Dense-Time MTL Verification Through Sampling

Carlo A. Furia, Matteo Pradella, and Matteo Rossi

April 2007

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 [FR06]. Since the reduction is from an undecidable problem to a decidable one, the technique is necessarily incomplete, so it fails to provide conclusive answers on some problem instances. The paper discusses this shortcoming and hints at how it can be mitigated in practice. The verification technique has been implemented on top of a tool for discrete-time bounded validity checking; the paper also reports on in-the-small experiments with the tool, which show some promising results.

Contents

1 Introduction

When modeling the behavior of real-time systems, the nature of the time domain plays a prominent role, and it must be carefully chosen. In particular, modelers must decide whether to use a dense set (possibly continuous) or a discrete one [FMMR07]. From a purely modeling viewpoint, dense time offers advantages in terms of naturalness and completeness of description (being of the same quality as "physical time")[1], in particular in describing the composition of purely asynchronous processes (that can occur at any instant in time), and it is usually strictly more expressive [AH93]. Conversely, in practice, discrete-time models are in general more amenable to (automated) verification than dense-time ones. In fact, dense-time formalisms are often undecidable or with highly complex decidability problems [AH93]; in addition, while verification methods for discrete-time models can often be built upon existing techniques (e.g., for LTL, automata, and untimed formalisms), the native 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*. In a nutshell, discretization techniques consist in reducing the verification problem from dense to discrete time; therefore, they permit the re-use of existing techniques (and tools). Of course, for formalisms that are strictly more expressive in their dense-time variant, discretization techniques are necessarily *incomplete*; that is, they fail to give conclusive results on some instances of the verification problem. In the following sub-section, we survey briefly some discretization techniques.

Most — nearly all — of these techniques are grounded on the notion of *digitization*, first introduced by Henzinger, Manna, and Pnueli [HMP92], or variations thereof. The digitization framework usually considers semantics based on timed state sequences, it assumes a weakly monotonic time, and it focuses on discrete-time verification. In general, it is not easy to characterize the digitizability of (syntactic) classes of languages, such as metric temporal logics.

In [FR06], we introduced a different framework for discretization, based on the notion of *sampling*, which is an idealization of physical sampling processes. The framework targets $\frac{\mathbb{R}}{\mathbb{Z}}$TRIO, a metric temporal logic interpreted over *behaviors*, that is total functions of time. It defines an $\frac{\mathbb{R}}{\mathbb{Z}}$TRIO subset that is *sampling invariant*; informally, this means that formulas in this language subset can be interpreted consistently whether over dense-time behaviors, or over discrete-time samplings thereof. While the results of [FR06] were derived for $\frac{\mathbb{R}}{\mathbb{Z}}$TRIO, it is immediate to translate them for the well-known Metric Temporal Logic (MTL); hence, in this paper we always refer to MTL rather than $\frac{\mathbb{R}}{\mathbb{Z}}$TRIO, also when citing results from [FR06].

The framework of [FR06] naturally opens the door to interesting applications in the field of formal verification of logic specifications. In this paper, we provide techniques to reduce the validity problem for (flat) dense-time MTL formulas

[1] At least declined in its classical flavor [Smo01].

to the validity problem of discrete-time MTL formulas. The latter problem is known to be decidable [AH93], and therefore it can be solved with known techniques (which have been implemented in a number of automatic tools); in particular, the experiments in this paper exploit a tool based on bounded validity checking techniques [BHJ+06].

More precisely, our verification technique builds *two approximations* (ϕ^+ and ϕ^-) of the formula representing the instance of the verification problem. These approximations represent a (simple) 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 ϕ^+ over discrete time implies the validity of the original formula over dense time; conversely, the non-validity of ϕ^- over discrete time implies the non-validity of the original formula over dense time. As we have discussed above, the technique must be incomplete; in particular, it may happen that the validity check of the approximations yields inconclusive results. In the paper, we discuss how this can be mitigated in practice, and especially how the choice of the sampling period (when building the approximations) impacts the completeness coverage.

Finally, we report briefly about some experimental results we obtained with a tool implemented to perform the discretization, based on top of a discrete-time tool called ℤot [PMS]. Although limited to a small set of examples, our tests show interesting results, and they are a first assessment of the feasibility of our discretization techniques. In particular, they show that the incompleteness is not often a practical hurdle (especially because other limitations, such as the inherent scalability even of discrete-time methods, are more prominent).

The paper is organized as follows. Section 1.1 surveys some related works on discretization techniques, with particular focus on those that are closest to ours. Then, Section 2 introduces the ingredients of our techniques; that is, the MTL subset we consider in this paper is formally introduced, the notions of sampling (and sampling invariance) from [FR06] are recalled, and the ℤot tool we used in the experiments is briefly presented. Section 3 is the core of the paper, as it introduces the technical results that permit discretization according to the notion of sampling. Section 4 reports on the experiments we performed with our techniques. Finally, Section 5 concludes.

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 [HMP92]. Their discretization techniques are based on the notion of *digitization*; a (semantic) property (that is, a set of timed state sequences) is digitizable if it is both closed under digitization and closed under inverse digitization. Roughly speaking, a property is closed under digitization if all the timed state sequences obtained by digitizing the real-timed state sequences are already also integer-timed state sequences of the property; conversely, a prop-

erty 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.

For digitizable properties, discrete-time verification completely captures dense-time verification; more precisely, if a system formalization is closed under digitization, and a property is closed under inverse digitization, the problem of determining if the system complies with the property is perfectly reducible to the discrete-time case. Although the framework of [HMP92] is semantic, the paper then focuses on determining the digitizability for formal languages, and namely timed transition systems and the logic MTL. We refer the reader to [Fur07, Chapter 9] for a detailed comparison of the notion of digitization with our notion of sampling invariance (recalled in Section 2.2). In particular, we recall that in [Fur07, Chapter 9] it is shown that the notions of sampling invariance and digitizability are orthogonal: there are digitizable properties that are not sampling invariant, and there are sampling invariant properties that are not digitizable. This may imply that the use of these notions for practical discrete-time verification of dense-time properties would also yield orthogonal results; that is each approach has its own practical benefits. The comparison of the results of this paper with those listed below goes in the direction of assessing this prediction.

Many subsequent works have applied the notion of digitization introduced in [HMP92] to specific classes of formalisms. In the remainder of this section we report on some of these works.

Digitization for automata formalisms. Göllü, Puri, and Varaiya [GPV94] apply the notion of discretization to Alur and Dill's timed automata [AD94]. More precisely, they define two discrete semantics for timed automata, where clocks are restricted to vary in a discrete state-space. For these semantics, they show that the untimed language of any (dense-timed) timed automaton coincides with that generated by applying the untime operation to the discrete semantics. This may lead to simpler verification algorithms through analysis in the discrete semantics.

Krčál and Pelánek's work [KP05] is in the same vein, as it studies the relations between (standard) continuous semantics for timed automata and discrete semantics with a fixed time step. In particular, they investigate relations with respect to different behavioral equivalences, and they study the decidability of reachability for an extension of timed automata known as *stopwatch automata* [HKPV98].

Bouajjani, Echahed, and Robbana [BER94] extend the notion of digitizability to deal with timed graphs with duration variables; some of their results impact indirectly the digitization of timed automata. On the other hand, Bošnački [Boš99] studies directly how the notion of digitizability applies to timed au-

tomata, by determining sufficient conditions for a timed automaton to be closed under (inverse) digitization. More precisely, it is shown that the following classes of timed automata are closed under digitization: (1) weakly constrained timed automata (i.e., automata with no clock constraints with a $<$ or a $>$); and (2) timed automata where all edges forming a cycle are weakly constrained. Dually, strongly constrained timed automata (i.e., automata with no clock constraints with a $\leq$ or $\geq$) are closed under inverse sampling. Another, more complex, condition for closure under inverse sampling is also given; it is based on an ordering of clock constraints, which basically requires that the "maximum" or "minimum" constraint element in the ordering does not use a $\leq$ or a $\geq$. Finally, it is discussed how digitizability could be checked on-the-fly with suitable heuristics, and the usage of automata relaxations for incomplete integer-time verification. It is also remarked that the results on the digitizability of timed automata imply some results for formalisms that are of similar expressive power. In particular, [GD00] shows that MTL formulas that contain only intervals of the form $[0, d]$, for some integer d, can be translated — through a tableau construction — to timed automata which satisfy the more complex condition for closure under inverse digitization; therefore, they are suitable properties to be checked through discretization.

There have been several other works on discretization techniques to ease the verification of timed automata. Without reporting about them in full, for the sake of brevity, let us cite [MP95, BMT99, BLN03, OW03, CLT07].

Digitization for descriptive formalisms. Ouaknine [Oua02] studies the notion of digitizability for timed CSP [Sch00]. This requires to extend the definition of digitizability to handle (timed) *failures*, which are at the basis of the denotational semantics of (timed) CSP; failures allow a behavioral expression of liveness properties. The main result of Ouaknine's paper is the proof of the closure under digitization of timed CSP; as a consequence, properties that are closed under inverse digitization can be checked over dense-time timed CSP through reduction to the discrete-time case. On the other hand, notice that it is not simple to characterize the closure under inverse digitization of timed CSP. Two other important technical contributions of [Oua02] are the notion of integral multiplication on processes, and the characterization of full abstraction. The former consists in scaling all time constants in a CSP by an integer multiple. It may happen that a process becomes closed under inverse digitization if scaled by a large enough constant; thus, scaling may render verification through discretization possible. The results on abstraction, instead, show that properties that are closed under inverse digitization cannot separate processes that are equivalent in the discrete-time semantics; this shows that the discretization techniques based on reduction through digitizability are not only sufficient, but also necessary.

The digitizability of duration calculus [CHR91] is studied by Hung and Giang [HG96]. Besides the standard real-time semantics of duration calculus, they introduce a *sampling semantics* where the time model is a discrete set of

evenly-spaced steps. They relate this semantics to the standard semantics, by introducing a number of inference rules that derive formulas in the standard semantics from formulas in the sampling semantics, and *vice versa*. Note that some of them introduce the requirement that predicates hold their true values over intervals of length (at least) δ. This is similar to the notion of χ-regular behaviors, to be introduced in Section 2.2, except that it is asymmetric (i.e., it constraints the *true* value only). From these inference rules, they also derive *refinement rules* that allow one to relate valid duration calculus formulas in the sampling semantics from valid formulas in the standard semantics. These results are generalizations of the notion of digitizability to the case of arbitrary clock-step (where in the case of digitization the clock-step has always a length of one), although one can always scale time constants to accomplish the same results. The paper also shows that the sampling semantics coincides with the standard semantics "in the limit", that is for arbitrarily small clock-steps.

Chakravorty and Pandya [CP03] apply the notion of digitization to Interval Duration Logic (IDL) [Pan02], a duration calculus variant where formulas are interpreted over timed state sequences (finite ones, in that work). 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 such 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 (and notice that so is duration calculus over bounded-variable behaviors [Frä96]). Using these techniques, Sharma, Pandya, and Chakravorty [SPC05] report experimental trials through a variety of discrete-time verification tools (namely, automata-based and bounded validity checking based on SAT [Frä02]), as well as comparison with a bounded validity checking technique that directly encodes IDL formulas into *lin-sat* formulas (that is, Boolean combinations of propositional variables, and linear constraints over real variables that can be solved by linear programming techniques).

Other approaches to discretization. Asarin, Maler, and Pnueli in [AMP98] tackle the problem of discretizing the behavior of digital circuits in a way that preserves the qualitative behavior of the circuit in a strict sense, that is such that the ordering of events in continuous time must be unchanged in the discretized behavior. In particular, this implies that events occurring at different times, however close, must occur at *distinct* discrete time instant, in the same order. This is a stronger notion than — in particular — that by Henzinger et al. [HMP92], where a strictly monotonic sequence is allowed to be discretized into a weakly monotonic one. More precisely, the authors are basically considering Boolean signals, that is interval-based behaviors over infinite time (i.e.,

the time domain is $\mathbb{R}_{\geq 0}$). The same authors have shown in [MP95] that the subclass of signals that are definable by their digital circuits can also be accepted by timed automata. The results about digitizability are demonstrated by geometric arguments about polyhedra. While for the simpler acyclic circuits digitizability is always possible for some suitable choice of discretization step δ, for general cyclic circuits the distance between state changes can become arbitrarily small; in this case, discretization is impossible for any choice of step $\delta > 0$. On the contrary, the same cyclic circuits are discretizable according to the weaker notion of discretization introduced in [HMP92].

De Alfaro and Manna [dM95] approach the problem of discretization with reference to the temporal logic TL of [MP93], a particular flavor of predicative modal logic, and to the timed trace semantics (as well as an extension of it to represent hybrid behaviors). As usual in this kind of works, the goal is to relate the discrete-time and the continuous-time semantics for the logic. To this end, 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 timed traces that are sample equivalent. In turn, the notion of sample equivalence for two timed traces requires the existence of a (sufficiently fine-grained) trace that refines both. Then, a translation function Ω from continuous to discrete traces that preserves refinement is introduced. Finally, the notion of *finite variability* is introduced: roughly speaking, a formula ϕ is of finite variability if, for each timed trace, one can find a refinement (called ground trace) such that any subformula of ϕ 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 ϕ in the continuous semantics corresponds to that of $\Omega(\phi)$ in the discrete semantics. Similarly, the validity of finitely variable formulas over discrete time implies the validity of the same formulas over continuous time (but the converse implication does not hold in general). 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. Finally, the same results are extended to hybrid systems under the phase transition system semantics [MP93].

2 Preliminaries

This section introduces the syntax and semantics of the variant of the MTL language we use in this paper (Section 2.1); it recalls the notion of *sampling invariance* [FR06] and some related results that will be used in Section 3 (Section 2.2); and it briefly presents the discrete-time verification tools that we will experiment with in Section 4 (Section 2.3).

2.1 Yet Another MTL Variant

In this paper, we consider a variant of purely propositional Metric Temporal Logic (MTL, [AH93]) as our reference specification language. For brevity, we refer to this variant simply as "MTL".

2.1.1 MTL Syntax and Semantics

Syntax. We consider MTL endowed with past, as well as future, operators. The basic temporal operator of this language is the *bounded until* U_I, and its past counterpart *bounded since* S_I.

The results of sampling invariance, recalled in Section 2.2, as well as the discretization techniques introduced in Section 3 require MTL formulas to be in a normal form where negations are pushed on (Boolean combinations of) atomic propositions, and not to nest temporal operators. Therefore, in order to ease the discussion, we introduce directly the MTL syntax for this normal form; hence, we have the operators *bounded release* R_I and *bounded trigger* T_I — dual to the *until* and *since*, respectively — as primitive.

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.[2] Usually, one considers intervals with nonnegative endpoints, but we permit negative endpoints to render the presentation more uniform and straightforward. In this paper $\mathbb{T}$ coincides with either the reals $\mathbb{R}$ (dense time) or the integers $\mathbb{Z}$ (discrete time) — or with some subset of them. More precisely, we call *bi-infinite* the sets $\mathbb{R}$ and $\mathbb{Z}$, and *mono-infinite* their subsets $\mathbb{R}_{\geq 0}$ and $\mathbb{N} = \mathbb{Z}_{\geq 0}$. The following grammar defines the syntax of MTL, where $\mathsf{p} \in \mathcal{P}$ and $I \in \mathcal{I}$.

$$\begin{array}{lcl} \beta & ::= & \mathsf{p} \mid \neg\beta \mid \beta_1 \wedge \beta_2 \\ \phi & ::= & \beta \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ & & \mathsf{U}_I(\beta_1, \beta_2) \mid \mathsf{S}_I(\beta_1, \beta_2) \mid \mathsf{R}_I(\beta_1, \beta_2) \mid \mathsf{T}_I(\beta_1, \beta_2) \end{array}$$

It is customary to define a number of abbreviations, such as $\bot, \top, \Rightarrow, \Leftrightarrow$, and derived operators; Table 1 lists the derived temporal operators we use in the remainder.[3]

Semantics. Let us now define the semantics of MTL as defined above, parametrically with respect to the choice of the time domain $\mathbb{T}$. The semantics is defined over *behaviors*, that is 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 $\mathbb{T}$. The semantics is then defined as follows:

[2] That is any $\mathcal{I} \ni I = \langle l, u \rangle$ for some $l \leq u$ where $l \in \mathbb{T} \cap \mathbb{Q}$ and $u \in (\mathbb{T} \cap \mathbb{Q}) \cup \{\pm\infty\}$, $\langle$ is one of (and [, and similarly for $\rangle$.

[3] $\bigcirc$ and $\overleftarrow{\bigcirc}$ denote the *nowon* and *uptonow* operators, whose definition will be justified later.

Operator	$\equiv$	Definition
$\Diamond_I(\beta)$	$\equiv$	$\mathsf{U}_I(\top, \beta)$
$\overleftarrow{\Diamond}_I(\beta)$	$\equiv$	$\mathsf{S}_I(\top, \beta)$
$\Box_I(\beta)$	$\equiv$	$\mathsf{R}_I(\bot, \beta)$
$\overleftarrow{\Box}_I(\beta)$	$\equiv$	$\mathsf{T}_I(\bot, \beta)$
$\bigcirc(\beta)$	$\equiv$	$\mathsf{U}_{(0,+\infty)}(\beta, \top) \vee (\neg\beta \wedge \mathsf{R}_{(0,+\infty)}(\beta, \bot))$
$\overleftarrow{\bigcirc}(\beta)$	$\equiv$	$\mathsf{S}_{(0,+\infty)}(\beta, \top) \vee (\neg\beta \wedge \mathsf{T}_{(0,+\infty)}(\beta, \bot))$

Table 1: MTL derived temporal operators

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

Whenever $\models_{\mathbb{T}} \phi$ we say that ϕ is $\mathbb{T}$-valid.

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. Moreover, *until* and *since* are in their matching versions: this means that $\mathsf{U}_I(\phi_1, \phi_2)$ requires ϕ_2 to hold together with ϕ_1 (at some instant in I); conversely, *release* and *trigger* are in their non-matching versions. These small variations introduce an asymmetry under complement in the language. In fact, one can check that $\mathsf{U}_I(\phi_1, \phi_2)$ (resp. $\mathsf{S}_I(\phi_1, \phi_2)$) is equivalent to $\neg\mathsf{R}_I(\neg\phi_1, \neg(\phi_1 \wedge \phi_2))$ (resp. $\neg\mathsf{T}_I(\neg\phi_1, \neg(\phi_1 \wedge \phi_2))$). Discussing how these variations, as well as the restriction to nesting-free formulas, impact the expressiveness of the language is outside the scope of the present paper. While we refer to other works, we also second Maler et al.'s remark [MNP06, Section 5] that:

> [...] giving preference to results proved with respect to the most general existing definitions is a mathematicians attitude that should not be adopted without a critical examination. Such an attitude can be counter-productive in young domains where "classical" results and definitions are only decades old, and the most appropriate formalization has not yet stabilized.

