* is defined over behaviors, parametrically with respect to the choice of the time domain $\mathbb{T}$.

| | | |
|---|---|---|
| $b(t) \models_{\mathbb{T}} \mathsf{p}$ | iff | $\mathsf{p} \in b(t)$ |
| $b(t) \models_{\mathbb{T}} \neg\mathsf{p}$ | iff | $\mathsf{p} \notin b(t)$ |
| $b(t) \models_{\mathbb{T}} \mathsf{U}_I(\beta_1, \beta_2)$ | iff | there exists $d \in I$ such that: $b(t+d) \models_{\mathbb{T}} \beta_2$ and, for all $u \in [0, d]$ it is $b(t+u) \models_{\mathbb{T}} \beta_1$ |
| $b(t) \models_{\mathbb{T}} \mathsf{R}_I(\beta_1, \beta_2)$ | iff | for all $d \in I$ it is: $b(t+d) \models_{\mathbb{T}} \beta_2$ or there exists a $u \in [0, d)$ such that $b(t+u) \models_{\mathbb{T}} \beta_1$ |
| $b(t) \models_{\mathbb{T}} \phi_1 \wedge \phi_2$ | iff | $b(t) \models_{\mathbb{T}} \phi_1$ and $b(t) \models_{\mathbb{T}} \phi_2$ |
| $b(t) \models_{\mathbb{T}} \phi_1 \vee \phi_2$ | iff | $b(t) \models_{\mathbb{T}} \phi_1$ or $b(t) \models_{\mathbb{T}} \phi_2$ |
| $b \models_{\mathbb{T}} \phi$ | iff | for all $t \in \mathbb{T}$: $b(t) \models_{\mathbb{T}} \phi$ |

Whenever for all $b \in \mathcal{B}_{\mathbb{T}} : b \models_{\mathbb{T}} \phi$ we say that $\phi$ is $\mathbb{T}$-valid and write $\models_{\mathbb{T}} \phi$.

We remark that a global satisfiability semantics is assumed, i.e., the satisfiability of formulas is implicitly evaluated over *all* time instants in the time domain. This permits the direct and natural expression of most common real-time specifications (e.g., time-bounded response) without resorting to nesting of temporal operators. In addition, every generic MTL formulas with nesting temporal operators can be "flattened" to the form we introduced beforehand by introducing auxiliary propositions; in other words flat MTL and full MTL are equi-satisfiable [8, 10]. Also notice that our MTL variant uses operators that are *non-strict* in their first argument, i.e., the future and past include the present instant, and the *until* and *since* operators are *matching*, i.e., they require their two arguments to hold together at some instant in $I$. Other work [14] analyzes the impact of these variants on expressiveness.

**MTL$^+$/MTL$^*$ syntax and semantics.** In order to express the discretization relations in Section 3, it is necessary to introduce some variations of the four basic temporal operators *until*, *since*, *release*, and *trigger*, denoted as $\mathsf{U}_I^\uparrow$, $\mathsf{S}_I^\uparrow$, $\mathsf{R}_I^\downarrow$, and $\mathsf{T}_I^\downarrow$, respectively. Notice that they are not part of the language in which dense-time specifications and properties are to be expressed, and they are needed only to illustrate the discretization techniques. We call "MTL$^+$" the *extension* of MTL with these operators, and "MTL$^*$" the variant where we *replace* the operators $\mathsf{U}_I$, $\mathsf{S}_I$, $\mathsf{R}_I$, $\mathsf{T}_I$ with $\mathsf{U}_I^\uparrow$, $\mathsf{S}_I^\uparrow$, $\mathsf{R}_I^\downarrow$, and $\mathsf{T}_I^\downarrow$, respectively.

Let us define the semantics of the new variants of *until* and *release*.

$b(t) \models_{\mathbb{T}} \mathsf{U}_I^\uparrow(\beta_1, \beta_2)$ iff there exists $d \in I$ such that: $b(t+d) \models_{\mathbb{T}} \beta_2$ and, for all $u \in [0, d)$ it is $b(t+u) \models_{\mathbb{T}} \beta_1$

$b(t) \models_{\mathbb{T}} \mathsf{R}_I^\downarrow(\phi_1, \phi_2)$ iff for all $d \in I$ it is: $b(t+d) \models_{\mathbb{T}} \phi_2$ or there exists a $u \in [0, d]$ such that $b(t+u) \models_{\mathbb{T}} \phi_1$

**Granularity.** For an MTL formula $\phi$, let $\mathcal{I}_\phi$ be the set of all non-null, finite interval bounds appearing in $\phi$. Given a formula $\phi$, its *granularity* $\rho_\phi$ is a pair of values $(r_\phi, R_\phi)$ where $r_\phi$ is the greatest common divisor of the numerators of the elements in $\mathcal{I}_\phi$, and $R_\phi$ is the least common multiple of the denominators of the elements in $\mathcal{I}_\phi$. For any formula $\phi$, we define $\mathcal{D}_\phi$ as the set of positive values $\delta$ such that any interval bound in $\mathcal{I}_\phi$ is an integer if divided by $\delta$; it is not difficult to show that $\mathcal{D}_\phi$ can be derived from the granularity $\rho_\phi$ as the set of all fractions $d/D$ such that: (1) $D$ is a multiple of $R_\phi$; and (2) $d$ divides $r_\phi$. Also notice that $\mathcal{D}_\phi$ has a maximum (given by $r_\phi/R_\phi$) but no minimum. $\mathcal{D}$ is generalized to sets of formulas $\Phi$ in an obvious manner.

## 2.2 Sampling Invariance

The discretization technique developed in Section 3 is based on the notion of *sampling invariance* [13]. This sub-section recalls the basic definitions and results about sampling invariance that are needed in the remainder; we refer to [13] for details. Note that, although sampling invariance results are presented in terms of bi-infinite time domains, they are valid in the mono-infinite case as well.

