Bounded Satisfiability Checking of Metric Temporal Logic Specifications

MATTEO PRADELLA, ANGELO MORZENTI and PIERLUIGI SAN PIETRO
DEI, Politecnico di Milano

We introduce Bounded Satisfiability Checking, a verification technique that extends Bounded Model Checking by allowing also the analysis of a *descriptive model*, consisting of temporal logic formulae, instead of the more customary *operational model*, consisting of a state-transition system. We define techniques for encoding temporal logic formulae into Boolean logic that support the use of bi-infinite time domain and of metric time operators. In the framework of Bounded Satisfiability Checking, we show how a descriptive model can be refined into an operational one, and how the correctness of such a refinement can be verified for the bounded case, setting the stage for a stepwise system development method based on a bounded model refinement. Finally, we show how the adoption of a modular approach can make the bounded refinement process more manageable and efficient. All introduced concepts are extensively applied to a set of case studies, and thoroughly experimented through Zot, our SAT solver-based verification toolset.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*Languages*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; *Model Checking*; *Validation*; F.3.1 [**Logics and meanings of programs**]: Specifying and Verifying and Reasoning about Programs—*Mechanical verification*; *Specification Techniques*

General Terms: Languages, Verification

Additional Key Words and Phrases: Formal Methods, Temporal Logic, Bounded Model Checking, Refinement, Bi-infinite time

1. INTRODUCTION

Bounded model checking is a well established technique for modeling, analysis and verification of reactive, time-critical systems. It shares its motivations and many features with traditional model checking: the analyzed system is modeled as a state-transition structure; the property to be checked is expressed in temporal logic; the result of the analysis is either the confirmation of the conjectured property or its refutation, consisting of a counterexample, i.e., a possible behavior where the property does not hold. In bounded model checking, one first decides a bound, i.e., a natural number $k > 0$, and then both the state transition model and the conjectured property are encoded into a Boolean logic formula and analyzed by means of a SAT solver. The resulting Boolean formula is satisfiable if, and only if, the state transition system has a counterexample of length k to the conjectured property. Infinite, ultimately periodic behaviors of the analyzed systems are represented by means of additional logical variables that encode loops in the underlying time domain. If the Boolean formula is unsatisfiable, then there is no counterexample of length k to the property.

Our research group has a long lasting experience in the requirements specification and analysis of critical embedded, real-time systems. Based on a detailed description of the systems requirements in

This work is a revised and extended version of papers presented at ESEC/FSE 2007, ICTAC 2008, ASE 2008, and FM 2009.
This research has been partially funded by the European Community's IDEAS-ERC Programme, Project 227977 (SMSCom).
Authors' addresses: Matteo Pradella, DEI, Politecnico di Milano, Milano, Italy. E-mail: pradella@elet.polimi.it.
Angelo Morzenti, DEI, Politecnico di Milano, Milano, Italy. E-mail: morzenti@elet.polimi.it.
Pierluigi San Pietro, DEI, Politecnico di Milano, Milano, Italy. E-mail: sanpietro@elet.polimi.it.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

the TRIO temporal logic [Ghezzi et al. 1990], we defined methods and tools to carry out simulation in the form of history checking [Felder and Morzenti 1994], functional test case generation [Mandrioli et al. 1995], analysis of putative properties possibly ensured by the specified system under the assumption that the stated requirements are met [Gargantini and Morzenti 2001].

The present work generalizes bounded model checking encoding techniques to the case of bi-infinite time domains and metric extensions of LTL, allowing for for more compact and elegant system models consisting of temporal logic formulas rather than transition systems. This paves the way for a new approach to time-critical system analysis that we call *bounded satisfiability checking*, to emphasize the fact that the verification tasks are encoded into suitable instances of the satisfiability problem for quite large temporal logic formulae which encompass a model of the analyzed system and, possibly, some conjectured property that the designer intends to investigate.

Bounded satisfiability checking offers an unprecedented degree of flexibility in the analysis of the system under development. For instance, in its simplest form satisfiability checking can be used for a sort of testing or a *sanity check* of the requirements specification [Morzenti et al. 2003; Rozier and Vardi 2007]: if the requirements specification is inconsistent it obviously will not be satisfiable, so no possible behavior can be generated from it; if the requirements are formalized in an incorrect way the generated behaviors will exhibit unintended features. Bounded satisfiability checking also supports model checking with reference to a model that is *not* a state transition system, but consists of a temporal logic formula φ expressing the system assumed properties: a conjectured, (un)desired property can then be stated as a further temporal logic formula ψ and then the formula $\varphi \rightarrow \psi$ is checked.

This novel extension of model checking allows for concise, intuitive representation of the system's modeled features, because the formula φ that stands for the model is written in a temporal logic language that has the least possible restrictions concerning the adopted temporal operators and the underlying time domain, and facilitates the statement of *quantitative* time properties, which are most important for reactive time critical systems. To this end we defined original encoding techniques for various types of temporal logic languages, that support the translation into Boolean logic formulae of features such as infinite time domain in both future and past, past time operators, and metric time (i.e., use of numeric constants in formulae representing time distances among events and length of time intervals). The encodings we defined allow for the presence of any of the above features in a completely orthogonal way, so that the designer can choose the combination of constructs that she finds most appropriate for modeling and analyzing the system at hand.

In the above described approach to model checking the φ formula can be viewed both as a specification of the assumed system properties and also as a generalization of the notion of *model* adopted in traditional model checking. We therefore introduce the notion of a *descriptive model*, to indicate our view of a model, in which its salient features are provided by means of a descriptive formalism like temporal logic, as opposed to the abstract machine adopted in traditional model checking, which by contrast we call an *operational* model.

Of course, for a given (sub)system of interest, one can provide both a descriptive model and an operational one. In most cases the descriptive model would consist of a first, initial formalization of the system requirements, stated in an abstract style on the visible, external specification items, with no reference to internal, implementation-oriented features. The operational model, to be typically provided after the descriptive one, will be expressed in a state-transition based notation, and will possibly incorporate additional elements such as, for instance, counter variables or any memory necessary to store the current system *state*, and any mechanism to obtain the features defined in the descriptive model. The operational model can thus be viewed as a *refinement* of the descriptive model in a development process where the designer can go from requirements to implementation through a series of refinements steps that preserve the desired properties. In the present paper we will formalize in a temporal logic setting such a notion of property preserving refinement step among descriptive and operational models.

Embedded, reactive systems that are encountered in industrial applications exhibit a rich and complex behavior, often include several devices that cooperate to reach a common goal, and in-

teract with the environment through interfaces such as sensors and actuators. For all these reasons the design and even the specification of such complex systems exploit modularity [Morasca et al. 2000], which makes the development process more manageable by means of compositional techniques [de Roever 1997]. We show that in our temporal logic setting the descriptive and operational models can be smoothly combined, and composed with modules that include desired properties that the designer intends to analyze. We illustrate on a case study how the refinement process can be applied in a partial, incremental way, focusing on the modules that represent the components to be actually development and implemented, leaving the other components (those corresponding to the environment or to other existing subsystems) unchanged, thus obtaining substantial gains in the computational effort needed for the automatic check of correct refinement.

The paper is structured as follows. In Section 2 we provide general background material on bounded model- and satisfiability-checking, on temporal logic and its encoding, on the setup (hardware and software platform) and the test cases adopted for the experiments. Sections 3 and 4 illustrate the encoding techniques, and the relative experimental results, for bounded satisfiability checking of metric past linear temporal logic, concerning mono-infinite time structures (where time is unbound only towards the future) and bi-infinite ones. While these two sections are more technical in nature, dealing with logic and complexity features of the encodings, the subsequent sections are more methodological. Section 5 compares the performance of bounded model- and bounded satisfiability-checking, discussing the notion of operational vs. descriptive models and the correct refinement relation among them. Section 5.5 shows how the adoption of a modular approach can make much more manageable the process of model refinement, and much more efficient the verification of its correctness.

We exemplify and validate the encoding procedures and the system modeling and analysis methods by means of a very wide and rich set of case studies, taken from the literature on time critical system verification and specifically on (bounded) model checking.

We point out that some of these case studies include temporal logic formulae of quite significant size: this is due to the fact that those formulae not only represent the properties to be analyzed, but also constitute the actual model on which the analysis is conducted.

2. BACKGROUND AND EXPERIMENTAL SETUP

2.1. Bounded Satisfiability Checking vs. Bounded Model Checking

In Model Checking, a system S is characterized by a finite-state model M_S, where each state is associated with an assignment of a set of Boolean variables. The designer can prove a property ψ that is conjectured to hold for M_S and in general carry out various kinds of analysis (simulation in the form of trace generation, generation of functional test cases). We call *operational* a model such as M_S, provided in terms of a state-transition system, because it is more oriented to the definition of internal features and mechanisms that permit to ensure its properties, by means of a machine providing the set of transitions from a state to the next one.

Satisfiability Checking shares its goals with model checking but the model of the system S is itself a set of properties Φ_S, that are assumed to hold for the modeled system, in the same logic language as property ψ. To prove ψ for Φ_S it is enough to show that $\Phi_S \rightarrow \psi$ is valid, i.e., $\neg(\Phi_S \rightarrow \psi)$ is not satisfiable: the result is then valid for every system that satisfies Φ_S. We call a model such as Φ_S *descriptive* because it provides a description of the system behavior by means of (e.g., PLTL) formulae over the abstract interface items, with no reference to the internal mechanisms needed for implementing its assumed properties. Hence, a descriptive model provides an artifact on which to carry out the analysis of the system, at a higher level of abstraction than an operational model. This is an alternative to the view of model checking where the artifact used to analyze the system (i.e., its model) is a state-transition system. The size of the descriptive model is typically smaller than that of the operational one, as logic-based specifications allow for a more concise statement of the required properties.

The verification techniques for Model Checking and Satisfiability Checking are typically very different. For instance, in the automata-theoretic approach to model checking, (the negation of) formula ψ itself is translated into an automaton, and the verification is reduced to emptiness checking of the product automaton. Satisfiability Checking, on the other hand, may use for instance tableaux methods for trying to build a logic model of the formula $\neg(\Phi_S \rightarrow \psi)$. However, in *Bounded* Model Checking (BMC), after choosing a bound $k > 0$ for the cardinality of the time structure, state-transition systems and temporal logic properties are all translated into a Boolean logic formula, which is then checked for satisfiability by feeding it into a SAT solver. Since in the end everything is just a large Boolean formula, a toolkit may be able to treat homogeneously both operational and descriptive models, by translating all of them into Boolean logic. When the model is of the form Φ_S, we call this approach *Bounded Satisfiability Checking* (BSC).

Although the main thrust of this paper is on BSC, the above discussion should make clear that the results presented in this paper are also valid for BMC, by applying the various enhanced encoding techniques to the temporal logic property ψ only. For instance, all case studies used in this paper have been developed both in a descriptive and in an operational version, which will be extensively compared and discussed, also performance-wise, in Sections 5 and 5.5.

From the *methodological* point of view, we note that analyzing a property through bounded *satisfiability* checking provides the same support that can be obtained by means of bounded model checking: when a conjectured property is not implied by a descriptive model, a counterexample is obtained that gives the designer a useful insight into the features of the analyzed system that lead to that result.

In addition, bounded *satisfiability* checking can be usefully employed during the requirements analysis and elicitation, by just using the tool to check the satisfiability of the formulae that constitute the descriptive model, therefore obtaining a form of animation of the model that can exemplify and possibly clarify and make explicit its properties. Adopting a stepwise construction of the descriptive model of the system, the designer can start with the expression of its basic properties and subsequently, with the support of the generation of possible behaviors offered by the satisfiability checker, provide further formulae to model additional, specific features. In this process, additional subformulae that are combined with an existing descriptive model can be used, if they are composed by means of a conjunction, to provide additional constraints that act as a sort of filter to select or rule out certain desired or undesired behaviors, or otherwise to characterize certain classes of inputs that, in the intention of the designer who is carrying out the analysis, can lead the modeled system to certain behaviors that exhibit interesting system features. Subformulae that are composed with an existing, partial descriptive model by means of a disjunction can instead be used to enlarge the set of admissible system behaviors by considering, for instance, exceptions to the usual, ordinary functioning, or additional classes of acceptable input values.

Incompleteness of BMC and BSC. It is well known from the literature [Biere et al. 1999] that the results of analysis carried out by means of bounded model checking–and by means of BSC as well–are *partial*: if a counterexample to the conjectured property is found then the property does not hold, but in case no counterexample is found (i.e., if the formula submitted to the SAT solver is unsatisfiable) this only proves that the conjectured property holds for all finite or eventually periodic infinite system behaviors with length of the initial and periodic parts limited by the chosen bound. This does not rule out the possibility that there are other, longer behaviors that do not satisfy the property. Some recent approaches to bounded model checking (see e.g. [Kroening and Strichman 2003] and [Heljanko et al. 2005]) provide methods for finding a completeness bound, i.e., a value for the cardinality of the time structure which ensures that the results of the analysis are valid for every finite or eventually periodic infinite system behavior. However, the completeness bound can be very hard to compute for real systems.

Completeness issues are out of the scope of the present paper, therefore our technique shares the limitations that are typical of verification by means of *testing*: if a counterexample is found then the conjectured property is definitely disproved; if no counterexample is found it is still possible that

the property does not hold. In many practical cases a high degree of confidence on the results of the analysis can however be obtained if the size of the adopted time domain is significantly larger than the time constants of the analyzed system, so that the bounded satisfiability checker can in fact consider the vast majority, or even all significant combinations of events that can possibly take place.

2.2. PLTL: Linear Temporal Logic with past operators

We first recall here Linear Temporal Logic with past operators (PLTL), in the version introduced by Kamp [Kamp 1968].

Syntax of PLTL The alphabet of PLTL includes: a finite set Ap of propositional letters; two propositional connectives $\neg, \wedge$ (from which other traditional connectives such as $\mathrm{True}, \mathrm{False}, \vee, \rightarrow, \ldots$ 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 formula is a propositional letter $p \in Ap$ or $\neg\phi, \phi \wedge \psi, \phi\mathcal{U}\psi, \circ\phi, \phi\mathcal{S}\psi, \bullet\phi$, where ϕ, ψ are formulae; nothing else is a formula.

The traditional "eventually" and "globally" operators may be defined as follows: $\Diamond\phi$ is $\mathrm{True}\,\mathcal{U}\phi$, $\Box\phi$ is $\neg\Diamond\neg\phi$. Their past counterparts are: $\blacklozenge\phi$ is $\mathrm{True}\,\mathcal{S}\phi$, $\blacksquare\phi$ is $\neg\blacklozenge\neg\phi$. Another useful operator 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 $\neg\mathcal{A}lw\neg\phi$. For the sake of brevity, we 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: $\exists p\ gr(p)$, with p ranging over $\{1, 2, 3\}$, as a shorthand for $\bigvee_{p\in\{1,2,3\}} gr_p$. Also as a shorthand, $\circ^t\phi$ (and analogously $\bullet^t\phi$), where $t > 0$ is a constant, stands for t nested application of $\circ$ to ϕ: $\circ(\circ \ldots (\circ\phi) \ldots)$.

Standard semantics of PLTL The semantics of PLTL is defined here on ω-words, i.e., assuming time is finite in the past and infinite in the future. A different semantics is shown in Section 4, where time is considered to be infinite both in the past and in the future (i.e., on $\mathbb{Z}$-words). This bi-infinite semantics is actually simpler and includes the mono-infinite one as a special case, but it is not yet universally adopted as the standard one.

In particular, since the temporal structure is mono-infinite, formulae may refer to time instants before 0 (e.g., by using $\bullet$ at instant 0), where the evaluation is not defined. The typical approach (see e.g. [Biere et al. 2006]) is to use a default value for operators referring to instants outside the temporal domain. Hence, $\bullet\phi$ is false at 0 for any ϕ. Given a finite alphabet Σ, Σ^* denotes the set of finite words over Σ. An ω-word over Σ is an infinite sequence $w = a_0a_1a_2 \ldots$, with $a_j \in \Sigma$ for every $j \geq 0$. The set of all ω-words over Σ is denoted as Σ^ω, while an element a_j of $w = a_0a_1a_2 \ldots$ is denoted as $w(j)$.

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

$$
\begin{array}{ll}
w, i \models p & \longleftrightarrow\ p \in w(i), \text{for } p \in Ap \\
w, i \models \neg\phi & \longleftrightarrow\ w, i \not\models \phi \\
w, i \models \phi \wedge \psi & \longleftrightarrow\ w, i \models \phi \text{ and } w, i \models \psi \\
w, i \models \circ\phi & \longleftrightarrow\ w, i+1 \models \phi \\
w, i \models \phi\mathcal{U}\psi & \longleftrightarrow\ \exists k \geq 0 : w, i+k \models \psi, \text{ and } \forall 0 \leq j < k : w, i+j \models \phi \\
w, i \models \bullet\phi & \longleftrightarrow\ i > 0 \text{ and } w, i-1 \models \phi \\
w, i \models \phi\mathcal{S}\psi & \longleftrightarrow\ \exists k \geq 0 : i-k \geq 0,\ w, i-k \models \psi \text{ and } \forall 0 \leq j < k : w, i-j \models \phi.
\end{array}
$$

As customary, PLTL formulae are evaluated at time 0, i.e., a PLTL formula ϕ is *satisfied* on an ω-word w iff $w, 0 \models \phi$. A formula is *valid* if it is satisfied on all ω-words.

Figure 1 presents, for the sake of illustration, the initial part of w, where w is an ω-word with prefix:

$$\emptyset\{\phi\}\{\phi, \psi\}\{\phi\}\emptyset\{\phi\}\{\phi\}\{\phi\}\{\psi\}\emptyset\emptyset\emptyset.$$

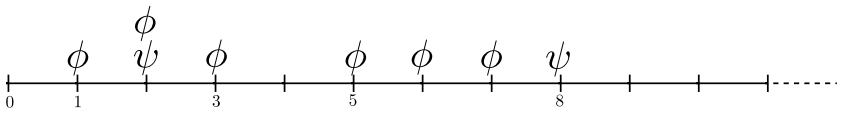

Fig. 1. An example ω-word.

For instance, the following holds: $w, i \models \phi \mathcal{U} \psi$, for $i \in \{1, 2, 5, 6, 7, 8\}$; $w, i \models \phi \mathcal{S} \psi$, for $i \in \{2, 3, 8\}$; $w, i \models \circ\phi$, for $i \in \{0, 1, 2, 4, 5, 6\}$; and $w, i \models \bullet\phi$, for $i \in \{2, 3, 4, 6, 7, 8\}$

Positive Normal Form and Dual operators We introduce here a normal form, where negations may only occur on atoms, which is very convenient when defining encodings of PLTL into propositional logic. Define the dual operator for each operator in the syntax: the dual of $\wedge$ is just $\vee$, the dual of $\circ$ is $\circ$ itself, the dual of Until is "Release" $\mathcal{R}$: $\phi \mathcal{R} \psi$ is $\neg(\neg\phi \mathcal{U} \neg\psi)$; the dual of Since is "Trigger" $\mathcal{T}$: $\phi \mathcal{T} \psi$ is $\neg(\neg\phi \mathcal{S} \neg\psi)$; the dual of $\bullet$ is $\bullet'$: $\bullet'\phi$ is $\neg \bullet \neg\phi$. A formula is in *positive normal form* if its alphabet is $\{\wedge, \vee, \mathcal{U}, \mathcal{R}, \circ, \mathcal{S}, \mathcal{T}, \bullet, \bullet'\} \cup Ap \cup \overline{Ap}$, where $\overline{Ap}$ is the set of formulae of the form $\neg p$ for $p \in Ap$. Every PLTL formula ϕ on the alphabet $\{\neg, \wedge, \mathcal{U}, \circ, \mathcal{S}, \bullet\} \cup Ap$ may be transformed into an equivalent formula ϕ' in positive normal form.

By definition, it follows that the truth value of $\bullet'\phi$ on ω-words is the same of $\bullet\phi$ in every instant except for instant 0 (where the former is true and the latter is false).

2.3. Metric temporal logic

Metric operators were introduced in Linear Temporal Logic by [Koymans 1990] as a convenient, general way to model hard real time systems, with quantitative time constraints. The specialized version of the metric operators introduced in this section does not actually extend the expressive power of PLTL, but it leads to more succinct formulae.

Metric PLTL (MPLTL for short) extends the alphabet of PLTL with a *bounded until* operator $\mathcal{U}_{\sim c}$ and a *bounded since* operator $\mathcal{S}_{\sim c}$, where $\sim$ represents any relational operator (i.e., $\sim \in \{\leq, =, \geq\}$), and c is a natural number. In the following, as a useful shorthand, we will use also the versions of the bounded operators with a strict bound. For instance, $\phi \mathcal{U}_{>0} \psi$ stands for $\phi \wedge \circ(\phi \mathcal{U}_{\geq 0} \psi)$; the other strict bounded operators are defined similarly.

In order to define *positive normal form* for Metric PLTL, we have also to define the dual operators of $\mathcal{U}_{\sim c}, \mathcal{S}_{\sim c}$. The dual of $\mathcal{U}_{\sim c}$ is the bounded Release $\mathcal{R}_{\sim c}$: $\phi \mathcal{R}_{\sim c} \psi$ is $\neg(\neg\phi \mathcal{U}_{\sim t} \neg\psi)$. The dual of $\mathcal{S}_{\sim c}$ is the bounded Trigger $\mathcal{T}_{\sim c}$: $\phi \mathcal{T}_{\sim c} \psi$ is $\neg(\neg\phi \mathcal{S}_{\sim t} \neg\psi)$.

Semantics The semantics of Metric PLTL on ω-words is defined by the following additional clauses:

$$
\begin{aligned}
w, i \models \phi \mathcal{U}_{\sim t} \psi \iff & \exists k \geq 0 : k \sim t,\ w, i + k \models \psi, \text{ and } \forall 0 \leq j < k : w, i + j \models \phi \\
w, i \models \phi \mathcal{S}_{\sim t} \psi \iff & \exists k \geq 0 : k \sim t,\ i - k \geq 0,\ w, i - k \models \psi, \text{ and} \\
& \forall 0 \leq j < k :\ w, i - j \models \phi.
\end{aligned}
$$

Bounded eventually and globally and their past versions The bounded eventually $\Diamond_{\sim c}\phi$ and bounded past $\blacklozenge_{\sim c}\phi$ are defined as $\mathrm{True}\,\mathcal{U}_{\sim c}\phi$ and $\mathrm{True}\,\mathcal{S}_{\sim c}\phi$, respectively. The bounded globally operator is the dual of $\Diamond_{\sim c}$: $\Box_{\sim c}\phi$ is $\neg\Diamond_{\sim c}\neg\phi$. The bounded globally in-the-past operator $\blacksquare_{\sim c}\phi$ is $\neg\blacklozenge_{\sim c}\neg\phi$.

We note that, in practice, the bounded until and since operators are not frequently used, the most common metric operators being the bounded eventually $\Diamond_{\sim t}\phi$, and the bounded globally $\Box_{\sim t}\phi$ and their past counterparts. Therefore, in the following, encodings for metric temporal operators will be provided focusing on $\Diamond_{\sim c}\phi$, $\Box_{\sim c}\phi$, $\blacklozenge_{\sim c}\phi$, and $\blacksquare_{\sim c}\phi$.

Example 2.1. *A synchronous shift register* Consider a simple shift register, that receives a single bit at one end and delivers it at the opposite end with a fixed delay $d > 0$ (d is a constant representing the number of memory bits in the register). A specification of this system can be described by the

Table I.

Formula	Equivalent formula
$\blacklozenge_{\geq t}\phi$	$\blacklozenge_{=t}\blacklozenge\phi$
$\blacksquare_{\geq t}\phi$	$\blacksquare_{=t}\blacksquare\phi$
$\Box_{=t}\phi$	$\Diamond_{=t}\phi$
$\Diamond_{\geq t}\phi$	$\Diamond_{=t}\Diamond\phi$
$\Box_{>t}\phi$	$\Box_{=t}\Box\phi$
$\blacksquare'_{<t}\phi$	$\blacklozenge_{=t}\,\mathrm{True}\wedge\blacksquare_{\leq t}\phi$
$\blacklozenge'_{<t}\phi$	$\blacksquare_{=t}\,\mathrm{False}\vee\blacklozenge_{\leq t}\phi$
$\blacklozenge'_{\geq t}\phi$	$\blacksquare_{=t}\blacklozenge\phi$
$\blacksquare'_{\geq t}\phi$	$\blacklozenge_{=t}\blacksquare\phi$

Definition of metric operators in terms of $\Diamond_{=t}$, $\Diamond_{\leq t}$, $\Box_{\leq t}$, $\blacklozenge_{=t}$, $\blacksquare_{=t}$, $\blacklozenge_{\leq t}$, $\blacksquare_{\leq t}$

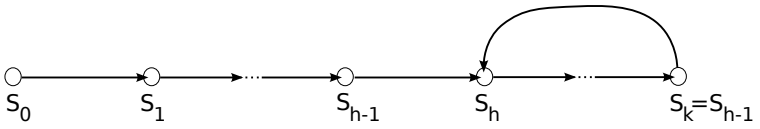

Fig. 2. An infinite bounded path.

following simplest formula:

$$\Box(in \longleftrightarrow \Diamond_{=d} out)$$

where in is the value of the bit entering the shift register (True stands for 1, False for 0), and out is the value of the bit exiting the shift register after delay d.

