A Metric Encoding for Bounded Model Checking*

Matteo Pradella[1], Angelo Morzenti[2], and Pierluigi San Pietro[2]

[1] CNR IEIIT-MI, Milano, Italy
[2] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
{pradella, morzenti, sanpietr}@elet.polimi.it

Abstract. In Bounded Model Checking both the system model and the checked property are translated into a Boolean formula to be analyzed by a SAT-solver. We introduce a new encoding technique which is particularly optimized for managing quantitative future and past metric temporal operators, typically found in properties of hard real time systems. The encoding is simple and intuitive in principle, but it is made more complex by the presence, typical of the Bounded Model Checking technique, of backward and forward loops used to represent an ultimately periodic infinite domain by a finite structure. We report and comment on the new encoding technique and on an extensive set of experiments carried out to assess its feasibility and effectiveness.

Keywords: Bounded model checking, metric temporal logic.

1 Introduction

In Bounded Model Checking [1] a system under analysis is modeled as a finite-state transition system and a property to be checked is expressed as a formula in temporal logic. The model and the property are both suitably translated into boolean logic formulae, so that the model checking problem is expressed as an instance of a SAT problem, that can be solved efficiently thanks to the significant improvements that occurred in recent years in the technology of the SAT-solver tools [9, 3]. Infinite, ultimately periodic temporal structures that assign a value to every element of the model alphabet are encoded through a finite set of boolean variables, and the cyclic structure of the time domain is encoded into a set of *loop selector variables* that mark the start and end points of the period. As it usually occurs in a model checking framework, a (bounded) model-checker tool can either prove a property or disprove it by exhibiting a counter example, thus providing means to support simulation, test case generation, etc.

In previous work [10], we introduced techniques for managing bi-infinite time in bounded model checking, thus allowing for a more simple and systematic use of past operators in Linear Temporal Logic. In [11, 12], we took advantage of the fact that, in bounded model-checking, both the model and the formula to be checked are ultimately translated into boolean logic. This permits to provide the model not only as a state-transition system, but, alternatively, as a set of temporal logic formulae. We call this a *descriptive* model, as opposed to the term *operational* model used in case it consists of

* Work partially supported by the European Commission, Programme IDEAS-ERC, Project 227977-SMSCom.

a state-transition system. The descriptive model is much more readable and concise if the adopted logic includes past and metric temporal operators, allowing for a great flexibility in the degree of detail and abstraction that the designer can adopt in providing the system model. The model-checking problem is reduced to the problem of satisfiability for a boolean formula that encodes both the modeled system and its conjectured property to be verified, hence the name Bounded *Satisfiability* Checking that we adopted for this approach.

In this paper we take a further step to support efficient Bounded Satisfiability- and Bounded Model-checking by introducing a new encoding technique that is particularly efficient in case of temporal logic formulae that contain time constants having a high numerical value.

In previous approaches [2, 10–12] the operators of temporal logic that express in a precise and quantitative way some timing constraints were encoded by (rather inefficiently) translating them into combinations of non-metric Linear Temporal Logic operators. For instance, the metric temporal logic formula $\Diamond_{=d} P$, which asserts that property P holds at d time units in the future (w.r.t the implicit present time at which the formula is asserted) would be translated into d nested applications of the LTL next-time operator, $\circ^d P$, and then encoded as a series of operator applications, with obvious overhead.

The new encoding for the metric operators translates the time constants in a way that makes the resulting boolean formula much more compact, and hence the verification carried out by the SAT solver-based tools may be significantly faster.

Thus our technique can be usefully applied to all cases where temporal logic formulae that embed important time constants are used. This is both the case of Bounded Satisfiability Checking, where the system model is expressed as a (typically quite large) set of metric temporal logic formulae, and also of more traditional Bounded Model Checking, when the model of the system under analysis is provided by means of a state transition system but one intends to check a hard real-time property with explicit, quantitatively stated timing constraints.

The paper is structured as follows. In Section 2 we provide background and motivations for our work. Section 3 introduces the new metric encoding and analyzes its main features and properties. Section 4 provides an assessment of the new encoding by reporting the experimental results obtained on a set of significant benchmark case studies. Finally, in Section 5 we draw conclusions.

2 Preliminaries

In this section, to make the paper more readable and self-contained, we provide background material on Metric Temporal Logic and bi-infinite time, on Boundel Model- and Satisfiability-Checking, and on the Zot toolkit.

2.1 A metric temporal logic on bi-infinite time

We first recall here Linear Temporal Logic with past operators (PLTL), in the version introduced by Kamp [6], and next extend it with metric temporal operators.

Syntax of PLTL The alphabet of PLTL includes: a finite set Ap of propositional letters; two propositional connectives $\neg, \vee$ (from which other traditional connectives such as $\top, \bot, \neg, \vee, \wedge, \rightarrow, \ldots$ may be defined); four temporal operators (from which other temporal operators can be derived): "until" $\mathcal{U}$, "next-time" $\circ$, "since" $\mathcal{S}$ and "past-time" (or Yesterday) , $\bullet$. Formulae are defined in the usual inductive way: a propositional letter $p \in Ap$ is a formula; $\neg\phi, \phi \vee \psi, \phi\mathcal{U}\psi, \circ\phi, \phi\mathcal{S}\psi, \bullet\phi$, where ϕ, ψ are formulae; nothing else is a formula.

The traditional "eventually" and "globally" operators may be defined as: $\Diamond\phi$ is $\top\mathcal{U}\phi$, $\Box\phi$ is $\neg\Diamond\neg\phi$. Their past counterparts are: $\blacklozenge\phi$ is $\top\mathcal{S}\phi$, $\blacksquare\phi$ is $\neg\blacklozenge\neg\phi$. Another useful operator for PLTL is "Always" $\mathcal{A}lw$, defined as $\mathcal{A}lw\ \phi := \Box\phi \wedge \blacksquare\phi$. The intended meaning of $\mathcal{A}lw\ \phi$ is that ϕ must hold in every instant in the future and in the past. Its dual is "Sometimes" $\mathcal{S}om\ \phi$ defined as $\neg\mathcal{A}lw\neg\phi$.

The dual operators of $\mathcal{U}$ and $\S$ are "Release" $\mathcal{R}$: $\phi\mathcal{R}\psi$ is $\neg(\neg\phi\mathcal{U}\neg\psi)$, and, respectively, "Trigger" $\mathcal{T}$: $\phi\mathcal{T}\psi$ is $\neg(\neg\phi\mathcal{S}\neg\psi)$, which allow the convenient *positive normal form*: Formulae are in positive normal form if their alphabet is $\{\wedge, \vee, \mathcal{U}, \mathcal{R}, \circ, \mathcal{S}, \bullet, \mathcal{T}\} \cup Ap \cup \overline{Ap}$, where $\overline{Ap}$ is the set of formulae of the form $\neg p$ for $p \in Ap$. This form, where negations may only occur on atoms, is very convenient when defining encodings of PLTL into propositional logic. Every PLTL formula ϕ on the alphabet $\{\neg, \vee, \mathcal{U}, \circ, \mathcal{S}, \bullet\} \cup Ap$ may be transformed into an equivalent formula ϕ' in positive normal form.

For the sake of brevity, we also allow n-ary predicate letters (with $n \geq 1$) and the $\forall, \exists$ quantifiers as long as their domains are finite. Hence, one can write, e.g., formulae of the form: $\exists p\ \mathrm{gr}(p)$, with p ranging over $\{1, 2, 3\}$ as a shorthand for $\bigvee_{p\in\{1,2,3\}} \mathrm{gr}_p$.