The notion of sampling invariance characterizes formulas whose truth value is "consistent" whether they are interpreted over dense-time or discrete-time behaviors. Informally, a formula $\phi$ is sampling invariant if the discrete-time behaviors that satisfy $\phi$ coincide with those obtained by "sampling" all the sufficiently slow dense-time behaviors that satisfy another formula $\phi'$ (where $\phi'$ is obtained from $\phi$ by suitably relaxing its interval bounds), and *vice versa* when $\phi$ is interpreted as a dense-time formula. Below, we recall the precise definition of sampling invariance, after briefly introducing the basic notions that are needed.

*Bounded variability.* As mentioned above, sampling invariance requires behaviors to be "sufficiently slow", with respect to a chosen period $\delta \in \mathbb{R}_{>0}$. Informally, the truth value of any atomic proposition must change at most once every $\delta$ time units; in other words, the change rate is bounded above by $1/\delta$. In fact, this requirement is sometimes called *bounded variability* [22]. The bounded variability requirement can be expressed as an MTL formula $\chi$, which we do not report here for brevity [13]. We denote the set of all dense-time behaviors satisfying $\chi$ by $\mathcal{B}_\chi \subset \mathcal{B}_\mathbb{R}$, and we call them $\chi$-regular behaviors. A formula $\phi$ is called $\chi$-valid if $b \models_\mathbb{R} \phi$ for all $b \in \mathcal{B}_\chi$, and $\chi$-satisfiable if $b \models_\mathbb{R} \phi$ for some $b \in \mathcal{B}_\chi$.

*Zeno and Berkeley.* A Zeno behavior is one where time progresses only by infinitesimal amounts, and thus it stops, instead of diverging. The name "Zeno" (introduced by Abadi and Lamport [1]) is a reference to the Greek philosopher Zeno of Elea and his paradoxes on time advancement. In this vein, we designate $\chi$-regular behaviors "non-Berkeley" [10], from the Irish philosopher George Berkeley[3] and his investigations arguing against the notion of infinitesimal. So, a behavior is "Berkeley" when it does not obey constraint $\chi$ for any value of $\delta$; thus the minimum distance in time between consecutive state changes is infinitesimal. Zeno behaviors are a special case of Berkeley behaviors; more generally, in a Berkeley behavior time can diverge, but with the system becoming arbitrarily "fast". See [10] for more details.

*Sampling of a behavior.* Let $b \in \mathcal{B}_\mathbb{R}$ be a dense-time behavior. Its *sampling*, with *sampling period* $\delta \in \mathbb{R}_{>0}$, is a discrete-time behavior $b' = \sigma_\delta[b] \in \mathcal{B}_\mathbb{Z}$ that agrees with $b$ at all integer multiples of $\delta$. Formally: $b'(k) = b(k\delta)$ for all $k \in \mathbb{Z}$.[4]

Let us point out a straightforward property of the sampling function $\sigma_\delta[\cdot]$ with respect to the set of behaviors $\mathcal{B}_\chi$.

**Lemma 1 (Properties of $\sigma_\delta[\cdot]$).** *For any $\delta \in \mathbb{R}_{>0}$, $\sigma_\delta[\cdot]$ is* onto *and* total.

*Adaptation functions.* To "switch" from the discrete-time to the dense-time interpretation of a formula $\phi$ in a way that preserves the truth value of $\phi$, one has to "adapt" the interval bounds appearing in $\phi$. This adaptation is formalized by two functions $\eta_\delta^\mathbb{R}\{\cdot\}$ and $\eta_\delta^\mathbb{Z}\{\cdot\}$: the former adapts dense-time formulas to be discrete-time ones, while the latter performs the converse.

[3] See e.g., http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Berkeley.html

[4] The original definition in [13] also introduced a basic offset $z \in \mathbb{R}$, but since it does not play any role in the present discussion we simply take it to be zero.

The exact definition of $\eta^{\mathbb{R}}_{\delta}\{\cdot\}$ and $\eta^{\mathbb{Z}}_{\delta}\{\cdot\}$, omitted for the lack of space, is given in [13, 12]. We note that, if $\phi$ is an MTL formula, then $\eta^{\mathbb{R}}_{\delta}\{\phi\}$ is an MTL$^*$ formula and $\eta^{\mathbb{Z}}_{\delta}\{\phi\}$ is an MTL formula.

*Sampling invariance.* Let us introduce precisely the notion of sampling invariance as a property of MTL formulas.

**Theorem 1 (Sampling Invariance [13]).** *Any MTL formula $\phi$ is sampling invariant, that is, for any sampling period $\delta$: (1) (closure under sampling) for all dense-time behavior $b \in \mathcal{B}_{\chi}$, if $b \models_{\mathbb{R}} \phi$ then $\sigma_{\delta}[b] \models_{\mathbb{Z}} \eta^{\mathbb{R}}_{\delta}\{\phi\}$; and (2) (closure under inverse sampling) for all discrete-time behavior $b \in \mathcal{B}_{\mathbb{Z}}$, if $b \models_{\mathbb{Z}} \phi$ then $\forall b' \in \mathcal{B}_{\chi}$ such that $\sigma_{\delta}[b'] = b$, it is $b' \models_{\mathbb{R}} \eta^{\mathbb{Z}}_{\delta}\{\phi\}$.*

## 3 Discretization of Dense-Time MTL Through Sampling

This section presents a discretization technique to solve the verification problem for MTL specifications.

First of all, given a dense-time MTL formula $\phi$ and a sampling period $\delta > 0$, we define two functions $\Omega_{\delta} : \mathrm{MTL} \to \mathrm{MTL}^*$, $\mathrm{O}_{\delta} : \mathrm{MTL} \to \mathrm{MTL}$ that approximate $\phi$ through the discrete-time formulas $\Omega_{\delta}(\phi)$ and $\mathrm{O}_{\delta}(\phi)$; basically, these retain some properties of the *samplings* of the dense-time behaviors satisfying $\phi$, in a way that allows us to infer the validity of $\phi$ from the validity of its approximations. For reasons that will become apparent shortly, we name $\Omega_{\delta}(\phi)$ the *under-approximation* of $\phi$, and $\mathrm{O}_{\delta}(\phi)$ the *over-approximation*. They are presented in Sections 3.1 and 3.2, respectively.