Relations among the various bounded operators. First, one may notice that $\Box_{=t}\phi$ is equivalent to $\Diamond_{=t}\phi$, while on a monoinfinite semantics $\blacksquare_{=t}\phi$ is *not* equivalent to $\blacklozenge_{=t}\phi$: when there is no instant at distance t in the past, $\blacksquare_{=t}\phi$ is true and $\blacklozenge_{=t}\phi$ is false. In fact, $\blacksquare$ assumes that everything outside the temporal domain is true by default, while in the same case $\blacklozenge$ assumes that it is false. For instance, $\blacksquare_{=1}\phi \longleftrightarrow \bullet'\phi$, while $\blacklozenge_{=1}\phi \longleftrightarrow \bullet\phi$.

In general, when dealing with a monoinfinite semantics, the usage of operators $\blacklozenge_{\leq t}\phi$ and $\blacksquare_{\leq t}\phi$ may not be very convenient to specify all relevant behaviors. In fact, if instant t in the past does not exist and ϕ is true from the current instant to instant 0, then $\blacksquare_{\leq t}\phi$ is true, even though the interval of validity of ϕ is shorter than t: instead, the intuitive meaning of $\blacksquare_{\leq t}\phi$ is that ϕ must hold for at least t instants in the past.

This motivates the introduction of a new $\blacksquare'_{\leq t}$ operator, that is false by default when referring to instants outside the temporal domain, and of its dual $\blacklozenge'_{\leq t}$ that in the same case is true by default. Hence, $\blacksquare'_{\leq t}\phi$ is false whenever the interval of validity of ϕ is shorter than t. Define $\blacksquare'_{\leq t}\phi$ as $\blacklozenge_{=t}\,\mathrm{True}\wedge\blacksquare_{\leq t}\phi$, with its dual $\blacklozenge'_{\leq t}\phi$ being equivalent to $\blacksquare_{=t}\,\mathrm{False}\vee\blacklozenge_{\leq t}\phi$.

Analogously, define $\blacklozenge'_{\geq t}\phi$ as $\blacksquare_{=t}\blacklozenge\phi$, with its dual $\blacksquare'_{\geq t}\phi$ being equivalent to $\blacklozenge_{=t}\blacksquare\phi$. More on these operators may be found in Section 4.

All various versions of bounded globally and eventually operators can actually be derived from

$$\Diamond_{=t}, \Diamond_{\leq t}, \Box_{\leq t}, \blacklozenge_{=t}, \blacksquare_{=t}, \blacklozenge_{\leq t}, \blacksquare_{\leq t}.$$

Table I summarizes the relations among various operators, for every formula ϕ, when a monoinfinite semantics is considered.

2.4. A Boolean Encoding for PLTL

Here we describe a standard encoding of PLTL formulae and state-transition systems future fragment of PLTL, into Boolean logic over a finite temporal structure. The encoding includes additional

information 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) mono-infinite structure. This encoding is a modified version of the one presented in [Biere et al. 2006], Section 5.1: BMC for PLTL with Eventualities.

For the goal of this section, define a state transition system M_s as tuple $(\mathcal{S}, I, T, Ap, \lambda)$, where $\mathcal{S}$ is a finite set of states, $I \subseteq \mathcal{S}$ is the set of initial states (i.e., a boolean predicate), $T \subseteq \mathcal{S} \times \mathcal{S}$ is the transition relation, Ap is a finite alphabet of propositions, and $\lambda : \mathcal{S} \rightarrow 2^{Ap}$ is a labeling function, associating each state with the set of propositions holding in that state. A computation of M_S is an infinite sequence of states $S_0 S_1 \ldots$ such that $I(S_0)$ holds and for every $i \geq 0$, $S_i \in \mathcal{S}$ and $T(S_i, S_{i+1})$ holds. A PLTL formula ϕ on alphabet Ap is *valid on* M_S if for every computation $S_0 S_1 \ldots$ of M_S the ω-word $w = \lambda(S_0)\lambda(S_1)\ldots$ satisfies it, i.e., $w, 0 \models \phi$ holds. Model Checking is the activity of verifying whether a given formula (i.e., a property) is valid on a given state transition system (i.e., a model).

Following the conventions of [Biere et al. 2006], the notation $|[X]|$ denotes the Boolean variables introduced in the encoding to represent some entity X. We also use the notation $\langle\langle X \rangle\rangle$ (or variants thereof) to represent auxiliary Boolean variables, that are related to X and explained later.

To perform bounded model checking with bound k, we represent symbolically the transition relation of the system M_S as a propositional formula, where the states S_i are represented as bit vectors. The k-times unrolling of the transition relation represents all the finite paths of length k:

$$|[M_S]|_k \longleftrightarrow I(S_0) \wedge \bigwedge_{0 \leq i < k} T(S_i, S_{i+1})$$

The idea on which the encoding is based is graphically depicted in Figure 2. Following the notation presented in the picture, in the rest we will use h to denote the instant i in which the loop starts. An ultimately periodic mono-infinite structure has a finite representation that includes the initial non periodic portion, and the periodic portion with a cycle that is encoded by having two equal states in the sequence, S_{h-1} and S_k: the interpreter of the formula (in our case, the SAT solver), when it needs to evaluate the subformula at a state beyond the last state S_k, will follow the "backward link" and consider the states S_h, S_{h+1}, ... as the states following S_k.

Let Φ be a LTL formula in positive normal form. 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 $k+1$ is not explicitly shown in Figure 2, but has a particular role in the encoding, as shown next in the *Last state constraints*, namely if there is a loop at position h then state S_{k+1} and S_h are equivalent.

First, to allow for the representation of a mono-infinite structure into a finite one composed of $k+1$ states $S_0, S_1, \ldots S_k$, other $k+1$ fresh propositional variables $l_0, l_1, \ldots l_k$ must be introduced, called *loop selector variables*, which describe the loop that may exist in the finite structure. At most one of these loop selector variables may be true. If l_h is true then state $S_{h-1} = S_k$, i.e., the bit vector representing the state S_{h-1} is identical to that for state S_k. Further propositional variables, InLoop_i $(0 \leq i \leq k)$ and LoopExists, respectively mean that position i is inside a loop and that a loop does actually exist in the structure.

The variables defining the loops in the finite structure are constrained by the following Table (1).

Loop constraints:

Base	$\neg l_0 \wedge \neg \mathrm{InLoop}_0$
$1 \leq i \leq k$	$(l_i \rightarrow S_{i-1} = S_k) \wedge (\mathrm{InLoop}_i \longleftrightarrow \mathrm{InLoop}_{i-1} \vee l_i)$ $(\mathrm{InLoop}_{i-1} \rightarrow \neg l_i) \wedge (\mathrm{LoopExists} \longleftrightarrow \mathrm{InLoop}_k)$

(1)

The Loop constraints of Table (1) (with one formula for each value of i, $1 \leq i \leq k$) state that the structure may have at most one loop. When a loop exists they allow the SAT solver to nondeterministically select exactly one of the many possible values for the loop selector variable.

The following Table (2) provides the additional constraints, for every subformula ϕ of Φ, needed 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 then everything is false beyond the k-th state) or its presence (the second line states that if there is a loop at position i then in states S_{k+1} and S_i formula variables must have the same truth value).

Last state constraints:

Base	$\neg\text{LoopExists} \rightarrow \neg\lvert[\phi]\rvert_{k+1}$
$1 \leq i \leq k$	$l_i \rightarrow (\lvert[\phi]\rvert_{k+1} \longleftrightarrow \lvert[\phi]\rvert_i)$

(2)

Table (3) constrains in a natural way the formula variables. For instance, if $\phi_i \wedge \phi_2$ is a subformula of Φ, then each variable $\lvert[\phi_i \wedge \phi_2]\rvert_i$ must be equivalent to the conjunction of variables $\lvert[\phi_1]\rvert_i$ and $\lvert[\phi_2]\rvert_i$.

The following tables are composed of two columns: the left column, labeled φ, denotes the various cases of subformulae considered by the table; the right column shows the encoding of each case, and it is labeled with the set of all integer values i the encoding is applied to.

Propositional constraints, with p denoting a propositional symbol:

φ	$0 \leq i \leq k$
p	$\lvert[p]\rvert_i \longleftrightarrow p \in S_i$
$\neg p$	$\lvert[\neg p]\rvert_i \longleftrightarrow p \notin S_i$
$\phi_1 \wedge \phi_2$	$\lvert[\phi_1 \wedge \phi_2]\rvert_i \longleftrightarrow \lvert[\phi_1]\rvert_i \wedge \lvert[\phi_2]\rvert_i$
$\phi_1 \vee \phi_2$	$\lvert[\phi_1 \vee \phi_2]\rvert_i \longleftrightarrow \lvert[\phi_1]\rvert_i \vee \lvert[\phi_2]\rvert_i$

(3)

As a simple example, consider the formula $A \wedge B \vee \neg C$. For each $0 \leq i \leq k$, we introduce the propositional formulae:

$\lvert[A \wedge B \vee \neg C]\rvert_i \longleftrightarrow \lvert[A \wedge B]\rvert_i \vee \lvert[\neg C]\rvert_i$, and $\lvert[A \wedge B]\rvert_i \longleftrightarrow \lvert[A]\rvert_i \wedge \lvert[B]\rvert_i$.

The following Table (4) defines the basic temporal behavior of PLTL future operators $\circ, \mathcal{U}$ and $\mathcal{R}$, by using their traditional fixpoint characterizations.

Future temporal subformulae constraints:

φ	$0 \leq i \leq k$
$\circ\phi_1$	$\lvert[\circ\phi_1]\rvert_i \longleftrightarrow \lvert[\phi_1]\rvert_{i+1}$
$\phi_1 \mathcal{U} \phi_2$	$\lvert[\phi_1 \mathcal{U} \phi_2]\rvert_i \longleftrightarrow \lvert[\phi_2]\rvert_i \vee (\lvert[\phi_1]\rvert_i \wedge \lvert[\phi_1 \mathcal{U} \phi_2]\rvert_{i+1})$
$\phi_1 \mathcal{R} \phi_2$	$\lvert[\phi_1 \mathcal{R} \phi_2]\rvert_i \longleftrightarrow \lvert[\phi_2]\rvert_i \wedge (\lvert[\phi_1]\rvert_i \vee \lvert[\phi_1 \mathcal{R} \phi_2]\rvert_{i+1})$

(4)

These constraints do not consider the implicit eventualities that the definitions of $\mathcal{U}$ and $\mathcal{R}$ impose (e.g. $\phi_1 \mathcal{U} \phi_2$ requires that ϕ_2 must *eventually* hold).

To define properly eventualities, it is necessary to introduce a new propositional letter $\langle\langle \Diamond \phi_2 \rangle\rangle_i$, for each subformula of Φ of the form $\phi_1 \mathcal{U} \phi_2$, for every i such that $0 \leq i \leq k$. Analogously, add a new propositional letter $\langle\langle \Box \phi_2 \rangle\rangle_i$ for each subformula of the form $\phi_1 \mathcal{R} \phi_2$, $0 \leq i \leq k$. Then, constraints on these eventuality propositions are quite naturally stated as follows.

Eventuality constraints:

φ	Base
$\phi_1 \mathcal{U} \phi_2$	$\neg\langle\langle \Diamond \phi_2 \rangle\rangle_0 \wedge (\text{LoopExists} \rightarrow (\ \lvert[\phi_1 \mathcal{U} \phi_2]\rvert_k \rightarrow \langle\langle \Diamond \phi_2 \rangle\rangle_k\ ))$
$\phi_1 \mathcal{R} \phi_2$	$\langle\langle \Box \phi_2 \rangle\rangle_0 \wedge (\text{LoopExists} \rightarrow (\ \lvert[\phi_1 \mathcal{R} \phi_2]\rvert_k \leftarrow \langle\langle \Box \phi_2 \rangle\rangle_k\ ))$

(5)

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

(6)

The encoding of past operators is not completely symmetrical to the encoding for future operators, because their definition is not symmetrical with mono-infinite time. Table (7) below is the analogous of Table (4), but it also has operator $\bullet'$. Notice that this part of the encoding is perfectly symmetrical to its future counterpart.

Past Temporal subformulae constraints:

$$\begin{array}{c|c} \varphi & 1 \leq i \leq k+1 \\ \hline \bullet\phi_1 & |[\bullet\phi_1]|_i \longleftrightarrow |[\phi_1]|_{i-1} \\ \bullet'\phi_1 & |[\bullet'\phi_1]|_i \longleftrightarrow |[\phi_1]|_{i-1} \\ \phi_1\mathcal{S}\phi_2 & |[\phi_1\mathcal{S}\phi_2]|_i \longleftrightarrow |[\phi_2]|_i \vee (|[\phi_1]|_i \wedge |[\phi_1\mathcal{S}\phi_2]|_{i-1}) \\ \phi_1\mathcal{T}\phi_2 & |[\phi_1\mathcal{T}\phi_2]|_i \longleftrightarrow |[\phi_2]|_i \wedge (|[\phi_1]|_i \vee |[\phi_1\mathcal{T}\phi_2]|_{i-1}) \end{array} \tag{7}$$

Indeed, the main difference with the treatment of future is in the following constraints. Being the temporal structure mono-infinite, there exists a *first* state, i.e. time 0. Constraints in Table (8) are used to state that, for $\phi_1\mathcal{S}\phi_2$ to hold at 0, ϕ_2 must also hold there, and vice versa, as there are not time instants before 0. $\mathcal{T}$ is treated analogously, while $\bullet$ and $\bullet'$ at 0 refer to a point outside the temporal domain, so they assume their conventional values (false and true, respectively). Notice also that eventuality constraints are not needed for past operators, since past is finite.
First state constraints:

$$\begin{array}{c|c} \varphi & \text{Base} \\ \hline \phi_1\mathcal{S}\phi_2 & |[\phi_1\mathcal{S}\phi_2]|_0 \longleftrightarrow |[\phi_2]|_0 \\ \phi_1\mathcal{T}\phi_2 & |[\phi_1\mathcal{T}\phi_2]|_0 \longleftrightarrow |[\phi_2]|_0 \\ \bullet\phi_1 & \neg|[\bullet\phi_1]|_0 \\ \bullet'\phi_1 & |[\bullet'\phi_1]|_0 \end{array} \tag{8}$$

The complete encoding of Φ consists of the logical conjunction of the above components regarding loops, propositional connectives, temporal operators, and eventualities, together with $|[\Phi]|_0$ (i.e. Φ is evaluated only at instant 0).

The encoding of [Biere et al. 2006], unlike the one presented here, adopts a virtual unrolling technique, where extra variables are introduced for the past-time operators, in order to obtain minimal-length counterexamples. Our approach does not strive instead for minimal-length: rather, we adopt a cardinality of the temporal domain which is typically much larger than the largest time constant d appearing in the time operators, such as $\Box_{<d}$ or $\Diamond_{<d}$, occurring in the formulae that constitute our models. Naturally, the above unrolling technique could be adapted to the encodings presented here, to obtain minimal-length counterexamples when they are really needed (e.g., when using k-induction or trying to understand complex execution traces).

2.5. Case studies

Besides the *Synchronous Shift Register* of Example 2.1 we conduct our experiments with reference to the following case studies, and report them throughout the paper.

2.5.1. Fischer's protocol. Fischer's algorithm [Lamport 1987] is a timed mutual exclusion algorithm that allows a number of timed processes to access a shared resource. These processes are usually described as timed automata, and are often used as a benchmark for timed automata verification tools.

We consider the system in two variants. The first one, called *fischer-3-5*, considers 3 processes with a delay after the request of 5 time units. The second one, called *fischer-4-10*, considers 4 processes with a delay after the request of 10 time units.

We used the tool to check the safety property of the system i.e., it is never possible that two different processes enter their critical sections at the same time instant.

As a last test for this system, we added a constraint to generate a behavior in which there is always at least an alive process in the system.

2.5.2. Kernel Railway Crossing (KRC). The Kernel Railway Crossing problem has extensively been used as a benchmark for comparing real-time notations and analysis methods and tools [Heitmeyer and Mandrioli 1996]. We adopt here a simplified version which considers only one track and one direction of movement for the trains, but at the same time we enrich the case study by adding an interlocking system, which is usually disregarded.

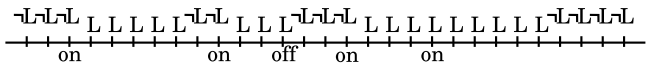

Fig. 3. A history for the example of the timed lamp, assuming $\Delta = 5$.

The detailed description of the case study, together with its descriptive and operational models, and the analyzed properties, are reported in Appendix I.1. Two sets of time constants are considered, allowing for different degrees of nondeterminism.

2.5.3. Real-time allocator. The real-time allocator, originally presented in [Felder and Morzenti 1994], serves a set of client processes competing for a shared resource.

Each process p requires the resource by issuing the message $rq(p)$, by which it identifies itself to the allocator. Requests have a time out: they must be served within T_{req} time units, or else be ignored by the allocator. If the allocator is able to satisfy p's request within the time-out, then it grants the resource to p by a $gr(p)$ signal. Once a process is assigned the resource by the allocator, it releases the resource, by issuing a rel signal, within a maximum of T_{rel} time units. The allocator grants the request to processes according to a FIFO policy, considering only requests that are not yet timed out and in a timely manner, i.e., no process will have to wait for the resource while it is not assigned to any other process. The analysis of the real time allocator can provide significant results only with reference to bi-infinite time domains requiring the bi-infinite semantics of PLTL presented in Section 4, because the assumption of *unconstrained rotation*, under which the property of *conditional fairness* can be correctly checked, implies a series of requests by the various processes which repeats itself indefinitely towards the past (a detailed exemplification of this circumstance is reported in Example 4.3 of Section 4). Hence, verification results on mono-infinite time are incorrect and are not reported. The temporal logic formulae describing the allocation policy and its properties are presented in Appendix I.2.

2.5.4. Timer Reset Lamp. The timer-reset-lamp case study was first introduced in paper [Pradella et al. 2008b]. Here we present the descriptive model, consisting of axioms specifying its key timing features; the operational model is reported in Section 5.1.

The lamp has two buttons, *ON* and *OFF*: when the *ON* button is pressed the lamp is lighted and it remains so, if no other event occurs, for Δ time units, after which it goes off spontaneously. It is also possible, before the above expiration deadline, to either turn off the lamp, by pushing the *OFF* button, or to extend the lighting of further Δ time units, by pushing again the *ON* button. This behavior is expressed by the following axiom (where predicate letter *L* means that the light is on, while letters *ON* and *OFF* respectively indicate that the button to turn the lamp on or off is pressed).

$$(\mathrm{D1})\ \ L \longleftrightarrow \bullet(\neg \mathit{OFF}\ \mathcal{S}_{<\Delta} \mathit{ON})$$

To ensure that the pressure of a button is always meaningful, it is assumed that *ON* and *OFF* cannot be pressed simultaneously.

$$(D2)\ \ \neg(\mathit{ON} \wedge \mathit{OFF})$$

The descriptive model of the timer-reset-lamp consists of the conjunction of the two above formulae, to which the temporal universal quantification operator $\mathcal{A}lw$ is applied.

$$(DM)\ \ \mathcal{A}lw(D1 \wedge D2)$$

Starting from the above characterization one can disprove the following (conjectured) property

$$(DP1)\ \ \mathcal{A}lw(\neg\Box_{\leq\Delta+1} L)$$

(i.e., the lamp will never remain on for more than Δ time units), by generating a counter-example similar to the one shown in Figure 3, including two push actions of the *ON* button at distance less

than Δ; the following property holds and can be checked starting from the descriptive model:

$$(DP2) \quad \mathcal{S}om(\Box_{\leq \Delta+1} L) \rightarrow \mathcal{S}om(ON \wedge \Diamond_{\leq \Delta} ON)$$

(i.e., the lamp remains lighted for more than Δ time units only in case of two consecutive press actions of the *ON* button at a distance of less than Δ time units).

Various versions of the example have been studied, for different values of constant Δ (10, 15, 20, corresponding to *lamp-10, lamp-15, lamp-20*).

2.5.5. Asynchronous Shift Register. This is a variation of the Synchronous Shift Register discussed in Example 2.1, 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, where the number of bits is $n = 10$ and $n = 20$; we prove satisfiability of the specification, and analyze one timed property (if the *Shift* signal remains true for *n* time units 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). The descriptive and operational models for the Asynchronous Shift Register are reported in Appendix I.3.

2.6. Experimental setup

In the last years we designed and implemented an open, plug-in based tool called *Zot*.[1] Zot was not designed to be a new, more efficient model checker, but to define a simple and flexible open environment, to experiment with new approaches and encodings for bounded model checking. Zot is written in Common Lisp and it is scriptable, thus favoring experimentation, as plugins are typically simple, compact (usually around 500 lines of code), easily modifiable, and extensible. Zot supports various SAT solvers, like MiniSat [Eén and Sörensson 2003], and MiraXT [Lewis et al. 2007], and in general can be extended to any SAT solver that adopts the standard format DIMACS for its interface data. 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 [Ghezzi et al. 1990], essentially equivalent to Metric PLTL. Other notations are also available in the toolset, namely dense-time metric temporal logic through approximation, and variants of timed automata and Petri nets [Furia et al. 2008a; 2008b; Bersani et al. 2009]. All the results and encodings presented in this paper, together with their implementations as Zot plugins, are freely available, and can be ported to other tools.

Zot provides a simple language to define both descriptive and operational models, and to mix them freely; this is possible because both types of models are eventually translated into Boolean logic, to be fed to the SAT solver. When Zot finds a counterexample, i.e., a possible trace of the submitted model, the trace reported to the user is expressed in a readable form, through the value of the variables and predicates of the submitted formula, rather than in terms of the internal encoding adopted by the tool.

2.7. Assumptions in reporting the experiments

The experiments were run on a PC equipped with two XEON 5335 processors at 2.0 Ghz, with 16 GB RAM, running under GNU/Linux Gentoo X86-64. The SAT-solver was MiniSat 2.

For every experiment we report total Generation time (Gen) in seconds, total SAT Solver time (Solver) in seconds, and number of CNF clauses submitted by Zot to the SAT solver. The memory footprint of the SAT solver is not reported, for the sake of simplicity: memory is strongly correlated to both CNF clauses and Solver time, and it is never critical (ranging from a few MBs for the smallest examples to less than one Gbyte for the biggest example presented in this paper). Overall, CNF clauses is a better measure of the quality of the encoding techniques. All numerical results reported in this paper are actually averaged over analyses carried out with three bounds: 30, 60 and 90, to reduce the dependence of the behavior of the Solver with the bound itself (in certain cases, a

[1] Available at http://home.dei.polimi.it/pradella/Zot/.