2.1.2 MTL$^{+/*}$ Syntax and Semantics

In order to express the discretization relations in Section 3, we have to introduce 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, but they are needed only to illustrate the discretization techniques. We call "MTL^{+}" the extension of MTL with these operators, and "MTL*" the MTL variant where we *replace* the operators U_I, S_I, R_I, T_I with the variants $\mathsf{U}_I^{\uparrow}$, $\mathsf{S}_I^{\uparrow}$, $\mathsf{R}_I^{\downarrow}$, and $\mathsf{T}_I^{\downarrow}$, respectively. Their purpose will become clear in the following sections.

Let us define the semantics of the non-matching variants of basic *until* and *since*, and of the matching variants of basic *release* and *trigger*.

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

$b(t) \models_{\mathbb{T}} \mathsf{S}_I^{\uparrow}(\phi_1, \phi_2)$ iff there exists $d \in I$ such that: $b(t-d) \models_{\mathbb{T}} \phi_2$ and, for all $u \in [0, d)$ it is $b(t-u) \models_{\mathbb{T}} \phi_1$

$b(t) \models_{\mathbb{T}} \mathsf{R}_I^{\downarrow}(\phi_1, \phi_2)$ 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 $b(t+u) \models_{\mathbb{T}} \phi_1$

$b(t) \models_{\mathbb{T}} \mathsf{T}_I^{\downarrow}(\phi_1, \phi_2)$ 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 $b(t-u) \models_{\mathbb{T}} \phi_1$

It is now apparent that $\mathsf{U}_I(\phi_1, \phi_2)$ (resp. $\mathsf{R}_I(\phi_1, \phi_2)$) is equivalent to $\neg\mathsf{R}_I^{\downarrow}(\neg\phi_1, \neg\phi_2)$ (resp. $\neg\mathsf{U}_I^{\uparrow}(\neg\phi_1, \neg\phi_2)$), and similarly for their past counterparts.

2.1.3 Granularity

For an MTL formula ϕ, let $\mathcal{I}_\phi$ the set of all non-null, finite interval bounds appearing in ϕ. More explicitly, if ϕ contains exactly n intervals: $\langle l_i/L_i, u_i/U_i\rangle$[4] for $i = 1, \ldots, n$, let $\mathcal{I}_\phi = \bigcup_{i=1,\ldots,n}\{l_i/L_i, u_i/U_i\} \cap \mathbb{Q}_{\neq 0} = \{\gamma_i/\Gamma_i \,|\, i = 1, \ldots, m\}$.[5]

Given a formula ϕ, its *granularity* ρ_ϕ is a pair of values (r_ϕ, R_ϕ) where r is the greatest common divisor of the integers $\{\gamma_i\}_{i=1,\ldots,m}$ and R is the least common multiple of the integers $\{\Gamma_i\}_{i=1,\ldots,m}$. Formally:

$$\rho_\phi \quad = \quad (r_\phi, R_\phi) \quad = \quad \left(\gcd_{i=1,\ldots,m} \gamma_i, \operatorname*{lcm}_{i=1,\ldots,m} \Gamma_i \right)$$

In the remainder we will employ the notion of set $\mathcal{D}_\phi$ for a formula ϕ: $\mathcal{D}_\phi$ is the set of positive values δ such that any interval bound in ϕ is an integer if divided by δ:[6]

$$\mathcal{D}_\phi \quad = \quad \left\{ \delta \in \mathbb{Q}_{>0} \;\middle|\; \mathbb{Z} \ni \frac{\gamma_i}{\Gamma_i}\frac{1}{\delta} \text{ for } i = 1, \ldots, m \right\}$$

[4] u_i/U_i can be $+\infty$, and l_i/L_i can be $-\infty$.

[5] Clearly, $m \leq 2n$.

[6] We assume naturally that $\pm\infty/x = \pm\infty$ for any finite, positive x.

It is not difficult to show that $\mathcal{D}_\phi$ is the set of all fractions d/D such that: (1) D is a common multiple of all denominators Γ_i; and (2) d divides all numerators γ_i. Therefore, any element d/D of $\mathcal{D}_\phi$ can be expressed as a divisor d of r_ϕ over a multiple D of R_ϕ; in formulas:

$$\mathcal{D}_\phi \quad = \quad \left\{ \frac{d}{D} \;\middle|\; d | r_\phi \text{ and } D = kR_\phi \text{ for some } k \in \mathbb{N} \right\}$$

Notice that $\mathcal{D}_\phi$ has a maximum (given by r_ϕ/R_ϕ) but no minimum.

2.2 Sampling Invariance

The discretization technique that will be introduced in Section 3 is based on the notion of *sampling invariance* [FR06]. In this section, we recall the basic definitions and results about sampling invariance that are needed in the remainder; we refer to [FR06, FR05] for details.

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 ϕ is sampling invariant if the discrete-time behaviors that satisfy ϕ coincide with those obtained by "sampling" all the sufficiently slow dense-time behaviors that satisfy another formula ϕ', where ϕ' is obtained from ϕ by suitably relaxing its interval bounds.

Below, we recall the precise definition of sampling invariance, after briefly formally introducing the basic notions that are needed.

χ-regular behaviors. As we hinted 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 δ time units; in other words, the change rate is bounded above by $1/\delta$. This is why this requirement is sometimes called *bounded variability* [Wil94].

More precisely, given a set of atomic proposition $\mathcal{P}$ and a positive real δ, we can express the regularity constraint by the following formula χ:

$$\chi \quad \equiv \quad \forall s \in 2^{\mathcal{P}} : \left(s \Rightarrow \overleftarrow{\Diamond}_{[0,\delta]} \left(\Box_{[0,\delta]}(s) \right) \right)$$

where $s \in 2^{\mathcal{P}}$ denotes any possible Boolean combination of atomic propositions.

We denote the set of all dense-time behaviors satisfying χ by $\mathcal{B}_\chi \subset \mathcal{B}_\mathbb{R}$, and we call them χ-regular behaviors. A formula ϕ is called χ-valid if $b \models_\mathbb{R} \phi$ for all $b \in \mathcal{B}_\chi$, and χ-satisfiable if $b \models_\mathbb{R} \phi$ for some $b \in \mathcal{B}_\chi$. Notice that a χ-regular behavior (for some δ) is also non-Zeno *a fortiori* [GM01].

Zeno and Berkeley. The name "Zeno" was introduced by Abadi and Lamport [AL94], with reference to Zeno's paradoxes on time advancement; in fact, a Zeno behavior is one where time progresses only by infinitesimal amounts, and

thus it "stops", instead of diverging. In this vein, we refer to χ-regular behaviors as *non-Berkeley* behaviors [Fur07]. The name is from another philosopher[7] who argued against the notion of infinitesimal in his investigations. In fact, a behavior is "Berkeley" when it does not obey the constraint χ for any values of δ; thus the minimum distance in time between consecutive state changes is infinitesimal. Zeno behaviors are a special case of this; more generally, in a Berkeley behavior time can diverge, but with the system becoming arbitrarily "fast". See [Fur07] 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 δ. Formally: $b'(k) = b(k\delta)$ for all $k \in \mathbb{Z}$.[8]

Let us point out two straightforward properties 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*, that is: for all $b \in \mathcal{B}_{\mathbb{Z}}$ there exists a b' such that $b' \in \mathcal{B}_\chi$ and $\sigma_\delta[b'] = b$.*
- $\sigma_\delta[\cdot]$ *is* total*, that is: for all $b \in \mathcal{B}_\chi$ there exists a $b' \in \mathcal{B}_{\mathbb{Z}}$ such that $\sigma_\delta[b] = b'$.*

Adaptation functions. To "switch" from the discrete-time to the dense-time interpretation of a formula ϕ, in a way that preserves the truth value of ϕ, one has to "adapt" the bounds of intervals appearing in ϕ. 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 adaptation.

Let us recall the exact definition of $\eta_\delta^{\mathbb{R}}\{\cdot\}$ and $\eta_\delta^{\mathbb{Z}}\{\cdot\}$ in Tables 2 and 3, respectively. We point out that: if ϕ is an MTL formula: (1) $\eta_\delta^{\mathbb{R}}\{\phi\}$ is an MTL* formula; and (2) $\eta_\delta^{\mathbb{Z}}\{\phi\}$ is an MTL formula.

Sampling invariance. Let us introduce precisely the notion of sampling invariance.

Definition 2 (Sampling Invariance [FR06])**.** Given a formula ϕ and a sampling period δ:

- ϕ is *closed under sampling* iff for any dense-time behavior $b \in \mathcal{B}_\chi$:

$$b \models_{\mathbb{R}} \phi \quad \Rightarrow \quad \sigma_\delta[b] \models_{\mathbb{Z}} \eta_\delta^{\mathbb{R}}\{\phi\}$$

- ϕ is *closed under inverse sampling* iff for any discrete-time behavior $b \in \mathcal{B}_{\mathbb{Z}}$:

$$b \models_{\mathbb{Z}} \phi \quad \Rightarrow \quad \forall b' \in \mathcal{B}_\chi : \big(\sigma_\delta[b'] = b \;\Rightarrow\; b' \models_{\mathbb{R}} \eta_\delta^{\mathbb{Z}}\{\phi\}\big)$$

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

[8] The original definition in [FR06] 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.

$\eta_\delta^{\mathbb{R}}\{\beta\}$	$\equiv$	β
$\eta_\delta^{\mathbb{R}}\{\phi_1 \wedge \phi_2\}$	$\equiv$	$\eta_\delta^{\mathbb{R}}\{\phi_1\} \wedge \eta_\delta^{\mathbb{R}}\{\phi_2\}$
$\eta_\delta^{\mathbb{R}}\{\phi_1 \vee \phi_2\}$	$\equiv$	$\eta_\delta^{\mathbb{R}}\{\phi_1\} \vee \eta_\delta^{\mathbb{R}}\{\phi_2\}$
$\eta_\delta^{\mathbb{R}}\left\{\mathsf{U}_{\langle l,u\rangle}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{U}^{\uparrow}_{[\lfloor l/\delta\rfloor,\lceil u/\delta\rceil]}\left(\eta_\delta^{\mathbb{R}}\{\phi_1\},\eta_\delta^{\mathbb{R}}\{\phi_2\}\right)$
$\eta_\delta^{\mathbb{R}}\left\{\mathsf{S}_{\langle l,u\rangle}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{S}^{\uparrow}_{[\lfloor l/\delta\rfloor,\lceil u/\delta\rceil]}\left(\eta_\delta^{\mathbb{R}}\{\phi_1\},\eta_\delta^{\mathbb{R}}\{\phi_2\}\right)$
$\eta_\delta^{\mathbb{R}}\left\{\mathsf{R}_{\langle l,u\rangle}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{R}^{\downarrow}_{\langle l',u'\rangle}\left(\eta_\delta^{\mathbb{R}}\{\phi_1\},\eta_\delta^{\mathbb{R}}\{\phi_2\}\right)$
		where $l' = \begin{cases} \lfloor l/\delta\rfloor & \text{if } \langle \text{ is } (\\ \lceil l/\delta\rceil & \text{if } \langle \text{ is } [\end{cases}$
		and $u' = \begin{cases} \lceil u/\delta\rceil & \text{if } \rangle \text{ is }) \\ \lfloor u/\delta\rfloor & \text{if } \rangle \text{ is }] \end{cases}$
$\eta_\delta^{\mathbb{R}}\left\{\mathsf{T}_{\langle l,u\rangle}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{T}^{\downarrow}_{\langle l',u'\rangle}\left(\eta_\delta^{\mathbb{R}}\{\phi_1\},\eta_\delta^{\mathbb{R}}\{\phi_2\}\right)$
		where $l' = \begin{cases} \lfloor l/\delta\rfloor & \text{if } \langle \text{ is } (\\ \lceil l/\delta\rceil & \text{if } \langle \text{ is } [\end{cases}$
		and $u' = \begin{cases} \lceil u/\delta\rceil & \text{if } \rangle \text{ is }) \\ \lfloor u/\delta\rfloor & \text{if } \rangle \text{ is }] \end{cases}$

Table 2: The adaptation function $\eta_\delta^{\mathbb{R}}\{\cdot\}$.

$\eta_\delta^{\mathbb{Z}}\{\beta\}$	$\equiv$	β
$\eta_\delta^{\mathbb{Z}}\{\phi_1 \wedge \phi_2\}$	$\equiv$	$\eta_\delta^{\mathbb{Z}}\{\phi_1\} \wedge \eta_\delta^{\mathbb{Z}}\{\phi_2\}$
$\eta_\delta^{\mathbb{Z}}\{\phi_1 \vee \phi_2\}$	$\equiv$	$\eta_\delta^{\mathbb{Z}}\{\phi_1\} \vee \eta_\delta^{\mathbb{Z}}\{\phi_2\}$
$\eta_\delta^{\mathbb{Z}}\left\{\mathsf{U}_{[l,u]}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{U}_{\langle (l-1)\delta,(u+1)\delta\rangle}\left(\eta_\delta^{\mathbb{Z}}\{\phi_1\},\eta_\delta^{\mathbb{Z}}\{\phi_2\}\right)$
$\eta_\delta^{\mathbb{Z}}\left\{\mathsf{S}_{[l,u]}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{S}_{\langle (l-1)\delta,(u+1)\delta\rangle}\left(\eta_\delta^{\mathbb{Z}}\{\phi_1\},\eta_\delta^{\mathbb{Z}}\{\phi_2\}\right)$
$\eta_\delta^{\mathbb{Z}}\left\{\mathsf{R}_{[l,u]}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{R}_{[(l+1)\delta,(u-1)\delta]}\left(\eta_\delta^{\mathbb{Z}}\{\phi_1\},\eta_\delta^{\mathbb{Z}}\{\phi_2\}\right)$
$\eta_\delta^{\mathbb{Z}}\left\{\mathsf{T}_{[l,u]}(\phi_1,\phi_2)\right\}$	$\equiv$	$\mathsf{T}_{[(l+1)\delta,(u-1)\delta]}\left(\eta_\delta^{\mathbb{Z}}\{\phi_1\},\eta_\delta^{\mathbb{Z}}\{\phi_2\}\right)$

Table 3: The adaptation function $\eta_\delta^{\mathbb{Z}}\{\cdot\}$.

- ϕ is *sampling invariant* iff it is closed under sampling (when interpreted in the dense-time domain) and it is closed under inverse sampling (when interpreted in the discrete-time domain).

We have the following, a straightforward corollary of [FR06, Th. 1].

Theorem 3 (Sampling invariance of MTL)**.** *All MTL formulas are sampling invariant.*

2.3 Discrete-Time Bounded Validity Checking

Overall, our technique 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 [AH93]. 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.

In order to assess the practical outcomes of our discretization technique, we verified some examples using the tool ℤot. We enhanced ℤot to accept MTL$^+$ formulas, as well as with a module to perform the discretization routine on formulas. This section briefly describes the salient features of the tool.

ℤot [Pra07] is an agile and easily extensible bounded satisfiability checker. ℤot is available for download together with the examples described in Section 4. The tool supports different logic languages through a multi-layered approach: its core uses PLTL, and on top of it a decidable predicative fragment of TRIO is defined. An interesting feature of ℤot is its ability to support different encodings of temporal logic as SAT problems. Indeed, the user can choose a particular encoding to carry out verification, while the tool automatically loads the corresponding plug-in. This approach encourages experimentation, as plug-ins are usually quite simple, compact (usually around 500 lines of code), easily modifiable, and extensible. At the moment, a few variants of some of the encodings presented in [BHJ$^+$06] are supported, together with the encoding over ℤ presented in [PMS]. Figure 1 shows ℤot's internal architecture.

At present, ℤot essentially adapts bounded model checking techniques to purely descriptive TRIO specifications. This means that both the model and the property are expressed as TRIO formulas, like in [MPSS03], and unlike typical model checking.

ℤot offers two basic usage modalities:

1. *Bounded satisfiability checking (BSC)*: given as input a specification formula, the tool returns a (possibly empty) history (i.e., an execution trace of the specified system) which satisfies the specification. An empty history means that it is impossible to satisfy the specification.

2. *History checking and completion (HCC)*: The input specification file can also contain a partial (or complete) history H. In this case, if H complies with the specification, then a completed version of H is returned as output, otherwise the output is empty.

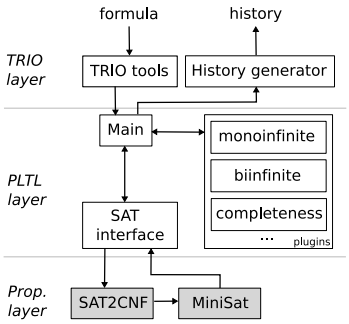

Figure 1: ℤot's architecture (external tools are shadowed in grey).

The provided output histories have always temporal length $\leq k$, the bound given by the user, but may represent infinite behaviors thanks to the loop selector variables, marking the start of the periodic sections of the history. The BSC modality can be used to check if a property *prop* of the given specification *spec* holds over every periodic behavior with period $\leq k$. In this case, the input file contains spec $\wedge \neg$prop, and, if prop indeed holds, then the output history is empty. If this is not the case, the output history is a useful counterexample which explains why prop does not hold.

ℤot also supports completeness checks, through a logic-based variant of some of the algorithms described in [SSS00]. This has nothing to do with the incompleteness of our discretization techniques, and it refers to the bound given to the validity checker. In other words, the completeness check consists in determining whether a given time bound of $k > 0$ steps is sufficient to determine with certainty the unsatisfiability of a given formula (i.e., if k is greater than the *diameter* of the formula). To avoid ambiguities with the problem of completeness of our discretization techniques, in this paper we refer to ℤot's completeness check as *bound completeness* check. Essentially, the bound completeness encoding considers only the *finite* component of the temporal structure. To assure bound completeness, constraints are added asserting that it is impossible to have two states S_i and S_j, such that $S_i = S_j$, for any $i \neq j$. Therefore, we are considering all the possible finite behaviors of the system under analysis. For systems described using pure logic, this check can be very expensive, because they usually contain a high degree of nondeterminism, and the bound completeness check is by its very nature exhaustive. We will see some consequences of this behavior when discussing the tests of Section 4.