In general, the verification problem consists in checking whether a system, described by a *specification* formula $\phi_{\mathsf{sys}}$, satisfies a given *property* $\phi_{\mathsf{prop}}$; in other words, whether $b \models_{\mathbb{R}} \phi_{\mathsf{prop}}$ holds for any behavior $b$ for which $b \models_{\mathbb{R}} \phi_{\mathsf{sys}}$ holds. Section 3.3 shows how to construct two discrete-time formulas $\phi^+, \phi^-$ that are both built upon the over- and under-approximations of $\phi_{\mathsf{sys}}$ and $\phi_{\mathsf{prop}}$. Then:

- the validity of $\phi^+$ over discrete time implies that system $\phi_{\mathsf{sys}}$ satisfies property $\phi_{\mathsf{prop}}$ over dense-time non-Berkeley behaviors;
- the non-validity of $\phi^-$ over discrete time implies that system $\phi_{\mathsf{sys}}$ does not satisfy property $\phi_{\mathsf{prop}}$ over some non-Berkeley dense-time behavior.

Finally, Section 3.4 shows how the previously introduced approximations can be used in an algorithm to verify a system specified in dense time. The resulting verification technique is however *incomplete*, in that the results from the validity checking of the approximations of a formula can be inconclusive; therefore the algorithm can fail. The incompleteness may be partially mitigated by suitably choosing the sampling period $\delta$, but it cannot be entirely avoided. This is inevitable, since the approximation is a *simplification* of the dense-time verification problem, which cannot be fully captured by discrete-time reasoning only, for a number of well-known reasons [2, 15].

### 3.1 Under-Approximation

The approximation function $\Omega_\delta\left(\cdot\right)$ maps dense-time MTL formulas to discrete-time MTL* formulas such that the non-validity of the latter implies the non-validity of the former, over behaviors in $\mathcal{B}_\chi$. More precisely, for MTL formulas such that the chosen sampling period $\delta$ is in $\mathcal{D}_\phi$ (see Section 2.1), $\Omega_\delta\left(\cdot\right)$ is defined as follows.

$$\begin{array}{lcl}
\Omega_\delta\left(\beta\right) & \equiv & \beta \\
\Omega_\delta\left(\phi_1 \wedge \phi_2\right) & \equiv & \Omega_\delta\left(\phi_1\right) \wedge \Omega_\delta\left(\phi_2\right) \\
\Omega_\delta\left(\phi_1 \vee \phi_2\right) & \equiv & \Omega_\delta\left(\phi_1\right) \vee \Omega_\delta\left(\phi_2\right) \\
\Omega_\delta\left(\mathsf{U}_{\langle l,u\rangle}(\phi_1, \phi_2)\right) & \equiv & \mathsf{U}^{\uparrow}_{[l/\delta, u/\delta]}(\Omega_\delta\left(\phi_1\right), \Omega_\delta\left(\phi_2\right)) \\
\Omega_\delta\left(\mathsf{R}_{\langle l,u\rangle}(\phi_1, \phi_2)\right) & \equiv & \mathsf{R}^{\downarrow}_{\langle l/\delta, u/\delta\rangle}(\Omega_\delta\left(\phi_1\right), \Omega_\delta\left(\phi_2\right))
\end{array}$$

The following lemma, proved in [12], justifies the name *under-approximation*.

**Lemma 2 (Under-approximation).** *For any MTL formula $\phi$, for any $\delta \in \mathcal{D}_\phi$, and for any $b \in \mathcal{B}_\mathbb{Z}$: if $b \not\models_\mathbb{Z} \Omega_\delta\left(\phi\right)$ then for all $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta\left[b'\right] = b$ it is $b' \not\models_\mathbb{R} \phi$.*

### 3.2 Over-Approximation

The approximation function $\mathrm{O}_\delta\left(\cdot\right)$ maps dense-time MTL formulas to discrete-time MTL formulas such that the validity of the latter implies the validity of the former, over behaviors in $\mathcal{B}_\chi$. More precisely, for MTL formulas such that the chosen sampling period $\delta$ is in $\mathcal{D}_\phi$ (see Section 2.1), $\mathrm{O}_\delta\left(\cdot\right)$ is defined as follows.

$$\begin{array}{lcl}
\mathrm{O}_\delta\left(\beta\right) & \equiv & \beta \\
\mathrm{O}_\delta\left(\phi_1 \vee \phi_2\right) & \equiv & \mathrm{O}_\delta\left(\phi_1\right) \vee \mathrm{O}_\delta\left(\phi_2\right) \\
\mathrm{O}_\delta\left(\phi_1 \wedge \phi_2\right) & \equiv & \mathrm{O}_\delta\left(\phi_1\right) \wedge \mathrm{O}_\delta\left(\phi_2\right) \\
\mathrm{O}_\delta\left(\mathsf{U}_{\langle l,u\rangle}(\phi_1, \phi_2)\right) & \equiv & \mathsf{U}_{[l/\delta+1, u/\delta-1]}(\mathrm{O}_\delta\left(\phi_1\right), \mathrm{O}_\delta\left(\phi_2\right)) \\
\mathrm{O}_\delta\left(\mathsf{R}_{\langle l,u\rangle}(\phi_1, \phi_2)\right) & \equiv & \mathsf{R}_{[l/\delta-1, u/\delta+1]}(\mathrm{O}_\delta\left(\phi_1\right), \mathrm{O}_\delta\left(\phi_2\right))
\end{array}$$

The following lemma justifies the name *over-approximation*.