Semantics of PLTL In our past work [10], we have introduced a variant of bounded model checking where the underlying, ultimately periodic timing structure was not bounded to be infinite only in the future, but may extend indefinitely also towards the past, thus allowing for a simple and intuitive modeling of continuously functioning systems like monitoring and control devices. In [11], we investigated the performance of verification in many case studies, showing that tool performance on bi-infinite structures is comparable to that on mono-infinite ones. Hence adopting a bi-infinite notion of time does not impose very significant penalties to the efficiency of bounded model checking and bounded satisfiability checking. Therefore, in what follows, we present only the simpler bi-infinite semantics of PLTL. Each experiment of Section 4 use either bi-infinite time (when there are past operators) or mono-infinite time (typically, when there are only future operators).

A bi-infinite word S over alphabet 2^{Ap} (also called a $\mathbb{Z}$-word) is a function $S : \mathbb{Z} \longrightarrow 2^{Ap}$. Hence, each position j of S, denoted by S_j, is in 2^{Ap} for every j. Word S is also denoted as $\ldots S_{-1}S_0S_1\ldots$. The set of all bi-infinite words over 2^{Ap} is denoted by $(2^{Ap})^{\mathbb{Z}}$.

For all PLTL formulae ϕ, for all $S \in (2^{Ap})^{\mathbb{Z}}$, for all integer numbers i, the satisfaction relation $S, i \models \phi$ is defined as follows.

$S, i \models p, \iff p \in S_i, \text{for } p \in Ap$
$S, i \models \neg\phi \iff S, i \not\models \phi$
$S, i \models \phi \vee \psi \iff S, i \models \phi \text{ or } S, i \models \psi$
$S, i \models \circ\phi \iff S, i+1 \models \phi$
$S, i \models \phi \mathcal{U} \psi \iff \exists k \geq 0 \mid S, i+k \models \psi, \text{ and } S, i+j \models \phi \; \forall 0 \leq j < k$
$S, i \models \bullet\phi \iff S, i-1 \models \phi$
$S, i \models \phi \mathcal{S} \psi \iff \exists k \geq 0 \mid S, i-k \models \psi, \text{ and } S, i-j \models \phi \; \forall 0 \leq j < k$

Metric temporal operators Metric operators are very convenient for modeling hard real time systems, with quantitative time constraints. The operators introduced in this section do not actually extend the expressive power of PLTL, but may lead to more succinct formulae. Their semantics is defined by a straightforward translation τ into PLTL.

Let $\sim \in \{\leq, =, \geq\}$), and c be a natural number. We consider here two metric operators, one in the future and one in the past: the bounded eventually $\Diamond_{\sim c}\phi$, and its past counterpart $\blacklozenge_{\sim c}\phi$. The semantics of the future operators is the following (the past versions are analogous):

$$\begin{array}{l} \tau(\Diamond_{=0}\phi) := \phi \\ \tau(\Diamond_{=t}\phi) := \circ\tau(\Diamond_{=t-1}\phi), \text{for } t > 0 \\ \hline \tau(\Diamond_{\leq 0}\phi) := \phi \\ \tau(\Diamond_{\leq t}\phi) := \phi \vee \circ\tau(\Diamond_{\leq t-1}\phi), \text{for } t > 0 \\ \hline \tau(\Diamond_{\geq 0}\phi) := \Diamond\phi \\ \tau(\Diamond_{\geq t}\phi) := \circ\tau(\Diamond_{\geq t-1}\phi), \text{for } t > 0 \end{array}$$

Versions of the bounded operators with $\sim \in \{<, >\}$ may be introduced as a shorthand. For instance, $\Diamond_{>0}\phi$ stands for $\circ\Diamond_{\geq 0}\phi$. Other two dual operators are "bounded globally": $\Box_{\sim c}\phi$ is $\neg\Diamond_{\sim c}\neg\phi$, and its past counterpart is $\blacksquare_{\sim c}\phi$, which is defined as $\neg\blacklozenge_{\sim c}\neg\phi$. Other metric operators are the bounded versions of $\mathcal{U}, \mathcal{R}, \mathcal{S}, \mathcal{T}$ ((see e.g. [10]), which are typically defined as primitive in metric temporal logics, with the various $\Diamond$ operators above defined as derived. In our experience, however, $\Diamond$ operators above are much more common in specifications, therefore we chose to implement them as native in the metric encoding. The latter bounded operators are instead translated into PLTL without a native metric encoding. For instance, $\phi_1 \mathcal{U}_{\leq t} \phi_2$ is defined by $\tau(\phi_1 \mathcal{U}_{\leq 0} \phi_2) := \phi_2$, $\tau(\phi_1 \mathcal{U}_{\leq t} \phi_2) := \phi_2 \vee (\phi_1 \wedge \circ\phi_1 \mathcal{U}_{\leq t-1} \phi_2)$ for $t > 0$. Hence, all operators of metric temporal logic are supported, although not all bounded operators have an optimized definition.

2.2 The Zot toolkit

Zot is an agile and easily extendible bounded model checker, which can be downloaded at http://home.dei.polimi.it/pradella/, together with the case studies and results described in Section 4. Zot provides a simple language to describe both descriptive and operational models, and to mix them freely. This is possible since both models are finally to be translated into boolean logic, to be fed to a SAT solver (Zot supports various SAT solvers, like MiniSat [3], and MiraXT [8]). 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 [4] is defined (essentially, equivalent to Metric PLTL). An interesting feature of Zot is its ability to support different encodings of temporal logic as SAT problems by means of plugins. This approach encourages experimentation, as plugins are expected to be quite simple, compact (usually around 500 lines of code), easily modifiable, and extendible.

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. *Bounded model checking (BMC)*: given as input an operational model of the system and a property, the tool returns a (possibly empty) history (i.e., an execution trace of the specified system) which satisfies it.

The provided output histories have temporal length $\leq k$, the bound *k* being chosen by the user, but may represent infinite behaviors thanks to the encoding techniques illustrated in Section 3. The BSC/BMC modalities 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 counterexample, explaining why $prop$ does not hold.

3 Encoding of metric temporal logic

We describe next the encoding of PLTL formulae into boolean logic, whose result includes additional information on the finite structure over which a formula is interpreted, so that the resulting boolean formula is satisfied in the finite structure if and only if the original PLTL formula is satisfied in a (finite or possibly) infinite structure. For simplicity, we present a variant of the bi-infinite encoding originally published in [10], and then introduce metric operators on it. Indeed, when past operators are introduced over a mono-infinite structure (e.g., [2]), however, the encoding can be tricky to define, because of the asymmetric role of future and past: future operators do extend to infinity, while past operators only deal with a finite prefix. The reader may refer to [10], and [11] for a more thorough comparison between mono- and bi-infinite approaches to bounded model checking. The complete mono-infinite encoding can be found in the extended version of the present paper [13].

For brevity in the following we call state S_i the set of assignments of truth values to propositional variables at time i. The idea on which the encoding is based is graphically depicted in Figure 1. A ultimately periodic bi-infinite structure has a finite representation that includes a non periodic portion, and two periodic portions (one towards the future, and one towards the past). The interpreter of the formula (in our case, the SAT solver), when it needs to evaluate a formula at a state beyond the last state S_k, will follow the "backward link" and consider the states S_l, S_{l+1}, ... as the states following S_k. Analogously, to evaluate a formula at a state precedent to the first state S_0, it will

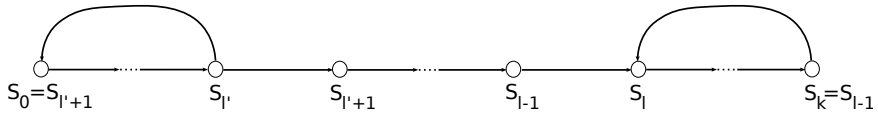

Fig. 1. A bi-infinite bounded path.

follow the "forward link" and consider the states S'_l, $S_{l'-1}$, ... as the states preceding S_0.

The encoding of the model (i.e. the operational description of the system, if any) is standard - see e.g. [2]. In the following we focus on the encoding of the logic part Φ of the system (or its properties).