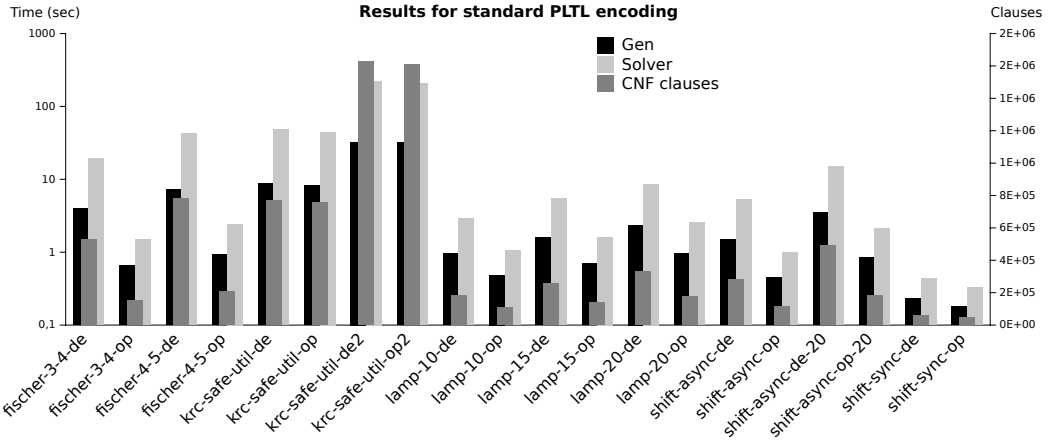

Fig. 4. Summary of raw experimental data for the standard PLTL mono-infinite encoding described in Section 2.4. For each case study (defined on the horizontal axis), the figure reports the time in seconds for the Generation (Gen) and Solver phases (vertical axis at the left, on a logarithmic scale), and the number of CNF clauses fed to the SAT-solver (vertical axis at the right).

particular bound can by chance help in simplifying the verification). The results are also averaged over various properties. Details and names are as follows:

— *alloc* denotes the verification of the allocator, averaged over the verification of all considered properties.
— *krc* and *krc-2* denote the verification of the railway crossing problem, in the versions with, respectively, the smaller and the larger time constants; *krc* allows however more nondeterminism. Various versions and properties of krc will be defined in Section 5. The notation *krc-safe-util* is used for denoting the results averaged over verification of satisfiability, safety and utility properties.
— *fischer-3-5* and *fischer-4-10* are the Fischer protocol, averaged over satisfiability of the specification and the safety property, using the version with 3 processes and delay 5 and the version with 4 processes and delay 10.
— *lamp-10, lamp-15, lamp-20* denote the satisfiability verification of versions of the timer-reset-lamp with a delay of, respectively, 10, 15 and 20 units.
— *shift-sync* denotes the result of satisfiability verification for the synchronous shift register;
— *shift-async* and *shift-async-20* denote the average result of satisfiability and timing verifications for the asynchronous shift register, with delays 10 and 20, respectively.

The suffix -de is added to denote that a model is descriptive (therefore, BSC is used) and the suffix -op to denote that the model is operational with properties described by temporal logic formulae (therefore, BMC is used).

2.8. Preliminary experiments

We report in Figure 4 experimental results with the encoding of Section 2.4, where the metric temporal operators are encoded by the naive technique illustrated in the initial part of Section 3.1. Such experimental results provide some baseline for comparison when evaluating results in later experiments.

The experiments report the results for both BSC and BMC in the mono-infinite case. The allocator case study is not included since it requires the bi-infinite semantics of Section 4.1. Also, the synchronous shift register is not reported, since it will be extensively discussed in Section 3.

The experimental results show the feasibility of BSC, even though BSC is often (but not always) less efficient than BMC, which is not surprising since in general time complexity of BSC may be exponential in the size of the specification. The case for reduced efficiency is not always so clear cut (e.g., for *krc* example), but further discussion of the issue is deferred to Section 5.

3. BOUNDED SATISFIABILITY CHECKING FOR METRIC TEMPORAL LOGIC

This section presents additional constraints to the encoding for PLTL of Section 2.4, to support some metric operators natively and more efficiently. For the sake of simplicity, first only the future fragment of metric PLTL is considered, and then, after some experimental assessment, the encoding is extended to metric past operators. The results of this section were originally presented in [Pradella et al. 2009].

3.1. Mono-infinite encoding of future metric operators

In our definition of metric temporal logic in Section 2.3, bounded until $\mathcal{U}_{\sim t}$ (where $\sim \in \{\leq, =, \geq\}$ and t is a natural number) is a primitive operator: all other metric operators are derived from it. The encoding of PLTL defined in Section 2.4 can still be applied to $\mathcal{U}_{\sim t}$ after an application of the following translation τ, which may be considered as an alternative way of providing the semantics of Metric PLTL.

$$\begin{array}{l}\tau(\phi_1 \mathcal{U}_{\leq 0} \phi_2) := \phi_2 \\ \tau(\phi_1 \mathcal{U}_{\leq t} \phi_2) := \phi_2 \vee \phi_1 \wedge \circ\tau(\phi_1 \mathcal{U}_{\leq t-1} \phi_2), \text{ with } t > 0 \\ \hline \tau(\phi_1 \mathcal{U}_{\geq 0} \phi_2) := \phi_1 \mathcal{U} \phi_2 \\ \tau(\phi_1 \mathcal{U}_{\geq t} \phi_2) := \phi_1 \wedge \circ\tau(\phi_1 \mathcal{U}_{\geq t-1} \phi_2), \text{ with } t > 0 \\ \hline \tau(\phi_1 \mathcal{U}_{= 0} \phi_2) := \phi_2 \\ \tau(\phi_1 \mathcal{U}_{= t} \phi_2) := \phi_1 \wedge \circ\tau(\phi_1 \mathcal{U}_{= t-1} \phi_2), \text{ with } t > 0\end{array}$$

This completely removes the metric nature of these operators. For instance, formula $\phi_1 \mathcal{U}_{\leq 2} \phi_2$ is translated into the PLTL formula:

$$\phi_2 \vee (\phi_1 \wedge \circ(\phi_2 \vee (\phi_1 \wedge \circ\phi_2))) .$$

The PLTL formula is then encoded as usual. When constant t is large, however, this method generates a very large formula, and, consequently, a very large Boolean encoding, which can slow down verification. As already noted in Section 2, in practice, the bounded until operator is not so useful, since the most common metric operators are actually the bounded eventually $\Diamond_{\sim t}\phi$, and the bounded globally $\Box_{\sim t}\phi$.

For instance, formula $\Diamond_{\leq d} p$ may be used to denote that p must occur within a deadline $d > 0$, and $\Box_{\geq d} q$ may be used to denote that condition q must hold indefinitely, starting from d instants in the future.

For these operators, it is possible to introduce an explicit, more compact Boolean encoding, and to show that in many cases this so called *metric encoding* gives considerable advantages, both in the size of the encoding and in SAT time, over their definition as derived from $\mathcal{U}_{\sim t}$.

To provide the encoding of every (future) metric operator of the form $\Diamond_{\sim t}$, and $\Box_{\sim t}$ with $t \geq 0, \sim \in \{=, \leq, \geq\}$, notice first that in a mono-infinite structure, by Table I, only the following cases are to be considered:

$$\Diamond_{=t}, \Diamond_{\leq t}, \Box_{\leq t}\phi$$

since all other cases of future bounded globally and eventually operators are derived immediately from them, with very simple formulae. For instance, a direct encoding of formula $\Diamond_{\geq t}\phi$ would not give substantially different results from encoding its equivalent definition $\Diamond_{=t}\Diamond\phi$.

Ideally, with an *unbounded* time structure, the encoding of the future metric operators should be:

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

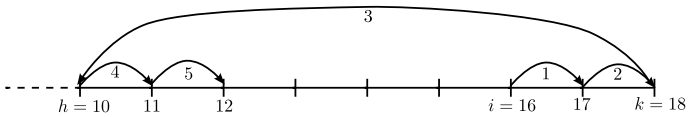

Fig. 5. Evaluation of formula $\Diamond_{=5}\phi$ at time $i = 16$, with $k = 18$, and $h = 10$.

The presence of a *bounded* time structure, in which infinity is encoded through a loop, makes the encoding less straightforward.

For the sake of readability, let $\lambda_i = k - i + 1$ be the number of time instants from i to the bound k and, for every integer m, n, let $\text{mod}(m, n)$ be the remainder of the integer division of m by n. To represent the values of subformulae inside the future loop, a new propositional variable, $\langle\langle \text{MF}(\phi, j)\rangle\rangle$, is introduced for every $0 \leq j \leq t - 1$ and for every formula ϕ such that one among $\Diamond_{=t}\phi$, $\Box_{\leq t}\phi$, $\Diamond_{\leq t}\phi$ is a subformula of Φ. The idea is that $\langle\langle \text{MF}(\phi, j)\rangle\rangle$ holds whenever there is an instant i, $1 \leq i \leq k$, such that l_i holds (i.e., there is a loop at i) and ϕ holds at $i + \text{mod}(j, \lambda_i)$ (i.e., ϕ holds j instants in the future of i, considering the loop). For instance, if the future loop selector is at instant $h = 10$ (i.e., l_{10} holds), then $\langle\langle \text{MF}(\phi, 2)\rangle\rangle$ represents $|[\phi]|_{12}$ (i.e. ϕ at instant 10+2). For $\Diamond_{=5}\phi$, variable $\langle\langle \text{MF}(\phi, j)\rangle\rangle$, $0 \leq j \leq 4$, represents the value of ϕ *j* time units after the starting point of the loop.

Table (9) defines ***MF*** for every metric future subformula of Φ.

$$\begin{array}{c|c} \varphi & 0 \leq j \leq t-1 \\ \hline \Diamond_{=t}\phi,\ \Box_{\leq t}\phi,\ \Diamond_{\leq t}\phi & \langle\langle \text{MF}(\phi, j)\rangle\rangle \longleftrightarrow \bigvee_{i=1}^{k} l_i \wedge |[\phi]|_{i+\text{mod}(j,\lambda_i)} \end{array} \tag{9}$$

Table (10) reports the translation of the above metric operators. It is composed of two parts: the first one defines the translation inside the bounded portion of the time domain (i.e., for instants i such that $i + t \leq k$), and the other one is based on ***MF*** in the loop portion.

$$\begin{array}{c|c} \varphi & 0 \leq i \leq k \\ \hline \Diamond_{=t}\phi & |[\Diamond_{=t}\phi]|_i \longleftrightarrow |[\phi]|_{i+t},\ \text{when } i + t \leq k \\ & |[\Diamond_{=t}\phi]|_i \longleftrightarrow \langle\langle \text{MF}(\phi, t - \lambda_i)\rangle\rangle,\ \text{otherwise} \\ \Box_{\leq t}\phi & |[\Box_{\leq t}\phi]|_i \longleftrightarrow \bigwedge_{j=0}^{\min(t,\lambda_i-1)} |[\phi]|_{i+j} \wedge \bigwedge_{j=\lambda_i}^{t} \langle\langle \text{MF}(\phi, j - \lambda_i)\rangle\rangle \\ \Diamond_{\leq t}\phi & |[\Diamond_{\leq t}\phi]|_i \longleftrightarrow \bigvee_{j=0}^{\min(t,\lambda_i-1)} |[\phi]|_{i+j} \vee \bigvee_{j=\lambda_i}^{t} \langle\langle \text{MF}(\phi, j - \lambda_i)\rangle\rangle \end{array} \tag{10}$$

Figure 5 shows a graphical interpretation of the second clause of Table (10), the one asserting that, when $i + t > k$, $|[\Diamond_{=t}\phi]|_i \longleftrightarrow \langle\langle \text{MF}(\phi, t - \lambda_i)\rangle\rangle$. Considering, for instance, formula $\Diamond_{=5}\phi$, current time $i = 16$, bound $k = 18$, future loop loop selector at $h = 10$, and $\lambda_i = k - i + 1 = 3$. The figure illustrates that the clause $|[\Diamond_{=t}\phi]|_i \longleftrightarrow \langle\langle \text{MF}(\phi, t - \lambda_i)\rangle\rangle$ is in this case $|[\Diamond_{=5}\phi]|_{16} \longleftrightarrow \langle\langle \text{MF}(\phi, 2)\rangle\rangle$: starting from time $i = 16$, 5 steps forwards are taken: after the first two steps the next one goes backwards to $h = 10$, and the last two steps proceed to time 12, which corresponds to the fact that, for $h = 10$, $\langle\langle \text{MF}(\phi, 2)\rangle\rangle$ is $|[\phi]|_{12}$.

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

LEMMA 3.1. Correctness of the mono-infinite future metric encoding. *The above metric encodings of the three basic metric operators, $\Diamond_{=t}\phi$, $\Box_{\leq t}\phi$, and $\Diamond_{\leq t}\phi$, are equivalent to those of equivalent non-metric formulae in the encoding presented in Section 2.4 for LTL.*

PROOF. Formula $\Diamond_{=t}\phi$ is equivalent to the LTL formula $\circ^t\phi$, hence we show that the metric encoding of $\Diamond_{=t}\phi$ is equivalent to the LTL encoding of $\circ^t\phi$.

In case $i+t \leq k$, according to the metric encoding, $|[\Diamond_{=t}\phi]|_i \longleftrightarrow |[\phi]|_{i+t}$ (by the first line of Table (10)), while, according to the LTL encoding, $|[\circ^t\phi]|_i \longleftrightarrow |[\circ^{t-1}\phi]|_{i+1} \longleftrightarrow |[\circ^{t-2}\phi]|_{i+2} \longleftrightarrow$ ... $\longleftrightarrow |[\phi]|_{i+t}$, by repeated application of the first clause of Table (4).

In case $i + t > k$, in the metric encoding we have $|[\Diamond_{=t}\phi]|_i \longleftrightarrow \langle\langle \text{MF}(\phi, t - \lambda_i)\rangle\rangle$; since the future loop selector position h is the (only) value such that l_h holds, then, from Table (9) and the second line of Table (10), $\langle\langle \text{MF}(\phi, t - \lambda_i)\rangle\rangle \longleftrightarrow |[\phi]|_{h+mod(t-\lambda_i,\lambda_h)}$; on the other hand, according to the LTL encoding, we have, by repeated application of the first clause of Table (4), $|[\circ^t\phi]|_i \longleftrightarrow |[\circ^{t-(k-i)}\phi]|_k$; then by the last state constraint (clause 2), $|[\circ^{t-(k-i)}\phi]|_k \longleftrightarrow |[\circ^{t-k+i-1}\phi]|_h \longleftrightarrow |[\circ^{t-\lambda_i}\phi]|_h$; by repeatedly applying the first clause of Table (4) and the last state constraint to cycle, possibly many times, through the forward loop, $|[\circ^{t-\lambda_i}\phi]|_h \longleftrightarrow |[\circ^{mod(t-\lambda_i,\lambda_h)}\phi]|_h \longleftrightarrow |[\phi]|_{h+mod(t-\lambda_i,\lambda_h)}$.

Formula $\Box_{\leq t}\phi$, whose metric encoding is reported in the third line of Table (10), is equivalent to the LTL formula $\bigwedge_{j=0}^{t} \Diamond_{=j}\phi$. The encoding of those conjuncts $\Diamond_{=j}\phi$, for which $i + j \leq k$, corresponds to the part $\bigwedge_{j=0}^{\min(t,\lambda_i-1)} |[\phi]|_{i+j}$, while the encoding of those $\Diamond_{=j}\phi$ for which $i+j > k$ (if there are any, i.e., if $i + t > k$) is obtained by the part $\bigwedge_{j=\lambda_i}^{t} \langle\langle \text{MF}(\phi, j - \lambda_i)\rangle\rangle$ in the same line of the Table (10). Therefore the correctness of the encoding of formula $\Box_{\leq t}\phi$ follows directly from that of $\Diamond_{=t}\phi$.

A similar reasoning applies to the metric encoding of formula $\Diamond_{\leq t}\phi$, reported in the fourth line of Table (10): notice that $\Diamond_{\leq t}\phi$ is equivalent to the LTL formula $\bigvee_{j=0}^{t} \Diamond_{=j}\phi$, that the part $\bigvee_{j=0}^{\min(t,\lambda_i-1)} |[\phi]|_{i+j}$ takes care of the time points before the bound k, while the part $\bigvee_{j=\lambda_i}^{t} \langle\langle \text{MF}(\phi, j - \lambda_i)\rangle\rangle$ considers the values of j (if any) such that $i + j > k$. $\Box$

A first assessment of the encoding The behavior of the metric encoding has been first experimented on the very simple specification of Example 2.1, using the non-metric encoding of Section 2.4 and comparing it to the above metric encoding.

The experimental results (with the hardware and software setup described in Section 2.6) are graphically shown in Figures 6,7, where Gen represents the generation phase, i.e., the generation starting from the above specification, of a Boolean formula in conjunctive normal form, of size CNF clauses, and Solver 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 diagrams show the time, in seconds, for Gen and Solver phases, using either a PLTL encoding or the metric encoding, as a function of delay d, also reporting the number of CNF clauses, while the last diagram shows the speedup, as the ratio between the PLTL encoding values and the metric encoding values, again as a function of delay d. As one can see, the speedup obtained for both the Gen and Solver phases is roughly proportional to delay d, and can be quite substantial (up to 300% for Solver and 700% for Gen phases). Also the ratio between the size of the generated Boolean formula for PLTL and metric encodings increases with the value of d and tends to reach a stable value around 145%.

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 non-metric encoding 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 non-metric encoding, due to the presence of a larger number of subformulae.

Regarding the size of the generated constraints for the formula $\Diamond_{=d}out$, computed here as the sum of the number of formula variables for each constraint, it is immediate to notice that propositional, eventuality and loop constraints 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 constraints are those of Tables (9) and (10) introduced for the metric encoding only.

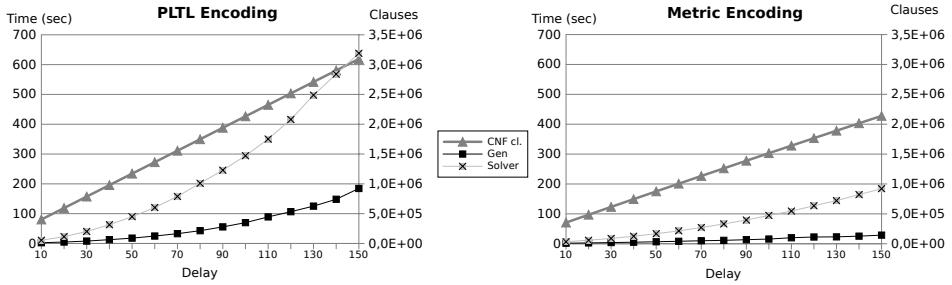

Fig. 6. Summary of experimental data for the synchronous version of a Shift Register, with Generation time, Solver time and number of generated CNF clauses plotted as functions of the delay. In each diagram, the left-hand vertical axis measures time in seconds (for Generation and Solver), while the right-hand vertical axis measures the number of clauses (for CNF clauses).

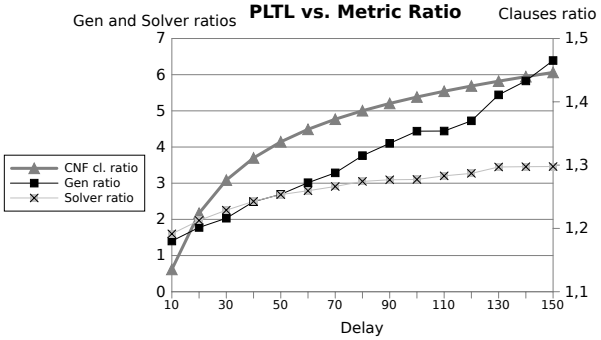

Fig. 7. Ratios for Generation times, Solver times and CNF-clauses of PLTL vs. Metric encodings. The left-hand vertical axis measures the ratio for Generation and Solver, while the right-hand vertical axis measures the ratio for CNF clauses.

	Last state	Temporal Sub.	MF	Total
PLTL	$3k \cdot (d+1)$	$2(k+1) \cdot d$	0	$5k \cdot d + 3k + 2d$
Metric	$6k$	$2(k+1)$	$2(k+1) + d \cdot (2k+1)$	$2k \cdot d + 10k + d + 4$

Thus, in the metric encoding we have $O(d + k)$ variables (i.e., less than in the non-metric case) and a size of constraints that is $O(d \cdot k)$, i.e. the same as in the non-metric case but with a smaller constant factor (2 rather than 5). This is also clear from Figure 7, 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.

3.2. Mono-infinite encoding of past metric operators

Next, we present the constraints for past metric operators of Section 2.3, for a mono-infinite time structure. Again, we will not consider the bounded since operator, because it is rarely used in practice and hard to optimize. Its current implementation is based on its translation using an approach analogous to the one defined for until at the beginning of Section 3.1 (namely, using $\bullet$ instead of $\circ$). By virtue of Table I, we can consider only the following cases: $\blacklozenge_{=t}$, $\blacksquare_{=t}$, $\blacklozenge_{\leq t}$, $\blacksquare_{\leq t}\phi$, assuming moreover that $t > 0$ (since the case $t = 0$ is trivial). In the actual implementation in Zot, operators $\blacklozenge'_{\leq t}$ and $\blacksquare'_{\leq t}$ are natively encoded. Their encodings are straightforward variants of those of $\blacksquare_{\leq t}$, and $\blacklozenge_{\leq t}$, so we prefer their presentation here, for the sake of brevity.

The first group of constraints encode the past operators for all time points without considering the possible presence of a future loop.

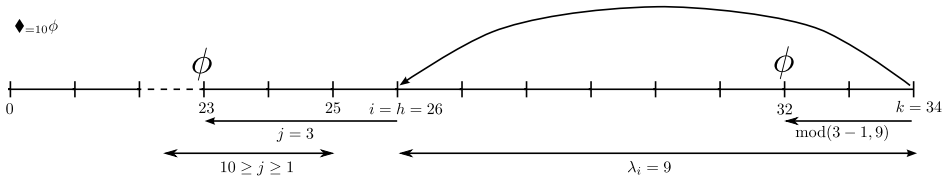

Fig. 8. An illustration of the *Past in Loop* constraint (ix) for $\blacklozenge_{=10}\phi$.

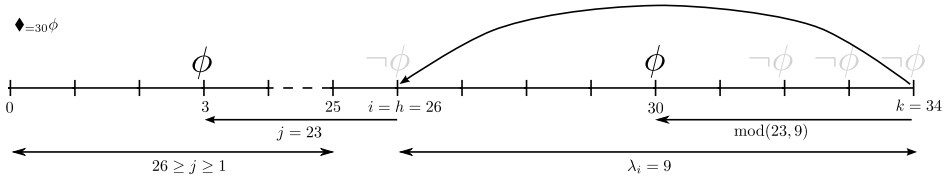

Fig. 9. An illustration of the *Past in Loop* constraint (ix) for $\blacklozenge_{=30}\phi$. Grey $\neg\phi$'s are due to the bottom line of the constraint (for $27 \leq j \leq 30$).

φ		$0 \leq i < t$
(i)	$\blacklozenge_{=t}\phi$	$\neg\lvert[\blacklozenge_{=t}\phi]\rvert_i$
(ii)	$\blacksquare_{=t}\phi$	$\lvert[\blacksquare_{=t}\phi]\rvert_i$
(iii)	$\blacklozenge_{\leq t}\phi$	$\lvert[\blacklozenge_{\leq t}\phi]\rvert_i \longleftrightarrow \bigvee_{j=0}^{i} \lvert[\phi]\rvert_j$
(iv)	$\blacksquare_{\leq t}\phi$	$\lvert[\blacksquare_{\leq t}\phi]\rvert_i \longleftrightarrow \bigwedge_{j=0}^{i} \lvert[\phi]\rvert_j$

φ		$t \leq i \leq k+1$
(v)	$\blacklozenge_{=t}\phi$	$\lvert[\blacklozenge_{=t}\phi]\rvert_i \longleftrightarrow \lvert[\phi]\rvert_{i-t}$
(vi)	$\blacksquare_{=t}\phi$	$\lvert[\blacksquare_{=t}\phi]\rvert_i \longleftrightarrow \lvert[\phi]\rvert_{i-t}$
(vii)	$\blacklozenge_{\leq t}\phi$	$\lvert[\blacklozenge_{\leq t}\phi]\rvert_i \longleftrightarrow \bigvee_{j=0}^{t} \lvert[\phi]\rvert_{i-j}$
(viii)	$\blacksquare_{\leq t}\phi$	$\lvert[\blacksquare_{\leq t}\phi]\rvert_i \longleftrightarrow \bigwedge_{j=0}^{t} \lvert[\phi]\rvert_{i-j}$

The next part is about the behavior of the past operators when a future loop is present. These constraints are called *stabilization forcing constraints* in [Biere et al. 2006]. They are necessary, because we do not use any virtual unrolling technique, so we need to force all past subformulae to "stabilize" in the loop. In this case, since the state after S_k is S_h, then state S_h has two previous states, namely S_{h-1} and S_k. Therefore if any past formula stated inside the loop asserts a property concerning states $S_{h-1}, S_{h-2}, S_{h-3}, ...$, then the encoding must ensure that the same property holds for the state sequence $S_k, S_{k-1}, S_{k-2}, ...$ as well.