**Lemma 3 (Over-approximation).** *For any MTL formula $\phi$, for any $\delta \in \mathcal{D}_\phi$, and for any $b \in \mathcal{B}_\mathbb{Z}$: if $b \models_\mathbb{Z} \mathrm{O}_\delta\left(\phi\right)$ then for all $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta\left[b'\right] = b$ it is $b' \models_\mathbb{R} \phi$.*

*Proof (sketch, see also [12]).* $\mathrm{O}_\delta\left(\phi\right)$ is an MTL formula, which is therefore sampling invariant according to Theorem 1, and in particular closed under inverse sampling. Therefore, let $b \in \mathcal{B}_\mathbb{Z}$ such that $b \models_\mathbb{Z} \mathrm{O}_\delta\left(\phi\right)$. Then the definition of closure under inverse sampling implies that all $b' \in \mathcal{B}_\chi$ such that $b = \sigma_\delta\left[b'\right]$ satisfy $b' \models_\mathbb{R} \eta^\mathbb{Z}_\delta\{\mathrm{O}_\delta\left(\phi\right)\}$. According to the definition of $\eta^\mathbb{Z}_\delta\{\cdot\}$ (given in [12, Tab. 3]), one can check that $\eta^\mathbb{Z}_\delta\{\mathrm{O}_\delta\left(\phi\right)\} \Rightarrow \phi$ is valid. More precisely, $\eta^\mathbb{Z}_\delta\{\cdot\}$ allows one to choose arbitrarily if any interval $\langle l, u\rangle$ of *until* and *since* should be closed or not, so that it is possible to match the original intervals in $\phi$. Moreover,

$\eta_\delta^{\mathbb{Z}}\{\cdot\}$ always yields a closed interval in instances of *release* and *trigger*; therefore, it gives either the same subformula as in $\phi$, or a *strengthening* of it, when it replaces an open interval with its closure. It is easy to check that this property is lifted to whole formulas. All in all, $b' \models_{\mathbb{R}} \eta_\delta^{\mathbb{Z}}\{\mathrm{O}_\delta(\phi)\}$ implies $b' \models_{\mathbb{R}} \phi$.

## 3.3 System Approximations

Let us now consider a system formally described by an MTL formula $\phi_{\mathsf{sys}}$, and a putative property described by another MTL formula $\phi_{\mathsf{prop}}$. Verification amounts to proving (or disproving) that all behaviors that satisfy $\phi_{\mathsf{sys}}$ also satisfy $\phi_{\mathsf{prop}}$.

Let us abbreviate by $\mathrm{Alw}(\phi)$ the nesting MTL formula $\phi \wedge \Box_{(0,+\infty)}(\phi) \wedge \overleftarrow{\Box}_{(0,+\infty)}(\phi)$; $b \models_{\mathbb{T}} \mathrm{Alw}(\phi)$ iff $b \models_{\mathbb{T}} \phi$, for any behavior $b$, so $\mathrm{Alw}(\phi)$ can be expressed without nesting if $\phi$ is flat, through the global satisfiability semantics. Then, the verification problem can be reduced to that of determining the validity of the MTL formula $\mathrm{Alw}(\phi_{\mathsf{sys}}) \Rightarrow \mathrm{Alw}(\phi_{\mathsf{prop}})$. To this end we prove the following.

**Proposition 1 (Approximations).** *For any MTL formulas $\phi_1, \phi_2$, and for any $\delta \in \mathcal{D}_{\phi_1,\phi_2}$: (1) if $\mathrm{Alw}(\Omega_\delta(\phi_1)) \Rightarrow \mathrm{Alw}(\mathrm{O}_\delta(\phi_2))$ is $\mathbb{Z}$-valid, then $\mathrm{Alw}(\phi_1) \Rightarrow \mathrm{Alw}(\phi_2)$ is $\chi$-valid; and (2) if $\mathrm{Alw}(\mathrm{O}_\delta(\phi_1)) \Rightarrow \mathrm{Alw}(\Omega_\delta(\phi_2))$ is not $\mathbb{Z}$-valid, then $\mathrm{Alw}(\phi_1) \Rightarrow \mathrm{Alw}(\phi_2)$ is not $\chi$-valid.*

*Proof.* Let $\delta \in \mathcal{D}_{\phi_1,\phi_2}$.

*Proof of (1).* Assume that $\phi^+ = \mathrm{Alw}(\Omega_\delta(\phi_1)) \Rightarrow \mathrm{Alw}(\mathrm{O}_\delta(\phi_2))$ is $\mathbb{Z}$-valid. That is, for all $b \in \mathcal{B}_{\mathbb{Z}}$ it is $b \models_{\mathbb{Z}} \phi^+$; equivalently: either $b \not\models_{\mathbb{Z}} \Omega_\delta(\phi_1)$ or $b \models_{\mathbb{Z}} \mathrm{O}_\delta(\phi_2)$. From Lemmas 3 and 2, this implies that for all $b \in \mathcal{B}_{\mathbb{Z}}$, for all $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta[b'] = b$, it is either $b' \not\models_{\mathbb{R}} \phi_1$ or $b' \models_{\mathbb{R}} \phi_2$. Thus, let $b'$ be any dense-time behavior in $\mathcal{B}_\chi$; from Lemma 1, there exists a $b \in \mathcal{B}_{\mathbb{Z}}$ such that $\sigma_\delta[b'] = b$. We conclude that for all $b' \in \mathcal{B}_\chi$, either $b' \not\models_{\mathbb{R}} \phi_1$ or $b' \models_{\mathbb{R}} \phi_2$. All in all, $\mathrm{Alw}(\phi_1) \Rightarrow \mathrm{Alw}(\phi_2)$ is $\chi$-valid.

*Proof of (2).* We note that the proof of (2) can be obtained from the proof of (1) by duality. Thus, assume that $\phi^- = \mathrm{Alw}(\mathrm{O}_\delta(\phi_1)) \Rightarrow \mathrm{Alw}(\Omega_\delta(\phi_2))$ is not $\mathbb{Z}$-valid. That is, for some $b \in \mathcal{B}_{\mathbb{Z}}$ it is $b \not\models_{\mathbb{Z}} \phi^-$; equivalently: $b \models_{\mathbb{Z}} \mathrm{O}_\delta(\phi_1)$ and $b \not\models_{\mathbb{Z}} \Omega_\delta(\phi_2)$. From Lemmas 3 and 2, this implies that there exists a $b \in \mathcal{B}_{\mathbb{Z}}$ such that, for all $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta[b'] = b$, it is $b' \models_{\mathbb{R}} \phi_1$ and $b' \not\models_{\mathbb{R}} \phi_2$. Next, Lemma 1 states that, for all $b \in \mathcal{B}_{\mathbb{Z}}$, there exists some $b'$ such that $b' \in \mathcal{B}_\chi$ and $\sigma_\delta[b'] = b$. We conclude that there exists a $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta[b'] = b$, $b' \models_{\mathbb{R}} \phi_1$ and $b' \not\models_{\mathbb{R}} \phi_2$. All in all, $\mathrm{Alw}(\phi_1) \Rightarrow \mathrm{Alw}(\phi_2)$ is not $\chi$-valid. $\Box$

## 3.4 Validity Checking Procedure

Let us finally present the validity checking algorithm based on the approximations described above.