3 Discretization of Dense-Time MTL Through Sampling

This section presents a discretization technique to check the validity of MTL formulas.

Given a dense-time MTL formula ϕ and a sampling period $\delta > 0$, we define two functions $\mathrm{O}_\delta(\cdot), \Omega_\delta(\cdot)$ that approximate ϕ through the discrete-time formulas $\mathrm{O}_\delta(\phi)$ and $\Omega_\delta(\phi)$; basically, these retain some properties of the *samplings* of the dense-time behaviors satisfying ϕ, in a way that allows us to infer the validity of ϕ from the validity of its approximations. For reasons that will become apparent, we name $\mathrm{O}_\delta(\phi)$ the *over-approximation* of ϕ, and $\Omega_\delta(\phi)$ the *under-approximation*. These approximations 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 ϕ_{sys}, satisfies a given *property* ϕ_{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. Therefore, in Section 3.3 we show how to build two discrete-time formulas ϕ^+, ϕ^- that both embed suitable approximations of ϕ_{sys} and ϕ_{prop}. Namely:

- the validity of ϕ^+ over discrete-time implies that the system ϕ_{sys} satisfies the property ϕ_{prop} over dense-time behaviors satisfying the constraint χ (see Section 2.2);
- the non-validity of ϕ^- over discrete-time implies that the system ϕ_{sys} fails to satisfy the property ϕ_{prop} over some dense-time behaviors satisfying the constraint χ.

Finally, Section 3.4 shows how the previously introduced approximations techniques can be put into use in an algorithm to verify a system specified in dense-time. The resulting approximation technique is however *incomplete*, in that we can get inconclusive results from the validity checking of the approximations of a formula; therefore the algorithm can fail. The incompleteness may be partially mitigated by suitably choosing the sampling period δ, but it cannot be entirely avoided. This is somewhat 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 [AH93, Hen98].

3.1 Over Approximation

The approximation function $\mathrm{O}_\delta(\cdot)$ 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, $\mathrm{O}_\delta(\cdot)$ is defined as follows, for MTL formulas such that the chosen sampling period δ is in $\mathcal{D}_\phi$ (see Section

2.1.3).

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

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

Lemma 4 (Over approximation)**.** *For any MTL formula ϕ, for any $\delta \in \mathcal{D}_\phi$, and for any $b \in \mathcal{B}_\mathbb{Z}$: if $b \models_\mathbb{Z} \mathrm{O}_\delta(\phi)$ then for all $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta[b'] = b$ it is $b' \models_\mathbb{R} \phi$.*

Proof. $\mathrm{O}_\delta(\phi)$ is an MTL formula, which is therefore sampling invariant according to Theorem 3, and in particular closed under inverse sampling. Therefore, let $b \in \mathcal{B}_\mathbb{Z}$ such that $b \models_\mathbb{Z} \mathrm{O}_\delta(\phi)$. Then the definition of closure under inverse sampling implies that all $b' \in \mathcal{B}_\chi$ such that $b = \sigma_\delta[b']$ satisfy $b' \models_\mathbb{R} \eta_\delta^\mathbb{Z}\{\mathrm{O}_\delta(\phi)\}$. According to the definition of $\eta_\delta^\mathbb{Z}\{\cdot\}$ given in Table 3, one can check that $\eta_\delta^\mathbb{Z}\{\mathrm{O}_\delta(\phi)\} \Rightarrow \phi$ is valid. More precisely, $\eta_\delta^\mathbb{Z}\{\cdot\}$ 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 ϕ. 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 ϕ, 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 $b' \models_\mathbb{R} \phi$. □

Non-monotonicity of $\mathrm{O}_\delta(\cdot)$. The notation for the over-approximation stresses the fact that the function depends on the parameter δ. We also note the fact that the satisfiability of the approximation of a formula is not monotonic with δ. In other words, it may happen that the satisfiability of a formula $\mathrm{O}_\delta(\psi)$ over some discrete-time behavior b keeps on switching as δ decreases (or increases).

For instance, let ψ be the formula $\mathsf{p} \Rightarrow \Diamond_{[1,2]}(\mathsf{q})$; notice that $\mathcal{D}_\psi = \{1/k \,|\, k \in \mathbb{N}_{>0}\}$. Then $\mathrm{O}_{1/k}(\psi) = \mathsf{p} \Rightarrow \Diamond_{[l_k,u_k]}(\mathsf{q})$, with $l_k = k+1$ and $u_k = 2k-1$; let $I_k = [l_k, u_k]$. Then, let us consider the following discrete-time behavior $\widetilde{b}$: p holds at 0 and nowhere else; q holds at all integer instants 2^h for $h = 1, 2, \ldots$. For any positive h, let us determine what values of k satisfy $2^h \in I_k$. From $k+1 \leq 2^h \leq 2k-1$, we get $2^{h-1}+1 \leq k \leq 2^h - 1$, after some manipulations. Also, for each positive j, the interval $[2^j+1, 2^{j+1}-1]$ does not contain any instant in which q holds. As a consequence, $\widetilde{b} \not\models_\mathbb{Z} \mathrm{O}_{1/k}(\psi)$ for all $k = 2^h$, and $\widetilde{b} \models_\mathbb{Z} \mathrm{O}_{1/k}(\psi)$ for all other positive integer values of k. In other words, the Boolean-valued function $f(1/k) \equiv \widetilde{b} \models_\mathbb{Z} \mathrm{O}_{1/k}(\psi)$ is non-monotonic in $1/k = \delta$.

3.2 Under Approximation

The approximation function $\Omega_\delta(\cdot)$ 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, $\Omega_\delta(\cdot)$ is defined, as follows, for MTL formulas such that the chosen sampling period δ is in $\mathcal{D}_\phi$ (see Section 2.1.3).[9]

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

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

Lemma 5 (Under approximation)**.** *For any MTL formula ϕ, for any $\delta \in \mathcal{D}_\phi$, and for any $b \in \mathcal{B}_\mathbb{Z}$: if $b \not\models_\mathbb{Z} \Omega_\delta(\phi)$ then for all $b' \in \mathcal{B}_\chi$ such that $\sigma_\delta[b'] = b$ it is $b' \not\models_\mathbb{R} \phi$.*

Proof. Since ϕ is an MTL formula, it is sampling invariant, and in particular closed under sampling, as mentioned in Section 2.2. Therefore, for any dense-time behavior $b \in \mathcal{B}_\chi$, if $b \models_\mathbb{R} \phi$ then $\sigma_\delta[b] \models_\mathbb{Z} \eta^\mathbb{R}_\delta\{\phi\}$; by taking the contrapositive we have that for any dense-time behavior $b \in \mathcal{B}_\chi$, if $\sigma_\delta[b] \not\models_\mathbb{Z} \eta^\mathbb{R}_\delta\{\phi\}$ then $b \not\models_\mathbb{R} \phi$. It is straightforward to check that, for all $\delta \in \mathcal{D}_\phi$, $\Omega_\delta(\phi) = \eta^\mathbb{R}_\delta\{\phi\}$ (where $\eta^\mathbb{R}_\delta\{\cdot\}$ is defined as in Table 2) by construction. Therefore, for any $b \in \mathcal{B}_\mathbb{Z}$ such that $b = \sigma_\delta[b']$ for some $b' \in \mathcal{B}_\chi$ and $b \not\models_\mathbb{Z} \Omega_\delta(\phi)$, it is $b' \not\models_\mathbb{R} \phi$. All in all, we have $b' \not\models_\mathbb{R} \phi$ as required. □

Non-monotonicity of $\Omega_\delta(\cdot)$. Similarly as for the over-approximation, the notation for the under-approximation stresses the fact that the function depends on the parameter δ. We also note the fact that the satisfiability of the approximation of a formula is not monotonic with δ. In other words it may happen that the satisfiability of a formula keeps on switching as δ decreases (or increases). We omit a full demonstration of this fact, which can however be built from the over-approximation example: just take ψ as before and $\widetilde{b}$ such that p holds at 0 and nowhere else, and q holds at all integer instants 2^{2h} for $h = 1, 2, \ldots$.

[9] We stress the fact that the parentheses in the interval of the under-approximation for the *release* and *trigger* operators must match those in the original formula.

3.3 System Approximations

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

Let us abbreviate by $\mathrm{Alw}(\phi)$ the MTL formula $\phi \wedge \Box_{[0,+\infty)}(\phi) \wedge \overleftarrow{\Box}_{[0,+\infty)}(\phi)$; notice that $b \models_{\mathbb{T}} \mathrm{Alw}(\phi)$ iff $b \models_{\mathbb{T}} \phi$, for any behavior b. 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 6 (Approximations)**.** *For any MTL formulas ϕ_1, ϕ_2, and for any $\delta \in \mathcal{D}_{\phi_1,\phi_2}$:*[10]

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 χ-valid;*
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 χ-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 4 and 5, 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 χ-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 4 and 5, 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 χ-valid. □

3.4 Validity Checking Procedure

Let us finally present the validity checking algorithm based on the approximation techniques described above. The algorithm is presented in Section 3.4.1, whereas Section 3.4.2 discusses its incompleteness.

[10] The definition of $\mathcal{D}$ is extended to sets of formulas as obvious; for instance one can take $\mathcal{D}_{\{\phi_j\}_{j\in J}} \equiv \mathcal{D}_{\bigwedge_{j\in J}\phi_j}$.

3.4.1 Approximation Checking Algorithm

The algorithm takes as input a set of MTL formulas $\phi^1_{\mathsf{sys}}, \ldots, \phi^m_{\mathsf{sys}}, \phi_{\mathsf{prop}}$, where ϕ^i_{sys} are the formulas describing the system, and ϕ_{prop} is the property to be verified.

The algorithm checks the validity of $\phi = \bigwedge_{i=1,\ldots,m} \mathrm{Alw}\big(\phi^i_{\mathsf{sys}}\big) \Rightarrow \mathrm{Alw}(\phi_{\mathsf{prop}})$ as follows.

1. Input $\phi^1_{\mathsf{sys}}, \ldots, \phi^m_{\mathsf{sys}}, \phi_{\mathsf{prop}}$.
2. Input a δ such that $\delta \in \mathcal{D}_{\phi^1_{\mathsf{sys}},\ldots,\phi^m_{\mathsf{sys}},\phi_{\mathsf{prop}}}$.
3. For each formula $\gamma \in \phi_{\mathsf{prop}} \cup \bigcup_{i=1,\ldots,m} \phi^i_{\mathsf{sys}}$, compute:
   - the over approximation $\mathrm{O}_\delta(\gamma)$; let $\mathrm{o}_i = \mathrm{O}_\delta\left(\phi^i_{\mathsf{sys}}\right)$ and $\mathrm{O} = \mathrm{O}_\delta\left(\phi_{\mathsf{prop}}\right)$;
   - the under approximation $\Omega_\delta(\gamma)$; let $\omega_i = \Omega_\delta\left(\phi^i_{\mathsf{sys}}\right)$ and $\Omega = \Omega_\delta\left(\phi_{\mathsf{prop}}\right)$.
4. Compute:
   - $\phi^+ = \bigwedge_{i=1,\ldots,m} \mathrm{Alw}(\omega_i) \Rightarrow \mathrm{Alw}(\mathrm{O})$;
   - $\phi^- = \bigwedge_{i=1,\ldots,m} \mathrm{Alw}(\mathrm{o}_i) \Rightarrow \mathrm{Alw}(\Omega)$.
5. If ϕ^+ is $\mathbb{Z}$-valid, then print "ϕ is χ-valid for sampling period δ".
6. Otherwise: if ϕ^- is not $\mathbb{Z}$-valid, then print "ϕ is not χ-valid for sampling period δ".
7. Otherwise: print "Cannot decide the validity of ϕ through sampling with sampling period δ".

The correctness of the algorithm follows directly from Proposition 6, keeping in mind that $b \models_{\mathbb{T}} \mathrm{Alw}(\psi_1) \wedge \mathrm{Alw}(\psi_2)$ iff $b \models_{\mathbb{T}} \mathrm{Alw}(\psi_1)$ and $b \models_{\mathbb{T}} \mathrm{Alw}(\psi_2)$.

3.4.2 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 ϕ, but only those "sufficiently slow", i.e., obeying the constraint χ for the chosen sampling period δ. Choosing a smaller δ 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 δ decreases, not only do we change the approximation formulas, but we also allow more behaviors (namely, faster ones); thus the effects of shortening the sampling periods are subtle and they may become difficult to predict. We leave an accurate 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 the formula ϕ. Therefore, it is possible that ϕ^+ is non-valid and ϕ^- is valid; from this case we could draw no conclusion about the validity of ϕ.

An incompleteness example. Let us build a trivial example to convince ourselves that the second incompleteness scenario can actually take place; then, in Section 4 we will discuss the effects of incompleteness from a more practical viewpoint, through examples of system specification.

Let us consider a system described by the following formulas.

$$\mathsf{p} \vee \Diamond_{[0,+\infty)}(\mathsf{p}) \vee \overleftarrow{\Diamond}_{[0,+\infty)}(\mathsf{p}) \tag{1}$$

$$\mathsf{p} \;\Rightarrow\; \Box_{(0,+\infty)}(\neg\mathsf{p}) \wedge \overleftarrow{\Box}_{(0,+\infty)}(\neg\mathsf{p}) \tag{2}$$

$$\mathsf{p} \;\Rightarrow\; \Diamond_{[1,1]}(\mathsf{q}) \tag{3}$$

In practice, p is required to happen exactly once, and q holds exactly one instant after p does. Let us also notice that the set of Formulas (1–3) is unsatisfiable for non-Berkeley behaviors, as Formula (2) forces p to have a punctual behavior. Nonetheless, we now show that our approximation technique fails to solve the trivial problem of verifying that (3) is also a property of the system described by Formulas (1–3).

Let $\delta = 1/k$ for some positive k; these are the only possible values for δ for the above formulas. After computing the approximations, we get the formulas in Table 4.[11]

γ	$\mathrm{O}_\delta(\gamma)$	$\Omega_\delta(\gamma)$
(1)	$\mathrm{o}_1 = \mathsf{p} \vee \Diamond_{[1,+\infty)}(\mathsf{p}) \vee \overleftarrow{\Diamond}_{[1,+\infty)}(\mathsf{p})$	$\omega_1 = \mathsf{p} \vee \Diamond_{[0,+\infty)}(\mathsf{p}) \vee \overleftarrow{\Diamond}_{[0,+\infty)}(\mathsf{p})$
(2)	$\mathrm{o}_2 = \mathsf{p} \Rightarrow \Box_{[-1,+\infty)}(\neg\mathsf{p}) \wedge \overleftarrow{\Box}_{[-1,+\infty)}(\neg\mathsf{p}) \equiv \neg\mathsf{p}$	$\omega_2 = \mathsf{p} \Rightarrow \Box_{[1,+\infty)}(\neg\mathsf{p}) \wedge \overleftarrow{\Box}_{[1,+\infty)}(\neg\mathsf{p})$
(3)	$\mathrm{o}_3 = \mathsf{p} \Rightarrow \Diamond_{[k+1,k-1]}(\mathsf{q}) \equiv \neg\mathsf{p}$	$\omega_3 = \mathsf{p} \Rightarrow \Diamond_{[k,k]}(\mathsf{q})$

Table 4: Approximations of Formulas (1–3)

Now, consider the formulas ϕ^+ and ϕ^-, for any choice of k.

- $\phi^+ = \mathrm{Alw}(\omega_1) \wedge \mathrm{Alw}(\omega_2) \wedge \mathrm{Alw}(\omega_3) \Rightarrow \mathrm{Alw}(\mathrm{o}_3)$.
  o_3 requires p to be always false, but $\omega_1, \omega_2, \omega_3$ force p to happen exactly once. Therefore, ϕ^+ is not $\mathbb{Z}$-valid.

- $\phi^- = \mathrm{Alw}(\mathrm{o}_1) \wedge \mathrm{Alw}(\mathrm{o}_2) \wedge \mathrm{Alw}(\mathrm{o}_3) \Rightarrow \mathrm{Alw}(\omega_3)$.
  $\mathrm{o}_1, \mathrm{o}_2, \mathrm{o}_3$ are not satisfied by any model, as o_1 requires p to happen at least once, while $\mathrm{o}_2, \mathrm{o}_3$ force it to be always false. Therefore, ϕ^- is $\mathbb{Z}$-valid, being an implication with identically false antecedent.

Thus, in this case incompleteness arises and prevents us from concluding an obvious fact. We will see more concrete examples of these problems in Section 4, and how to mitigate them in practice.

[11] Recall that $\Box_{[-1,+\infty)}(\mathsf{p}) \equiv \overleftarrow{\Box}_{[0,1]}(\mathsf{p}) \wedge \Box_{[0,+\infty)}(\mathsf{p})$.

4 Examples and Experiments

This section presents two system verification problems (Section 4.1) and reports the results obtained in solving them using the tool presented in Section 2.3 (Section 4.2).

4.1 Examples

It was not possible to satisfactorily perform the same experiments as in other approaches to discretization (see Section 1.1), as every formal notation has its own peculiarities; thus, often the literal translation of a specification into another, different, notation results clumsy. In particular, this was the case in translating the examples in duration calculus of [SPC05], so we preferred to devise examples of our own.

4.1.1 The Controlled Reservoir

The first example is a simple model of a controlled reservoir, a similar example of which has also been presented in [FR06].

System specification. The system consists of a reservoir and a controller. The reservoir can nondeterministically leak (predicate L) and being filled with new liquid by the controller (predicate F). The level of fluid in the reservoir is described by two predicates $\ell \geq \mathsf{thres}$ and $\ell \geq \mathsf{min}$: the latter holds when the lever of fluid is above a minimum level, and the former holds if the level is above a control threshold, assumed to be higher than the minimum.[12]

The system is formally described by the following formulas, where $\nu > 0$ is a parameter of the system description.