Past in Loop constraints:

φ		$1 \leq i \leq k$
(ix)	$\blacklozenge_{=t}\phi$	$l_i \rightarrow \left(\begin{array}{c} \bigwedge_{j=1}^{\min(t,i)} (\lvert[\phi]\rvert_{i-j} \longleftrightarrow \lvert[\phi]\rvert_{k-\mathrm{mod}(j-1,\lambda_i)}) \wedge \\ \bigwedge_{j=i+1}^{t} \left(\neg\lvert[\phi]\rvert_{k-\mathrm{mod}(j-1,\lambda_i)} \right) \end{array} \right)$
(x)	$\blacksquare_{=t}\phi$	$l_i \rightarrow \left(\begin{array}{c} \bigwedge_{j=1}^{\min(t,i)} (\lvert[\phi]\rvert_{i-j} \longleftrightarrow \lvert[\phi]\rvert_{k-\mathrm{mod}(j-1,\lambda_i)}) \wedge \\ \bigwedge_{j=i+1}^{t} \left(\lvert[\phi]\rvert_{k-\mathrm{mod}(j-1,\lambda_i)} \right) \end{array} \right)$

φ	$1 \leq i \leq k$
(xi) $\blacklozenge_{\leq t}\phi$	$\mathrm{InLoop}_i \rightarrow \left(\lvert[\blacklozenge_{\leq t}\phi]\rvert_i \longleftrightarrow \left(\begin{array}{c} \bigvee_{j=0}^{\min(i,t)} (\mathrm{InLoop}_{i-j} \wedge \lvert[\phi]\rvert_{i-j}) \vee \\ \bigvee_{j=0}^{\min(k-i,t)} (\neg\mathrm{InLoop}_{\max(0,i-t+j)} \wedge \lvert[\phi]\rvert_{k-j}) \end{array} \right) \right)$
(xii) $\blacksquare_{\leq t}\phi$	$\mathrm{InLoop}_i \rightarrow \left(\lvert[\blacksquare_{\leq t}\phi]\rvert_i \longleftrightarrow \left(\begin{array}{c} \bigwedge_{j=0}^{\min(i,t)} (\neg\mathrm{InLoop}_{i-j} \vee \lvert[\phi]\rvert_{i-j}) \wedge \\ \bigwedge_{j=0}^{\min(k-i,t)} (\mathrm{InLoop}_{\max(0,i-t+j)} \vee \lvert[\phi]\rvert_{k-j}) \end{array} \right) \right)$

Clause (ix) provides constraints that relate the value of ϕ inside the loop with that at time points immediately preceding it, both inside and outside the temporal domain. Let us preliminarily remark

that $i - \min(i,t) = 0$ if $i \leq t$, and $i - \min(i,t) = i - t$ if $i \geq t$; therefore subformula $|[\phi]|_{i-j}$ of clause (ix) refers to the time points preceding i but only up to the distance t, or to time point 0, if the latter is closer. Then, assuming that l_i holds, i.e., that the loop selector variable is true for position i, the first line of the clause, $\bigwedge_{j=1}^{\min(t,i)}(|[\phi]|_{i-j} \longleftrightarrow |[\phi]|_{k-\mathrm{mod}(j-1,\lambda_i)})$, asserts that the value of ϕ at the time points from $i-1$ back to $i-t$ (or to 0 if $t \geq i$) are respectively equal to the value of ϕ at the corresponding time points in the segment of the loop immediately preceding point k: for each j, with $1 \leq j \leq \min(t,i)$, the time point $i-j$ outside the loop is matched by the point $k - \mathrm{mod}(j-1,\lambda_i)$ inside the loop. The expression $\mathrm{mod}(j-1,\lambda_i)$ takes into account the fact that, if $j-1$ is greater than the loop length λ_i, one step is taken "back" from point i to point k, then the loop is traversed $\mathrm{div}(j-1,\lambda_i)$ times and finally $\mathrm{mod}(j-1,\lambda_i)$ steps back from the last point k are taken. The second line of the clause, $\bigwedge_{j=i+1}^{t}\left(\neg|[\phi]|_{k-\mathrm{mod}(j-1,\lambda_i)}\right)$, asserts that, if $t > i$ so that the formula $\blacklozenge_{=t}\phi$ refers to a point $i - t < 0$ outside the temporal domain, since $|[\phi]|_{i-t}$ is false by convention, then the value $|[\phi]|_{k-\mathrm{mod}(j-1,\lambda_i)}$ referring to the corresponding point inside the loop, must also be false.

Figure 8 illustrates the application of Clause (ix), to the formula $\blacklozenge_{=10}\phi$, evaluated at time $i = h = 26$ so that l_i is true, highlighting the possible values of index j and showing the correspondence between two particular points outside the loop and inside it. Since $i > t$ the top line of clause (ix) considers values of j such that $1 \leq j \leq 10$; for the specific value of $j = 3$, the two matching points inside and outside the loop are at instants 23 and 32.

Figure 9 also illustrates the application of clause (ix), now considering formula $\blacklozenge_{=30}\phi$ (still evaluated at time $i = h = 26$), for which $t > i$, and shows the consequence of applying both the top and bottom line parts. Concerning the top line, for the specific value of $j = 23$ the two matching time points outside and inside the loop are the instants 3 and 30 (notice that $30 = 34 - \mathrm{mod}(23-1,9)$: the loop is traversed twice). Concerning the bottom line of clause (ix), the grey $\neg\phi$'s correspond to values of index j such that $27 \leq j \leq 30$.

Clause (x) is analogous to clause (ix), except for the different default value of ϕ outside the time domain.

Clause (xi) provides constraints, additional to those of clauses (iii) and (vii), for the case in which formula $\blacklozenge_{\leq t}\phi$ is evaluated inside the loop (i.e., variable InLoop_i is true). Its two lines refer to sets of time points, where the subformula ϕ may hold, that precede point i (top line) or follow it (bottom line). The bottom line uses a correspondence between time points $i - t + j$, and time points $k - j$ based on the common value for variable j, as explained, and illustrated through examples, in the next paragraph.

Figure 10 and Figure 11 illustrate how index j is used in the bottom line of clause (xi) to denote corresponding time points inside and outside the loop. Figure 10 refers to a case in which $i > t$, hence all referenced time points are inside the time domain, while Figure 11 considers a case in which $i < t$, so that some of the referenced time points are outside the time domain. It can be noticed that, in the bottom line of clause (xi), increasing values of index j denote a sequence of time points from left to right when outside the loop, and from right to left when inside the loop.

Clause (xii) is the dual of clause (xi) and uses the same indexing method to denote the relevant time points.

In Figure 12 a scenario for the formula $\blacksquare_{\leq 5}\phi$, evaluated at instant $i = 8$, and with l_6 true, is presented. It can be noticed that $i > t$, so that all referenced time points are inside the time domain. The top line of constraint (xii) requires the three occurrences of ϕ at instants from 6 to 8, while the bottom line corresponds to the parts depicted in grey. In fact, instants from 3 to 5 are outside the loop, hence ϕ must hold at the corresponding instants $k-2$, $k-1$, and k.

Figure 13 depicts a scenario for the formula $\blacksquare_{\leq 10}\phi$, evaluated at instant $i = 8$, and with l_6 true. Now $i < t$, so that some of the time points referenced by the $\blacksquare_{\leq 10}\phi$ formula are outside the time domain. Again the first line requires the three ϕ at instants from 6 to 8; now the referenced time points outside the loop include all instants from 0 to 5 and they correspond to time instants from $k-5$ to k, where the ϕ is depicted in grey.

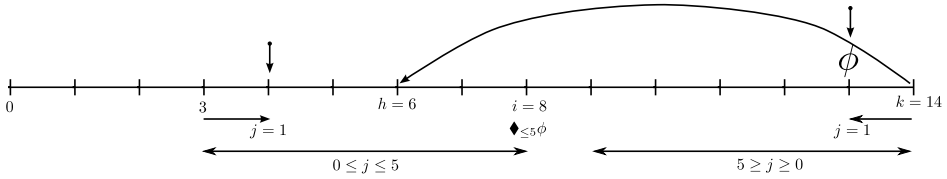

Fig. 10. An illustration of the *Past in Loop* constraint (xi) for $\blacklozenge_{\leq 5}\phi$.

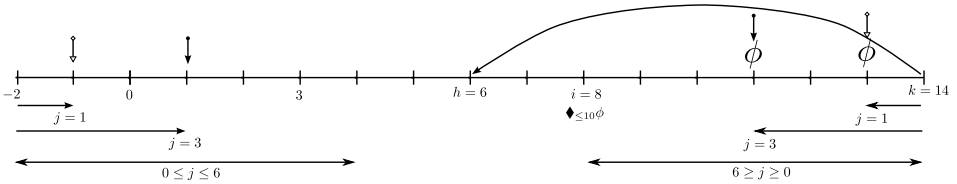

Fig. 11. An illustration of the *Past in Loop* constraint (xi) for $\blacklozenge_{\leq 10}\phi$.

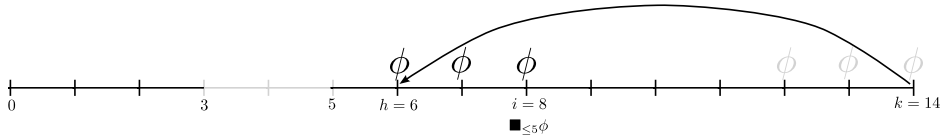

Fig. 12. An illustration of the *Past in Loop* constraint (xii) for $\blacksquare_{\leq 5}\phi$. The black ϕ's derive from the top line of the constraint, while the parts from the bottom line are depicted in grey.

LEMMA 3.2. Correctness of the mono-infinite encoding for past metric operators. *The above metric encodings of the four basic operators $\blacklozenge_{=t}$, $\blacksquare_{=t}$, $\blacklozenge_{\leq t}$, $\blacksquare_{\leq t}\phi$ are consistent with the semantics of PLTL as defined in Sections 2.2 and 2.3.*

PROOF. We consider in turn the four operators.

Let us first consider the operator $\blacklozenge_{=t}$. Clause (i) encodes the fact that $\blacklozenge_{=t}\phi$ is false when asserted at a time point $i < t$, and clause (v) that $\blacklozenge_{=t}\phi$ asserted at time $i \geq t$ is equivalent to ϕ asserted at $i - t$. Clause (ix) encodes the stabilization forcing constraints, ensuring that past formulas are stable in the future loop. It encodes the relation between the values of the ϕ subformula at corresponding time points, inside the loop and outside it, that are within the stated distance t: the top line concerns points, outside the loop, that are inside the time domain, while the bottom line covers any point outside the time domain, where ϕ is conventionally false.

Considering the operator $\blacksquare_{=t}$, clauses (ii), (vi), and (x) are dual of clauses (i), (v), and (ix), in line with the fact that formula $\blacksquare_{=t}\phi$ is conventionally true when asserted at a time point $i < t$.

For what concerns operator $\blacklozenge_{\leq t}$, clause (iii) considers the case when $i < t$ and states that only the values of ϕ at time points from 0 to i determine the value of $\blacklozenge_{\leq t}\phi$ at time $i < t$. Clause (vii) covers the case in which $i \geq t$: then it considers the value of ϕ at all time points from $i-t$ to i, since they are all inside the temporal domain. Clause (xi) encodes the stabilization forcing constraints. Its top line concern points, where the subformula ϕ may hold, that precede point i. In this case the subformula $\bigvee_{j=0}^{\min(i,t)}(\mathrm{InLoop}_{i-j} \wedge |[\phi]|_{i-j})$ states that $|[\phi]|$ holds at some of the time points preceding i: by means of the upperbound $\min(i,t)$ in the value of index j, it ensures that the considered time points, preceding i, where $|[\phi]|$ is required to hold, are at most t and do not precede the initial time instant

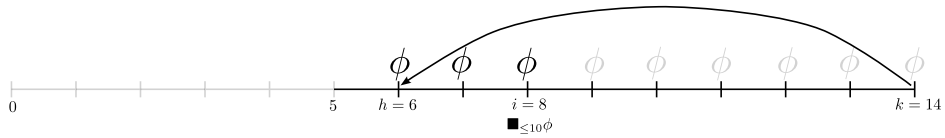

Fig. 13. An illustration of the *Past in Loop* constraints (xii) for $\blacksquare_{\leq 10}\phi$. The black ϕ's derive from the top line of the constraint, while the parts from the bottom line are depicted in grey.

0. The bottom line of clause (xi)

$$\bigvee_{j=0}^{\min(k-i,t)} (\neg \mathrm{InLoop}_{\max(0,i-t+j)} \wedge |[\phi]|_{k-j})$$

encodes the stabilization forcing constraints by ensuring that if $|[\phi]|$ holds for states S_{h-1}, S_{h-2}, S_{h-3}, ..., then it holds also for the state sequence S_k, S_{k-1}, S_{k-2}, ... The subformula asserts that $|[\phi]|$ holds at some of the time points that are inside the loop, following point i: these points are denoted by $k-j$, with $j \in [0..\min(k-i,t)]$, and match, through the value of index j, points $i-t+j$ that precede i and are positioned outside the loop, as expressed by the negated variable *InLoop*. Since variable *InLoop* is defined only for non-negative time instants, the subformula $\neg \mathrm{InLoop}_{\max(0,i-t+j)}$ includes the subscript $\max(0, i-t+j)$, which ensures that no reference is made to negative time points positioned outside the time domain.

Concerning the operator $\blacksquare_{\leq t}$, clauses (iv) and (viii) are dual to clauses (iii) and (vii) where disjunctions are substituted by conjunctions, in agreement with the universal quantification expressed by the operator. Clause (xii) is dual of clause (xi), and uses the same indices to denote the same points. □

3.3. Experimental results

Figure 14 summarizes the comparison of metric and PLTL encodings on the various case studies (with the usual exception of the bi-infinite allocator), including both operational and descriptive models. We computed the ratio of the results obtained by using the PLTL encoding of Section 2.4 and the results obtained by using the metric encoding of this section, and then averaged over the usual bounds 30, 60 and 90 and the various properties checked for each model. On average, the ratio are 1.60, 1.88 and 1.28 for Gen time, Solver time and CNF clauses, respectively. However, there is large variability, since operational models have often a lower ratio. For instance, the operational models for Fischer and the synchronous shift register are close to a ratio of 1. This is due to the fact that the operational models do not use metric operators: the ratio for an operational model is around 1 (i.e., no gain and no loss), unless the property to be analyzed is a metric formula. Both descriptive models with metric constraints and operational models to be checked against complex metric temporal properties may instead considerably gain by using the metric encoding, such as in the case of the Kernel Railroad Crossing and the Timed Lamp.

4. BI-INFINITE TIME

Historically, past operators and bi-infinite time are as old as temporal logic itself [Prior 1967], and widely adopted in logic and philosophy [Rescher and Urquhart 1971]. The reason is that all definitions are symmetrical and one has not to worry of a "first time instant". In computer science, however, mono-infinite time has been almost universally used. One of the likely reasons is that temporal logic was introduced to study properties of programs and of finite-state models [Pnueli 1977], i.e., of computational systems with an initial state, whose typical models are ω-words or mono-infinite computation paths. Following this approach, only future temporal operators were considered. When past operators [Lichtenstein et al. 1985] were introduced in computer science,

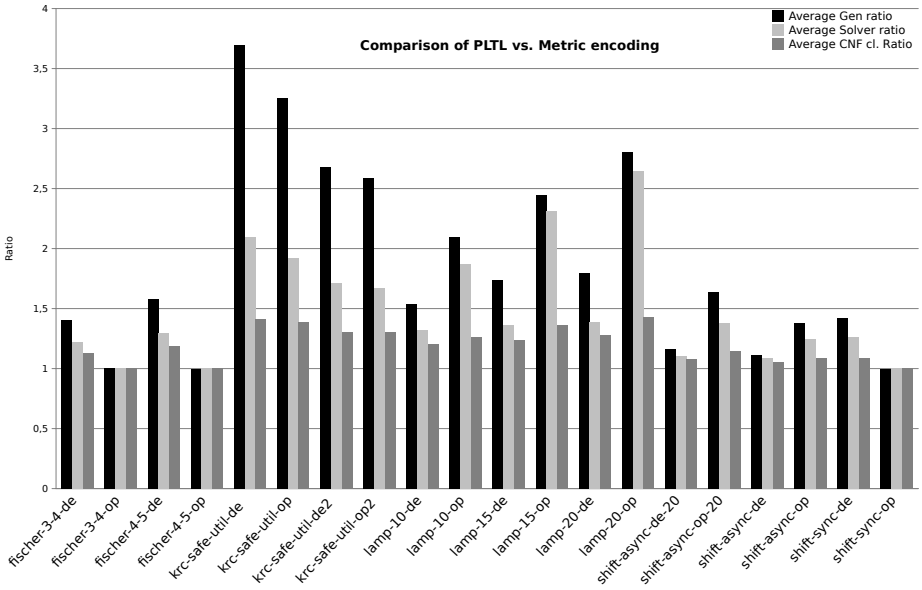

Fig. 14. Summary of experimental comparison of PLTL vs. metric encodings. The comparison is expressed as the average ratio of time (for Gen and Sat) or Clauses (for CNF clauses) obtained in the PLTL encoding and in the metric encoding of the same specification. The higher the ratio the better the performance of the metric encoding with respect to the PLTL one.

because of their convenience and conciseness in many cases, the underlying models were still ω-words. Later, also model checking, developed for the verification of finite transition systems with initial states, again considered only ω-words.

However, when temporal logic is used to describe not only properties but also to specify (parts of) a complete system, then it is more natural to adopt bi-infinite time. Hence, our own past works with TRIO [Ghezzi et al. 1990; Morzenti and San Pietro 1994; Ciapessoni et al. 1999; Gargantini and Morzenti 2001] on temporal logic specifications always considered bi-infinite time, although allowing for different time domains when needed [Morzenti et al. 1992; Felder and Morzenti 1994; Mandrioli et al. 1995; San Pietro et al. 2000], e.g. for verification.

Here we argue that interpreting a PLTL specification on bi-infinite time may be convenient to allow for more abstract and simpler specifications.

First, analogously to the mono-infinite case where termination may be ignored, bi-infinite time is convenient to deal with system models where initialization may be ignored. This may add another layer of abstraction, since one can write specifications that are simpler and more easily understandable, because they do not include the description of the operations (such as configuration, installation, ...) typically performed at system deployment time. For instance, for reactive systems embedded into devices that continuously monitor or control some process, initialization may often be ignored and one may focus only on routine behavior.

Second, bi-infinite time for PLTL also helps in solving a technical problem, called the "border effect" [Morzenti et al. 1992; Coen-Porisini et al. 1998], which may make specification rather cumbersome. The problem arises in conjunction with bounded temporal operators, such as the "previous time" $\bullet$ operator. We show here some of these issues through examples.

Example 4.1. *A PLTL version of the synchronous shift register* Consider again Example 2.1, where a transmission line receives one bit at one end and delivers it at the opposite end with a fixed delay d. Assume that $d = 1$. The following PLTL formula, which uses $\bullet$ rather than $\circ$, expresses that every received message is delivered, and no spurious message is emitted (i.e., every out is preceded

by an in and if there is no out then there was no in).

$$\Box\left((out \rightarrow \bullet in) \wedge (\neg out \rightarrow \bullet \neg in)\right) \tag{11}$$

Since time is finite "on the left", the evaluation of past operators may be problematic at the very first time instant. The traditional solution, already used in the encoding of Section 2.4 is to return a "default" false value when the evaluation of a subformula is outside the time domain (the typical PLTL semantics of $\bullet\phi$ is $w, i \models \bullet\phi \iff i - 1 \geq 0 \wedge w, i-1 \models \phi$, which is false if $i = 0$). This definition may easily lead to subtle specification errors. For instance, the above specification not only is not equivalent to the one given in Example 2.1 for $d = 1$, but it is even unsatisfiable: at instant 0, both $\bullet\neg in$ and $\bullet in$ are false. Therefore, at instant 0, if out holds then formula $out \rightarrow \bullet in$ is false; otherwise, out does not hold at 0 and $\neg out \rightarrow \bullet\neg in$ is false. But if we rewrite $\bullet\neg in$ as $\neg \bullet in$, the original formula becomes satisfiable. This is because, in general $\neg\bullet\phi$ may have a different value from $\bullet\neg\phi$. Hence, for instance, Property (11) above is not equivalent to $\Box(out \longleftrightarrow \bullet in)$, which, by definition of Boolean operator $\longleftrightarrow$, is $\Box((out \rightarrow \bullet in) \wedge (\bullet in \rightarrow out))$. This behavior may be somehow "fixed" by allowing two different forms of the $\bullet$ operators, the second one being defined to the default true value when its argument cannot be evaluated: this dual version of $\bullet$ is usually denoted by $\bullet'$ and was introduced in Section 2.4 with the goal of allowing positive normal form. This is clearly cumbersome and counterintuitive.

Similar, and sometimes more serious, semantic problems arise for the other past operators of metric temporal logic. Clearly, the same example above, for the case $d > 1$, can be stated as:

$$\Box\left((out \rightarrow \blacklozenge_{=d} in) \wedge (\neg out \rightarrow \blacklozenge_{=d} \neg in)\right) \tag{12}$$

which gives the same semantics problems. A correct reformulation is:

$$\Box\left((out \rightarrow \blacklozenge_{=d} in) \wedge (\neg out \rightarrow \blacksquare_{=d} \neg in)\right)$$

since $\blacklozenge_{=d}, \blacksquare_{=d}$ are, respectively, false and true by default in the time instants from 0 to $d-1$, regardless of the value of their arguments.

Example 4.2. *A simple monitoring system* Another example with other metric operators is a monitoring system that must guarantee that when the monitored system is $warm$ for at least 5 consecutive time instants then an $alarm$ condition is raised, which is then withdrawn immediately when the system cools down. A very simple, but wrong, specification of this property is:

$$\Box(alarm \longleftrightarrow \blacksquare_{\leq 5} warm).$$

The above formula is wrong since, for example, a single instance of $warm$ at instant 0 would cause an incongruous $alarm$ at instant 1. Its correct specification relies instead on operator $\blacksquare'_{\leq 5}$, which is false by default in the first 4 time instants, hence no spurious alarm may be raised:

$$\Box(alarm \longleftrightarrow \blacksquare'_{\leq 5} warm).$$

Other examples can be found in [Coen-Porisini et al. 1998], where a different solution is proposed, which however changes the semantics of temporal logic: Kleene's three-valued logic is used for the propositional part, with the third value standing for "unevaluatable". That approach has not found generalized adoption, also because it may hamper bounded model checking verification, since a three-valued logic has a more complex and lengthy Boolean encoding.

The simplest and most effective solution is to adopt *bi-infinite time*, where the past operators are always defined. Examples 4.1 and 4.2 can be written, respectively, as:

$$\mathcal{A}lw\left((out \rightarrow \blacklozenge_{=d} in) \wedge (\neg out \rightarrow \blacklozenge_{=d} \neg in)\right),$$

and

$$\mathcal{A}lw(alarm \longleftrightarrow \blacksquare_{\leq 5} warm).$$

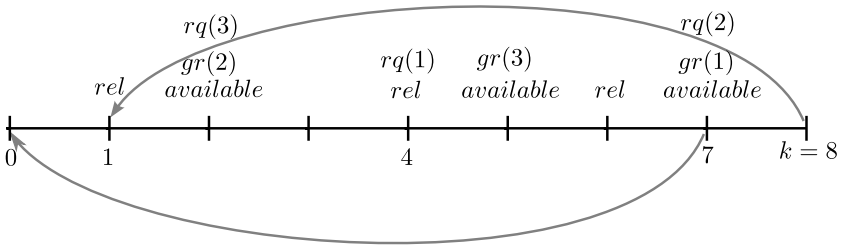

Fig. 15. A bi-infinite behavior of the real-time allocator.

With bi-infinite time, the outermost operator is typically $\mathcal{A}lw$, rather than $\Box$, since the intended meaning is that the formula must hold "for every instant", while $\Box$ only means "for every instant from now on".

For instance, there is no spurious alarm in this version of Example 4.2, since $\blacksquare_{\leq 5} warm$ is true if and only $warm$ holds for at least 5 consecutive instants in the past, without exceptions.

Notice that the usage of bi-infinite time does not rule out finite or infinite-in-future behaviors. The last formula allows finite, infinite-in-the-future, infinite-in-the-past and bi-infinite intervals where alarm and warm may occur. If infinite-in-the-past intervals are to be ruled out, e.g., because the system has an initial state, it is enough to add the axiom: $\mathcal{S}om(\blacksquare \neg warm)$ (again, notice the use of $\mathcal{S}om$ with bi-infinite time rather than $\Diamond$). In fact, bi-infinite time allows for the explicit modeling of the initial state of a system, and hence it incurs in no loss of expressive power (e.g., just use a propositional symbol $start$, with the additional constraint that $start$ must occur exactly once at some time and nothing happens before it).

Similarly, above formula (12) (which was "wrong" on mono-infinite time) when written as:

$$\mathcal{A}lw\,((out \rightarrow \blacklozenge_{=d} in) \wedge (\neg out \rightarrow \blacklozenge_{=d} \neg in))$$

has the correct meaning and it is equivalent, unlike in the mono-infinite case, to:

$$\mathcal{A}lw\,((out \rightarrow \blacklozenge_{=d} in) \wedge (\neg out \rightarrow \blacksquare_{=d} \neg in))$$

since in bi-infinite time $\blacklozenge_{=d} in$ is equivalent to $\blacksquare_{=d} in$.