The algorithm takes as input a set of MTL formulas $\phi_{\mathsf{sys}}^1, \ldots, \phi_{\mathsf{sys}}^m, \phi_{\mathsf{prop}}$, where $\phi_{\mathsf{sys}}^i$ are the formulas describing the system, and $\phi_{\mathsf{prop}}$ is the property to be verified, as well as a suitable value $\delta$. The algorithm checks the validity of $\phi = \bigwedge_{i=1,\ldots,m} \mathrm{Alw}\big(\phi_{\mathsf{sys}}^i\big) \Rightarrow \mathrm{Alw}(\phi_{\mathsf{prop}})$ as follows.

1. For each formula $\gamma \in \phi_{\mathsf{prop}} \cup \bigcup_{i=1,\ldots,m} \phi^{i}_{\mathsf{sys}}$, compute the over-approximation $\mathrm{O}_\delta(\gamma)$ and the under-approximation $\Omega_\delta(\gamma)$.
2. Compute:
   $\phi^+ = \bigwedge_{i=1,\ldots,m} \mathrm{Alw}\big(\Omega_\delta\left(\phi^{m}_{\mathsf{sys}}\right)\big) \Rightarrow \mathrm{Alw}(\mathrm{O}_\delta\left(\phi_{\mathsf{prop}}\right))$;
   $\phi^- = \bigwedge_{i=1,\ldots,m} \mathrm{Alw}\big(\mathrm{O}_\delta\left(\phi^{m}_{\mathsf{sys}}\right)\big) \Rightarrow \mathrm{Alw}(\Omega_\delta\left(\phi_{\mathsf{prop}}\right))$.
3. If $\phi^+$ is $\mathbb{Z}$-valid, then $\phi$ is $\chi$-valid for sampling period $\delta$;
4. otherwise, if $\phi^-$ is not $\mathbb{Z}$-valid, then $\phi$ is not $\chi$-valid for sampling period $\delta$;
5. otherwise, fail.

**Incompleteness of the algorithm.** The incompleteness of the algorithm in determining the validity of MTL formulas is two-fold. First, the algorithm does not check *all* dense-time behaviors for satisfaction of an MTL formula $\phi$, but only those obeying constraint $\chi$ for the chosen sampling period $\delta$. Choosing a smaller $\delta$ may mitigate this shortcoming, as this amounts to choosing a finer sampling of behaviors or, equivalently, to allowing faster behaviors. However, this may also *not* bring better results. In fact, as $\delta$ decreases, not only do the approximation formulas change, but also more behaviors (namely, faster ones) are allowed; thus the effects of shortening the sampling periods are subtle and they may become difficult to predict. We leave a comprehensive study of this phenomenon to future work.

The second source of incompleteness lies in the technique itself, that is based on two different approximations for formula $\phi$. Therefore, it is possible that $\phi^+$ is non-valid and $\phi^-$ is valid; in this case, no conclusion about the validity of $\phi$ can be drawn.

## 4 Implementation and Experiments

This section describes the implementation of the verification algorithm (Section 4.1), presents two system verification problems (Section 4.2), and reports some of the results obtained in solving them using the tool presented in Section 4.1 (Section 4.3). Several more results can be found in [12].

### 4.1 Discrete-Time Bounded Validity Checking

The technique introduced in Section 3 reduces the validity-checking problem for MTL formulas over dense time to that over discrete time; the latter is known to be decidable and EXPSPACE-complete [2]. Recently, validity-checking techniques based on the use of propositional satisfiability (SAT) checkers have been developed for discrete-time verification, and they have yielded very encouraging performances in practical tests. Recent variants of these techniques offer the possibility to check completeness.

$\mathbb{Z}$ot is an agile and easily extensible bounded satisfiability checker ($\mathbb{Z}$ot and the examples described in this section are available for download [19]). The tool supports different logic languages through a multi-layered approach: its core uses PLTL, and a decidable predicative fragment of TRIO (in practice equivalent to ${}^{\mathbb{R}}_{\mathbb{Z}}$TRIO and MTL) is defined on top of it. $\mathbb{Z}$ot supports different encodings of temporal logic as SAT problems. Indeed, the user can choose a particular encoding to carry out verification, and the tool loads automatically the corresponding plug-in. At the moment, a few variants of some of the encodings presented in [3] are supported, and the encoding over $\mathbb{Z}$ presented in [20].

In order to assess the practical feasibility of our discretization technique, we verified some examples using $\mathbb{Z}$ot. To this end, $\mathbb{Z}$ot was extended to accept MTL$^+$ formulas, and to perform the discretization routine on formulas. The experimental results are described in Section 4.3.

### 4.2 Examples

We modeled two systems: a simple controlled reservoir (similar to the one in [13]), and a coffee machine. They are described only informally here; the exact formalization is given in the Appendix and details are in [12].

*The controlled reservoir.* The controlled reservoir system consists of a reservoir and a controller. The reservoir can nondeterministically leak and being filled with new liquid by the controller. The level of fluid in the reservoir is described by two predicates: $\ell \geq \mathsf{min}$ holds when the level of fluid is above a minimum level, $\ell \geq \mathsf{thres}$ holds if the level is above a control threshold, assumed to be higher than the minimum. The system is described by five formulas, shown in the Appendix, stating the behavior of the fluid level under all combinations of filling and leaking, and the control action (filling is triggered as soon as the level goes below the control threshold). The property (1) to be verified requires that, after the system is "initialized" by setting the level above the control threshold, the level stays above the minimum forever in the future:

$$\ell \geq \mathsf{thres} \;\Rightarrow\; \Box_{(0,+\infty)}(\ell \geq \mathsf{min}) \tag{1}$$

The system description is parametric with respect to a single parameter $\nu$. The desired property holds if and only if the sampling period $\delta$ equals $\nu$. Otherwise the property does not hold since the sampling period is "too short" with respect to $\nu$: this corresponds to allowing faster behaviors for which the given specification is too weak to assess the desired property (more details can be found in [12]).