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

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

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

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

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

Formulas (4–6) describe the behavior of the fluid level under all combinations of filling and leaking; in particular, Formula (5) corresponds to the assumption that the level, even when leaking at the maximum rate, cannot drop from above the control threshold to below the minimum in ν time units. On the other hand, Formula (8) describes the control action: filling is triggered as soon as the level goes below the control threshold. Finally, Formula (7) simply expresses the assumption that the control threshold is higher than the minimum.

[12] For notational convenience, we abbreviate $\neg(\ell \geq \mathsf{min})$ and $\neg(\ell \geq \mathsf{thres})$ by $\ell < \mathsf{min}$ and $\ell < \mathsf{thres}$, respectively.

Notice that we put to good use the rule of thumb that emerged from the example in Section 3.4.2: all exact time distances are written with a $\Box$ operator if unnegated (in normal form, that is representing $\Rightarrow$ with $\vee$ and $\neg$), and with a $\Diamond$ otherwise. This avoids degenerate intervals in computing the over-approximation. Another thing we notice is that choosing an open or a closed interval in duration sub-formulas such as $\Box_{(0,\nu)}(\mathsf{F})$ makes no difference in this case, as they are under negation; therefore, they correspond to U sub-formulas in normal form, whose approximations do not distinguish between open or closed intervals.

System property. The overall goal of the controller is to maintain the level above the minimum, that is to keep $\ell \geq \mathsf{min}$ always true. Therefore, we would like to verify the following property: after the system is "initialized" by setting the level above the control threshold, the level stays above the minimum forever.

$$\ell \geq \mathsf{thres} \;\Rightarrow\; \Box_{[\nu,+\infty)}(\ell \geq \mathsf{min}) \tag{9}$$

Notice that the stronger property $\ell \geq \mathsf{thres} \Rightarrow \Box_{[0,+\infty)}(\ell \geq \mathsf{min})$ does not hold for the above axiomatization. In fact, Formulas (4–8) relate the value of the level after ν time units with that at the current instant; nothing is said explicitly about what happens in between the current instant and ν. Obviously, one could work around this by modifying the property or the specification; for brevity we omit discussing these variations.

Verification. For any given ν, Formula (9) is a property of the system if and only if we pick the sampling period $\delta = \nu$. Otherwise the property does not hold since the sampling period is "too short" with respect to ν. In fact, if $\delta = \nu$, the predicates are allowed to change their values at most once every two discrete time steps; thus, Formulas (4–6) can be applied as any behavior satisfies one of their antecedents. On the contrary, if $\delta \neq \nu$ it must be $\delta < \nu$; in this case, the predicates can change more than once every two discrete time steps. Thus, there exist behaviors that fail to satisfy all of the antecedents of Formulas (4–6); then, the system description is too weak to force the invariance of the level in this case.

4.1.2 The Coffee Machine

The second example consists in the description of a coffee machine, in operational fashion.

States and events in dense time. In order to render an operational specification with a descriptive formalism, we introduce some *ontological constructs* known as *states* and *events*; their use simplifies the system description and the reasoning about the system in dense time, as discussed extensively in [GM01]. Let us briefly recall their definitions; in the remainder, we will use them in the coffee machine specification.

First of all, let us notice the following: for behaviors that are non-Zeno, the definitions of the $\bigcirc$ and $\overleftarrow{\bigcirc}$ operators coincide with that of the TRIO NowOn and UpToNow operators. Namely, $\bigcirc(\mathsf{p})$ holds iff there exists a positive length $\epsilon > 0$ such that p is true on the (relative) open interval $(0, \epsilon)$; $\overleftarrow{\bigcirc}$ is the dual for the past. χ-regular behaviors are in particular non-Zeno, as discussed in Section 2.2; therefore the definitions for $\bigcirc$ and $\overleftarrow{\bigcirc}$ we introduced in Table 1 are well-founded with respect to our needs.

A *state* is a predicate whose truth value holds over non-empty intervals of time; formally, p is a state when it obeys the following.

$$(\overleftarrow{\bigcirc}(\mathsf{p}) \wedge \mathsf{p} \wedge \bigcirc(\mathsf{p})) \vee (\overleftarrow{\bigcirc}(\neg\mathsf{p}) \wedge \neg\mathsf{p} \wedge \bigcirc(\neg\mathsf{p})) \vee (\overleftarrow{\bigcirc}(\neg\mathsf{p}) \wedge \bigcirc(\mathsf{p})) \vee (\overleftarrow{\bigcirc}(\mathsf{p}) \wedge \bigcirc(\neg\mathsf{p})) \tag{10}$$

An *event* is instead a predicate that holds only at isolated time instants; formally p is an event when it obeys the following.

$$\overleftarrow{\bigcirc}(\neg\mathsf{p}) \ \wedge \ \bigcirc(\neg\mathsf{p}) \tag{11}$$

Typically, in a system description with states and events, events trigger states to change. However, if we consider χ-regular behaviors, it is apparent that Formula (11) is satisfied only by predicates that are identically false, whereas Formula (10) is identically true for any predicate. Therefore, a "classic" description based on states and events does not fit our techniques, and in particular the restriction to χ-regular behaviors. We will show how to work around this shortcoming and produce acceptable specifications under this assumption.

System description with states and events. Let us forget for a moment the restriction to χ-regular behaviors, and let us produce a formalization of the coffee machine based on states and events. We introduce the events: prepare_cup, press_button, start_pour, end_pour, 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 states: pour, cup_present, coffee_ready, and 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. Finally, we introduce three constants $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$ to describe the various delays in the operations.

The following are the formulas describing (part of the behavior) of the coffee machine. For simplicity, we introduce only those that are needed to verify the properties we are interested in. It is not difficult, however, to extend them to get a "complete" (in some sense) specification.

$$\mathsf{prepare\_cup} \;\Rightarrow\; \overleftarrow{\Diamond}_{(0,\mathrm{T}_1)}(\mathsf{press\_button}) \tag{12}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \overleftarrow{\Diamond}_{(0,\mathrm{T}_2)}(\mathsf{prepare\_cup}) \tag{13}$$

$$\mathsf{end\_pour} \;\Rightarrow\; \overleftarrow{\Box}_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{start\_pour}) \wedge \overleftarrow{\Box}_{[0,\mathrm{T}_3)}(\mathsf{pour}) \tag{14}$$

$$\mathsf{press\_button} \;\Rightarrow\; \mathsf{key\_in} \tag{15}$$

$$\neg\mathsf{pour} \wedge \bigcirc(\mathsf{pour}) \;\Leftrightarrow\; \mathsf{start\_pour} \tag{16}$$

$$\Box_{(0,\mathrm{T}_3)}(\mathsf{pour}) \;\Rightarrow\; \mathsf{start\_pour} \wedge \Box_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{end\_pour}) \tag{17}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \mathsf{cup\_present} \wedge \neg\mathsf{coffee\_ready} \tag{18}$$

$$\mathsf{cup\_present} \wedge \bigcirc(\neg\mathsf{cup\_present}) \;\Leftrightarrow\; \mathsf{get\_cup} \tag{19}$$

$$\mathsf{get\_cup} \;\Rightarrow\; \mathsf{coffee\_ready} \tag{20}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \Box_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{end\_pour}) \wedge \Box_{(0,\mathrm{T}_3]}(\mathsf{pour}) \tag{21}$$

The properties that we prove from the above formulas are the following

$$\mathsf{end\_pour} \;\Rightarrow\; \overleftarrow{\Diamond}_{[\mathrm{T}_3,\mathrm{T}_1+\mathrm{T}_2+\mathrm{T}_3]}(\mathsf{key\_in}) \tag{22}$$

$$\mathsf{pour} \;\Rightarrow\; \mathsf{cup\_present} \tag{23}$$

In practice, (22) states that the pouring ends only if a key was inserted in the past (between T_3 and $\mathrm{T}_1 + \mathrm{T}_2 + \mathrm{T}_3$ time units ago); and (23) asserts that a cup is present while the coffee is being poured. It can be shown that from (12–15) alone one can prove (22), and from (16–21) one can prove (23).

System description for χ-regular behaviors. Let us now modify Formulas (12–23) to make them a suitable system description still over dense time, but for χ-regular behaviors; at the same time we also make them amenable to applying our discretization technique.

Formulas (15), (18), (20), (22), and (23) make no peculiar use of the state and event properties, and therefore they need not be changed. Therefore, they are repeated unchanged as Formulas (24), (25), (26), (27), and (28), respectively.

$$\mathsf{press\_button} \;\Rightarrow\; \mathsf{key\_in} \tag{24}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \mathsf{cup\_present} \wedge \neg\mathsf{coffee\_ready} \tag{25}$$

$$\mathsf{get\_cup} \;\Rightarrow\; \mathsf{coffee\_ready} \tag{26}$$

$$\mathsf{end\_pour} \;\Rightarrow\; \overleftarrow{\Diamond}_{[\mathrm{T}_3,\mathrm{T}_1+\mathrm{T}_2+\mathrm{T}_3]}(\mathsf{key\_in}) \tag{27}$$

$$\mathsf{pour} \;\Rightarrow\; \mathsf{cup\_present} \tag{28}$$

Next, let us consider Formula (16); similar considerations apply to the structurally similar Formula (19) *mutatis mutandis*. Formula (16) describes a transition of state pour from false to true, triggered by event start_pour: pour becomes true left-continuously, whenever start_pour happens. We can render a similar

state transition over χ-regular behaviors as follows: pour is false and becomes true "after some time" whenever start_pour happens. In other words, we do not make the state transition instantaneous, but we allow "some time" for it to happen. We do this in practice by introducing a positive constant ν in our specification that quantifies this notion of "some time". Then, Formula (16) can be rewritten as $\neg\mathsf{pour} \wedge \Diamond_{[\nu,\nu]}(\mathsf{pour}) \Leftrightarrow \mathsf{start\_pour}$; more precisely we split the double implication into two and specify the exact distance of ν with a $\Box$ when unnegated, and with a $\Diamond$ otherwise. The resulting Formulas (29–30) are listed below (together with Formulas (31–32) replacing Formula (19))

$$\neg\mathsf{pour} \wedge \Diamond_{[\nu,\nu]}(\mathsf{pour}) \;\Rightarrow\; \mathsf{start\_pour} \tag{29}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \neg\mathsf{pour} \wedge \Box_{[\nu,\nu]}(\mathsf{pour}) \tag{30}$$

$$\mathsf{cup\_present} \wedge \Diamond_{[\nu,\nu]}(\neg\mathsf{cup\_present}) \;\Rightarrow\; \mathsf{get\_cup} \tag{31}$$

$$\mathsf{get\_cup} \;\Rightarrow\; \mathsf{cup\_present} \wedge \Box_{[\nu,\nu]}(\neg\mathsf{cup\_present}) \tag{32}$$

Finally, we have to deal with the remaining Formulas (12), (13), (17), (14), and (21), and more precisely with the open (or half-open) intervals appearing in them, such as $(0, \mathrm{T}_1)$ in Formula (12).

There are two separate issues to be tackled here. One is simply a feature of our implementation of the algorithm: it requires all intervals to be expressed as closed ones. This is without loss of generality, as an open interval over discrete time can always be expressed as closed. Thus, when dealing with open intervals of *release* or *trigger* operators, and more precisely when computing under-approximations of them, we subtract ν time units on each open side of an open interval, changing it to a closed one. For instance, the interval $[0, \mathrm{T}_3)$ is changed to $[0, \mathrm{T}_3 - \nu]$ in Formula (14). The second problem, instead, is still linked to the state/event description of the system that we rely upon in dense time. In particular, consider Formula (17): there, in normal form, the open interval $(0, \mathrm{T}_3)$ would belong to an *until* operator, and therefore its being open would make no difference in computing the approximations. However, the resulting approximation would be one with too strong (i.e., with too long an interval) an antecedent, in that it would be false in all but trivial cases. This would weaken the overall specification, and would lead us more often to hit into the incompleteness of the verification technique (in particular, in proving Formula (23)). Therefore, we change the open interval $(0, \mathrm{T}_3)$ to the closed interval $[\nu, \mathrm{T}_3 - \nu]$, thus achieving better verification results. Informally, this corresponds to a strengthening of the resulting discretized specification. Finally, for uniformity we do the same with the open intervals of Formulas (12–13), even if in this case experiments have shown that we have basically the same performances whether we shift the end-points or not. All the resulting formulas

are as follows.

$$\mathsf{prepare\_cup} \;\Rightarrow\; \overleftarrow{\Diamond}_{[\nu,\mathrm{T}_1-\nu]}(\mathsf{press\_button}) \tag{33}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \overleftarrow{\Diamond}_{[\nu,\mathrm{T}_2-\nu]}(\mathsf{prepare\_cup}) \tag{34}$$

$$\Box_{[\nu,\mathrm{T}_3-\nu]}(\mathsf{pour}) \;\Rightarrow\; \mathsf{start\_pour} \wedge \Box_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{end\_pour}) \tag{35}$$

$$\mathsf{end\_pour} \;\Rightarrow\; \overleftarrow{\Box}_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{start\_pour}) \wedge \overleftarrow{\Box}_{[0,\mathrm{T}_3-\nu]}(\mathsf{pour}) \tag{36}$$

$$\mathsf{start\_pour} \;\Rightarrow\; \Box_{[\mathrm{T}_3,\mathrm{T}_3]}(\mathsf{end\_pour}) \wedge \Box_{[\nu,\mathrm{T}_3]}(\mathsf{pour}) \tag{37}$$

Verification. When the parameters are chosen to be all integer-valued, if $\delta = 1$, then the verification is successful, that is (27–28) are properties of the specified system. If $\delta \neq 1$, then the properties may not hold, also depending on the particular values for the constants $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$ (and if they may yield degenerate intervals). Notice however that the choice $\delta = 1$ is natural, in the sense that we take a sampling period which is of the same size (or smaller) as the "change delays" in the system.

4.2 Experiments

The following Tables 5–10 report the results obtained in an array of tests with the discretization techniques and Zot. For each test we report: the value k of the bound given to Zot (in other words, the size of the explored space); the value of the parameter ν in the models; the value of δ, according to which the discretizations are built; the value of other parameters in the models (in the case of the coffee machine, they are $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$); the outcomes of (1) the bound completeness test (for the given k) on the systems (i.e., if the check is exhaustive for the system), and (2) the validity check for the properties to be checked (namely, Formula (9) for the reservoir example, and Formulas (22–23) (denoted in the tables as TH_1 and TH_2, respectively) for the coffee machine example. Each test is done both over mono-infinite domain, and over bi-infinite domain. For each test we report the outcome ($\top$ means valid, $\bot$ means non-valid, $\sim$ means that the approximation technique has been inconclusive, ∞ means that the process was killed after running for more than 24 hours), the real time ("wall-time") and the total (maximum) amount of memory taken in the whole verification 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. Zot used GNU CLisp v. 2.39, and Minisat v. 1.14 as SAT-solving engine. Below, we briefly comment on the outcomes with the two examples.

Reservoir example. Tables 5–6 report the results obtained with the reservoir example.

The example is a simple one, and in fact the results are highly predictable and satisfactory. The diameter of the example is small, and it is reached already with $k \geq 10$. We never obtain inconclusive results in applying our discretization

k	ν	δ	COMPLETE (TIME / MEM)	OUTCOME (TIME / MEM)
5	1	1	$\bot$ (0.4 s / 2.1 Mb)	$\top$ (0.5 s / 2.2 Mb)
5	1	1/3	$\bot$ (0.9 s / 4.3 Mb)	$\bot$ (0.7 s / 5.8 Mb)
5	5	5	$\bot$ (0.2 s / 2.1 Mb)	$\top$ (0.2 s / 2.2 Mb)
5	5	5/3	$\bot$ (0.4 s / 4.3 Mb)	$\bot$ (0.7 s / 5.8 Mb)
5	10	10	$\bot$ (0.2 s / 2.1 Mb)	$\top$ (0.2 s / 2.2 Mb)
5	10	10/3	$\bot$ (0.4 s / 4.3 Mb)	$\bot$ (0.7 s / 5.8 Mb)
5	20	20	$\bot$ (0.2 s / 2.1 Mb)	$\top$ (0.3 s / 2.3 Mb)
5	20	20/3	$\bot$ (0.5 s / 4.3 Mb)	$\bot$ (0.8 s / 5.8 Mb)
10	1	1	$\top$ (0.6 s / 2.9 Mb)	$\top$ (0.6 s / 3.2 Mb)
10	1	1/3	$\top$ (2.3 s / 6 Mb)	$\bot$ (1.6 s / 12.1 Mb)
10	5	5	$\top$ (0.6 s / 2.9 Mb)	$\top$ (0.6 s / 3.2 Mb)
10	5	5/3	$\top$ (2.3 s / 6 Mb)	$\bot$ (1.6 s / 12.1 Mb)
10	10	10	$\top$ (0.6 s / 2.9 Mb)	$\top$ (0.6 s / 3.2 Mb)
10	10	10/3	$\top$ (2.2 s / 6 Mb)	$\bot$ (1.6 s / 12.1 Mb)
10	20	20	$\top$ (0.6 s / 2.9 Mb)	$\top$ (0.6 s / 3.2 Mb)
10	20	20/3	$\top$ (2 s / 6 Mb)	$\bot$ (1.2 s / 12.1 Mb)
50	1	1	$\top$ (4.1 s / 25.6 Mb)	$\top$ (2.8 s / 15.3 Mb)
50	1	1/3	$\top$ (12.4 s / 53.3 Mb)	$\bot$ (8.5 s / 110.8 Mb)
50	5	5	$\top$ (4.3 s / 25.6 Mb)	$\top$ (2.6 s / 15.3 Mb)
50	5	5/3	$\top$ (11 s / 53.3 Mb)	$\bot$ (8.6 s / 110.8 Mb)
50	10	10	$\top$ (4.2 s / 25.6 Mb)	$\top$ (2.8 s / 15.3 Mb)
50	10	10/3	$\top$ (11.1 s / 53.3 Mb)	$\bot$ (8.6 s / 110.8 Mb)
50	20	20	$\top$ (3.8 s / 25.6 Mb)	$\top$ (2.9 s / 15.3 Mb)
50	20	20/3	$\top$ (11.1 s / 53.3 Mb)	$\bot$ (8.6 s / 110.8 Mb)
100	1	1	$\top$ (20.7 s / 83.4 Mb)	$\top$ (9.6 s / 30.4 Mb)
100	1	1/3	$\top$ (52.6 s / 171.6 Mb)	$\bot$ (30.6 s / 361.9 Mb)
100	5	5	$\top$ (21.3 s / 83.4 Mb)	$\top$ (9.6 s / 30.4 Mb)
100	5	5/3	$\top$ (50.3 s / 171.6 Mb)	$\bot$ (30.6 s / 361.9 Mb)
100	10	10	$\top$ (21.2 s / 83.4 Mb)	$\top$ (9.5 s / 30.4 Mb)
100	10	10/3	$\top$ (50 s / 171.6 Mb)	$\bot$ (29.9 s / 361.9 Mb)
100	20	20	$\top$ (21.6 s / 83.4 Mb)	$\top$ (9.4 s / 30.4 Mb)
100	20	20/3	$\top$ (49.5 s / 171.6 Mb)	$\bot$ (30.1 s / 361.9 Mb)
200	1	1	$\top$ (150.9 s / 297.4 Mb)	$\top$ (33.9 s / 60.7 Mb)
200	1	1/3	$\top$ (344.9 s / 604.6 Mb)	$\bot$ (106.9 s / 1240.1 Mb)
200	5	5	$\top$ (152.1 s / 297.4 Mb)	$\top$ (33.9 s / 60.7 Mb)
200	5	5/3	$\top$ (345.6 s / 604.6 Mb)	$\bot$ (107.9 s / 1240.1 Mb)
200	10	10	$\top$ (153 s / 297.4 Mb)	$\top$ (33.3 s / 60.7 Mb)
200	10	10/3	$\top$ (346.1 s / 604.6 Mb)	$\bot$ (108.6 s / 1240.1 Mb)
200	20	20	$\top$ (154.2 s / 297.4 Mb)	$\top$ (33.4 s / 60.7 Mb)
200	20	20/3	$\top$ (342.3 s / 604.6 Mb)	$\bot$ (111.8 s / 1240.1 Mb)