Let Φ be a PLTL formula. Its semantics is given as a set of boolean constraints over the so called *formula variables*, i.e., fresh unconstrained propositional variables. There is a variable $|[\phi]|_i$ for each subformula ϕ of Φ and for each instant $0 \leq i \leq k+1$ (instant $k+1$, which is not explicitly shown in Figure 1, has a particular role in the encoding, as we will show next).

First, one needs to constrain the propositional operators in Φ. For instance, if $\phi_1 \wedge \phi_2$ is a subformula of Φ, then each variable $|[\phi_1 \wedge \phi_2]|_i$ must be equivalent to the conjunction of variables $|[\phi_1]|_i$ and $|[\phi_2]|_i$.

Propositional constraints, with p denoting a propositional symbol:

ϕ	$0 \leq i \leq k$
p	$\|[p]\|_i \iff p \in S_i$
$\neg p$	$\|[\neg p]\|_i \iff p \notin S_i$
$\phi_1 \wedge \phi_2$	$\|[\phi_1 \wedge \phi_2]\|_i \iff \|[\phi_1]\|_i \wedge \|[\phi_2]\|_i$
$\phi_1 \vee \phi_2$	$\|[\phi_1 \vee \phi_2]\|_i \iff \|[\phi_1]\|_i \vee \|[\phi_2]\|_i$

The following formulae define the basic temporal behavior of future PLTL operators, by using their traditional fixpoint characterizations.

Temporal subformulae constraints:

$$
\begin{array}{c|c}
\phi & -1 \leq i \leq k \\
\hline
\circ\phi_1 & |[\circ\phi_1]|_i \iff |[\phi_1]|_{i+1} \\
\phi_1 \mathcal{U} \phi_2 & |[\phi_1 \mathcal{U} \phi_2]|_i \iff |[\phi_2]|_i \vee (|[\phi_1]|_i \wedge |[\phi_1 \mathcal{U} \phi_2]|_{i+1}) \\
\phi_1 \mathcal{R} \phi_2 & |[\phi_1 \mathcal{R} \phi_2]|_i \iff |[\phi_2]|_i \wedge (|[\phi_1]|_i \vee |[\phi_1 \mathcal{R} \phi_2]|_{i+1})
\end{array}
$$

$$
\begin{array}{c|c}
\phi & 0 \leq i \leq k+1 \\
\hline
\bullet\phi_1 & |[\bullet\phi_1]|_i \iff |[\phi_1]|_{i-1} \\
\phi_1 \mathcal{S} \phi_2 & |[\phi_1 \mathcal{S} \phi_2]|_i \iff |[\phi_2]|_i \vee (|[\phi_1]|_i \wedge |[\phi_1 \mathcal{S} \phi_2]|_{i-1}) \\
\phi_1 \mathcal{T} \phi_2 & |[\phi_1 \mathcal{T} \phi_2]|_i \iff |[\phi_2]|_i \wedge (|[\phi_1]|_i \vee |[\phi_1 \mathcal{T} \phi_2]|_{i-1})
\end{array}
\tag{1}
$$

Notice that such constraints do not consider the implicit eventualities that the definitions of $\mathcal{U}$ and $\mathcal{S}$ impose (they treat them as the "weak" until and since operators), nor consider loops in the time structure.

To deal with eventualities and loops, one has to encode an infinite structure into a finite one composed of $k+1$ states $S_0, S_1, \ldots S_k$. The "future" loop can be described

by means of other $k+1$ fresh propositional variables $l_0, l_1, \ldots l_k$, called *loop selector variables*. At most one of these loop selector variables may be true. If l_i is true then state $S_{i-1} = S_k$, i.e., the bit vectors representing the state S_{i-1} are identical to those for state S_k. Further propositional variables, InLoop_i ($0 \leq i \leq k$) and $\mathrm{LoopExists}$, respectively mean that position i is inside a loop and that a loop actually exists in the structure. Symmetrically, there are new loop selector variables l'_i to define the loop which goes towards the past, and the corresponding propositional letters $\mathrm{InLoop'}_i$, and $\mathrm{LoopEsists'}$.

The variables defining the loops are constrained by the following set of formulae.

Loop constraints:

Base	$\neg l_0 \wedge \neg \mathrm{InLoop}_0 \wedge \neg l'_k \wedge \neg \mathrm{InLoop'}_k$
$1 \leq i \leq k$	$(l_i \Rightarrow S_{i-1} = S_k) \wedge (\mathrm{InLoop}_i \iff \mathrm{InLoop}_{i-1} \vee l_i)$
	$(\mathrm{InLoop}_{i-1} \Rightarrow \neg l_i) \wedge (\mathrm{LoopExists} \iff \mathrm{InLoop}_k)$
	$(l'_i \Rightarrow S_{i+1} = S_0) \wedge (\mathrm{InLoop'}_i \iff \mathrm{InLoop'}_{i+1} \vee l'_i)$
	$(\mathrm{InLoop'}_{i+1} \Rightarrow \neg l'_i) \wedge (\mathrm{LoopExists'} \iff \mathrm{InLoop'}_0)$

The above loop constraints state that the structure may have at most one loop in the future and at most one loop in the past. In the case of a cyclic structure, they allow the SAT solver to select nondeterministically exactly one of the (possibly) many loops.

To properly define eventualities, we need to introduce new propositional letters $\langle\langle \Diamond \phi_2 \rangle\rangle_i$, for each $\phi_1 \mathcal{U} \phi_2$ subformula of Φ, and for every $0 \leq i \leq k+1$. Analogously, we need to consider subformulae containing the operator $\mathcal{R}$, such as $\phi_1 \mathcal{R} \phi_2$, by adding the new propositional letters $\langle\langle \Box \phi_2 \rangle\rangle_i$. This is also symmetrically applied to $\mathcal{S}$ and $\mathcal{T}$, using $\blacklozenge, \blacksquare$. Then, constraints on these eventuality propositions are quite naturally stated as follows.

Eventuality constraints:

ϕ	Base
$\phi_1 \mathcal{U} \phi_2$	$\neg\langle\langle \Diamond \phi_2 \rangle\rangle_0 \wedge \big(\mathrm{LoopExists} \Rightarrow \big(\lvert[\phi_1 \mathcal{U} \phi_2]\rvert_k \Rightarrow \langle\langle \Diamond \phi_2 \rangle\rangle_k \big)\big)$
$\phi_1 \mathcal{R} \phi_2$	$\langle\langle \Box \phi_2 \rangle\rangle_0 \wedge \big(\mathrm{LoopExists} \Rightarrow \big(\lvert[\phi_1 \mathcal{R} \phi_2]\rvert_k \Leftarrow \langle\langle \Box \phi_2 \rangle\rangle_k \big)\big)$
$\phi_1 \mathcal{S} \phi_2$	$\neg\langle\langle \blacklozenge \phi_2 \rangle\rangle_k \wedge \big(\mathrm{LoopExists'} \Rightarrow \big(\lvert[\phi_1 \mathcal{S} \phi_2]\rvert_0 \Rightarrow \langle\langle \blacklozenge \phi_2 \rangle\rangle_0 \big)\big)$
$\phi_1 \mathcal{T} \phi_2$	$\langle\langle \blacksquare \phi_2 \rangle\rangle_k \wedge \big(\mathrm{LoopExists'} \Rightarrow \big(\lvert[\phi_1 \mathcal{T} \phi_2]\rvert_0 \Leftarrow \langle\langle \blacksquare \phi_2 \rangle\rangle_0 \big)\big)$

ϕ	$1 \leq i \leq k$
$\phi_1 \mathcal{U} \phi_2$	$\langle\langle \Diamond \phi_2 \rangle\rangle_i \iff \langle\langle \Diamond \phi_2 \rangle\rangle_{i-1} \vee (\mathrm{InLoop}_i \wedge \lvert[\phi_2]\rvert_i)$
$\phi_1 \mathcal{R} \phi_2$	$\langle\langle \Box \phi_2 \rangle\rangle_i \iff \langle\langle \Box \phi_2 \rangle\rangle_{i-1} \wedge (\neg \mathrm{InLoop}_i \vee \lvert[\phi_2]\rvert_i)$