*The coffee machine.* The second example consists in the description of a coffee machine, in operational fashion. We introduce the predicates: $\mathsf{prepare\_cup}$, $\mathsf{press\_button}$, $\mathsf{start\_pour}$, $\mathsf{end\_pour}$, $\mathsf{get\_cup}$. They represent, respectively, the actions of inserting a new cup in the coffee machine, pressing the button to start the brewing process, beginning and ending of the pouring of coffee, retrieving a cup (presumably filled with coffee) from the machine. We also introduce the predicates: $\mathsf{pour}$, $\mathsf{cup\_present}$, $\mathsf{coffee\_ready}$, and $\mathsf{key\_in}$. They are meant to hold when, respectively, the coffee is pouring into the cup, a cup is inserted in the machine, the coffee has been completely brewed, and a key (to operate the machine, say by recording the coffee credits of the user) is inserted in the machine. Three constants $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$ describe the various delays in the operations of the machine; also, a parameter $\nu$ is introduced to relate the various delays in a suitable manner (see [12] for details). The behavior of the machine is modeled through ten formulas, shown in the Appendix. From these, two candidate properties of the system should be verified: the first (2) states that the pouring ends only if a key was inserted in the past (between $\mathrm{T}_3$ and $\mathrm{T}_1 + \mathrm{T}_2 + \mathrm{T}_3$ time units ago); the second (3) asserts that a cup is present while the coffee is being poured.

$$\mathsf{end\_pour} \;\Rightarrow\; \overleftarrow{\Diamond}_{[\mathrm{T}_3, \mathrm{T}_1+\mathrm{T}_2+\mathrm{T}_3]}(\mathsf{key\_in}) \tag{2}$$

$$\mathsf{pour} \;\Rightarrow\; \mathsf{cup\_present} \tag{3}$$

Some modifications were required in order to obtain a system formalization which avoids some idiosyncrasies of the dense-time description that obstruct the discretization process. They are discussed in [12].

Both candidate properties hold if $\delta = 1$. Otherwise, the properties may not hold, also according to the particular values for the constants $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$, which interact in a subtle way. See [12] for more details.

### 4.3 Experiments

Tables 1–2 report a small sub-set of the results obtained in an array of tests with the discretization techniques and $\mathbb{Z}$ot; more of them can be found in [12]. For each test the tables report: the value $k$ of the bound given to $\mathbb{Z}$ot (in other words, the size of the explored space); the value of parameter $\nu$ in the models; the value of $\delta$, according to which the discretizations are built; the value of other parameters in the models (i.e., $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$ in the case of the coffee machine); the outcome of the validity check for the properties to be verified. Each test is done both over mono-infinite domain, and over bi-infinite domain. For each test the tables report, in addition to the outcome ($\top$ means valid, $\bot$ means non-valid, $\sim$ means that the approximation technique has been inconclusive), the net (CPU) time and the total amount of memory taken in the process.

The tests have been performed on a PC equipped with an AMD Athlon64 x2 4600+ processor, 2 Gb of RAM, and Ubuntu GNU/Linux. $\mathbb{Z}$ot used GNU CLisp v. 2.39, and MiniSat v. 1.14 as SAT-solving engine.

The reservoir example (in Table 1) is a simple one, and in fact the results are highly predictable and satisfactory. Inconclusive results are never obtained when applying the discretization technique, and the property is confirmed to be valid if and only if $\nu = \delta$. The times and spaces required to obtain the results are always relatively small, and they scale rather well with the increase of the bound. Finally, notice that it usually takes a shorter time to check the validity than to check the non-validity; this is obvious, as the latter requires to submit both $\phi^+$ and $\phi^-$ to the validity checker, while the former checks just $\phi^+$.

Table 2 reports some of the results obtained with the coffee machine example. Property (2) is shown to be valid for all the choices of parameters made in the experiments reported in Table 2. The times needed to get this result are rather

| $k$ | $\nu$ | $\delta$ | Pr.(1) $\mathbb{T} = \mathbb{N}$ (time / mem) | Pr.(1) $\mathbb{T} = \mathbb{Z}$ (time / mem) |
|---|---|---|---|---|
| 5 | 10 | 10 | $\top$ (0.2 s / 2.2 Mb) | $\top$ (0.4 s / 2.6 Mb) |
| 5 | 10 | 10/3 | $\bot$ (0.7 s / 5.8 Mb) | $\bot$ (1.2 s / 9 Mb) |
| 10 | 10 | 10 | $\top$ (0.6 s / 3.2 Mb) | $\top$ (0.7 s / 4.9 Mb) |
| 10 | 10 | 10/3 | $\bot$ (1.6 s / 12.1 Mb) | $\bot$ (2 s / 18.1 Mb) |
| 50 | 10 | 10 | $\top$ (2.8 s / 15.3 Mb) | $\top$ (5.3 s / 23.5 Mb) |
| 50 | 10 | 10/3 | $\bot$ (8.6 s / 110.8 Mb) | $\bot$ (19.5 s / 149.5 Mb) |
| 100 | 10 | 10 | $\top$ (9.5 s / 30.4 Mb) | $\top$ (20.2 s / 46.7 Mb) |
| 100 | 10 | 10/3 | $\bot$ (29.9 s / 361.9 Mb) | $\bot$ (66.1 s / 470.6 Mb) |
| 200 | 10 | 10 | $\top$ (33.3 s / 60.7 Mb) | $\top$ (72.6 s / 93.1 Mb) |
| 200 | 10 | 10/3 | $\bot$ (108.6 s / 1240.1 Mb) | $\bot$ (245.1 s / 1588.3 Mb) |