Table 5: Results of the reservoir example over mono-infinite.

k	ν	δ	COMPLETE (TIME / MEM)	OUTCOME (TIME / MEM)
5	1	1	$\bot$ (0.2 s / 2.2 Mb)	$\top$ (0.3 s / 2.6 Mb)
5	1	1/3	$\bot$ (0.6 s / 4.8 Mb)	$\bot$ (1.2 s / 9 Mb)
5	5	5	$\bot$ (0.3 s / 2.2 Mb)	$\top$ (0.3 s / 2.6 Mb)
5	5	5/3	$\bot$ (0.6 s / 4.8 Mb)	$\bot$ (1.2 s / 9 Mb)
5	10	10	$\bot$ (0.3 s / 2.2 Mb)	$\top$ (0.4 s / 2.6 Mb)
5	10	10/3	$\bot$ (0.6 s / 4.8 Mb)	$\bot$ (1.2 s / 9 Mb)
5	20	20	$\bot$ (0.3 s / 2.2 Mb)	$\top$ (0.5 s / 2.6 Mb)
5	20	20/3	$\bot$ (0.5 s / 4.8 Mb)	$\bot$ (1 s / 9 Mb)
10	1	1	$\top$ (0.5 s / 3.3 Mb)	$\top$ (0.7 s / 4.9 Mb)
10	1	1/3	$\top$ (1.9 s / 7.3 Mb)	$\bot$ (2.2 s / 18.1 Mb)
10	5	5	$\top$ (0.5 s / 3.3 Mb)	$\top$ (0.8 s / 4.9 Mb)
10	5	5/3	$\top$ (2.2 s / 7.3 Mb)	$\bot$ (1.8 s / 18.1 Mb)
10	10	10	$\top$ (0.5 s / 3.3 Mb)	$\top$ (0.7 s / 4.9 Mb)
10	10	10/3	$\top$ (1.9 s / 7.3 Mb)	$\bot$ (2 s / 18.1 Mb)
10	20	20	$\top$ (0.5 s / 3.3 Mb)	$\top$ (0.7 s / 4.9 Mb)
10	20	20/3	$\top$ (2.2 s / 7.3 Mb)	$\bot$ (2.5 s / 18.1 Mb)
50	1	1	$\top$ (4.2 s / 27.7 Mb)	$\top$ (6.1 s / 23.5 Mb)
50	1	1/3	$\top$ (12.4 s / 59.7 Mb)	$\bot$ (18.8 s / 149.5 Mb)
50	5	5	$\top$ (4.1 s / 27.7 Mb)	$\top$ (5.5 s / 23.5 Mb)
50	5	5/3	$\top$ (12.6 s / 59.7 Mb)	$\bot$ (19 s / 149.5 Mb)
50	10	10	$\top$ (4.1 s / 27.7 Mb)	$\top$ (5.3 s / 23.5 Mb)
50	10	10/3	$\top$ (12.3 s / 59.7 Mb)	$\bot$ (19.5 s / 149.5 Mb)
50	20	20	$\top$ (4.4 s / 27.7 Mb)	$\top$ (5.4 s / 23.5 Mb)
50	20	20/3	$\top$ (12.4 s / 59.7 Mb)	$\bot$ (18.9 s / 149.5 Mb)
100	1	1	$\top$ (22.2 s / 87.3 Mb)	$\top$ (19.8 s / 46.7 Mb)
100	1	1/3	$\top$ (58.3 s / 183.6 Mb)	$\bot$ (66.2 s / 470.6 Mb)
100	5	5	$\top$ (21.9 s / 87.3 Mb)	$\top$ (20.5 s / 46.7 Mb)
100	5	5/3	$\top$ (56.9 s / 183.6 Mb)	$\bot$ (65.4 s / 470.6 Mb)
100	10	10	$\top$ (21.6 s / 87.3 Mb)	$\top$ (20.2 s / 46.7 Mb)
100	10	10/3	$\top$ (56.6 s / 183.6 Mb)	$\bot$ (66.1 s / 470.6 Mb)
100	20	20	$\top$ (22.5 s / 87.3 Mb)	$\top$ (19.9 s / 46.7 Mb)
100	20	20/3	$\top$ (57.5 s / 183.6 Mb)	$\bot$ (65.7 s / 470.6 Mb)
200	1	1	$\top$ (158.9 s / 304 Mb)	$\top$ (71.2 s / 93.1 Mb)
200	1	1/3	$\top$ (374.3 s / 626.1 Mb)	$\bot$ (245.1 s / 1588.3 Mb)
200	5	5	$\top$ (158.8 s / 304 Mb)	$\top$ (70.6 s / 93.1 Mb)
200	5	5/3	$\top$ (374 s / 626.1 Mb)	$\bot$ (244.8 s / 1588.3 Mb)
200	10	10	$\top$ (159.7 s / 304 Mb)	$\top$ (72.6 s / 93.1 Mb)
200	10	10/3	$\top$ (370 s / 626.1 Mb)	$\bot$ (245.1 s / 1588.3 Mb)
200	20	20	$\top$ (159.2 s / 304 Mb)	$\top$ (71.4 s / 93.1 Mb)
200	20	20/3	$\top$ (367 s / 626.1 Mb)	$\bot$ (244.2 s / 1588.3 Mb)

Table 6: Results of the reservoir example over bi-infinite.

technique, and Formula (9) is confirmed to be valid if and only if $\nu = \delta$. The times and spaces required to obtain the results (or to check the bound completeness) are always relatively small, and they scale rather well with the increase of the bound. This is also a predictable result, as the complexity of the validity check is relatively insensitive to the increase of k, whereas it strongly depends on the timing constants in the formulas (always small in the case of the reservoir, as they are either 1 or 3, i.e., ν/δ). Finally, notice that it usually takes a shorter time to check the validity than to check the non-validity; this is also obvious, as the latter requires to submit two formulas to the validity checker (ϕ^+ and ϕ^-), while the former just checks one formula.

Coffee machine example. Tables 7–10 report the results obtained with the coffee machine example; they are more variegated than the simple reservoir.

First of all, notice that the validity of Formula (22) is robust with respect to the choices of parameters. Indeed, Formula (22) (tagged TH_1 in the tables) is shown to be valid for all the choices of parameters that we made in the experiments. The times needed to get this result are rather short, and scale with the length of k. This is reasonable, as the main factor in determining the length of the check are the parameters $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$ that however stay within a short span of values in our tests.

The outcomes of the validity check of Formula (23) are, on the other hand, more varied. First of all, as we asserted in presenting the example, if $\delta = 1$ then Formula (23) is valid for the system; this is confirmed by all our tests. If, instead, $\delta \neq 1$, then the results timing constants interact more subtly, and it is more difficult to predict the outcome of the verification. In some cases we fall in the incompleteness area of our method, and we get inconclusive results from analyzing the approximations. Notice that this tends to happen a bit more often with the bi-infinite tool, probably due to (the absence of) some border effects.

Finally, let us consider the bound completeness check. Unfortunately, but inevitably, the time required for the completeness check grows very quickly with the value of k. In practice, it tends to diverge for values of about 40 (or less, in the bi-infinite case, where it is usually more complex). Of course, this is a feature inherent to the bound completeness check itself, which is orthogonal to our discretization techniques. Experiments with the same specification but with a different discrete-time verification tool, such as [MPSS03, Spo05], belong to future work.

5 Conclusions

We presented a technique to reduce the validity problem for dense-time MTL formulas to the corresponding validity problem over discrete-time models. The technique is based on the notions of *sampling* and *sampling invariance*, introduced in [FR06]. In a nutshell, we perform simple syntactic transformations on the MTL formulas that we want to check for validity; the resulting formulas retain properties of discrete-time samplings of the dense-time behaviors the

k	ν	δ	T_1, T_2, T_3	COMPLETE (TIME / MEM)	TH_1 OUTCOME (TIME / MEM)	TH_2 OUTCOME (TIME / MEM)
5	1	1	4,4,4	$\bot$ (1.2 s / 5.8 Mb)	$\top$ (1.9 s / 6.2 Mb)	$\top$ (0.7 s / 5.2 Mb)
5	1	1	7,6,5	$\bot$ (0.6 s / 6.9 Mb)	$\top$ (1.3 s / 7.6 Mb)	$\top$ (0.8 s / 6 Mb)
5	1	1	10,7,8	$\bot$ (0.9 s / 8.8 Mb)	$\top$ (1.7 s / 9.6 Mb)	$\top$ (1 s / 7.4 Mb)
5	1	1/3	4,4,4	$\bot$ (0.9 s / 9.9 Mb)	$\top$ (2.2 s / 11.8 Mb)	$\top$ (1.3 s / 8.9 Mb)
5	1	1/3	7,6,5	$\bot$ (1.1 s / 14 Mb)	$\top$ (2.8 s / 15.8 Mb)	$\top$ (1.6 s / 11.2 Mb)
5	1	1/3	10,7,8	$\bot$ (1.7 s / 20.7 Mb)	$\top$ (5.3 s / 21.9 Mb)	$\top$ (2.8 s / 15.4 Mb)
5	2	1	4,4,4	$\bot$ (0.5 s / 5.5 Mb)	$\top$ (1.1 s / 6.1 Mb)	$\top$ (0.7 s / 5 Mb)
5	2	1	7,6,5	$\bot$ (0.7 s / 6.2 Mb)	$\top$ (1.3 s / 7.4 Mb)	$\top$ (1 s / 5.8 Mb)
5	2	1	10,7,8	$\bot$ (0.9 s / 8.7 Mb)	$\top$ (1.7 s / 9.5 Mb)	$\top$ (0.9 s / 7.2 Mb)
5	2	1/3	4,4,4	$\bot$ (0.8 s / 8.6 Mb)	$\top$ (1.9 s / 11.3 Mb)	$\top$ (1.2 s / 8.3 Mb)
5	2	1/3	7,6,5	$\bot$ (1.1 s / 12.4 Mb)	$\top$ (3.4 s / 15.3 Mb)	$\top$ (1.7 s / 10.7 Mb)
5	2	1/3	10,7,8	$\bot$ (1.4 s / 19 Mb)	$\top$ (5.3 s / 21.3 Mb)	$\top$ (2.4 s / 14.8 Mb)
5	3	1	4,4,4	$\top$ (0.4 s / 3.7 Mb)	$\top$ (0.8 s / 5.9 Mb)	$\top$ (0.8 s / 4.8 Mb)
5	3	1	7,6,5	$\top$ (0.7 s / 4.2 Mb)	$\top$ (1.3 s / 7.2 Mb)	$\top$ (0.7 s / 5.6 Mb)
5	3	1	10,7,8	$\bot$ (0.7 s / 7.8 Mb)	$\top$ (1.2 s / 9.3 Mb)	$\top$ (0.9 s / 7 Mb)
5	3	1/3	4,4,4	$\top$ (0.7 s / 5.3 Mb)	$\top$ (1.8 s / 10.7 Mb)	$\top$ (1.1 s / 7.8 Mb)
5	3	1/3	7,6,5	$\top$ (1.1 s / 6.7 Mb)	$\top$ (3 s / 14.6 Mb)	$\top$ (1.5 s / 10 Mb)
5	3	1/3	10,7,8	$\bot$ (1.4 s / 18.5 Mb)	$\top$ (5.4 s / 20.8 Mb)	$\top$ (2.3 s / 14.3 Mb)
10	1	1	4,4,4	$\bot$ (1.1 s / 13.7 Mb)	$\top$ (1.8 s / 11.7 Mb)	$\top$ (1.3 s / 9.8 Mb)
10	1	1	7,6,5	$\bot$ (1.2 s / 15.1 Mb)	$\top$ (2.2 s / 14.3 Mb)	$\top$ (1.6 s / 11.2 Mb)
10	1	1	10,7,8	$\bot$ (1.7 s / 22.1 Mb)	$\top$ (3.4 s / 18 Mb)	$\top$ (2.1 s / 13.8 Mb)
10	1	1/3	4,4,4	$\bot$ (1.8 s / 26.8 Mb)	$\top$ (5.3 s / 22.1 Mb)	$\top$ (2.8 s / 16.6 Mb)
10	1	1/3	7,6,5	$\bot$ (2.4 s / 41.8 Mb)	$\top$ (9 s / 29.5 Mb)	$\top$ (4.5 s / 20.9 Mb)
10	1	1/3	10,7,8	$\bot$ (3.5 s / 64.3 Mb)	$\top$ (17.1 s / 40.5 Mb)	$\top$ (8.7 s / 28.6 Mb)
10	2	1	4,4,4	$\bot$ (1 s / 12.9 Mb)	$\top$ (1.7 s / 11.4 Mb)	$\top$ (1.4 s / 9.5 Mb)
10	2	1	7,6,5	$\bot$ (1.2 s / 14.7 Mb)	$\top$ (2.2 s / 14 Mb)	$\sim$ (3.5 s / 30.1 Mb)
10	2	1	10,7,8	$\bot$ (1.4 s / 20.1 Mb)	$\top$ (3.3 s / 17.7 Mb)	$\sim$ (4.4 s / 40 Mb)
10	2	1/3	4,4,4	$\bot$ (1.7 s / 21 Mb)	$\top$ (4.9 s / 21.1 Mb)	$\top$ (2.6 s / 15.6 Mb)
10	2	1/3	7,6,5	$\bot$ (2.3 s / 32.3 Mb)	$\top$ (8.5 s / 28.5 Mb)	$\top$ (4.1 s / 19.9 Mb)
10	2	1/3	10,7,8	$\bot$ (3.3 s / 50.1 Mb)	$\top$ (16.2 s / 39.5 Mb)	$\top$ (8.1 s / 27.6 Mb)
10	3	1	4,4,4	$\top$ (1 s / 7.3 Mb)	$\top$ (1.6 s / 11 Mb)	$\top$ (1.3 s / 9.1 Mb)
10	3	1	7,6,5	$\top$ (1 s / 8.2 Mb)	$\top$ (2 s / 13.5 Mb)	$\top$ (1.5 s / 10.5 Mb)
10	3	1	10,7,8	$\bot$ (1.4 s / 18.2 Mb)	$\top$ (3.1 s / 17.4 Mb)	$\sim$ (4.2 s / 35.7 Mb)
10	3	1/3	4,4,4	$\top$ (1.4 s / 10.3 Mb)	$\top$ (4.2 s / 20 Mb)	$\top$ (2.5 s / 14.5 Mb)
10	3	1/3	7,6,5	$\top$ (1.9 s / 12.8 Mb)	$\top$ (8 s / 27.2 Mb)	$\top$ (3.8 s / 18.6 Mb)
10	3	1/3	10,7,8	$\bot$ (3.4 s / 46.4 Mb)	$\top$ (15.7 s / 38.5 Mb)	$\top$ (7.8 s / 26.6 Mb)
20	1	1	4,4,4	$\bot$ (2.6 s / 38.2 Mb)	$\top$ (5.1 s / 22.7 Mb)	$\top$ (3.6 s / 18.9 Mb)
20	1	1	7,6,5	$\bot$ (2.9 s / 44.5 Mb)	$\top$ (7.9 s / 27.6 Mb)	$\top$ (4.6 s / 21.7 Mb)
20	1	1	10,7,8	$\bot$ (3.9 s / 59 Mb)	$\top$ (12.6 s / 34.8 Mb)	$\top$ (7.4 s / 26.7 Mb)
20	1	1/3	4,4,4	$\bot$ (4.8 s / 77.2 Mb)	$\top$ (17.8 s / 42.6 Mb)	$\bot$ (22.1 s / 151.7 Mb)
20	1	1/3	7,6,5	$\bot$ (7.5 s / 105.1 Mb)	$\top$ (30.6 s / 56.8 Mb)	$\bot$ (33.3 s / 173.1 Mb)
20	1	1/3	10,7,8	$\bot$ (12 s / 201.5 Mb)	$\top$ (55.5 s / 77.7 Mb)	$\top$ (28.5 s / 55 Mb)
20	2	1	4,4,4	$\bot$ (2.5 s / 36.4 Mb)	$\top$ (4.9 s / 22 Mb)	$\top$ (3.3 s / 18.3 Mb)
20	2	1	7,6,5	$\bot$ (3 s / 41.7 Mb)	$\top$ (7.4 s / 27 Mb)	$\sim$ (9.4 s / 65.6 Mb)
20	2	1	10,7,8	$\bot$ (3.6 s / 56.8 Mb)	$\top$ (11.8 s / 34.2 Mb)	$\sim$ (14.6 s / 80.3 Mb)
20	2	1/3	4,4,4	$\bot$ (4.3 s / 58 Mb)	$\top$ (16.5 s / 40.7 Mb)	$\sim$ (17.7 s / 92.3 Mb)
20	2	1/3	7,6,5	$\bot$ (6.3 s / 93.2 Mb)	$\top$ (28.9 s / 54.9 Mb)	$\bot$ (31.2 s / 168.6 Mb)
20	2	1/3	10,7,8	$\bot$ (11.5 s / 179 Mb)	$\top$ (53.4 s / 75.8 Mb)	$\top$ (27.4 s / 53.1 Mb)
20	3	1	4,4,4	$\top$ (2.3 s / 16.8 Mb)	$\top$ (4.9 s / 21.3 Mb)	$\top$ (3.1 s / 17.6 Mb)
20	3	1	7,6,5	$\top$ (2.7 s / 18.4 Mb)	$\top$ (6.7 s / 26.1 Mb)	$\top$ (4 s / 20.2 Mb)
20	3	1	10,7,8	$\bot$ (3.4 s / 51.7 Mb)	$\top$ (11.6 s / 33.6 Mb)	$\sim$ (13.8 s / 78.3 Mb)
20	3	1/3	4,4,4	$\top$ (3.6 s / 22.4 Mb)	$\top$ (15.2 s / 38.6 Mb)	$\top$ (8.5 s / 28 Mb)
20	3	1/3	7,6,5	$\top$ (5.2 s / 27.2 Mb)	$\top$ (27.1 s / 52.4 Mb)	$\top$ (13.2 s / 35.9 Mb)
20	3	1/3	10,7,8	$\bot$ (12.2 s / 152.9 Mb)	$\top$ (51.5 s / 73.9 Mb)	$\top$ (25.8 s / 51.2 Mb)
25	1	1	4,4,4	$\bot$ (3.7 s / 56.9 Mb)	$\top$ (7.7 s / 28.1 Mb)	$\top$ (5.4 s / 23.5 Mb)
25	1	1	7,6,5	$\bot$ (4.2 s / 67.4 Mb)	$\top$ (11.5 s / 34.3 Mb)	$\top$ (7.2 s / 27 Mb)
25	1	1	10,7,8	$\bot$ (5.8 s / 92.9 Mb)	$\top$ (18.5 s / 43.2 Mb)	$\top$ (11.1 s / 33.1 Mb)
25	1	1/3	4,4,4	$\bot$ (8.2 s / 118.2 Mb)	$\top$ (26.2 s / 52.9 Mb)	$\bot$ (33 s / 211.3 Mb)
25	1	1/3	7,6,5	$\bot$ (11.9 s / 151.8 Mb)	$\top$ (45.8 s / 70.4 Mb)	$\bot$ (50.1 s / 273.6 Mb)
25	1	1/3	10,7,8	$\bot$ (18.2 s / 310.1 Mb)	$\top$ (82.8 s / 96.3 Mb)	$\bot$ (93.5 s / 524.8 Mb)
25	2	1	4,4,4	$\bot$ (3.6 s / 52.9 Mb)	$\top$ (7.5 s / 27.4 Mb)	$\top$ (5.2 s / 22.7 Mb)
25	2	1	7,6,5	$\bot$ (4.1 s / 61.3 Mb)	$\top$ (11.2 s / 33.5 Mb)	$\sim$ (14.3 s / 86.1 Mb)
25	2	1	10,7,8	$\bot$ (5.3 s / 71.9 Mb)	$\top$ (17.9 s / 42.4 Mb)	$\sim$ (22.5 s / 118.9 Mb)
25	2	1/3	4,4,4	$\bot$ (7.1 s / 83.5 Mb)	$\top$ (24.6 s / 50.5 Mb)	$\sim$ (27.1 s / 128.5 Mb)
25	2	1/3	7,6,5	$\bot$ (10.5 s / 125.2 Mb)	$\top$ (43.3 s / 68.1 Mb)	$\bot$ (47 s / 242.2 Mb)
25	2	1/3	10,7,8	$\bot$ (17.6 s / 247.5 Mb)	$\top$ (80.3 s / 94 Mb)	$\bot$ (85.4 s / 374 Mb)
25	3	1	4,4,4	$\top$ (3.1 s / 22.7 Mb)	$\top$ (7.1 s / 26.5 Mb)	$\top$ (4.7 s / 21.8 Mb)
25	3	1	7,6,5	$\top$ (3.8 s / 24.6 Mb)	$\top$ (10.7 s / 32.4 Mb)	$\top$ (6.2 s / 25.1 Mb)
25	3	1	10,7,8	$\bot$ (5.1 s / 72.8 Mb)	$\top$ (17.5 s / 41.7 Mb)	$\sim$ (20.6 s / 105.1 Mb)
25	3	1/3	4,4,4	$\top$ (5.5 s / 29.5 Mb)	$\top$ (22.8 s / 47.9 Mb)	$\top$ (12.7 s / 34.7 Mb)
25	3	1/3	7,6,5	$\top$ (8.3 s / 35.5 Mb)	$\top$ (39.9 s / 64.9 Mb)	$\top$ (19.7 s / 44.5 Mb)
25	3	1/3	10,7,8	$\bot$ (16.6 s / 270.1 Mb)	$\top$ (77.2 s / 91.6 Mb)	$\bot$ (81.1 s / 337.1 Mb)
30	1	1	4,4,4	$\bot$ (5.5 s / 78.9 Mb)	$\top$ (11 s / 33.6 Mb)	$\top$ (7.7 s / 28.1 Mb)
30	1	1	7,6,5	$\bot$ (6.8 s / 95.5 Mb)	$\top$ (16.4 s / 40.9 Mb)	$\top$ (10.1 s / 32.2 Mb)
30	1	1	10,7,8	$\bot$ (7.8 s / 122.7 Mb)	$\top$ (25.6 s / 51.6 Mb)	$\top$ (15.1 s / 39.6 Mb)
30	1	1/3	4,4,4	$\bot$ (10.6 s / 148.2 Mb)	$\top$ (37 s / 63.1 Mb)	$\bot$ (45.9 s / 303.2 Mb)
30	1	1/3	7,6,5	$\bot$ (15.4 s / 225.2 Mb)	$\top$ (63.4 s / 84.1 Mb)	$\bot$ (70 s / 362.1 Mb)
30	1	1/3	10,7,8	$\bot$ (27.3 s / 424.1 Mb)	$\top$ (116.3 s / 114.9 Mb)	$\bot$ (129.8 s / 1043.3 Mb)
30	2	1	4,4,4	$\bot$ (87.3 s / 72.9 Mb)	$\top$ (10.4 s / 32.7 Mb)	$\top$ (7.1 s / 27.2 Mb)
30	2	1	7,6,5	$\bot$ (6 s / 84.7 Mb)	$\top$ (15.5 s / 40 Mb)	$\sim$ (20.7 s / 129 Mb)
30	2	1	10,7,8	$\bot$ (7.4 s / 108.4 Mb)	$\top$ (24.6 s / 50.7 Mb)	$\sim$ (30.2 s / 152.3 Mb)
30	2	1/3	4,4,4	$\bot$ (9.5 s / 112.5 Mb)	$\top$ (34 s / 60.3 Mb)	$\sim$ (37.4 s / 162.7 Mb)
30	2	1/3	7,6,5	$\bot$ (14 s / 178.5 Mb)	$\top$ (60 s / 81.3 Mb)	$\bot$ (64.6 s / 334.6 Mb)
30	2	1/3	10,7,8	$\bot$ (24.7 s / 339.4 Mb)	$\top$ (111.5 s / 112.1 Mb)	$\bot$ (118.4 s / 477.6 Mb)
30	3	1	4,4,4	$\top$ (4.3 s / 29.2 Mb)	$\top$ (9.7 s / 31.6 Mb)	$\top$ (6.8 s / 26.1 Mb)
30	3	1	7,6,5	$\top$ (5 s / 31.6 Mb)	$\top$ (14.4 s / 38.8 Mb)	$\top$ (8.8 s / 30.1 Mb)
30	3	1	10,7,8	$\bot$ (7.2 s / 101.6 Mb)	$\top$ (24.2 s / 49.7 Mb)	$\sim$ (28.6 s / 153.3 Mb)
30	3	1/3	4,4,4	$\top$ (7.8 s / 37.3 Mb)	$\top$ (31.2 s / 57.2 Mb)	$\top$ (17.3 s / 41.4 Mb)
30	3	1/3	7,6,5	$\top$ (11.5 s / 44.5 Mb)	$\top$ (55.4 s / 77.5 Mb)	$\top$ (27.2 s / 53.2 Mb)
30	3	1/3	10,7,8	$\bot$ (22.9 s / 305.2 Mb)	$\top$ (106.7 s / 109.3 Mb)	$\bot$ (110.9 s / 418 Mb)