Example 4.3. *Real-time allocator* As a more complex example of a formula that can be correctly analyzed only with respect to a bi-infinite time domain, let us consider the assumption of unconstrained rotation, for the real-time allocator case study (see Section 2.5.3), under which the property of *conditional fairness* can be correctly checked. The property specifies that a process will ask for the resource only after all other ones have requested and obtained it, without imposing any specific order in the requests (the predicates used in the formula are explained in Section 2.5.3).

$$\forall p\ \mathcal{A}lw \left(\begin{array}{c} rq(p) \rightarrow \\ \forall q \left(q \neq p \rightarrow \bullet \left(\neg rq(p)\ \mathcal{S} \left(\begin{array}{c} rq(q) \wedge \\ \circ \Diamond_{\leq T_{req}} gr(q) \end{array} \right) \right) \right) \end{array} \right)$$

If unconstrained rotation is checked for satisfiability with reference to a mono-infinite time domain (this requires to replace the outermost $\mathcal{A}lw$ operator with the future-only operator $\Box$) the only obtained behavior is the one where there is no request and the resource is always available; if one imposes, by means of an additional formula, the presence of some request, then the overall property (unconstrained rotation and presence of some request) is unsatisfiable. The property of unconstrained rotation is however satisfiable in a non-trivial way with respect to a bi-infinite time domain: Figure 15 shows a bi-infinite behavior that satisfies it. The bi-infinite time domain is represented by means of two loops, one for the future and one for the past (the past loop is introduced in the coming Section 4.1); the two loops overlap, as they share the time points from 1 to 7. For the

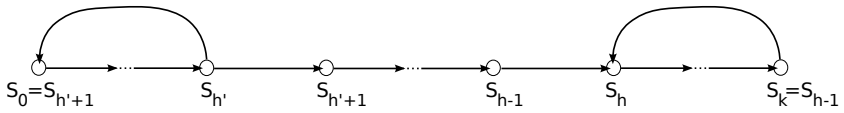

Fig. 16. A bi-infinite temporal structure

sake of readability the figure includes only the predicates *available*, *rq*, *gr*, *rel*; requests by the three processes repeat indefinitely, in the order $rq(1) - rq(2) - rq(3)$, both in the future and in the past.

Semantics The semantics of PLTL and MPLTL is here defined on *bi-infinite words*. A bi-infinite word w over a finite alphabet Σ is a function $w : \mathbb{Z} \longrightarrow \Sigma$. A bi-infinite language is a set of bi-infinite words. Finite automata and linear temporal logic formulas with an outermost $\mathcal{A}lw$ or $\mathcal{S}om$ operator may only define shift invariant languages [Perrin and Pin 2004], i.e., languages where instant 0 (the "origin" for ω-words) has no special role.

The definition in Sections 2.2, 2.3 of the semantics is unchanged when w is a bi-infinite word and index i ranges on $\mathbb{Z}$, except for the following cases, corresponding to past operators ($\mathcal{S}$ is not reported since it is equivalent to $\mathcal{S}_{\geq 0}$):

$$w, i \models \bullet\phi \quad \longleftrightarrow \; w, i-1 \models \phi$$
$$w, i \models \phi \mathcal{S}_{\sim t} \psi \; \longleftrightarrow \; \exists k \geq 0 : k \sim t,\; w, i-k \models \psi, \text{ and } \forall 0 \leq j < k :\; w, i-j \models \phi.$$

A formula ϕ on a propositional alphabet Ap is satisfiable if there exists a bi-infinite word $w \in (2^{Ap})^{\mathbb{Z}}$ such that $w, 0 \models \phi$. Satisfiability at instant 0 is only conventional, because of shift invariance.

This semantics is simpler than the mono-infinite one, being symmetrical, since $\bullet$ has no condition $i \geq 0$ and $\mathcal{S}, \mathcal{S}_{\sim t}$ have no condition $i - k \geq 0$: $\bullet$ and $\mathcal{S}$ are indeed defined as the past duals of $\circ$ and $\mathcal{U}$, respectively. Clearly, all "primed" versions of the past operators are not needed now, e.g., the bi-infinite semantics of $\bullet'$ is the same as that of $\bullet$.

Finite automata can easily be adapted to work with bi-infinite words. While a standard Büchi automaton at instant 0 must start in one of the prescribed initial states, a bi-infinite automaton has an initial condition which is analogous to the Büchi acceptance condition, but towards the past: the automaton must traverse an initial state infinitely often in the past. Instant 0 has instead no special role since it is not used to represent the initial configuration of the system. We refer the interested reader to [Gire and Nivat 1991; Perrin and Pin 2004] for formal definitions and basic properties of automata on bi-infinite words.

4.1. Bi-infinite Encoding of PLTL

The goal of this section is to show that the techniques developed for Bounded Model Checking [Biere et al. 1999] may be adapted to deal with temporal logic specifications on bi-infinite time. Defining semantics for PLTL is more natural on a bi-infinite time domain, because in this case past and future have the same role and importance, and actually leads to a simpler encoding than in the mono-infinite case. The encodings and basic results of this section were originally presented in [Pradella et al. 2007; 2008a], and [Pradella et al. 2009] for the metric part.

A bounded bi-infinite temporal structure is shown in Figure 16. As the reader may notice, it is a natural extension of the mono-infinite structure: the loops are now at most two, one towards the future (as before), and a new one towards the past. 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 $\mathrm{LoopExists'}$. Like in the mono-infinite case, we will use in the description of the encoding the letter h to denote the value of i for which l_i, and analogously we will use h' for l'_i. In the picture the two loops do not overlap, since $h' < h$, but the encoding admits also the case in which h' is greater than h.

Analogously to the mono-infinite case, to perform BMC we represent symbolically the transition relation of the system M_S as a propositional formula. The k-times unrolling of the transition relation represents all the finite paths of length k:

$$|[M_S]|_k \longleftrightarrow I(S_0) \wedge \bigwedge_{0 \leq i < k} T(S_i, S_{i+1})$$

where T is a total transition relation predicate. Notice that, being the transition system M_S defined on a bi-infinite temporal domain, $I(S_0)$ is a condition on a special instant 0, which is propagated forward and backward by the unrolling of the transition relation. Although $I(S_0)$ may be used, analogously to an initial state predicate in mono-infinite time, to avoid incorrect states, such as unacceptable system configurations, S_0 is not the initial state of the system.

To define the bi-infinite encoding of PLTL, *Propositional constraints, Loop constraints, Eventuality constraints* and *Last state constraints* are defined, respectively, as in Tables (3), (1), (5) and (2) of Section 2.4. For the remaining constraints, it is necessary to introduce a special instant -1, which has a symmetric role of instant $k+1$: if there is a loop in the past at position i then at state S_{-1} and S_i formulae must have the same truth value. Hence, it is necessary to add a new variable $|[\phi]|_{-1}$ for each subformula ϕ of Φ. *Temporal subformulae constraints* include a slightly modified version of Table (4) for the future operators and a new set of constraints for the past operators (i.e., the back-loops):

Temporal subformulae constraints:

φ	$-1 \leq i \leq k$
$\circ\phi_1$	$\|[\circ\phi_1]\|_i \longleftrightarrow \|[\phi_1]\|_{i+1}$
$\phi_1 \mathcal{U} \phi_2$	$\|[\phi_1 \mathcal{U} \phi_2]\|_i \longleftrightarrow \|[\phi_2]\|_i \vee (\|[\phi_1]\|_i \wedge \|[\phi_1 \mathcal{U} \phi_2]\|_{i+1})$
$\phi_1 \mathcal{R} \phi_2$	$\|[\phi_1 \mathcal{R} \phi_2]\|_i \longleftrightarrow \|[\phi_2]\|_i \wedge (\|[\phi_1]\|_i \vee \|[\phi_1 \mathcal{R} \phi_2]\|_{i+1})$

(13)

φ	$0 \leq i \leq k+1$
$\bullet\phi_1$	$\|[\bullet\phi_1]\|_i \longleftrightarrow \|[\phi_1]\|_{i-1}$
$\phi_1 \mathcal{S} \phi_2$	$\|[\phi_1 \mathcal{S} \phi_2]\|_i \longleftrightarrow \|[\phi_2]\|_i \vee (\|[\phi_1]\|_i \wedge \|[\phi_1 \mathcal{S} \phi_2]\|_{i-1})$
$\phi_1 \mathcal{T} \phi_2$	$\|[\phi_1 \mathcal{T} \phi_2]\|_i \longleftrightarrow \|[\phi_2]\|_i \wedge (\|[\phi_1]\|_i \vee \|[\phi_1 \mathcal{T} \phi_2]\|_{i-1})$

(14)

Symmetrically to the mono-infinite encoding of eventualities, there are new loop selector variables l'_i, $0 \leq i \leq k$, to define the loop which goes towards the past, and the corresponding propositional letters $\mathrm{InLoop'}_i$, and $\mathrm{LoopExists'}$.

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

Loop constraints in the past:

Base	$\neg l'_k \wedge \neg \mathrm{InLoop'}_k$
$1 \leq i \leq k$	$(l'_i \rightarrow S_{i+1} = S_0) \wedge (\mathrm{InLoop'}_i \longleftrightarrow \mathrm{InLoop'}_{i+1} \vee l'_i)$ $(\mathrm{InLoop'}_{i+1} \rightarrow \neg l'_i) \wedge (\mathrm{LoopExists'} \longleftrightarrow \mathrm{InLoop'}_0)$

(15)

The above loop constraints state that the structure may have 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 many possible values of the loop selector variables.

Past Eventuality constraints, symmetrical to the future ones of Tables (5) and (6), are applied to operators $\mathcal{S}$ and $\mathcal{T}$, by introducing propositional letters $\langle\langle \blacklozenge \phi_2 \rangle\rangle$ and $\langle\langle \blacksquare \phi_2 \rangle\rangle$, as follows.

Past Eventuality constraints:

φ	Base
$\phi_1 \mathcal{S} \phi_2$	$\neg \langle\langle \blacklozenge \phi_2 \rangle\rangle_k \wedge (\mathrm{LoopExists'} \rightarrow (\ \|[\phi_1 \mathcal{S} \phi_2]\|_0 \rightarrow \langle\langle \blacklozenge \phi_2 \rangle\rangle_0\ ))$
$\phi_1 \mathcal{T} \phi_2$	$\langle\langle \blacksquare \phi_2 \rangle\rangle_k \wedge (\mathrm{LoopExists'} \rightarrow (\ \|[\phi_1 \mathcal{T} \phi_2]\|_0 \leftarrow \langle\langle \blacksquare \phi_2 \rangle\rangle_0\ ))$

(16)

$$\begin{array}{c|c} \varphi & 0 \leq i \leq k-1 \\ \hline \phi_1 \mathcal{S} \phi_2 & \langle\langle \blacklozenge \phi_2 \rangle\rangle_i \longleftrightarrow \langle\langle \blacklozenge \phi_2 \rangle\rangle_{i+1} \vee (\ \mathrm{InLoop'}_i \wedge |[\phi_2]|_i \ ) \\ \phi_1 \mathcal{T} \phi_2 & \langle\langle \blacksquare \phi_2 \rangle\rangle_i \longleftrightarrow \langle\langle \blacksquare \phi_2 \rangle\rangle_{i+1} \wedge (\ \neg \mathrm{InLoop'}_i \vee |[\phi_2]|_i \ ) \end{array} \tag{17}$$

Finally, symmetrically to the last state constraints of Table (2), we define first state constraints, to ensure that if there is no loop in the past then everything is false before instant 0, and that state S_{-1} and S_i are equivalent if there is a loop at instant i.

First state constraints:

$$\begin{array}{c|c} \text{Base} & \neg \mathrm{LoopExists'} \rightarrow \neg |[\phi]|_{-1} \\ \hline 0 \leq i \leq k-1 & l'_i \rightarrow (|[\phi]|_{-1} \longleftrightarrow |[\phi]|_i) \end{array} \tag{18}$$

4.2. Bi-infinite Encoding of metric temporal logic

This section presents the encoding of temporal operators for bi-infinite time, first focusing on past time operators and then on future ones.

The primitive past operator of metric temporal logic is Bounded Since, $\mathcal{S}_{\sim t}$, where $\sim \in \{\leq, =, \geq\}$ and t is a natural number. As it was the case for the future fragment of MPLTL, the usage of $\mathcal{S}_{\sim t}$ as the only primitive operator, while sufficient for what concerns expressive power, may lead to a very large encoding when constants t are large. In practice, $\blacklozenge_{\sim t}$ and its dual $\blacksquare_{\sim t}$ are much more common, and their encoding, rather than following their definition as derived from $\mathcal{S}_{\sim t}$, can be made much more compact.

The encoding is defined as follows. First, analogously and symmetrically to the *MF* variables, $\langle\langle \mathrm{MP}(\psi, j) \rangle\rangle$ are introduced for past operators with argument ψ, and represent the value of ψ *j* time units before the starting point of the past loop. E.g., if the past loop selector is at instant 7 (i.e. l'_7), then $\langle\langle \mathrm{MP}(\psi, 2) \rangle\rangle$ represents $|[\psi]|_{7-2}$.

Analogously to the mono-infinite case, we introduce the two following abbreviations $\lambda_i = k - i + 1$ and $\lambda'_i = i + 1$, denoting the length of the intervals $[i..k]$ and $[0..i]$, respectively.

The first constraints are introduced to define *MP* for every metric subformula of Φ: Table (19) is the symmetrical version, for the past operators, of Table (9), valid for future ones.

$$\begin{array}{c|c} \varphi & 0 \leq j \leq t-1 \\ \hline \blacklozenge_{=t} \phi, \ \blacksquare_{\leq t} \phi, \ \blacklozenge_{\leq t} \phi & \langle\langle \mathrm{MP}(\phi, j) \rangle\rangle \longleftrightarrow \bigvee_{i=0}^{k-1} l'_i \wedge |[\phi]|_{i - \mathrm{mod}(j, \lambda'_i)} \end{array} \tag{19}$$

Next we provide the translation of the past metric operators. Table (20) is the past counterpart of Table (10).

$$\begin{array}{c|c} \varphi & 0 \leq i \leq k+1 \\ \hline \blacklozenge_{=t} \phi & |[\blacklozenge_{=t} \phi]|_i \longleftrightarrow |[\phi]|_{i-t}, \text{ when } i \geq t \\ & |[\blacklozenge_{=t} \phi]|_i \longleftrightarrow \langle\langle \mathrm{MP}(\phi, t - \lambda'_i) \rangle\rangle, \text{ elsewhere} \\ \blacksquare_{\leq t} \phi & |[\blacksquare_{\leq t} \phi]|_i \longleftrightarrow \bigwedge_{j=0}^{\min(t, \lambda'_i - 1)} |[\phi]|_{i-j} \wedge \bigwedge_{j=\lambda'_i}^{t} \langle\langle \mathrm{MP}(\phi, j - \lambda'_i) \rangle\rangle \\ \blacklozenge_{\leq t} \phi & |[\blacklozenge_{\leq t} \phi]|_i \longleftrightarrow \bigvee_{j=0}^{\min(t, \lambda'_i - 1)} |[\phi]|_{i-j} \vee \bigvee_{j=\lambda'_i}^{t} \langle\langle \mathrm{MP}(\phi, j - \lambda'_i) \rangle\rangle \end{array} \tag{20}$$

Up to this point, the encoding was completely symmetrical to the future case. However, the presence of both past and future loops determines some change in the Loop Constraints; we illustrate them on the past operators.

Table (21) reports the Loop Constraints for the past operators $\blacklozenge_{=t} \phi$, $\blacklozenge_{\leq t} \phi$, and $\blacksquare_{\leq t} \phi$.

Table (22) displays the Loop Constraints for the future operators $\Diamond_{=t} \phi$, $\Box_{\leq t} \phi$, and $\Diamond_{\leq t} \phi$, which are perfectly symmetrical to the clauses of Table (21).

Past in Loop constraints:

$$
\begin{array}{c|c}
\varphi & 1 \le i \le k \\
\hline
\blacklozenge_{=t}\phi & l_i \rightarrow \left(\begin{array}{l} \bigwedge_{j=1}^{\min(t,\lambda_i'-1)} (|[\phi]|_{i-j} \longleftrightarrow |[\phi]|_{k-\mathrm{mod}(j-1,\lambda_i)}) \wedge \\ \bigwedge_{j=\lambda_i'}^{t} \left(\langle\langle \mathrm{MP}(\phi, j-\lambda_i') \rangle\rangle \longleftrightarrow |[\phi]|_{k-\mathrm{mod}(j-1,\lambda_i)} \right) \end{array} \right) \\
\blacklozenge_{\le t}\phi & \mathrm{InLoop}_i \rightarrow \left(|[\blacklozenge_{\le t}\phi]|_i \longleftrightarrow \left(\begin{array}{c} \bigvee_{j=0}^{\min(i,t)} (\mathrm{InLoop}_{i-j} \wedge |[\phi]|_{i-j}) \vee \\ \bigvee_{j=0}^{\min(k-i,t)} (\neg \mathrm{InLoop}_{\max(0,i-t+j)} \wedge |[\phi]|_{k-j}) \end{array} \right) \right) \\
\blacksquare_{\le t}\phi & \mathrm{InLoop}_i \rightarrow \left(|[\blacksquare_{\le t}\phi]|_i \longleftrightarrow \left(\begin{array}{c} \bigwedge_{j=0}^{\min(i,t)} (\neg \mathrm{InLoop}_{i-j} \vee |[\phi]|_{i-j}) \wedge \\ \bigwedge_{j=0}^{\min(k-i,t)} (\mathrm{InLoop}_{\max(0,i-t+j)} \vee |[\phi]|_{k-j}) \end{array} \right) \right)
\end{array}
\tag{21}
$$

Future in Loop constraints:

$$
\begin{array}{c|c}
\varphi & 0 \le i \le k-1 \\
\hline
\diamondsuit_{=t}\phi & l_i' \rightarrow \left(\begin{array}{l} \bigwedge_{j=1}^{\min(t,\lambda_i-1)} (|[\phi]|_{i+j} \longleftrightarrow |[\phi]|_{\mathrm{mod}(j-1,\lambda_i')}) \wedge \\ \bigwedge_{j=\lambda_i}^{t} \left(\langle\langle \mathrm{MF}(\phi, j-\lambda_i) \rangle\rangle \longleftrightarrow |[\phi]|_{\mathrm{mod}(j-1,\lambda_i')} \right) \end{array} \right) \\
\diamondsuit_{\le t}\phi & \mathrm{InLoop}_i' \rightarrow \left(|[\diamondsuit_{\le t}\phi]|_i \longleftrightarrow \left(\begin{array}{c} \bigvee_{j=0}^{\min(k-i,t)} (\mathrm{InLoop}_{i+j}' \wedge |[\phi]|_{i+j}) \vee \\ \bigvee_{j=0}^{\min(i,t)} (\neg \mathrm{InLoop}_{\min(k,i+t-j)}' \wedge |[\phi]|_j) \end{array} \right) \right) \\
\square_{\le t}\phi & \mathrm{InLoop}_i' \rightarrow \left(|[\square_{\le t}\phi]|_i \longleftrightarrow \left(\begin{array}{c} \bigwedge_{j=0}^{\min(k-i,t)} (\neg \mathrm{InLoop}_{i+j}' \vee |[\phi]|_{i+j}) \wedge \\ \bigwedge_{j=0}^{\min(i,t)} (\mathrm{InLoop}_{\min(k,i+t-j)}' \vee |[\phi]|_j) \end{array} \right) \right)
\end{array}
\tag{22}
$$

LEMMA 4.4. Correctness of the bi-infinite encoding for past metric operators. *The bi-infinite encodings of the metric temporal logic for the past operators $\blacklozenge_{=t}, \blacklozenge_{\le t}, \blacksquare_{\le t}\phi$ (Tables from (19) to (22)) are consistent with the semantics of metric PLTL as defined in Section 4.*

PROOF. We focus on the clauses of Table (21) because Tables (19) and (20) are the symmetric counterpart of Tables (9) and (10), and the clauses of Table (22) are analogous, for the future operators, to those of Table (21).

First, let us notice that the two operators $\blacklozenge_{=t}\phi$ and $\blacksquare_{=t}\phi$ now coincide, because of the presence of the past loop, which makes it unnecessary to assign a conventional value to formulae at time points preceding instant 0, hence the two clauses (ix) and (x) of Section 3.2 are replaced by the first clause of Table (21).

The top line of this clause states that, for every time point (denoted as $i-j$) preceding the current time i and back to a distance t or to the initial time point 0, ϕ holds if and only if it also holds at a corresponding instant inside the future loop (denoted by the usual notation $k-\mathrm{mod}(j-1,\lambda_i)$). The bottom line of the clause for $\blacklozenge_{=t}\phi$ concerns the case in which $t > i$, hence the formula refers to a time point preceding the first time instant of the time domain. Due to the past loop, the value of the subformula ϕ at that point is encoded as $\langle\langle \mathrm{MP}(\phi, j-\lambda_i') \rangle\rangle$, where $j-\lambda_i'$ is the distance between the initial time 0 and the referenced time point $i-j$; the value of subformula ϕ inside the past loop is equal to the value of ϕ at the corresponding position inside the future loop, which is, as usual, denoted as $k-\mathrm{mod}(j-1,\lambda_i)$.

Next, concerning operators $\blacklozenge_{\le t}\phi$ and $\blacksquare_{\le t}\phi$, we notice that the second and third clauses of Table (21), are identical to clauses (xi) and (xii) of Section 3.2: all these clauses do not refer to the value of subformula ϕ before the initial time 0, because their purpose is to encode the value of ϕ at the time points that belong to the future loop. □

Figure 17 illustrates the application of the bottom line of the first clause of Table (21) to the formula $\blacklozenge_{=11}\phi$ with reference to a case where $k=14$, $i=h=8$, and $h'=5$, so that $\lambda_i=7$ and $\lambda_i'=9$

The presence of bi-infinite time doubles the loop constraints but does not introduce any additional complexity. In fact, thanks to the complete symmetry of bi-infinite time, the constraints of Table

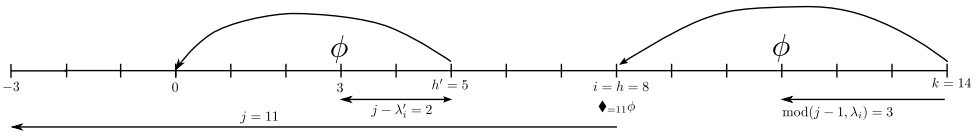

Fig. 17. An illustration of the loop constraint for $\blacklozenge_{=11}\phi$.

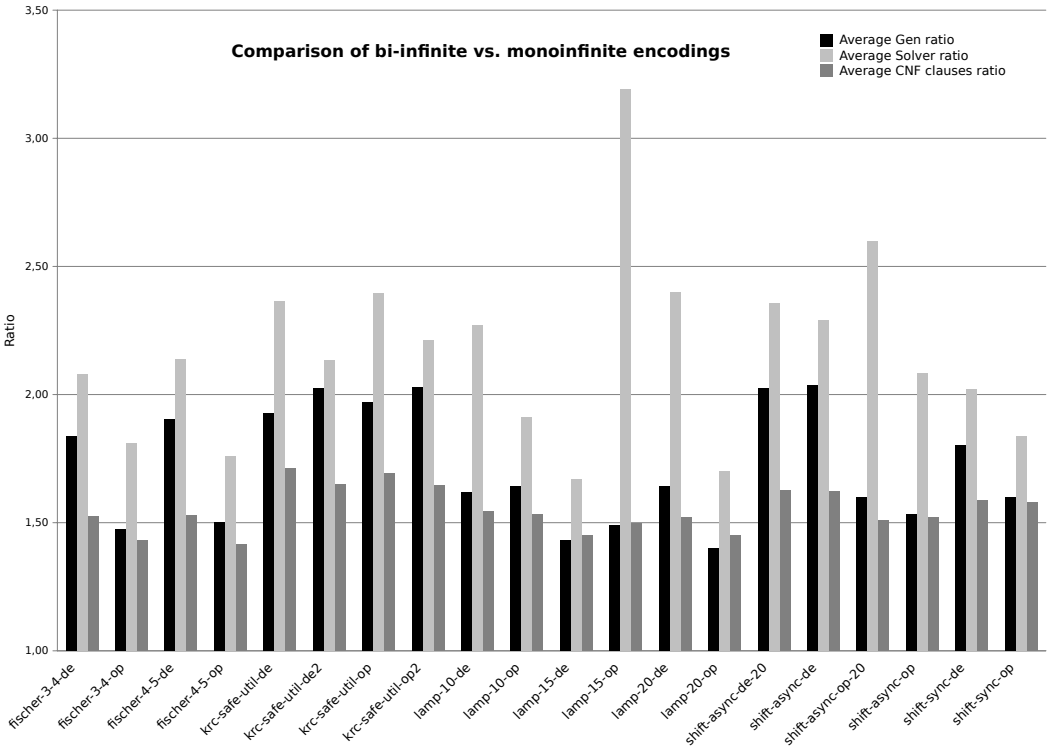

Fig. 18. Summary of experimental comparisons for bi-infinite vs. mono-infinite encodings. The comparison is expressed as the average ratio of time (for Gen and Sat) or Clauses (for CNF clauses) obtained in the bi-infinite case and in the monoinfinite case of the same specification. The higher the ratio the lower the performance of a bi-infinite verification with respect to the mono-infinite one.