**Table 1.** Checking property (1) of the reservoir example.

| $k$ | $\nu$ | $\delta$ | $T_1, T_2, T_3$ | Pr.(2) $\mathbb{N}$ (time / mem) | Pr.(3) $\mathbb{N}$ (time / mem) | Pr.(2) $\mathbb{Z}$ (time / mem) | Pr.(3) $\mathbb{Z}$ (time / mem) |
|---|---|---|---|---|---|---|---|
| 10 | 1 | 1 | 4,4,4 | $\top$ (1.8 s / 11.7 Mb) | $\top$ (1.3 s / 9.8 Mb) | $\top$ (3.5 s / 17.9 Mb) | $\top$ (2.6 s / 14.9 Mb) |
| 10 | 2 | 1 | 4,4,4 | $\top$ (1.7 s / 11.4 Mb) | $\top$ (1.4 s / 9.5 Mb) | $\top$ (3.4 s / 17.6 Mb) | $\top$ (2.2 s / 14.6 Mb) |
| 10 | 3 | 1 | 10,7,8 | $\top$ (3.1 s / 17.4 Mb) | $\sim$ (4.2 s / 35.7 Mb) | $\top$ (7.3 s / 27 Mb) | $\sim$ (8.6 s / 52.5 Mb) |
| 20 | 1 | 1 | 4,4,4 | $\top$ (5.1 s / 22.7 Mb) | $\top$ (3.6 s / 18.9 Mb) | $\top$ (11.6 s / 34.7 Mb) | $\top$ (8 s / 28.9 Mb) |
| 20 | 2 | 1 | 4,4,4 | $\top$ (4.9 s / 22 Mb) | $\top$ (3.3 s / 18.3 Mb) | $\top$ (10.9 s / 34.1 Mb) | $\top$ (7.6 s / 28.3 Mb) |
| 20 | 3 | 1 | 10,7,8 | $\top$ (11.6 s / 33.6 Mb) | $\sim$ (13.8 s / 78.3 Mb) | $\top$ (25.6 s / 52.5 Mb) | $\sim$ (30.4 s / 114.3 Mb) |
| 30 | 1 | 1 | 4,4,4 | $\top$ (11 s / 33.6 Mb) | $\top$ (7.7 s / 28.1 Mb) | $\top$ (24 s / 51.6 Mb) | $\top$ (16.7 s / 43 Mb) |
| 30 | 2 | 1 | 4,4,4 | $\top$ (10.4 s / 32.7 Mb) | $\top$ (7.1 s / 27.2 Mb) | $\top$ (23.7 s / 51.1 Mb) | $\top$ (16.1 s / 42.2 Mb) |
| 30 | 3 | 1 | 10,7,8 | $\top$ (24.2 s / 49.7 Mb) | $\sim$ (28.6 s / 153.3 Mb) | $\top$ (55.5 s / 78.3 Mb) | $\sim$ (65.3 s / 189.2 Mb) |
| 40 | 1 | 1 | 4,4,4 | $\top$ (18.4 s / 45 Mb) | $\top$ (12.9 s / 37.3 Mb) | $\top$ (39.6 s / 68.9 Mb) | $\top$ (27.6 s / 57.1 Mb) |
| 40 | 2 | 1 | 4,4,4 | $\top$ (17.8 s / 43.8 Mb) | $\top$ (12.3 s / 36.1 Mb) | $\top$ (38.3 s / 67.7 Mb) | $\top$ (26.4 s / 56 Mb) |
| 40 | 3 | 1 | 10,7,8 | $\top$ (40.6 s / 66.3 Mb) | $\sim$ (47.8 s / 200 Mb) | $\top$ (89.3 s / 103.8 Mb) | $\sim$ (106.7 s / 284.8 Mb) |
| 50 | 1 | 1 | 4,4,4 | $\top$ (27.7 s / 56 Mb) | $\top$ (19.4 s / 46.5 Mb) | $\top$ (60.8 s / 85.8 Mb) | $\top$ (42.1 s / 71.2 Mb) |
| 50 | 2 | 1 | 4,4,4 | $\top$ (26.6 s / 54.5 Mb) | $\top$ (18.4 s / 45 Mb) | $\top$ (57.9 s / 84.4 Mb) | $\top$ (39.9 s / 69.8 Mb) |
| 50 | 3 | 1 | 10,7,8 | $\top$ (60.8 s / 82.6 Mb) | $\sim$ (71.8 s / 272.1 Mb) | $\top$ (136.2 s / 129.3 Mb) | $\sim$ (160.3 s / 389.1 Mb) |

**Table 2.** Checking properties (2) and (3) of the coffee machine example.

short, and scale with the length of $k$. This is reasonable, as the main factors affecting the complexity of the check are the values of the parameters $T_1, T_2, T_3$, which however stay in a small range in all tests.

The outcomes of the validity check of the other property (3) are, on the other hand, more varied. As stated when presenting the example, if $\delta = 1$ the second property is valid for the system. While this is confirmed by several of the tests, some cases fall in the incompleteness area of the method, and analyzing the approximations gives inconclusive results. In any case, the time and space required are rather small.

## 5 Conclusion

We presented a technique to reduce the verification problem for dense-time MTL specifications to the corresponding problem over discrete-time models, based on the notions of *sampling* and *sampling invariance*. In a nutshell, we perform simple syntactic transformations on the MTL formulas to be checked for validity; the resulting formulas retain (partial) information about the discrete-time samplings of the dense-time behaviors described by the the original formulas.

This approach, which considers only a subset of generic MTL formulas, has a two-fold incompleteness: it verifies only "sufficiently slow" dense-time behaviors (although the "speed" of the behaviors can often be modulated), and the analysis of the discretized formulas may yield inconclusive results.

The technique is however simple to implement in practice, and it was used, on top of the ℤot bounded validity checker for discrete-time formulas, to carry out some experiments. The results are promising in that they show that the effects of incompleteness can often be mitigated in practice, and the computational effort required to check the discretized formulas is usually acceptably small.

Future work in this line of research will follow three main directions. First, the technique and tool of this paper will be applied to real-life industrial case-studies. Second, our verification technique will be extended to deal with systems described through operational formalisms such as timed automata or Petri nets. Third, methods will be developed to guide the writing of dense-time specifications in a form that is amenable to the application of discretization.