ϕ	$0 \leq i \leq k-1$
$\phi_1 \mathcal{S} \phi_2$	$\langle\langle \blacklozenge \phi_2 \rangle\rangle_i \iff \langle\langle \blacklozenge \phi_2 \rangle\rangle_{i+1} \vee \big(\mathrm{InLoop'}_i \wedge \lvert[\phi_2]\rvert_i\big)$
$\phi_1 \mathcal{T} \phi_2$	$\langle\langle \blacksquare \phi_2 \rangle\rangle_i \iff \langle\langle \blacksquare \phi_2 \rangle\rangle_{i+1} \wedge \big(\neg \mathrm{InLoop'}_i \vee \lvert[\phi_2]\rvert_i\big)$

The formulae in the following table provide the constraints that must be included in the encoding, for any subformula ϕ, to account for the absence of a forward loop in the structure (the first line of the table states that if there is no loop nothing is true beyond the k-th state) or its presence (the second line states that if there is a loop at position i then state S_{k+1} and S_i are equivalent).

Last state constraints:

$$\frac{\text{Base} \quad \left| \neg\text{LoopExists} \Rightarrow \neg|[\phi]|_{k+1} \right.}{1 \leq i \leq k \left| l_i \Rightarrow (|[\phi]|_{k+1} \iff |[\phi]|_i) \right.} \tag{2}$$

Then, symmetrically to the last state, we must define first state (i.e. 0 time) constraints (notice that in the bi-infinite encoding instant -1 has a symmetric role of instant $k+1$).

First state constraints:

$$\frac{\text{Base} \quad \left| \neg\text{LoopExists}' \Rightarrow \neg|[\phi]|_{-1} \right.}{0 \leq i \leq k-1 \left| l'_i \Rightarrow (|[\phi]|_{-1} \iff |[\phi]|_i) \right.} \tag{3}$$

The complete encoding of Φ consists of the logical conjunction of all above components, together with $|[\Phi]|_0$ (i.e. Φ is evaluated only at instant 0).

3.1 Encoding of the metric operators

We present here the additional constraints one has to add to the previous encoding, to natively support metric operators. We actually implemented also a mono-infinite metric encoding in Zot, but for simplicity we are focusing here only on the bi-infinite one.

Notice that

$$\Diamond_{\leq t}\phi \iff \neg\Box_{\leq t}\neg\phi, \qquad \Diamond_{=t}\phi \iff \Box_{=t}\phi, \qquad \Diamond_{\geq t}\phi \iff \Diamond_{=t}\Diamond\phi$$

(the past versions are analogous). Hence, in the following we will not consider the $\Box_{=t}$, $\Diamond_{\geq t}$, $\Box_{\geq t}$ operators, and their past counterparts.

Ideally, with an *unbounded* time structure, the encoding of the metric operators should be the following one (considering only the future, as the past is symmetrical):

$$|[\Diamond_{=t}\phi]|_i \iff |[\phi]|_{i+t}, \qquad\qquad |[\Box_{\leq t}\phi]|_i \iff \bigwedge_{j=1}^{t} |[\phi]|_{i+j}$$

Unfortunately, the presence of a *bounded* time structure, in which bi-infinity is encoded through loops, makes the encoding less straightforward. With simple PLTL one refers at most to one instant in the future (or in the past) or to an eventuality. As the reader may notice in the foregoing encoding, this is still quite easy, also in the presence of loops. On the other hand, the presence of metric operators, affects the loop-based structure, as logic formulae can now refer to time instants well beyond a single future (or past) unrolling of the loop.

To represent the values of subformulae inside the future and past loops, we introduce new propositional variables, $\langle\langle \text{MF}(\cdot,\cdot)\rangle\rangle$ for the future-tense operators, and $\langle\langle \text{MP}(\cdot,\cdot)\rangle\rangle$ for the past ones. For instance, for $\Diamond_{=5}\psi$, we introduce $\langle\langle \text{MF}(\psi, j)\rangle\rangle$, $0 \leq j \leq 4$, where the propositions $\langle\langle \text{MF}(\psi, j)\rangle\rangle$ are used to represent the value of ψ *j* time units after the starting point of the future loop. This means that, if the future loop selector is at instant 18 (i.e. l_{18} holds), then $\langle\langle \text{MF}(\psi, 2)\rangle\rangle$ represents $|[\psi]|_{20}$ (i.e. ψ at instant 18+2). Analogously and symmetrically, $\langle\langle \text{MP}(\psi, j)\rangle\rangle$ are introduced for past operators

with argument ψ, and represent the value of ψ j time units after the starting point of the past loop. That is, if the past loop selector is at instant 7 (i.e. l'_7), then $\langle\langle \text{MP}(\psi, 2)\rangle\rangle$ represents $|[\psi]|_{7-2}$.

The first constraints are introduced for any future or past metric formulae in Φ.

$$\begin{array}{c|c} \phi & 0 \leq j \leq t-1 \\ \hline \Diamond_{=t}\phi,\ \Box_{\leq t}\phi,\ \Diamond_{\leq t}\phi & \langle\langle \text{MF}(\phi, j)\rangle\rangle \iff \bigvee_{i=1}^{k} l_i \wedge |[\phi]|_{i+\text{mod}(j,k-i+1)} \\ \blacklozenge_{=t}\phi,\ \blacksquare_{\leq t}\phi,\ \blacklozenge_{\leq t}\phi & \langle\langle \text{MP}(\phi, j)\rangle\rangle \iff \bigvee_{i=0}^{k-1} l'_i \wedge |[\phi]|_{i-\text{mod}(j,i+1)} \end{array} \tag{4}$$

We now provide the encoding of every metric operator, composed of two parts: the first one defines it inside the bounded portion of the temporal structure (i.e. for instants i in $0 \leq i \leq k$), and the other one, based on *MF* and *MP*, for the loop portion.

$$\begin{array}{c|c} \phi & -1 \leq i \leq k \\ \hline \Diamond_{=t}\phi & |[\Diamond_{=t}\phi]|_i \iff |[\phi]|_{i+t}, \text{ when } i+t \leq k \\ & |[\Diamond_{=t}\phi]|_i \iff \langle\langle \text{MF}(\phi, t+i-k-1)\rangle\rangle, \text{ elsewhere} \\ \Box_{\leq t}\phi & \Box_{\leq t}\phi \iff \bigwedge_{j=1}^{\min(t,k-i)} |[\phi]|_{i+j} \wedge \bigwedge_{j=k+1-i}^{t} \langle\langle \text{MF}(\phi, i+j-k-1)\rangle\rangle \\ \Diamond_{\leq t}\phi & \Diamond_{\leq t}\phi \iff \bigvee_{j=1}^{\min(t,k-i)} |[\phi]|_{i+j} \vee \bigvee_{j=k+1-i}^{t} \langle\langle \text{MF}(\phi, i+j-k-1)\rangle\rangle \end{array} \tag{5}$$

$$\begin{array}{c|c} \phi & 0 \leq i \leq k+1 \\ \hline \blacklozenge_{=t}\phi & |[\blacklozenge_{=t}\phi]|_i \iff |[\phi]|_{i-t}, \text{ when } i \geq t \\ & |[\blacklozenge_{=t}\phi]|_i \iff \langle\langle \text{MP}(\phi, t-i-1)\rangle\rangle, \text{ elsewhere} \\ \blacksquare_{\leq t}\phi & \blacksquare_{\leq t}\phi \iff \bigwedge_{j=1}^{\min(t,i)} |[\phi]|_{i-j} \wedge \bigwedge_{j=i+1}^{t} \langle\langle \text{MP}(\phi, i+j-1)\rangle\rangle \\ \blacklozenge_{\leq t}\phi & \blacklozenge_{\leq t}\phi \iff \bigvee_{j=1}^{\min(t,i)} |[\phi]|_{i-j} \vee \bigvee_{j=i+1}^{t} \langle\langle \text{MP}(\phi, i+j-1)\rangle\rangle \end{array}$$