(21) and (22) are even more compact than the corresponding *Past in Loop Constraints* presented in Section 3.2 for the mono-infinite case.

4.3. Experimental results with the bi-infinite encoding

The results of the experiments carried out to compare the tool performance on the bi-infinite vs. mono-infinite time domain are shown in Figure 18, which, as in previous comparisons, shows figures for Generation time, Solver time, and number of generated CNF clauses for all case studies except the Real-time allocator example (where the mono-infinite case is not meaningful). The values concerning generation (generation time and number of generated CNF clauses) are quite uniform, with the ratio bi-infinite/mono-infinite never rising significantly above the value of 2. Such value clearly derives from the fact of having to encode the presence of two loops in the time structure. The values for the ratio of the SAT solver times are less uniform, since they are more dependent on the intrinsic timing features of each single case study, but they are around 2 and all below 3.5.

Based on these results we can consider the overhead deriving by the adoption of bi-infinite time structure for Bounded Satisfiability Checking to be acceptable in practice; in addition, bi-infinite time allows us to avoid the subtle semantic problems of combining past operators with a mono-infinite semantics. However, this advantage is less clear with the much more common BMC approach to specification and verification: in most cases in the literature, the abstract model of the system is operational and some simple PLTL formulae only state a few general system properties to be verified. Operational models are typically mono-infinite, with an initial state predicate ruling out certain "bad" states, since this is the way operational models have always been conceived, and the main advantage of bi-infinite time (avoiding very frequent and subtle semantic problems in PLTL formulae) is at least partially lost.

5. A COMPARISON OF BSC AND BMC: DESCRIPTIVE VS. OPERATIONAL MODELS

In the present section, we explore in more depth analogies and differences between bounded *satisfiability* checking and (bounded) *model* checking, in order to assess the merits of the two approaches, also by means of a detailed performance comparison. We observe that, from a temporal logic specification viewpoint, a state-transition system M_S differs from a descriptive model Φ_S of the same system. As already outlined, in our approach Φ_S is a MPLTL formula, without particular restrictions on the temporal operators occurring in the formula itself. Hence, Φ_S may refer to any instant in the past or in the future. Instead, M_S directly relates pairs of states occurring in two consecutive time instants, (the current state and the next one) with no possible reference to states in other time points, neither in the past nor in the future. The actual encoding of the state-transition system M_S is the one given in Section 2.4. On the other hand, we note that M_S may be expressed as a temporal logic formula, of the very special form

$$\mathcal{A}lw\left(\bigwedge_{1\leq i\leq h} C_i \rightarrow \circ N_i\right)$$

where $h > 1$ is the number of transitions and C_i, N_i are Boolean formulae. For simplicity and to avoid the introduction of another notation, in the rest of the paper we will use this representation and often leave $\mathcal{A}lw$ and $\bigwedge$ implicit, writing only the clauses.

M_S, though it can be made more abstract through nondeterminism, is more *constrained* than Φ_S, because it only relates states in two consecutive time instants. As a consequence of these stronger syntactic constraints, M_S is also bound to be more *detailed*: in order to characterize the behavior of the modeled system by means of a set of transitions that refer only to the present and next time instants, it must convey, for each time instant, a larger amount of information, adequate to completely determine the subsequent evolution of the modeled system.

Therefore, M_S typically (but not necessarily) has a larger alphabet than a descriptive model, with the additional elements of the alphabet consisting of *internal state variables* needed to implement the properties that are stated, in an abstract style, by the descriptive model.

As a simplest example, consider the Synchronous Shift Register of Example 2.1 (where, at each clock tick, an input bit is shifted one position to the right), and assume that the observable interface of this system consists only of the input and output bits at the two ends of the register. A descriptive model of this system is provided by its concise MPLTL specification (where the outermost operator is $\mathcal{A}lw$, since bi-infinite time is considered):

$$\mathcal{A}lw(in \longleftrightarrow \diamondsuit_{=d} out).$$

To provide a description of the synchronous shift register in terms of a state-transition system which, based on the current state, determines the next one, the alphabet of the above descriptive model, consisting of the predicate letters in and out, would not suffice, for the obvious reason that the current value of the in predicate does not determine the next value of out (in fact, it is immaterial for that purpose). The next value of the out predicate is determined by whether or not the in predicate held d time units before. Therefore, to convey in the current state the information which suffices to

determine the next state (and the following ones), an additional unary predicate is needed, call it shr. For each i, $0 \leq i \leq d$, $shr(i)$ "encodes" the fact that an in event occurred i time units before. Then the out predicate is characterized by the following formula

$$\mathcal{A}lw(out \longleftrightarrow shr(d))$$

and the new predicate must be in its turn specified by a suitable axiom, also to be provided in a present-to-next-state style, as follows:

$$\mathcal{A}lw \left(\begin{array}{c} (shr(0) \longleftrightarrow in) \\ \wedge \\ \forall x(0 \leq x \leq d-1 \rightarrow (shr(x) \longleftrightarrow \circ(shr(x+1)))) \end{array} \right)$$

Because of its simplicity and uniformity, the operational model can be provided in a variety of semantically equivalent notations, like tabular or graphical representations of the transition relation as in [Gargantini and Morzenti 2001], or by means of programming language-like notations as in Spin [Holzmann 1997] or NuSMV [Cimatti et al. 2002]. In the present work, which is focused on bounded satisfiability checking, where the model is expressed in terms of temporal logic formulae, we adopt a representation of the operational model as a set of temporal logic formulae that are constrained to use (beside the unique outermost $\mathcal{A}lw$ operator) only the next state temporal operator '$\circ$': this makes the operational model directly comparable with the descriptive one, and facilitates an immediate encoding for analysis and verification through logic-oriented tools such as SAT-solvers.

In summary, we call *bounded satisfiability checking* the verification method based on bounded model checking when the model is a MPLTL formula; if the formula includes only unnested applications of the $\circ$ next-time then we have a state-based, implementation oriented *operational model*; otherwise, i.e., if other MPLTL operators are allowed, we have an abstract, property-based, specification-oriented *descriptive model*.

5.1. Operational model of the Timer Reset Lamp

The logic characterization of the timer-reset-lamp provided in Section 2.5 constitutes a descriptive model of it. We now show how an operational model can be provided for the this system. As mentioned above, the idea is to define, for each instant, the next system state based on the current state and, possibly, of the stimuli coming, still at the current time, from the environment. A brief reflection shows that the current state of the timer-reset-lamp is *not* completely characterized by the value of predicate letter L; e.g., if at a given time we know that the lamp is on (predicate letter L holds) and that no button is pressed, this does *not* allow us to conclude that the lamp will still be on at the next time instant, since this depends on the time that has elapsed from the last press action on the *ON* button. To model explicitly this component of the state it is therefore necessary to introduce a further element in the alphabet of the model: a counter variable ranging over the interval $[0 \,..\, \Delta]$ to store exactly this information. With this addition the definition of the operational model becomes an easy exercise, yielding the following clauses.

The pressure of the *ON* button sets the counter to Δ.

(O1) $\mathit{ON} \rightarrow \circ(\mathit{count} = \Delta)$

The pressure of the *OFF* counter resets the counter to 0.

(O2) $\mathit{OFF} \rightarrow \circ(\mathit{count} = 0)$

When the counter has a positive value and no external event occurs, it is decremented.

(O3) $\forall x(\mathit{count} = x \wedge x > 0 \wedge \neg \mathit{ON} \wedge \neg \mathit{OFF} \;\; \rightarrow \;\; \circ(\mathit{count} = x - 1))$

If the counter is null and the *ON* button is not pressed, it remains null.

$(\mathrm{O4})$ $\mathit{count} = 0 \wedge \neg \mathit{ON} \rightarrow \circ(\mathit{count} = 0)$

The lamp is on if and only if the counter is positive.

$(\mathrm{O5})$ $L \longleftrightarrow \mathit{count} > 0$

The two buttons cannot be pressed simultaneously.

$(\mathrm{O6})$ $\neg(\mathit{ON} \wedge \mathit{OFF})$

The operational model of the timer-reset-lamp system simply consists of the conjunction of the six clauses above (O1-O6), with the usual outermos $\mathcal{A}lw$ operator

(OM) $\mathcal{A}lw(\bigwedge_{1 \leq i \leq 6} \mathrm{O}i)$

5.2. An example of descriptive and operational models with the same alphabet

It is interesting to note that in some special case the descriptive and the operational models may have the same alphabet. This typically happens in systems, being relatively simple from the temporal point of view, in which the consequences of the occurrence of any event follow immediately their cause, so that in a state-based operational model there is no need of additional variables to carry that information to further future time instants.

As an instance, consider the interlocking component of the KRC example: the purpose of this device is to prevent two trains from being simultaneously present in the critical region R. To reach this goal, the interlocking sets the light to red immediately after a train entrance in region R, and sets it to green immediately after a train exit with no simultaneous train entrance.

The descriptive model for the Interlocking subsystem consists of the following formula:

$$\mathcal{A}lw\,(red \longleftrightarrow \bullet(\neg(ExitI \wedge \neg EnterR)\ \mathcal{S}\ EnterR))$$

The operational model can be provided on the basis of the same alphabet with no additional specification item, due to the fact that the value of the light at the next time always depends only on its current value and on the events (train entrance or exit of the critical region) occurring at the present time. This is demonstrated by the following operational model.

$$\begin{aligned} red \wedge \neg ExitI &\rightarrow \circ red \\ red \wedge ExitI \wedge \neg EnterR &\rightarrow \circ \neg red \\ \neg red \wedge EnterR &\rightarrow \circ red \\ \neg red \wedge \neg EnterR &\rightarrow \circ \neg red \end{aligned}$$

5.3. Performance comparison

When, for a given system under development, one has produced both a descriptive model that abstractly specifies the requirements and an operational one that characterizes, in a more implementation-oriented way, the same set of behaviors, the question arises whether it would be preferable to analyze some further properties with reference to the descriptive model or to the operational one[2].

To investigate this issue, we defined the operational model of all the case studies presented in Section 2.5, of which we already provided a descriptive model. Then we carried out, on the operational model, the analysis of the same properties. The results of the performance comparison between descriptive and operational models are reported in Figure 19. The encoding is the metric, mono-infinite one for all examples, with the exception of the allocator where the metric, bi-infinite

[2]The issue of checking whether an operational model constitutes a correct implementation of a given descriptive model will be dealt with in the next section.

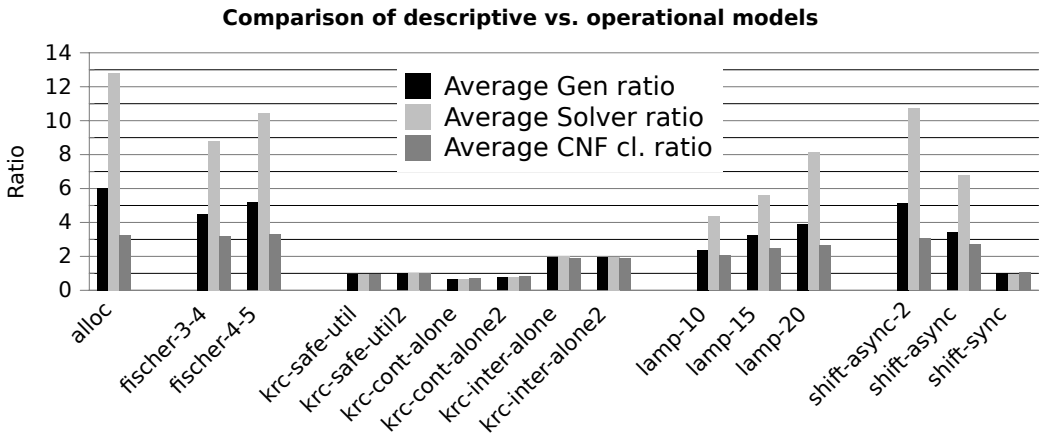

Fig. 19. Summary of experimental data for comparison of descriptive vs. operational models. The comparison is expressed as the average ratio of time (for Gen and Sat) or Clauses (for CNF clauses) obtained in the descriptive case and in the operational case of the same specification. The higher the ratio the lower the performance of a descriptive model respect to an operational one.

encoding was used. Again, the ratio of each verification case between descriptive and operational models was computed, and then averaged over bounds 30, 60 and 90 and over the various properties verified for each model. Two more cases have been added for the KRC, namely *krc-inter-alone* and *krc-cont-alone* (for both versions of KRC), representing the verification of, respectively, the Interlocking and the Controller components of KRC.

As it can be noticed, the ratio varies considerably from case to case. The most significant and uniformly valid result is that the time needed to analyze a certain property is always directly (but not linearly) related with the overall size of the Boolean formula fed to the SAT solver, resulting from the translation of both the model and the property.

In other words, when analyzing an operational model a performance improvement, with respect to the descriptive model, can be expected if the Boolean encoding of the operational model is more compact than the encoding of the corresponding descriptive one. This is very often, but not always, the case, as discussed next. By carefully considering the adopted case studies, we have identified three significant, recurring patterns in the features of the models and their influence on the encoding size.

The first pattern is that of a system that exhibits a simple temporal property, like a time-out or a delay between events, which in the descriptive model is typically expressed by means of a (sub)formula like $\Diamond_{\sim c}\phi$ or $\blacklozenge_{\sim c}\phi$. In the corresponding operational model the feature is easily rendered by means of a numerical variable that represents a counter, and very simple axioms that specify the increase or decrease of the counter variable from one time instant to the next. The representative example of this category of systems is the Timer Reset Lamp, whose behavior depends essentially on the expiration of a time-out (cases *lamp-10*, *lamp-15*, and *lamp-20* in Figure 19). Since the encoding of a numeric counter variable and of axioms specifying its increase or decrease is more compact than that of the $\Diamond_{\sim c}\phi$ and $\blacklozenge_{\sim c}\phi$ operators, the analysis of operational models exhibiting such kind of features can be significantly more efficient when performed on the operational model than on the descriptive one.

A second, significant category of systems is the one, represented in our examples by the interlocking subsystem of the Kernel Railway Crossing (cases *krc-inter-alone* and *krc-inter-alone-2* in Figure 19), in which the alphabet of the formulae that constitute the operational model is exactly the same as that of the descriptive model. This also results in an advantage for the analysis carried out on the operational model, because for such systems there is no actual metric property to be charac-

terized (only qualitative time relations, like precedence or immediate effect among events), hence the operational model is composed of formulae that are simpler in their syntactic structure and do not include any actual metric time operator, whose presence in the descriptive model instead makes the encoding more complex.

A third pattern, which instead is less favorable for the operational models, is the case where some complex temporal property can be expressed, in the descriptive model, in some very concise way using a few specification items and suitable metric temporal operators, while in the operational model it is rendered by means of quite a few additional specification items encoding a significant amount of temporal constraints. This is, for instance, the case of the *synchronous* shift register (case *shift-sync* in Figure 19), which, despite the conciseness of the descriptive model (consisting of the single, brief formula $\mathcal{A}lw(in \longleftrightarrow \Diamond_{=d} out)$) depicts a system that can *remember* the time of occurrence of all the *in* events in a time window of length d. In the operational version of the model the representation of this feature requires the introduction, as an additional specification item, of a predicate with a numeric argument, and of a set of axioms, with universal quantifiers, that account for its change in time. In cases like this the Boolean SAT formula resulting from the encoding of the operational model may turn out to be of comparable, or even larger size than that of the descriptive model, and the time needed for the analysis is increased accordingly. Another example in this category is the Controller component of the KRC case study (cases *krc-cont-alone* and *krc-cont-alone-2* in Figure 19), whose descriptive and operational models are both reported in the Appendix (Section I.1)

We should also point out, however, that in most non-trivial systems all the above features are often simultaneously present, and their combined effect may possibly vary depending on the numerical value of the time constants or the structure of the overall formula, so that a strong performance gain for one kind of model over the other is usually present only in the simplest systems (like the few representative ones discusses above) or in systems in which a specific type of property has a dominating influence.

In summary, the experimental results of this paper show that BSC can often be reasonably efficient, although it is rarely competitive with BMC; in some cases, however, it may be much slower. In general, since BSC allows using very concise models, there is a trade-off between succinctness and abstraction on the one hand and efficiency on the other one, to be decided on a case by case basis.

5.4. Bounded Refinement Checking

All the above illustrated example systems, of which we provided both the descriptive and the operational model, share the following features:

(1) The descriptive model is more abstract and concise than the operational one;
(2) The alphabet of the formulae that constitute the operational model is a superset (most often a proper superset) of the alphabet of the descriptive model;
(3) When present, the additional elements of the alphabet in the operational model incorporate in their value the information, resulting from the previous computation, that suffices to determine the next states;
(4) The properties that hold valid for the descriptive model also hold for the operational one.

These features are typical of what, in many system development methods, is called a *refinement step*, where a version of the system under development that is more abstract, close to the requirements specification, is substituted, possibly through systematic transformation steps, by another version, more concrete and close to implementation, which is bound to satisfy the same properties as the first one.

In our formal setting features (1)-(3) above are an immediate consequence of having a descriptive model and a corresponding operational one. Feature (4) constitutes a form of *semantic monotony*: the operational model can replace the descriptive one because it satisfies the same properties. This is an essential feature in our notion of refinement: we say that an operational model OM *correctly*

implements a descriptive model DM if OM ensures all the properties that are guaranteed by DM, i.e., for every property ϕ built on the alphabet of DM (hence expressible also in the context of OM, whose alphabet is a superset of that of DM), if $DM \rightarrow \phi$ holds then $OM \rightarrow \phi$ also holds.

A sufficient condition for the correct implementation relation to hold between an operational model OM and a descriptive model DM is the following implication,

$$OM \rightarrow DM$$

as ensured by the tautological form $(OM \rightarrow DM) \wedge (DM \rightarrow \phi) \rightarrow (OM \rightarrow \phi)$.

In the literature the correct implementation relation between an abstract model and a corresponding concrete one is very often formalized by an implication [Abrial 1996] also in approaches that are quite far from our own. In our setting the notion of refinement is especially characterized by the transition from the "abstract" to the "concrete" by renouncing the use of all the operators of temporal logic except for the next-state operator $\circ$, possibly introducing in the specification alphabet further items to encode the additional information incorporated into the current state.

We also notice that the formula $OM \rightarrow DM$, when interpreted as a relation between the sets of possible executions of the modeled systems, asserts that the set of traces of the operational model is a subset of that of the descriptive one. This interpretation also corresponds to the customary, intuitive interpretation of the refinement operation, where the implementation does not introduce any "surprise" in the form of new, unexpected behaviors, hence it satisfies the same properties as the abstract model.

The property of correct implementation, as expressed by $OM \rightarrow DM$, can itself be analyzed by the bounded satisfiability checker, with the usual incompleteness caveats. For instance, in the simplest case of the previously discussed Synchronous Shift Register example, the formula $OM \rightarrow DM$ is the following.

$$\mathcal{A}lw\left(\begin{array}{c}(out \longleftrightarrow shr(d))) \wedge \\ (shr(0) \longleftrightarrow in) \wedge \\ \forall x(0 \leq x \leq d-1 \rightarrow (shr(x) \longleftrightarrow \circ(shr(x+1))))\end{array}\right) \rightarrow \mathcal{A}lw(in \longleftrightarrow \diamondsuit_{=d} out)$$

The check of correct refinement using BSC is called here Bounded Refinement Checking (BRC). We have carried out BRC for the descriptive and operational model of all the case studies introduced in the previous sections. We report the results in Figure 20 (the suffix -ref stands for BRC based on the considered descriptive and operational models). As it can be seen, BRC is considerably slower than the verification of a single property of a descriptive or operational model.

From a *methodological* viewpoint, BRC, although an incomplete technique (being based on BSC) can still provide useful insights, in particular when the check fails, i.e., the operational model does not constitute a correct refinement of the descriptive one, thanks to the counterexample that is provided by the tool. If the implication $OM \rightarrow DM$ does not hold then the counterexample constitutes an admissible behavior of the operational model that is not a legal behavior of the descriptive model. The appropriate interpretation of this result by the practitioner who is using the tool depends on whether the counterexample corresponds to a behavior that is intuitively acceptable for the modeled system or not. If the behavior encoded into the counterexample is acceptable this might be the sign that the descriptive model is too restrictive, in that it rules out some behavior that is in fact compliant with the actual system requirements. In this case the descriptive model could be modified by adding some subformulae (typically in disjunction with those that constitute the descriptive model) that enlarge the set of possible behaviors, or by removing some subformulae included in the model that unduly constrain its set of admissible behaviors. Somewhat symmetrically, if the counterexample provided by the satisfiability checker corresponds to an unacceptable system behavior, this is a symptom of an error in the operational model, which must be adjusted by restricting the set of its behaviors by means of suitable additional constraints, typically to be conjoined with the formulae which constitute the existing operational model, or by means of removal of some clauses corresponding to inappropriate behaviors. A very useful guideline for the practitioner comes from

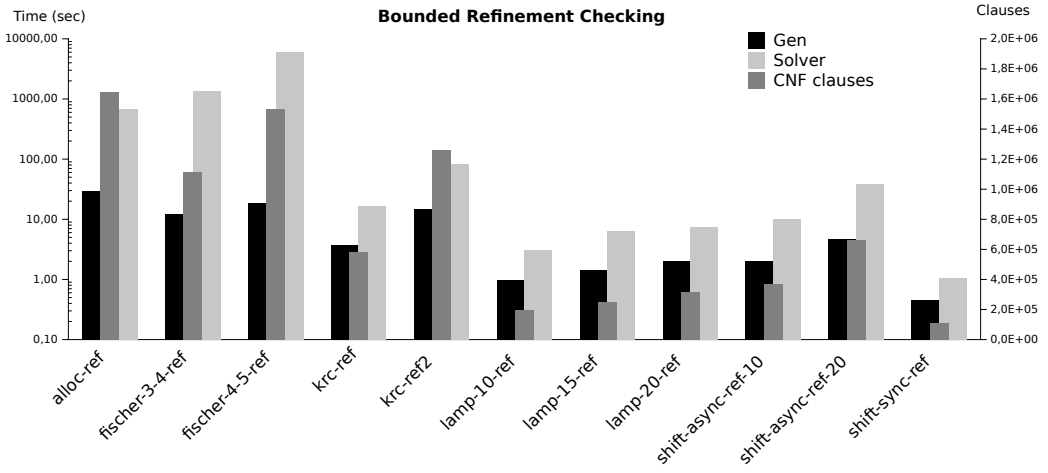

Fig. 20. Summary of experimental data for Bounded Refinement Checking. The vertical axis at the left measures time seconds in a logarithmic scale, for Generation and Solver; the vertical axis on the right measures the number of CNF clauses.

the remark, deriving from our own experience in analyzing descriptive and operational models of the same system. When writing an operational model for a system for which a descriptive model has already been provided, one is often induced to formalize, by means of suitable axioms, transitions among classes of states, without stating precisely the set configurations that are admissible, or possibly disregarding the system initial state, if there is any. This results in a typical modeling error, because the operational model will admit a set of behaviors that is larger than the one actually intended, therefore resulting in a failure when attempting to check the correct implementation relation $OM \rightarrow DM$. In this case, a careful inspection of the counterexample is very useful to identify the salient features of the inappropriate behavior of the operational model, and therefore to correct the exposed defect.

5.5. Modularization

When developing a complex, reactive, (time-)critical system, the designers must not only state in an explicit, abstract, and precise way the user/application requirements, but also adopt a modular approach for dealing adequately with the complexity of the system under development and provide a model of the environment surrounding the computer-based system under development that incorporates environment assumptions to be taken into account in the validation and verification of the system requirements.

For these reasons the modular structure of the overall system model will include a set *ENV* of components representing the environment, a set *CUD* (for Component Under Development) corresponding to the parts that will be actually implemented as computer based applications, and a set *REQ* of components specifying the requirements that the components under development must satisfy in their interaction with the environment.

The modular structure of the system model can be exploited in various ways to improve the efficiency of its analysis and verification: in the following we discuss an example centered on the bounded refinement checking of the KRC example.