Table 7: Results of the coffee machine example over mono-infinite for $k \leq 30$.

k	ν	δ	T_1, T_2, T_3	COMPLETE (TIME / MEM)	TH_1 OUTCOME (TIME / MEM)	TH_2 OUTCOME (TIME / MEM)
35	1	1	4,4,4	NO (7.4 s / 103.6 Mb)	YES (14.2 s / 39.1 Mb)	YES (10.1 s / 32.7 Mb)
35	1	1	7,6,5	NO (8.6 s / 126.1 Mb)	YES (21.2 s / 47.6 Mb)	YES (13.5 s / 37.5 Mb)
35	1	1	10,7,8	NO (10.5 s / 160.5 Mb)	YES (33.9 s / 60 Mb)	YES (19.9 s / 46 Mb)
35	1	1/3	4,4,4	NO (14.4 s / 194.8 Mb)	YES (48.4 s / 73.4 Mb)	NO (61 s / 399.7 Mb)
35	1	1/3	7,6,5	NO (21.9 s / 272.3 Mb)	YES (84.3 s / 97.7 Mb)	NO (92.8 s / 529.4 Mb)
35	1	1/3	10,7,8	NO (34.6 s / 497.6 Mb)	YES (154.8 s / 133.5 Mb)	NO (167.2 s / 784.4 Mb)
35	2	1	4,4,4	∞	$\top$ (13.7 s / 38.4 Mb)	$\top$ (9.5 s / 31.6 Mb)
35	2	1	7,6,5	∞	$\top$ (20.4 s / 47 Mb)	$\sim$ (26.6 s / 137.3 Mb)
35	2	1	10,7,8	∞	$\top$ (32.8 s / 59.3 Mb)	$\sim$ (39.8 s / 165.4 Mb)
35	2	1/3	4,4,4	∞	$\top$ (45.5 s / 70.5 Mb)	$\sim$ (50 s / 200.5 Mb)
35	2	1/3	7,6,5	∞	$\top$ (78.9 s / 94.8 Mb)	$\bot$ (85.4 s / 402.7 Mb)
35	2	1/3	10,7,8	∞	$\top$ (149.1 s / 130.6 Mb)	$\bot$ (157.7 s / 711.9 Mb)
35	3	1	4,4,4	∞	$\top$ (13.1 s / 37.2 Mb)	$\top$ (9 s / 30.4 Mb)
35	3	1	7,6,5	∞	$\top$ (19 s / 45.5 Mb)	$\top$ (11.7 s / 35 Mb)
35	3	1	10,7,8	∞	$\top$ (32.3 s / 58.2 Mb)	$\sim$ (37.4 s / 161.5 Mb)
35	3	1/3	4,4,4	∞	$\top$ (41.3 s / 66.8 Mb)	$\top$ (23.2 s / 48.2 Mb)
35	3	1/3	7,6,5	∞	$\top$ (73.1 s / 90.5 Mb)	$\top$ (36.3 s / 61.8 Mb)
35	3	1/3	10,7,8	∞	$\top$ (142 s / 127.3 Mb)	$\bot$ (147.6 s / 603 Mb)
40	1	1	4,4,4	∞	$\top$ (18.4 s / 45 Mb)	$\top$ (12.9 s / 37.3 Mb)
40	1	1	7,6,5	∞	$\top$ (27.2 s / 54.7 Mb)	$\top$ (16.8 s / 42.8 Mb)
40	1	1	10,7,8	∞	$\top$ (42.5 s / 68.8 Mb)	$\top$ (25.3 s / 52.5 Mb)
40	1	1/3	4,4,4	∞	$\top$ (62.2 s / 84 Mb)	$\bot$ (77.5 s / 496.7 Mb)
40	1	1/3	7,6,5	∞	$\top$ (107.7 s / 111.7 Mb)	$\bot$ (118.5 s / 564.6 Mb)
40	1	1/3	10,7,8	∞	$\top$ (198.4 s / 152.4 Mb)	$\bot$ (211.2 s / 896.1 Mb)
40	2	1	4,4,4	∞	$\top$ (17.8 s / 43.8 Mb)	$\top$ (12.3 s / 36.1 Mb)
40	2	1	7,6,5	∞	$\top$ (26.1 s / 53.5 Mb)	$\sim$ (33.2 s / 163.6 Mb)
40	2	1	10,7,8	∞	$\top$ (40.9 s / 67.6 Mb)	$\sim$ (52.5 s / 215.8 Mb)
40	2	1/3	4,4,4	∞	$\top$ (57.9 s / 80.3 Mb)	$\sim$ (62.8 s / 239.1 Mb)
40	2	1/3	7,6,5	∞	$\top$ (102.3 s / 108 Mb)	$\bot$ (111.5 s / 585.6 Mb)
40	2	1/3	10,7,8	∞	$\top$ (190.4 s / 148.7 Mb)	$\bot$ (201.7 s / 911.6 Mb)
40	3	1	4,4,4	∞	$\top$ (16.5 s / 42.3 Mb)	$\top$ (11.6 s / 34.6 Mb)
40	3	1	7,6,5	∞	$\top$ (24 s / 51.8 Mb)	$\top$ (14.7 s / 39.9 Mb)
40	3	1	10,7,8	∞	$\top$ (40.6 s / 66.3 Mb)	$\sim$ (47.8 s / 200 Mb)
40	3	1/3	4,4,4	∞	$\top$ (52.8 s / 76.1 Mb)	$\top$ (29 s / 55 Mb)
40	3	1/3	7,6,5	∞	$\top$ (93.5 s / 103 Mb)	$\top$ (45.9 s / 70.4 Mb)
40	3	1/3	10,7,8	∞	$\top$ (182.5 s / 145 Mb)	$\bot$ (188.4 s / 805.7 Mb)
45	1	1	4,4,4	∞	$\top$ (22.9 s / 50.5 Mb)	$\top$ (15.7 s / 41.9 Mb)
45	1	1	7,6,5	∞	$\top$ (34 s / 61.4 Mb)	$\top$ (21.3 s / 48.1 Mb)
45	1	1	10,7,8	∞	$\top$ (53 s / 77.2 Mb)	$\top$ (31.5 s / 59 Mb)
45	1	1/3	4,4,4	∞	$\top$ (77.1 s / 94.2 Mb)	$\bot$ (95.8 s / 540.4 Mb)
45	1	1/3	7,6,5	∞	$\top$ (133.9 s / 125.3 Mb)	$\bot$ (148.7 s / 671.2 Mb)
45	1	1/3	10,7,8	∞	$\top$ (246.3 s / 171 Mb)	$\bot$ (266.7 s / 1044.8 Mb)
45	2	1	4,4,4	∞	$\top$ (21.8 s / 49.1 Mb)	$\top$ (15.3 s / 40.5 Mb)
45	2	1	7,6,5	∞	$\top$ (32.2 s / 60 Mb)	$\sim$ (42 s / 195.6 Mb)
45	2	1	10,7,8	∞	$\top$ (52.4 s / 75.8 Mb)	$\sim$ (63.9 s / 292.3 Mb)
45	2	1/3	4,4,4	∞	$\top$ (71.7 s / 90.1 Mb)	$\sim$ (77.8 s / 285.2 Mb)
45	2	1/3	7,6,5	∞	$\top$ (127.2 s / 121.1 Mb)	$\bot$ (139.3 s / 619.4 Mb)
45	2	1/3	10,7,8	∞	$\top$ (237 s / 166.8 Mb)	$\bot$ (252.7 s / 1054.7 Mb)
45	3	1	4,4,4	∞	$\top$ (20.5 s / 47.5 Mb)	$\top$ (14.5 s / 38.9 Mb)
45	3	1	7,6,5	∞	$\top$ (30.5 s / 58.2 Mb)	$\top$ (18.9 s / 44.8 Mb)
45	3	1	10,7,8	∞	$\top$ (50.4 s / 74.4 Mb)	$\sim$ (60.1 s / 233.8 Mb)
45	3	1/3	4,4,4	∞	$\top$ (65.5 s / 85.4 Mb)	$\top$ (36 s / 61.8 Mb)
45	3	1/3	7,6,5	∞	$\top$ (116.5 s / 115.5 Mb)	$\top$ (57 s / 79.1 Mb)
45	3	1/3	10,7,8	∞	$\top$ (227.8 s / 162.6 Mb)	$\bot$ (236.5 s / 1199 Mb)
50	1	1	4,4,4	∞	$\top$ (27.7 s / 56 Mb)	$\top$ (19.4 s / 46.5 Mb)
50	1	1	7,6,5	∞	$\top$ (40.9 s / 68.1 Mb)	$\top$ (25.4 s / 53.3 Mb)
50	1	1	10,7,8	∞	$\top$ (63.4 s / 85.6 Mb)	$\top$ (38.2 s / 65.4 Mb)
50	1	1/3	4,4,4	∞	$\top$ (93.8 s / 104.5 Mb)	$\bot$ (116.5 s / 704.6 Mb)
50	1	1/3	7,6,5	∞	$\top$ (162.9 s / 138.9 Mb)	$\bot$ (184.6 s / 1016.1 Mb)
50	1	1/3	10,7,8	∞	$\top$ (300.7 s / 189.5 Mb)	$\bot$ (331.2 s / 1791.3 Mb)
50	2	1	4,4,4	∞	$\top$ (26.6 s / 54.5 Mb)	$\top$ (18.4 s / 45 Mb)
50	2	1	7,6,5	∞	$\top$ (39.9 s / 66.6 Mb)	$\sim$ (50.7 s / 230.3 Mb)
50	2	1	10,7,8	∞	$\top$ (61.8 s / 84.1 Mb)	$\sim$ (76.9 s / 273.9 Mb)
50	2	1/3	4,4,4	∞	$\top$ (87.5 s / 99.9 Mb)	$\sim$ (94.3 s / 322.9 Mb)
50	2	1/3	7,6,5	∞	$\top$ (154.9 s / 134.3 Mb)	$\bot$ (168.2 s / 741.3 Mb)
50	2	1/3	10,7,8	∞	$\top$ (290.6 s / 184.9 Mb)	$\bot$ (307.9 s / 1397.1 Mb)
50	3	1	4,4,4	∞	$\top$ (24.6 s / 52.7 Mb)	$\top$ (17 s / 43.2 Mb)
50	3	1	7,6,5	∞	$\top$ (36.3 s / 64.5 Mb)	$\top$ (22.3 s / 49.7 Mb)
50	3	1	10,7,8	∞	$\top$ (60.8 s / 82.6 Mb)	$\sim$ (71.8 s / 272.1 Mb)
50	3	1/3	4,4,4	∞	$\top$ (79.8 s / 94.7 Mb)	$\top$ (43.7 s / 68.5 Mb)
50	3	1/3	7,6,5	∞	$\top$ (141.5 s / 128.1 Mb)	$\top$ (69.9 s / 87.8 Mb)
50	3	1/3	10,7,8	∞	$\top$ (279.2 s / 180.2 Mb)	$\bot$ (289.5 s / 1200.1 Mb)