The most complex part of the metric encoding is the one considering the behavior on the past loop of future operators, and on the future loop of the past operators. First, let us consider the behavior of future metric operators on the past loop.

$$\begin{array}{c|c} \phi & 0 \leq i \leq k-1 \\ \hline \Diamond_{=t}\phi & l'_i \Rightarrow \left(\begin{array}{c} \bigwedge_{j=1}^{\min(t,k-i)} (|[\phi]|_{i+j} \iff |[\phi]|_{\text{mod}(j-1,i+1)}) \wedge \\ \bigwedge_{j=k-i+1}^{t} \left(\begin{array}{c} \langle\langle \text{MF}(\phi, i+j-k-1)\rangle\rangle \iff \\ |[\phi]|_{\text{mod}(j-1,i+1)} \end{array} \right) \end{array} \right) \\ \Box_{\leq t}\phi & \text{InLoop}'_i \Rightarrow \left(|[\Box_{\leq t}\phi]|_i \iff \left(\begin{array}{c} \bigwedge_{j=1}^{\min(k-i,t)} (\neg\text{InLoop}'_{i+j} \vee |[\phi]|_{i+j}) \wedge \\ \bigwedge_{j=0}^{\min(i,t-1)} (\text{InLoop}'_{\min(k,i+t-j)} \vee |[\phi]|_j) \end{array} \right) \right) \\ \Diamond_{\leq t}\phi & \text{InLoop}'_i \Rightarrow \left(|[\Diamond_{\leq t}\phi]|_i \iff \left(\begin{array}{c} \bigvee_{j=1}^{\min(k-i,t)} (\text{InLoop}'_{i+j} \wedge |[\phi]|_{i+j}) \vee \\ \bigvee_{j=0}^{\min(i,t-1)} (\neg\text{InLoop}'_{\min(k,i+t-j)} \wedge |[\phi]|_j) \end{array} \right) \right) \end{array} \tag{6}$$

The main aspect to consider is the fact that, if l'_i (i.e. the past loop selector variable holds at instant i), then i has two possible successors: $i+1$ and 0. Therefore, if $\Diamond_{=4}\phi$ holds at i (which is inside the past loop), then ϕ must hold both at $i+4$, and at 3. This kind of constraint is captured by the upper formula for $\Diamond_{=t}\phi$, which relates the truth

values of ϕ in instants outside of the past loop (i.e., $|[\phi]|_{i+j}$) with the instants inside (i.e., $|[\phi]|_{\mathrm{mod}(j-1,i+1)}$ represents the value of ϕ at instants going from 0 to i, if l'_i holds).

Another aspect to consider is related to the size of the time constant used (i.e. t in this case). Indeed, if $i + t > k$, then we are considering the behavior of ϕ outside the bound $0..k$. This means that we need to consider the behavior of ϕ also in the future loop, hence we refer to $\langle\langle \mathrm{MF}(\phi, i + j - k - 1)\rangle\rangle$ (see the lower formula for $\diamondsuit_{=t}\phi$).

As far as $\Box_{\leq t}\phi$ is concerned, its behavior inside the past loop is in general expressed by two parts. The first one considers ϕ inside the past loop, starting from instant i and going forward, towards the right end of the loop (i.e. where l' holds, say i'). This situation is covered by the upper formula for $\Box_{\leq t}\phi$. If $i + t$ is still inside the past loop (i.e. $i + t \leq i'$), this suffices. If this is not the case, we must consider the remaining instants, going from $i' + 1$ to $i + t$. Because we are considering the behavior *inside* the past loop, the instant after i' is 0, so we must translate instants outside of the loop (i.e. where $InLoop'$ does not hold), to instants going from 0 to $i + t - i' - 1$: in all these instants ϕ must hold. This constraint is given by the lower formula for $\Box_{\leq t}\phi$.

The encoding for the past operators is symmetrical, and is the following:

	$1 \leq i \leq k$
$\blacklozenge_{=t}\phi$	$l_i \Rightarrow \left(\begin{array}{c} \bigwedge_{j=2}^{\min(t,i)} (\lvert[\phi]\rvert_{i-j} \iff \lvert[\phi]\rvert_{k-\mathrm{mod}(j-1,k-i+1)}) \wedge \\ \bigwedge_{j=1+i}^{t} \left(\begin{array}{c} \langle\langle \mathrm{MP}(\phi, j-i-1)\rangle\rangle \iff \\ \lvert[\phi]\rvert_{k-\mathrm{mod}(j-1,k-i+1)} \end{array} \right) \end{array} \right)$
$\blacksquare_{\leq t}\phi$	$\mathrm{InLoop}_i \Rightarrow \left(\lvert[\blacksquare_{\leq t}\phi]\rvert_i \iff \left(\begin{array}{c} \bigwedge_{j=1}^{\min(i,t)} (\neg \mathrm{InLoop}_{i-j} \vee \lvert[\phi]\rvert_{i-j}) \wedge \\ \bigwedge_{j=0}^{\min(k-i,t-1)} (\mathrm{InLoop}_{\max(0,i-t+j)} \vee \lvert[\phi]\rvert_{k-j}) \end{array} \right) \right)$
$\blacklozenge_{\leq t}\phi$	$\mathrm{InLoop}_i \Rightarrow \left(\lvert[\blacklozenge_{\leq t}\phi]\rvert_i \iff \left(\begin{array}{c} \bigvee_{j=1}^{\min(i,t)} (\mathrm{InLoop}_{i-j} \wedge \lvert[\phi]\rvert_{i-j}) \vee \\ \bigvee_{j=0}^{\min(k-i,t-1)} (\neg \mathrm{InLoop}_{\max(0,i-t+j)} \wedge \lvert[\phi]\rvert_{k-j}) \end{array} \right) \right)$

The actual implementation of the metric encoding contains some optimizations, not reported here for the sake of brevity, like the re-use, whenever possible, of the various $\langle\langle \mathrm{MF}(\cdot,\cdot)\rangle\rangle$, and $\langle\langle \mathrm{MP}(\cdot,\cdot)\rangle\rangle$ propositional letters.

A first assessment of the encoding The behavior of the new encoding has been first experimented on a very simple specification of a synchronous shift-register, where, at each clock tick, an input bit is shifted of one position to the right. A specification of this system can be described by the following formula:

$$Alw(in \leftrightarrow \diamondsuit_{=d} out)$$

where in is true when a bit enters the shift register, out is true when a bit "exits" the register after a delay $d > 0$ (a constant representing the number of memory bits in the register). The Zot toolkit has been applied to this simple specification, using the nonmetric, PLTL-only encoding of ([10]) and the new metric encoding.

The implemented nonmetric encoding is actually the one presented in the current section, without the metric part of Sub-section 3.1. In practice, this means that every metric temporal operator is translated into PLTL before applying the encoding, by means of its definition of Section 2.

The experimental results (with the same hardware and software setup described in Section 4.1) are graphically shown in Figure 2, where Gen represents the generation phase, i.e., the generation of a boolean formula in conjunctive normal form, starting from the above specification, and SAT represents the verification phase, performed by a SAT solver, with a bound $k = 400$ and various values of delay d (from 10 to 150). The first two upper diagrams show the time, in seconds, for Gen and SAT phases, using either a PLTL encoding or the metric encoding, as a function of delay d, while the third upper diagram shows the speedup, as a percentage of speed increase over the PLTL encoding, when using the metric encoding, again as a function of delay d. As one can see, the speedup obtained for both the Gen and SAT phases is proportional to delay d, and can be quite substantial (up to 250% for SAT and 300% for Gen phases). The three lower diagrams report, in a similar way, on the size of the generated boolean formula, in terms of the thousands of variables (Kvar) and clauses (Kcl): the reduction in the size of the generated encoding increases with the value of d and tends to reach a stable value around 60%.