The KRC example includes, for what concerns the environment, a description of the movement of the trains and of the gate, while the gate controller and interlocking are the object of the development process, and the safety and utility properties constitute the overall requirements. Therefore we model the whole KRC system as composed of five modules, as shown in Figure 21, where $Train\&Track$ and $Gate$ modules represent the environment and are labeled by *ENV*, the

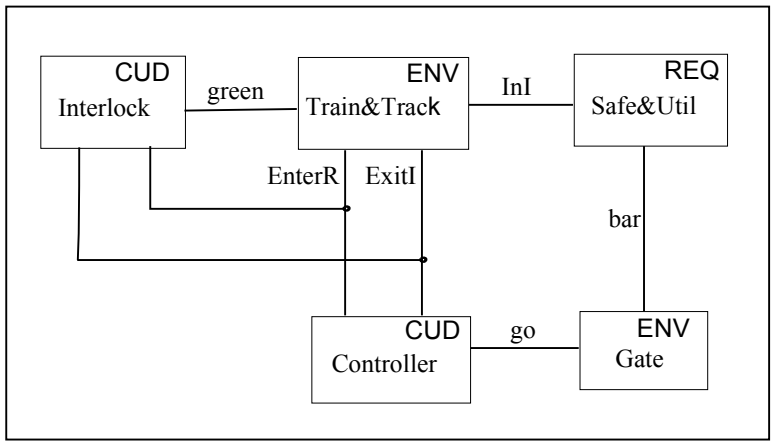

Fig. 21. Modular structure of the Kernel Railroad Crossing.

$Controller$ and $Interlock$ modules are labeled *CUD*, and the $Safe\&Util$ module is labeled by *REQ*. In the figure the lines connecting the modules represent specification items that are shared among the connected modules: for instance $EnterR$ and $ExitI$ are Boolean variables, respectively representing the event of a train entering region R or exiting region I, that appear in the logical axioms included in the three modules: $Train\&Track$, $Controller$, and $Interlock$.

In the Appendix (Section I.1) the MPLTL formulae modeling the KRC system are grouped into the five modules shown in Figure 21.

In the initial version of the system modular structure the *CUD* will be naturally provided in a descriptive version that includes its requirements specification, hence we will indicate it as CUD_{DE}. The *ENV* and *REQ* modules, which are not the object of the development process, will be provided in the form that the designer considers more suitable, and will not be refined, hence in the following they will not be annotated by any *DE* nor *OP* subscript.

Therefore, in a logic-based setting, the ability of the overall system to satisfy its requirements is stated as follows

$$ENV \wedge CUD_{DE} \rightarrow REQ \tag{23}$$

Formula (23) is analyzed, before any further development takes place, to validate the system requirements and verify that the features specified, in an abstract way, by CUD_{DE} ensure their satisfaction.

Then the descriptive model CUD_{DE} is refined into an operational model CUD_{OP}, which incorporates design decisions and implementation choices, suitable to accomplish the properties stated in the CUD_{DE} model. The correctness of the new system version, incorporating the refined version of *CUD*, will be stated as follows.

$$ENV \wedge CUD_{OP} \rightarrow REQ \tag{24}$$

Formula (24) can similarly be checked to verify that the implementation choices incorporated into the operational model are adequate to satisfy the overall system requirements. Checking formula (24) will be easier than checking formula (23), because of the presence of the operational model CUD_{OP} taking the place of the descriptive model CUD_{DE}. However, since the only part that has changed in the system model is the *CUD* component, a further substantial reduction of the effort required for checking (24) can be obtained, under the assumption that (23) holds, by simply showing that the new version CUD_{OP} is a correct refinement of the original, abstract version CUD_{DE}. In general, the correctness of the refinement might depend on some features of the environment in

which it operates, hence it would be stated by means of the following formula.

$$ENV \rightarrow (CUD_{OP} \rightarrow CUD_{DE}) \tag{25}$$

In many significant cases, however, CUD_{OP} correctly implements CUD_{DE} in all possible circumstances, hence the following, stronger property can be checked.

$$CUD_{OP} \rightarrow CUD_{DE} \tag{26}$$

In other terms, if in a system composed of several modules one or more of these modules are refined, a sufficient condition for the correctness of the refined system is provided by the correctness of the refinement of the implemented modules, provided that the abstract version of the entire system has been previously verified.

Checking Formula (26) is in practice much more convenient than checking (24), for two reasons. First, Formula (26) is much more compact than (24), hence its automatic verification is likely to require a significantly slighter effort. Second, the Component Under Development, that we have globally denoted as *CUD*, may be in its turn composed of several parts that may be developed and analyzed separately, adopting an incremental approach. This would bring several advantages in terms of better manageability of the development process: the validation and verification activity, which could focus on individual components of the model, and the overall effort would be reduced since the analysis would be conducted in several, relatively simple incremental steps focusing only on the parts to be effectively developed, leaving the parts representing the environment untouched.

In the Kernel Railway Crossing case study the $Controller$ and the $Interlock$ modules correspond to the *CUD*): we call $Controller_{DE}$ and $Interlock_{DE}$ their descriptive models while $Controller_{OP}$ and $Interlock_{OP}$ denote the corresponding operational models (all models are reported in the Appendix I.1). Therefore the correctness of the refined version of the KRC (24) can be verified by checking the two properties of type (26)

$$Controller_{OP} \rightarrow Controller_{DE} \tag{27}$$

and

$$Interlock_{OP} \rightarrow Interlock_{DE} \tag{28}$$

Figure 22 depicts graphically the comparison between modular and non-modular refinement using Bounded Refinement Checking. BRC for the two separately refined components (cases *modular-krc* and *modular-krc-2*) requires an effort that is one or two orders of magnitude less than that for the entire system (cases *non-modular-krc* and *non-modular-krc-2*, respectively).

6. RELATED WORK

Since the seminal paper [Biere et al. 1999] on SAT-based model checking, several bounded model checkers were introduced. To name some of them, we cite NuSMV [Cimatti et al. 2002], SAL [de Moura et al. 2004], CBMC [Clarke et al. 2004], EBMC [Clarke et al. 2005], and BAT [Manolios et al. 2007].

NuSMV is probably the most renowned, as it was the first, freely available bounded model checking tool, and many new encodings were originally tested in modified versions of NuSMV (e.g. those collected in [Biere et al. 2006]). Bounded model checking was introduced as an alternative, SAT-based approach to classical symbolic model checking. Indeed, NuSMV is a re-implementation of the traditional symbolic model checker SVM, and also provides its original BDD-based approach.

SRI's SAL (Symbolic Analysis Laboratory) is in some respects comparable with NuSMV, as it supports both BDD-based and SAT-based model checking. Its input language is not dissimilar from that of NuSMV, and is based on concurrent operational modules. SAL supports also infinite-state model checking through SMT (Satisfiability Modulo Theory) - the default tool for SAT or SMT checking is Yices, also available from SRI.

CBMC is a bounded model checker for C and C++ programs that can be used to verify array bounds, pointer safety, exceptions and user-specified assertions. Moreover, it can check consistency

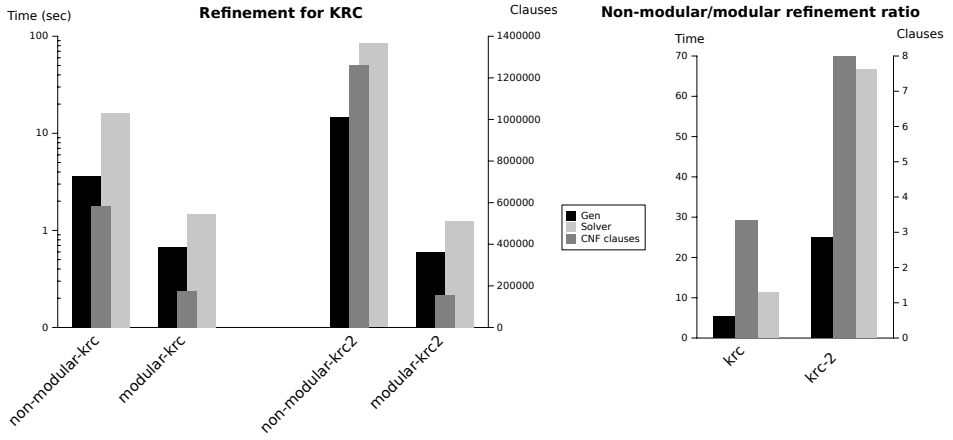

Fig. 22. Summary of experimental data for Bounded Refinement Checking of KRC. For the leftmost picture, the vertical axis on the left measures time seconds in a logarithmic scale, for Generation and Solver, while the vertical axis on the right measures the number of CNF clauses; In the rightmost picture, the ratios for the time (left axis) and the size (right axis) are reported in a linear scale.

of C programs with specifications written in hardware-oriented languages, like Verilog. EBMC, from the same research group, is geared towards hardware design, and supports various input notations: Netlists, Verilog, and SMV. Both CBMC and EBMC can use either SAT solvers or SMT solvers for verification.

BAT (Bit-level Analysis Tool) is a very efficient bounded model checker that supports quantifier-free formulae over the extensional theory of fixed-size bit-vectors and fixed-size bit-vector arrays. Properties can be specified using a future-only fragment of LTL.

To the best of our knowledge, all the bounded model checkers considered in the literature support, as in classical model checking, an *operational* model, offering either a more hardware-oriented language, as e.g. in NuSMV and BAT, or a more software-oriented language, as in CMBC, while they all use classical temporal logic languages, usually PLTL or CTL, only for expressing properties of the model.

As stated in the introduction, our approach has a different scope. We started working on automatic *satisfiability checking* tools for temporal logic since our works [Morzenti et al. 2003; Pradella et al. 2003], based on classical explicit-state model checking tools. Another relevant work adopting a similar approach is the one by Vardi [Rozier and Vardi 2007]. Our aim is to perform verification of purely descriptive real-time models, typically arising from high-level requirement analysis of critical systems, or hybrid descriptive/operational ones, e.g. those obtained after refinement steps following the approach presented in Section 5.5. To this end, we developed Zot, the tool introduced in Section 2.6.

The Alloy Analyzer [Jackson 2003] is a mature SAT-based tool, well-known in the Software Engineering community. The scope of application of Alloy is complementary to that of Zot: Alloy provides a rich language for describing complex data structures, while Zot is focused and optimized for complex metric-time specifications, an issue not considered by Alloy.

Uppaal [Bengtsson et al. 1995] is another very efficient tool suitable for verification of real-time systems. It is based on timed automata extended with some useful data types, so it was designed for managing quantitative metric time. It is a mature, industrial-strength tool, with an easy to use graphical interface. Uppaal is different in scope from Zot, as it provides a strong support to operational, timed automata-based models, while the logic used for expressing properties is quite limited in expressiveness and readability, if compared to the one offered by Zot. On the other hand, Uppaal performance is superior in many notable cases.

Program refinement is most often viewed in Software Engineering as a verifiable transformation of a high-level specification into an executable program. Usually this method is based on several intermediate steps, and is supported by suitable tools like, e.g., in the B method [Abrial 1996]. Our approach to refinement, as covered by Sections 5.4 and 5.5, has many similarities with that of B: we start from a description S of the system, and we *refine* it, by substituting part of it with a more operational version. The lower-level system S' thus obtained is said to be a *refinement* of S if S' implies S. The most notable differences of our approach from the B method are that Bounded Refinement Checking is in general incomplete and that we focus on high-level requirements, while the B method is complete and is based on an operational notation called AMN (Abstract Machine Notation) that covers relatively lower levels of the development reaching actual code generation (an aspect that we do not consider).

Instead, our concept of refinement is quite different from the one presented in [Clarke et al. 2002], where SAT-based abstraction refinement is introduced. In that work, abstraction/refinement is a means to more efficiently generate counterexamples through a SAT solver.

Compositional analysis techniques, based on modularization of models and requirements specifications, apply some, possibly formal, method to infer global properties of a large, complex system through a hierarchical and iterative process that exploits the system's modular structure. The need for compositionality has become undeniable in the formal methods community, due to the increased complexity of the analyzed systems and of the addressed verification issues. Therefore almost every newly introduced specification formalism and verification method encompasses some sort of compositional technique or permits compositional specifications. A general (and historical) introduction to compositional methods can be found in [de Roever 1997; de Roever et al. 2001]. Among the numerous contributions to this field we cite, without aiming at exhaustiveness, those approaches that most share our goals and motivations.

Two important issues underlying our work, and still largely unexplored in the present literature on compositionality, are the consideration of hard real-time aspects, which require a metric modeling of time, and the modularization of requirements expressed in a descriptive way by means of temporal logic formulae.

The work presented in [Ostroff 1999], uses the metric temporal logic RTTL in a compositional framework, and provides a verification method based on inference rules and a notion of module refinement. Nonetheless the approach is rather different from ours, as time is treated as a separate variable (while in our approach time is an implicit item of the language) and adopts state machines as the unique kind of model.

The contribution [Furia et al. 2007] introduces an automated compositional proof system for modular specifications expressed in the TRIO metric linear temporal logic, adopts a simple compositional rely/guarantee circular inference rule plus a methodology for the integration of different parts into a whole system. The main difference with respect to our approach in the present work derives from the provided tool support, implemented on top of the proof-checker PVS, allowing for deduction-based verification through theorem-proving of modular real-time axiom systems.

The works of McMillan ([McMillan 2000] and [Jhala and McMillan 2001]) present some similarities both with that of [Furia et al. 2007] and with ours, though they are focused on verification of hardware architectures and they use somewhat domain specific proof techniques. An abstract model of the architecture is used as a specification, against which a more detailed one, considered as an implementation, is verified. The correct implementation is stated in terms of a set of refinement relations, and the proof relies on circular compositional techniques such as mutual temporal induction.

A rather different compositional framework to support the top-down development of real-time systems based on logical formulae is studied by Hooman [Hooman 1998]. Although the framework is independent of semantic assumptions, its set-theoretic model of semantic primitives naturally relates to interleaving semantics models. The notion of refinement adopted by the framework is focused on decomposition, and basically consists of an inference rule that allows one to deduce that

the decomposition of a module into its refined parts correctly implements the original unrefined module.

An early work related with ours is [Grumberg and Long 1994], which introduces a framework based on finite state processes as operational models and requirements expressed in a subset of CTL to provide efficient verification methods and an assume/guarantee style of reasoning.

7. CONCLUSIONS

In the present paper we discussed an approach to system specification and verification based on temporal logic, called Bounded Satisfiability Checking (BSC). The approach shares some features with the techniques of Bounded Model Checking (BMC), where a model consisting of a state-transition system and a temporal logic property are translated into a Boolean logic formula to be verified by a SAT solver. BSC is more general than BMC, which can thus be regarded as a special case of BSC, in that it can handle the verification of descriptive models, i.e., models consisting of generic temporal logic formulae. All the techniques presented in this paper have been tested on a rich set of case studies, ranging from simple protocols to relatively complex real-time systems.

First, we introduced a technique to handle efficiently metric temporal logic formulae, which is of crucial importance for verifying real-time or time-dependent systems. The encoding exhibits a gain in efficiency which increases with the value of the time constants of the systems, and can be helpful also in case of an operational model (i.e., in the case of BMC) when the property to be checked is expressed in metric temporal logic.

Next, we defined a specialized encoding to tackle bi-infinite time, which is more natural and less error-prone when dealing with past time operators. In fact, past operators are very useful to write compact and readable specifications and properties, but the usage of default values when an operator requires the evaluation of a formula before the initial time instant of a mono-infinite time structure may be counter-intuitive and easily lead to writing mistaken formulae. Although efficiency is reduced when using the bi-infinite encoding, the decrease is not dramatic, since solving time is typically doubled, and hence the bi-infinite encoding can be used with no significant penalty to check a model whenever past operators such as Yesterday are present. Again, this encoding can be usefully applied to BMC as well.

Finally, the paper addresses the issue of model refinement using Bounded Satisfiability Checking, and therefore called Bounded Refinement Checking. BSC supports very naturally the verification that a model M_1 is a refinement of another model M_0, at least within a chosen temporal bound. When M_0 is descriptive and M_1 is operational, then the latter may be considered an implementation of the former. Experimental results have shown that checking correctness of model refinement is feasible with Bounded Satisfiability Checking, although typically incomplete. Moreover, applying modularization techniques can significantly speed up BRC. In fact, often only parts of a system model are actually refined, so BRC can be done separately for a subset of the modules. For instance, in the modular version of the Kernel Railroad Crossing problem, refining separately the models for the controller and the interlocking subsystems led to a 70-fold increase of efficiency with respect to simultaneously refining the two models.

Overall, the results obtained on the adopted case studies concerning modeling, simulation, property analysis, and model refinement, allow us to consider Bounded Satisfiability Checking a viable method for the design and verification of complex time critical systems.

Further developments of the present work include the extension of the techniques here illustrated to more expressive temporal logic languages and the adoption of an SMT solver [Nieuwenhuis et al. 2006] as verification engine.

Acknowledgments

We thank the anonymous referees for their many insightful comments and suggestions, which allowed us to improve significantly the quality of the presentation.

REFERENCES

ABRIAL, J.-R. 1996. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press.

BENGTSSON, J., LARSEN, K. G., LARSSON, F., PETTERSSON, P., AND YI, W. 1995. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*. Number 1066 in Lecture Notes in Computer Science. Springer–Verlag, 232–243.

BERSANI, M. M., FURIA, C. A., PRADELLA, M., AND ROSSI, M. 2009. Integrated modeling and verification of real-time systems through multiple paradigms. In *SEFM*, D. V. Hung and P. Krishnan, Eds. IEEE Computer Society, 13–22.

BIERE, A., CIMATTI, A., CLARKE, E., AND ZHU, Y. 1999. Symbolic model checking without BDDs. *Lecture Notes in Computer Science 1579*, 193–207.

BIERE, A., HELJANKO, K., JUNTTILA, T., LATVALA, T., AND SCHUPPAN, V. 2006. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science 2,* 5, 1–64.

CIAPESSONI, E., MIRANDOLA, P., COEN-PORISINI, A., MANDRIOLI, D., AND MORZENTI, A. 1999. From formal models to formally based methods: An industrial experience. *ACM Trans. Softw. Eng. Methodol. 8,* 1, 79–113.

CIMATTI, A., CLARKE, E. M., GIUNCHIGLIA, E., GIUNCHIGLIA, F., PISTORE, M., ROVERI, M., SEBASTIANI, R., AND TACCHELLA, A. 2002. NuSMV 2: An opensource tool for symbolic model checking. In *CAV '02: Proceedings of the 14th Intern. Conf. on Computer Aided Verification*. Springer-Verlag, London, UK, 359–364.

CLARKE, E., KROENING, D., AND LERDA, F. 2004. A tool for checking ANSI-C programs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, K. Jensen and A. Podelski, Eds. Lecture Notes in Computer Science Series, vol. 2988. Springer, 168–176.

CLARKE, E., KROENING, D., OUAKNINE, J., AND STRICHMAN, O. 2005. Computational challenges in bounded model checking. *Software Tools for Technology Transfer (STTT) 7,* 2, 174–183.

CLARKE, E. M., GUPTA, A., KUKULA, J. H., AND STRICHMAN, O. 2002. SAT based abstraction-refinement using ILP and machine learning techniques. In *CAV*, E. Brinksma and K. G. Larsen, Eds. Lecture Notes in Computer Science Series, vol. 2404. Springer, 265–279.

COEN-PORISINI, A., PRADELLA, M., AND SAN PIETRO, P. 1998. A finite-domain semantics for testing temporal logic specifications. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th Intern. Symposium, FTRTFT'98, Lyngby, Denmark, September 14-18, 1998, Proceedings*. 41–54.

DE MOURA, L. M., OWRE, S., RUESS, H., RUSHBY, J. M., SHANKAR, N., SOREA, M., AND TIWARI, A. 2004. SAL 2. In *CAV*, R. Alur and D. Peled, Eds. Lecture Notes in Computer Science Series, vol. 3114. Springer, 496–500.

DE ROEVER, W. P. 1997. The need for compositional proof systems: A survey. In *COMPOS*, W. P. de Roever, H. Langmaack, and A. Pnueli, Eds. Lecture Notes in Computer Science Series, vol. 1536. Springer, 1–22.

DE ROEVER, W.-P., DE BOER, F., HANNEMANN, U., HOOMAN, J., LAKHNECH, Y., POEL, M., AND ZWIERS, J. 2001. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press.

EÉN, N. AND SÖRENSSON, N. 2003. An extensible SAT-solver. In *SAT Conference*. LNCS Series, vol. 2919. Springer-Verlag, 502–518.

FELDER, M. AND MORZENTI, A. 1994. Validating real-time systems by history-checking TRIO specifications. *ACM Trans. Softw. Eng. Methodol. 3,* 4, 308–339.

FURIA, C. A., PRADELLA, M., AND ROSSI, M. 2008a. Automated verification of dense-time mtl specifications via discrete-time approximation. In *FM*, J. Cuéllar, T. S. E. Maibaum, and K. Sere, Eds. Lecture Notes in Computer Science Series, vol. 5014. Springer, 132–147.

FURIA, C. A., PRADELLA, M., AND ROSSI, M. 2008b. Practical automated partial verification of multi-paradigm real-time models. In *ICFEM*, S. Liu, T. S. E. Maibaum, and K. Araki, Eds. Lecture Notes in Computer Science Series, vol. 5256. Springer, 298–317.

FURIA, C. A., ROSSI, M., MANDRIOLI, D., AND MORZENTI, A. 2007. Automated compositional proofs for real-time systems. *Theor. Comput. Sci. 376,* 3, 164–184.

GARGANTINI, A. AND MORZENTI, A. 2001. Automated deductive requirements analysis of critical systems. *ACM Trans. Softw. Eng. Methodol. 10,* 3, 255–307.

GHEZZI, C., MANDRIOLI, D., AND MORZENTI, A. 1990. TRIO: A logic language for executable specifications of real-time systems. *Journal of Systems and Software 12,* 2, 107–123.

GIRE, F. AND NIVAT, M. 1991. Langages algébriques de mots biinfinis. *Theoret. Comput. Sci. 86,* 2, 277–323.

GRUMBERG, O. AND LONG, D. E. 1994. Model checking and modular verification. *ACM Trans. Program. Lang. Syst. 16,* 3, 843–871.

HEITMEYER, C. AND MANDRIOLI, D. 1996. *Formal Methods for Real-Time Computing*. John Wiley & Sons, Inc., New York, NY, USA.

HELJANKO, K., JUNTTILA, T. A., AND LATVALA, T. 2005. Incremental and complete bounded model checking for full PLTL. In *CAV*, K. Etessami and S. K. Rajamani, Eds. Lecture Notes in Computer Science Series, vol. 3576. Springer, 98–111.

HOLZMANN, G. J. 1997. The model checker SPIN. *IEEE Trans. on Software Engineering 23,* 5, 279–295. Special Issue: Formal Methods in Software Practice.

HOOMAN, J. 1998. Compositional verification of real-time applications. In *COMPOS'97*. LNCS Series, vol. 1536. 276–300.

JACKSON, D. 2003. Alloy: A logical modelling language. In *ZB*, D. Bert, J. P. Bowen, S. King, and M. A. Waldén, Eds. Lecture Notes in Computer Science Series, vol. 2651. Springer, 1.

JHALA, R. AND MCMILLAN, K. L. 2001. Microarchitecture verification by compositional model checking. In *CAV*, G. Berry, H. Comon, and A. Finkel, Eds. Lecture Notes in Computer Science Series, vol. 2102. Springer, 396–410.

KAMP, J. A. W. 1968. *Tense Logic and the Theory of Linear Order (Ph.D. thesis)*. University of California at Los Angeles.

KOYMANS, R. 1990. Specifying real-time properties with metric temporal logic. *Real-Time Systems 2,* 4, 255–299.

KROENING, D. AND STRICHMAN, O. 2003. Efficient computation of recurrence diameters. In *VMCAI*, L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, Eds. Lecture Notes in Computer Science Series, vol. 2575. Springer, 298–309.

LAMPORT, L. 1987. A fast mutual exclusion algorithm. *ACM TOCS-Transactions On Computer Systems 5,* 1, 1–11.

LEWIS, M., SCHUBERT, T., AND BECKER., B. 2007. Multithreaded SAT solving. In *12th Asia and South Pacific Design Automation Conference*.

LICHTENSTEIN, O., PNUELI, A., AND ZUCK, L. D. 1985. The glory of the past. In *Proceedings of the Conf. on Logic of Programs*. Springer-Verlag, London, UK, 196–218.

MANDRIOLI, D., MORASCA, S., AND MORZENTI, A. 1995. Generating test cases for real-time systems from logic specifications. *ACM Trans. Comput. Syst. 13,* 4, 365–398.