Table 8: Results of the coffee machine example over mono-infinite for $k > 30$.

k	ν	δ	T_1, T_2, T_3	COMPLETE (TIME / MEM)	TH_1 OUTCOME (TIME / MEM)	TH_2 OUTCOME (TIME / MEM)
5	1	1	4,4,4	$\perp$ (0.6 s / 6.6 Mb)	$\top$ (1.5 s / 9.4 Mb)	$\top$ (1.1 s / 7.8 Mb)
5	1	1	7,6,5	$\perp$ (0.7 s / 7.6 Mb)	$\top$ (1.9 s / 11.5 Mb)	$\top$ (1.3 s / 9 Mb)
5	1	1	10,7,8	$\perp$ (0.8 s / 9 Mb)	$\top$ (2.3 s / 14.6 Mb)	$\top$ (1.6 s / 11.2 Mb)
5	1	1/3	4,4,4	$\perp$ (1 s / 10.4 Mb)	$\top$ (3.4 s / 17.7 Mb)	$\sim$ (4.5 s / 34 Mb)
5	1	1/3	7,6,5	$\perp$ (1.3 s / 13.4 Mb)	$\top$ (6.3 s / 23.5 Mb)	$\sim$ (6.3 s / 44.1 Mb)
5	1	1/3	10,7,8	$\perp$ (1.8 s / 18.7 Mb)	$\top$ (11.3 s / 32.4 Mb)	$\sim$ (12.1 s / 54.9 Mb)
5	2	1	4,4,4	$\perp$ (0.6 s / 6.7 Mb)	$\top$ (1.3 s / 9.3 Mb)	$\top$ (1.4 s / 7.7 Mb)
5	2	1	7,6,5	$\perp$ (0.7 s / 7.5 Mb)	$\top$ (1.9 s / 11.3 Mb)	$\sim$ (2.5 s / 21.4 Mb)
5	2	1	10,7,8	$\perp$ (0.8 s / 8.8 Mb)	$\top$ (2.3 s / 14.4 Mb)	$\sim$ (3.4 s / 26.5 Mb)
5	2	1/3	4,4,4	$\perp$ (1.2 s / 10 Mb)	$\top$ (3.7 s / 17.2 Mb)	$\top$ (1.9 s / 13 Mb)
5	2	1/3	7,6,5	$\perp$ (1.2 s / 13.9 Mb)	$\top$ (5.7 s / 23 Mb)	$\sim$ (6.2 s / 38.8 Mb)
5	2	1/3	10,7,8	$\perp$ (2 s / 16.5 Mb)	$\top$ (11 s / 31.9 Mb)	$\sim$ (11.5 s / 60.1 Mb)
5	3	1	4,4,4	$\top$ (0.6 s / 4.4 Mb)	$\top$ (1.2 s / 9 Mb)	$\top$ (1 s / 7.5 Mb)
5	3	1	7,6,5	$\top$ (0.6 s / 4.9 Mb)	$\top$ (1.8 s / 11 Mb)	$\top$ (1.2 s / 8.5 Mb)
5	3	1	10,7,8	$\perp$ (0.8 s / 8.5 Mb)	$\top$ (2.5 s / 14.3 Mb)	$\sim$ (3.2 s / 25.9 Mb)
5	3	1/3	4,4,4	$\top$ (0.9 s / 6.8 Mb)	$\top$ (3 s / 16.6 Mb)	$\top$ (1.9 s / 12.4 Mb)
5	3	1/3	7,6,5	$\perp$ (1.2 s / 13.2 Mb)	$\top$ (6 s / 22.1 Mb)	$\top$ (2.7 s / 15.5 Mb)
5	3	1/3	10,7,8	$\perp$ (1.7 s / 15.2 Mb)	$\top$ (10.8 s / 31.4 Mb)	$\sim$ (11 s / 52.7 Mb)
10	1	1	4,4,4	$\perp$ (1.3 s / 16.2 Mb)	$\top$ (3.5 s / 17.9 Mb)	$\top$ (2.6 s / 14.9 Mb)
10	1	1	7,6,5	$\perp$ (1.4 s / 18.7 Mb)	$\top$ (4.8 s / 21.8 Mb)	$\top$ (3 s / 17.1 Mb)
10	1	1	10,7,8	$\perp$ (2.1 s / 22.5 Mb)	$\top$ (7.7 s / 27.7 Mb)	$\top$ (4.6 s / 21.2 Mb)
10	1	1/3	4,4,4	$\perp$ (2.3 s / 28.6 Mb)	$\top$ (11.5 s / 33.5 Mb)	$\sim$ (13.9 s / 72.6 Mb)
10	1	1/3	7,6,5	$\perp$ (2.7 s / 33.6 Mb)	$\top$ (19.5 s / 44.2 Mb)	$\sim$ (21.6 s / 95.7 Mb)
10	1	1/3	10,7,8	$\perp$ (4.4 s / 47.4 Mb)	$\top$ (36.1 s / 60.7 Mb)	$\sim$ (39.5 s / 109.1 Mb)
10	2	1	4,4,4	$\perp$ (1.3 s / 15.7 Mb)	$\top$ (3.4 s / 17.6 Mb)	$\top$ (2.2 s / 14.6 Mb)
10	2	1	7,6,5	$\perp$ (1.5 s / 17.8 Mb)	$\top$ (4.7 s / 21.5 Mb)	$\sim$ (5.8 s / 43.3 Mb)
10	2	1	10,7,8	$\perp$ (1.8 s / 21.6 Mb)	$\top$ (7.9 s / 27.3 Mb)	$\sim$ (9.1 s / 53.7 Mb)
10	2	1/3	4,4,4	$\perp$ (2.2 s / 24.4 Mb)	$\top$ (10.8 s / 32.5 Mb)	$\top$ (6.2 s / 24.7 Mb)
10	2	1/3	7,6,5	$\perp$ (2.8 s / 30.3 Mb)	$\top$ (18.9 s / 43.3 Mb)	$\sim$ (20.2 s / 86.9 Mb)
10	2	1/3	10,7,8	$\perp$ (4.5 s / 43.4 Mb)	$\top$ (35.5 s / 59.8 Mb)	$\sim$ (37.6 s / 107.2 Mb)
10	3	1	4,4,4	$\top$ (1.1 s / 8.8 Mb)	$\top$ (3.1 s / 17.2 Mb)	$\top$ (2.1 s / 14.2 Mb)
10	3	1	7,6,5	$\top$ (1.3 s / 9.7 Mb)	$\top$ (4.3 s / 20.9 Mb)	$\top$ (2.7 s / 16.2 Mb)
10	3	1	10,7,8	$\perp$ (1.7 s / 21.8 Mb)	$\top$ (7.3 s / 27 Mb)	$\sim$ (8.6 s / 52.5 Mb)
10	3	1/3	4,4,4	$\top$ (1.9 s / 13.3 Mb)	$\top$ (10 s / 31.4 Mb)	$\top$ (5.6 s / 23.5 Mb)
10	3	1/3	7,6,5	$\top$ (2.5 s / 16 Mb)	$\top$ (17.4 s / 41.6 Mb)	$\top$ (8.7 s / 29.4 Mb)
10	3	1/3	10,7,8	$\perp$ (4.1 s / 40.8 Mb)	$\top$ (33.8 s / 58.8 Mb)	$\sim$ (36.2 s / 105.2 Mb)
20	1	1	4,4,4	$\perp$ (3.5 s / 47 Mb)	$\top$ (11.6 s / 34.7 Mb)	$\top$ (8 s / 28.9 Mb)
20	1	1	7,6,5	$\perp$ (3.7 s / 55.1 Mb)	$\top$ (17.1 s / 42.3 Mb)	$\top$ (10.6 s / 33.1 Mb)
20	1	1	10,7,8	$\perp$ (4.4 s / 65.4 Mb)	$\top$ (27.2 s / 53.7 Mb)	$\top$ (16.4 s / 41.1 Mb)
20	1	1/3	4,4,4	$\perp$ (6.6 s / 78.4 Mb)	$\top$ (39 s / 64.9 Mb)	$\sim$ (47.6 s / 141 Mb)
20	1	1/3	7,6,5	$\perp$ (9.6 s / 116.7 Mb)	$\top$ (66.8 s / 85.6 Mb)	$\sim$ (72.4 s / 171.4 Mb)
20	1	1/3	10,7,8	$\perp$ (15.2 s / 152 Mb)	$\top$ (124.3 s / 117.3 Mb)	$\sim$ (134.3 s / 235.2 Mb)
20	2	1	4,4,4	$\perp$ (3.2 s / 43.9 Mb)	$\top$ (10.9 s / 34.1 Mb)	$\top$ (7.6 s / 28.3 Mb)
20	2	1	7,6,5	$\perp$ (3.4 s / 50.6 Mb)	$\top$ (16.5 s / 41.7 Mb)	$\sim$ (20.6 s / 99.2 Mb)
20	2	1	10,7,8	$\perp$ (4.3 s / 61.6 Mb)	$\top$ (26.5 s / 53.1 Mb)	$\sim$ (32 s / 115.9 Mb)
20	2	1/3	4,4,4	$\perp$ (5.9 s / 65.9 Mb)	$\top$ (36.2 s / 63.1 Mb)	$\top$ (21.8 s / 48 Mb)
20	2	1/3	7,6,5	$\perp$ (8.9 s / 98.9 Mb)	$\top$ (63.4 s / 83.8 Mb)	$\sim$ (68.4 s / 176.3 Mb)
20	2	1/3	10,7,8	$\perp$ (14.2 s / 139.9 Mb)	$\top$ (119.2 s / 115.5 Mb)	$\sim$ (129.4 s / 231.3 Mb)
20	3	1	4,4,4	$\top$ (3 s / 19.7 Mb)	$\top$ (10.2 s / 33.4 Mb)	$\top$ (7.2 s / 27.6 Mb)
20	3	1	7,6,5	$\top$ (3.1 s / 21.5 Mb)	$\top$ (15.2 s / 40.7 Mb)	$\top$ (9.2 s / 31.5 Mb)
20	3	1	10,7,8	$\perp$ (4.3 s / 59.1 Mb)	$\top$ (25.6 s / 52.5 Mb)	$\sim$ (30.4 s / 114.3 Mb)
20	3	1/3	4,4,4	$\top$ (5.1 s / 28.3 Mb)	$\top$ (34.2 s / 60.9 Mb)	$\top$ (19.5 s / 45.7 Mb)
20	3	1/3	7,6,5	$\top$ (7.3 s / 33.6 Mb)	$\top$ (58.4 s / 80.6 Mb)	$\top$ (29.9 s / 57.1 Mb)
20	3	1/3	10,7,8	$\perp$ (14.2 s / 146.6 Mb)	$\top$ (117.9 s / 113.7 Mb)	$\sim$ (122.8 s / 227.7 Mb)
25	1	1	4,4,4	$\perp$ (4.4 s / 67.5 Mb)	$\top$ (17 s / 43.2 Mb)	$\top$ (12 s / 36 Mb)
25	1	1	7,6,5	$\perp$ (5.4 s / 79.9 Mb)	$\top$ (25.3 s / 52.6 Mb)	$\top$ (15.9 s / 41.2 Mb)
25	1	1	10,7,8	$\perp$ (7 s / 95 Mb)	$\top$ (39.7 s / 66.7 Mb)	$\top$ (23.9 s / 51 Mb)
25	1	1/3	4,4,4	$\perp$ (10 s / 114 Mb)	$\top$ (57.7 s / 80.6 Mb)	$\sim$ (71.1 s / 197.6 Mb)
25	1	1/3	7,6,5	$\perp$ (13.9 s / 170.8 Mb)	$\top$ (100.4 s / 106.3 Mb)	$\sim$ (108.7 s / 225.7 Mb)
25	1	1/3	10,7,8	$\perp$ (22.4 s / 238.6 Mb)	$\top$ (184.8 s / 145.6 Mb)	$\sim$ (200.2 s / 379.2 Mb)
25	2	1	4,4,4	$\perp$ (4.5 s / 63.8 Mb)	$\top$ (16.3 s / 42.4 Mb)	$\top$ (11.3 s / 35.2 Mb)
25	2	1	7,6,5	$\perp$ (4.9 s / 75.6 Mb)	$\top$ (24.8 s / 51.9 Mb)	$\sim$ (31 s / 126.5 Mb)
25	2	1	10,7,8	$\perp$ (6.7 s / 96.6 Mb)	$\top$ (39.6 s / 66 Mb)	$\sim$ (47.6 s / 158.4 Mb)
25	2	1/3	4,4,4	$\perp$ (10 s / 111.2 Mb)	$\top$ (54.5 s / 78.4 Mb)	$\top$ (32 s / 59.6 Mb)
25	2	1/3	7,6,5	$\perp$ (12.9 s / 149.7 Mb)	$\top$ (93.9 s / 104.1 Mb)	$\sim$ (102.6 s / 251.1 Mb)
25	2	1/3	10,7,8	$\perp$ (22.5 s / 227 Mb)	$\top$ (178.3 s / 143.4 Mb)	$\sim$ (187.5 s / 360.3 Mb)
25	3	1	4,4,4	$\top$ (4 s / 26.3 Mb)	$\top$ (16.3 s / 41.5 Mb)	$\top$ (11.2 s / 34.3 Mb)
25	3	1	7,6,5	$\top$ (4.5 s / 28.5 Mb)	$\top$ (23.1 s / 50.5 Mb)	$\top$ (14.1 s / 39.1 Mb)
25	3	1	10,7,8	$\perp$ (6.8 s / 83.9 Mb)	$\top$ (38.4 s / 65.2 Mb)	$\sim$ (46.9 s / 155.5 Mb)
25	3	1/3	4,4,4	$\top$ (7.9 s / 36.9 Mb)	$\top$ (49.7 s / 75.6 Mb)	$\top$ (29.6 s / 56.8 Mb)
25	3	1/3	7,6,5	$\top$ (12.3 s / 43.4 Mb)	$\top$ (87.3 s / 100.1 Mb)	$\top$ (45.2 s / 70.9 Mb)
25	3	1/3	10,7,8	$\perp$ (21.4 s / 196.2 Mb)	$\top$ (173 s / 141.2 Mb)	$\sim$ (185.7 s / 356.5 Mb)

Table 9: Results of the coffee machine example over bi-infinite for $k \leq 25$.