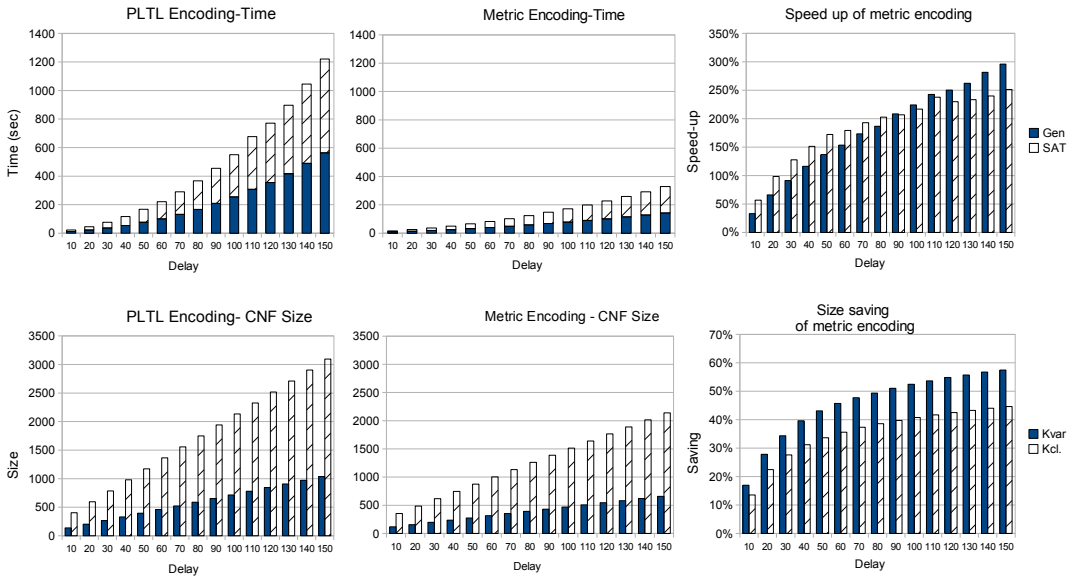

Fig. 2. Summary of experimental data for the synchronous version of a Shift Register.

These results can be explained by comparing the two encodings. In general, if a formula ϕ contains a time constant, then the number of subformulae is much higher in the nonmetric enconding than in the metric one. For instance, in the above example, the non-metric encoding of $\Diamond_{=d} out$ is translated into d nested applications of the next-time operator, $\circ^d out$, hence there are $d + 1$ subformulae, $\circ^i out$ for $0 \leq i \leq d$, while in the metric encoding of $\Diamond_{=d} out$ there are only two subformulae, $\Diamond_{=d} out$ itself and

out. Concerning the number of generated boolean variables, this is much higher for the nonmetric encoding, due to the presence of a larger number of subformulae. Again with reference to the example, we have $(d+1)\cdot(k+2)$ subformula variables in the nonmetric case and $2\cdot(k+2)+2\cdot d$ variables for the metric encoding (of which $2\cdot(k+2)$ subformula variables and $2\cdot d$ MF and MP variables).

Regarding the size of the generated constraints, it is immediate to notice that propositional, eventuality and loop contraints have the same size, which is $O(k)$, in both encodings. The size of the remaining constraints is shown in the following table, where MF and MP constraints are those of type (4), (5), and (6) introduced for the metric encoding only.

	First and last state	Temporal Subformula	MF and MP	Total
Nonmetric	$(d+1)\cdot(k+2)$	$2\cdot k(d+1)$	0	$3\cdot d\cdot k+2\cdot d+3\cdot k+2$
Metric	0	$4\cdot k$	$2\cdot d\cdot k+k$	$2\cdot d\cdot k+5\cdot k$

Thus, in the metric encoding we have $O(d+k)$ (i.e., less than in the nonmetric case) variables and $O(d\cdot k)$ constraints, which is same as in the nonmetric case but with a smaller constant factor (in this case, 2 rather than 3). This is also clear from Fig. 2, where the size saving tends to a constant when d is large enough.

The analysis of the other metric temporal operators, $\Box_{\leq t}\phi$ and $\Diamond_{\leq t}\phi$, leads to similar conclusions.

4 Experimental results

First we briefly describe the five case studies that we adopted for our experiments. For all of them we provide both a descriptive and an operational model. A complete archive with the files used for the experiments, and the details of the outcomes, can be found in the Zot web page at http://home.dei.polimi.it/pradella/.

Real-time allocator (RTA) This case study, described in [12], consists of a real-time allocator which serves a set of client processes, competing for a shared resource. The system numeric parameters are the number of processes n_p and the constants T_{req} within which the allocator must respond to the requests, and the maximum time T_{rel} that a process can keep the resource before releasing it. In our experiments, both a descriptive and an operational model were considered, using three processes, and with two different system settings for each version: a first one with $T_{rel}=T_{req}=3$, and a second one with $T_{rel}=T_{req}=10$. We first generated a simple run of the system (Property Sat); then we considered four hard real time properties, described in [12], called *Simple Fairness*, *Conditional Fairness*, *Precedence*, and *Suspend Fairness*. It is worth noticing that the formula specifying *Suspend Fairness* includes a relatively high time constant ($T_{rel}\cdot n_p$) and is therefore likely to benefit from the metric encoding. We adopted the bi-infinite encoding for this case study, which allowed to consider only regime behaviors, thus abstracting away system initialization.

Fischer's protocol (FP) FP [7] is a timed mutual exclusion algorithm that allows a number of timed processes to access a shared resource. We considered the system in two variants: one with 3 processes and a delay 5 t.u.; the other one with 4 processes and a delay of 10 t.u. We used the tool to check the safety property (i.e. it is never possible

that two different processes enter their critical sections at the same time instant) and to generate a behavior in which there is always at least one alive process. We adopted the bi-infinite encoding, for reasons similar to those already explained for RTA case study.

Kernel Railway Crossing (KRC) This is a standard benchmark in real time systems verification [5], which we used and described in a previous work [12]. In our example we adopted a descriptive model and studied the KRC problem with two sets of time constants, allowing a high degree of nondeterminism on train behavior. In particular, the first set of constants was: $d_{Max} = 9$ and $d_{min} = 5$ t.u. for the maximum and minimum time for a train to reach the critical region, $h_{Max} = 6$ and $h_{min} = 3$ for the maximum and minimum time for a train to enter the critical region once it is first sensed, and $\gamma = 3$ for the movement of the bar from up to down and vice versa. The set of time constants for the second experiment was $d_{Max} = 19$, $d_{min} = 15$, $h_{Max} = 16$, $h_{min} = 13$, and $\gamma = 10$. For each of the two settings we proved both satisfiability of the specification (Sat) and the safety property, using a mono-infinite encoding.

Timer Reset Lamp (TRL) This is the Timer Reset Lamp first presented in [12], with three settings ($\Delta = 10$, $\Delta = 15$, and $\Delta = 20$) and two analyzed properties (the first one, that the lamp is never lighted for more than Δ t.u.: it is false, and the tool generates a counter-example; the second one, namely that the lamp can remain lighted for more than Δ t.u. only if the *ON* button is pushed twice within Δ t.u., is true). This system was analyzed with a bi-infinite encoding.