MANOLIOS, P., SRINIVASAN, S. K., AND VROON, D. 2007. BAT: The bit-level analysis tool. In *CAV*, W. Damm and H. Hermanns, Eds. Lecture Notes in Computer Science Series, vol. 4590. Springer, 303–306.

MCMILLAN, K. L. 2000. A methodology for hardware verification using compositional model checking. *Sci. Comput. Program. 37,* 1-3, 279–309.

MORASCA, S., MORZENTI, A., AND SAN PIETRO, P. 2000. A case study on applying a tool for automated system analysis object oriented logic specification of time-critical systems. based on modular specifications written in TRIO. *Autom. Softw. Eng. 7,* 2, 125–155.

MORZENTI, A., MANDRIOLI, D., AND GHEZZI, C. 1992. A model parametric real-time logic. *ACM Trans. Program. Lang. Syst. 14,* 4, 521–573.

MORZENTI, A., PRADELLA, M., SAN PIETRO, P., AND SPOLETINI, P. 2003. Model-checking TRIO specifications in SPIN. In *FME*, K. Araki, S. Gnesi, and D. Mandrioli, Eds. Lecture Notes in Computer Science Series, vol. 2805. Springer, 542–561.

MORZENTI, A. AND SAN PIETRO, P. 1994. Object-oriented logical specification of time-critical systems. *ACM Trans. Softw. Eng. Methodol. 3,* 1, 56–98.

NIEUWENHUIS, R., OLIVERAS, A., AND TINELLI, C. 2006. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM 53,* 6, 937–977.

OSTROFF, J. S. 1999. Composition and refinement of discrte real-time systems. *ACM Trans. Softw. Eng. Methodol. 8,* 1, 1–48.

PERRIN, D. AND PIN, J.-É. 2004. *Infinite Words*. Pure and Applied Mathematics Series, vol. 141. Elsevier. ISBN 0-12-532111-2.

PNUELI, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*. IEEE Computer Society Press, Providence, Rhode Island, 46–57.

PRADELLA, M., MORZENTI, A., AND SAN PIETRO, P. 2007. 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.

PRADELLA, M., MORZENTI, A., AND SAN PIETRO, P. 2008a. Benchmarking model- and satisfiability-checking on bi-infinite time. In *ICTAC 2008*. Lecture Notes in Computer Science Series, vol. 5160. Springer, Istanbul, Turkey, 290–304.

PRADELLA, M., MORZENTI, A., AND SAN PIETRO, P. 2008b. 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*. 119–127.

PRADELLA, M., MORZENTI, A., AND SAN PIETRO, P. 2009. A metric encoding for bounded model checking. In *Proceedings of FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009.*, A. Cavalcanti and D. Dams, Eds. Lecture Notes in Computer Science Series, vol. 5850. Springer, 741–756.

PRADELLA, M., SAN PIETRO, P., SPOLETINI, P., AND MORZENTI, A. 2003. Practical model checking of LTL with Past. In *ATVA03, Taipei, Taiwan, 2003*.

PRIOR, A. 1967. *Past. Present and Future*. Oxford University Press (reprinted 2002), Oxford.

RESCHER, N. AND URQUHART, A. 1971. *Temporal Logic*. Springer-Verlag, New York, NY, USA.

ROZIER, K. Y. AND VARDI, M. Y. 2007. LTL satisfiability checking. In *SPIN*. Lecture Notes in Computer Science Series, vol. 4595. Springer, 149–167.

SAN PIETRO, P., MORZENTI, A., AND MORASCA, S. 2000. Generation of execution sequences for modular time critical systems. *IEEE Trans. Software Eng. 26,* 2, 128–149.

Received R; revised e; accepted c
eived Month Year; revised Month Year; accepted Month Year

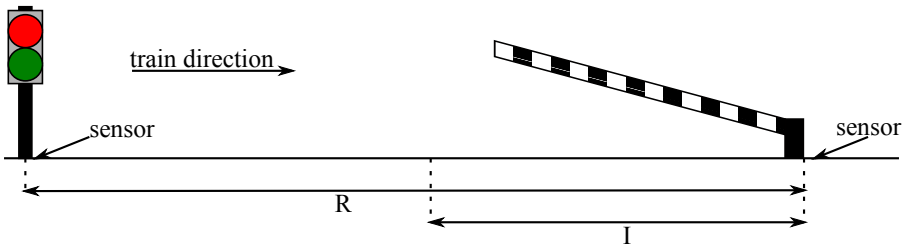

Fig. 23. Regions of interest in the Kernel Railroad Crossing.

I. APPENDIX

We report here the completely formalized version of the case studies that were presented informally in Section 2.5, namely, the KRC, the Allocator, Timer Reset Lamp, and the Asynchronous Shift Register.

I.1. The Kernel Railway Crossing Case Study

A rail road crossing is an intersection between a road and a train track with a gate to prevent crossing of the track by vehicles during train passage. Two regions R and I, surrounding the crossing, are defined as depicted in Figure 23.

Trains enter region R, then enter critical region I and finally leave the area. Trains entering and leaving region R are detected by means of sensors placed on the track, at the ends of the R region. Notice that only one train at a time can enter and exit the R and I regions, but in principle several trains might be simultaneously present in those regions. However, an interlocking system is set up with the purpose of ensuring that at most one train is present, at any time, in the R (and hence in the I) region. This avoids the risk of collisions among trains by maintaining a distance among them greater than or equal to the length of the R region, and at the same time facilitates the management of the bar rising and lowering, since the control system must only keep track of the possible presence of at most one train inside the R region. The Interlocking system operates on a light which is placed at the entrance of region R: the light is turned red when a train enters R, and is turned green when a train exits region I. Trains can enter region R only when the light is green. It takes a train a minimum time d_m and a maximum time d_{Max} to go from the beginning of R to the beginning of I, and then a minimum time h_m and a maximum time h_{Max} to go from the beginning of region I to its end (thus exiting also the region of interest for the KRC). The controller must ensure that the bar is closed when a train is in region I (*safety* property), but, to avoid needless blocks on the road, it must also ensure that the bar is down only when strictly necessary (*utility* property). The controller operates the bar by means of *up* and *down* commands; the bar current position or state of motion is one of: *closed*, *open*, *movingUp* (when opening), and *movingDown* (when closing). It takes the bar γ time units to reach the closed (respectively, open) position starting from the open (respectively, closed) state. The controller, based on the data coming from sensors, determines the presence (or absence) of a train that might possibly be in the region I, and sends to the bar, in a timely manner, suitable commands to rise or lower it.

In our experiments we analyzed the KRC problem with two sets of time constants. The first one allows for a high degree of nondeterminism on train behavior, since it considers quite different values for the minimal and maximal allowed train speed ($d_{Max} = 22$ and $d_m = 16$, $h_{Max} = 14$ and $h_m = 8$). The second set of time constants considers a more constrained system, where the difference between maximal and and minimal train speed is more limited ($d_{Max} = 22$ and $d_m = 20$, $h_{Max} = 14$ and $h_m = 12$). In both settings the considered time necessary to complete the bar movement was $\gamma = 4$.

The experiments considered satisfiability of the specification, analysis of the safety and utility properties, and various forms of refinement of the descriptive model into an operational one.

Since the KRC case study includes a quite large number of formulae, for ease of reading and reference we will introduce the axioms referring to its various components: the Controller, the Interlock, the Gate, the Train&Track. We also provide the safety and utility properties expressed as temporal logic axioms. For all the above components we provide first the descriptive model, which is the one analyzed in the experiments presented in Sections 3 and 4. Then, for the two components, the Controller and the Interlocking, whose descriptive model is refined into an operational one, we also provide the operational version. For every considered component, the model consists of the conjunction of the listed axioms, with an outermost $\mathcal{A}lw$ operator. For the sake of readability, every axiom is preceded by an illustrative comment.

Axioms for the Controller

A $go(up)$ command is issued to raise the bar exactly when the train exits critical region *I*.

$go(up) \longleftrightarrow ExitI$

A $go(down)$ command is issued to lower the bar exactly $d_m - \gamma$ time units after the train enters region R

$go(down) \longleftrightarrow \blacklozenge_{d_m-\gamma} EnterR$

Axioms for the Interlocking system

Red and green colors for the signaling semaphores are mutually exclusive

$red \longleftrightarrow \neg green$

The light is red iff there was no *ExitI* (with no simultaneous *EnterR*) since the last *EnterR* (hence the light is green in the case there was no *EnterR* in the past)

$red \longleftrightarrow \bullet(\neg(ExitI \wedge \neg EnterR)\ \mathcal{S}\ EnterR)$

Axioms for the Gate

We use two unary time dependent predicates $go(d)$, with $d \in \{up, down, notin\}$, and $bar(p)$, with $p \in \{open, closed, mvup, mvdown\}$, to represent commands to the bar and its position. Suitable additional axioms, not reported here for brevity, ensure existence and uniqueness of these values.

When the bar is closed and it receives a go(up) command it starts moving immediately, it moves up for γ time units, it reaches the open position and remains in that position until a subsequent $go(down)$ command, or indefinitely if no successive $go(down)$ command is issued.

$$\bullet bar(closed) \wedge go(up) \rightarrow \Box_{<\gamma} bar(mvup) \wedge \Diamond_{=\gamma}((bar(open) \mathcal{U} go(down)) \vee \Box bar(open))$$

Symmetrically to the previous axiom: when the bar is open and it receives a go(down) command it starts moving immediately, it moves down for γ time units, it reaches the closed position and remains in that position until a subsequent $go(up)$ command, or indefinitely if no successive $go(up)$ command is issued.

$$\bullet bar(open) \wedge go(down) \rightarrow \Box_{<\gamma} bar(mvdown) \wedge \Diamond_{=\gamma}((bar(closed) \mathcal{U} go(up)) \vee \Box bar(closed))$$

Axioms describing Train movement

The axioms below are inspired to similar ones in [Gargantini and Morzenti 2001]. Their complexity derives from the fact that they characterize in a very general and abstract way the movement of an unbound number of trains that are not individually identified and move at a speed that is not constant but is constrained between a minimum and a maximum value. It is to be noticed that the nondeterminism of the train movement is the major source of complexity in the analysis of the KRC case study.

The trains enter region R only when the semaphore is green

$EnterR \rightarrow green$

Next we introduce axioms defining the relationship predicate ER_EI between $EnterR$ events (a train entering region R) and $EnterI$ events (a train entering region I). The ER_EI is used in the coming axioms to state that $EnterR$ and $EnterI$ are related in pairs, that every $EnterR$ event causes a unique $EnterI$ event, and that every $EnterI$ event is caused by a unique $EnterR$ event.

Occurrence axiom for ER_EI
$\forall t1(d_m \leq t1 \leq d_{Max} \wedge ER\_EI(t1) \rightarrow EnterR \wedge \Diamond_{=t1} EnterI)$
Cause axiom for ER_EI
$EnterR \rightarrow \exists t1(d_m \leq t1 \leq d_{Max} \wedge ER\_EI(t1))$
Effect axiom for ER_EI
$EnterI \rightarrow \exists t1(d_m \leq t1 \leq d_{Max} \wedge \blacklozenge_{=t1} ER\_EI(t1))$
Unique effect axiom for ER_EI
$\forall t1 \forall t2(d_m \leq t1 \leq d_{Max} \wedge d_m \leq t2 \leq d_{Max} \wedge ER\_EI(t1) \wedge ER\_EI(t2) \rightarrow t1 = t2)$
Unique cause axiom for ER_EI

$$\forall t1 \forall t2 \left(\begin{array}{c} d_m \leq t1 \leq d_{Max} \wedge d_m \leq t2 \leq d_{Max} \\ \wedge \blacklozenge_{=t1} ER\_EI(t1) \wedge \blacklozenge_{=t2} ER\_EI(t2) \\ \rightarrow \\ t1 = t2 \end{array} \right)$$

Similar axioms define the relationship predicate EI_ExI between $EnterI$ events (a train entering region I) and $ExitI$ events (a train exiting region I)
Occurrence axiom for EI_ExI
$\forall t1(h_m \leq t1 \leq h_{Max} \wedge EI\_ExI(t1) \rightarrow EnterI \wedge \Diamond_{=t1} ExitI)$
Cause axiom for EI_ExI
$EnterI \rightarrow \exists t1(h_m \leq t1 \leq h_{Max} \wedge EI\_ExI(t1))$
Effect axiom for EI_ExI
$ExitI \rightarrow \exists t1(h_m \leq t1 \leq h_{Max} \wedge \blacklozenge_{=t1} EI\_ExI(t1))$
Unique effect axiom for EI_ExI

$$\forall t1 \forall t2 \left(\begin{array}{c} h_m \leq t1 \leq h_{Max} \wedge h_m \leq t2 \leq h_{Max} \wedge EI\_ExI(t1) \wedge EI\_ExI(t2) \\ \rightarrow \\ t1 = t2 \end{array} \right)$$

Unique cause axiom for EI_ExI

$$\forall t1 \forall t2 \left(\begin{array}{c} h_m \leq t1 \leq h_{Max} \wedge h_m \leq t2 \leq h_{Max} \\ \wedge \blacklozenge_{=t1} EI\_ExI(t1) \wedge \blacklozenge_{=t2} EI\_ExI(t2) \\ \rightarrow \\ t1 = t2 \end{array} \right)$$

The train is InI iff no $ExitI$ occurred since the last $EnterI$
$InI \longleftrightarrow (\neg ExitI \; \mathcal{S}_i \; EnterI)$

Safety and utility properties

Safety: Whenever the train is in critical region *I*, the bar is in the *closed* position.
$\mathcal{A}lw(InI \rightarrow bar(closed))$
Utility: we introduce the two time constants D_{pre} and D_{post}, denoting the maximal length of the time interval in which we accept that the bar is not in the *open* position before the train enters region *I* (and, respectively, after it exits it). For the first set of time constants adopted in our experiments ($d_{Max} = 22$ and $d_m = 16$, $h_{Max} = 14$, $h_m = 8$, and $\gamma = 5$), we take $D_{pre} = 11$ and $D_{post} = 5$, while for the second set of time constants ($d_{Max} = 22$ and $d_m = 20$, $h_{Max} = 14$, $h_m = 12$, and $\gamma = 5$) we take $D_{pre} = 11$ and $D_{post} = 5$ (notice that $D_{post} = 5$ in both cases, since it is only related with the time γ taken by the bar to go from closed to open position). The utility property asserts that at any time instant for which no train has been in the *I* region for the last D_{post} time units, and also no train will be in the *I* region for the next D_{pre} time units, the bar must be in the open position.
$\mathcal{A}lw(\Box_{\leq D_{pre}} \neg InI \wedge \blacksquare_{\leq D_{post}} \neg InI \rightarrow bar(open))$

Axioms for the Controller, operational version

We introduce the additional state predicate $PastER$; $PastER(i)$, for $1 < i < d_m - \gamma$, means that an ***EnterR*** event occurred *i* time units ago. The $PastER$ predicate is defined by the following axioms.
$EnterR \longleftrightarrow \circ PastER(1)$

$\forall x(1 \leq x < (d_m - \gamma) \rightarrow (PastER(x) \longleftrightarrow \circ PastER(x+1)))$
Initially, when no $EnterR$ event has yet occurred, predicate $PastER(x)$ is false for all values of its argument. The following axiom is asserted at time 0, which is conventionally assumed as the instant when the KRC starts its operation.
$\forall x(1 \leq x < (d_m - \gamma) \rightarrow \neg PastER(x))$
A $go(down)$ command is issued to lower the bar exactly $d_m - \gamma$ time units after the train enters region R
$go(down) \longleftrightarrow PastER(d_m - \gamma)$
A $go(up)$ command is issued to raise the bar exactly when the train exits critical region I.
$go(up) \longleftrightarrow ExitI$

Axioms for the Interlocking, operational version
If the light is green and a train enters, then the next time unit the light will be red
$green \wedge EnterR \rightarrow \circ red$
If the light is green and no train enters, then the next time unit the light will still be green
$green \wedge \neg EnterR \rightarrow \circ green$
If the light is red, a train exits and no train enters, the next time unit the light will be green
$red \wedge ExitI \wedge \neg EnterR \rightarrow \circ green$
If the light is red and a train exits, and a train enters, the next time unit the light will still be red
$red \wedge ExitI \wedge EnterR \rightarrow \circ red$
If the light is red and no train exits, the next time unit the light will still be red
$red \wedge \neg ExitI \rightarrow \circ red$

I.2. The Real-Time Allocator Case Study

In the formalization of the Real Time Allocator we use the following predicates, with the indicated meaning.
APR(p): (Active Pending Request) in the recent past process p has issued a request that is still active (timeout T_{req} not elapsed yet) and is pending (it has not been satisfied so far).
LRAPR(p): (Least Recent Active Pending Request) there is an active pending request (see predicate $APR(p)$) by process p, and it is the least recent one (all other active pending requests are more recent).
available: the resource is currently available (not assigned to any process).

The description of the real-Time Allocator is composed of the following axioms, where variables p and q denoting processes are ranging over the integer set $[1..n_p]$, n_p being the assumed number of processes.
The resource is assigned to process p iff it is available and p has the least recent active pending request

$$\forall p\,(gr(p) \longleftrightarrow available \wedge LRAPR(p))$$

The resource is available iff it has not been granted to any process, either from the release by the last process that was assigned it, or forever in the past

$$available \longleftrightarrow \neg\exists p \;\bullet\, (gr(p)\; \mathcal{S}\; rel) \vee \bullet\blacksquare\neg\exists p\; gr(p)$$

A request by process p is active and pending if it was issued by a process p less than T_{req} time units ago, and the resource was not granted to p since then

$$APR(p) \longleftrightarrow \bullet\left(\neg gr(p)\; \mathcal{S}_{<T_{req}} rq(p)\right)$$

A request by process p is the least recent active pending one if it is an active pending request and if there is no other less recent request by another process q. Here variables t_p and t_q indicate time

distances, as well as every variable of the form t_i, and range on the set $[1..T_{req}]$.

$$LRAPR(p) \longleftrightarrow \exists t_p \left(\begin{array}{c} APR(p) \\ \wedge \\ \neg rq(p) \mathcal{S}_{=t_p} rq(p) \end{array} \wedge \neg \exists q \; \exists t_q \left(\begin{array}{c} q \neq p \wedge \\ t_q > t_p \wedge \\ APR(q) \wedge \\ \neg rq(q) \mathcal{S}_{=t_q} rq(q) \end{array} \right) \right)$$

Once granted the resource, any process p keeps it for at least one time unit and releases it within T_{rel} time units

$$\forall p \; (gr(p) \rightarrow \neg rel \wedge \circ \Diamond_{<T_{rel}} rel)$$

There are no spurious release signals: a release signal is issued only if there has be no previous release since the last grant of the resource to any process p

$$rel \rightarrow \bullet (\neg rel \; \mathcal{S} \; \exists p \; gr(p))$$

There can be no simultaneous requests by two distinct processes

$$\neg \exists p \; \exists q \; (rq(p) \wedge rq(q) \wedge p \neq q)$$

There are no spurious resource requests by any process, i.e., a process will not issue a resource request if there is an active pending request by the same process, or if the process is holding the resource (the resource has been granted to the process and since then it has not released it)

$$\forall p \; ((APR(p) \vee gr(p) \vee \bullet (\neg rel \; \mathcal{S} \; gr(p))) \rightarrow \neg rq(p))$$

The overall specification of the real-time allocator system is obtained by prefixing all axioms by universal quantifications on any free variable, by conjoining the axiom and prefixing universal temporal quantification operator $\mathcal{A}lw$.

In the reported experiments we considered the case of a system with three processes and $T_{rel} = T_{req} = 3$ (referred to as *alloc*). As in the previous case studies, we first used Zot to generate a simple "run" of the system (history generation); then, based on the above formalization, the following properties, named *Simple Fairness*, *Conditional Fairness*, *Precedence*, and *Suspend Fairness*, were analyzed by means of the Zot tool.

Simple Fairness If a process that does not obtain the resource always requests it again immediately after the request is expired, then if it requests the resource it will eventually obtain it. This property holds only for $T_{rel} < T_{req}$, hence not in our case, and Zot generates a counterexample. We indicate the following formula as SFAIR:

$$\mathcal{A}lw \left(rq(p) \wedge \circ \Box_{<T_{req}} \neg gr(p) \rightarrow \Diamond_{=T_{req}+1} rq(p) \right) \;\; \rightarrow \;\; \mathcal{A}lw \left(rq(p) \rightarrow \Diamond gr(p) \right)$$

Conditional Fairness Let us first define the notion of "unconstrained rotation" among processes: a process will require the resource only after all other ones have requested and obtained it. Notice that this requirement does not impose any precise ordering among the requests made by the processes (though, once requests take place in a given order, the order remains unchanged from one round among processes to the next one). This property is described by the following formula:

$$\forall p \; \mathcal{A}lw \left(\begin{array}{c} rq(p) \rightarrow \\ \forall q \left(q \neq p \rightarrow \bullet \left(\neg rq(p) \; \mathcal{S} \left(\begin{array}{c} rq(q) \wedge \\ \circ \Diamond_{\leq T_{req}} gr(q) \end{array} \right) \right) \right) \end{array} \right)$$

Under this assumption of "unconstrained rotation" the allocator system is fair for all processes: if a process, when it requests the resource and does not obtain it, always requests it again after the request is expired, then, when it requests the resource, it will eventually obtain it. If for brevity we

symbolically indicate the property of "unconstrained rotation" as UNROT, this conditional fairness property may be stated as:

$$UNROT \rightarrow SFAIR$$

Precedence The allocator system cannot grant the resource to a process a asking for it after another process b, if the resource has not yet been granted to b.

$$\forall a \forall b \forall c \left(\begin{array}{c} rq(a) \wedge \\ \exists t_1 \Diamond_{=t_1} (rq(b) \wedge b \neq a) \wedge \\ \exists t_2 \left(\Diamond_{=t_2} gr(c) \wedge \circ \Box_{<t_2} \neg \exists p \, gr(p) \right) \\ \rightarrow \\ b \neq c \end{array} \right)$$

Suspend Fairness Simple fairness holds under the assumption that every process, after obtaining the resource, suspends itself for $n_p \cdot T_{rel}$ time units, n_p being the number of processes.

$$\mathcal{A}lw \left(\forall p \left(\begin{array}{c} rq(p) \wedge \circ \Diamond_{<T_{req}} gr(p) \\ \rightarrow \\ \circ \Box_{<n_p \cdot T_{rel}} \neg rq(p) \end{array} \right) \right) \rightarrow SFAIR$$

I.3. The Asynchronous Shift Register Case Study

We represent the value of the n bits of the register by the predicate $R(x)$, with $x \in [0..n-1]$, the Shift signal by predicate letter Sh, and the bit which is input at one end of the register by predicate letter In.

Descriptive model
The first bit of the register, $R(0)$, is true iff no $Shift$ command occurred since the last time when the In signal was true simultaneously with the $Shift$ command.

$$R(0) \longleftrightarrow \bullet (\neg Sh \ \mathcal{S} \ (Sh \wedge In))$$

Notice the operator $\bullet$ in front of the $\mathcal{S}$ to ensure that the $R(0)$ bit changes one time unit after the signals Sh and In.
The specification for the remaining $n-1$ bits $R(x)$, for $x \in [1..n-1]$, is similar to the previous one, with the difference that, for any of the $R(x)$ bits, the role of the input signal In is played by the previous bit $R(x-1)$.

$$\forall x \, (1 \leq x \leq n-1 \rightarrow (R(x) \longleftrightarrow \bullet (\neg Sh \, \mathcal{S} \ (Sh \wedge R(x-1)))))$$

Operational model
As in the descriptive model, we have two similar (group of) axioms, the first for bit $R(0)$ and the second for the successive bits. Every bit remains unchanged, from one time instant to the next, when the $Shift$ command does not occur. Otherwise (i.e., when the $Shift$ command occurs) at the next time instant the bit takes the value of the In signal if it is the first one, or otherwise the value of the previous bit.
The following clauses specify the value of the first bit.

$$\begin{aligned} \neg Sh \wedge R(0) &\rightarrow \circ R(0) \\ \neg Sh \wedge \neg R(0) &\rightarrow \circ \neg R(0) \\ Sh \wedge In &\rightarrow \circ R(0) \\ Sh \wedge \neg In &\rightarrow \circ \neg R(0) \end{aligned}$$

The next clauses specify the value of the successive bits.

$$\forall x \left(1 \leq x \leq n-1 \rightarrow \left(\begin{array}{ll} \neg Sh \wedge R(x) \rightarrow \circ R(x) & \wedge \\ \neg Sh \wedge \neg R(x) \rightarrow \circ \neg R(x) & \wedge \\ Sh \wedge R(x-1) \rightarrow \circ R(x) & \wedge \\ Sh \wedge \neg R(x-1) \rightarrow \circ \neg R(x) & \end{array} \right) \right).$$