k	ν	δ	T_1, T_2, T_3	COMPLETE (TIME / MEM)	TH_1 OUTCOME (TIME / MEM)	TH_2 OUTCOME (TIME / MEM)
30	1	1	4,4,4	$\perp$ (6.8 s / 93.2 Mb)	$\top$ (24 s / 51.6 Mb)	$\top$ (16.7 s / 43 Mb)
30	1	1	7,6,5	∞	$\top$ (34.9 s / 63.2 Mb)	$\top$ (21.6 s / 49.3 Mb)
30	1	1	10,7,8	∞	$\top$ (55.3 s / 80 Mb)	$\top$ (33 s / 61 Mb)
30	1	1/3	4,4,4	∞	$\top$ (82.3 s / 96.6 Mb)	$\sim$ (101.5 s / 248.5 Mb)
30	1	1/3	7,6,5	∞	$\top$ (142.7 s / 127.3 Mb)	$\sim$ (156.1 s / 343 Mb)
30	1	1/3	10,7,8	∞	$\top$ (274.2 s / 174.2 Mb)	$\sim$ (287.8 s / 476.1 Mb)
30	2	1	4,4,4	∞	$\top$ (23.7 s / 51.1 Mb)	$\top$ (16.1 s / 42.2 Mb)
30	2	1	7,6,5	∞	$\top$ (34.5 s / 62.3 Mb)	$\sim$ (43.9 s / 159.6 Mb)
30	2	1	10,7,8	∞	$\top$ (56.2 s / 79.2 Mb)	$\sim$ (68.1 s / 193.7 Mb)
30	2	1/3	4,4,4	∞	$\top$ (78.2 s / 94 Mb)	$\top$ (45.5 s / 71.2 Mb)
30	2	1/3	7,6,5	∞	$\top$ (135.3 s / 124.7 Mb)	$\sim$ (149.3 s / 315.1 Mb)
30	2	1/3	10,7,8	∞	$\top$ (261.6 s / 171.6 Mb)	$\sim$ (274.4 s / 443.5 Mb)
30	3	1	4,4,4	∞	$\top$ (22.3 s / 50 Mb)	$\top$ (15.5 s / 41 Mb)
30	3	1	7,6,5	∞	$\top$ (33.3 s / 60.8 Mb)	$\top$ (19.6 s / 46.8 Mb)
30	3	1	10,7,8	∞	$\top$ (55.5 s / 78.3 Mb)	$\sim$ (65.3 s / 189.2 Mb)
30	3	1/3	4,4,4	∞	$\top$ (72.9 s / 90.7 Mb)	$\top$ (41.3 s / 67.9 Mb)
30	3	1/3	7,6,5	∞	$\top$ (124.5 s / 119.9 Mb)	$\top$ (63.5 s / 84.7 Mb)
30	3	1/3	10,7,8	∞	$\top$ (250.4 s / 168.9 Mb)	$\sim$ (268.1 s / 377 Mb)
35	1	1	4,4,4	∞	$\top$ (31.2 s / 60.4 Mb)	$\top$ (21.6 s / 50.1 Mb)
35	1	1	7,6,5	∞	$\top$ (46.1 s / 73.5 Mb)	$\top$ (28.2 s / 57.3 Mb)
35	1	1	10,7,8	∞	$\top$ (73.3 s / 93 Mb)	$\top$ (43.5 s / 71 Mb)
35	1	1/3	4,4,4	∞	$\top$ (105.7 s / 112.3 Mb)	$\sim$ (129.7 s / 305 Mb)
35	1	1/3	7,6,5	∞	$\top$ (182.9 s / 148 Mb)	$\sim$ (199.4 s / 378.8 Mb)
35	1	1/3	10,7,8	∞	$\top$ (344.2 s / 202.5 Mb)	$\sim$ (369.4 s / 608.9 Mb)
35	2	1	4,4,4	∞	$\top$ (29.9 s / 59.4 Mb)	$\top$ (21.1 s / 49.1 Mb)
35	2	1	7,6,5	∞	$\top$ (45.2 s / 72.5 Mb)	$\sim$ (56.4 s / 199.5 Mb)
35	2	1	10,7,8	∞	$\top$ (71.9 s / 92 Mb)	$\sim$ (87.6 s / 234.7 Mb)
35	2	1/3	4,4,4	∞	$\top$ (100.2 s / 109.3 Mb)	$\top$ (58.2 s / 82.8 Mb)
35	2	1/3	7,6,5	∞	$\top$ (175.8 s / 144.9 Mb)	$\sim$ (189.2 s / 360.7 Mb)
35	2	1/3	10,7,8	∞	$\top$ (338.1 s / 199.4 Mb)	$\sim$ (351.4 s / 543.2 Mb)
35	3	1	4,4,4	∞	$\top$ (28.6 s / 58.1 Mb)	$\top$ (19.6 s / 47.7 Mb)
35	3	1	7,6,5	∞	$\top$ (42 s / 70.7 Mb)	$\top$ (25.8 s / 54.5 Mb)
35	3	1	10,7,8	∞	$\top$ (69.7 s / 91 Mb)	$\sim$ (83.7 s / 235.3 Mb)
35	3	1/3	4,4,4	∞	$\top$ (93.4 s / 105.4 Mb)	$\top$ (53.2 s / 79 Mb)
35	3	1/3	7,6,5	∞	$\top$ (161.1 s / 139.4 Mb)	$\top$ (82.3 s / 98.5 Mb)
35	3	1/3	10,7,8	∞	$\top$ (320.4 s / 196.3 Mb)	$\sim$ (339.6 s / 517.7 Mb)
40	1	1	4,4,4	∞	$\top$ (39.6 s / 68.9 Mb)	$\top$ (27.6 s / 57.1 Mb)
40	1	1	7,6,5	∞	$\top$ (59.6 s / 83.8 Mb)	$\top$ (36.4 s / 65.4 Mb)
40	1	1	10,7,8	∞	$\top$ (94.3 s / 106.1 Mb)	$\top$ (55.5 s / 81 Mb)
40	1	1/3	4,4,4	∞	$\top$ (135.6 s / 128 Mb)	$\sim$ (167.2 s / 341.8 Mb)
40	1	1/3	7,6,5	∞	$\top$ (244.7 s / 168.6 Mb)	$\sim$ (255.4 s / 442.7 Mb)
40	1	1/3	10,7,8	∞	$\top$ (446.2 s / 230.7 Mb)	$\sim$ (472.8 s / 755.3 Mb)
40	2	1	4,4,4	∞	$\top$ (38.3 s / 67.7 Mb)	$\top$ (26.4 s / 56 Mb)
40	2	1	7,6,5	∞	$\top$ (57.4 s / 82.7 Mb)	$\sim$ (72 s / 232.6 Mb)
40	2	1	10,7,8	∞	$\top$ (93.1 s / 104.9 Mb)	$\sim$ (112.1 s / 294.8 Mb)
40	2	1/3	4,4,4	∞	$\top$ (127.5 s / 124.5 Mb)	$\top$ (74.6 s / 94.5 Mb)
40	2	1/3	7,6,5	∞	$\top$ (224.6 s / 165.2 Mb)	$\sim$ (240.6 s / 400 Mb)
40	2	1/3	10,7,8	∞	$\top$ (431.4 s / 227.2 Mb)	$\sim$ (452.9 s / 761.2 Mb)
40	3	1	4,4,4	∞	$\top$ (36.3 s / 66.2 Mb)	$\top$ (25.5 s / 54.5 Mb)
40	3	1	7,6,5	∞	$\top$ (53.3 s / 80.6 Mb)	$\top$ (32.3 s / 62.2 Mb)
40	3	1	10,7,8	∞	$\top$ (89.3 s / 103.8 Mb)	$\sim$ (106.7 s / 284.8 Mb)
40	3	1/3	4,4,4	∞	$\top$ (118.6 s / 120.1 Mb)	$\top$ (67.6 s / 90.1 Mb)
40	3	1/3	7,6,5	∞	$\top$ (207.2 s / 158.8 Mb)	$\top$ (104.9 s / 112.3 Mb)
40	3	1/3	10,7,8	∞	$\top$ (420.4 s / 223.7 Mb)	$\sim$ (439.5 s / 630.8 Mb)
45	1	1	4,4,4	∞	$\top$ (50.1 s / 77.4 Mb)	$\top$ (34.2 s / 64.2 Mb)
45	1	1	7,6,5	∞	$\top$ (73.5 s / 94.1 Mb)	$\top$ (45.2 s / 73.5 Mb)
45	1	1	10,7,8	∞	$\top$ (116.5 s / 119.1 Mb)	$\top$ (69.1 s / 91 Mb)
45	1	1/3	4,4,4	∞	$\top$ (169.4 s / 143.7 Mb)	$\sim$ (209 s / 409 Mb)
45	1	1/3	7,6,5	∞	$\top$ (294.6 s / 189.3 Mb)	$\sim$ (321.7 s / 603.8 Mb)
45	1	1/3	10,7,8	∞	$\top$ (575.1 s / 258.9 Mb)	$\sim$ (592.8 s / 963.7 Mb)
45	2	1	4,4,4	∞	$\top$ (48 s / 76.1 Mb)	$\top$ (32.9 s / 62.9 Mb)
45	2	1	7,6,5	∞	$\top$ (72 s / 92.8 Mb)	$\sim$ (90.1 s / 276 Mb)
45	2	1	10,7,8	∞	$\top$ (114.5 s / 117.8 Mb)	$\sim$ (140 s / 332.7 Mb)
45	2	1/3	4,4,4	∞	$\top$ (161 s / 139.8 Mb)	$\top$ (94.1 s / 106.2 Mb)
45	2	1/3	7,6,5	∞	$\top$ (283.6 s / 185.4 Mb)	$\sim$ (302.4 s / 551 Mb)
45	2	1/3	10,7,8	∞	$\top$ (544.1 s / 255 Mb)	$\sim$ (566.2 s / 652.6 Mb)
45	3	1	4,4,4	∞	$\top$ (45.6 s / 74.4 Mb)	$\top$ (31.2 s / 61.2 Mb)
45	3	1	7,6,5	∞	$\top$ (66.8 s / 90.5 Mb)	$\top$ (40.8 s / 69.8 Mb)
45	3	1	10,7,8	∞	$\top$ (112.2 s / 116.5 Mb)	$\sim$ (133.8 s / 340.1 Mb)
45	3	1/3	4,4,4	∞	$\top$ (149.2 s / 134.9 Mb)	$\top$ (85.3 s / 101.3 Mb)
45	3	1/3	7,6,5	∞	$\top$ (260.2 s / 178.3 Mb)	$\top$ (131.9 s / 126.2 Mb)
45	3	1/3	10,7,8	∞	$\top$ (525.9 s / 251.1 Mb)	$\sim$ (549.1 s / 755.2 Mb)
50	1	1	4,4,4	∞	$\top$ (60.8 s / 85.8 Mb)	$\top$ (42.1 s / 71.2 Mb)
50	1	1	7,6,5	∞	$\top$ (89.8 s / 104.4 Mb)	$\top$ (54.4 s / 81.6 Mb)
50	1	1	10,7,8	∞	$\top$ (143 s / 132.1 Mb)	$\top$ (84 s / 101 Mb)
50	1	1/3	4,4,4	∞	$\top$ (208 s / 159.4 Mb)	$\sim$ (254.9 s / 461.5 Mb)
50	1	1/3	7,6,5	∞	$\top$ (360.4 s / 209.9 Mb)	$\sim$ (390.9 s / 603.7 Mb)
50	1	1/3	10,7,8	∞	$\top$ (684.2 s / 287.2 Mb)	$\sim$ (741.3 s / 907.5 Mb)
50	2	1	4,4,4	∞	$\top$ (57.9 s / 84.4 Mb)	$\top$ (39.9 s / 69.8 Mb)
50	2	1	7,6,5	∞	$\top$ (86.4 s / 103 Mb)	$\sim$ (108.4 s / 334.5 Mb)
50	2	1	10,7,8	∞	$\top$ (140.9 s / 130.7 Mb)	$\sim$ (169.2 s / 386.5 Mb)
50	2	1/3	4,4,4	∞	$\top$ (195.6 s / 155.1 Mb)	$\top$ (114 s / 117.9 Mb)
50	2	1/3	7,6,5	∞	$\top$ (344.4 s / 205.6 Mb)	$\sim$ (368.9 s / 626.2 Mb)
50	2	1/3	10,7,8	∞	$\top$ (665.5 s / 282.8 Mb)	$\sim$ (692.2 s / 796.8 Mb)
50	3	1	4,4,4	∞	$\top$ (54.9 s / 82.5 Mb)	$\top$ (37.6 s / 67.9 Mb)
50	3	1	7,6,5	∞	$\top$ (80.8 s / 100.4 Mb)	$\top$ (49 s / 77.5 Mb)
50	3	1	10,7,8	∞	$\top$ (136.2 s / 129.3 Mb)	$\sim$ (160.3 s / 389.1 Mb)
50	3	1/3	4,4,4	∞	$\top$ (183 s / 149.7 Mb)	$\top$ (103.3 s / 112.4 Mb)
50	3	1/3	7,6,5	∞	$\top$ (314.3 s / 197.7 Mb)	$\top$ (160.4 s / 140.1 Mb)
50	3	1/3	10,7,8	∞	$\top$ (642.5 s / 278.5 Mb)	$\sim$ (663.9 s / 815.3 Mb)

Table 10: Results of the coffee machine example over bi-infinite for $k > 25$.

original formulas represent.

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

The technique is simple to implement in practice, and we performed some experiments with it, through a bounded validity checker for discrete-time formulas. The results are promising in that they show that the effects of incompleteness can 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, advances and generalizations in the notion of sampling invariance will yield extensions of the discretization techniques, both in terms of classes of MTL formulas that can be handled (for instance a relaxation of the nesting-freeness requirement) and in the strength and form of the χ constraint that restricts the behaviors we consider. Second, heuristics and methods will be developed to guide the writing of dense-time specifications in a form that is amenable to the application of the discretization techniques. Third, the tool support can be extended and enhanced, through the use of additional engines, as well as in supporting the aforementioned methods.

Acknowledgments

We thank Paritosh Pandya for discussions suggesting to apply the notion of sampling invariance to dense-time verification through discretization, and Mario Arrigoni Neri for providing a sketch of the coffee machine example.

References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[AL94] Martín Abadi and Leslie Lamport. An old-fashioned recipe for real-time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.

[AMP98] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In Davide Sangiorgi and Robert de Simone, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 470–484. Springer-Verlag, 1998.

[BER94] Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. Verifying invariance properties of timed systems with duration variables. In *Proceedings of the 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 193–210. Springer-Verlag, 1994.

[BHJ+06] Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5:5):1–64, 2006.

[BLN03] Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for BDD-based verification of real-time systems. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 122–125. Springer-Verlag, 2003.

[BMT99] Marius Bozga, Oded Maler, and Stavros Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In Laurence Pierre and Thomas Kropf, editors, *Proceedings of the 10th Correct Hardware Design and Verification Methods Advanced Research Working Conference (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 125–141. Springer-Verlag, 1999.

[Bošn99] Dragan Bošnački. Digitization of timed automata. In *Proceedings of the 4th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'99)*, pages 283–302, 1999.

[CCPC+99] Emanuele Ciapessoni, Alberto Coen-Porisini, Ernani Crivelli, Dino Mandrioli, Piergiorgio Mirandola, and Angelo Morzenti. From formal models to formally-based methods: an industrial experience. *ACM Transactions on Software Engineering and Methodology*, 8(1):79–113, 1999.

[CHR91] Zhou Chaochen, Charles Anthony Richard Hoare, and Anders P. Ravn. A calculus of duration. *Information Processing Letters*, 40(5):269–276, 1991.

[CLT07] Edmund M. Clarke, Flavio Lerda, and Muralidhar Talupur. An abstraction technique for real-time verification. In *Proceedings of the GM R&D Workshop on Next Generation Design and Verification Methodologies for Distributed Embedded Control System*, 2007.

[CP03] Gaurav Chakravorty and Paritosh K. Pandya. Digiziting interval duration logic. In Warren A. Hunt, Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 167–179. Springer-Verlag, 2003.

[dM95] Luca de Alfaro and Zohar Manna. Verification in continuous time by discrete reasoning. In Vangalur S. Alagar and Maurice Nivat, editors, *Proceedings of the 4th International Conference on Algebraic Methodology and Software Technology (AMAST'95)*, volume 936 of *Lecture Notes in Computer Science*, pages 292–306. Springer-Verlag, 1995.

[FMMR07] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. Modeling time in computing: a taxonomy and a comparative survey. Technical Report 2007.22, Dipartimento di Elettronica e Informazione, Politecnico di Milano, January 2007.

[FR05] Carlo A. Furia and Matteo Rossi. When discrete met continuous: on the integration of discrete- and continuous-time metric temporal logics. Technical Report 2005.44, Dipartimento di Elettronica e Informazione, Politecnico di Milano, October 2005.

[FR06] Carlo A. Furia and Matteo Rossi. Integrating discrete- and continuous-time metric temporal logics through sampling. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 215–229. Springer-Verlag, September 2006.

[Frä96] Martin Fränzle. Decidability of duration calculi on restricted model classes. Technical Report Kiel MF 21/1, Institut für Informatik und Praktische Mathematik, Christian-Albrechts Universität Kiel, July 1996.

[Frä02] Martin Fränzle. Take it NP-easy: Bounded model construction for Duration Calculus. In Werner Damm and Ernst-Rüdiger Olderog, editors, *Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'02)*, volume 2469 of *Lecture Notes in Computer Science*, pages 245–264. Springer-Verlag, 2002.

[Fur07] Carlo Alberto Furia. *Scaling up the formal analysis of real-time systems*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, May 2007.

[GD00] Marc Geilen and Dennis Dams. An on-the-fly tableau construction for a real-time temporal logic. In *Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)*, volume 1926 of *Lecture Notes in Computer Science*, pages 276–290. Springer-Verlag, 2000.

[GM01] Angelo Gargantini and Angelo Morzenti. Automated deductive requirement analysis of critical systems. *ACM Transactions on Software Engineering and Methodology*, 10(3):255–307, 2001.

[GPV94] Aleks Göllü, Anuj Puri, and Pravin Varaiya. Discretization of timed automata. In *Proceedings of the 33rd Conference on Decision and Control*, pages 957–958, 1994.

[Hen98] Thomas A. Henzinger. It's about time: Real-time logics reviewed. In Davide Sangiorgi and Robert de Simone, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer-Verlag, 1998.

[HG96] Dang Van Hung and Phan Hong Giang. Sampling semantics of Duration Calculus. In Joachim Parrow Bengt Jonsson, editor, *Proceedings of the 4th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)*, volume 1135 of *Lecture Notes in Computer Science*, pages 188–207. Springer-Verlag, 1996.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata. *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In Werner Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558. Springer-Verlag, 1992.

[KP05] Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In R. Ramanujam and Sandeep Sen, editors, *Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 310–321. Springer-Verlag, 2005.

[MNP06] Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer-Verlag, 2006.

[MP93] Zohar Manna and Amir Pnueli. Models for reactivity. *Acta Informatica*, 30(2):609–678, 1993.

[MP95] Oded Maler and Amir Pnueli. Timing analysis of asynchronous circuits using timed automata. In Paolo Camurati and Hans Eveking, editors, *Proceedings of the Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 987 of *Lecture Notes in Computer Science*, pages 189–205. Springer-Verlag, 1995.

[MPSS03] Angelo Morzenti, Matteo Pradella, Pierluigi San Pietro, and Paola Spoletini. Model-checking TRIO specifications in SPIN. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *Proceedings of International Symposium of Formal Methods Europe (FME'03)*, volume 2805 of *Lecture Notes in Computer Science*, pages 542–561. Springer-Verlag, 2003.

[Oua02] Joël Ouaknine. Digitisation and full abstraction for dense-time model checking. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 2002.

[OW03] Joël Ouaknine and James Worrell. Revisiting digitization, robustness, and decidability for timed automata. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 198–207. IEEE Computer Society Press, 2003.

[Pan02] Paritosh K. Pandya. Interval Duration Logic: expressiveness and decidability. In *Proceedings of the Workshop on Theory and Practice of Timed Systems (TPTS'02)*, volume 62(6) of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[PMS] Matteo Pradella, Angelo Morzenti, and Pierluigi San Pietro. The symmetry of the past and of the future: Bi-infinite time in the verification of temporal properties. Submitted.

[Pra07] Matteo Pradella. ℤot. `http://www.elet.polimi.it/upload/pradella`, March 2007.

[Sch00] Steven Schneider. *Concurrent and Real-Time Systems: The CSP Approach*. John Wiley & Sons, 2000.

[Smo01] Lee Smolin. *Three roads to quantum gravity*. Basic Books, 2001.

[SPC05] Babita Sharma, Paritosh K. Pandya, and Supratik Chakraborty. Bounded validity checking of interval duration logic. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Proceedings of the 11th International Conference*

on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), volume 3440 of *Lecture Notes in Computer Science*, pages 301–316. Springer-Verlag, 2005.

[Spo05] Paola Spoletini. *Verification of Temporal Logic Specification via Model Checking*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, May 2005.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design (FMCAD'00)*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2000.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In Hans Langmaack, Willem P. de Roever, and Jan Vytopil, editors, *Proceedings of the 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *Lecture Notes in Computer Science*, pages 694–715. Springer-Verlag, 1994.