Asynchronous Shift Register (ASR) The simplest case study is an *asynchronous* version of the Shift Register example discussed in Section 3, where the shift does not occur at every tick of the clock, but only at a special, completely asynchronous *Shift* command. We consider two cases, with the number of bits $n = 16$ and $n = 24$, and we prove satisfiability of the specification and one timed property (if the *Shift* signal remains true for *n* time units (t.u.) then the value *In* which was inserted in the Shift register at the beginning of the time interval will appear at the opposite side of the register at the end of the time interval). This case study was analyzed with reference to a bi-infinite encoding.

4.1 Results

The experiments were run on a PC equipped with two XEON 5335 (Quadcore) processors at 2.0 Ghz, with 16 GB RAM, running under Gentoo X86-64 (2008.0). The SAT-solver was MiniSat. The experimental results are shown in Table 1. The suffix *-de* indicates analysis carried out on the descriptive version of the model, while *-op* is used for the operational version. The table reports, for various values of the bound k (30, 60, and 90), both Generation time, i.e., the time in seconds taken for building the encoding and transforming it into conjunctive normal form, and SAT time, i.e., the time in seconds taken by the SAT solver to answer. Only the timings of the metric version is reported, since the ones of the non-metric version can be obtained by the following speed up measures. Performance is gauged by providing three measures of speed up as a percentage of the time taken by the metric version (e.g., 0% means no speed-up, 100% means double speed, i.e., the encoding is twice as fast, etc.): $\frac{T_{\texttt{PLTL}} - T_{\texttt{metric}}}{T_{\texttt{metric}}}$, where $T_{\texttt{metric}}$ and $T_{\texttt{PLTL}}$ represent the time taken by the metric and the PLTL encodings, respectively. The first measure shows the speed up in the generation phase, the second in

Case	Property	Gen. (s): Metric			SAT (s): Metric			Gen. Speed-up			SAT Speed up			Total Speed-up		
		k=30	k=60	k=90	k=30	k=60	k=90	k=30	k=60	k=90	k=30	k=60	k=90	k=30	k=60	k=90
RTA-3-de	Sat	14,6	42,4	84,0	12,6	45,1	99,5	4%	2%	2%	2%	4%	1%	3%	3%	1%
	Simple	16,8	49,6	97,6	14,9	54,6	117,8	8%	8%	9%	11%	10%	11%	10%	9%	10%
	Cond	19,7	59,1	116,8	18,6	68,2	146,7	9%	9%	9%	14%	9%	20%	12%	9%	15%
	Prec	22,4	69,7	142,0	21,5	78,3	170,9	2%	−1%	−2%	1%	1%	5%	2%	0%	2%
	Suspend	18,5	55,3	109,3	19,6	64,2	145,1	28%	31%	33%	35%	38%	47%	31%	35%	41%
RTA-3-op	Sat	2,4	5,3	8,8	0,9	2,6	5,2	3%	9%	12%	2%	2%	2%	3%	6%	8%
	Simple	3,6	8,4	14,4	1,7	5,3	11,2	14%	24%	25%	39%	30%	24%	22%	26%	25%
	Cond	5,1	12,7	22,4	3,0	9,1	19,2	16%	25%	22%	27%	26%	32%	20%	26%	27%
	prec	6,6	17,1	31,3	4,0	13,9	29,2	0%	5%	1%	4%	−2%	−1%	1%	2%	0%
	suspend	4,7	11,2	19,7	3,2	11,1	21,4	49%	72%	75%	94%	58%	74%	67%	65%	75%
RTA-10-de	Sat	72,1	243,9	506,8	82,8	324,3	779,9	7%	3%	4%	4%	4%	−8%	5%	4%	−3%
	Simple	76,9	255,2	541,8	100,0	334,9	768,6	20%	18%	19%	23%	24%	−34%	22%	21%	−13%
	Cond	83,9	277,7	586,2	100,7	384,2	539,9	17%	17%	18%	27%	21%	−34%	22%	20%	−7%
	Prec	103,2	344,6	734,5	124,7	498,6	66,4	6%	4%	3%	5%	−2%	11%	6%	0%	4%
	Suspend	85,3	274,3	577,1	94,8	419,3	6294	53%	63%	62%	79%	57%	−23%	67%	60%	−16%
RTA-10-op	Sat	6,3	13,6	24,7	2,5	7,4	18,2	−1%	3%	1%	0%	0%	−6%	−1%	2%	−2%
	Simple	8,1	19,2	33,4	14,7	19,1	50,9	32%	40%	45%	12%	82%	18%	19%	61%	29%
	Cond	9,9	24,6	42,4	6,5	24,0	37,0	29%	36%	46%	44%	31%	63%	35%	33%	54%
	Prec	15,4	43,0	78,5	10,3	36,2	75,5	2%	−1%	5%	3%	1%	3%	2%	0%	4%
	Suspend	9,6	24,1	44,5	4,7	45,3	633,6	159%	196%	224%	377%	136%	88%	231%	157%	97%
FP-3-5-de	Sat	11,4	31,7	60,3	9,2	31,1	66,9	28%	33%	37%	39%	49%	51%	33%	41%	44%
	Safety	11,9	32,5	62,6	9,5	34,1	73,9	28%	33%	37%	42%	44%	50%	34%	39%	44%
FP-3-5-op	Sat	2,7	6,2	10,4	1,3	3,9	7,1	0%	−1%	0%	−1%	−3%	0%	0%	−2%	0%
	Safety	2,9	6,7	11,7	1,5	4,7	10,8	0%	−1%	0%	−3%	0%	0%	−1%	−1%	0%
FP-4-10-de	Sat	26,3	80,3	159,0	25,6	94,9	200,6	67%	74%	81%	95%	115%	102%	81%	96%	93%
	Safety	27,3	83,7	163,1	27,7	101,2	221,0	70%	71%	80%	87%	90%	89%	78%	82%	85%
FP-4-10-op	Sat	4,3	10,6	18,0	4,3	8,4	15,3	0%	−1%	−1%	0%	0%	0%	0%	−1%	0%
	Safety	4,8	11,6	19,9	4,1	14,9	27,1	−1%	−1%	−1%	0%	0%	−1%	−1%	0%	−1%
KRC-9-5-6-3-de	Sat	2,1	4,7	7,9	0,9	3,3	6,3	24%	30%	34%	57%	34%	80%	34%	32%	54%
	Safety	2,2	5,0	8,4	0,2	0,4	0,6	23%	29%	36%	26%	32%	38%	23%	29%	36%
KRC-9-5-6-3-op	Sat	1,4	3,0	4,8	0,5	1,3	2,9	−1%	−1%	−0%	0%	0%	1%	0%	−1%	0%
	Safety	1,5	3,2	5,5	0,1	0,2	0,3	−2%	−2%	−6%	0%	0%	−1%	−2%	−2%	−6%
KRC-19-15-16-13-de	Sat	2,4	5,4	9,1	1,0	3,4	6,4	102%	127%	149%	213%	231%	279%	135%	167%	202%
	Safety	2,5	5,7	9,6	0,2	0,4	0,7	95%	124%	146%	115%	145%	175%	96%	125%	148%
KRC-19-15-16-13-op	Sat	2,0	4,2	6,8	1,0	1,8	4,0	−1%	−1%	3%	−1%	−1%	−1%	−1%	−1%	1%
	Safety	2,0	4,4	7,1	0,1	0,3	0,4	−1%	0%	0%	2%	6%	0%	0%	0%	0%
TRL-10-de	p1	2,9	6,9	11,6	1,4	4,4	9,2	34%	39%	48%	61%	67%	70%	43%	50%	58%
	p2	3,1	8,0	13,7	2,1	8,5	12,5	49%	49%	61%	66%	60%	83%	56%	55%	71%
TRL-10-op	p1	1,1	2,3	3,6	0,3	0,9	1,7	35%	39%	47%	71%	68%	85%	43%	47%	59%
	p2	1,4	3,0	4,7	0,5	1,5	2,9	57%	61%	75%	105%	121%	253%	70%	81%	143%
TRL-15-de	p1	3,9	9,2	16,3	2,1	6,6	14,0	43%	56%	68%	71%	90%	87%	53%	70%	77%
	p2	4,3	10,9	18,3	2,6	12,2	31,0	68%	72%	90%	110%	97%	35%	84%	85%	55%
TRL-15-op	p1	1,1	2,4	3,7	0,3	0,9	1,6	63%	60%	69%	104%	117%	140%	72%	75%	91%
	p2	1,4	3,1	5,0	0,5	1,6	3,0	95%	107%	130%	186%	208%	242%	120%	141%	172%
TRL-20-de	p1	5,3	13,0	23,0	3,0	10,2	21,8	49%	69%	77%	91%	97%	99%	65%	81%	88%
	p2	5,7	14,4	26,5	3,6	15,7	46,9	77%	91%	100%	148%	166%	183%	104%	130%	153%
TRL-20-op	p1	1,4	3,1	5,1	0,4	1,1	2,1	67%	67%	64%	122%	145%	157%	79%	88%	92%
	p2	1,8	3,8	6,3	0,6	2,5	8,5	104%	134%	143%	242%	226%	84%	140%	170%	109%
ASR-24-de	Sat	11,3	31,3	59,4	14,6	31,4	67,9	3%	−1%	−1%	−4%	0%	−1%	−1%	0%	−1%
	Prop	12,7	33,8	64,0	9,8	34,4	78,5	22%	31%	35%	41%	38%	30%	31%	35%	32%
ASR-24-op	Sat	1,9	4,4	6,9	1,7	1,9	3,7	0%	−7%	−1%	−1%	−1%	−1%	0%	−5%	−1%
	Prop	2,5	5,9	9,4	1,0	3,2	5,7	68%	73%	92%	118%	125%	168%	83%	92%	121%
ASR-16-de	Sat	6,7	17,6	33,2	4,6	15,7	33,4	0%	2%	2%	0%	0%	0%	0%	1%	1%
	Prop	7,4	20,0	36,4	5,1	18,3	40,3	22%	25%	28%	34%	31%	27%	27%	28%	27%
ASR-16-op	Sat	1,4	3,0	5,0	0,7	1,3	2,4	−2%	7%	2%	3%	6%	3%	0%	6%	2%
	Prop	1,9	4,2	6,9	0,7	2,1	3,7	57%	67%	69%	94%	97%	131%	67%	77%	90%