**Acknowledgments.** We thank Paritosh Pandya for discussions suggesting to apply the notion of sampling to dense-time verification through discretization, Mario Arrigoni Neri for providing a sketch of the coffee machine example, and the anonymous reviewers of several conferences for their remarks.

## References

1. M. Abadi and L. Lamport. An old-fashioned recipe for real-time. *ACM TOPLAS*, 16(5):1543–1571, 1994.
2. R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.
3. A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Comp. Sci.*, 2(5:5):1–64, 2006.
4. G. Chakravorty and P. K. Pandya. Digiziting interval duration logic. In *Proc. of CAV'03*, volume 2725 of *LNCS*, pages 167–179, 2003.
5. E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti. From formal models to formally-based methods: an industrial experience. *ACM TOSEM*, 8(1):79–113, 1999.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 2000.
7. L. de Alfaro and Z. Manna. Verification in continuous time by discrete reasoning. In *Proc. of AMAST'95*, volume 936 of *LNCS*, pages 292–306, 1995.
8. D. D'Souza, R. Mohan M., and P. Prabhakar. Eliminating past operators in metric temporal logic. Technical Report IISc-CSA-TR-2006-11, 2006.
9. G. E. Fainekos and G. J. Pappas. Robust sampling for MITL specifications. In *Proc. of FORMATS'07*, volume 4763 of *LNCS*, 2007.
10. C. A. Furia. *Scaling up the formal analysis of real-time systems.* PhD thesis, DEI, Politecnico di Milano, May 2007.
11. C. A. Furia, D. Mandrioli, A. Morzenti, and M. Rossi. Modeling time in computing: A taxonomy and a comparative survey. Technical Report 2007.22, DEI, Politecnico di Milano, 2007.
12. C. A. Furia, M. Pradella, and M. Rossi. Dense-time MTL verification through sampling. Technical Report 2007.37, DEI, Politecnico di Milano, April 2007.
13. C. A. Furia and M. Rossi. Integrating discrete- and continuous-time metric temporal logics through sampling. In *Proc. of FORMATS'06*, volume 4202 of *LNCS*, pages 215–229, 2006.

14. C. A. Furia and M. Rossi. On the expressiveness of MTL variants over dense time. In *Proc. of FORMATS'07*, volume 4763 of *LNCS*, pages 163–178, 2007.
15. T. A. Henzinger. It's about time: Real-time logics reviewed. In *Proc. of CONCUR'98*, volume 1466 of *LNCS*, pages 439–454, 1998.
16. T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Proc. of ICALP'92*, volume 623 of *LNCS*, pages 545–558, 1992.
17. Y. Hirshfeld and A. M. Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.
18. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
19. M. Pradella. ℤot. `http://home.dei.polimi.it/pradella`, March 2007.
20. M. Pradella, A. Morzenti, and P. San Pietro. The symmetry of the past and of the future. In *Proc. of ESEC/FSE 2007*, 2007.
21. B. Sharma, P. K. Pandya, and S. Chakraborty. Bounded validity checking of interval duration logic. In *Proc. of TACAS'05*, volume 3440 of *LNCS*, pages 301–316, 2005.
22. T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. of FTRTFT'94*, volume 863 of *LNCS*, pages 694–715, 1994.

## Appendix: Example Specifications

*The reservoir system.*

$$\ell \geq \mathsf{min} \land \Box_{(0,\nu)}(\mathsf{F}) \;\Rightarrow\; \Box_{[\nu,\nu]}(\ell \geq \mathsf{min}) \tag{4}$$

$$\ell \geq \mathsf{thres} \;\Rightarrow\; \Box_{[\nu,\nu]}(\ell \geq \mathsf{min}) \tag{5}$$

$$\ell \geq \mathsf{min} \land \Box_{(0,\nu)}(\neg\mathsf{F} \land \neg\mathsf{L}) \;\Rightarrow\; \Box_{[\nu,\nu]}(\ell \geq \mathsf{min}) \tag{6}$$

$$\ell \geq \mathsf{thres} \;\Rightarrow\; \ell \geq \mathsf{min} \tag{7}$$

$$\ell < \mathsf{thres} \;\Rightarrow\; \mathsf{F} \tag{8}$$

*The coffee machine.*[5]

$$\mathsf{prepare\_cup} \;\Rightarrow\; \overleftarrow{\Diamond}_{(0,\mathrm{T}_1)}(\mathsf{press\_button}) \tag{9}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \overleftarrow{\Diamond}_{(0,\mathrm{T}_2)}(\mathsf{prepare\_cup}) \tag{10}$$

$$\mathsf{end\_pour} \;\Rightarrow\; \overleftarrow{\Box}_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{start\_pour}) \land \overleftarrow{\Box}_{[0,\mathrm{T}_3)}(\mathsf{pour}) \tag{11}$$

$$\mathsf{press\_button} \;\Rightarrow\; \mathsf{key\_in} \tag{12}$$

$$\neg\mathsf{pour} \land \bigcirc(\mathsf{pour}) \;\Leftrightarrow\; \mathsf{start\_pour} \tag{13}$$

$$\Box_{(0,\mathrm{T}_3)}(\mathsf{pour}) \;\Rightarrow\; \mathsf{start\_pour} \land \Box_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{end\_pour}) \tag{14}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \mathsf{cup\_present} \land \neg\mathsf{coffee\_ready} \tag{15}$$

$$\mathsf{cup\_present} \land \bigcirc(\neg\mathsf{cup\_present}) \;\Leftrightarrow\; \mathsf{get\_cup} \tag{16}$$

$$\mathsf{get\_cup} \;\Rightarrow\; \mathsf{coffee\_ready} \tag{17}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \Box_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{end\_pour}) \land \Box_{(0,\mathrm{T}_3]}(\mathsf{pour}) \tag{18}$$

[5] The $\bigcirc(\beta)$ operator is defined as $\mathsf{U}_{(0,+\infty)}(\beta,\top) \lor (\neg\beta \land \mathsf{R}_{(0,+\infty)}(\beta,\bot))$.