Table 1. Summary of collected experimental data.

SAT time and the third one in Total time (i.e., in the sum of Gen and SAT time). On average, the speed up is 42,2% for Gen and 62,2% for SAT, allowing for a 47,9% speed up in the total time. The best results give speed up of, respectively, 224%, 377% and 231%, while the worst results are -7%, -34% and -16%.

Speed up for SAT time appears to be more variable and less predictable than the one for Gen time, although often significantly larger. This is likely caused by the complex and involved ways in which the SAT algorithm is influenced by the numerical values of the k bound, of the time constants in the specification formulae and by their interaction, due to the heuristics that it incorporates. For instance, the speed up for Gen increases very regularly with the bound k, because of the smaller size of the formula to be generated, while SAT may vary unpredictably and significantly with the value of k (e.g., compare property op-P2 for TRL-10, when the speed up increases with k, and TRL-20, when the speed up actually decreases with k). A thorough discussion of these aspects is out of the scope of the present paper, also because they may change from one SAT-solver to another one.

It is easy to realize, as already noticed in Section 3 for the example of the synchronous shift register, that significant improvements are obtained, with the new metric encoding, for analysing Metric temporal logic properties with time constants having a fairly high numerical value. The larger the value, the larger the speed up. This is particularly clear for TRL, RTA and FP case studies.

The fact that the underlying model was descriptive or operational may have a significant impact on verification speed, but considering only the speed up the results are much more mixed. For instance, the operational versions of FP and KRC, although more efficient, had a worse speed up than their corresponding descriptive cases, while the reverse occurred for the operational versions of RTA, ASR and TRL. The only exception is for the Sat case, where no property is checked against the model, and hence no gain can be obtained for the operational model. A decrease in benefit for certain descriptive models may be caused by cases where subformulae in metric temporal logic with large time constants are combined with other non-metric subformulae.

The measure of the size of the generated formulae is not reported here, but it is worth pointing out that, thanks to the new metric encoding, the size may be reduced of 50% or more when there are high time constants and/or large k bounds. For instance, in case KRC-19-15-16-13-de (Safety) the number of clauses of the metric encoding is less than half of the nonmetric one, while in case FP-4-10-de (both sat and Safety), the metric encoding is around one third smaller.

5 Conclusions

In this paper, a new encoding technique of linear temporal logic into boolean logic is introduced, particularly optimized for managing quantitative future and past metric temporal operators. The encoding is simple and intuitive in principle, but it is made more complex by the presence, typical of the technique, of backward and forward loops used to represent an ultimately periodic infinite domain by a finite structure.

We have shown that, for formulae that include an explicit time constant, like e.g., $\Diamond_{=t}\phi$, the new metric encoding permits an improvement, in the size of the generated

SAT formula and in the SAT solving time, that is proportional to the numerical value of the time constant. In practical examples, the overall performance improvement is limited by other components of the encoding algorithm that are not related with the value of the time constants (namely, those that encode the structure of the time domain, or the non-metric operators). Therefore, the gain in performance can be reduced in the less favorable cases in which the analyzed formula contains few or no metric temporal operators, or the numerical value of the time constants is quite limited.

An extensive set of experiments has been carried out to asses its feasibility and effectiveness for Bounded Model Checking (and Bounded Satisfiability Checking). Average speed up in SAT solving time was 62%. The experimental results show that the new metric encoding can successfully be applied when the property to analyze includes time constants with a fairly high numerical value.

Acknowledgements: We thank Davide Casiraghi for his valuable work on Zot's metric plugins.

References

1. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. *Lecture Notes in Computer Science*, 1579:193–207, 1999.
2. A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5):1–64, 2006.
3. N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT Conference*, volume 2919 of *LNCS*, pages 502–518. Springer-Verlag, 2003.
4. C. Ghezzi, D. Mandrioli, and A. Morzenti. TRIO: A logic language for executable specifications of real-time systems. *Journal of Systems and Software*, 12(2):107–123, 1990.
5. C. Heitmeyer and D. Mandrioli. *Formal Methods for Real-Time Computing*. John Wiley & Sons, Inc., New York, NY, USA, 1996.
6. J. A. W. Kamp. *Tense Logic and the Theory of Linear Order (Ph.D. thesis)*. University of California at Los Angeles, 1968.
7. L. Lamport. A fast mutual exclusion algorithm. *ACM TOCS-Transactions On Computer Systems*, 5(1):1–11, 1987.
8. M. Lewis, T. Schubert, and B. Becker. Multithreaded SAT solving. In *12th Asia and South Pacific Design Automation Conference*, 2007.
9. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *DAC '01: Proceedings of the 38th Conf. on Design automation*, pages 530–535, New York, NY, USA, 2001. ACM Press.
10. M. Pradella, A. Morzenti, and P. San Pietro. The symmetry of the past and of the future: Bi-infinite time in the verification of temporal properties. In *Proc. of The 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering ESEC/FSE*, Dubrovnik, Croatia, September 2007.
11. M. Pradella, A. Morzenti, and P. San Pietro. Benchmarking model- and satisfiability-checking on bi-infinite time. In *ICTAC 2008*, volume 5160 of *Lecture Notes in Computer Science*, pages 290–304, Istanbul, Turkey, September 2008. Springer.
12. M. Pradella, A. Morzenti, and P. San Pietro. Refining real-time system specifications through bounded model- and satisfiability-checking. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008*, pages 119–127, 2008.
13. M. Pradella, A. Morzenti, and P. San Pietro. A metric encoding for bounded model checking (extended version). *ArXiv-CoRR*, 0907.3085, July 2009.